**Spring Boot By Dhanamjaya**

Date : 22/06/2020

Spring Boot New Batch - 7:45 PM

---------------------------------------------------

7:45PM to 9:20 PM

FB:

https://www.facebook.com/groups/thejavatemple/

Email:

javabyraghu@gmail.com

Admin: Sirkanth

+91- 630 296 8665

-----------------------------------------------------

Pre-Req:

Core Java | Adv Java basics

Spring Core Basics ( Covered in Course)

Hibernate basics ( Covered in Course)

Spring Boot RestFul websevices (Covered in Course)

----------------------------------------------

Spring Boot and Microservices - 4 Months

Tools :

Eclipse/STS

Maven/Gradle

JUnit + Mock

Docker

Cloud Deploy (PCF)

Mini Project

Angular UI

Security Project

OAuth | JWT Project

Curd Operation MYSQL/Oracle DB

Thymeleaf / JSP

Pagination / AJAX

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

Spring Boot

Spring Boot is a Spring based framework that behaves as a Project

which reduces common works (AutoConfiguration) by Programmer to create

our Project.

\*) AutoConfiguration reduces work done by programmer, by providing common files

(classes, jars..etc)

\*) In Spring, programmer is writing code from line#1, In Spring boot it reduces

common code by programmer.

------Spring f/w Database connection Example-----------

@Configuration

public class AppConfig {

@Bean

public DataSource dsObj(){

DriverManagerDataSource ds = new DriverManagerDataSource();

ds.setDriverClassName("OracleDriver");

ds.setUrl("jdbc:oracle");

ds.setUsername("nit");

ds.setPassword("root");

return ds;

}

}

-------Spring Boot Database connection Example ----

spring-boot-starter-jdbc (AutoConfiguration)

=> This line provide above code into child project from Parent Project.

=> But, programmer please provide values using properties file

--application.properties--

driver-class-name=OracleDriver

url=jdbc:oracle

username=nit

password=root

----------------------------------------------

\*) Here starters (ex: spring-boot-starter-jdbc) means READY MADE CODE

GIVEN BY PARENT PROJECT (SPRING BOOT), BUT IT IS PARTIAL CODE.

spring-boot-starter-web

spring-boot-starter-security

spring-boot-starter-thymeleaf

spring-boot-starter-devtools

spring-boot-starter-data-jpa (ORM/Hibernate)

..etc

spring-boot-starter-web reduces below things by programmer

a. FrontController (web.xml)=>DispatcherServlet

b. Enable MVC Annotations

c. ViewResolver

d. HandlerMapping

Q) Where can we add starter in our Project?

A) Spring boot project must be created either using Maven Tool or using Gradle Tool

Maven Tool

pom.xml :

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

<version>2.3.1.RELEASE</version>

</dependency>

Gradle Tool

build.gradle:

compile group: 'org.springframework.boot',

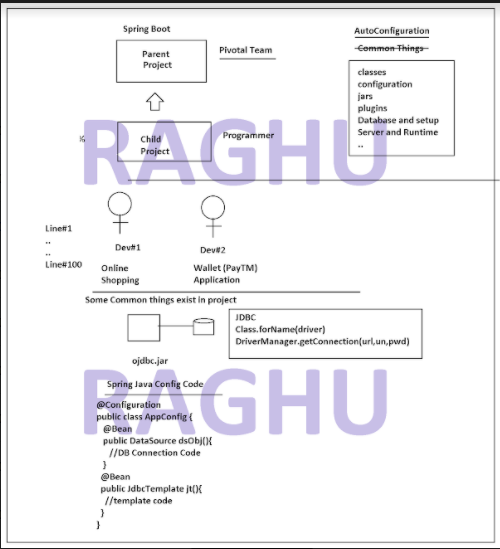
name: 'spring-boot-starter-jdbc',

version: '2.3.1.RELEASE'

---------------------------------------------------------

Duration : 4Months

7:45 PM - 9:20 PM (6 days)



Date : 23/06/2020

Spring Boot 7:45 PM

--------------------------------------------------------------

Core Java Good OOPs | Basic of Adv Java

Duration : 3 Months

Timing : 7:45 PM - 9:30 PM

Spring Boot with Cloud and Microservices

Question: Chat box (end of session)

Email : javabyraghu@gmail.com

FB : https://www.facebook.com/groups/thejavatemple/

Admin: Sirkanth

+91- 630 296 8665

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

Distributed Application using Spring Boot

Distributed Application :

An application that runs at multiple devices is called as Distributed Application

Ex: SBI Bank App

NetBaking ( https://online.sbi.com)

ATM

Mobile Application (SBI YONO APP)

Card (Credit/Deibt Cards)

Google Pay + SBI (3rd Party Integration)

IVR (Interactive Voice Response)

\*) Distributed Applications are implemented using Frontend and Backend technologies

\*) Here Few FrontEnd Technologies : Angular, React, Vue, Andiod, iOS,..etc

\*) Backend technologies : Spring Boot with Cloud

\*)Backend Application is implemented using Layers. Those are given as:

a. DAL = Data Access Layer (To perform Db Operation )

Concept : Spring Boot Data JPA

b. SL = Service Layer / Business layer

Calculation, Logics, Sorting, Validations,Transaction management..etc

c. IL = Integration Layer

It uses Http Protocol to communicate with All client application.

Data is exchnaged using JSON format.

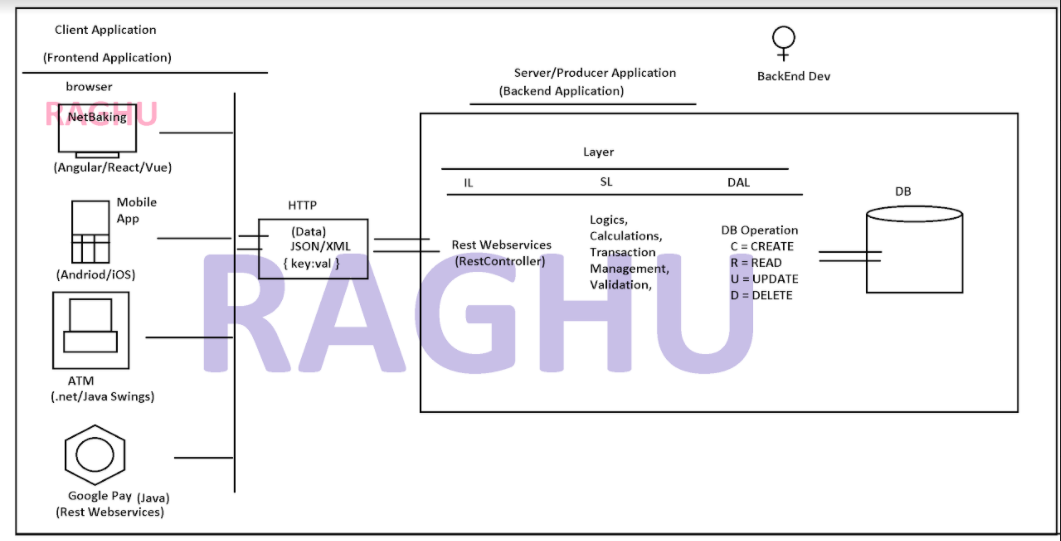
JSON = Java Script Object Notation. This is used to trasnfer data in Global Format.

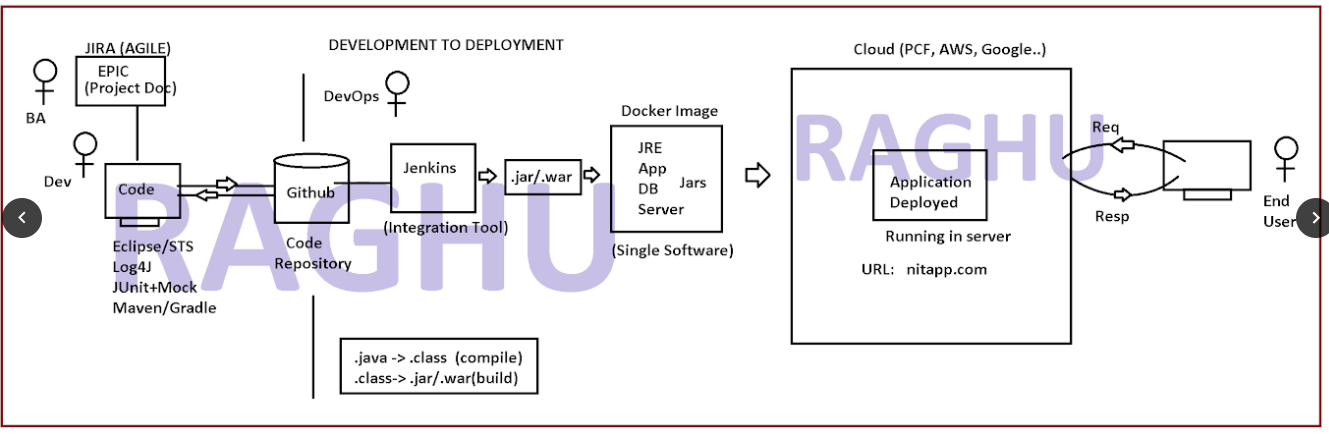
JSON can be understand by every high level programming/Script Language.

Ex: Angular, PHP, Pythong, Java..etc

\*\*\* Fullstack Developer = UI technologies + BackEnd technologies + Database + DevOps

------------------------------------------------------------------------

****

****

Date : 24/06/2020

Spring Boot 7:45 PM

Mr. RAGHU

-----------------------------------------------------------

Duration: 3 to 4 Months

Timing : 7:50 - 9:20

Spring Boot with Cloud and Microservices

Core Java | Basics of Adv Java

Spring Core | Hibernate | Webservices

Question: Chat box (end of session)

Email : javabyraghu@gmail.com

FB : https://www.facebook.com/groups/thejavatemple/

Admin: Sirkanth

+91- 630 296 8665

----------------------------------------------------------------------------

Demo Session | Tools Videos:

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

------------------------------------------------------------------------------

Spring Boot First Application

=> To write Spring Boot application we have to either Maven or Gradle Tool.

=> Every Spring Boot application contains 3 files (Setup files).

a. Main class | Starter class

b. Input Files ( application.properties | application.yml )

c. Build Information file ( pom.xml - Maven | build.gradle - Gradle)

=> These setup files are auto-generated while creating project in STS(Eclipse)

--------------------------

a. Main class | Starter class

=> This class is used to run application and creates Spring container

--Sample code--

@SpringBootApplication

public class MyApp {

public static void main(String[] args){

SpringApplication.run(MyApp.class, args);

}

}

=> Spring container are two types

a. BeanFactory (old container) - XML configuration

b.\*\*\* ApplicationContext (new container) - XML/Annotation/Java Config

=> In Spring Boot uses new container only.

=>\*\*\*\*\* Spring container does

a. Find/Scan/Detected Programmer classes (Component Scan)

b. Create object to detected/find classes.

c. Provide data to objects

d. Link one object with another object (Based HAS-A relation)

e. Destroy container when we stop application.

