Spring Bean Life cycle

## 1.Introduction of bean

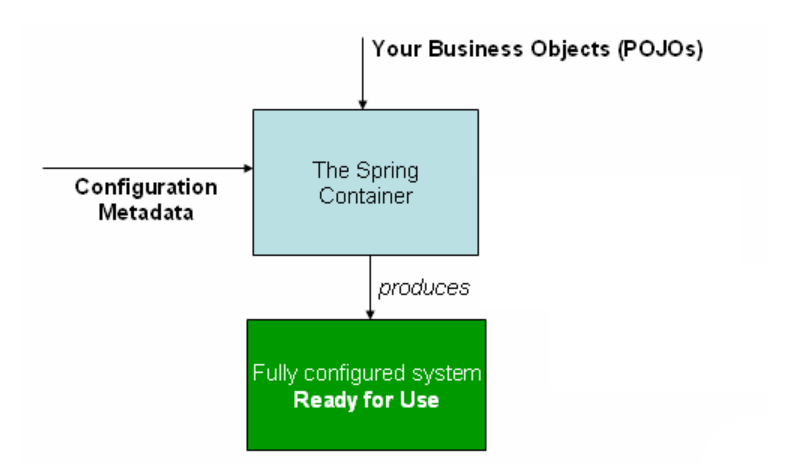
What is bean?

* It should have a pre-defined no argument constructor.
* It should be Serializable.
* It should provide methods to set and get the values of the properties, known as getter and setter methods.

## 1.1 Spring Bean

What is spring bean ?

Spring beans are object instances which is managed by the Spring IOC container. They’re created based on BeanDefinition objects.



## Java Bean VS Spring Bean

* Spring bean is managed by Spring IOC, Java Bean is not.
* Java Bean is always serializable, Spring Bean doesn’t need to.
* Java Bean must have a default no-arg constructor, Spring Bean doesn’t need to.
* A Java object can be a JavaBean, a POJO and a Spring bean all at the same time.

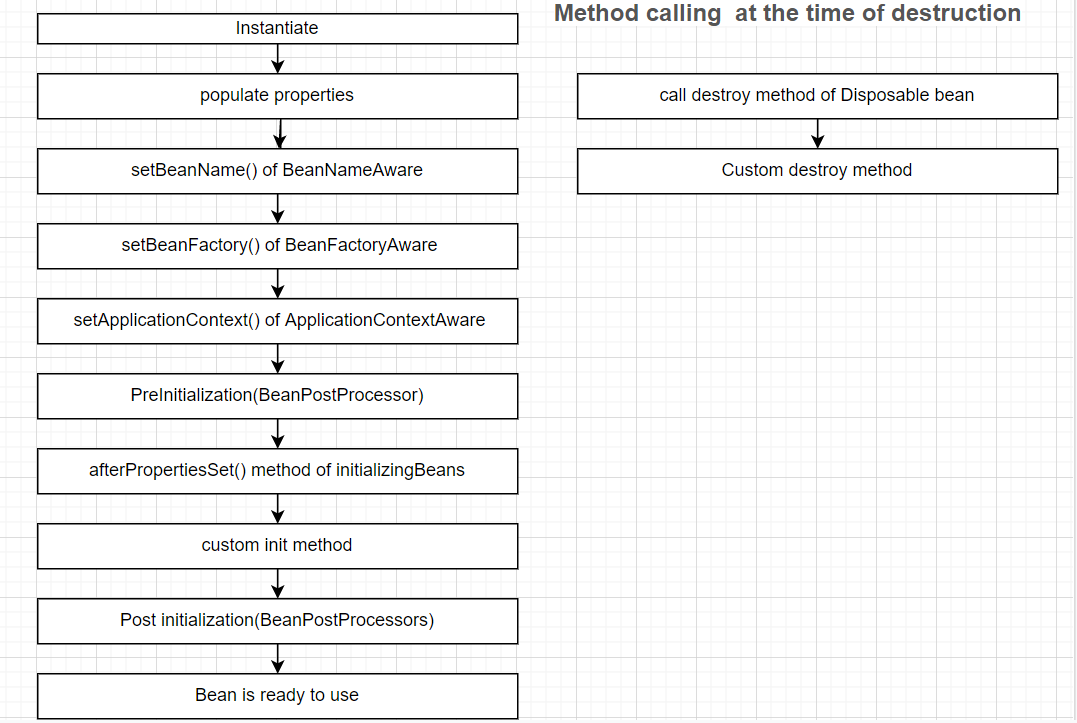
## 2. Spring bean life cycle and its call back method

