22_精选面试题：spring IOC 如何解决循环依赖问题？

1、开篇

上节课介绍@Value 基本用途和定义，并且通过基于配置文件的注入、基于非配置文件的注入以及默认注入三个方面给大家讲解了@Value 的使用方法。本节课会聊聊 spring IOC 如何解决循环依赖问题。，包括如下内容：

- 什么是循环依赖
- Spring IoC 处理循环依赖的思路
- 处理循环依赖举例

2、什么是循环依赖

Spring IoC 中的循环依赖其实就是循环引用，两个或者两个以上的 Bean 互相持有对方，最终形成闭环。如图 1 所示，如 A 依赖于 B，B 依赖于 C，C 又依赖于 A。这样这样一个场景，初始化 A 的时候需要完成 B 的初始化，而完成 B 的初始化又需要完成 C 的初始化，最后 C 又依赖于 A，如此这般 A 永远也无法完成初始化的操作。这种对象的相互依赖形成闭环的关系被称作循环依赖。

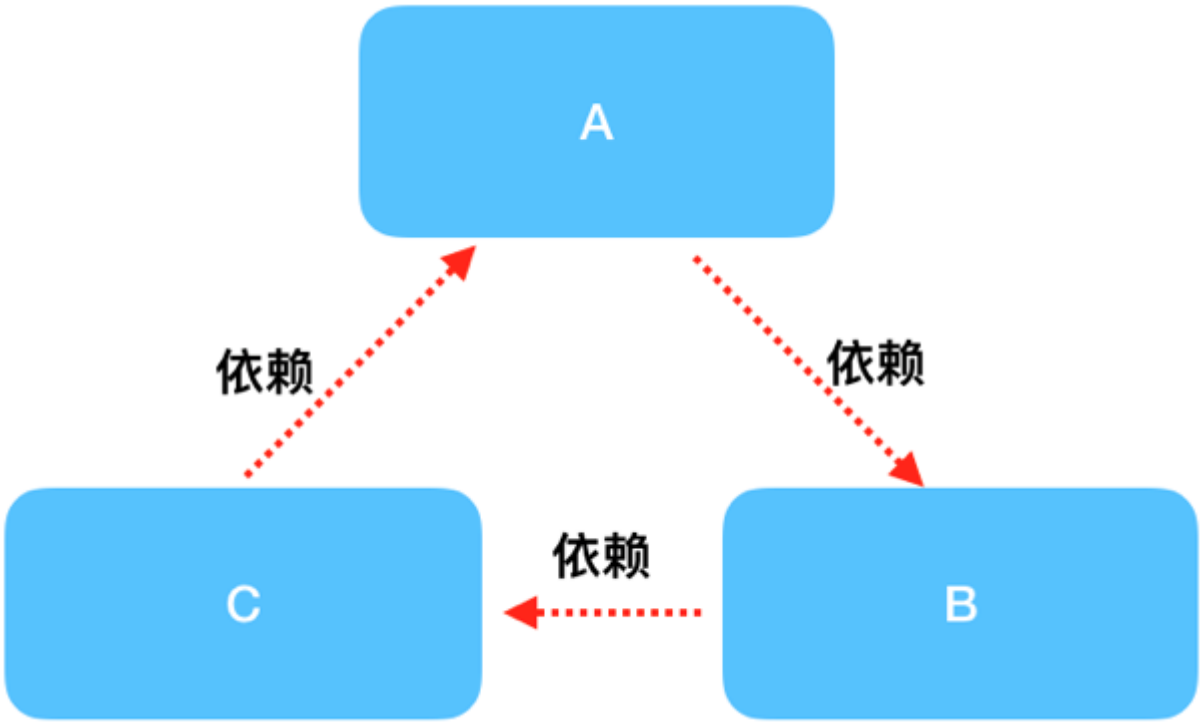

图 1 循环依赖

在 Spring IoC 的使用场景中有两类循环依赖是无解的：

- 构造器的循环依赖：构造器要调用构造函数 new 一个对象出来，而参数又依赖于另一个对象。创建类 A 依赖于类 B，new 的时候去创建类 B 发现类 B 不存在就会出错抛出 BeanCurrentlyInCreationException 异常。
- prototype 原型 bean 循环依赖：原型 bean 的初始化过程中不论是通过构造器参数循环依赖还是通过 set 方法产生的循环依赖也会抛出异常。

然而针对 singleton bean 的循环依赖的场景可以通过三级缓存的方式解决。下面就根据该解决方案展开说明。

3、Spring IoC 处理循环依赖的思路

在整理 Spring IoC 处理 singleton bean 循环依赖的思路之前先来复习一下 bean 的生命周期，其包括的三个步骤：

- 实例化：执行了 bean 的构造方法，bean 中依赖的对象还未赋值
- 设置属性：给 bean 中依赖的对象赋值，若被依赖的对象尚未初始化，则先进行该对象的生命周期（递归）。
- 初始化：执行 bean 的初始化方法，回调方法等。