=> Programmer has to provide 2 files.

a. Spring Bean : class + that follows rules given by Spring f/w

b. Spring Configuration : XML/\*\*Java/Annotation\*\*\*

( It provides details of objects/data and links)

-----------------------------------

b. Input Files ( application.properties | application.yml )

=> In case of Spring, most of code (and object creations) is given by Spring boot only

when we add starters : ex: spring-boot-starter-jdbc (AutoConfiguration)

Here, Database connection object is created by Spring boot only in container.

=> But, Programmer has to provide inputs like driverclassname, url, username and password

with values inside application.properties | application.yml.

=> key are pre-defined, to find them link is:

https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html

--application.properties---

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:ORCL

spring.datasource.username=nareshit

spring.datasource.password=raghu

---------------------------

--application.yml--

spring:

datasource:

driver-class-name: oracle.jdbc.driver.OracleDriver

url: jdbc:oracle:thin:@localhost:1521:ORCL

username: nareshit

password: raghu

-------------------

\*) Incase of Spring, to load properties file, we need to write code like:

@PropertySource("classpath:application.properties")

which is not required in Spring boot.

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

c. Build Information file ( pom.xml - Maven | build.gradle - Gradle)

If we take pom.xml for Maven concept, then that file contains

=> Parent Project

=> Current Project

=> Properties (versions-Java version, Spring cloud version..etc)

=> Dependencies (Jars)

=> Dependency Management

=> Build Plugins (Our code => .jar/.war)

--------------------------------------

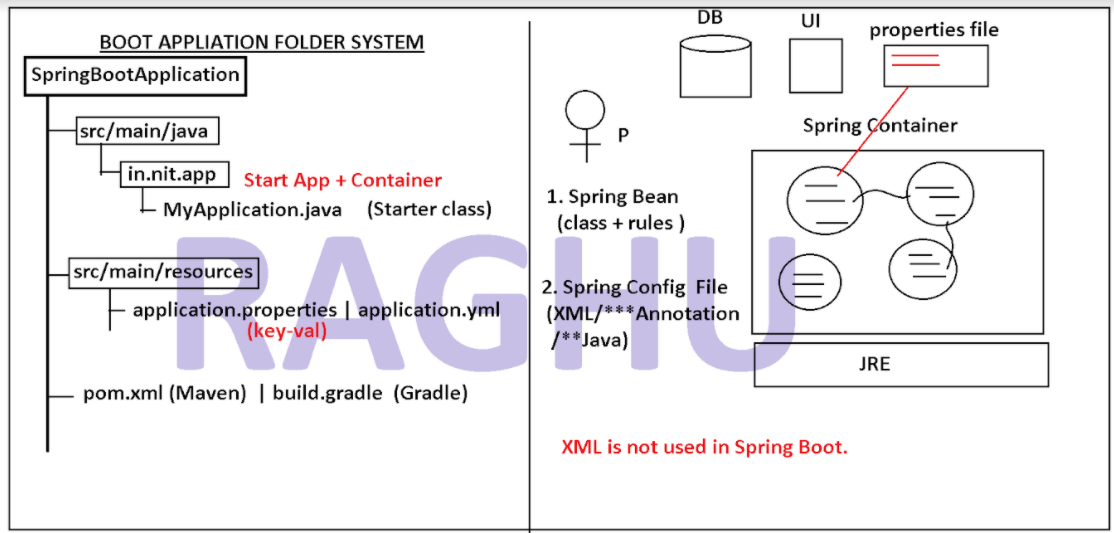
0-2+ => 3-8 LPA

3+ yrs => 6-12 LPA

Full Stack +4 LPA

Angular + Spring Boot + Oracle

Angular + Spring Boot + DevOps

****

Date : 25/06/2020

Spring Boot 7:45 PM

-------------------------------------------------------------------------

Duration: 3 to 4 Months

Timing : 7:50 - 9:20

Spring Boot with Cloud and Microservices

Core Java | Basics of Adv Java

Spring Core | Hibernate | Webservices

Question: Chat box (end of session)

Email : javabyraghu@gmail.com

FB : https://www.facebook.com/groups/thejavatemple/

Admin: Sirkanth

+91- 630 296 8665

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Spring Configuration in Spring Boot

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Configuration is used to provide information to container.

Details like

a. Object name and Object scope

b. Data for variables

c. Links details (HAS-A Relation details).

----------Annotation Configuration ----------------------

a. StereoType Annotation (5):-

------------------------------

If we apply StereoType annotation over class then it indicates (inform) to container

"CREATE OBJECT TO THIS CLASS".

StereoType Annotation means "CRETE OBJECT" => Container.

These are 5 types:

i. @Component = Creates object your classs

ii. @Repository = Creates object your classs + DB Operations

iii. @Service = Creates object your classs + Transaction Management + Logics/Calculation..etc

iv. @Controller = Creates object your classs + HTTP Protocol + Web Application + UI

v. @RestController = Creates object your classs + HTTP + Webservices

\*\*\* StereoType Annotation can not be applied to abstract class/interface.

---Examples---------

EX#1

@Component

public class DbConnection { }

::-> Spring container will create object to 'DbConnection'.

Like : DbConnection dbConnection = new DbConnection();

EX#2

@Component("emp")

public class EmployeeExport { }

::-> Spring container will create object to EmployeeExport

Like: EmployeeExport emp = new EmployeeExport();

EX#3

@Component("pob")

public class ProductPdfView { }

::-> Spring container will create object to ProductPdfView

Like : ProductPdfView pob = new ProductPdfView();

-------------------------------------------------------

b. Data Annotations (4):-

-------------------------

@Value

@Autowired

@Qualifier

@Primary

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Softwares:

JDK 1.8 to 14

STS : https://spring.io/tools

> After Download STS (Downloaded as JAR) > Double click for AutoExtract

> Open Folder "sts-4.5.1.RELEASE"

> Click on icon (option) "SpringToolSuite4.exe"

> Workspace (folder name) : ex: D:/SpringBoot745PM\_JUNE\_2020

> Finish.

#1. Create one new project

> File > new > Spring Starter project

> Enter details

Name : SpringBoot2ConfigFirstApp

GroupId: in.nareshit.raghu

Package: in.nareshit.raghu

> Next > Next > Finish.

\*\*\*\*\* ctrl + or ctrl - (to Zoom In/Out of Font in Text Editor)

\*\*\*\*\* ctrl+shift+O (to get imports)

#2. Create one class under package

> right click on package name (in.nareshit.raghu)

> new > class > enter details

name : EmployeeExport

--code---

package in.nareshit.raghu;

//ctrl+shift+O (imports)

import org.springframework.stereotype.Component;

@Component("eobj")

public class EmployeeExport {

public EmployeeExport() {

System.out.println("OBJ CREATED");

}

public void showExportMsg() {

System.out.println("FROM EXPORT MESSAGE");

}

}

#3. Modify Starter class

package in.nareshit.raghu;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class SpringBoot2ConfigFirstAppApplication {

public static void main(String[] args) {

//get container reference

ApplicationContext ac=SpringApplication.run(SpringBoot2ConfigFirstAppApplication.class, args);

//read object-getBean(objName,className.class);

EmployeeExport ob=ac.getBean("eobj",EmployeeExport.class);

ob.showExportMsg();

}

}

#4. Run Application (ctrl+F11)

> right click on starter class (main method class)

> run as > Spring Boot application

\*)Note:

a) if we specify wrong object name while reading (ac.getBean)

then Spring container throws Exception:

NoSuchBeanDefinitionException: No bean named 'employeeExport' available

b) Every class which we created in project must be under

either same package name of starter class (Ex: in.nareshit.raghu)

or sub-package of starter class (ex: in.nareshit.raghu.bean.info)

c) If we want to change package for scanning classes

use @ComponentScan("com.app") at starter class.

-------------------------------------------------------------------------

Date :26/06/2020

Spring Boot 7:50 PM

Mr. RAGHU

----------------------------------------------------

YoutbeLive : JUnit Tool : Session-2 9AM

https://www.youtube.com/nareshit

My Videos Play List in Youtube:

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

--------------------------------------------------------------------

Duration: 3 to 4 Months

Timing : 7:50 - 9:20

Spring Boot with Cloud and Microservices

Core Java | Basics of Adv Java

Spring Core | Hibernate | Webservices

Question: Chat box (end of session)

Email : javabyraghu@gmail.com

FB : https://www.facebook.com/groups/thejavatemple/

Admin: Sirkanth

+91- 630 296 8665

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

@Value : This annotation is used to provide data to variable.

=> It supports providing data

a. static value

b. \*\*Properties/Yml Input

c. Expression.

------------------------------------------

@Component("con")

public class DbConn {

@Value("OracleDriver")

private String driver;

@Value("jdbc:oracle")

private String url;

//toString() method

}

\*) In above example @Value is used to provide static data to variables.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. create one Spring Boot Project

> File > new > Spring Starter project > Enter details

name : SpringBoot2ValueEx

Group: in.nareshit.raghu

package: in.nareshit.raghu

> Next > next > finish

2. Create one class under package : in.nareshit.raghu

> right click on 'in.nareshit.raghu' package > new > class

> Enter name : DbConn

> Finish

--code--

package in.nareshit.raghu;

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

import org.springframework.stereotype.Component;

//ctrl+shift+O

@Component("con")

public class DbConn {

@Value("OracleDriver")

private String driver;

@Value("jdbc:oracle")

private String url;

@Override

public String toString() {

return "DbConn [driver=" + driver + ", url=" + url + "]";

}

}

3. Modified starter class

package in.nareshit.raghu;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class SpringBoot2ValueExApplication {

public static void main(String[] args) {

ApplicationContext ac=SpringApplication.run(SpringBoot2ValueExApplication.class, args);

//reading object (objName,className)

DbConn ob=ac.getBean("con",DbConn.class);

System.out.println(ob);

}

}

4. Run Application

> Right click on main class > run as > Spring boot application

\*\*\* Hint : override toString() method to see readable output

> Source menu > Generate toString() option > Finish.

------------------------------------------------------------------------

HardCoding : Provide direct value to a variable inide

.java file is called as HardCoding

\*) We can use application.properties to read values from outside of .java file

which avoid re-compile of our code, but we need to re-start application

(server must be restarted if properties file gets modified)

\*) Syntax to read data: @Value("${key}")

--code--

a. application.properties

app.db=OracleDriver

app.url=jdbc:oracle

b. Class code

--------------

package in.nareshit.raghu;

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

import org.springframework.stereotype.Component;

//ctrl+shift+O

@Component("con")

public class DbConn {

//@Value("${key}")

@Value("${app.db}")

private String driver;

@Value("${app.url}")

private String url;

@Override

public String toString() {

return "DbConn [driver=" + driver + ", url=" + url + "]";

}

}

\*) Run application.

-----------------------------------------------------

Q) How Spring boot loads application.properties file by default?

A) In case of Spring F/w : @PropertySource("classpath:application.properties")

But in Spring Boot this is taken care by annotation: @SpringBootApplication

which is written on top of main class.

In Spring:

XML:

<context:property-placeholder location="classpath:application.properties"/>

Java:

@PropertySource("classpath:application.properties")

Q) If key not exist while reading @Value , then what will happen?

