

Spring源码初探-IOC(4)-Bean的初始化-循环依赖的解决

💎 2 2016.08.03 20:22:56 字数 1839 阅读 11784

前言

在实际工作中，经常由于设计不佳或者各种因素，导致类之间相互依赖。这些类可能单独使用时不会出问题，但是在使用Spring进行管理的时候可能就会抛出BeanCurrentlyInCreationException等异常。当抛出这种异常时表示Spring解决不了该循环依赖，本文将简要说明Spring对于循环依赖的解决方法。

循环依赖的产生和解决的前提

循环依赖的产生可能有很多种情况，例如：

- 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
- A的构造方法中依赖了B的实例对象，同时B的某个field或者setter需要A的实例对象，以及反之
- 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，以及反之

当然，Spring对于循环依赖的解决不是无条件的，首先前提条件是针对scope单例并且没有显式指明不需要解决循环依赖的对象，而且要求该对象没有被代理过。同时Spring解决循环依赖也不是万能，以上三种情况只能解决两种，第一种在构造方法中相互依赖的情况Spring也无力回天。结论先给在这，下面来看看Spring的解决方法，知道了解决方案就能明白为啥第一种情况无法解决了。

Spring对于循环依赖的解决

Spring循环依赖的理论依据其实是Java基于引用传递，当我们获取到对象的引用时，对象的field或者属性是可以延后设置的。

Spring单例对象的初始化其实可以分为三步：

- createBeanInstance，实例化，实际上就是调用对应的构造方法构造对象，此时只是调用了构造方法，spring xml中指定的property并没有进行populate
- populateBean，填充属性，这步对spring xml中指定的property进行populate
- initializeBean，调用spring xml中指定的init方法，或者AfterPropertiesSet方法会发生循环依赖的步骤集中在第一步和第二步。

三级缓存

对于单例对象来说，在Spring的整个容器的生命周期内，有且只存在一个对象，很容易想到这个对象应该存在Cache中，Spring大量运用了Cache的手段，在循环依赖问题的解决过程中甚至使用了“三级缓存”。

“三级缓存”主要是指

```
1 /** Cache of singleton objects: bean name --> bean instance */
2 private final Map<String, Object> singletonObjects = new ConcurrentHashMap<String, Object>()
3 /** Cache of singleton factories: bean name --> ObjectFactory */
4 private final Map<String, ObjectFactory<?>> singletonFactories = new HashMap<String, ObjectFactory<?>>()
5 /** Cache of early singleton objects: bean name --> bean instance */
6 private final Map<String, Object> earlySingletonObjects = new HashMap<String, Object>(16)
```



LNamp

拥有6钻 (约0.73元)

关注

记录下多个BeanPostProcessor代理同个Bean的问题

阅读 178

【内推】【实习】阿里巴巴新零售事业群春季实习生招聘

阅读 87

精彩继续

她48岁学画，一年就办个人画展！专家：...

阅读 24826



考上清华后，他拆穿了父亲的谎言：有一种...

阅读 1065



从字面意思来说：singletonObjects指单例对象的cache，singletonFactories指单例对象工厂的cache，earlySingletonObjects指提前曝光的单例对象的cache。以上三个cache构成了三级缓存，Spring就用这三级缓存巧妙的解决了循环依赖问题。

解决方法

回想上篇文章中关于Bean创建的过程，首先Spring会尝试从缓存中获取，这个缓存就是指singletonObjects，主要调用的方法是：

```
1 protected Object getSingleton(String beanName, boolean allowEarlyReference) {
2     Object singletonObject = this.singletonObjects.get(beanName);
3     if (singletonObject == null && isSingletonCurrentlyInCreation(beanName)) {
4         synchronized (this.singletonObjects) {
5             singletonObject = this.earlySingletonObjects.get(beanName);
6             if (singletonObject == null && allowEarlyReference) {
7                 ObjectFactory<?> singletonFactory = this.singletonFactories.get(beanName);
8                 if (singletonFactory != null) {
9                     singletonObject = singletonFactory.getObject();
10                    this.earlySingletonObjects.put(beanName, singletonObject);
11                    this.singletonFactories.remove(beanName);
12                }
13            }
14        }
15    }
16    return (singletonObject != NULL_OBJECT ? singletonObject : null);}
```