#### 2.1 How spring bean(s) interact with container?

For interaction with the container's management of the bean lifecycle, you can implement the Spring InitializingBean and DisposableBean interfaces. The container calls afterPropertiesSet() for InitializingBean  and destroy() for DisposableBean  to allow the bean to perform certain actions upon initialization and destruction of your beans.

Internally, the Spring Framework uses BeanPostProcessor implementations to process any callback interfaces it can find and call the appropriate methods. If you need custom features of lifecycle behavior Spring does not support out-of-the-box, you can implement a BeanPostProcessor yourself.

#### 2.1.1 Spring bean lifecycle pictorial view



#### 2.1.1 InitializingBean and DisposbleBean callback interfaces

* org.springframework.beans.factory.InitalizingBean

Method to override : void afterPropertiesSet() throws Exception;

* org.springframework.beans.factory.DisposableBean

Method to override: void destroy() throws Exception;

This approach is simple to use but it’s **not recommended** because it will create tight coupling with the Spring framework in our bean implementations.

#### 2.2 Sample java code for implementation of bean lifecycle methods

#### 2.2.1 Java sample code to implement InitalizingBean and DisposableBean interface

import org.springframework.beans.factory.DisposableBean;

import org.springframework.beans.factory.InitializingBean;

public class SampleBean implements InitializingBean,DisposableBean{

private String name;

  public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

public SampleBean() {

System.out.println("Constructor of Sample bean is called ");

}

@Override

public void destroy() throws Exception {

System.out.println("destroy method of Sample bean is called ");

}

@Override

public void afterPropertiesSet() throws Exception {

System.out.println("afterPropertiesSet method of Sample bean is called !! ");

}

}

#### 2.2.1.2 Sample code of Beans.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="<http://www.springframework.org/schema/beans>"

     xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

     xsi:schemaLocation="http://www.springframework.org/schema/beans

     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

     <bean id="sampleBean" class=" SampleBean" >

         <property name="name" value="Dummy Name"/>

         </bean>

</beans>

#### 2.2.1.3 Class to load beans.xml

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestSampleBean {

    public static void main(String[] args) {

       ApplicationContext context =

                new ClassPathXmlApplicationContext("beans.xml");

       SampleBean bean = (PersonBean)context.getBean("sampleBean");

           System.out.println(bean.getName());

       ((AbstractApplicationContext) context).registerShutdownHook();

    }

}

## 2.2.2 Custom init() and destroy() methods in bean configuration file

Xml based sample example

<beans>

    <bean id="bean\_id" class="bean.class"

      init-method="customSingleclassInitmethod"

     destroy-method="customsingleclassDestroymethod">

  </bean>

</beans>

We can configure the **default init-method  and destroy-method** which will be applied on all the beans .They are useful when we have a pattern of defining common method names such as init() and destroy() for all your beans consistently.

<beans default-init-method=”customGlobalDefaultInitMethod” default-destroy-method=”customGlobalDefaultDestroyMethod” >

    <bean id="bean\_id" class="bean.class" >

</bean>

</beans>

#### 2.2.2.1 Java sample Code Example for custom init and destroy method

public class CustomSingleClassLifeCycleMethodBean {

private String name;

    public CustomSingleClassLifeCycleMethodBean()

    {

        System.out.println("Constructor called ");

    }

    public void customDestroy() throws Exception {

        System.out.println("Bean’s custom destroy method called ");

    }

    public void customInit() throws Exception {

        System.out.println("Bean’s custom Init method called");

    }

     public String getName() {

        return name;

    }

     public void setName(String name) {

        this.name = name;

