simple

Injecting Data to Spring bean properties

=> Injecting values to spring bean properties is called data Injection and this can be done in two ways in spring boot apps

1) using @Value annotation (works in both spring,springboot apps) =>here we need to inject values to each bean property seperately

2) using @ Configuration Properties(works only in spring boot apps) => Here bulk injection is possible. values to all/multiple spring bean properites can be injected at once.

note::@Autowired performs singular injection towards injecting one spring bena class obj to another spring class obj

Every spring bean allows 3 types of properties

a) simple type properties (primitive data type /wrapper data type properties including String)

b) Array or Collection type propeties

c) HAS-A type/Object type propeties (To Inject values HAS-A/Object Type properties we need to use @Autowired annotation)

to

=> In spring boot we can inject values simple, array, collection type properties by using

two Data annotations

For Injecting values to simple or string type properties

use @Value or @Configuration Properties annotation

a) @Value (Given By spring and does not support bulk injection) (can be used in both spring,spring boot)

b) @Configuration Properties (Given by spring boot and can be used for bulk Injection) (can be used only in spring boot)

=>By using @ConfigurationProperties only for 1 time on the top of spring bean class

we can inject values to multiple spring bean properties.. This called bulk injection @Value does not support bulk injection where as @ConfigurationProperties supports the same bulk Injection.

eg1: if spring bean is having 10 simple properties then we need to use @Value for 10 times

to inject the simple values (Here @Value should used at field level)

eg1: if spring bean is having 10 simple properties then we need to use @ConfigurationProperties

for 1 time to inject the simple values (Here @ConfigurationProperties should be used at class level) to multiple spring bean properties

@Value

=> It is spring supplied annotation .. So it can be used in both spring and spring boot apps

=> It is filed level annotation

=>This annotation can be used for the following operations

a) To assign direct values to spring bean properties

b) To assign values gathered from properties file(s) to spring bean properties

sion,

c) To assign system property values to spring bean properites (like os.name, os.ver and etc...)

d) To assign env..variable values to spring bean properites (like Path env.. variable value)

SPEL is inspired from Jsp EL

(compledted in spring

classes as part of @Value annotation examples)

e) To work with SPEL (Spring Expression Language) (To perform arithmetic and logical opeartions towards the injection)

and etc..

=> application.properties/yml can have only pre-defined keys or only user-defined keys or both (mix of both)
=> we do not need nfigure application.properties/yml file using @PropertySource annotation

becoz it is automatically recognized during app startup activities happening in the SpringApplication.run(-) method call user-defined

=> if want to configure more other properties files then we can use @PropertySource annotation on top of spring bean classes or main class.

and logical

This is given to configure custom properties file

=>SPEL is given to perform arithemetic operations while injecting the values to spring beans properties syn:: #{<expression>}

=>SPEL is used always in combination with @Value annotation to perform values injection to spring bean properties by performing arithmetic and logical operatior's..

@Value("#{<expression>}")

can be a arithmetic or logical operation

@ImportResource --> To link the spring bean cfg file (xml) with @Configuration class

@Import ------>To link the one @Configuration With other @Configuration class

@PropertySource or @PropertySoruces -----> To confugure one or user-defined multiple properites files to the spring/spring boot app

Here expression is nothing but arithmetic or logical operaions

private <datatype> <propertyname>;

be

${<key>} --- represents placeholder

that means <key> is not the value to be injected..

It is like "?" (positional param) of SQL Query.

Example App

==========

#{<expr>} --> SPEL results will injected ${<key>} --> the value of key collected from

✓ BootProj04-ValueAnnotation-SPEL [boot] >Spring Elements

#src/main/java

✓ com.nt

properties file or system property

or env. variable or other place holder value will be injected

✓ BootProj04ValueAnnotationSpelApplication.java

› BootProj04ValueAnnotationSpelApplication

com.nt.sbeans

› Hotel.java

> MenuItemPrices.java

#src/main/resources

application.properties

>src/test/java

> JRE System Library [JavaSE-17]

Maven Dependencies

>

>

src

> target

WHELP.md

mynw

mvnw.cmd Mpom.xml

starter: lombok

application.properties

 # Menu prices Info menu.dosarate=60

menu.idlyrate=30 menu.poharate=50

menu.vprate=50

#Hotel Info

hotel.name=Amoga Hotel hotel.addrs=hyd

hotel.contactno-989998824

the value will come from other place based on the <key>

=>JSP -EL is different from SPEL but the SPEL is inspired from JSP-EL

Expression Languages related to JAva are JSP-EL (Jsp Epression Language) SPEL (Spring EL)

OGNL (Object Graph Notation Language) JBOSSEL

and etc...

from the properties file,

system props, env.. variables and etc..