解决循环依赖的思路就藏在这三个步骤中，在实例化与设置属性两个步骤之间引入缓存机制，将已经创建好实例但是并没有设置属性的 bean 放到缓存里，缓存中是没有属性设置的实例对象。假设 A 对象和 B 对象互相依赖，A 对象的创建需要引用到 B 对象，而 B 对象的创建也需要 A 对象。在创建 A 对象的时候可以将其放入到缓存中，当 B 对象创建的时候直接从缓存里引用 A 对象（此时的 A 对象只完成了实例化，没有进行设置属性的操作，因此不是完成的 A 对象，我们称之为半成品 A 对象），当 B 对象利用这个半成品的 A 对象完成实例创建以后（三个步骤都完成），再被 A 对象引用进去，则 A 对象也完成了创建。

上文提到的缓存在这里做一个解释，我们将其分为三级，每级缓存都起到不同的作用，如下表格所示：

- 一级缓存：用于存放完全初始化好的 bean，也就是完成三个步骤的 bean，拿出来的 bean 是可以直接使用的。
- 二级缓存：存放原始的 bean 对象，此时的对象只进行了实例化但是没有填充属性，也就是我们所说的“半成品对象”，它的建立是用来解决循环依赖的。
- 三级缓存：用来存放 bean 工厂对象，这个工厂对象是用来产生 bean 对象的实例的。

源码	级别	描述
singletonObjects	一级缓存	用于存放完全初始化好的 bean，从该缓存中取出的 bean 可以直接使用
earlySingletonObjects	二级缓存	存放原始的 bean 对象（尚未填充属性），用于解决循环依赖
singletonFactories	三级缓存	存放 bean 工厂对象，用于解决循环依赖

解决循环依赖的整个过程是：

先从一级缓存里取 bean 实例，如果没有对应的 bean 实例，二级缓存里取，如果二级缓存中也没有 bean 实例，singletonFactories 三级缓存里获取。由于三级缓存存放着产生 bean 实例的工厂类，因此可以通过该工厂类产生 bean 实例。

这里可以调用工厂类暴露的 getObject 方法返回早期暴露对象引用，也是我们所说的半成品 bean，也可以成为 earlySingletonObject。并且将这个半成品 bean 放到二级缓存里，在三级缓存里删除该 bean。什么时候这个半成品填充了属性以后，就被移动到一级缓存中，也就是被作为可以使用的已经完成初始化的实例 bean 了，处理循环依赖的过程宣告完毕。下面通过一个例子让大家更好理解这个思路。

4、处理循环依赖举例

根据上面的思路，这里假设 A 和 B 互相依赖，如图 2 所示，在 A 创建实例的时候使用了 getBean 方法，通过 createBeanInstatnce 方法对 A 进行实例化。此时的 A 只是被实例化出来了，并没有进行填充属性的操作，然后通过 addSingletonFactory 的方法将创建 A 的工厂类添加到三级缓存中。上面的思路中提到了这个放到三级缓存中的工厂类是用来生成 bean 实例用的。

接着往下，当通过 populateBean 填充实例 A 属性的时候发现，A 依赖 B。此时开始通过 getBean 方法创建 B 的实例，依旧通过 createBeanInstatnce 方法对 B 进行实例化，也把创建 B 实例的工厂类通过 addSingletonFactory 方法添加到三级缓存中。在使用 populateBean 方法填充 B 的属性时，发现 B 依赖 A，此时通过 getBean 方法对 A 进行实例化。

这个时候就出现循环依赖的情况了，getBean 方法先从一级缓存中获取 A 的实例，发现没有，再去二级缓存中找，还是找不到，没有办法只有找三级缓存中的 A 实例创建工厂去创建 A 的实例。在前面的步骤中 A 已经将工厂类通过 addSingletonFactory 方法存放到了三级缓存中，于是调用 A 的工厂类创造 A 的实例，并且将其放到二级缓存中返回给 B 用来填充 B 的属性，当 B 完成属性填充以后产生了 B 的实例，返回给 populateBean（A）使用，此时 A 获取了 B 的实例（完成属性填充的 B 实例）。

所以，A 也可以完成属性填充从而产生 A 的初始化以后的实例并且将其放到一级缓存中。由于 B 之前使用的是 A 的实例是没有做属性填充的，也就是半成品的 A 实例，因此此时从一级缓存中获取成品的 A 实例完成 B 对象的初始化。