    }

 }

#### 2.2.2.1.2 Sample code of Beans.xml

<?xml version="1.0" encoding="UTF-8"?>

 <beans xmlns="<http://www.springframework.org/schema/beans>"

    xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

        <bean id=" customSingleClassLifeCycleMethodBean "

               class=" CustomSingleClassLifeCycleMethodBean "

               init-method="customInit"

               destroy-method="customDestroy">

        <property name="name" value="custom bean" ></property>

        </bean>

</beans>

#### 2.2.2.1.3 sample Class to load bean

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestClassLifeCycleBean {

     public static void main(String[] args) {

        ApplicationContext context =

            new ClassPathXmlApplicationContext("beans.xml");

         CustomLifeCycleMethodBean bean = (customSingleClassLifeCycleMethodBean)context.getBean("customSingleClassLifeCycleMethodBean");

           ((AbstractApplicationContext) context).registerShutdownHook();

      }

}

## 2.2.2.2.1 Java sample example to configure global init and destroy methods in bean

public class CustomGlobalLifeCycleMethodBean {

    public CustomGlobalLifeCycleMethodBean()

   {

    System.out.println("Constructor of  bean is called !! ");

   }

   public void globalCustomDestroy() throws Exception {

    System.out.println("Bean’s global custom destroy method called");

   }

   public void globalCustomInit() throws Exception {

    System.out.println("Bean’s global custom Init  method called");

   }

}

## 2.2.2.2.2 Sample code of Beans.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="<http://www.springframework.org/schema/beans>"

    xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"

    default-init-method="globalCustomInit"

    default-destroy-method="globalCustomDestroy">

      <bean id="customGlobalLifeCycleMethodBean"

          class="CustomGlobalLifeCycleMethodBean" />

</beans>

## 2.2.2.2.3 Class to load beans.xml and test custom class

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestCustomGlobalMethodLifeCycleBean {

    public static void main(String[] args) {

           ApplicationContext context =

                new ClassPathXmlApplicationContext("beans.xml");

           CustomGlobalLifeCycleMethodBean bean = (CustomGlobalLifeCycleMethodBean)context.getBean("customGlobalLifeCycleMethodBean");

           ((AbstractApplicationContext) context).registerShutdownHook();

     }

## 2.2.3 Bean Name, Factory, Application context Aware interfaces

* org.springframework.beans.factory.BeanFactoryAware # Interface

void setBeanFactory(BeanFactory beanFactory) throws BeansException; # method to override

* org.springframework.beans.factory.BeanNameAware # Interface

void setBeanName(String name); # method to override

* org.springframework.context.ApplicationContextAware # Interface

void setApplicationContext(ApplicationContext applicationContext) throws BeansException; # method to override

#### 2.2.3.1 sample java example code to implement ApplicationContextAware, BeanNameAware, BeanFactoryAware

import java.util.Arrays;

import org.springframework.beans.BeansException;

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;

public class AlertBean implements ApplicationContextAware,BeanNameAware,BeanFactoryAware{

@Override

public void setBeanFactory(BeanFactory beanFactory) throws BeansException {

       System.out.println("setBeanFactory method of AlertBean is called");

       System.out.println("setBeanFactory:: AlertBean singleton="

                + beanFactory.isSingleton("alertBean"));

}

@Override

public void setBeanName(String beanName) {

     System.out.println("setBeanName method of AlertBean is called");

     System.out.println("setBeanName:: Bean Name defined in context="

                    + beanName);

}

@Override

public void setApplicationContext(ApplicationContext applicationContext)

        throws BeansException {

        System.out.println("setApplicationContext method of AlertBean is called");

        System.out.println("setApplicationContext:: Bean Definition Names="

                + Arrays.toString(applicationContext.getBeanDefinitionNames()));

}

}

#### 2.2.3.1.2 Beans.xml to inject AlertBean.class as bean

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="<http://www.springframework.org/schema/beans>"

    xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id=" alertBean " class=" AlertBean " >

    </bean>

</beans>

## 2.2.3.1.3 Sample Class to load bean and test

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestAlertBean {

    public static void main(String[] args) {

           ApplicationContext context =

                new ClassPathXmlApplicationContext("beans.xml");

           AlertBean bean = (AlertBean)context.getBean("alertBean");

           ((AbstractApplicationContext) context).registerShutdownHook();

     }

}

## 2.2.3.2 Aware Interfaces to Add Specific Behavior

|  |  |  |
| --- | --- | --- |
| 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. |
|  |  |  |

## 2.2.4 Annotation based customization

@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. If initialized inside constructor, you will get NPE.

