Spring Bean scopes

We can make the IOC container to keep spring bean class obj in different scopes

6.x

The spring bean scopes in 5.x version are

=>singleton (default)

=>prototype

can be used in standalone apps

and also in web applications

webservices app (restful app) = web application++

In standalone apps we can keep the data in the following scopes

a) block scope b) method/Local scope c)instance scope d) class scope e)Thread scope

In web applications of java, we can keep data in the following scopes

a) request scope b)session scope c) page scope d) application scope

=>request

=>session

=> application

=>websocket .

can be used only in web applications & Restful applications (Distributed Apps)

"scope" attribute of <bean> tag in xml driven cfgs (not so important) @Scope annotation in annotation driven cfgs required here)

--

=>To specify the spring bean scope we use =>To specify the spring bean scope we use =>@Scope is class level, method level annotation i.e it can be applied on the top of the class and on the top of the method.

=> Singleton class is no way related to singleton

What is singleton java class?

scope of spring bean note:: We generally use @Scope annotation along with @Component annotation and along with @Bean annotation

Ans) The java class that allows us to create only One object in any situation is called Singleton java class

as

Single == one ton == object

note:: Instead of creating multiple objects for java class either with no data or with fixed data or with shrable data where memory and CPU time will be wasted.. it is recomanded to create single obj and use it for multiple times to use the Memory and CPU time effectively.

note: Though java class allows us to create multiple objs, if we are happy with creating single object for that class.. then that java class is not called singleton java class.

eg:: For Servlet comp class, the Seervletcontainer creates single object.. though Servlet comp class allows to have multiple objs.. So we can not say Servlet comp class is singleton java class.

eg: Though programmer wants to create multiple objs, the singleton Java class allows us create only one object my Buation

pre-defined singleton java classes are

java.lang.Runtime, java.awt.Destkop, org.apache.log4j.Logger and etc..

To develop user-defined java class as the singleton Java class (minimum standards)

open

close all the doors of creating the Java class obj being from the outside of the class and only one door (static factory method) to create the obj by checking wheather object is already created or not (singleton logic)

(a) keep the private constructor(s)

(b) take static reference variable of

current class type

(c) static factory method having

singleton logic

IOCProj07-SingletonClass-CoreJava

^

src/main/java

com.nt.ston

› D Printer.java

com.nt.test

> SingletonTest.java

src/test/java

1 //Printer.java

2 package com.nt.ston;

3 public class Printer {

to

single ton I

one object

note::: The class that allows us to create only one object in any situation is called single ass

=>"singleton" scope in spring bean makes the IOC container to create only one for spring bean class through spring bean can have multiple objs

=> Singleton class allows the programmers to create only one object in any given situation

=>While designing any java class its better to take member variables(data/state) as private and methods/operations as public this makes us to operate data/state only through methods (not directly)

=> Take class as singleton class in the following situations

(a) if the class is designed with out state

(b) if the class is designed with read only member variables

(c) if the class is designed with sharable/common data member variables

System.out.println("Printer:: O-param constructor");

4

private static Printer INSTANCE;

5

>

6 //private constructor

>

JRE System Library [JavaSE-1.7]

7

private Printer() {

>

<

>

Maven Dependencies

src

target Mpom.xml

Be

9

}

D

1

2

3-

45 6IN

// 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);

9

3 a 8

B

De

1

}

2

}

L //ClientApp

2 package com.nt.test;

3

↓ import com.nt.ston.Printer;

5

5 public class SingletonTest {

System.out.println(p1.hashCode()+" "+p2.hashCode());

7

30

public static void main(String[] args) {

// get Printer class object

)

Printer p1=Printer.getInstance();

L

Printer p2=Printer.getInstance();

2

3

+

5

5

7

} }

}

> }

L

System.out.println("p1==p2?"+(p1==p2));

=>The singleton class that is given here is singleton class with minimum standards.. To make this class as more perfect singleton class take we need to additional logics

=>Make the code as Reflection API proof => Give protection from multi threaded env => Give protection from Deserialization => Give protection from cloning

[Watch the vedios kept in google drive for all these addtions]

Problems Servers Terminal Data Source Explorer Proper <terminated> SingletonTest [Java Application] E:\Software\Eclipse\ecli Printer:: 0-param constructor

1365202186 1365202186

p1==p2?true

=> one object can be pointed by multiple reference variables

=> Generally we take static or final static variables in UPPERCASE letters for differentiation

=> "==" operator is given Feferences comparision like to check wheather two ref variables are pointing to same object or not

