@Scope("prototype")

=> The IOC container creates separate object for spring bean class for every ctx.getBean(-) method call and that spring class object will not be placed in the internal cache of IOC container

take

=> if the Spring Bean class obj state is changing time to time.. then it is recomanded to the scope

as the "prototype" scope.. So that for every data change one new object will be created for the spring bean

Example

=>Object state means object data (data that is stored in the memervariables)

=> Object behavior means object logics (the logics that are placed in the methods)

@Lazy(true) is applicable only for the singleton scope spring beans to disable the pre/eager instantiation on singleton scope spring bean.. @Lazy(true) does not give effect on other scope spring beans becoz they are always enable with lazy instantiation

```
@Component("wmg")
@Scope("prototype")
public class WishMessageGenerator {
@Autowired
@Qualifier("dt")
private LocalDateTime ldt; //HAS-A property
.....
```

In Client App

==========

```
WishMessageGenerator generator=ctx.getBean("wmg",Wish MessageGenerator.class);
WishMessageGenerator generator1=ctx.getBean("wmg",WishMessageGenerator.class);
WishMessageGenerator generator2=ctx.getBean("wmg",WishMessageGenerator.class);
System.out.println(generator.hashCode()+" "+generator1.hashCode()+" "+generator2.hashCode()); //gives the different hashCodes System.out.println("generator== generator1?"+(generator==generator1)); //gives false System.out.println("generator1== generator2?"+(generator1==generator2)); //gives false
}
```

=> if the spring bean scope is "prototype" the spring bean class obj ref will not be placed in the internal cache of the IOC container, So ctx.getBean(-) does not find prototype scope spring bean class obj ref in the internal cache of the IOC container.. hence it creates new object for every ctx.getBean(-) method call

What happens if we configure the real singleton java class as the prototype scope spring bean?

Ans1) if IOC container is creating singleton Java class obj by accessing private constructor then the prototype scope will be continued i.e the IOC container creates the multiple objs for multiple ctx.getBean(-,-) method calls (Singleton Java class will be broken)

Code

=====

```
@Component("prn1") @Scope("prototype")
public class Printer {
private static Printer INSTANCE;
```

```
//private constructor
private Printer() {
}
System.out.println("Printer:: O-param constructor");
// static factory method having singleton logic
public static Printer getInstance() {
if(INSTANCE==null)
INSTANCE=new Printer();
return INSTANCE;
}
//b.method
public void printMessage(String msg) {
System.out.println(msg);
}
}
```

In Client App

```
System.out.println("--
-");
Printer p1=ctx.getBean("prn1",Printer.class); Printer p2=ctx.getBean("prn1", Printer.class);
System.out.println(p1.hashCode()+" "+p2.hashCode()); // gives two different hashCodes
```

note:: generally the private constructor of java class is not visible to outside of the java class, but we can make it accessible by using the support of reflection api.. IOC container uses reflection api internally to access the private constructor of the spring bean class as part of spring bean instantiation

(The real singleton java class behaviour is broken)

[Reason the IOC container is not doing static factory method based instantiation for the singleton java class.. It is instantiating directly by using 0-param private constructor] by accessing the private constructor using refletion api

Code

Ans2) if we enable static factory based instantiation for the prototype scope spring bean who class is singleton java class, then the singleton behavior will be continued.

In @Configuration class

```
@Bean(name="prn1")
@Scope("prototype")
public Printer createPrinter1() {
}
System.out.println("AppConfig.createPrinter1()");
return Printer.getInstance();
```

Enabled the static factory method

bean instantiation for singleton java class

note: Since ur configuring Printer class as the spring bean class using @Bean method.. So commmnet @Component Annotation in the Printer.java class

In Client App

```
System.out.println("
-");
Printer p1=ctx.getBean("prn1",Printer.class); Printer p2 ctx.getBean("prn1", Printer.class);
System.out.println(p1.hashCode()+" "+p2.hashCode();
```

gives same hashCodes

becoz the singleton java class

is gettting instantiated through static factory method where the singleton logic is avaiable

Can u we keep private constructors in the spring bean java class? Ans) Yes, we can keep becoz IOC cointainer can access the private constructor while creating the object (Spring bean class obj) (internally the reflection api will be used)

Can we take the spring bean class as the private java class? Ans) No.. as we can not take outer classes the private classes ..we can not Inner think about taking spring bean classes as the private classes

if we do not provide bean id to the spring bean class then what happens?

effect

How can we give singleton scope for the spring bean though its scope is prototype scope?

