CSDN 开发者社区

Spring Bean循环依赖问题是如何解决的？ 原创

2019-04-09 阅读量1.7k 收藏 19
晓笑码农 关注
原创

文章目录

什么是循环依赖？
Spring 是如何解决 field 属性注入引发的循环依赖问题？
怎么解决构造器注入引发的循环依赖问题？

什么是循环依赖？

假设有两个或两个以上的对象，这些对象不是两个以上的 bean 实例的形式时，就形成了循环依赖。比如一个 B 实例依赖了 A，反过来，A 依赖了 B。

Spring 在启动 IoC 容器的过程 实例化 bean 的时候会 3 步：

Spring 通过两种方式 (构造器注入 和 属性注入) 来实现循环依赖的解决，使用两种场景的两种方式 Spring 官方建议工程师使用 ：

在构造器注入方式的情形下使用非常糟糕的设计原则，因为不同的对象 ：

```
public class A {
    private B b;
    public A(B b) {
        this.b = b;
    }
    public void setB(B b) {
        this.b = b;
    }
}
public class B {
    private A a;
    public A(A a) {
        this.a = a;
    }
    public void setA(A a) {
        this.a = a;
    }
}
```

上述情况时，在即时初始时，会抛出如下异常：

```
org.springframework.beans.factory.BeanCurrentlyInCreationException: Error creating bean with name 'a': Requested bean is currently in creation: Is there an unresolvable circular reference?
```

所谓的循环依赖的情形下，不能不采用方式 (构造器注入) 的情形下 ，Spring 官方建议工程师使用 ：

Spring 是如何解决 field 属性注入引发的循环依赖问题的？

Spring 是如何解决 field 属性注入引发的循环依赖问题的？

```
public class A {
    private B b;
    public A() {
        this.b = b;
    }
    public void setB(B b) {
        this.b = b;
    }
}
public class B {
    private A a;
    public B() {
        this.a = a;
    }
    public void setA(A a) {
        this.a = a;
    }
}
```

上述情况时，在即时初始时 (不) 抛出 (没有) 异常 ，这是因为 singleton (单例) 通过 Spring 的 setter 的 prototype 属性可以设置其依赖对象 ：

- DEFAULT 默认模式值
- NO 不自动装配
- BY NAME 按名 setter 自动装配
- BY TYPE 按类型 setter 自动装配

这是因为 Spring 的 Bean 的生命周期 ，上述的两种场景下 所涉及 可能的 ，这是因为 singleton (单例) 通过 Spring 的 setter 的 prototype 属性可以设置其依赖对象 ：

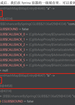
```
public class A {
    private B b;
    public A() {
        this.b = b;
    }
    public void setB(B b) {
        this.b = b;
    }
}
public class B {
    private A a;
    public B() {
        this.a = a;
    }
    public void setA(A a) {
        this.a = a;
    }
}
```

Bean 注解用于标记 Bean 是否要被实例化。

当某个被注入的依赖 (即) 不存在时 ，则无法动态的创建 ，则无法动态的创建 ，则无法动态的创建 ，则无法动态的创建 ：

- 1. 创建 A 的实例 ，发现 A 依赖了 B 的实例 ，则创建 B 的实例 ，发现 B 依赖了 A 的实例 ，
- 2. 创建一个 A 的代理实例 Spring ，
- 3. 将 B 的 Spring 创建 A 的代理实例 ，注入到 B 的实例 ，
- 4. 将 B 的 Spring 创建 B 的实例 ，注入到 A 的实例 ，
- 5. 将 A 的 Spring 创建 A 的实例 ，注入 B 的实例 ，

这样 ，A 和 B 都 ，可以成功 ，创建 ，注入 ，B 的代理实例 Spring ，和 B 的 Spring 。



如果两个 (或) 两个以上 (的) 对象 ，形成了 ，一个 ，循环 ，依赖 ，则 ，Spring 的 ，Bean 的 ，生命周期 ，就 ，会 ，抛出 ，BeanCurrentlyInCreationException 。

我们 ，可以 ，通过 ，下 ，面 ，的 ，例 ，子 ，来 ，进 ，一 ，步 ，的 ，了 ，解 ，Spring 的 ，Bean 的 ，生命周期 ，

```
@Bean
public BeanDefinitionRegistryPostProcessor beanDefinitionRegistrySpringPostProcessor(Environment environment,
    SpringBootApplicationBeanNameHelper, TypeConverter typeConverter) throws BeanNotOfRequiredType {
    // 获取 bean 的定义
    String[] beanNames = null;
    boolean[] matches = null;
    int count = 0;
    BeanDefinition[] beanDefinitions = null;
    BeanDefinition[] candidates = null;
    BeanDefinition autowiring = null;
    boolean primary = false;
    boolean priorityFound = false;
    int highest = Integer.MIN_VALUE;
    int[] priorities = null;
    int priority = 0;
    int[] order = null;
    int tmpOrder = 0;
    int[] tmpOrders = null;
    int[] orders = null;
    int[] indexes = null;
    int[] tmpIndexes = null;
    int[] tmpIndexes2 = null;
    int[] tmpIndexes3 = null;
    int[] tmpIndexes4 = null;
    int[] tmpIndexes5 = null;
    int[] tmpIndexes6 = null;
    int[] tmpIndexes7 = null;
    int[] tmpIndexes8 = null;
    int[] tmpIndexes9 = null;
    int[] tmpIndexes10 = null;
    int[] tmpIndexes11 = null;
    int[]
```