Container

==========

=>Container is a special software program that manages the the life cycle of the given component/resource from birth to death (Object creation to object destruction)

eg1:: Servlet container manages the servlet comp life cycle

eg2:: Jsp container manages the jsp comp life cycle

eg3:: EJB container manages the EJB comp life cycle

eg4:: Spring/Spring boot containers or IOC containers manage the Spring bean life cycle

IOC Inversion Of Control

=> Component means reusable class or file

eg: servlet comp, jsp comp, ejb comp, spring bean and etc..

Spring Bean

=> Spring Bean is java class whose object is created and managed by IOC container

=> we can configure pre-defined or user-defined or third party supplied

java class as the spring bean

=>we can not configure interface or abstract class as the spring bean becoz

we can not create object for interface and abstract class

=>Container is like an aquarium and the comps placed inside the

container are like fishes.

like fishes

Container (Aquirium) comp1 Comp2 (compB

Comp4 comps

IRE/IVM

Once we place comp inside the container that will take care of all life cycle activities of comp lie

==>Load java class

==> creating object

==> Initializing the object

==> managing the object

==> Destroying the object

Why the Spring containers are called IOC containers?

Normal Java App

=>programmer loads the java class

=>Programmer creates the object for java class

=> Programmer initializes the object

=> Programmer manages the object

=> Programmer destroys the object if needed

Spring App/Spring boot App

==========

=> IOC container loads the java class (spring bean)

Here all operations are taken care by programmers with support of JRE/JVM

=> IOC container creates the java class object (spring bean class obj) => IOC container initializes the java class object (spring bean class obj) => IOC container manages the java class object (spring bean class obj) => IOC container destroys the Java class object (spring bean class obj)

=> Web server (like Tomcat), Application server (like weblogic) gives built-in Servlet Container and jsp container to automate the process the of executing web applications(websites) and its web comps

=> Spring/Spring boot frameworks give two built-in IOC container to manage the life cycle of spring beans

a) BeanFactory IOC container (Basic)

b) ApplicationContext IOC container (Advanced) |----> Bean Factory IOC container++

Here most of the operations are taken care by spring/spring boot container which is reverse of regular process..So the the spring/spring boot container is called as IOC container (Inversion of Control)

ApplicationContext IOC container

Bean

Factory 10g container

=>IOC containers are not alternate for Servlet Container and Jsp container both are given for different purposes

=>servlet container, jsp container are useful to manage the life cycle of comps belonging to web applications and Distributed apps

=>IOC Containers are useful to manage the life cycle of spring Beans that are part of spring/springboot Apps development

Why the containers the not involved in standalone apps and why the containers are involved in the web applications and distributed apps?

Ans)

standalone apps are specific to one computer and will be operated by 1 user at a time, so manual execution of the App is sufficient and there no need of automation for executing the Apps

web applications and distributed Apps get huge no.of simultaneous from

from huge no.of requests from the clients, So the manual execution of comps belonging to web applications and distributed apps is not going to workout So we need special softwares or containers that automates the management and execution of the comps

eg: servlet container and jsp container of web server /App server eg:: IOC containers of spring/spring boot

=> All these java based containers directly or indirectly run on the top of JRE/JVM after all each java line (code) executes with the support of JRE/JVM

note:: JRE/JVM executes the container and container runs the java class code

Servlet container

servlet1 (servlet2

JRE/JVM

IOC

container

Jsp container

sp1 jsp2

JRE/JVM

EJB container

ejk1

ejbx

IRE/IVM

spring spring bean1 bean2 IRE/IVM

=>if u want to configure java class as spring bean, we need to give either xml

driven cfgs or annotation driven configurations(inputs and instructions)

to IOC container as the input values

=> In xml driven configurations, we use <bean> tag to configure java class as spring bean

WishMessageGenerator.java

package com.nt.sbeans;

public class Wish MessageGenerator{

}

=>any <filename>.xml can be taken as the spring bean cfg file . the recommended name

is applicationContext.xml

applicationContext.xml (spring bean configuration file)

<bean id="dt" class="java.util.Date"/>

<bean id="wmg" class="com.nt.sbeans.Wish MessageGenerator"/>

bean id

This bean id

fully qualified class name

IOC container

internally becomes obj name Wish MessageGenerator class obj(wmg)

jayantil.Date class obj(dt)

JRE/JVM

=> Providing additional info about the code to underlying container/server/

fran

/ JRE either using xml files or using annotations or using special

java code us called configuration

=> Xml based configurations are fading out, so prefer using annotation driven

or java code driven cfgs

Limitations of xml

==========

=> Xml (eXtensible Markup Language) is another language to learn

=> Xml file is separate file to develop i.e we need write additional information about the java code in a separate file

=> To read and process the xml files (docs) we need the heavy weight Xml API like SAX API, DOM API, JDOM API, DOM4J API and etc...

SAX: Simple API for Xml processing

DOM :: Document Object Model

JDOM :: Java Document Object Model DOM4J:: DOM for Java

To overcome these problems of Xml related to Code Configurations we need to use java Annotations

Syn: @<annotation -name>(param1=val1, param2=val2,....)

