**Spring – Bean Life Cycle**

In this article, learn about **spring bean life cycle**. We will learn about life cycle stages, initialization and destroy call back methods. We will learn to control the bean life cycle events using XML configuration as well as annotation configuration.

Table of Contents

[1. Bean life cycle](https://howtodoinjava.com/spring-core/spring-bean-life-cycle/#1)

[2. Life cycle callback methods](https://howtodoinjava.com/spring-core/spring-bean-life-cycle/#2)

**1. Bean life cycle**

When container starts – a Spring bean needs to be instantiated, based on Java or XML bean definition. It may also be required to perform some post-initialization steps to get it into a usable state. *Same bean life cycle is for*[*spring boot*](https://howtodoinjava.com/spring-boot-tutorials/)*applications as well.*

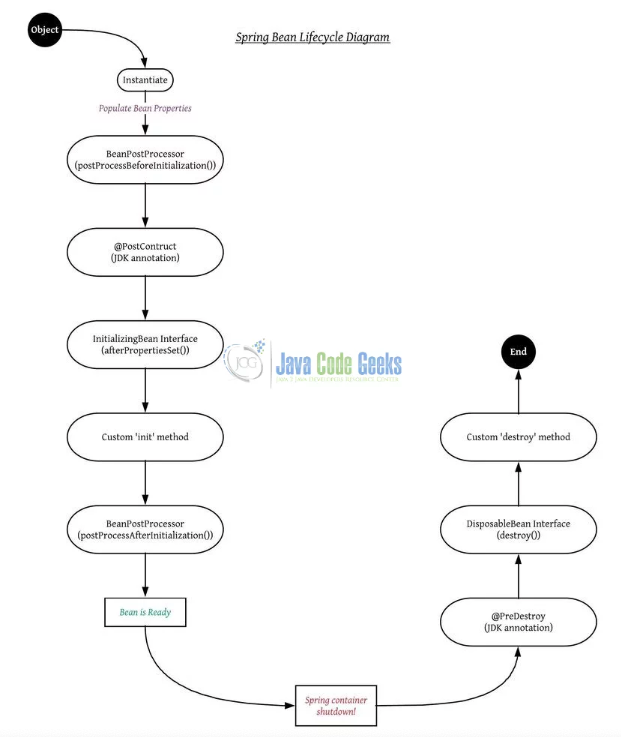
After that, when the bean is no longer required, it will be removed from the IoC container.

Spring bean factory is responsible for managing the life cycle of beans created through spring container.

**1.1. Life cycle call-backs**

Spring bean factory controls the creation and destruction of beans. To execute some custom code, it provides the call back methods which can be categorized broadly in two groups:

* **Post-initialization** call-back methods
* **Pre-destruction** call-back methods
  1. **Life cycle in diagram**

****



Spring Bean Life Cycle

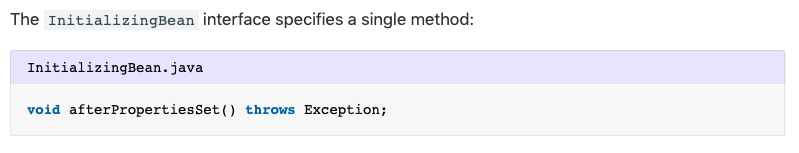
**2. Life cycle call-back methods**

Spring framework provides following **4 ways for controlling life cycle events** of a bean:

1. InitializingBean and DisposableBean callback interfaces
2. \*Aware interfaces for specific behaviour.
3. Custom init() and destroy() methods in bean xml configuration file.
4. @PostConstruct and @PreDestroy annotations

