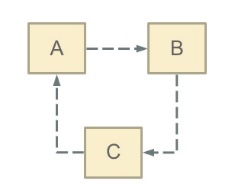
Singleton循环依赖

* 构造函数循环依赖（无法解决）
* 属性循环依赖（可以解决）

1. 什么是循环依赖？

循环依赖其实就是循环引用，也就是两个或则两个以上的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：



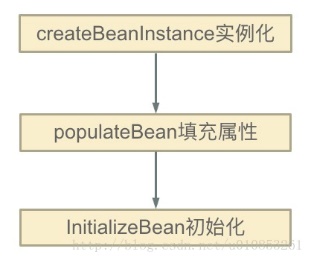
Spring中循环依赖场景有：

（1）构造器的循环依赖；

（2）field属性的循环依赖。

1. Spring怎么解决循环依赖？

Spring的循环依赖的理论依据其实是基于Java的引用传递，当我们获取到对象的引用时，对象的field或则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)。



（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。

从上面讲述的单例bean初始化步骤我们可以知道，循环依赖主要发生在第一、第二部。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

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。

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。