讲一讲 Spring循环依赖及解决方式

题目标签

学习时长：20分钟

题目难度：中等

知识点标签：Spring、循环依赖

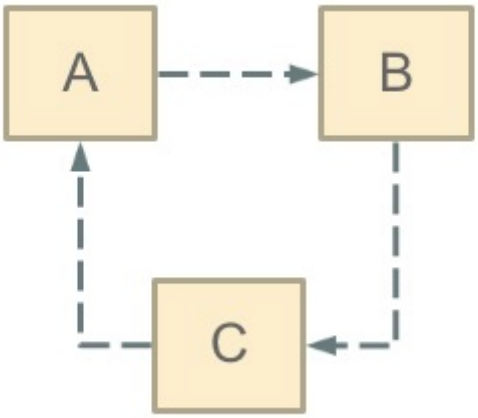
题目描述

什么是Spring循环依赖及解决方式

题目解决

1. 什么是循环依赖?

循环依赖其实就是循环引用，也就是两个或者两个以上的bean互相持有对方，最终形成闭环。比如A依赖于B，B依赖于C，C又依赖于A。如下图：



注意，这里不是函数的循环调用，是对象的相互依赖关系。循环调用其实就是一个死循环，除非有终结条件。

Spring中循环依赖场景有：

（1）构造器的循环依赖

（2）field属性的循环依赖

其中，构造器的循环依赖问题无法解决，只能抛出BeanCurrentlyInCreationException异常，在解决属性循环依赖时，spring采用的是提前暴露对象的方法。

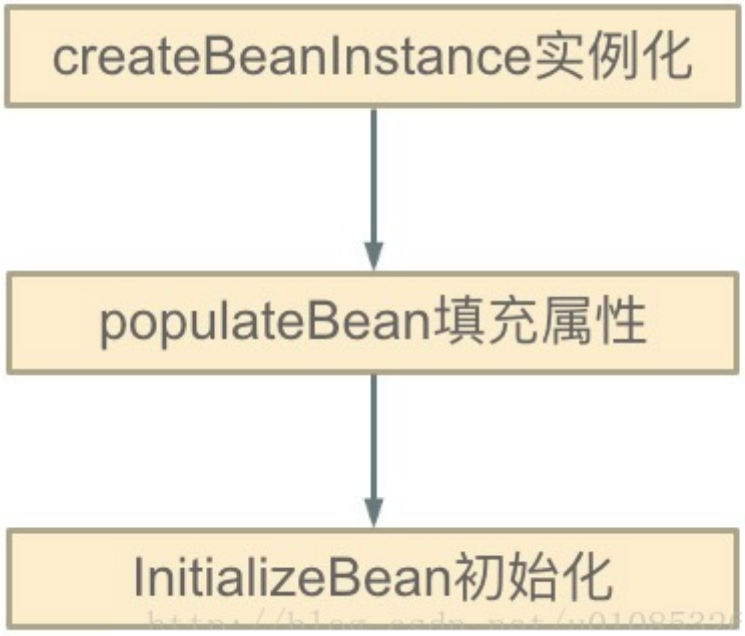
2. 怎么检测是否存在循环依赖

检测循环依赖相对比较容易，Bean在创建的时候可以给该Bean打标，如果递归调用回来发现正在创建中的话，即说明了循环依赖了。

3. Spring怎么解决循环依赖

Spring的循环依赖的理论依据基于Java的引用传递，当获得对象的引用时，对象的属性是可以延后设置的。（但是构造器必须是在获取引用之前）

Spring的单例对象的初始化主要分为三步：



（1）createBeanInstance：实例化，其实也就是调用对象的构造方法实例化对象

（2）populateBean：填充属性，这一步主要是多bean的依赖属性进行填充

（3）initializeBean：调用spring xml中的init 方法。

从上面单例bean的初始化可以知道：循环依赖主要发生在第一、二步，也就是构造器循环依赖和field循环依赖。那么我们要解决循环引用也应该从初始化过程着手，对于单例来说，在Spring容器整个生命周期内，有且只有一个对象，所以很容易想到这个对象应该存在Cache中，Spring为了解决单例的循环依赖问题，使用了**三级缓存**。

这三级缓存分别指：

singletonFactories： 单例对象工厂的cache

earlySingletonObjects ：提前暴光的单例对象的Cache

singletonObjects：单例对象的cache

在创建bean的时候，首先想到的是从cache中获取这个单例的bean，这个缓存就是singletonObjects。如果获取不到，并且对象正在创建中，就再从二级缓存earlySingletonObjects中获取。如果还是获取不到且允许singletonFactories通过getObject()获取，就从三级缓存singletonFactory.getObject()(三级缓存)获取，如果获取到了则：从singletonFactories中移除，并放入earlySingletonObjects中。其实也就是从三级缓存移动到了二级缓存。

