

Spring注解驱动开发第13讲——使用InitializingBean和DisposableBean来管理bean的生命周期，你真的了解吗？

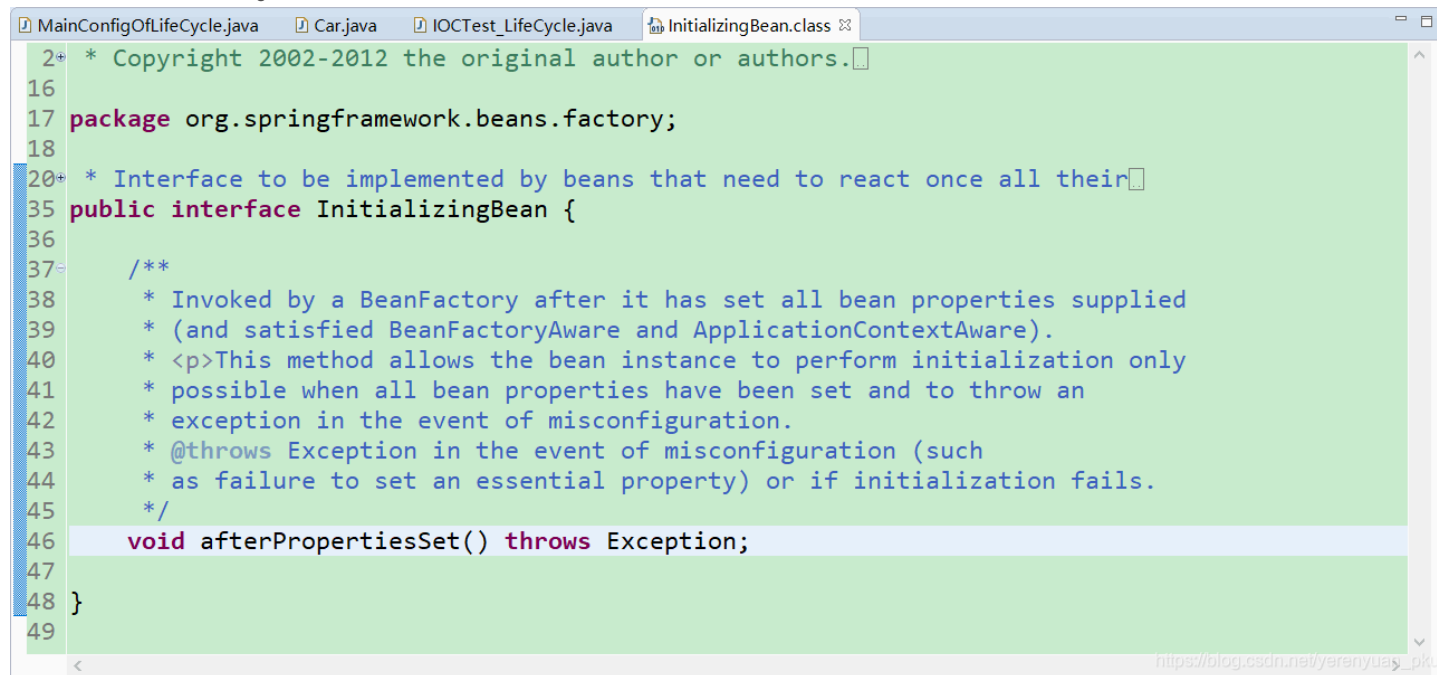
写在前面

在上一讲中，我们讲述了如何使用@Bean注解来指定bean初始化和销毁的方法，具体的用法就是在@Bean注解中使用init-method属性和destroy-method属性来指定 **初始化方法** 和销毁方法。除此之外，Spring中是否还提供了其他方式来对bean实例进行初始化和销毁呢？