首先解释两个参数：

- isSingletonCurrentlyInCreation 判断对应的单例对象是否在创建中，当单例对象没有被初始化完全(例如A定义的构造函数依赖于B对象，得先去创建B对象，或者在populatebean过程中依赖于B对象，得先去创建B对象，此时A处于创建中)
- allowEarlyReference 是否允许从singletonFactories中通过getObject拿到对象

分析getSingleton的全过程，Spring首先从singletonObjects（一级缓存）中尝试获取，如果获取不到并且对象在创建中，则尝试从earlySingletonObjects(二级缓存)中获取，如果还是获取不到并且允许从singletonFactories通过getObject获取，则通过singletonFactory.getObject()(三级缓存)获取。如果获取到了则

```
1 this.earlySingletonObjects.put(beanName, singletonObject);
2 this.singletonFactories.remove(beanName);
```

则移除对应的singletonFactory,将singletonObject放入到earlySingletonObjects，其实就是将三级缓存提升到二级缓存中！

Spring解决循环依赖的诀窍就在于singletonFactories这个cache，这个cache中存的是类型为ObjectFactory，其定义如下：

```
1 public interface ObjectFactory<T> {
2     T getObject() throws BeansException;}
```

在bean创建过程中，有两处比较重要的匿名内部类实现了该接口。一处是

```
1 new ObjectFactory<Object>() {
2     @Override public Object getObject() throws BeansException {
3         try {
4             return createBean(beanName, mbd, args);
5         } catch (BeansException ex) {
6             destroySingleton(beanName);
7             throw ex;
8         }
9     }
```

在上文已经提到，Spring利用其创建bean（这样做真的很不明确呀...）

另一处就是：

```
1 addSingletonFactory(beanName, new ObjectFactory<Object>() {
2     @Override public Object getObject() throws BeansException {
3         return getEarlyBeanReference(beanName, mbd, bean);
4     }});
```

此处就是解决循环依赖的关键，这段代码发生在createBeanInstance之后，也就是说单例对象此时已经被创建出来的。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，长大成人，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对象也蜕变完美了！一切都是这么神奇！！

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！

总结

Spring通过三级缓存加上“提前曝光”机制，配合Java的对象引用原理，比较完美地解决了某些情况下的循环依赖问题！

 56人点赞 >



 Spring




 **LNAmP**
拥有6钻 (约0.73元)

[关注](#)

"小礼物走一走，来简书关注我"

 共1人赞赏


[赞赏](#)





写下你的评论...

全部评论 15 [只看作者](#)

[按时间倒序](#) [按时间正序](#)

 **最爱xiaoyu77**
7楼 09.16 14:46

我的理解是：
完成第一步，放到singleFactories中；
完成第二步，放到earlySingletonBean中；
完成第三步，放到singletonBean中；

 赞  回复

ibmHuang



6楼 05.23 21:06

我想没有singletonFactories，直接放到earlySingletonObjects也没有问题啊

👍 赞 💬 回复



ibmHuang

5楼 05.23 21:06

为什么要三级缓存，其实singletonObjects和earlySingletonObjects也够了我认为对象提早实例化好了，直接扔到earlySingletonObjects就行了。。。为啥还要先放到singletonFactories

👍 赞 💬 回复



程序狗儿

06.21 16:06

singletonFactories里面做了一些额外操作吧，因为这个Bean可能还需要坐一些其他操作，所以你直接丢到Early里面有点问题。
AbstractAutowireCapableBeanFactory#getEarlyBeanReference()这个方法对Bean做了一个额外的操作。

💬 回复



沙丘01

06.29 15:53

@程序狗儿 额外操作后放入singletonObjects和直接放入earlySingletonObjects有什么区别？

💬 回复

✍ 添加新评论



itmanong

4楼 08.02 11:26

楼主，我想问下：那个“提前曝光”机制，到底是怎么提前曝光的，这个和java初始化有什么关系？

👍 赞 💬 回复



hou0781

3楼 04.13 11:05

您还，我想请问一下，在初始化bean之前，会有段源码判断当前bean是否有依赖的bean，如果有依赖的话，会提前初始化依赖的bean，那么，这里的依赖是哪种情况的呢？因为我在进行断点的时候，发现无论是通过构造函数依赖还是通过field域依赖，此处的判断始终都是空的。希望楼主能帮忙解答一下，感谢

```
```java
```