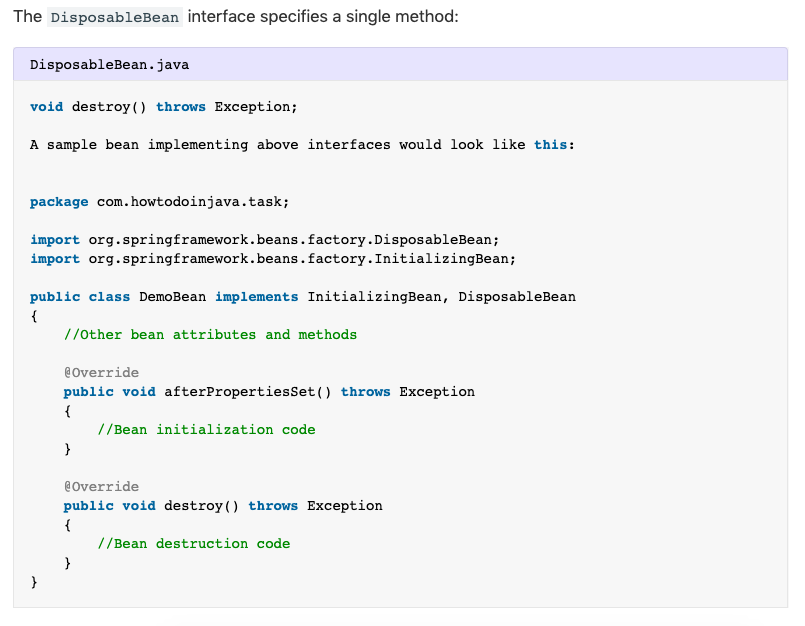
**2.1. InitializingBean and DisposableBean**

The [org.springframework.beans.factory.InitializingBean](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/beans/factory/InitializingBean.html) interface allows a bean to perform initialization work after all necessary properties on the bean have been set by the container.



This is not a preferable way to initialize the bean because it tightly couple your bean class with spring container. A better approach is to use “*init-method*” attribute in bean definition in applicationContext.xml file.

Similarly, implementing the [org.springframework.beans.factory.DisposableBean](http://static.springsource.org/spring/docs/1.2.9/api/org/springframework/beans/factory/DisposableBean.html) interface allows a bean to get a callback when the container containing it is destroyed.

****

**2.2. \*Aware interfaces for specific behavior**

Spring offers a range of \*Aware interfaces that allow beans to indicate to the container that they require a certain infrastructure dependency. Each interface will require you to implement a method to inject the dependency in bean.

These interfaces can be summarized as:

|  |  |  |
| --- | --- | --- |
| **AWARE INTERFACE** | **METHOD TO OVERRIDE** | **PURPOSE** |
| ApplicationContextAware | void setApplicationContext (ApplicationContext applicationContext) throws BeansException; | Interface to be implemented by any object that wishes to be notified of the ApplicationContext that it runs in. |
| ApplicationEventPublisherAware | void setApplicationEventPublisher (ApplicationEventPublisher applicationEventPublisher); | Set the ApplicationEventPublisher that this object runs in. |
| BeanClassLoaderAware | void setBeanClassLoader (ClassLoader classLoader); | Callback that supplies the bean class loader to a bean instance. |
| BeanFactoryAware | void setBeanFactory (BeanFactory beanFactory) throws BeansException; | Callback that supplies the owning factory to a bean instance. |
| BeanNameAware | void setBeanName(String name); | Set the name of the bean in the bean factory that created this bean. |
| BootstrapContextAware | void setBootstrapContext (BootstrapContext bootstrapContext); | Set the BootstrapContext that this object runs in. |
| LoadTimeWeaverAware | void setLoadTimeWeaver (LoadTimeWeaver loadTimeWeaver); | Set the LoadTimeWeaver of this object’s containing ApplicationContext. |
| MessageSourceAware | void setMessageSource (MessageSource messageSource); | Set the MessageSource that this object runs in. |
| NotificationPublisherAware | void setNotificationPublisher (NotificationPublisher notificationPublisher); | Set the NotificationPublisher instance for the current managed resource instance. |
| PortletConfigAware | void setPortletConfig (PortletConfig portletConfig); | Set the PortletConfig this object runs in. |
| PortletContextAware | void setPortletContext (PortletContext portletContext); | Set the PortletContext that this object runs in. |
| ResourceLoaderAware | void setResourceLoader (ResourceLoader resourceLoader); | Set the ResourceLoader that this object runs in. |
| ServletConfigAware | void setServletConfig (ServletConfig servletConfig); | Set the ServletConfig that this object runs in. |
| ServletContextAware | void setServletContext (ServletContext servletContext); | Set the ServletContext that this object runs in. |