A) Exception:

IllegalArgumentException: Could not resolve placeholder 'app.db' in value "${app.db}"

=> First Spring boot loads : application.properties file

if key is found then, container reads using @Value, else exception.

Q) If we change file name from: application.properties to any other , then what will happen?

A) Spring Boot will not load no other properties file by default.

It loads only application.properties on startup.

\*\* Try to rename application.properties and run

> Right click on file > Refactor > Rename > Enter new name > finish

Q) How can we inform to spring container, to load our properties file?

A) Just write @PropertySource("classpath:dev.properties") at start class level

Date: 29/06/2020

Spring Boot 7:45

Mr. RAGHU

--------------------------------------------------------------------

Admin: Sirkanth

+91- 630 296 8665

hr@nareshit.com

Fee: 5000 /- Full Online batch

Youtube:

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

email : javabyraghu@gmail.com

FB: https://www.facebook.com/groups/thejavatemple/

Duration: 3 to 4 Months

Timing : 7:50 - 9:20

Spring Boot with Cloud and Microservices

Core Java | Basics of Adv Java

Spring Core | Hibernate | Webservices

---------------------------------------------------------------------------------

Spring Boot Introduction:

1. Runners:-

To execute any logic only one time while starting Spring boot application

runner are used.

\*) JdbcRunner, EmailSetupRunner, SecurityConfigRunner..etc

Q) What is the difference between static block and runner?

A) blocks will never take any input and those are called when class is loaded.

Runners support reading input from different sources and

execute when boot is application is running.

Q)Difference between starter and runner?

A) starter is used to start application and create container

Runner : it is used to execute any logic one time give by programmer.

\*\*\* Runners are executed by starter class.

------------------------------------------------------------------------

Types of Runners:(2)

a. CommandLineRunner (I)

b. ApplicationRunner (I)

------------------------------------------------------------------------

\*) Here, to create runner class we need to implement CommandLineRunner and

override run() method. Else use ApplicationRunner and override run() method.

Q) What happens when we run application?

A) Spring container is created with all setup.

Search for classes using basePackage(starter class package: ex: in.nareshit.raghu)

Creates object if it has @Component (or its any equals)

call run method once.

---------------------------------------------------------------------------

Q) Can we use Runner in real time application?

A) Not completely by programmer.

We never write any runner in realtime application

but spring boot internally use Runners to reduce common logic

as onetime setup.

Q) Which scenarios we will use runners in realtime?

A) Test any logic by executing once.

Q) How many runner can be there in application

(internal or given by programmer)?

A) Multiple runner.

-------------------------------------------------------------------

Q) How will the application identifies the runner

when multiple runners are present ?

A) Order annotation/ Order interface

Date : 30/06/2020

Spring Boot 7:45

Mr. RAGHU

javabyraghu@gmail.com

------------------------------------------------------------------------

Admin: Sirkanth

+91- 630 296 8665 (10 AM to 6 PM)

hr@nareshit.com

Fee: 5000 /- Full Online batch

My Videos Links:

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=1

Core Java | Adv Java Basics

Spring Core | Hibernate with JPA | Webservices

Duration : 3 - 4 months

FB : https://www.facebook.com/groups/thejavatemple/

Email: javabyraghu@gmail.com

------------------------------------------------------

JDK : 1.8 / 14

https://www.oracle.com/java/technologies/javase-jdk14-downloads.html

STS : (Spring Tool Suite)

https://spring.io/tools

> choose OS

> Download AS JAR

(It may harm your PC? Do you want to Keep? YES/OK)

> Double click on JAR Download

> Extracted to one folder : C:\Downloads\sts-4.5.1.RELEASE

> Goto folder and click on "SpringToolSuite4" (it will be started)

\*\* (NO INSTALL REQUIRED)

> Change workspace (any folder name to save projects)

> Click on Launch Option

\*\*) Create new project

> File > new > Spring Starter Project

> Enter details

Name : SpringBoot2FirstApp

GroupId: in.nareshit.raghu

Package: in.nareshit.raghu

\*\*\* All other classes must be inside given package(or its sub package)

---------------------------------------------------------------------------

Runners :-

In Spring boot we can define multiple Runner CLASSES, those are executed in order

(Naming order A-Z).

=> We can define multiple classes those implements Runner interface

=> Those are executed in our own order using

a.\*\* @Order Annotation

b. Ordered Interface (Legacy Style)

=> Ex: JdbcRunner, EmailRunner, SecurityRunner..etc

------------------------------------------------------------------------

\*) If we write multiple runners then Spring boot will compare Runner class names

which ever comes first (A-Z order) that is executed first.

\*) We can define our own order by using annotation: @Order.

\*\* Max value runner is executed at last position.

\*) If runners are having same order then again class name sorting is applied.

\*) -ve number runners are executed first, then zero, then +ve number.

--Example Application--------------------------

1. Create Starter Project in STS

> File > new > Spring Starter project > Enter details

Name : SpringBoot2MultipleRunners

Package: in.nareshit.raghu

> finish

2. Create multiple class under package "in.nareshit.raghu" or its subpackage.

package in.nareshit.raghu.runner;

//ctrl+shift+O

import org.springframework.boot.CommandLineRunner;

import org.springframework.core.annotation.Order;

import org.springframework.stereotype.Component;

@Component

@Order(45)

public class EmailRunner implements CommandLineRunner {

@Override

public void run(String... args) throws Exception {

System.out.println("FROM EMAIL RUNNER ");

}

}

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.core.annotation.Order;

import org.springframework.stereotype.Component;

@Component

@Order(10)

public class SecurityRunner implements CommandLineRunner {

@Override

public void run(String... args) throws Exception {

System.out.println("FROM SECURITY RUNNER");

}

}

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.core.annotation.Order;

import org.springframework.stereotype.Component;

@Component

@Order(5)

public class JdbcRunner implements CommandLineRunner {

@Override

public void run(String... args) throws Exception {

System.out.println("FROM JDBC RUNNER");

}

}

3. Run main class (ctrl+F11)

> Right click on starter class(main class) > Run as > Spring boot Application

Output:

FROM JDBC RUNNER

FROM SECURITY RUNNER

FROM EMAIL RUNNER

---Ex#1---------------------------------------------------------------------

@Order Runner name

10 ARunner

-5 BRunner

12 CRunner

---------------------------------------------------------

Output: B, A, C

---Ex#2---------------------------------------------------------------------

@Order Runner name

5 ARunner

5 BRunner

5 CRunner

---------------------------------------------------------

Output: If two or more runners are having same order then

apply name sorting rule

A, B, C

---Ex#3---------------------------------------------------------------------

@Order Runner name

5 ARunner

<VALUE NOT PROVIDED> BRunner

-9 CRunner

---------------------------------------------------------

Output: If @Order is provided without any value then default value is

set to Integer.MAX\_VALUE (2147483647)

C, A, B

---Ex#4---------------------------------------------------------------------

@Order Runner name

56 ARunner

<VALUE NOT PROVIDED> BRunner

-98 CRunner

<VALUE NOT PROVIDED> MRunner

12 KRunner

<VALUE NOT PROVIDED> RRunner

-99 DRunner

57 URunner

Output: D C K A U B M R

Order:

First -ve numbers

Then +ve numbers

Then NO VALUES RUNNRES (Sorting name : A-Z)

-------------------------------------------------------------------------------

Q) Can we restrict executing particular runners only ?

A) yes, Remvoe @Component.

Q) Are we going to write Runner class in real time and give the @Order?

A) Spring boot internally provides runners to reduce onetime setup code by

programmer (AutoConfiguration) also provided with @Order.

Q) Is providing @Order must for Runners?

A) NO. if we do not provider Order then default value is applied to all runners

Integer.MAX\_VALUE.

-----------------------------------------------------------------------

b) Ordered (I) [org.springframework.core]

This interface is given by Spring Core. We can use this interface to provide

execution order for our Runners.

=> It has one abstract method:: getOrder():int

=>\*\* If we provide both @Order (new) and Ordered (I) [Legacy]

then Ordered(I) value is taken.

--Example Runner ------

package in.nareshit.raghu.runner;

//ctrl+shift+O

import org.springframework.boot.CommandLineRunner;

import org.springframework.core.Ordered;

import org.springframework.stereotype.Component;

@Component

public class EmailRunner implements CommandLineRunner,Ordered {

@Override

public void run(String... args) throws Exception {

System.out.println("FROM EMAIL RUNNER ");

}

@Override

public int getOrder() {

return 99;

}

}

------Ex#2-------------------------------------------------------

\*\*\* Order value taken by Spring boot is:\_\_99\_\_\_.

package in.nareshit.raghu.runner;

//ctrl+shift+O

import org.springframework.boot.CommandLineRunner;

import org.springframework.core.Ordered;

import org.springframework.core.annotation.Order;

import org.springframework.stereotype.Component;

@Component

@Order(2)

public class EmailRunner implements CommandLineRunner,Ordered {

@Override

public void run(String... args) throws Exception {

System.out.println("FROM EMAIL RUNNER ");

}

@Override

public int getOrder() {

return 99;

}

}

--------JDK 1.8--------------------------------------------------

Functional Interface: An interface that contains only one abstract methods

It can have multiple default methods and static methods.

Q) How can we validate given one is VALID FUNCTIONAL INTERFACE or not?

A) we can apply @FunctionalInterface on top of our interface then java compiler

will validate all rules of Functional interface.

\*\*\* Adding this annotation is optional

=> We can even add Object class methods as abstract methods in functional interface

Impl class will get logic from Object class.

// valid one.

@FunctionalInterface

interface Sample {

void show();

//object class methods

String toString();

boolean equals(Object obj);

int hashCode();

}

//vaild one

@FunctionalInterface

interface Sample {

void show();

public default void showA() {

System.out.println("WELCOME-A");

}

public static void showB() {

System.out.println("WELCOME-B");

}

}

//invaild

@FunctionalInterface

interface Sample {

void show();

void showA();

}

Q) A Method has no body ?? it is called as What?

A) abstract or native.

abstract = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

native = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Spring Boot Introduction

Spring Boot work flow and Tools

Spring Core basics and Runner

Date : 01/07/2020

Spring Boot 7:50 PM

Mr. RAGHU (Naresh IT, Hyderabad)

------------------------------------------------------

Admin: Sirkanth

+91- 630 296 8665 (10 AM to 6 PM)

hr@nareshit.com

Fee: 5000 /- Full Online batch

My Videos Links:

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=1

Core Java | Adv Java Basics

Spring Core | Hibernate with JPA | Webservices

Duration : 3 - 4 months

FB : https://www.facebook.com/groups/thejavatemple/

Email: javabyraghu@gmail.com

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

Functional Interface:

=> An interface that contains one abstract method.

=> It can even have multiple default and static methods.

=> @FunctionalInterface (it is optional)

if we add this annotation over interface, then it will inform to Java Compiler

"Verify Rules of Functional Interface"

--Examples--------------------

interface Sample { //Vaild

void show();

}

\*\* Including Parent interface, current interface should contain only one abstract method

interface Test extends Sample { //VALID

}

------------------------------------

interface Sample { //Vaild

void show();

}

interface Data { //Vaild

void find();

}

//Zero Abstract methods (no parenr interface)

interface Model { } //Invaild Functional interface

//one abstract method from sample and zero from Mode = one abstract method

interface Chart extends Model,Sample { } //Vaild

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

\*\*\* Multiple Inheritance is possible in Java for Interfaces.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Lambda Expressions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This is used to write logic (code) in short format.

