In Core Programming of Spring boot we use the following different categories of annotations

a) Annotations for configuration

b) Annotations for Data supply /injection

c) Annotation for Spring bean life cycle

d) Misc Annotations

a) Annotations for configuration

=>These annotations are given to configure java classes as spring beans i.e making IOC contianer creating objects for java classes.

=> Providing info to underlying server/container/framwork by marking the java code is

called configuration. (This process gives inputs/instructions to underlying container/server/framework to recognize and use given java code in certain angel) a)@Component: To configure user-defined java class as spring bean stereo b)@Service :: To configure user-defined java class as spring bean cum service class type

c)@Repository:: To configure user-defined java class as spring bean cum DAO class annotation)@Controller :: To configure user-defined java class as spring bean cum web controller class

=> Service class contains b.methods having b.logic

=> webcontroller class can take and process the http requests given from the browser s/w b.logic/service logic :: calculations, analyzations, sorting, filtering and etc..

e)@RestController ::To configure user-defined java class as spring bean cum web controller class cum Rest API comp (Restful webservice) f)@ControllerAdvice::To configure user-defined java class as spring bean cum Exception Handler advice class in RestAPI (Resful web service) g)@Configuration :: To configure user-defined java class as spring bean cum configuration class

h) @Bean :: to configure pre-defined java class obj given by the method of @Configuration class as the spring bean and etc..

DAO class contains persistence logic -- the logic performing CURD operations on DB s/w

(insert, update, select, delete)

note:: we do not use @Configuration annotation directly in spring boot apps rather we use indirectly through @SpringBootApplication annotation

note:: All these are spring spefic annotations like note:: Multiple annotations doing the same work are called stereo type annotations

configuring java class as spring bean

b) Annotations for Data supply (For Injection activity)

=>These annotations are given to assign/inject data to spring bean properties (spring bean class obj member variables)

a) @Autowired :: To inject one spring bean class obj to another spring bean class obj's property either

byType or byName

:: To Inject simple vlaues to simple bean properties of spring bean either directly

(Dependency Injection)

b) @Value

or by collecting from properties file or system properties or env.. variables

c) @Qualifier

:: To resolve the ambiguity problem related to Depndency Injection

Should be used in

d) @Primary

:: To resolve the ambiguity problem in another way

note:: All these are spring specific annotations

Java config annotations

combiniation with @Autowired

(Given by jdk, jee apis and can be used in different frameworks)

=> These annotations make our code as more non-invasive becoz they are not spring specific annotations So moving code from one framework to another framework becomes easy

to

i) @Named :: To configure java class as spring bean and also resolve ambiguity Problem (Its work is the combination of stereo type annoation + @Qualifer)

ii) @Inject :: Alternate to @Autowired

@Component

iii) @Resource:: Alternate to @Autowired +@Qualifier

....

Why industry is not prefereing JAva Config annotation in spring /spring boot programming though they make our code as more non-invasive code ?

Ans) Very few annotationSare given as java config annotations so these annotations are not all sufficient to develop serious realtime project.. instead of using few spring api annotaiton s and few java config annotations in our spring or spring boot apps.. they are preferring to use only spring/spring boot api annotations in real projects

=> As of now there is no serious alternate for spring /spring boot framework.. So moving spring/spring boot project code another framework and executing in that framework with out code changes is pratically not going to happen.

C) annotations for spring bean life cycle

@PostConstruct :: To make spring bean class method as init life cycle method @PreDestroy :: To make spring bean class method as destroy life cycle method

|-->@EnableAutoConfiguration + @Componentscan + @Configuration

=> if the IOC container picksup the dependent spring bean based on the

type of HAS-A property on which @Autowired

is applied then it is called "ByType" mode of Autowiring

eg: @Primary picksup the dependent spring bean using ByType(class name)

=> if the IOC container picks up the

dependent spring bean based on the

bean id of the Dependent spring bean that is specified in

the @Qualifier(-) then it is called "ByName" mode of Autowiring

invasive = Tight coupling with F/w or Technology Apis/Libraries non-invasive = :Loose Coupling with F/w or Technology APs/Libraries

if needed we can place/use both spring /spring boot api annotations and java config annotations in single project

Both these annoations are

JavaConfig annotations.. i.e they

are not spring/spring boot annotations

D) Misc Annotations

a)@PropertySource/@PropertySources :: To cfg single or multiple properties files with spring app or spring boot app

b) @ComponentScan

:: To make IOC container to scan for stereo type annoation classes in the given package and sub pkgs

what is the advantages of spring/spring boot frameworks compare to Struts or JSF?