InitializingBean接口

InitializingBean接口概述

Spring中提供了一个InitializingBean接口，该接口为bean提供了属性初始化后的处理方法，它只包括afterPropertiesSet方法，凡是继承该接口的类，在bean的属性初始化后都会执行该方法。InitializingBean接口的源码如下所示。



```
2 * Copyright 2002-2012 the original author or authors.
16
17 package org.springframework.beans.factory;
18
20 * Interface to be implemented by beans that need to react once all their
35 public interface InitializingBean {
36
37     /**
38      * Invoked by a BeanFactory after it has set all bean properties supplied
39      * (and satisfied BeanFactoryAware and ApplicationContextAware).
40      * <p>This method allows the bean instance to perform initialization only
41      * possible when all bean properties have been set and to throw an
42      * exception in the event of misconfiguration.
43      * @throws Exception in the event of misconfiguration (such
44      * as failure to set an essential property) or if initialization fails.
45      */
46     void afterPropertiesSet() throws Exception;
47
48 }
49
```

根据InitializingBean接口中提供的afterPropertiesSet()方法的名字不难推断出，afterPropertiesSet()方法是在属性赋好值之后调用的。

那到底是不是这样的呢？下面我们来分析下afterPropertiesSet()方法的调用时机。

何时调用InitializingBean接口？

我们定位到Spring中的org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory这个类里面的invokeInitMethods()方法中，来查看Spring加载bean的方法。

题外话：不要问我为什么会是这个invokeInitMethods()方法，问我我也不知道，因为我就不熟悉Spring框架的源码，那就更加没看过它的源码了，可能有些小伙伴比较熟悉Spring框架的源码，那他自然就知道是这个invokeInitMethods()方法了。不熟悉Spring框架的源码的同学（比如笔者本人），也没关系，你只须定位到该方法中即可。所以，小伙伴们不要只顾着使用Spring，还是要多看看Spring的源码啊！Spring框架中使用了大量优秀的设计模型，其代码的编写规范和严谨程度也是业界开源框架中数一数二的，非常值得阅读。

我们来到AbstractAutowireCapableBeanFactory类中的invokeInitMethods()方法处，如下所示。

```
protected void invokeInitMethods(String beanName, final Object bean, RootBeanDefinition mbd)
    throws Throwable {
    // 判断该bean是否实现了InitializingBean接口。如果实现了InitializingBean接口。那么调用bean的afterPropertiesSet方法
    boolean isInitializingBean = (bean instanceof InitializingBean);
    if (isInitializingBean && (mbd == null || !mbd.isExternallyManagedInitMethod("afterPropertiesSet"))) {
        if (logger.isDebugEnabled()) {
            logger.debug("Invoking afterPropertiesSet() on bean with name '" + beanName + "'");
        }
        if (System.getSecurityManager() != null) {
            try {
                AccessController.doPrivileged(new PrivilegedExceptionAction<Object>() {
                    @Override
                    public Object run() throws Exception {
                        ((InitializingBean) bean).afterPropertiesSet(); // 调用bean的afterPropertiesSet方法
                        return null;
                    }
                }, getAccessControlContext());
            } catch (PrivilegedActionException pae) {
                throw pae.getException();
            }
        } else {
            ((InitializingBean) bean).afterPropertiesSet(); // 调用bean的afterPropertiesSet方法
        }
    }

    if (mbd != null) {
        String initMethodName = mbd.getInitMethodName();
        if (initMethodName != null && !(isInitializingBean && "afterPropertiesSet".equals(initMethodName)) &&
            !mbd.isExternallyManagedInitMethod(initMethodName)) {
            invokeCustomInitMethod(beanName, bean, mbd); // 通过反射的方式调用init-method
        }
    }
}
```

https://blog.csdn.net/yerenyuan_pku

分析上述代码后，我们可以初步得出如下信息：

1. Spring为bean提供了两种初始化的方式，实现InitializingBean接口（也就是要实现该接口中的afterPropertiesSet方法），或者在配置文件或@Bean注解中通过init-method来指定，两种方式可以同时使用。
2. 实现InitializingBean接口是直接调用afterPropertiesSet()方法，与通过反射调用init-method指定的方法相比，效率相对来说要高一点。但是init-method方式消除了对Spring的依赖。
3. 如果调用afterPropertiesSet方法时出错，那么就不会调用init-method指定的方法了。