=> @Component, @Override, @SupressWarnings and etc..

Advantages with annotations based Code Configurations

=> Annotations java statements and they will be placed directly in java source code

=> since annotations are there with java code, improves the readability of the java code

=> Annotations are recognized and processed by compilers and JVM directly, So they give better performence

We can configure JAVA class as the spring bean using 3 approaches

a) Using xml driven cfgs (use <bean> tag in xml file as shown above)

b) using Annotation driven cfgs (use @Component annotation)

c) using Java Code confiturations (use @Bean methods in @Configuration class)

b) using Annotation driven cfgs (use @Component annotation)

=>@Component is class level annotation this very useful to configure user-defined java class

as the spring bean

|---> bean id

```
@Component("wmg")
public class Wish MessageGenerator{
}
```

IOC container

Wish MessageGenerator class (obj(wmx)

JRE/JVM

note: if the java class is pre-defined class then we can not use @Component annotation becoz we can not edit the source code of pre-defined class having

the annotation. To overcome this problem, we need to use java code configuration approache to make java classes as the spring beans (@Bean methods of @Configuration class)

c) using Java Code confiturations (use @Bean methods in @Configuration class)

=> The java class using which we provide inputs and instructions to IOC container is called @Configuration class (class with @Configuration annotation).. This class alternate to spring bean cfg file (xml file)

=> To cfg any pre-defined java class as spring bean we can place @Bean method inside

the @Confiuration class

=> The IOC container automatically calls @Bean methods of the @Configuration class and makes that

method returned objects as the spring beans

AppConfig.java (Configuration class)

@Configuration

public class AppConfig{

|--> bean id

@Bean(name="ldt")

public LocalDateTime createLDT(){ return LocalDateTime.now();

}

}//class

static factory method

note:: IOC container automatically calls @Bean methods and makes the method returned objects as spring beans having the given bean ids

IOC container

LocalDateTime class obj(ldt)

sys date and time

JRE/JVM

Conclusion::

=> To configure pre-defined or user-defined java classes as spring beans use <bean>tags

in spring bean cfg file (In xml driven cfgs) outdated approach - Legacy approach

=> In Annotation driven /JAva Code configuration approach

=> use @Component annotation to configure java class as the spring bean

=> use @Bean method in @Configuration class to configure pre-defined java class

as the spring bean

user-defined

Q) Can use @Bean method in @Configuration class to make the java class as

the spring bean?

Ans) Possible, but not recomanded

WishMessageGenerator.java

public class Wish MessageGenerator{

AppConfig.java

@Configuration public class AppConfig{

=> we can place multiple @Bean methods (0 or more) in one @Configuration class

=> we can use all the 3 approaches in one application to make

the java classes as the spring beans

@Bean(name="wing bean id

public WishMessageGenerator createWMG(){ return new Wish MessageGenerator();

}

Here the programmer is creating

the object manually.. this is unnecessary for user-defined class

Spring Core module

==========

=> It is base module for other modules

=> Using this module alone, we can develop at max standalone java apps

=> if this module is in combination with other modules then we can develop

web applications, distributed apps, enterprise apps and etc..

=> This module basically gives two IOC containers /Spring Containers

a) BeanFactory Container (basic)

b) ApplicationContext Container (advanced)

=> The IOC containers of Spring framework/Spring boot frameworks are basically given

to perform two operations

a) Spring Bean Life cycle management

b) Dependency Management

a) Spring Bean Life cycle management

=> The process making IOC container taking care of all the life cycle activities

of spring bean from birth to death is called Life cycle managment

=>The life cycle activities of spring bean are

a) Loading java class (spring bean)

b) creating object

c) Managing the object

d) Destroying the object

b) Dependency Management

=>The process of making the IOC container arranging dependent spring bean

class obj to target spring bean class obj is called Dependency MAmagement

=>The spring bean that uses the services of other spring bean is called target spring bean =>The spring bean who services are being used is called Dependent spring bean

Dependent Spring bean

Engine

Target spring bean

Vehicle

Student

Flipkart

DTDC

Cricketer

Bat

==> Vehicle uses the Engine Course Material ==> Student needs course material for preparation =>Flipkart uses DTDC courier service for delivery

==> Cricketer uses bat for batting

Core java App dealing Flikart and DTDC classes

==========

==========

=> Programmer makes the JVM/JRE to load both Flipkart and DTDC classes

=> Programmer makes the JVM/JRE create objs for both Flipkart and DTDC classes

=> Programmer assigns DTDC class obj to Flipkart class object

=> Programmer uses both objs as neeed to call the b.methods

obis

=> Programmer keeps both ady for Garbage Collection by nullifying them

Here Programmer over burdended

Above App using Spring

==========

Life cycle

mgmt

=>IOC container loads both Flipkart and DTDC classes =>IOC container creates the objects for both Flipkart and DTDC classes

Depedency momOC container assigns DTDC class object to Flipkart class object Life cycle => Programmer collects both objs from IOC container to call the b.methods

=>IOC container assigns manages the both objects

mgmg => IOC container destroys both objs

Spring programming makes the programmer life

easy