Spring Bean life cycle

=> Life cycle means keeping track of all the activities from birth to dealth (object creation to object destruction)

=> IOC container manages the spring bean life cycle

=> Serlvet container managees the servlet comp life cycle

=> Jsp container manages the jsp comp life cycle

=> The action raised on the comp or object in the life cycle of comp is called life cycle event event.

Servlet comp life cycle events are

=>instantiation event (raises when the servlet contianer creates our servlet comp class object) => Request processing event (raises when the servlet container keeps our servlet comp ready to process the reuquest when it gets the request from client app)

IOC container life cycle activities on spring bean are

=> Destruction event (Raises when the servlet container is about to destroy our servlet comp class object)

Spring bean life cycle events are

=>Instatiation event (init- event) [raises when the IOC container creates the spring bean class object and completes all types of Injections on that object]

=> Destruction event (destroy-event) [raises when the IOC container is about to destroy our spring bean class object]

=> The method that is called by underlying container automatically when the life cycle event is raised is called Life cycle event handling method or life cycle method .. we generally place event handling logics in life cycle event handling method definitations in our choice comps.

Servlet comp life cycle methods are

lifecycle event

a) For instantiation event

b) Request Processing event

=> Loading spring bean class

=> instantiating the spring bean

=> managing the spring bean class obj

=> calling the life cycle methods

=> Destroying the spring bean class obj

=> JVM's Garbage collector algorithm is mark and sweep algorithm i.e first marks the objs that are not having any references later destroys the all the those objects

=> Servlet container,jsp container, IOC container uses their own Garbage collector (not the JVM's Garbage collector) to destroy spring bean class obj becoz the IOC container needs to destroy the spring bean class obj though object is in utilization

The life cycle methods are called callback methods becoz we do not call them i.e they will be called 'underlying container automatically based on the life cycle events that are raised

life cycle method

logics to execute

init(ServletConfig cg) method

--> we place servlet initialization logics in this method like creating jdbc con object and etc...

service(ServletRequest req, ServletResponse res) method

we place request processing logics and response generation logics in this method

what is the need of Life cycle methods in any Container managed comp development?

of the

Ans) Generally, the Container manages the whole life cycle comp right from loading the class to, creating the object to managing the object to destroying the object.. In this process the programmer wants to execute his choice logics For this the container says override the life cycle methods in the comp having programmer choice logics.. So that those logics will be executed automatically when the Containers calls the life cycle methods for the life cycle events.

c) Destruction event

destroy()

we place logics to release non-java resources associated with the servlet comp (closing jdbc connection)

=> servlet comp is invasive comp (class must implement javax.servlet.Servlet(1) directly or indirectly)..So the servlet life cycle methods are pre-defined methods with fixed signatures..

=>spring bean life cycle method names are not fixed becoz spring beans can be developed as non-invasive classses

In

=>Most of the times we prefer taking spring beans as the non-invasive classes.

=> Spring bean life cycle cfgs we need to configure life cycle method explicilty either using xml cfgs or using annotation cfgs..

life cycle methods in spring bean are

life cycle event

a) instantiation event

note:: The Spring bean init life cycle method will be called by IOC container automatically when the IOC container creates the Spring Bean class obj and completes the all kinds of Injections on that object

note:: The IOC container calls spring bean life cycle method, when the IOC container is about to destroy our Spring bean class obj

life cycle method

custom init method

(method with any name)

logics

=> To initialize left over properties of spring bean class that are not paritcipating in any kind of injections

=> To check wheather important spring bean properties injected with valid values or not

Initialization logics

b) Destruction event

custom destroy method

by

(method with any name)

=> To nullify spring bean properties assingning with 0 or 0.0 or null or false to spring bean properties

UnInitialization logics

=> To release non-java resources associated with spring beans like closing jdbc con .. closing

IO streams and etc..

The standard signature of custom init, custom destroy methods of spring bean life cycle is

public void <method name>(no params){

note: The spring bean life cycle methods are one time executing blocks

}

....

any name can be taken as the method name

We must follow this signature while desgning custom init life cycle method and custom destroy life cycle method for Spring bean

Bad becoz of

Different approaches of spring bean life cycle programming

a) Declarative approach (xml cfgs) (here the custom init and custom destory methods

xml configurations

Bad Becoz the

spring bean

becomes invasive

Best becoz

no need of xml cfgs

and spring bean

can be non-invasive

will be configured with xml cfgs) => Here life cycle method names are custom names

b) Programatic Approach (using interfaces implementation)

allows keep spring bean

as non-insive spring bean

afterProperitesSet() method (becomes init lifecycle method)

These are

pre-defined interfaces

destroy() method

(becomes destroy life cycle method)

=>Here life cycle method names are fixed name becoz they are taken from the implemented spring api interfaces.