=> In jsp programing, in jsp pages to perform arithmetic and logical operations with out using java code then prefere using JSP-EL

syn:: ${<expr>} ==> ${4+5} =>gives 9

=> In spring programming to inject values to spring bean properties by performing arithmetic and logical operations then prefer using SPEL

note: SPEL (Spring Expression Language) can be used in both spring and spring boot apps

using

syn:: #{<expr>} eg:: @Value("#{ct.idlyPrice + ct.dosPrice+ ct.wada Price}")

note: While SPEL logics performing arithmetic and logical operations

on spring bean properties we need to use <bean id>.<property name> not the keys placed in properties file

private double billAmt;

Here "ct" is the bean id and dosaPrice or

wadaPrice is bean properties

Hotel.java

#Customer Info customer.name=raja

To add extra starter in the middle of the Project development we need to use the following procedure

Right click on the Project ---> spring ---> add starter --> select starter ---> select pom.xml ---> apply (Lombok)

=>The SPEL based Arithmetic and Logical Operations must not be done on the same class spring bean properties i.e must happen on other class spring bean properties being from one spring bean class..

//Hotel.java

```
package com.nt.sbeans;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
@Component("hotel")
public class Hotel {
@Value("12345") //Direct value injection
private Integer hotelId; @Value("${hotel.name}") private String hotelName; @Value("${hotel.addrs}") private String hotelAddrs; @Value("${hotel.contactno}") private String mobileNo;
@Value("${customer.name}")
private String custName;
Values collected
from properites file
@Value("#{menup.idlyPrice + menup.dosa Price}") //SPEL for arithmetic operation
private Float billAmount;
@Value("${os.name}") // os.name is fixed system property key
private String osName;
@Value("${user.name}") //user.name is fixed system property key private String windowsUser;
@Value("${Path}") // Path is fixed env variable name
Injecting System properties
Injecting Environment variable value.
}
private String pathData;
//toStirng() (alt+shift+s, s)
@Override
```

```
public String toString() {
return "Hotel [hotelId=" + hotelId+", hotelName=" + hotelName + ", hotelAddrs=" + hotelAddrs + ",
mobileNo=" + mobileNo + ", custName=" + custName + ", billAmount" + billAmount +", osName=" + osName
+", windowsUser=" + windowsUser + ", pathData=" + pathData + "]";
}
//MenuItemPrices.java
package com.nt.sbeans;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import lombok.Data;
@Component("menup")
@Data
public class MenuItemPrices {
@Value("${menu.dosarate}") private Float dosaPrice; @Value("${menu.idlyrate}") private Float idlyPrice;
@Value("${menu.poharate}") private Float pohaPrice; @Value("${menu.vprate}") private Float vadaPavPrice;
Injecting values
by collecting
from properties file
}
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.Hotel;
@SpringBootApplication
public class BootProj04ValueAnnotationSpelApplication {
args);
public static void main(String[] args) {
//get IOC container
ApplicationContext ctx-SpringApplication.run(BootProj04ValueAnnotationSpelApplication.class,
//get Hotel object ref
Hotel hotel=ctx.getBean("hotel",Hotel.class);
System.out.println(hotel);
//close the IOC container
((ConfigurableApplicationContext) ctx).close();
}//main
```

}//class

Second Example App

✓ BootProj05-DataInjection-Using ValueAnnotation [boot]

> Spring Elements

✓ src/main/java

✓com.nt

› BootProj05DataInjection Using ValueAnnotationApplication.java

com.nt.sbeans

> DiagnosticCenterCatalog.java

> Patientinfo.java

#src/main/resources

application.properties

>

src/test/java

>

JRE System Library [JavaSE-17]

› Maven Dependencies

> src

> target

WHELP.md

mvnw

mynw.cmd Mpom.xml

//DiagnosticCenterCatalog.java

package com.nt.sbeans;

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

import lombok.Data;

@Component("dcc")

@Data

public class DiagnosticCenterCatalog {

@Value("${dc.xrayRate}")

private Double xrayPrice; @Value("${dc.ctscanRate}") private Double ctscanPrice; @Value("${dc.mriScanRate}") private Double MRIScanPrice; @Value("${dc.ecgRate}")

private Double ecgPrice;

Injecting values to spring bean

properties by collecting them from properties file

//setters && getters (getters are required for SPEL operations)

public double getXrayPrice() {

```
}
return xrayPrice;
public void setXrayPrice(double xrayPrice) {
}
this.xrayPrice = xrayPrice;
public double getCtscanPrice() {
}
return ctscanPrice;
public void setCtscan Price(double ctscanPrice) {
//PatientInfo.java
package com.nt.sbeans;
}
}
this.ctscanPrice = ctscanPrice;
public double getMRIScanPrice() {
}
return MRIScanPrice;
public void setMRIScanPrice(double mRIScanPrice) {
}
MRIScanPrice = mRIScanPrice;
}
public double getEcgPrice() {
return ecgPrice;
public void setEcgPrice(double ecgPrice) {
}
this.ecgPrice = ecgPrice;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import lombok.Data;
@Component("plnfo")
@Data
public class Patientinfo {
```

