**Spring Bean Life Cycle Callback Methods**

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

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

**1) InitializingBean and DisposableBean callback interfaces**

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.

The InitializingBean interface specifies a single method:

|  |
| --- |
| InitializingBean.java |
| void afterPropertiesSet() throws Exception; |

This is not a preferrable 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.

The DisposableBean interface specifies a single method:

|  |
| --- |
| DisposableBean.java |
| void destroy() throws Exception;    A sample bean implementing above interfaces would look like this:      package com.howtodoinjava.task;    import org.springframework.beans.factory.DisposableBean;  import org.springframework.beans.factory.InitializingBean;    public class DemoBean implements InitializingBean, DisposableBean  {      //Other bean attributes and methods        @Override      public void afterPropertiesSet() throws Exception      {          //Bean initialization code      }        @Override      public void destroy() throws Exception      {          //Bean destruction code      }  } |

**2) Other 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 ApplicationContextthat 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. |

A sample implementation will look like this:

|  |
| --- |
| DemoBean.java |
| 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      }  } |

**3) Custom init() and destroy() methods in bean configuration file**

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

Local definition is given as below.

|  |
| --- |
| beans.xml |
| <beans>        <bean id="demoBean" class="com.howtodoinjava.task.DemoBean"                      init-method="customInit"                      destroy-method="customDestroy"></bean>    </beans> |

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.

|  |
| --- |
| <beans default-init-method="customInit" default-destroy-method="customDestroy">            <bean id="demoBean" class="com.howtodoinjava.task.DemoBean"></bean>    </beans> |

A sample implementation for this type of life cycle will be:

|  |
| --- |
| DemoBean.java |
| package com.howtodoinjava.task;    public class DemoBean  {      public void customInit()      {          System.out.println("Method customInit() invoked...");      }        public void customDestroy()      {          System.out.println("Method customDestroy() invoked...");      }  } |

**4) @PostConstruct and @PreDestroy annotations**

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.

A sample implementation will look like this:

|  |
| --- |
| package com.howtodoinjava.task;    import javax.annotation.PostConstruct;  import javax.annotation.PreDestroy;    public class DemoBean  {      @PostConstruct      public void customInit()      {          System.out.println("Method customInit() invoked...");      }        @PreDestroy      public void customDestroy()      {          System.out.println("Method customDestroy() invoked...");      }  } |

So this is all about bean life cycle management inside Spring container. I hope that it has added some more knowledge in your kitty.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Spring Bean Post Processors BeanPostProcessor gives you a way to do some operations before creating the spring bean and immediately after creating the spring bean. This allows you to add some custom logic before and after spring bean creation. The BeanPostProcessors operate on bean (or object) instances which means that the Spring IoC container instantiates a bean instance and then BeanPostProcessor interfaces do their work.  Notice that the init and destroy methods related to bean are different from bean post processors. BeanPostProcessors are common for all beans. This example clearly shows the difference from them.  To implement bean post processor logic, we need to create a class which implements BeanPostProcessor interface and two of its methods.   |  |  |  | | --- | --- | --- | | [?](http://www.java2novice.com/spring/bean-post-processors/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | package com.java2novice.beans;    import org.springframework.beans.BeansException;  import org.springframework.beans.factory.config.BeanPostProcessor;    public class MyBeanInitProcessor implements BeanPostProcessor{        @Override      public Object postProcessAfterInitialization(Object bean, String beanName)              throws BeansException {          System.out.println("before initialization: "+beanName);          return bean;      }        @Override      public Object postProcessBeforeInitialization(Object bean, String beanName)              throws BeansException {          System.out.println("after initialization: "+beanName);          return bean;      }  } | |   Bean class:   |  |  |  | | --- | --- | --- | | [?](http://www.java2novice.com/spring/bean-post-processors/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | package com.java2novice.beans;    import java.io.IOException;  import java.net.HttpURLConnection;  import java.net.MalformedURLException;  import java.net.URL;    import javax.annotation.PostConstruct;  import javax.annotation.PreDestroy;    public class NetworkManager{        private HttpURLConnection connection;      private String urlStr;        public void setUrlStr(String urlStr){          this.urlStr = urlStr;      }        @PostConstruct      public void init(){            System.out.println("Inside init() method...");          URL obj;          try {              obj = new URL(this.urlStr);              //initialize http connection here              this.connection = (HttpURLConnection) obj.openConnection();          } catch (MalformedURLException e) {              // TODO Auto-generated catch block              e.printStackTrace();          } catch (IOException e) {              // TODO Auto-generated catch block              e.printStackTrace();          }      }        @PreDestroy      public void destroy(){          try{              System.out.println("Inside destroy() method...");              if(this.connection != null) {                  connection.disconnect();              }          } catch(Exception ex){            }      }        public void readData(){          try {              int responseCode = this.connection.getResponseCode();              System.out.println("Response code: "+responseCode);              /\*\*               \* do your business logic here               \*/          } catch (IOException e) {              // TODO Auto-generated catch block              e.printStackTrace();          }      }  } | |   Xml based configuration file, note that we registered BeanPostProcessor implementor class here:   |  |  |  | | --- | --- | --- | | [?](http://www.java2novice.com/spring/bean-post-processors/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <beans xmlns="<http://www.springframework.org/schema/beans>"      xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"      xmlns:context="<http://www.springframework.org/schema/context>"      xsi:schemaLocation="http://www.springframework.org/schema/beans      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd      http://www.springframework.org/schema/context      http://www.springframework.org/schema/context/spring-context-3.0.xsd">        <context:annotation-config />        <bean id="netManager" class="com.java2novice.beans.NetworkManager">          <property name="urlStr" value="<http://www.google.com/search?q=java2novice>"/>      </bean>        <bean class="com.java2novice.beans.MyBeanInitProcessor" />  </beans> | |   Spring bean demo class:   |  |  |  | | --- | --- | --- | | [?](http://www.java2novice.com/spring/bean-post-processors/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package com.java2novice.test;    import org.springframework.context.ConfigurableApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    import com.java2novice.beans.NetworkManager;    public class SpringDemo {        public static void main(String a[]){          String confFile = "applicationContext.xml";          ConfigurableApplicationContext context                              = new ClassPathXmlApplicationContext(confFile);          NetworkManager networkMng = (NetworkManager) context.getBean("netManager");          networkMng.readData();          context.close();      }  } | | |
| |  | | --- | | **Output:** | | after initialization: netManager  Inside init() method...  before initialization: netManager  Response code: 403  Inside destroy() method... | |