也就是说Spring为bean提供了两种初始化的方式，第一种方式是实现InitializingBean接口（也就是要实现该接口中的afterPropertiesSet方法），第二种方式是在配置文件或@Bean注解中通过init-method来指定，这两种方式可以同时使用，同时使用先调用afterPropertiesSet方法，后执行init-method指定的方法。

DisposableBean接口

DisposableBean接口概述

实现org.springframework.beans.factory.DisposableBean接口的bean在销毁前，Spring将会调用DisposableBean接口的destroy()方法。也就是说我们可以实现DisposableBean这个接口来定义咱们这个销毁的逻辑。

我们先来看下DisposableBean接口的源码，如下所示。

```
2+ * Copyright 2002-2012 the original author or authors.
16
17 package org.springframework.beans.factory;
18
19 /**
20  * Interface to be implemented by beans that want to release resources
21  * on destruction. A BeanFactory is supposed to invoke the destroy
22  * method if it disposes a cached singleton. An application context
23  * is supposed to dispose all of its singletons on close.
24  *
25  * <p>An alternative to implementing DisposableBean is specifying a custom
26  * destroy-method, for example in an XML bean definition.
27  * For a list of all bean lifecycle methods, see the BeanFactory javadocs.
28  *
29  * @author Juergen Hoeller
30  * @since 12.08.2003
31  * @see org.springframework.beans.factory.support.RootBeanDefinition#getDestroyMethodName
32  * @see org.springframework.context.ConfigurableApplicationContext#close
33  */
34 public interface DisposableBean {
35
36     /**
37      * Invoked by a BeanFactory on destruction of a singleton.
38      * @throws Exception in case of shutdown errors.
39      * Exceptions will get logged but not rethrown to allow
40      * other beans to release their resources too.
41      */
42     void destroy() throws Exception;
43
44 }
45
```

可以看到，在DisposableBean接口中只定义了一个destroy()方法。

在bean生命周期结束前调用destroy()方法做一些收尾工作，亦可以使用destroy-method。前者与Spring耦合高，使用类型强转.方法名()，效率高；后者耦合低，使用反射，效率相对来说较低。

DisposableBean接口注意事项

多实例bean的生命周期不归Spring容器来管理，这里的DisposableBean接口中的方法是由Spring容器来调用的，所以如果一个多实例bean实现了DisposableBean接口是没有啥意义的，因为相应的方法根本不会被调用，当然了，在XML配置文件中指定了destroy方法，也是没有任何意义的。所以，在多实例bean情况下，Spring是不会自动调用bean的销毁方法的。

单实例bean案例

首先，创建一个Cat的类来实现InitializingBean和DisposableBean这俩接口，代码如下所示，注意该Cat类上标注了一个@Component注解。

```
1 package com.meimeixia.bean;
2
3 import org.springframework.beans.factory.DisposableBean;
4 import org.springframework.beans.factory.InitializingBean;
5 import org.springframework.stereotype.Component;
6
7 @Component
8 public class Cat implements InitializingBean, DisposableBean {
9
10     public Cat() {
11         System.out.println("cat constructor...");
12     }
13
14     /**
15      * 会在容器关闭的时候进行调用
16      */
17     @Override
18     public void destroy() throws Exception {
19         // TODO Auto-generated method stub
20         System.out.println("cat destroy...");
21     }
22
23     /**
24      * 会在bean创建完成，并且属性都赋值以后进行调用
25      */
26     @Override
27
```

```
28     public void afterPropertiesSet() throws Exception {  
29         // TODO Auto-generated method stub  
30         System.out.println("cat afterPropertiesSet...");  
31     }  
32 }
```