=> In simple it is equals to "WRITING IMPL CLASS + OBJECT CREATION"

=> Syntax:

Interface ob = (methodParams) -> { method Body } ;

=> We can write Lambda Expressions only For Functional interface

--Examples-------------------

interface Maths {

int add(int a,int b);

}

Lambda Expression

#1

Maths om = (int a,int b) -> {

int c= 0;

c = a+b;

return c;

};

#2 Logic in Short Format

Maths om = (int a,int b) -> {

return a+b;

};

#3 If braces (method body) contains only one line then {} are optional

Maths om = (int a,int b) -> return a+b; //Invaild

\*\*\* If {} are not present then remove return keyword.

Maths om = (int a,int b) -> a+b; //Vaild

#4 DataTypes of variables/Params are optional

Maths om = (a,b) -> a+b; //Vaild

#5. Even We can change variable/param names

Maths om = (x,y) -> x+y; //Vaild

\*\* If no.of Params count is one then even () symbol is optional

---------------------------

interface Message {

void show(String name);

}

Lmabda Expressions:

Message mob = (String name) -> { sysout(name); }; //Vaild

Message mob = name -> sysout(name); //Vaild

--------------------------

interface Format {

void show(int code); //print "your code is" + code

}

Lmabda Expressions:

Format fm = (int code) -> { sysout("your code is" + code); };

Format fm = c -> sysout("your code is" + c) ;

----------------------------

interface Process {

double count(double a,double b,double c); //print => return a+b+c

}

Lmabda Expressions:

Process pob = (double a,double b,double c) -> { return a+b+c; }

Process pob = (a,b,c) -> a+b+c;

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

public interface CommandLineRunner { //print "welcome to Boot"

void run(String... args) throws Exception;

}

Lmabda Expressions:

CommandLineRunner clr = (String... args) -> { sysout("welcome to Boot"); }

CommandLineRunner clr = args -> sysout("welcome to Boot");

======Spring Java based Configuration==================================================

1. Define one public class with any name

2. Apply @Configuration over class

3. Define one method for One object

Return Type must be class or its super type(interface)..

public <class/interface> <objectName>(){

//logic..

}

4. Apply @Bean over method (that behaves as Object in container)

--class--

class Employee { }

--Spring Java Config code--

@Configuration //Current class is converted as Java Config File

public class AppConfig {

@Bean //Object created in container

public Employee eob() {

Employee e = new Employee();

return e;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring Boot Runner using Lambda + Java Config \*\*\*\*\*\*\*\*\*\*\*\*\*\*

package in.nareshit.raghu.config;

import org.springframework.boot.CommandLineRunner;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class AppConfig {

@Bean

public CommandLineRunner cob() {

CommandLineRunner clr = (String... args) -> {

System.out.println("Welcome to CLR + Java Config + Lambda Exp");

};

return clr;

}

}

Short Code:---------

@Configuration

public class AppConfig {

@Bean

public CommandLineRunner cob() {

return (String... args) -> {

System.out.println("Welcome to CLR + Java Config + Lambda Exp");

};

}

}

Very Short Format:--

@Configuration

public class AppConfig {

@Bean

public CommandLineRunner cob() {

return args ->

System.out.println("Welcome to CLR + Java Config + Lambda Exp");

}

}

Q) What is diff b/w @Component and @Bean?

A) @Component for Our class (not for pre-defined) it is annotation Config

Suports only one object

@Bean for Java Config (Supports both Pre-defined+ Programmer class)

even we careate multiple objects.

Q) What is diff b/w @Configuration and @Bean?

@Configuration: Make current class as Not a normal class, it is input to Spring container

that says to container "CREATE OBJECTS BY USING BELOW METHODS".

@Bean : Create object in container, so that it is reusable.

Q) When to use Java and Annotation Config?

A) Annotation Config is faster but it is appliable for only for Our class

it will not work for pre-defined classes

Java Config support any type class(pre-defined/programmer class)

but bit slow compared to Annotation Config.

Date : 02/07/2020

Spring Boot 7:45 PM

Mr. RAGHU

---------------------------------------------------------

Lambda Expression:

Interface ob = (MethodParams) -> { methodBody; }

Spring Boot Runner:

+ CommandLineRunner

+ run(String... args):void

Lambda Expression for Runner

CommandLineRunner clr = (args) -> {

Sysout("From Runner + lambda Exp + Java Config");

}

----Spring Core Java Config-----------------------------------------------

#1. Define one public class with any name

#2. Apply one Annotation @Configuration

#3. Define one method for one object

public <Class/Interface> <object>(){

//logic..

}

#4. Apply @Bean over method.

\*) @Configuration converts our class(normal class) into Spring container input class

\*) @Bean converts our method (normal method) into One object in container.

===============Code============================================

#1. Create Spring Starter Project

> File > new >Spring Starter project

> Enter details

Name : SpringBoot2RunnerusingLambda

GroupId: in.nareshit.raghu

Package: in.nareshit.raghu

> Next > Next > Finish

#2. Create one Java Config class under src/main/java

> right click on src/main/java > new > class

> Enter details

name : AppConfig

package: in.nareshit.raghu.config

> Next > Finish

#3. Write code for AppConfig

package in.nareshit.raghu.config;

import org.springframework.boot.CommandLineRunner;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class AppConfig {

// no.of methods = no.of objects

@Bean

public CommandLineRunner cob() {

//Interface ob = (param) -> { body; }

/\*

CommandLineRunner clr = (args) -> {

System.out.println("FROM CLR + JAVA CONFIG + LAMBDA");

};

return clr ;

\*/

return args -> System.out.println("FROM CLR + JAVA CONFIG + LAMBDA");

}

}

\*) Run main method class (Starter class).

=> Object created for CommandLineRunner Impl class using Lambda Expression

=> run() method is called by Spring Boot application.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Working with Spring Boot Properties file

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--------------------------------------------------------------------------

Spring Boot supports reading data from different sources like UI, Database, properties

System variables..etc

\*) properties : storing data in key=value format

\*) This is used as providing input to Spring Container

this data is given to Objects.

\*) Spring Boot loads "application.properties" as default file

(we can even change file name and file location later).

\*) properties file can have pre-defined keys and programmer defined keys.

\*) To view all pre-defined keys in Spring boot, Link is:

https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html

--Or else goto application.properties file and just type ctrl+space

=> here we have 2 different ways of reading data from Properties file

a. @Value (Spring f/w)

=> This is used to read one key value at a time into one variable

b. @ConfigurationProperties (Spring Boot f/w)

=> This is used to load multiple keys at a time into multiple variable

Date : 03/07/2020

Spring Boot 7:50 PM

Mr. RAGHU

--------------------------------------------

For Payments, Please contact

ADMIN : Srikanth 6302968665

Youtube:

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

FB:

https://www.facebook.com/groups/thejavatemple/

email:

javabyraghu@gmail.com

3 setup files

1. Input file (properties/yml)

2. Build Information File (Maven-pom.xml, Gradle-build.gradle)

3. Starter class : Start app + Create container

------1. Input file (properties/yml)---------------------------

Properties : Storing data in key=value format

Properties are used to provide input to Spring container object

which behaves like one time input/static data.

It holds details like(example)

=> Database connection keys

=> Connection Pooling keys

=> Logging Keys

=> Security Keys

=> Message Keys

=> Web Applications

=> Server keys ..etc

\*) Key can either pre-define[99.99%] or programmer defined key(your own key) [0.01%]

Pre-Defined keys in Spring Boot.

https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html

\*) One key in properties file = = one variable in our class.

\*) To read properties key into application use:

a) @Value [Given by Spring f/w]

b)\*\*\*\* @ConfigurationProperties [Given by Spring boot]

------ @Value (Spring Based Properties Reading) --------------------

To read one key value from properties file into one variable in our class use @Value.

Syntax: @Value("${key}") [On top of variable in class]

=> At a time we can read one key value into one variable.

=> Ex: We have 10 keys then 10 times we should write @Value on top of 10 variable.

=> If key is not present in properties file and we are trying to read using

@Value() then it gives Exception:

IllegalArgumentException: Could not resolve placeholder 'my.app.id' in value "${my.app.id}"

---Eclipse/STS keys-----------------------------------------------------------------

ctrl+shift+T (enter file name) To open any pre-define java file

F3 Goto Code from usage

======Example===================================================

1. Create Spring Starter Project

> File > new > Spring Starter Project > Enter Details

Name : SpringBoot2ValueExmaple

GroupId: in.nareshit.raghu

package: in.nareshit.raghu

> Next > Next > Finish

2. application.properties

# Comment Line

# Allowed Symbols dot(.) dash(-) and underscore(\_)

my.app.id=10

my.app.code=NIT

my.app.version=1.3

3. Runner class

package in.nareshit.raghu.runner;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

//ctrl+shift+O

@Component

public class DataReadRunner implements CommandLineRunner {

//@Value("${key}")

@Value("${my.app.id}")

private Integer appId;

@Value("${my.app.code}")

private String appCode;

@Value("${my.app.version}")

private Double appVer;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

//System.out.println(this.toString());

}

//Source > Generate toString()

//alt+shift+S, S>OK

@Override

public String toString() {

return "DataReadRunner [appId=" + appId + ", appCode=" + appCode + ", appVer=" + appVer + "]";

}

}

Q) How @Value works internally in Spring?

A)

i. First application.properties file is loaded using code like

@PropertySource("classpath:application.properties")

ii. Now, Spring container creates one MEMORY "Environment" (key-val pairs)

that holds all key-vals given by Properties file

iii. We can read those values into variable using Syntax

@Value("${key}")

iv. @Value will search for key in Environment memory

v. If found then value is updated into container object

----------------------------------------------------------------

Q) Can we modify/rename file name from : application.properties

to any other in Spring Boot?

A) YES, But it is not loaded by Spring Boot by default.

Spring boot checks by default application.properties

under location src/main/resources.

-> If we want to load other files

i) @PropertySource at starter class

Ex: @PropertySource("classpath:abcd.properties")

\*\* classpath = src/main/resource folder

file:/ = project folder

ii) Spring Boot Profiles\*\*

Q) How to explain Lambda Expression?

A) Lambda Expression are functional Programming in Java

that provides short coding for Functional interface

that creates implementation and object.

Syntax: interface ob = (params) -> { body} ;

Q) What are valid location of Properties files in Spring?

How to load them?