从上面三级缓存的分析，我们可以知道，Spring解决循环依赖的诀窍就在于singletonFactories这个三级cache。这个cache的类型是ObjectFactory。这里就是解决循环依赖的关键，发生在createBeanInstance之后，也就是说单例对象此时已经被创建出来(调用了构造器)。这个对象已经被生产出来了，虽然还不完美（还没有进行初始化的第二步和第三步），但是已经能被人认出来了（根据对象引用能定位到堆中的对象），所以Spring此时将这个对象提前曝光出来让大家认识，让大家使用。

这样做有什么好处呢？让我们来分析一下“A的某个field或者setter依赖了B的实例对象，同时B的某个field或者setter依赖了A的实例对象”这种循环依赖的情况。A首先完成了初始化的第一步，并且将自己提前曝光到singletonFactories中，此时进行初始化的第二步，发现自己依赖对象B，此时就尝试去get(B)，发现B还没有被create，所以走create流程，B在初始化第一步的时候发现自己依赖了对象A，

于是尝试get(A)，尝试一级缓存singletonObjects(肯定没有，因为A还没初始化完全)，尝试二级缓存earlySingletonObjects（也没有），尝试三级缓存singletonFactories，由于A通过ObjectFactory将自己提前曝光了，所以B能够通过ObjectFactory.getObject拿到A对象(虽然A还没有初始化完全，但是总比没有好呀)，B拿到A对象后顺利完成了初始化阶段1、2、3，完全初始化之后将自己放入到一级缓存singletonObjects中。此时返回A中，A此时能拿到B的对象顺利完成自己的初始化阶段2、3，最终A也完成了初始化，进去了一级缓存singletonObjects中，而且更加幸运的是，由于B拿到了A的对象引用，所以B现在hold住的A对象完成了初始化。

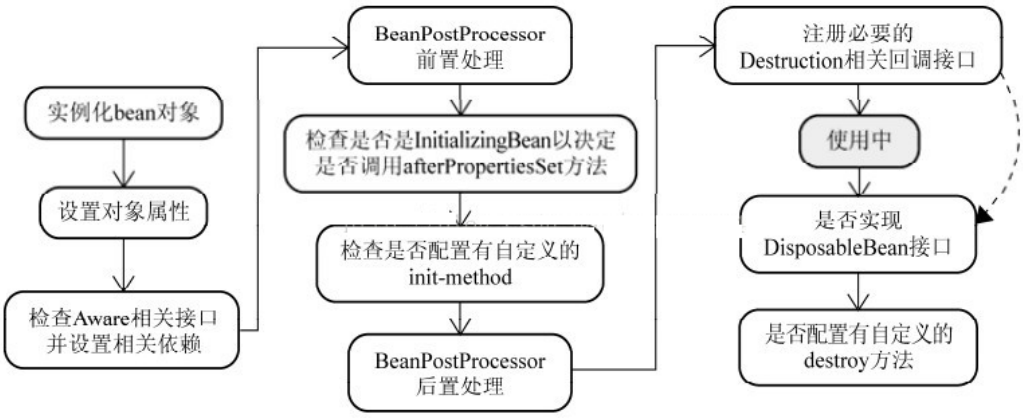
知道了这个原理时候，肯定就知道为啥Spring不能解决“A的构造方法中依赖了B的实例对象，同时B的构造方法中依赖了A的实例对象”这类问题了！因为加入singletonFactories三级缓存的前提是执行了构造器，所以构造器的循环依赖没法解决。

4.基于构造器的循环依赖

Spring容器会将每一个正在创建的Bean 标识符放在一个“当前创建Bean池”中，Bean标识符在创建过程中将一直保持在这个池中，因此如果在创建Bean过程中发现自己已经在“当前创建Bean池”里时将抛出BeanCurrentlyInCreationException异常表示循环依赖；而对于创建完毕的Bean将从“当前创建Bean池”中清除掉。

Spring容器先创建单例A，A依赖B，然后将A放在“当前创建Bean池”中，此时创建B,B依赖C ,然后将B放在“当前创建Bean池”中,此时创建C，C又依赖A， 但是，此时A已经在池中，所以会报错，，因为在池中的Bean都是未初始化完的，所以会依赖错误，（初始化完的Bean会从池中移除）

5.基于setter属性的循环依赖



我们结合上面那张图看，Spring先是用构造实例化Bean对象，创建成功后，Spring会通过以下代码提前将对象暴露出来，此时的对象A还没有完成属性注入，属于早期对象，此时Spring会将这个实例化结束的对象放到一个Map中，并且Spring提供了获取这个未设置属性的实例化对象引用的方法。 结合我们的实例来看，当Spring实例化了A、B、C后，紧接着会去设置对象的属性，此时A依赖B，就会去Map中取出存在里面的单例B对象，以此类推，不会出来循环的问题喽