AI写代码java运行



然后，在MainConfigOfLifeCycle配置类中通过包扫描的方式将以上类注入到Spring容器中。

```
1 package com.meimeixia.config;  
2  
3 import org.springframework.context.annotation.Bean;  
4 import org.springframework.context.annotation.ComponentScan;  
5 import org.springframework.context.annotation.Configuration;  
6 import org.springframework.context.annotation.Scope;  
7  
8 import com.meimeixia.bean.Car;  
9  
10 @ComponentScan("com.meimeixia.bean")  
11 @Configuration  
12 public class MainConfigOfLifeCycle {  
13  
14     @Scope("prototype")  
15     @Bean(initMethod="init", destroyMethod="destroy")  
16     public Car car() {  
17         return new Car();  
18     }  
19  
20 }
```

AI写代码java运行



接着，运行IOCTest_LifeCycle类中的test01()方法，输出的结果信息如下所示。

```
<terminated> IOCTest_LifeCycle.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月1日 下午1:27:43)  
十二月 01, 2020 1:27:44 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext pre  
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: sta  
cat constructor...  
cat afterPropertiesSet...  
容器创建完成  
十二月 01, 2020 1:27:44 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doC  
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startu  
cat destroy...
```

从输出的结果信息中可以看出，单实例bean情况下，IOC容器创建完成后，会自动调用bean的初始化方法；而在容器销毁前，会自动调用bean的销毁方法。

多实例bean案例

多实例bean的案例代码基本与单实例bean的案例代码相同，只不过是在Cat类上添加了一个@Scope("prototype")注解，如下所示。

```
1 package com.meimeixia.bean;  
2  
3 import org.springframework.beans.factory.DisposableBean;  
4 import org.springframework.beans.factory.InitializingBean;  
5 import org.springframework.context.annotation.Scope;  
6 import org.springframework.stereotype.Component;  
7  
8 @Scope("prototype")  
9 @Component  
10 public class Cat implements InitializingBean, DisposableBean {  
11  
12     public Cat() {  
13         System.out.println("cat constructor...");  
14     }  
15 }
```

```

16
17 /**
18  * 会在容器关闭的时候进行调用
19  */
20 @Override
21 public void destroy() throws Exception {
22     // TODO Auto-generated method stub
23     System.out.println("cat destroy...");
24 }
25
26 /**
27  * 会在bean创建完成，并且属性都赋好值以后进行调用
28  */
29 @Override
30 public void afterPropertiesSet() throws Exception {
31     // TODO Auto-generated method stub
32     System.out.println("cat afterPropertiesSet...");
33 }
34 }

```

AI写代码java运行



然后，我们在IOCTest_LifeCycle类中新增一个test02()方法来进行测试，如下所示。

```

1 @Test
2 public void test02() {
3     // 1. 创建IOC容器
4     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfLifeCycle.class);
5     System.out.println("容器创建完成");
6     System.out.println("-----");
7
8     // 调用时创建对象
9     Object bean = applicationContext.getBean("cat");
10    System.out.println("-----");
11
12    // 调用时创建对象
13    Object bean1 = applicationContext.getBean("cat");
14    System.out.println("-----");
15
16    // 关闭容器
17    applicationContext.close();
18 }

```

AI写代码java运行



接着，运行IOCTest_LifeCycle类中的test02()方法，输出的结果信息如下所示。

```

<terminated> IOCTest_LifeCycle.test02 [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月1日 下午1:34:31)
十二月 01, 2020 1:34:32 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext pre
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: sta
容器创建完成
-----
cat constructor...
cat afterPropertiesSet...
-----
cat constructor...
cat afterPropertiesSet...
-----
十二月 01, 2020 1:34:32 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doC
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startu

```

从输出的结果信息中可以看出，在多实例bean情况下，Spring不会自动调用bean的销毁方法。