Makes the spring bean as invasive

Spring Bean class should implement

(Exceptional case)

a) InitializingBean(1) b)DisposibleBean(1)

and @PreDeestroy)

Allows to keep

(Best)

c) Annotation cfgs Approach (using jdk supplied annotations like @PostConstruct

=> here the life cycle method names testom names having annotations

=>@PostConstruct to configure the Java method as the init life cycle

method

=>@PreDestroy to configure the java method as the destroy life cycle method

Is IOC container is creating spring bean class object using custom init method logics?

Ans) IOC container uses its own reflection api logics to create the spring bean class object.. So it is not using custom init method logics to create spring bean class objs

spirng beans as

non-invasive spring beans

Since the IOC Container calls init life cycle method

on the spring bean class obj, so we can say IOC container

Is IOC container is destroying sprng bean class object using the logics of custom destory method? Ans) No ... IOC container uses its own garbage collection logics to destroy our spring classes objs

is not using init life cycle method to create the spring bean class obj

(destroy() life cycle method executes just before the garbage Collection to execute some uninitialization logics w.r.t spring bean class obj destruction)

Q) Is it mandatory to configure/place the life cycle methods in spring bean class development?

Ans) optional, The IOC container looks to execute the life cycle methods of the spring bean

only when they are configured either using xml cfgs or annotations (best) or interfaces implementation

=> Though IOC container raises the life cycle events, it will not look execute the life cycle methods if no methods of the spring bean are configured as the life cycle methods

Spring Bean Life Cycle Management using the Annotation driven configurations

In this approach, we need to use

@PostConstruct ---- To cfg init life cycle method (Custom Method)

@PreDestroy

To cfg destroy life cycle method (Custom Method)

=>Initially, these two annotations are available in Jdk s/w (upto jdk1.8-Java8 version)

=> from Jdk 1.9 (java 9) it is given in a separate jar file, that is jakarta.annotation-api-<ver>.jar file

In pom.xml

<!-- https://mvnrepository.com/artifact/jakarta.annotation/jakarta.annotation-api -->

<dependency>

<groupId>jakarta.annotation</groupId>

<artifactId>jakarta.annotation-api</artifactId>

<version>2.1.1</version>

</dependency>

=> In the init life cycle method, we need to place initialization logic for left over properties that are not participating

in the injections.. and we can also place the validation logics checking wheather important properties are injected with correct values or not

method

=> In the destroy life cycle, we generally place, the nullification logics of the bean properties and logic to release non-java resources associated with spring bean class object (like DB connection)

>

IOCProj16-Spring BeanLifeCycle

src/main/java

com.nt.client

> SpringBeanLifeCycleTest.java

com.nt.commons

Info.properties

#com.nt.config

> AppConfig.java

com.nt.sbeans

> CheckingVotingElgibility.java

src/test/java

> JRE System Library [JavaSE-17]

Maven Dependencies

Dependencies in pom.xml

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context-support --> <dependency>

<groupId>org.springframework</groupId> <artifactId>spring-context-support</artifactId>

<version>6.1.2</version>

</dependency>

<!--https://mvnrepository.com/artifact/jakarta.annotation/jakartacannotation-api -->

<dependency>

<groupId>jakarta.annotation</groupId>

<artifactId>jakarta.annotation-api</artifactId>

For @PreDestroy and @PostConstruct

>

src

> target

pom.xml

<version>2.1.1</version>

</dependency>

Info.properties

voter.id=1001

voter.name=raja

voter.age=30

//CheckingVotingElgibility.java

package com.nt.sbeans;

import java.util.Date;

import javax.annotation.PostConstruct;

import javax.annotation.PreDestroy;

import org.springframework.beans.factory.annotation.Value; import org.springframework.context.annotation.PropertySource; import org.springframework.stereotype.Component;

@Component("voter")

@PropertySource("com/nt/commons/Info.properties") public class CheckingVotingElgibility {