A) Valid location for Properties are two

i. src/main/resource (classpath:)

@PropertySource("classpath:application.properties")

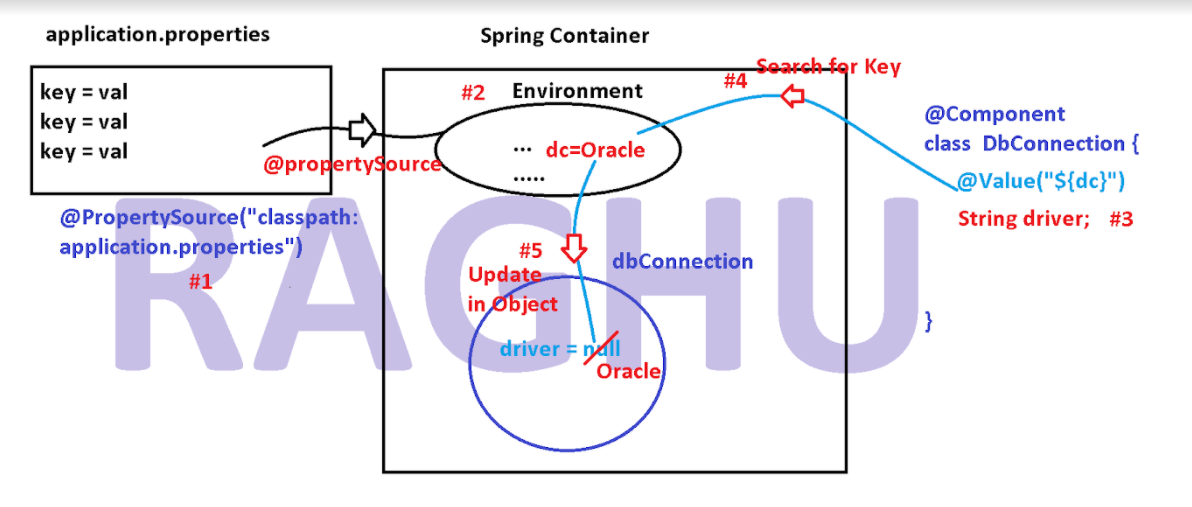
ii. Inside project folder ( file:/)

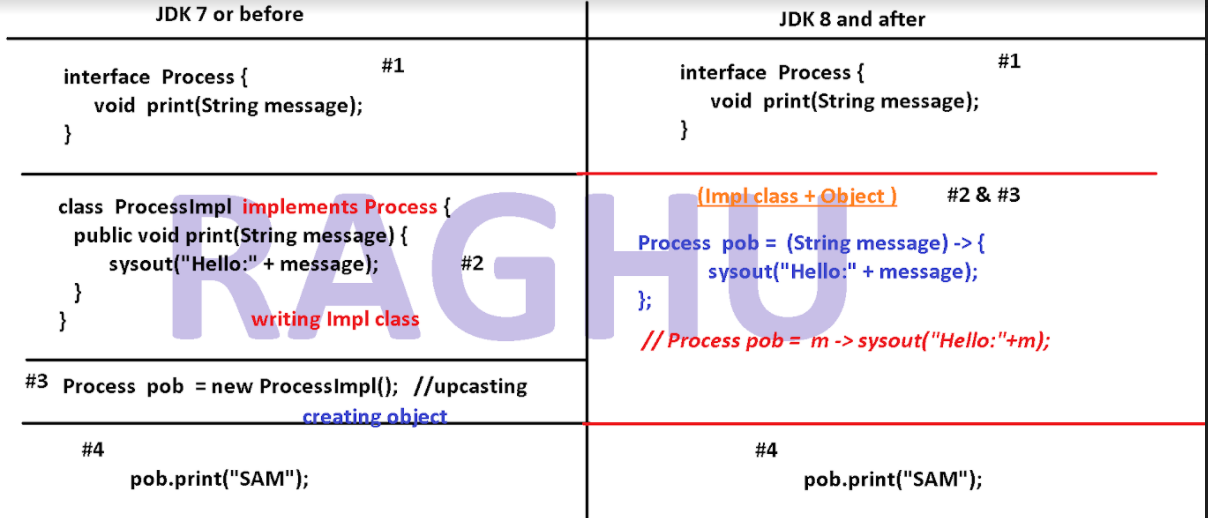
@PropertySource("file:/application.properties")

=> There no Default location for standalone application

=> Defaut location is for web applicatin are 'webapp' (root folder)

@PropertySource("application.properties")





Date : 06/07/2020

Spring Boot 7:50 PM

Mr. RAGHU

----------------------------------------------------

Spring Boot 3 setup file

1. Starter class (main method class)

2. Input file (application.properties | application.yml)

3. build information file (pom.xml/build.gradle)

-------- YAML (YAMalian Language)--------------------------------------

YAML (file extension is .yml) is another concept used in Spring boot

to provide key=val pairs as input to Spring container.

=> To write a key we can use symbols like dot(.), dash(-) and underscore(\_).

=> Here dot(.) changes level in key.

=> Compared to properties file, yml removes duplicate levels in keys.

=> Replace symbols dot(.) with colon (:) and

change to next line for new level

=> Do not write same level again at new line, start with next level data.=

=> Space count must match with same level data.

=> Symbol equals(=) should be replace with colon with one space (:<space>)

-------application.properties---------

my.app.id=10

my.app.code=NIT

my.app.version=3.3

-------application.yml---------

my:

app:

id: 10

code: NIT

version: 3.3

--------------------------------

-------application.properties---------

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=nit

-------application.yml---------

spring:

data:

mongodb:

host: localhost

port: 27107

database: nit

-------------------------------

List/Array/Set

-------application.properties---------

spring.data.user[0]=nit

spring.data.user[1]=abc

spring.data.user[2]=xyz

Use Symbol - (indicates index number)

-------application.yml---------

spring:

data:

user:

- nit

- abc

- xyz

-------------------------------

---application.properties-----------

my.app.version=3.4

my.one.product.type=GLOABL

my.app.code=NIT

my.one.product.access=true

my.app.format.style=NEW

my.app.format.model=false

---application.yml-----------

my:

app:

version: 3.4

code: NIT

format:

style: NEW

model: false

one:

product:

type: GLOBAL

access: true

-------or----------------

my:

one:

product:

type: GLOBAL

access: true

app:

version: 3.4

code: NIT

format:

style: NEW

model: false

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

---application.yml------------

server:

port: 9966

servlet:

context-path: /product-curd-app

spring:

datasource:

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://localhost:3306/boot

username: root

password: root

jpa:

show-sql: true

hibernate:

ddl-auto: update

database-platform: org.hibernate.dialect.MySQL55Dialect

------------------------------

---application.properties------------

server.port=9966

server.servlet.context-path=/product-curd-app

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/boot

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

spring.jpa.database-platform=org.hibernate.dialect.MySQL55Dialect

--------------------------------------------------

--------------application.properties-----------------

my.app.id=10

my.app.code=NIT

one.test.code=A

one.test.model=NEW

fmt.big.format=true

fmt.big.stage=ACCEPT

------------------------

my:

app:

id: 10

code: NIT

one:

test:

code: A

model: NEW

fmt:

big:

format: true

stage: ACCEPT

---------------------------------------------

Q) What is the difference between properties and yml file?

Which one should be used in coding?

A) yml is a new concept added in Boot to read properties file

yml is a user-readable format document for inputs

yml reduces duplicate level and make easy reading incase of complex or more no.of keys

.yml file also coverted to .properties file at runtime (internally)

Snake YAML API converts data into proeprtie format.

\*\*\* Use YML file if no.of keys in properties are more, else properties only good.

\*\*\* No.of keys are more then use YML

else use properties.

Q) Can we place both yml and properties file in project?

A) YES, then Properties file choosen with order. YML file is ignored.

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

7:50 PM - 9:30 PM

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

https://www.facebook.com/groups/thejavatemple/

javabyraghu@gmail.com

Backup: 10th and 11th

1 Runners

2 properties @value/@COnfig

3 yml

Duration : 3-4 months

Core Java + Spring Core + Spring Boot + Oracle DB + \*\*Angular/React UI

Admin : Srikanth : +91 630 2968 665

hr@nareshit.com

Date : 8/7/2020

Spring Boot 7:50PM

Mr. Raghu

javabyraghu@gmail.com

------------------------------------

Admin : Srikanth 630 296 8665.

Project Lombok:-

It is an Open Source Java API, given by ProjectLombok Team

this is used to generate code for

a. constrcutors

b. setters

c. getters

d. toString method

e. equals and hashcode methods.

=> It has provided annotations, those must be applied on top of

your class. So, code is generated.

=> If you modify your code like adding/removing/renaming any

variables, Lombok will takecare of new code generation.

--------Enable ProjectLombok in STS-------------------------

1. Create one Spring Starter project with Lombok Dependency

> File > new > Spring Starter Project

> Enter project name

Ex: SpringBoot2LombokSetupApp

>next

> Search 'lombok' > choose lombok > finish

2. Write one class and apply any one Lombok annotation.

--Ex code--

package in.nareshit.raghu.model;

import lombok.Setter;

@Setter

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

3. Close STS

4. Goto below location and double click on "lombok-1.18.12"

Location:

C:\Users\<username>\.m2\repository\org\projectlombok\lombok\1.18.12

It show one Lomboksetup screen, Still if you are not able to see??

\*)Open command window for above location

> shift+ right click on above location

> open command window here

> java -jar lombok-1.18.12.jar

\*) Click on Sepcify location

\*) Select STS Download Location

ex: F:\Downloads\sts-4.6.2.RELEASE

\*) click on Install/Update Button

---------------------------------------------------

Lombok Annotations:

@NoArgsConstructor : This annotation generates Default constrcutor

for our class.

@AllArgsConstructor : This annotation generates all param constrcutor

for our class.

@Setter : This annotation generates setter method for all variables

@Getter : This annotation generates getter method for all variables

@ToString : This annotation generates toString method for all variables

@EqualsAndHashCode : : This annotation generates hashCode and equals method for all variables

------------------------------------------------------------

Q) Who will provide default constrcutor ? and when?

A) Java compiler provides default constrcutor while compling

if class has zero(no) constructors.

Ex:

A.java

class A {}

>> javac A.java

--A.class---

class A{

A(){

super();

}

}

Q) What is the use of toString() method?

where it is exist? where it is used?

A) toString() method is used to convert object

data into String format.

This method exist in Object(java.lang) class

we can override this method, it gives default logic

className@HashCodeInHexaDecimalFormat.

When we try to print object on console then

it is internally called by println method

we can override this method in our class,

for change of logic.

Q) what is the use of equals() And hashCode() methods ??

A) To compare data of two objects use equals() method

that even uses internally hashCode() also

hashCode() : this is used to provide identity

for an object in Heap.

Q) What is the difference between equals() method

and == operator?

A) In our class if we do not override equals() method,

then logic is given by Object class that internally

uses == operator, hence both are same.

If you want to compare data(objects variables)

then override method equals() in your class.

Number Systems

Decimal Number : base 10 : 0-9

Octal Number : base 8 : 0-7

Binary Number: base 2 : 0,1

HexaDecimal Number: base 16 : 0-9 A,B,C,D,E,F

Q) can two different objects have same hashcode?

A) YES possible.

If you class has not overriden hashCode() method

then it is not possible, if we override hashcode()

method in your class, then

IF TWO OBJECTS HAVING SAME DATA GETS SAME HASHCODE.

Q) What is default hashing alog used by java for

hashCode() when we override?

A) PRIME HASHING.

We has 7 types of Hashing.

