Spring Bean Life Cycle

APRIL 2, 2018 BY PANKAJ — 24 COMMENTS

Today we will look into Spring Bean Life Cycle. Spring Beans are the most important part of any Spring application. Spring **ApplicationContext** is responsible to initialize the Spring Beans defined in spring bean configuration file.

Table of Contents

Spring Bean Life Cycle

Spring Context is also responsible for injection dependencies in the bean, either through setter or constructor methods or by spring autowiring.

Sometimes we want to initialize resources in the bean classes, for example creating database connections or validating third party services at the time of initialization before any client request. Spring framework provide different ways through which we can provide post-initialization and pre-destroy methods in a spring bean life cycle.

1. By implementing **InitializingBean** and **DisposableBean** interfaces – Both these interfaces declare a single method where we can initialize/close resources in the bean. For post-initialization, we can implement `InitializingBean` interface and provide implementation of `afterPropertiesSet()` method. For pre-destroy, we can implement `DisposableBean` interface and provide implementation of `destroy()` method. These methods are the callback methods and similar to servlet listener implementations.
   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. Providing **init-method** and **destroy-method** attribute values for the bean in the spring bean configuration file. This is the recommended approach because of no direct dependency to spring framework and we can create our own methods.

Note that both *post-init* and *pre-destroy* methods should have no arguments but they can throw Exceptions. We would also require to get the bean instance from the spring application context for these methods invocation.

Spring Bean Life Cycle – @PostConstruct, @PreDestroy Annotations

Spring framework also support `@PostConstruct` and `@PreDestroy` annotations for defining post-init and pre-destroy methods. These annotations are part of `javax.annotation` package. However for these annotations to work, we need to configure our spring application to look for annotations. We can do this either by defining bean of type `org.springframework.context.annotation.CommonAnnotationBeanPostProcessor` or by `context:annotation-config` element in spring bean configuration file.

Let's write a simple Spring application to showcase the use of above configurations for spring bean life cycle management. Create a Spring Maven project in Spring Tool Suite, final project will look like below image.

Spring Bean Life Cycle – Maven Dependencies

We don't need to include any extra dependencies for configuring spring bean life cycle methods, our pom.xml file is like any other standard spring maven project.

Spring Bean Life Cycle – Model Class

Let's create a simple java bean class that will be used in service classes.

Spring Bean Life Cycle – InitializingBean, DisposableBean

Let's create a service class where we will implement both the interfaces for post-init and pre-destroy methods.

Spring Bean Life Cycle – Custom post-init, pre-destroy

Since we don't want our services to have direct spring framework dependency, let's create another form of Employee Service class where we will have post-init and pre-destroy spring life cycle methods and we will configure them in the spring bean configuration file.

We will look into the spring bean configuration file in a bit. Before that let's create another service class that will use @PostConstruct and @PreDestroy annotations.

Spring Bean Life Cycle – @PostConstruct, @PreDestroy

Below is a simple class that will be configured as spring bean and for post-init and pre-destroy methods, we are using @PostConstruct and @PreDestroy annotations.

Spring Bean Life Cycle – Configuration File

Let's see how we will configure our beans in spring context file.

Notice that I am not initializing employee name in it's bean definition. Since EmployeeService is using interfaces, we don't need any special configuration here.

For MyEmployeeService bean, we are using init-method and destroy-method attributes to let spring framework know our custom methods to execute.

MyService bean configuration doesn't have anything special, but as you can see that I am enabling annotation based configuration for this.

Our application is ready, let's write a test program to see how different methods get executed.

Spring Bean Life Cycle – Test Program

When we run above test program, we get below output.

Spring Bean Life Cycle Important Points:

- From the console output it's clear that Spring Context is first using no-args constructor to initialize the bean object and then calling the post-init method.
- The order of bean initialization is same as it's defined in the spring bean configuration file.
- The context is returned only when all the spring beans are initialized properly with post-init method executions.
- Employee name is printed as "Pankaj" because it was initialized in the post-init method.
- When context is getting closed, beans are destroyed in the reverse order in which they were initialized i.e in LIFO (Last-In-First-Out) order.

You can uncomment the code to get bean of type `MyEmployeeService` and confirm that output will be similar and follow all the points mentioned above.

Spring Aware Interfaces

Sometimes we need Spring Framework objects in our beans to perform some operations, for example reading ServletConfig and ServletContext parameters or to know the bean definitions loaded by the ApplicationContext. That's why spring framework provides a bunch of *Aware interfaces that we can implement in our bean classes.

`org.springframework.beans.factory.Aware` is the root marker interface for all these Aware interfaces. All of the *Aware interfaces are sub-interfaces of Aware and declare a single setter method to be implemented

by the bean. Then spring context uses setter-based dependency injection to inject the corresponding objects in the bean and make it available for our use.

Spring Aware interfaces are similar to servlet listeners with callback methods and implementing observer design pattern.

Some of the important Aware interfaces are:

- **ApplicationContextAware** – to inject ApplicationContext object, example usage is to get the array of bean definition names.
- **BeanFactoryAware** – to inject BeanFactory object, example usage is to check scope of a bean.
- **BeanNameAware** – to know the bean name defined in the configuration file.
- **ResourceLoaderAware** – to inject ResourceLoader object, example usage is to get the input stream for a file in the classpath.
- **ServletContextAware** – to inject ServletContext object in MVC application, example usage is to read context parameters and attributes.
- **ServletConfigAware** – to inject ServletConfig object in MVC application, example usage is to get servlet config parameters.