@PreDestroy annotated method is invoked just before the bean is about be destroyed inside bean container.

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

package com.sample.example;

import javax.annotation.PostConstruct;

import javax.annotation.PreDestroy;

public class DemoBean

{

@PostConstruct

public void customInit()

{

System.out.println("customInit() Method invoked...");

}

@PreDestroy

public void customDestroy()

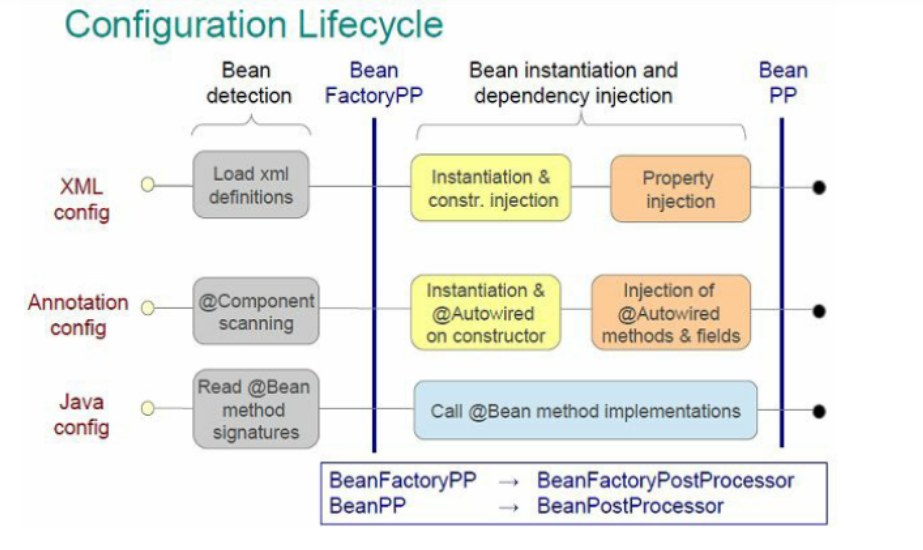
{

System.out.println("customDestroy() Method invoked...");

}

}

## 2.2.5 Configuration lifecycle with BeanPostProcessor and BeanfactoryPostProcessor



#### 2.2.5.1 Sample code for bean postprocessor

package com.sample;

public class Sample

{

    public void init()

    {

        System.out.println("In init block of sample");

    }

    public void destroy()

    {

        System.out.println("In destroy block of sample");

    }

}

#### 2.2.5.1.2 Custom Processor class

package com.sample;

import org.springframework.beans.BeansException;

import org.springframework.beans.factory.config.BeanPostProcessor;

public class InitSamplePostProcessor implements BeanPostProcessor{

    public Object postProcessAfterInitialization(Object bean, String beanName)

            throws BeansException {

        System.out.println("AfterInitialization : " + beanName);

        return bean;  // you can return any other object as well

    }

    public Object postProcessBeforeInitialization(Object bean, String beanName)

            throws BeansException {

        System.out.println("BeforeInitialization : " + beanName);

        return bean;  // you can return any other object as well

    }

}

#### 2.2.5.1.3 beans.xml for calling the custom processor

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="sample" class=" com.sample.Sample" init-method="init" destroy-method="destroy">

</bean>

      <bean class=" com.sample.InitSamplePostProcessor"/>

 </beans>

#### 2.2.5.1.4 Main class for calling bean

package com.sample;

import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class BeanPostProcessorExampleMain{

public static void main(String[] args) {

AbstractApplicationContext appContext = new ClassPathXmlApplicationContext("beans.xml");

Sample sampleObj = (Sample) appContext.getBean("sample");

appContext.registerShutdownHook();

}

}

**Points to remember:**

Spring 2.5 onwards we can use annotations to specify life cycle methods using @PostConstruct and @PreDestroy annotations.

If you use java 9+ then add dependency **javax.annotation:javax.annotation-api Or use spring-boot 2** .