Find All\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

---code------------------------------

package in.nareshit.raghu.model;

import lombok.AllArgsConstructor;

import lombok.EqualsAndHashCode;

import lombok.NoArgsConstructor;

import lombok.ToString;

@NoArgsConstructor

@AllArgsConstructor

@EqualsAndHashCode

@ToString

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

--------------------------------------------

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

@Component

public class EmployeeRunner implements CommandLineRunner {

@Override

public void run(String... args) throws Exception {

Employee e1 = new Employee(10, "A", 3.2);

Employee e2 = new Employee(10, "C", 3.2);

System.out.println(e1);

System.out.println(e1.toString());

System.out.println(e1.equals(e2));

System.out.println(e1.hashCode());

System.out.println(e2.hashCode());

String s="RAGHU"; //77735685

System.out.println(s.hashCode());

}

}

Date : 9/7/2020

Mr. Raghu

Spring Boot 7:50PM

------------------------------------------------------

Lombok Annotations:-

1. @NoArgsConstrcutor: This annotation is used to generate one

default constrcutor and modifies source code, before giving

to java compiler.

Ex:

@NoArgsConstructor

public class Employee {

}

Generated class:

public class Employee {

public Employee() {

super(); //which calls super class constrcutor;

}

}

Q) Can we add our own customzed code inside lombok generated code?

A) As of now NO.

Q) Is lombok required at runtime?

A) No. lombok required until source code compile.

-----------------------------------------------------------

2. @AllArgsConstructor:

This annotation generates default contructor if no variables found.

if variables exist then param constructor is generated.

Ex:

@AllArgsConstructor

public class Employee {

}

Generated class:

public class Employee {

public Employee() {

super(); //which calls super class constrcutor;

}

}

Ex#2

@AllArgsConstructor

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

Generated class:

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee(Integer empId,String empName,Double empSal) {

super(); //which calls super class constrcutor;

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

}

this: current class object reference.

Ex#3

@NoArgsConstructor

@AllArgsConstructor

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

Code Generated :

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee() {

super(); //which calls super class constrcutor;

}

public Employee(Integer empId,String empName,Double empSal) {

super(); //which calls super class constrcutor;

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

}

\*\*\* Ex#4 \*\*\*

@NoArgsConstructor

@AllArgsConstructor

public class Employee {

}

Code Generated :

public class Employee {

public Employee() {}

public Employee() {}

}

Lombok Generated two default const. as per its design,

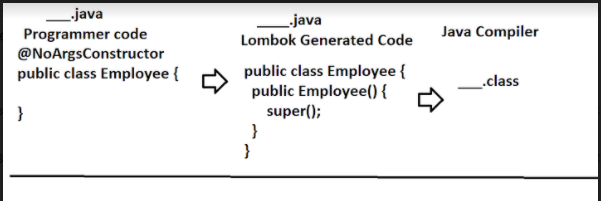
but java compiler gives error while compiling

saying that 'Duplicate method Employee() inside Employee'

Q) If constructor is a method, what is return type?

A) Return Type is same class type.

Employee e = new Employee();



Date : 13/07/2020

Spring Boot 7:50 PM

Mr. RAGHU

--------------------

Lombok : ProjectLombok

Generate code in model class

Constructor

Set/Get methods

toString

Equals and hashCode

--------Ex--------------------------------------------

package in.nareshit.raghu.model;

//ctrl+shift+O

import lombok.AllArgsConstructor;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

import lombok.ToString;

@NoArgsConstructor//def const

@AllArgsConstructor//3 param const.

@ToString //toString method

@Setter //set methods

@Getter //get methods

@EqualsAndHashCode // equals() + hashCode()

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

--------------------------------------------------

@RequiredArgsConstructor :

This annotation is used to provide selected variables as

parameters inside constrcutor, ie Parameterized constrcutor.

Variable should be selected using @NonNull annotation.

If no variable has this annotation, then zero variables selected

So, it generates default constructor.

--Ex#1-----------

package in.nareshit.raghu.model;

import lombok.NonNull;

import lombok.RequiredArgsConstructor;

@RequiredArgsConstructor

public class Employee {

@NonNull

private Integer empId;

private String empName;

private Double empSal;

}

Generated code:

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee(Integer empId) {

super();

this.empId=empId;

}

}

----Ex#2------------------------------

@RequiredArgsConstructor

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

Generated code:

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee() {

super();

}

}

--Ex#3----------------

@RequiredArgsConstructor

@AllArgsConstructor

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

Generated code:

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee() {

super();

}

public Employee(Integer empId,String empName,Double empSal) {

super();

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

}

--------Ex#4--------------------

@RequiredArgsConstructor

@AllArgsConstructor

public class Employee {

@NonNull

private Integer empId;

private String empName;

private Double empSal;

}

Generated code

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee(Integer empId) {

super();

this.empId=empId;

}

public Employee(Integer empId,String empName,Double empSal) {

super();

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

}

------Ex#5------------------------------

@RequiredArgsConstructor

@AllArgsConstructor

public class Employee {

//zero variables

}

Generated code: //Compiler error : duplicate method Employee() in Employee class

public class Employee {

public Employee(){

super();

}

public Employee(){

super();

}

}

--Ex#5---------------

@NoArgsConstructor

@RequiredArgsConstructor

@AllArgsConstructor

public class Employee {

@NonNull

private Integer empId;

private String empName;

private Double empSal;

}

Generated code:

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee() {

super();

}

public Employee(Integer empId) {

super();

this.empId=empId;

}

public Employee(Integer empId,String empName,Double empSal) {

super();

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

}

---Ex#6-------------------------

@NoArgsConstructor

@RequiredArgsConstructor

@AllArgsConstructor

public class Employee {

}

Generated Code: //Java compiler Error.duplicate method Employee()

public class Employee {

public Employee() {

super();

}

public Employee() {

super();

}

public Employee() {

super();

}

}

Q) What is use of @NonNull?

A) This annotation is used by @RequiredArgsConstructor,

to indicate that variable is selected for param const creation.

Date : 14/07/2020

Spring Boot 7:50 PM

Mr. Raghu

-------------------------------------------

@Data : Lombok Annotation

This annotation is used to generate multiple things in model

class. Those are:

a. Set methods

b. Get methods

c. ToString method

d. equals and hashcode method

f. Required args constructor.

=> In simple :

@Data = @Getter + @Setter + @ToString + @EqualsAndHashCode

+ @RequiredArgsConstructor

---Ex#1-------------------

@Data

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

Generated code:

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee() {} //default constructor

//set, get methods

//toString()

//equals and hashcode...

}

\*\* Note: @Data internally follows @RequiredArgsConstructor.

So, if variables having @NonNull then param-constructor is

generated else default const. is generated.

---Ex#2----------------------

@Data

public class Employee {

@NonNull

private Integer empId;

@NonNull

private String empName;

private Double empSal;

}

Generated code:

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee(Integer empId,String empName) {

super();

this.empId= empId;

this.empName=empName;

}

//set, get methods

//toString()

//equals and hashcode...

}

Q) When java compiler gives default constructor?

A) If no constrcutor is given in source code.

\*\*\* Note: @Data uses @RequiredArgsConstructor internally,

if there is no @\_\_ArgsConstructor provided by programmer.

---Ex#3\*\*\*\*--------------------------------------------------------

@Data

@AllArgsConstructor

public class Employee {

@NonNull

private Integer empId;

private String empName;

private Double empSal;

}

Generated code:

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee(Integer empId,String empName,Double empSal){

...;

}

}

---Ex#4\*\*----------------------

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Employee {

@NonNull

private Integer empId;

@NonNull

private String empName;

private Double empSal;

}

Generated Code:

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee() {..}

public Employee(Integer empId,String empName,Double empSal){

...;

}

}

---Ex#5\*\*-------------

@Data

@NoArgsConstructor

@AllArgsConstructor

@RequiredArgsConstructor

public class Employee {

@NonNull

private Integer empId;

private String empName;

private Double empSal;

}

Generated Code:

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

public Employee() {..}

public Employee(Integer empId) {..}

public Employee(Integer empId,String empName,Double empSal){

...;

}

}

Q) How can we add our own logic in generated set/get methods

or constrcutors in Lombok?

A)

Q) Can we define private/protected specifier at class?

A) No. But inner class supports them.

Outer class or class supports only no specifier or public.

But constructor can be any specifier.

Q)\*\* How to define multiple constructors (4th or more const )

using Lombok?

A) No possible. Define manually.

Q) What is the default access specifier level provided for

constrcutor and how to chnage it?

A) Lombok Provides 'public' Access Specifier level.

To change it use enum : 'AccessLevel'.

Ex:

@NoArgsConstructor(access = AccessLevel.PRIVATE)

@AllArgsConstructor(access = AccessLevel.PROTECTED)

@RequiredArgsConstructor(access = AccessLevel.PACKAGE) //default

public class Employee {

..

}

Q) Can we do constrcutor overload and override in Java?

A) Overload is possible. But not override (bcoz child class

constructor ie method name will get chnaged to child class

name).

Q) Is this code valid?

class A{

A() { this(10); }

A(int b){ this(10,20); }

A(int b,int c){ this(); }

}

A) It will give compile time error, as they are called in loop.

At least one constrcutor should have super();

super() will break loop here.

Q) what is super()?

A) It will call super class constructor.

Q) class A{

A(double a) { ..} #1

A(Integer b) {...} #2

}

A ob = new A(10);

A) \_\_\_\_\_\_.

Date : 15/07/2020

Spring Boot 7:50PM

Mr. RAGHU

--------------------------------------------------------

hr@nareshit.com

javabyraghu@gmail.com

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

Environment for Application:-

A compilete setup to run our application is called as Environment.

Environment contains = code in executable format + Server

+ Database + Jars ..etc

Few Examples for Environments are:

DevEnvironment, ProdutionEnvironment, QaEnvironment, UATEnvironment..etc

Q) If we move Spring boot application from

one Environment to another Environment

then what changes are required in Project?

A) Code file Remains same,

only properties file (or) yml file gets changed.

Because Database connection keys, server keys, log keys,

security properties..etc all are provided inside

application.properties (or) application.yml

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Profiles in Spring Boot

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Handling different properties/yml files based on Environment

specific is called as Profiles.

It means, if we move our application from one Environment to

another Environment, then we should load another properties/yml

file insted of actual one.

\*\* DO NOT DELETE/MODIFY EXISTED PROPERTIES FILE, PROVIDE NEW ONE.

\*\* Properties File naming rule:

application-[profilename].properties

ex: names like realtime,

application-qa.properties [ profile name - qa]

application-prodhtc.properties

application-uatnit.properties

..etc

Invaild names:

prod-application.properties

myapp-prod.properties

app-hibernate.properties

hibernate-jpa.properties

Q) What is Command Line Args and VM Args in Java? Write one Example?

A) Command Line Args : Passing input to main() method at runtime

VM Args : Passing input to all applications running at JVM

it is also called as System Argument.

Syntax for System Args : -Dkey=val

To read in application: System.getProperty("key")

--------------------------------------------------------------

\*\*\* If we did not specify any profile then default profile is

set to active. ie application.properties is loaded.