Ans) if the prototype spring bean is dependent to target singleton scope spring bean some how dependent spring bean behaves like singleton scope spring bean indirectly a) Prototype scope spring bean will also be pre-instantiated along with singleton scope spring bean

b) if ur always planning access dependent spring bean object through target spring bean object then only one dependent spring bean class obj will created to inject to target singleton scope spring bean though dependent spring bean scope is "prototype".

(c) if we configure real singleton java class as the prototype scope spring bean by enabling static factory method bean instantiation

Ans) The IOC container generates default bean ids to the Spring bean class based on the approach that we have used to configure the spring bean class

id

note1:: if the java class is cfg as the spring bean using @Component then the default bean is class name having first letter in lower class

```
@Component
@Scope("prototype")
public class Wish MessageGenerator {
@Autowired
@Qualifier("dt1")
private LocalDateTime ldt; //HAS-A property
}
....
```

In @Configuration class

The default bean id is :: wishMessageGenerator (class name)

@Bean methods of @Configuration class the method name will be taken as the default bean id

```
@Bean
@Scope("prototype")
public LocalDateTime createLDT() {
System.out.println("AppConfig.createLDT()"); return LocalDateTime.now();
}
```

The default bean id is :: createLDT (nothing but method name)

IOC container maintains lots of information about the spring beans that are being maintained

to get that information we can call various methods IOC container object

In Client App

========

```
System.out.println(" Spring Bean Definitations count ::"+ctx.getBeanDefinitionCount()); --->gives 8 count
System.out.println("Spring bean ids"+Arrays.toString(ctx.getBeanDefinitionNames()));
```

Spring bean ids[org.springframework.context.annotation.internalConfiguration Annotation Processor, org.springframework.context.annotation.internalAutowiredAnnotationProcessor, org.springframework.context.event.internalEventListenerProcessor, org.springframework.context.event.internalEventListenerFactory, appConfig, wishMessageGenerator, createLDT, prn]

In Xml driven cfgs, the default bean id is fully qualified class name>#<n>

In @Component based spring bean cfg, the default bean id is :: classname having first letter in lower case

In @Bean method based spring bean cfg, the default bean id is :: method name

What is the difference b/w IOC and dependency Injection?

Ans) IOC is the software specification having set of rules and guidelines to manage the dependency among the comps (like target and dependent spring beans).. "Dependency Lookup","Dependency Injection" are the two implementation models that are given baseanoC rules and guidelines

(DI)

IOC ---> Theory

"DL" and DI --> Praticals based on the Theory (IOC theory)

IOC-> Like Planning

"DL" and "DI" are the real implementations of the planning (IOC)

What is the latest version of spring f/w?

ans) 6.2.3 (latest) 7.0.0(Beta)

What is the latest version of the spring boot f/w? Ans) 3.4.3

Boot

=>All new projects in java are happening in spring 6.x and spring 3.2.x versions =>In maintainance projects java, the spring versions 4.x or 5.x Similarly the spring boot versions are 2.x

when should i take the spring bean scope as the prototype scope?

Ans) if the state of the spring bean is changing time to time (like storing form data to

the spring bean class obj) then we need to take that spring bean as prototype scope spring bean

@Scope("request")

=> This scope can be used only for spring beans of web applicaitons/distributed apps (spring mvc or spring rest apps) spring mvc for web application, spring rest for Restful webserivces (distributed apps)

=> IOC contianer makes the seervlet container to keep the spring bean class object in request scope

i.e request attribute value which is specific to one request where beanid will be used as request attribute name and spring bean class object will be used request attribute value.