=> "equals(-)" method is given for content comparision like to check wheather two objs of a java class are having same content or not

=> When ever class is loaded, the static variables will also be

loaded and will be initialized with default values or initial vlaues

In

Spring 5.x/6.x

we have 6 scopes of spring beans

1. singleton (default)

2. Prototype

3. request

4. session

5. application

6. websocket

=> To specify the scope use_@Scope annotation in annotation driven cfgs along with @Component and @Bean annotations

@Scope("singleton")

=> It is the default scope for spring bean if no scope is specified

=> In this scope, the IOC container creates single object for the spring bean class

with respect to Bean id and spring bean class and keeps that object in

the internal cache of the IOC container to reuse the object across the multiple calls given

to ctx.getBean(-) method having the bean id

@Component("wmg")

@Scope("singleton")

public class WishMessageGenerator{

and across the multiple Injections

@Configuration

@ComponentScan (base Packages=".....")

public class AppConfig{

@Bean(name="dt")

@Scope("singleton")

public Date createDate(){

}

return new Date();

@Bean(name="dt1")

internal cache of IOC container

@Scope("singleton")

public Date createDate1(){

return new Date();

}

wmg

WishMessageGenerator obj ref

0

Only singleton scope spring beans participate in

the pre/earger/early instantiation and they also will be

dt

Date class obj ref

1

kept in the internal cache of IOC container for reusability

dt1

Date class obj ref

2

....

3

bean ids (keys)

spring bean class objs

(values)

//WishMessageGenerator.java (target class)

package com.nt.sbeans;

import java.time.LocalDateTime;

we can get spring beans count and spring bean ids from the IOC container as shown below

In Client app

==========

System.out.println("all bean ids count::" +ctx.getBeanDefinitionCount()); System.out.println("all bean ids::" +Arrays.toString(ctx.getBeanDefinitionNames()));

@Component("wmg")

@Scope("singleton")

public class Wish MessageGenerator {

@Autowired @Qualifier("dt")

private LocalDateTime ldt; //HAS-A property

public WishMessageGenerator() {

output

=====

all bean ids count::8

all bean ids::

[org.springframework.context.annotation.internalConfigurationAnnotation Processor,
org.springframework.context.annotation.internalAutowiredAnnotationProcessor,

org.springframework.context.event.internalEventListenerProcessor,

org.springframework.context.event.internalEventListenerFactory, appConfig, wmg, ltime, ldate]

bean ids of container generated spring beans

bean ids of user-defined

spring beans

System.out.println("WishMessageGenerator:: O-param constructor");

}

....

}

70000

AppConfig.java

=============

0 @Configuration

1 @ComponentScan (basePackages = "com.nt.sbeans")

2 public class AppConfig {

3

4 public AppConfig() {

5 System.out.println("AppConfig:: O-param constructor");

6

}

7

8R

@Bean(name="dt")

9

@Scope("singleton")

TO

0

1

2

}

4-

3 4 5 6 7 8 9

public LocalDateTime createLDT() {

System.out.println("AppConfig.createLDT()");

return LocalDateTime.now();

@Bean(name="dt1") @Scope("singleton")

public LocalDateTime createLDT1() { System.out.println("AppConfig.createLDT1()");

return LocalDateTime.now();

}

0

1}

public class DependencyInjection Test {

public static void main(String[] args) {

//create the IOC container

AnnotationConfigApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);

// get target spring bean class obj

WishMessageGenerator generator=ctx.getBean("wmg",WishMessageGenerator.class);

WishMessageGenerator generator1=ctx.getBean("wmg",WishMessageGenerator.class);

WishMessageGenerator generator2=ctx.getBean("wmg",WishMessageGenerator.class);

System.out.println(generator.hashCode()+" "+generator1.hashCode()+" "+generator2.hashCode());
System.out.println("generator== generator1?"+(generator=-generator1)); //true
System.out.println("generator1== generator2?"+(generator1==generator2)); //true

Gives the same

hashcode

System.out.println("-

-");

LocalDateTime ldt1=ctx.getBean("dt",LocalDateTime.class); LocalDateTime
ldt2=ctx.getBean("dt",LocalDateTime.class);

System.out.println(ldt1.hashCode()+" "+ldt2.hashCode()); // gives the same hashcode

System.out.println("-

-");

LocalDateTime ldt3=ctx.getBean("dt1",LocalDateTime.class); LocalDateTime
ldt4-ctx.getBean("dt1",LocalDateTime.class);

System.out.println(ldt3.hashCode()+" "+ldt4.hashCode()); gives the same hashCode

When should i take spring bean class as singleton scope spring bean?