\*\*\* To specify current profile name use key

-Dspring.profiles.active=<profileName>

Ex: -Dspring.profiles.active=qa

then Spring boot will search for application-qa.properties

if it is not found then application.properties are loaded

to avoid errors/exceptions.

\*\* Even if a key-val is not present in current profile then

it can be taken from default profile application.properties

--Example-------------------

#1. Create one Spring Starter Project

with depenency : lombok

#2. create multiple proerties files

application.properties

app.env=DEV

app.db=Oracle

application-qa.proeprties

app.env=QA

app.db=MySQL

application-prod.properties

app.env=PROD

app.db=DB2

#3. Runner class

package in.nareshit.raghu.runner;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import lombok.ToString;

@Component

@ToString

public class MyProfileTestRunner implements CommandLineRunner {

@Value("${app.env}")

private String env;

@Value("${app.db}")

private String db;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

}

}

Q)How to active any one Profile using STS in Spring Boot?

A) Right click on Project

> Run As > Run Configuration

> Arguments

> Choose VM Arguments

> Enter

-Dspring.profiles.active=qa

> Apply and Run

Q) Can we write fully qualified name ?

-Dspring.profiles.active =application-qa.properties

A) NO. only above Syntax

-Dspring.profiles.active=<profileName>

Q) What happend if key is not present in current profile?

A) Boot will search in application.properties (default profile)

Q) What happend if key is not present in any profile/properties file?

A) If we use @Value and key not found then Exception

else if we use @ConfigurationProperties then null.

--Runner Using ConfigProperties--

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import lombok.Data;

@Component

@Data

@ConfigurationProperties("app")

public class MyProfileTestRunner implements CommandLineRunner {

private String env;

private String db;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

}

}

Q) Can we write both properties and yml files at a time

then which one is loaded?

A) YES. properties file is loaded.

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

Profiles using YAML Files

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

Syntax:

application-[profilename].yml

#1. Create one Spring Starter Project with Lombok

#2. create multiple YAML files

application.yml

app:

env: DEV

db: Oracle

application-qa.yml

app:

db: MySQL

env: QA

application-prod.yml

app:

db: DB2

env: PROD

#3. Runner class

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import lombok.Data;

@Component

@Data

@ConfigurationProperties("app")

public class MyProfileTestRunner implements CommandLineRunner {

private String env;

private String db;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

}

}

=> Run Configuration > -Dspring.profiles.active=prod

\*\*\*\* WE CAN DEFINE ONE YML FILE THAT HOLDS ALL PROFILES

SEPARATED USING 3 DASH SYMBOLS (---).

\*\* To specify profile name for that area

use

spring:

profiles: <profileNameForThisArea>

==Ex#3================================

1. Create one Spring Starter Project with Lombok

#2. Rename application.properties to application.yml

app:

env: DEV

db: Oracle

---

spring:

profiles: qa

app:

db: MySQL

env: QA

---

spring:

profiles: prod

app:

db: DB2

env: PROD

#3 Runner class

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import lombok.Data;

@Component

@Data

@ConfigurationProperties("app")

public class MyProfileTestRunner implements CommandLineRunner {

private String env;

private String db;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

}

}

Q) Which symbol is used to define single YAML file with multiple

profiles?

A) Symbol 3 dashes(---)

Q) In above case how to specify current area profile name?

A) spring:

profiles: <name>

-------------------------------------------------------------

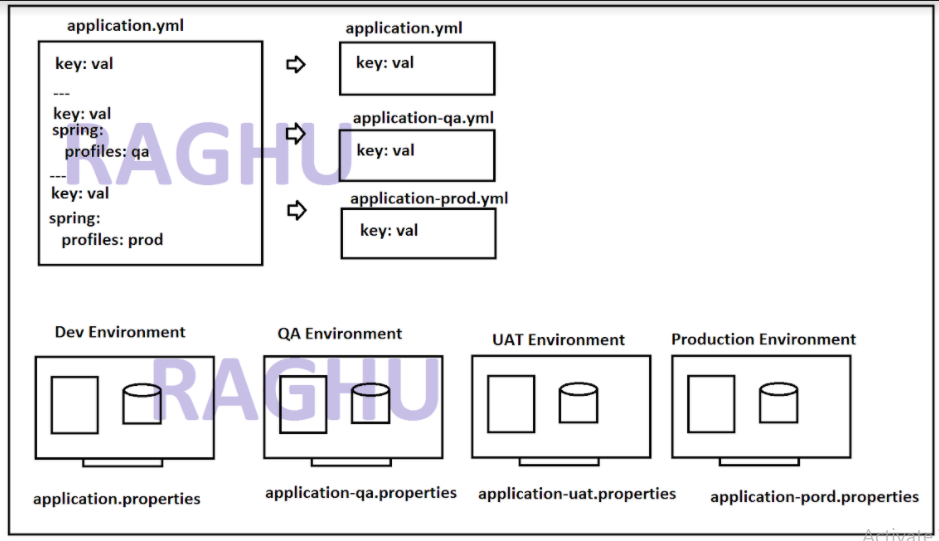
Q) What is Child Profiles?

Q) What is the use of @Profiles annotation?

Q) What is profile fallback?

A) If key is not present in current profile, then read from

default profile.



Date : 16/07/2020

Spring Boot 7:50 PM

Mr. RAGHU

---------------------------------------------------------------------

https://www.youtube.com/playlist?list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

https://docs.jboss.org/hibernate/orm/5.4/quickstart/html\_single/

https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

https://spring.io/guides/

-------------------------------------------------------------------------

https://w3.cs.jmu.edu/spragunr/CS240\_F12/ConciseNotes.pdf

https://docs.oracle.com/javase/tutorial/

Register and Solve Problems

https://www.hackerrank.com/

https://practice.geeksforgeeks.org/courses/#freeCourses

Child Profiles: [Profiles Include]

----------------------------------

If one properties file having multiple keys

(email, database, security, jpa, connection pool, cloud server..etc)

then we can divide them into multiple proeprties files (child profiles)

and link them together finally.

=> This one is easy to read and maintain only

=> There is no performance difference in before and now.

=> If same key is present in multiple properties in case of child profiles

then last loaded combination is taken by Spring container.

==============Example================================================

#1. Create one Spring Boot Starter Project with Dependency: Lombok

#2. create below properties files

application.properties

spring.profiles.active=prod

application-prod.properties

app.info.title=NIT

app.info.ver=1.1

spring.profiles.include=prodmail,proddb

application-proddb.properties

app.db.driver=Oracle

app.db.url=jdbc-orcl

#app.info.ver=3.3

application-prodmail.properties

app.mail.host=Gmail

app.mail.port=885

#app.info.ver=2.2

#3. Runner class

package in.nareshit.raghu;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import lombok.ToString;

@Component

@ToString

public class DataReadRunner implements CommandLineRunner {

@Value("${app.info.title}")

private String title;

@Value("${app.info.ver}")

private Double version;

@Value("${app.db.driver}")

private String driver;

@Value("${app.db.url}")

private String url;

@Value("${app.mail.host}")

private String host;

@Value("${app.mail.port}")

private String port;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

}

}

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

Q) Can we active two propties file at a time?

A) YES, but no one uses like this in realtime

Use child profiles. Confilcit chould not come incase of @Profiles

@Value - Exception if key is not present

@ConfigurationProperties - null if key is not present

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

@Profile : This is used to execute logic based on Env/Client Specific.

\*) @Primary / @Qualifier => This is used to select any Impl class manually. But always

selected class only executed. we can not change it.

But @Profile is used to specify profile name at runtime and can change impl class

use at runtime.

\*) @Primary / @Qualifier are given by Spring Core.

@Profile is given using Spring Boot style with spring.profile.active=\_\_.

\*) @Profile("profilename") : This is used to specify current profile name

that should be selected. Even properties file also loaded same.

EX: @Profile("dev"), @Profile("qa")..etc

Q) What if we mention multiple active profiles ?

A) This case will throw Exception by saying. Expected Single Bean but found multiple Objects.

===Ex===

1. propeties files

application.properties

spring.profiles.active=nit

app.code=DEFAULT

application-htc.properties

app.code=HTC

application-nit.properties

app.code=NIT

2. Service Interface

package in.nareshit.raghu.service;

public interface AlertService {

public void send();

}

3. Impl classes

package in.nareshit.raghu.service.impl;

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

import org.springframework.context.annotation.Profile;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.service.AlertService;

@Component

@Profile("default")

public class EmailAlertService implements AlertService {

@Value("${app.code}")

private String code;

@Override

public void send() {

System.out.println("FROM EMAIL(DEFAULT) ALERT SERVICE :" + code);

}

}

----

package in.nareshit.raghu.service.impl;

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

import org.springframework.context.annotation.Profile;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.service.AlertService;

@Component

@Profile("nit")

public class SmsAlertService implements AlertService {

@Value("${app.code}")

private String code;

@Override

public void send() {

System.out.println("FROM SMS(NIT) ALERT SERVICE :" + code);

}

}

-------

package in.nareshit.raghu.service.impl;

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

import org.springframework.context.annotation.Profile;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.service.AlertService;

@Component

@Profile("htc")

public class SocialAlertService implements AlertService {

@Value("${app.code}")

private String code;

@Override

public void send() {

System.out.println("FROM SOCIAL(HTC) ALERT SERVICE : " + code);

}

}

---

4. Runner class

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.service.AlertService;

@Component

public class TestAlertServiceRunner implements CommandLineRunner {

@Autowired

private AlertService service;

@Override

public void run(String... args) throws Exception {

System.out.println("FROM RUNNER..");

service.send();

}

}

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

Q) if we are not specifying any profile then?

A) Default profile is selected.

Q) What if @Profile("default") is not provided?

A) It works fine when default profile is active.

But not for other profiles.

Then it creates object to EmailService class and even another which is activated.

So, finally 2 Beans found, Exception.

Incase of Spring Application Exceptions looks like:

\*) NoUniqueBeanDefinitionException multiple impl class objects found

\*) NoSuchBeanDefinitionException not even single impl class object found.

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

\*) Fallback in Profiles : If any key-val is not present in current profile then

Spring boot will come back to default profile for same key. If present, it is taken

into code.