@Value("10001") | Direct value Injection

```
private Integer pid; @Value("${pi.name}") private String pname; @Value("${pi.mobileNo}") private Long mobileNo;
@Value("${pi.addrs}")
private String paddrs;
```

Injecting the values to spring bean

properties by collecting them

from properties file

```
@Value("#{dcc.xray Price + dcc.ctscan Price+ dcc.ecgPrice}") //SPEL based Injection (dcc- bean id, xray Price- property name)
private Double billamount;
@Value("#{dcc.ecgPrice<=0}") //SPEL
private Boolean ecgFree;
@Value("${os.name}")
private String osName; //To hold system property value
@Value("${Path}")
private String path Data; //To hold env.. variable value
//setters && getters
public Integer getPid() {
return pid;
}
public void setPid(Integer pid) {
this.pid = pid;
}
public String getPname() {
return pname;
}
public void setPname(String pname) {
this.pname = pname;
}
public Long getMobileNo() {
return mobileNo;
}
public void setMobileNo(Long mobileNo) {
this.mobileNo = mobileNo;
}
public String getPaddrs() {
return paddrs;
}
public void setPaddrs(String paddrs) {
this.paddrs = paddrs;
}
public Double getBillamount() {
```

```
return billamount;
}
public void setBillamount(Double billamount) {
this.billamount = billamount;
}
public String getOsName() {
return osName;
}
public void setOsName(String osName) {
this.osName = osName;
}
public String getPathData() {
return pathData;
}
public void setPathData(String pathData) {
this.pathData = pathData;
}
//toString()
public Boolean getEcgFree() {
return ecgFree;
}
public void setEcgFree(Boolean ecgFree) {
this.ecgFree = ecgFree;
}
@Override
```

fixed system property name

Fixed env.. variable name

=>Using SPEL, we inject one spring bean class obj to another spring bean class with the support of @Value annotation i.e @Autowired annotation is not required

```
@Autowired
private CourseCatalog catalog; (or)
|----------> another spring bean id
@Value("#{catalog}")
private CourseCatalog catalog1;
```

application.properties

```
#DiagnosticCenterCatalog
dc.xrayRate=1000
```

dc.ctscanRate=3000

dc.mriScanRate=10000 dc.ecgRate=600

 # patient information

pi.name=rajesh

pi.addrs=hyd

pi.mobileNo=9999999

public String toString() {

return "Patientinfo [pid=" + pid + ", pname=" + pname + ", mobileNo=" + mobileNo + ", paddrs=" + paddrs

+ ", billamount=" + billamount + ", ecgFree=" + ecgFree + ", osName=" + osName + ", pathData=" + pathData

}

}

+ "]";

//BootProj05DataInjection UsingValueAnnotationApplication.java

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.Patientinfo;

@SpringBootApplication

public class BootProj05 DataInjection UsingValueAnnotationApplication {

public static void main(String[] args) {

//get IOC container

ApplicationContext ctx-SpringApplication.run(BootProj05DataInjection UsingValueAnnotationApplication.class, args);

//get Patientinfo Spring bean class object

Patientinfo info=ctx.getBean("pInfo", Patientinfo.class);

System.out.println(info);

//close the container

note:: SPEL expressions can not applied on the same spring bean properties

((ConfigurableApplicationContext) ctx).close();

}

it must be applied other spring bean properties from the current spring bean definitation

}

AV

IV

_()_

| '_ | '_| | '_V_` | \ \ \ \

(_| |))))

__| | __ ____/ | / / //

:: Spring Boot ::

(v3.0.1)

\NTSPBMS616-Boot\BootProj05-DataInjection-UsingValueAnnotation)

2023-01-05T20:25:48.787+05:30 INFO 17252 --- [main] InjectionUsingValueAnnotationApplication: Starting BootProj05 DataInjection UsingValueAnnotationApplication using Java 17.0.2 with PID 17252 (E:\Worskpaces\Spring\NTSPBMS616-Boot\BootProj05-DataInjection-UsingValueAnnotation\target\classes started by NATARAJ in E:\Worskpaces\Spring 2023-01-05T20:25:48.790+05:30 INFO 17252 --- [main] InjectionUsingValueAnnotationApplication: No active profile set, falling back to 1 default profile: "default" 2023-01-05T20:25:49.245+05:30 INFO 17252 --- [main] InjectionUsingValueAnnotationApplication: Started BootProj05DataInjection UsingValueAnnotationApplication in 0.807 seconds (process running for 1.253) Patientinfo [pid=10001, pname=rajesh, mobileNo=9999999, paddrs=hyd, billamount=4600.0, ecgFree=false, osName=Windows 11, pathData=E:/Softwares/Eclipse/eclipse-jee-2022-03-R-

win32-x86_64/eclipse//plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208/jre/bin/server;E:/Softwares/Eclipse/eclipse-jee-2022-03-R-win32-x86_