```
// Guarantee initialization of beans that the current bean depends on.
```

```
//在初始化当前bean之前，会判断是否存在依赖。存在的话，回去先初始化依赖的bean
```

```
//在本人断点的时候，发现无论是通过构造函数依赖还是通过field域依赖，此处的判断始终都是空的，所以不知道本处的依赖值的是哪些，希望楼主能帮忙解答一下，感谢
```

```
String[] dependsOn = mbd.getDependsOn();
```

```
if (dependsOn != null) {
```

```
for (String dep : dependsOn) {
```

```
if (isDependent(beanName, dep)) {
```

```
throw new BeanCreationException(mbd.getResourceDescription(), beanName,
"Circular depends-on relationship between '" + beanName + "' and '" + dep + "'");
```

```
}
```

```
registerDependentBean(dep, beanName);
```

```
getBean(dep);
```

```
}
```

```
}
```

```
// Create bean instance.
//此处才会真正的去初始化当前bean对象
if (mbd.isSingleton()) {
 sharedInstance = getSingleton(beanName, new ObjectFactory<Object>() {
 @Override
 public Object getObject() throws BeansException {
 try {
 return createBean(beanName, mbd, args);
 }
 catch (BeansException ex) {
 destroySingleton(beanName);
 throw ex;
 }
 }
 });
 bean = getObjectForBeanInstance(sharedInstance, name, beanName, mbd);
}
````
```

👍 赞 💬 回复



hou0781
04.13 11:06

楼主，不好意思，在评论里面不知为什么Markdown格式好像没有支持的，所以附带的代码格式乱了😓

💬 回复



海绵爸爸_7260
06.06 17:03

@hou0781 你应该是理解错了，这里的depends-on并不是我们说的"依赖注入"中的依赖，而是一种顺序上的"依赖"，比如有两个Bean：A和B，我想要保证先创建A，再创建B，那么配置文件中再Bean B中加一句 depends-on="B"

💬 回复



海绵爸爸_7260
06.06 17:05

@海绵爸爸_7260 说错了。。。最后一句话是"在配置文件中bean B中加一句depends-on="A" "

💬 回复

✍️ 添加新评论



允雨琉
2楼 01.18 14:34

"A的构造方法中依赖了B的实例对象，同时B的某个field或者setter需要A的实例对象，以及反之" 这一种依赖真的可以解决吗，我试了一下，这种好像并不能解决

👍 赞 💬 回复



二哥很猛
03.06 13:45

构造方法不行,filed 或者setter可以

💬 回复



安迪猪
01.21 20:35

通过@@Autowired自动注入的好像不行，不过对于A的构造方法中依赖了B的实例对象，同时B的某个field或者setter需要A的实例对象这种情况，在xml中进行bean的配置的好像可以，前提是先实例化B,再实例化A

 回复



ibmHuang

05.23 16:47

为什么要三级缓存，其实singletonObjects和earlySingletonObjects也够了我认为对象提早实例化好了，直接扔到earlySingletonObjects就行了。。。为啥还要先放到singletonFactories

 回复



ibmHuang

05.23 16:48

我想想没有singletonFactories，直接放到earlySingletonObjects也没有问题啊

 回复

 添加新评论 | [收起](#)

被以下专题收入，发现更多相似内容



我爱编程



spring



spring

推荐阅读

[更多精彩内容 >](#)

Spring Cloud

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智...



卡卡罗2017

Spring应用、原理以及粗读源码系列（一）--框架总述、以Bean为核心的机制（IoC...

总述：spring框架是如今J2EE开发最重要框架之一，为企业级应用提供一系列轻量级解决方案，比如：基于依赖注入的...



JackFrost_fuzhu

Spring boot参考指南

Spring Boot 参考指南 介绍 转载自:https://www.gitbook.com/book/qbgb...



毛宇鹏

Spring-IOC-循环依赖检测与Bean的创建

Spring容器的循环依赖检测 Spring容器循环依赖包括：构造器循环依赖和setter循环依赖。 1- 构造器...



zhanglbjames

百战程序员V1.2——尚学堂旗下高端培训_ Java1573题

百战程序员_ Java1573题 QQ群：561832648489034603 掌握80%年薪20万掌握50%年薪...



Albert陈凯