@Component ("wmg")

@Scope("request")

public class WishMessageGenerator{

bean id as the attribute name

bean class obj ' as the attribute value

req.setAttribite("wmg", WishMessageGenerator class object ref);

I

object

=>creates the spring bean class on 1 per each request basis

web services based

distributed App = web application ++

=>singleton,prototype scopes can be used in both standalone apps and web applications =>request, session, application, websocket scopes can be used only in web applications

note:: if want to store the form data /front end UI of a web application or distributed app in the spring bean class obj which changes request to request then prefer keeping spring bean class obj in the request scope

(eg:: student registration form data, credit card/debit payment form data)

}

=>creates and keeps the Spring bean class obj inside the request obj which is 1 per request basis

@Scope("session")

=>This scope can be used only in web applications and restfull apps

=> IOC container makes the ServletContainer to keep spring bean class object in HttpSession object

as session attribute value nothing but 1 per browser s/w of each client machine basis

browser to software App interaction ---- website or web application

eg:: browser to flipkart.com

@Component ("wmg")

@Scope("session")

public class WishMessageGenerator{

}

@Scope("application")

software application interaction to another software app interaction eg: flipkart.com to Paytm app eg:: flipkart.com to razor pay app

HttpSession ses=req.getSession(); --creates the Sesison obj in server

bean id

on 1 browser s/w of client machine basis ses.setAttribute("wmg", WishMessageGenerator class obj ref); spring bean class obj ref

=> Keeps the spring bean class obj ref in the Session obj as the session attribute value i.e spring bean class obj is specific to one browser s/w of client machine eg:: Storing Login form data (username, password) to spring bean class obj needs put spring bean in Session object

=>This scope can be used only in web applications and restfull apps (Distributed apps)

=> IOC container makes the ServletContainer to keep spring bean class object in ServletContext obj

as ServletContext attribute value or application attribute value which is 1 per entire web application irrespective no.of reuques that are coming and irrespective from which browser s/ws those reuqests are coming

as

=> keeps spring bean class obj application attribute having bean id as the attribute name and spring bean class obj ref as attribute value.

@Component("wmg")

@Scope("applicatioh)

public class WishMessageGenerator{

}

Distributed App (Restful app)

Java Popular scopes in web application

request scope ==> specific to each request session scope ==> specific to each browser s/w of a client machine application scope ==>specific to each web application

=> if the spring bean state/data is empty/fixed/sharable then go for singleton scope => if the spring bean state/data is varying state then go for prototype scope

=> if the spring bean state/data is form data then go for request scope

=> if the spring bean state/data is Login username,password then go for session scope => if the spring bean state/data is common data for the entire app then go for application scope

ServletContext sc=getServletContext();

note: if u want to store requests count, users count

In a web application,

=>request object is specific to each request (request scope) => session object is specific to each browser s/w (session scope) => application object is specific to an web application (application scope)

sc.addAttribute("wmg", WishMessageGenerator class obj ref) request scope ---> 1 per each request session scope ---> 1 per each browser s/w of client machine application scope ---> 1 per web application

of a web application in spring bean class obj then better to put spring bean class obj in application scope

What is the diference b/w singleton scope and application scope in the web application? Ans) singleton scope means 1 obj for spring bean per bean id with respect to each JOC container application scope means 1 obj for spring bean per bean id with respect to entire web application

=>if we create "n" IOC containers in one web application then "n" objects will be created

for singleton scope spring bean and only 1 object will be created for application scope spring bean

=> Singleton scope spring bean class obj is kept internal cache of IOC container to make it reusable with in the IOC container where as the application scope spring bean class obj is kept in Servletcontext object

to make it reusable for entire web application.

=>"Singleton" scope can be used in both standalone Apps, web applications

where as application scope can be used only web applications

@Scope("websocket")

=>singleton scope spring bean is specific to each IOC container =>application scope spring bean is specific to each web application

=>protocol http is half duplex protocol i.e it supports two way communication but all operations takes place only_client(browser) intiaates the request..

client/browser

web, application

request1

in web server

response1

Server gives data to client only when the

request2

response2

send

client sends the request to server

(half duplex communication)

=> protocol "websocket" (ws) is full duplex protocol i.e it supports two way communication

and server can data to client

Client (browser)

with out getting request from the client..

web application

in web server

web application

IOC cotnainer1 IOC cotnainer2 IOC cotnaiher3

(singleton) (singleton)

Date class ob 2 class obj3

Date

Date

class ob 1

singleton),

"ServletContext obj

Date class obj (application scope

note::

request1

response1

response2

response 3

full duplex communication

i.e the server can give multiple response to the client once it receives the initial request i.e certain responses can be given with out getting any requests

Based on the connection established b/w client and server for the

1st request,the server can send multiple responses to client with out getting any request from client.