Ans) if spring bean class is not having state/data (no properties declaration)

if spring bean class is having only fixed state/data (only final properties declaration)

if spring bean class is having only sharable state/data (the state that can be shared across the multiple classes like class hold Countries info)

Does the "singleton" scope makes the spring bean class as the singleton java class?

Ans) No, our spring bean class does not become singleton java class.. But it makes the IOC container to create single object for Spring bean class w.r.t bean id note:: with out making the java class as the singleton java class, we can make the spring bean class just having object through singleton scope of the

only one

=>Singleton java class allows us to create single obj becoz of the logics placed inside the class

=>For singleton scope spring bean. the IOC container creates single obj though spring bean class is normal class and allows to create becoz of IOC container level restrictions

IOC container

What happens if we make the real singleton Java class as the singleton scope spring bean?

(not possible to enable in annotation driven cfgs)

Ans) if we configure same class as the multiple spring beans with different bean ids by enabling constructor based object creation then the singleton pattern can be broken.. if we enable factory method based instatiation then the singleton pattern can not be broken

Solution code using factory method based bean instantation in @Bean methods

======

======

```
//Printer.java package com.nt.ston;
import org.springframework.stereotype.Component;
//@Component("prn")
public class Printer {
private static Printer INSTANCE;
//private constructor
private Printer() {
System.out.println("Printer:: O-param constructor");
}
```

if we configure normal java class as multiple spring beans with different bean ids having singleton scope then the IOC container creates multiple objects for spring bean class w.r.t multiple beans ids

In AppConfig.java

======

```
@Bean(name="dt1")
@Scope("singleton")
public Date createDate1() {
System.out.println("AppConfig.createDate()1");
return new Date();
```

@Bean(name="dt2")

@Scope("singleton")

public Date createDate2() {

System.out.println("AppConfig.createDate()2");

return new Date();

obj1 for

// static factory method having singleton logic

public static Printer getInstance() {

if(INSTANCE==null)

INSTANCE=new Printer();

return INSTANCE;

Date class

having name

"dt1"

}

obj2 for

}

Date class

having nam

//b.method

"dt2"

}

public void printMessage(String msg) { System.out.println(msg);

}

}

In AppConfig.java

@Bean(name="prn1")

public Printer createPrinter1() {

}

return Printer.getInstance();

@Bean(name="prn2") public Printer createPrinter2() {

return Printer.getInstance();

=>Default scope is singleton

=>here same singleton java class is cfg

as spring beans having two different bean ids

with the default scope "singleton"

}

Client App

==========

// get spring bean

Printer p1=ctx.getBean("prn1", Printer.class);

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

System.out.println("

// get spring bean

-");

Printer p11=ctx.getBean("prn2", Printer.class);

Printer p22=ctx.getBean("prn2",Printer.class);

gives the same hashcode

System.out.println(p11.hashCode()+" "+p22.hashCode());

note: the singleton scope of the spring bean will be continued only when

factory method based spring bean instantation is enabled on singleon java class that is cfg as singleton scope spring bean

similar

class

note:: With out taking spring bean class code as singleton java class code, to bring the effect of singleton behavior

we need to take spring bean scope as the "singleton" scope

In real projects we take

DAO :: Data Access Object

service class (class with b.logics), DAO class (class with persistence logics), Controller class (class with monitoring logics)

as the singleton scope spring beans beoz generally these classes contains no state (data), fixed state or sharable state

=> All singleton scope spring beans participates in pre/eager/early instantiation(object creation) i.e the object will be created the moment IOC container is created irrespective of ctx.getBean(-,-) is called or not or dependency Injection are performed or not using

that spring bean. Only for singleton scope spring beans this pre-instantiation takes place

How to disable pre-instantiation on singleton scope spring beans?

ans) use @Lazy(true) along with @Component or @Bean methods to enable lazy/late instantiation on singleton scope spring

bean i.e the IOC container creates spring bean class obj only when need is there like ctx.getBean(-,-) is called having bean id

or the spring bean is made dependent to another spring bean

@Component("wmg")

@Scope("singleton")

@Lazy(true)

```
public class Wish MessageGenerator {

..

}
```

In AppConfig.java

==============

```
@Bean(name="dt1")

@Lazy(true)

@Scope("singleton") public Date createDate1() { System.out.println("AppConfig.createDate()1");

}

return new Date();

@Bean(name="dt2")

@Scope("singleton")

@Lazy(true)

public Date createDate2() { System.out.println("AppConfig.createDate()2");

}

return new Date();
```