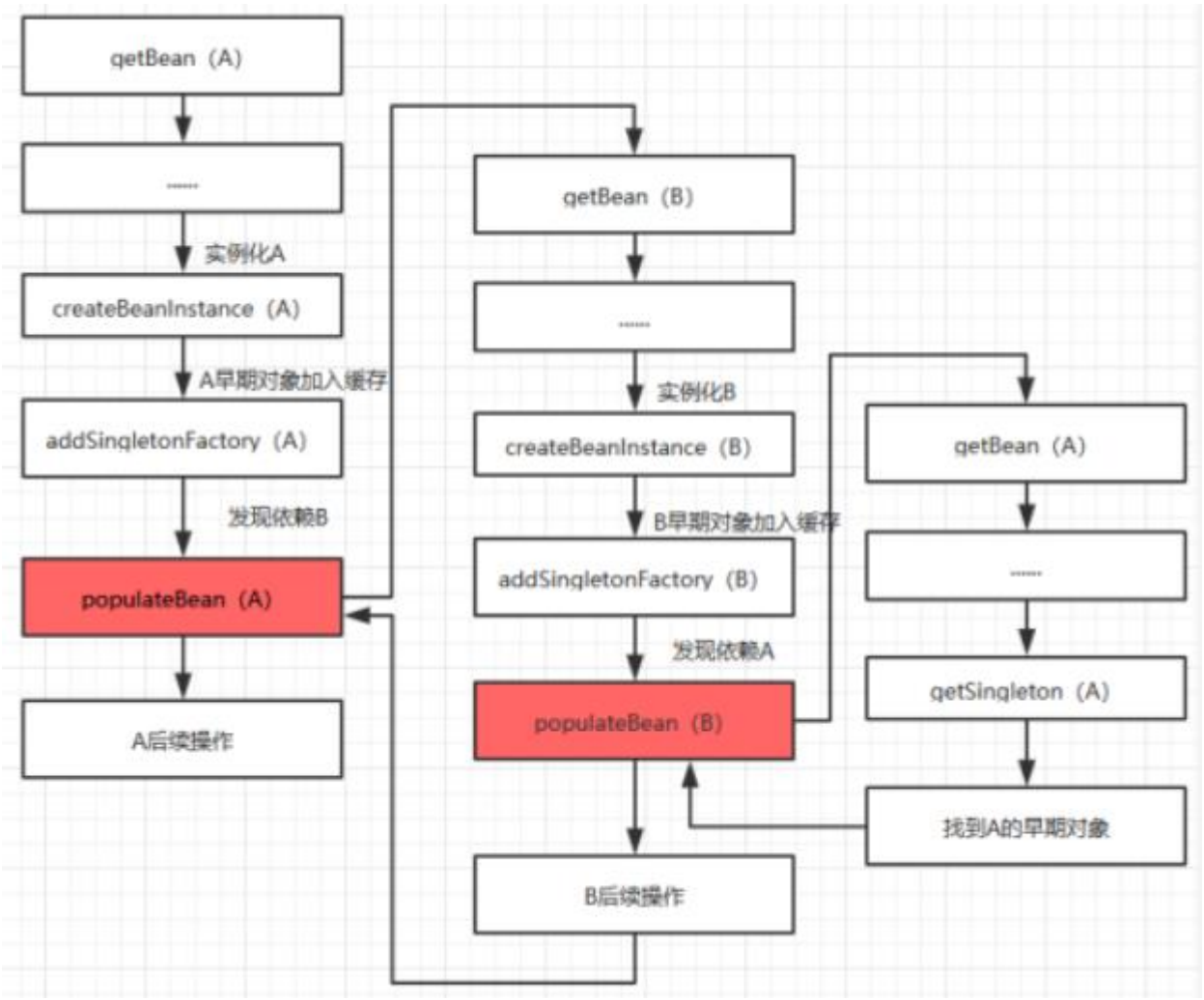

图 2 A B 相互循环依赖，如何处理。

5、总结

本节课提出了 Spring IoC 遇到的循环依赖的问题，并且通过分析 bean 创建的过程和三级缓存技术，找到了解决 singleton bean 循环依赖的办法，然后通过一个简单的循环依赖处理的例子加强对这一思路的理解。本节课作为本周的最后一节课，在这里给大家做一个总结。

本周从 Spring IoC 容器初始化过程开始，了解了 Spring IoC 依赖查找的方式，针对 bean 名称和类型查找进行了源码分析。并且提醒各位在 bean 查找不到一级不唯一情况下的异常处理。针对 bean 初始化流程进行了源码分析，了解了依赖注入的概念以及对应的实现方式。将依赖注入中的@AutoWired 进行了源码分析，并且将

其与常用的@Resource 方式进行对比。在介绍了 Spring IoC 依赖来源之后，又介绍@Value 如何获取外部配置信息，以及如何解决循环依赖问题的。

下周会专注于 Spring AOP 原理的分析。下周见，拜拜。