(eg: suitable for chatting apps like gmail chat, fb chat and etc..)

In spring programming, scope="websocket" or @Scope("websocket") makes the

IOC container to keep the spring bean class obj in the websocket session that supports

Full duplex communication.

@Component("wmg")

@Scope("websocket")

public class Wish MessageGenerator{

...

bean id

WebSocketSession ses=req.getWebSocketSession(); ses.setAttribute("wmg", WishMessageGenerator class obj ref); spring bean class obj ref

=>This scope can be used only in web applications.. which are running with protocol "websocket" (ws)

(Extension protocol http)

What is the pre-instantiation and eager instantiation of the spring bean?

Ans) if the IOC container is creating any spring bean class obj during the startup of the IOC container that is called pre-instantiation or eager instantiation of Spring bean

=> Only singleton scope spring beans support pre-instantiation of spring beans i.e the other scope spring beans

do not support pre-instantiation of spring beans

Q) Why there is pre-instantiation only for singleton scope spring beans?

ans)

The singleton scope spring beans that are having pre-instantiation during the IOC container startup

will be maintained in the internal cache of IOC container having reusability across the multiple ctx.getBean(-) calls

and across the multiple injections

.. so the singleton scope spring beans are having the pre-instantiation becoz the

the pre-instantiated objects will never be wasted.. (More over they will be resused

by keeping the internal cache)

=>The other scope spring bean classes objs will not be placed in the internal cache of IOC container..i.e

there is no reusability

for those objs becoz multiple objs will be created in different sceanarios. So no pre-instantiation is given for other than singleton scope spring beans..

note:: other than singleton spring beans support lazy instantiation

Q) if the target singleton scope spring bean is having prototype scope spring bean as the dependent, can u tell me the prototype scope

spring bean participates in the pre-instantiation or not?

Ans) The scope of prototype scope dependent spring bean will not be changed to singleton scope.. but

the prototype scope spring bean participates in instantiation along with the pre-instantiation of singleton scope spring bean to complete injection process on it

Q) How to disable pre-instantiation on singleton scope spring bean ?(nothing but enabling Lazy instationation)

Ans) @Lazy(true) along with @Component or @Bean annotations

```
@Component("wmg")
//@Component
@Scope("singleton") @Lazy(true)
public class WishMessageGenerator {
www.
....
...
....
}
```

In @Configuration

note:: we can not enable pre-instantiation

on other than singleton scope spring beans

```
@Bean
@Scope("singleton")
@Lazy(true)
public LocalDateTime createLDT() {
System.out.println("AppConfig.createLDT()"); return LocalDateTime.now();
}
```

Q) i have 10 spring beans, how can we make only 5 spring beans participating in the pre-instantiation?

Ans1) Take all the 10 spring beans as the singleton scope spring beans but enable @Lazy(true) on 5 spring beans, but make sure that these 5 spring beans are not dependent to other 5 singleton scope spring beans

(or)

Ans2) Take 5 spring beans as singleton scope spring beans and other spring beans as the prototype spring beans

but make sure that the prototype scope spring bean are not dependent to singleton scope spring beans

StereoType Annotations

=> The spring supplied multiple annotations having similar functionality are called Stereotype annotations i.e all these annotations perform similar operations with minor changes

note:: Boys dressing is called stereo type dressing, girls dressing is not called

stereo type dresssing

a) @Component ----> makes the java class as the spring bean (With TxMgmt support)

b) @Service ----> makes the java class as the spring bean cum service class

=> The java class that contains the b.logic in methods is called service

=> b.logic means dealing with calculations, analyzations, validations, sorting, filtering and etc,,