Let's see these Aware interfaces usage in action by implementing few of them in a class that we will configure as spring bean.

Spring *Aware Example Configuration File

Very simple spring bean configuration file.

Spring *Aware Test Program

Now when we execute above class, we get following output.

Console output of the test program is simple to understand, I won't go into much detail about that.

That's all for the Spring Bean Life Cycle methods and injecting framework specific objects into the spring beans. Please download sample project from below link and analyze it to learn more about them.

Download Spring Bean Lifycycle Project

« PREVIOUS
Spring @Autowired Annotation

NEXT »
Spring MVC Exception Handling – @ControllerAdvice, @ExceptionHandler, HandlerExceptionResolver

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: SPRING

Comments

Monir says
DECEMBER 20, 2017 AT 12:42 AM

@Pankaj,
Very special thanks for a details about bean scope.
What is the best way to contact you?
Reply

Tony says
SEPTEMBER 19, 2017 AT 7:42 AM
Hi Pankaj, excellent… , thank you so much for the detailed tutorial
Reply

Selva says
MAY 27, 2017 AT 5:28 PM
Great article !
Reply

kanchan tiwari says
OCTOBER 31, 2016 AT 9:51 AM
Great explanation !!! thanks for the guide..
Reply

Mengying Ye says
JULY 11, 2016 AT 4:15 AM
Thanks for your post, it's really great ,could I translate it into chinese?
Reply

shripad says
JUNE 9, 2016 AT 12:20 AM
Nice article…and thanks for such a nice description !!! □
Reply

Pankaj says
JUNE 9, 2016 AT 6:41 AM

Thanks Shri for nice words.

Reply

Pranita says

JUNE 5, 2016 AT 6:08 AM

Hi Sir,

Thanks for this ultimate post. Once any one read this article, no need to visit any other tutorial.

Reply

Pankaj says

JUNE 5, 2016 AT 11:34 AM

Thanks for the nice words Pranita.

Reply

Jagadeesh says

MAY 24, 2016 AT 10:48 PM

Excellent Pakaj.. Good Job..

Reply

Pankaj says

JUNE 2, 2016 AT 12:39 PM

Thanks Jagadeesh, appreciate your kind word.

Reply

DR says

FEBRUARY 16, 2016 AT 7:36 AM

Thanks For all the tutorials

Reply

Pankaj says

JUNE 2, 2016 AT 12:40 PM

you are welcome buddy.
Reply

anil says
OCTOBER 5, 2015 AT 11:29 PM
Thank you sir……
Reply

Pankaj says
JUNE 2, 2016 AT 12:40 PM
welcome Anil.
Reply

Raju.m says
AUGUST 4, 2015 AT 9:39 PM
Hi pankaj,
This is very interesting to note all point .can u pls share me spring qeb service in soap . Thanks in advance
Reply

Pankaj says
JUNE 2, 2016 AT 12:41 PM
I have written a lot on soap web services, spring don't support SOAP web services, they provide support for REST web services. I have written about them in detail, please search and read. □
Reply

Vishal Upadhyay says
JANUARY 14, 2015 AT 12:19 AM
A good explanation.
Reply

Pankaj says
JUNE 2, 2016 AT 12:41 PM
Thanks Vishal.
Reply

Ganesh says
NOVEMBER 7, 2014 AT 1:00 AM
Thanx sir.....
Reply

Pankaj says
JUNE 2, 2016 AT 12:41 PM
You are welcome Ganesh.
Reply

johnybasha shaik says
APRIL 2, 2014 AT 7:49 PM
Hi Pankaj.. I regularly follow your website. You are doing a great job for all the followers of this site in learing java/j2ee. I have a suggestion. Though the content you are publishing is very good and with in depth analysis. The way the articles structured are not user friendly. I mean to say. the articles are not as per index. For example, if I go to design patterns, user can't see which list goes under creational patterns and which does go to behavioural unless I open and read each pattern. If I come to spring tutorial, I dont know which article should I start first. My suggestion, is please publish them in a sequential order to make reading user friendly. I hope you take this advice positively.
Thanks.
Reply

Pankaj says
APRIL 2, 2014 AT 11:16 PM
Thanks for the feedback. Usually when I am done with a topic, I write a summary post with the order of articles to read.
For example summary post for design patterns: https://www.journaldev.com/1827/java-design-patterns-example-tutorial
Summary post for Struts2: https://www.journaldev.com/2310/struts2-tutorial-with-example-projects

Including all the links in every post is cumbersome and not a good idea too. Imagine you start reading a post and half of the page is with 20 link to other articles. I will write a summary post for Spring Tutorial too once I am done with most of the articles in the series.
Reply

Pankaj says
JUNE 2, 2016 AT 12:43 PM
Spring index post is ready, please go through https://www.journaldev.com/2888/spring-tutorial-spring-core-tutorial
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Current ye@r *

5.2

Search for tutorials...

SPRING FRAMEWORK

Spring Tutorial

Spring Core

- Spring Framework
- Spring Dependency Injection
- Spring IoC and Bean
- Spring Bean Life Cycle
- Spring REST
- Spring REST XML
- Spring RestTemplate
- Spring AOP
- Spring AOP Method Profiling
- Spring Annotations
- Spring @Autowired
- Spring @RequestMapping

Spring MVC

- Spring MVC Example
- Spring MVC Tutorial
- Spring MVC Exception Handling
- Spring MVC Validator
- Spring MVC Intercepter
- Spring MVC File Upload
- Spring MVC i18n
-