Ans) Struts or JSF are just web frameworks.. which can be used

c) @ImportResource:: To import spring bean cfg file(xml file) to @Configuration class (to link xml file with @Configuration class) :: To import one @Configuration to another @Configuration class

d) @Import

e) @Scope

f) @Lazy

and etc..

:: To specify the spring bean scope

:: To enable lazy instantiation on singleton scope spring bean

Second Example App in Spring Boot on Dependency Injection (strategy DP)

target class

==========

@Component("vehicle")

Vehicle @Qaulifier("cengg") (solution2) @Autowired

(Best)

|--->private Engine engine; (HAS-A property)

(spring Bean property)

public void jounery(String startPlace,

}

String endPlace){

How to resolve ambiguity Problem?

a) using @Qualifer(-) (Best)

b) Using @Primary

==========

Dependent classes

Primary (solution1) @Component("dengg")

a) DieselEngine

|-->implements Engine(I) @Component("pengg")

b) PetrolEngine

-->implements Engine(I) @Component("cengg")

c) CNGEngine

-->implements Engine(1) @Component("eengg")

c) EletricEngine

c) By matching target spring bean class property name with

one dependent class bean id;

-->implements Engine(1)

This solution makes @Autowire annotation

to perform ByType mode of Autowiring

These solutions make @Autowire performing ByName mode of Autowirng

note:: =>if all 3 solutions are applied pointing 3 different dependent spring beans then the @Qualifier(-) solution works as the final.

=> The bean id required for @Qualifier can be gathered from properties file if needed.

to just develop the web applications.. where as spring/spring boot are application frameworks /JEE frameworks which can be used to devleop the standalone Apps, webapplications, distributed Apps, Microservice architecture applications and etc..

Is Spring/Spring boot alternate to EJB?

Ans) No, EJB is given to develop the Distributed Apps, where as Spring/spring boot is given to develop the all kinds of apps including the Distributed apps

EJB is JEE Technology which gives complexity towards Distributed app developmentn

Spring/Spring boot are the frameworks which simplifies multiple Apps development like standalone apps, web apps, distributed apps and etc..

Is Spring/Spring boot alternate to JAVA-JEE Technologies?

Ans) No, Spring/spring boot framework compliments JAVA-JEE Technologies becoz they internally uses JAVA-JEE technologies to simplify the application development process.

note:: By default @Autowired performs ByType mode of Autowiring.

These two reasons makes @Qualifer(-) as the best solution for solving the ambiguity problem

Can i use @Service or @Repository in the place of @Component?

Ans) yes... But not recomanded

=>To make java class as spring bean use @Component

=>To make java class as spring bean cum service class (where b.logic is placed and TxMgmt is applied) use @Service. @Service annotation automatically enbles Transnaction Mgmt in certain conditions) note:: The process of executing related logics by applying do every thing or nothing principle is called TransactionMgmt..

transferMoney operations (withdraw money from source account

deposite moeny into destination account) needs Tx Mgmt support

=>To make java class as spring bean cum DAO class (where persistence logic is placed) use @Repository annotation This @Repository annotation translates techology specific exceptions to framework specific

exceptions like

jdbc exceptions/SQLException to spring /spring boot framework exceptions.

@Service = @Component++ @Repository = @Component++

@Controller = @Component++

@RestController = @Controller++

@Configuration = @Component++ and etc..

note:: These @Service, @Repository, @Controller, @RestController and etc.. annotations provide special indentity

for java classes that are acting as spring beans

@Service --> service class (for keeping b.logic) @Repository ---> Repository class (for keeping persistence logic) @Controller ----> web controller class (request delegation logic) @RestController --> Distriuted comp/Restful App

What happens if we configure service, DAO classes using @Component annotation?

Ans) Yes, we can configure .. But additional behaviour required for the spring bean should be added manually/explicitly

=>Like we need to enable Transactionmgment on service class manually

=> we need to enable Exception trasnlation facility on DAO class manually

Not a recomanded

process.