=> The java class that contains the persistence logic (CURD operations logics) is called DAO class /Repository clas DAO class ---> Data Access Object class

c) @Repository ---> makes the java class as the spring bean cum DAO class/ Repository class/Persistence class (Exception Translation)

d) @Controller ---> makes the java class as the spring bean cum web controller class having the ability to take the requests from the clients

e) @Configuration ----> makes the java class as the spring bean cum f) @ControllerAdvice -------> makes the java class as spring bean Configuration class

and etc....

cum advice class to handle the raised exceptions and to display exception related messages as the user-defined guiding messages

=> The java class that controls all the activities of the applications is called Controller class

=> The java class that handles the exceptions to display the generated exception related messages as the non-technical messages is called Controller Advice class

@Service

@ Component

@Repository

@Controller

=>While developing the spring/spring boot based layered Apps, we take the support of these stereo type annotations

@Controller

@Configuration

Advice

In Real projects, we develop the project as the Layered App i.e we keep different logics in different java classes and we make

them participating in the communication.. In this situation,we take the support of these special stereo type annotation to

make the java classes as the spriing beans cum special classes of layered app.

Client app

(@Controller)

controller class

(presentation logics/

(monitoring

UI Logics)

logics)

(@Service)

(@Repository)

->service class

-> DAO class

DB s/w

(b.logic)

(persistence logic

-- jdbc code)

note:: In any spring project of 100% code configuration we pass inputs to IOC container

using @Configuration class

@Service/@Repository/@Controller/@Cofiguration = @Component++

(These annotations make the java classes as the spring beans

cum special classes)

In Layered Apps, we take the classes of the different Layers as spring beans using different stero type anntoations,So that the injections can be one those spring beans using @Autowired =>Inject DAO class obj *Service class obj =>Inject Service class obj to Controller calss obj > Client app can get controller obj using ctx.getBean(-,-) method

properties file

==========

=>The text file that maintains the entries in the form of key-value pairs is called properties file

=>This file is also called ResourceBundle file

=> This file can have any extension, but the recomanded extension is .properties

Info.properties

per.id=101 -per.name=raja

per.addrs=hyd

per.mobileno=988989998

note:: we can write comments in properties file using # symbol

keys

values

file

note:: The properties file that is configured on top of one spring bean class using @PropertySource annotation can be used in all the spring bean' classes of the project

=>To configure the properties with the spring App using @PropertySource Annotation

@PropertySource(value="<name and location of the properties file>")

On the top of

spring bean class

=> we can use @Value Annotation to read the values from the properties file and to inject them to spring bean

Properties

gives raja

In spring bean class

gives hyd

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

private String pname;

@Value("${per.addrs}")

private String paddrs;

${<key>} ---> place holder -- represents

the value that is going to come to Spring bean property

from the properties file

=>@Value is the multipurpose annotation, which can be used for different operations

a) To Inject simple values to primitive/String bean properties directly

@Value("raja")

eg:: private String pname;

Hard coded value injection

b) To inject the values collected from the properties file to primitive/String spring bean properties

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

eg: private String pname;

per.name is the key in the properties file

c) To inject the system property values to primtive/String bean properties

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

private String os_name;

@Value("${os.ver}")

private String os_ver;

os.name, os.ver and etc.. are the fixed system properties

d) To inject env.. variable values to primitive/String bean properties

@Value("${Path}") private Stirng pathData;

Path, ClassPath, JAVA_HOME and etc.. are called env.. variables

The key in the ${...} is called Place holder that represents name of the value that going to be injected to spring bean properties

What is difference b/w @Value and @Autowired annotations?

Ans) To Inject one spring bean class obj to another spring bean class obj's HAS-A property take the support

of

@Autowired (For Injecting spring bean)

In target spring bean

@Autowired

private IEngine engg; //HAS-A property

(Spring bean to Spring bean Injection)

To inject simple values collected from different places (like properites file,system properies and etc..)

to the primitive /String spring bean properties take the support of @Value annotation

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

eg: private String pname;

Injecting simple values to SpringBean properties