Java program to show usage of aware interfaces to control string bean life cycle.

|  |
| --- |
| package com.howtodoinjava.task;    import org.springframework.beans.BeansException;  import org.springframework.beans.factory.BeanClassLoaderAware;  import org.springframework.beans.factory.BeanFactory;  import org.springframework.beans.factory.BeanFactoryAware;  import org.springframework.beans.factory.BeanNameAware;  import org.springframework.context.ApplicationContext;  import org.springframework.context.ApplicationContextAware;  import org.springframework.context.ApplicationEventPublisher;  import org.springframework.context.ApplicationEventPublisherAware;  import org.springframework.context.MessageSource;  import org.springframework.context.MessageSourceAware;  import org.springframework.context.ResourceLoaderAware;  import org.springframework.context.weaving.LoadTimeWeaverAware;  import org.springframework.core.io.ResourceLoader;  import org.springframework.instrument.classloading.LoadTimeWeaver;  import org.springframework.jmx.export.notification.NotificationPublisher;  import org.springframework.jmx.export.notification.NotificationPublisherAware;    public class DemoBean implements ApplicationContextAware,  ApplicationEventPublisherAware, BeanClassLoaderAware, BeanFactoryAware,  BeanNameAware, LoadTimeWeaverAware, MessageSourceAware,  NotificationPublisherAware, ResourceLoaderAware  {  @Override  public void setResourceLoader(ResourceLoader arg0) {  // TODO Auto-generated method stub  }    @Override  public void setNotificationPublisher(NotificationPublisher arg0) {  // TODO Auto-generated method stub    }    @Override  public void setMessageSource(MessageSource arg0) {  // TODO Auto-generated method stub  }    @Override  public void setLoadTimeWeaver(LoadTimeWeaver arg0) {  // TODO Auto-generated method stub  }    @Override  public void setBeanName(String arg0) {  // TODO Auto-generated method stub  }    @Override  public void setBeanFactory(BeanFactory arg0) throws BeansException {  // TODO Auto-generated method stub  }    @Override  public void setBeanClassLoader(ClassLoader arg0) {  // TODO Auto-generated method stub  }    @Override  public void setApplicationEventPublisher(ApplicationEventPublisher arg0) {  // TODO Auto-generated method stub  }    @Override  public void setApplicationContext(ApplicationContext arg0)  throws BeansException {  // TODO Auto-generated method stub  }  } |

**2.3. Custom init() and destroy() methods**

The default init and destroy methods in bean configuration file can be defined in two ways:

* **Bean local definition** applicable to a single bean
* **Global definition** applicable to all beans defined in beans context

**2.3.1. Bean local definition**

Local definition is given as below.



**2.3.2. Global definition**

Where’as global definition is given as below. These methods will be invoked for all bean definitions given under <beans> tag. They are useful when you have a pattern of defining common method names such as init() and destroy() for all your beans consistently. This feature helps you in not mentioning the init and destroy method names for all beans independently.



Java program to show methods configured in bean XML configuration file.

**2.4. @PostConstruct and @PreDestroy**

Spring 2.5 onwards, you can use annotations also for specifying life cycle methods using @PostConstruct and @PreDestroy annotations.

* @PostConstruct annotated method will be invoked after the bean has been constructed using default constructor and just before it’s instance is returned to requesting object.
* @PreDestroy annotated method is called just before the bean is about be destroyed inside bean container.

Java program to show usage of **annotation configuration** to control using annotations.



So this is all about **spring bean life cycle** inside Spring container. Remember given types of life cycle events, it is a commonly asked [spring interview question](https://howtodoinjava.com/interview-questions/top-spring-interview-questions-with-answers/).