64/eclipse//plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208/jre/bin;C:\Program Files\Java\jdk1.8.0_31\bin;E:\Worskpaces\advjava\NTAJ916\basics

\PATHDemo;C:\Program Files\MongoDB\Server\4.4\bin;.;C:\Program Files\dotnet\;E:\Softwares\Eclipse\eclipse-jee-2022-03-R-win32-x86_64\eclipse;]

=>IOC container creates and manages the special object called Enviroment object having the values collected from the properties file,

env variable values, System property values and etc...

properties file

key1=val1

✓ PropertyResolver

IOC container (Spring bean)

Enviroment object

Spring bean obj1

@Value({key1")

private String datal;

@Value ${os.name}") private String datal;

os.name=windows1

Spring Bean ob 2

ath.

✓ Environment

✓ ConfigurableEnvironment

✓ AbstractEnvironment

MockEnvironment

✓ StandardEnvironment

ApplicationEnvironment

> Standard ReactiveWebEnvironment

✓ Configurable ReactiveWebEnvironment

Standard ReactiveWebEnvironment

=>Enviroment object class name is "ApplicationEnviroment" that implements Environment(1)

=> Environment object means it is the object of java class that implements Environment obj

key1=val1

System properties

os.name=window11

Environmnet variables

path=e:/abc;d:/abc;.

How to configure custom properties file in spring boot app?

Ans) it can be done in two ways

eg1::

a) using @PropertySource annotation

b) using spring.config.import key of the application.properties file (best)

@SpringBootApplication

@PropertySource("myfile1.properties")

public class BootIocProj07ValueAnntationSpelApplication {

Q) if the custom properties files are configured using both approaches like (@PropertySource approach and spring.config.import key approarch) then whose value will be taken as the final value to inject to spring bean property if both the properties file are having same keys with different values? Ans1) The value kept in the properties file that is configured using spring.config.import key will be used as final inject value of the spring bean property

eg2:

in application.properties

#Configure the custom properties file spring.config.import=myfile.properties

if we take same keys with different values in application.properties file and

in custom properties then which value will be taken as the final value?

Ans) the value kept in custom properties file will be taken as the final value to Inject

(if the custom properties is configured using approach2 (spring.config.import key) then

the content of custom properties will be taken as the final value .if the same custom properties file is cfg using approach1(@PropertySource) then application.properties file

will be used)

note :: In spring programming if we give wrong <key> in the place holder ${<key>} of

@Value annotation then ${<key>} itself will be injected to the spring bean property

In spring boot programming, we get exception for the same

Procedure to perform Debugging on Spring boot Layered App

=====

========

=> Debugging is all about knowing the flow of execution in the code end to end

=> It is every useful to while fixing bugs and issues

=> Useful to observe the data change that happens in the variables time to time in the execution of the app

=> if u join the project that is already there in the Maintainenance mode then we need to think about working with Debugging activities to understand the code flow

=> Every IDE gives its own Debugger as the built-in tool

=> Important terminologies in Eclipse Debuggin

a) Break Point :: The point in the code (line number) upto which the code executes normally, From there the control comes to programmer to see the further lines of execution.

b) Short cut keys for seeing the flow

i) F5:: step Into (Gets into the details of the method call)

ii) F6:: step over (executes the current method call with out getting its details)

iii) F7 :: Step return (Executes current method definitation until the return statement executes)

iv) F8: Executes till next break point

c) Drop To Frame ::

if we perform this operation from the middle of method definitation.. the code executes one more time from the the beginning of the method.

Procedure to follow

==========

35

step1) create Break point in the first line of the main(-) method

(By double clicking in the status line or by using ctrl+shift+B)

JI。 ᵀᵀQᵀᵀᵀDD ᵀD ᵀDᵀ ᵀDᵀᵀNᵀᵀᵀᵀ

ApplicationContext ctx-SpringApplication.run(BootlocProj05MiniProjectLayeredAppApplication.class, args);

step2) Run the application in debugging mode (use Debug as java app)

Right click on main class ---> debug as --> java app --> ....

step3) switch to debugging prospective window and perform following operations

=> For user-defined method calls use F5 Option (Step Into)

=> For pre-defined method calls use F6 option (Step Over)

=> To complete method execution until the return the statemen use F7 (Step Return)

=> To see certain method logics execution from the beginning once again take the support (Drop to Frame option)