@Value("${voter.id}")

private Integer id;

@Value("${voter.name}")

private String name;

@Value("${voter.age}")

private Integer age;

private Date verfiedOn;

public CheckingVotingElgibility() {

Dependency Hierarchy

apiguardian-api: 1.1.2 [test] spring-context-support: 6.1.12 [compile]

✓ spring-beans: 6.1.12 [compile] spring-core: 6.1.12 [compile]

✓ spring-context: 6.1.12 [compile]

spring-aop: 6.1.12 [compile]

spring-beans: 6.1.12 [compile]

System.out.println("CheckingVotingElgibility::0-param constructor");

spring-core: 6.1.12 [compile]

spring-beans : 6.1.12 [compile]

spring-core: 6.1.12 [compile]

spring-expression: 6.1.12 [compile]

spring-core: 6.1.12 [compile]

micrometer-observation: 1.12.9 [compile]

micrometer-commons: 1.12.9 [compile]

spring-core: 6.1.12 [compile]

spring-jcl: 6.1.12 [compile]

```
}
@PostConstruct // init life cycle method
public void myInit() {
System.out.println("CheckingVotingElgibility.myInit()");
// initialize the left over properties that had not participated in the injections
verfiedOn=new Date();
// validation logics
if(name==null || age<=0) {
throw new IllegalArgumentException("set corrent values to name,age properties");
}
}
//b.method
public String checkElgibility() {
System.out.println("Checking VotingElgibility.checkElgibility()");
if (age<18)
return "Mr./Miss/Mrs."+name+" ur not elgible for voting---> verified on::"+verfiedOn;
else
return "Mr./Miss/Mrs."+name+" u r elgible for voting---> verified on::"+verfiedOn;
}
@PreDestroy //destroy life cycle method
public void myDestroy() {
System.out.println("CheckingVoting Elgibility.myDestroy()");
//nullification of the bean propertie
name=null;
age=null;
verfiedOn=null;
id=null;
}
}
//AppConfig.java
package com.nt.config;
import org.springframework.context.annotation.ComponentScan; import
org.springframework.context.annotation.Configuration;
@Configuration
@ComponentScan (basePackages = "com.nt.sbeans") public class AppConfig {
}
//ClientApp
```

```
package com.nt.client;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.nt.config.AppConfig;
import com.nt.sbeans.CheckingVotingElgibility;
public class SpringBeanLifeCycleTest {
public static void main(String[] args) {
//create the IOC container
AnnotationConfigApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);
//get Target spring bean class obj ref
CheckingVotingElgibility voter=ctx.getBean("voter", CheckingVotingElgibility.class);
//invoke the b.methods
try {
String result=voter.checkElgibility();
System.out.println(result);
}
catch(Exception e) {
e.printStackTrace();
}
//close the container
ctx.close();
}
}
```

output

Problems Servers Terminal Data Source Explorer Properties Console

<terminated> SpringBeanLifeCycleTest [Java Application]
E:\Softwares\Eclipse\eclipse-jee-2023-09-R-win32-x86_64\eclipse\plugins\or

CheckingVotingElgibility::0-param constructor

CheckingVotingElgibility.myInit()

CheckingVotingElgibility.checkElgibility()

Mr./Miss/Mrs.raja u r elgible for voting---> verified on::Tue Jan 09 20:16:45 IST 2024
CheckingVotingElgibility.myDestroy()

Spring Bean life cycle diagram

Does not exist __(object is not created for spring bean)

Spring bean class Loading (static block of spring bean class)

Spring

For Bean life cycle

Instantiatiuon

Event

Spring Bean Instantiation

(contructor of spring bean class)

+ constructor Injection

Injections on Spring bean class obj (field Injection/ setter Injection/ arbitrary method Injection)

(all the cfg injections)

Assignment:

=========

custom

init life cycle method

(Custom init method execution)

Access the spring bena class obj ref

Spring bean class

for B.methods execution

obj is ready (for use)

destroy life cycle method

(Custom destroy method)

Life cycle

For Spring Bean Destruction Event

(For ctx.close()

Spring Bean class obj -DeInstantiation (Garbage Collection)

method call)

Spring Bean class Unloading

Design another application to check injected values validness before using them in the b.mehtod as the input values (Develope the Marriage Elgibility Application)