Example Application

=====

```
//IEngine.java package com.nt.sbeans;
public interface IEngine {
}
public void start();
public void stop();
//EletricEngine.java
package com.nt.sbeans;
import org.springframework.context.annotation.Primary; import
org.springframework.stereotype.Component;
@Component("eEngine")
//@Primary
publicas EletricEngine implements IEngine {
public EletricEngine() {
}
//CNGEngine.java
package com.nt.sbeans;
import org.springframework.stereotype.Component;
```

```
@Component("cEngine")
publiss CNGEngine implements IEngine {
public CNGEngine() {
System.out.println("CNGEngine:: O-param constructor");
}
System.out.println("EletricEngine:: O-param constructor");
@Override
@Ove@ideerride
public void start() {
System.out.println("CNGEngine.start():: Engine started");
public void start() {
}
System.out.println("EletricEngine.start():: Engine started");
@Override
}
@Override
public void stop() {
System.out.println("EletricEngine.stop():: Engine stopped");
}
}
}
//DieselEngine.java
package com.nt.sbeans;
import org.springframework.stereotype.Component;
@Component("dEngine")
public inals Diesel Engine implements IEngine {
public DieselEngine() {
//DieselEngine.java
public void stop() {
System.out.println("CNGEngine.stop():: Engine stopped");
package com.nt.sbeans;
import org.springframework.stereotype.Component;
@Component("pEngine")
publfiness PetrolEngine implements IEngine {
public PetrolEngine() {
System.out.println("Pertrol Engine:: O-param constructor");
System.out.println("Diesel Engine:: O-param constructor");
```

```
}
@Override
}
@Override
public void start() {
public void start() {
}
@Override
public void stop() {
System.out.println("Diesel Engine.start():: Engine started");
}
@Override
public void stop() {
System.out.println("PertrolEngine.start():: Engine started");
System.out.println("Diesel Engine.stop():: Engine stopped");
System.out.println("Pertrol Engine.stop():: Engine stopped");
}
}
}
//Vehicle.java
package com.nt.sbeans;
import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.beans.factory.annotation.Qualifier; import org.springframework.stereotype.Component;
@Component("vehicle")
publiceass Vehicle{
//HAS-A property
@Autowired
@Qualifier("dEngine")
private IEngine engine;
public Vehicle() {
System.out.println("Vehicle:: 0-param constructor)");
}
//b.method
public void journey(String sourcePlace, String destPlace) {
System.out.println("Vehicle.joueney()");
engine.start();
System.out.println("Journey started at ::"+sourcePlace); System.out.println("Journey is going on.....");
```

```
engine.stop();
System.out.println("Journey stopped at::"+destPlace);
}
}
//main class
package com.nt;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import com.nt.sbeans.Vehicle;
@SpringBootApplication
public class BootlocProj02DependencyInjectionApplication {
public static void main(String[] args) {
//get IOC container
ApplicationContext ctx-SpringApplication.run(BootlocProj02DependencyInjectionApplication.class, args);
//get Access to target spring bean class object
Vehicle vehicle=ctx.getBean("vehicle",Vehicle.class);
//invoke the b.method
vehicle.journey("Hyd", "Delhi");
//close the container
((ConfigurableApplicationContext) ctx).close();
Boot-IOCProj02-DependencyInjection [boot]
> Spring Elements
#src/main/java
com.nt
> BootlocProj02DependencyInjectionApplication.
com.nt.sbeans
> CNGEngine.java
> DieselEngine.java
> EletricEngine.java
> IEngine.java
>PetrolEngine.java
› Vehicle.java
> src/main/resources
> JRE System Library [JavaSE-17]
> Maven Dependencies
```

src/test/java

}

}

TELS

========

====|_____|=|____

:: Spring Boot ::

(v3.1.4)

> src

› target

W HELP.md

mvnw

mvnw.cmd

M pom.xml

Jump to first unred

Who can see your messages? Recording

To: Abhijeet Kumar (Direct Message)

2023-10-07T18:51:38.806+05:30 INFO 22820 --- [main] tlocProj02DependencyInjectic Type message here...

2023-10-07T18:51:38.810+05:30 INFO 22820 --- [main] tlocProj02 DependencyInjectio

CNGEngine:: 0-param constructor

DieselEngine:: O-param constructor

EletricEngine:: O-param constructor

PertrolEngine:: O-param constructor

Vehicle:: O-param constructor)

Vehicle.joueney()

Journey started at ::Hyd

2023-10-07T18:51:39.684+05:30 INFO 22820 --- [main] tlocProj02DependencyInjectionApplication: Started BootlocProj(

DieselEngine.start():: Engine started

Journey is going on.....

DieselEngine.stop():: Engine stopped

Journey stopped at::Delhi

Activate Windows

Go to Settings to activate Windows.

Assignment

Student (target class)

|--> private

IMaterial material;

(HAS-A property)

|---> public void prepare(-,-){

//dependent classes

Implements IMaterial(I)

JavaCourseMaterial PhpCourseMaterial DotNetCourse Material

}

...