--Ex--

1. properties files

application.properties

app.title=NIT-DEFAULT

app.ver=3.3

spring.profiles.active=prod

application-prod.properties

app.title=NIT-PROD

app.ver=2.2

2. Runner class

package in.nareshit.raghu.runner;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import lombok.ToString;

@Component

@ToString

public class TestFallBackRunner implements CommandLineRunner {

@Value("${app.title}")

private String title;

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

private Double version;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

}

}

-------------------------------------------------

=> To add any dependency after creating project

> open pom.xml

> come under tag <depndencies>

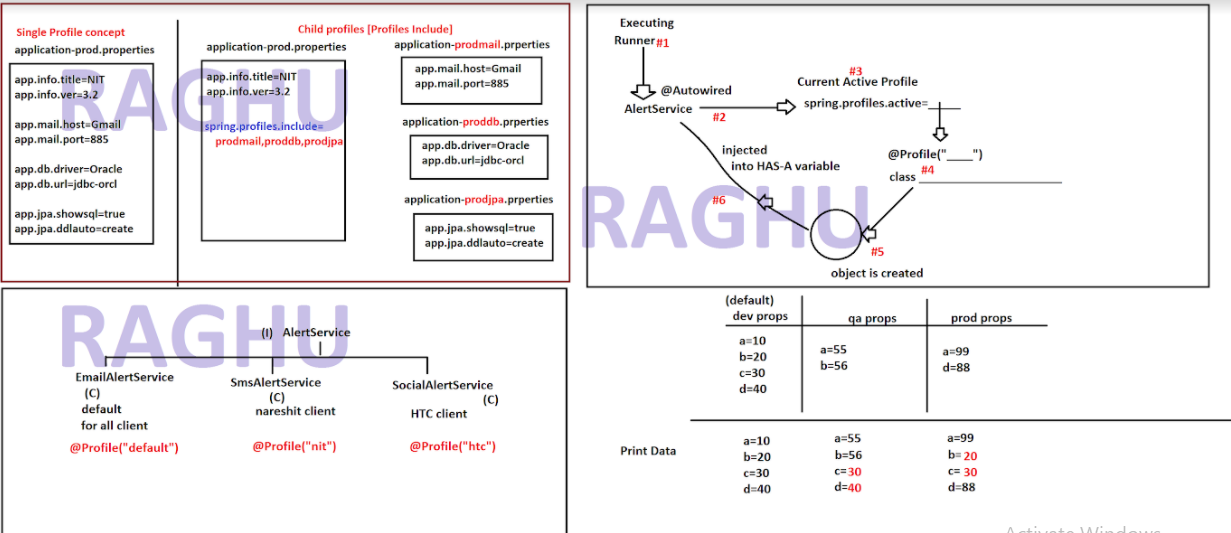
> pres ctrl+space

> choose EditStarters

> save Pom

> select dependnecies which are required

> Click on OK



Date : 17/07/2020

Spring Boot 7:50 PM

Mr. RAGHU

javabyraghu@gmail.com

--------------------------------------------------------------

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

https://www.youtube.com/c/NareshIT/playlists

Spring Boot Application 3 files (setup files)

a. Starter class / main method class

b. application.properties / application.yml

c. Build Information File | pom.xml - maven , build.gradle - Gradle

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

a. Starter class / main method class [Bootstraping class]

Starter class is used to create Spring container with all required objects(Beans)

and executes our Runners.

When we run starter class steps executed:-

a. Scan all classes using basePackage (@ComponentScan -Given by Spring Boot)

b. Check for class having StereoType Annotation (5)

@Component | @Service | @Repository | @Controller | @RestController

| @Configuration

@Scope() singleton\*\*/prototype/request/session/application-context

c. Create object and place in container.

d. Load Properties/Yml File into Environment Object (@PropertySource-Given by Spring Boot)

along with Profiles

e. Set Properties into required object (uses @Value/ @ConfigurationProps)

f. Execute Runners in given order.

=> Spring Supports 2 types of Container

1. BeanFactory (I) - XmlBeanFactory (C) [Old Container - Supports only XML Configuration]

2. ApplicationContext (I) [New Container XML/Java/Annotation config]

ClassPathXmlApplicationContext(C)

FileSystemXmlApplicationContext(C)

XmlWebApplicationContext(C)

AnnotationConfigApplicationContext(C)

...etc.

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

a) Spring Boot create new Container only.

AnnotationConfigApplicationContext (for stand alone apps)

AnnotationConfigServletWebServerApplicationContext (for web applications)

b) Starter class contains annotation @SpringBootApplication that internally

provides @ComponentScan("starter class package name").

Ex: @ComponentScan("in.nareshit.raghu")

It means A class which is inside "in.nareshit.raghu" package and its sub package classes

are detected. Other classes are ignored.

@Component = Just create object to our class

@Repository = create object + Database Operations

@Service = create object + Transaction Management + logic /calculation..etc

@Controller = creating object + Spring Web App (HTTP protocol)

@RestController = creating object + RestFul Webservices ( HTTP + JSON/XML)

@Configuration = Current class is Spring Java Configuration File.

=> In case of Pre-defined classes using Spring Java Configuration File

Use @Configuration with @Bean

@Configuration

public class AppConfig {

@Bean

public A oa(){

return new A();

}

}

-----------------------------------------------------------------------

Q) What is Environment in Spring?

A) It is a memory created inside Spring container that holds all keys

given by properties/yml files.

Q) How can we change basepackage from starter package to another package?

A) Provide externally @ComponentScan at starter class

@ComponentScan({"in.nareshit.raghu","xyz.abc","in.nit.test"})

------------------------------------------------------------------------------

Q) How to remove banner from Spring Boot Application?

A) By using setBannerMode(Banner.Mode.OFF); we can tru OFF.

@SpringBootApplication

public class SpringBoot2StarterInternalFlowApplication {

public static void main(String[] args) {

SpringApplication application = new SpringApplication(SpringBoot2StarterInternalFlowApplication.class);

application.setBannerMode(Banner.Mode.OFF);

ConfigurableApplicationContext ac = application.run(args);

}

}

Q) How can we chnage Banner in Spring Boot?

A) create a file banner.txt under src/main/resources folder.

> right click on src/main/resources > new File > Enter file name banner.txt

> Goto : https://devops.datenkollektiv.de/banner.txt/index.html

> Copy Content and paste in banner.txt file.

Q) Can we chnage banner.txt file name?

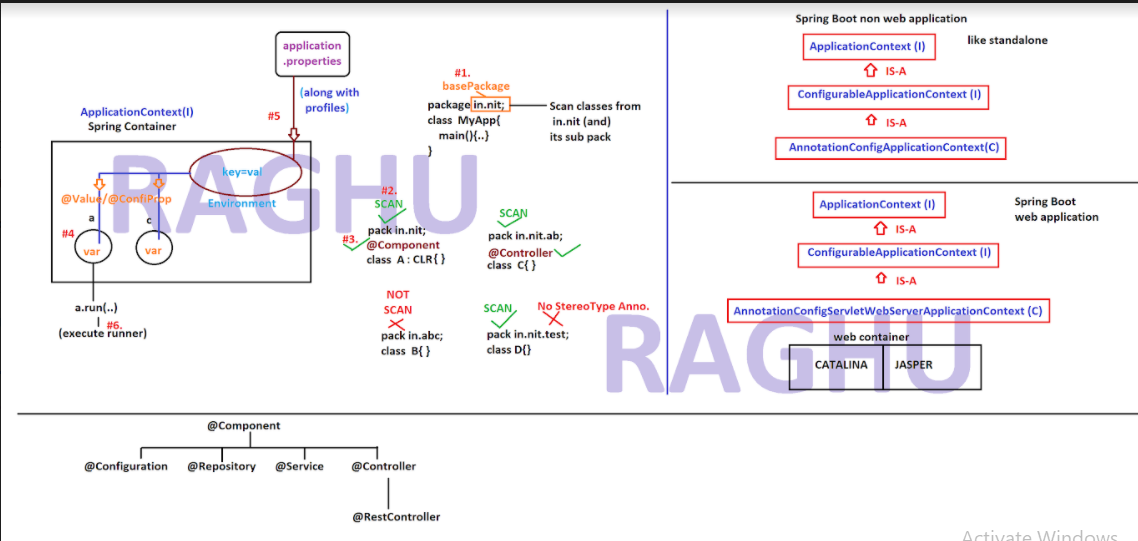
A) Yes use below key

spring.banner.location=classpath:abcd.txt

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

To See Pre-defined Keys:

https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html



Date : 18/07/2020

Spring Boot 7:50

Mr. Raghu

-------------------------------------------

https://www.youtube.com/c/NareshIT/search?query=java+8

PDF Copies:

https://www.mediafire.com/file/w5x9w5vcmkwkkdv/RaghuSirNareshITJavaPdfs.zip/file

---------------------------------------------------------------------------------------

Email: hr@nareshit.com

CC: javabyraghu@gmail.com

Name:

Mobile:

MailId:

PaySlip:

Spring Boot : Spring Based Framework that behaves even like Project.

Spring Boot is a Project that reduces common code done by programmer.

For Ex: Spring Project (online Shopping) 1000 LOC

Spring Boot Project (online Shopping) 400 LOC

Database Connection

Security

Log4j

Database CURD Operation

...common things

=> Our Project is called as Child Project, that is connected to Spring Boot

(Spring Boot Parent Project) that reduces common works

ex: Adding jars, common classes, creating container, loading properties,

scanning classes, linking objects, creating database connection,

writing Servlet Config (for web app)..etc

--in pom.xml--

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.1.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

\*) What is AutoConfiguration?

A) Providing common things to reduce no.of lines of code in Spring Boot Application.

-----------------------------Spring Boot Core--------------------------------------

This Chapter#1 is used to know how Spring boot uses concept internally.

Spring Boot Application - 3 setup files

\*a. Starter class/ Main method class

\*b. Input File (application.properties / application.yml) YAMALINA PROGRAMMING FILE

.yml -> Snake YAML -> .properties only

\*c. Build Information File

Concepts:

1. Runners \*\*

2. Reading Data from Input Files (@Value/@ConfigProperties) \*\*

3. Profiles

4. Lombok \*\*

5. Scheduling

6. Log4J(Logging)

Softwares Required:

JDK 1.8 (min) , work on : JDK 14

STS Tool: https://spring.io/tools

> Download as JAR > Double click to extract > Goto Folder > choose Option "SpringToolSuite4"

----Runners-----------------------

=> To execute any logic only one time use Runners concept

(ex: one time setup by taking inputs from properties file)

Ex: JdbcRunner code, SecurityRunner, WebMvcRunner, DataJpaRunner, EmailSetupRunner..etc

a. CommandLineRunner (mostly used internally)

b. ApplicationRunner (new Runner)

\*) Spring boot supports providing input to Spring container objects using

application.properties that holds data in key-val.

Here keys are two types

i. Pre-defined (mostly used)

ii. Programmer-defined

To see all pre-defined keys:

https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html

or goto application.properties and just type ctrl+space

--------------------------------------

=> @Value("${key}")

No set/get method required

=> if we provide key which is not matched/present with properties file then exception

IllegalArgumentException:

Could not resolve placeholder 'my.app.data' in value "${my.app.data}"

Lambda Exp: is used to recuce lines of code (LOC)

it is equals to writing Impl class and creating object

Interface ob = (params) -> { method body };

\*\* we can even configure CommandLineRunner using Spring Java Config File with Lambda Syntax:

Lambda Ex For CommandLineRunner:

CommandLineRunner cob = (args) -> {

System.out.println("FROM FIRST APP USING SPRING JAVA CONFIG+ LAMBDA");

};

--Spring Java based Configuration (Spring Core)--------------

a. write one public class with any name

Ex: public class AppConfig { .. }

b. Apply one annotation @Configuration

[So, that starter class will consider that as Spring Java Config file else ignored]

c. For one object creation in Spring container write one method

// no.of method = on.objects in container

public <class/interfaceName> <objectName>(){

}

d. Finally add @Bean over method.

\*\* Here @Configuration says to container this current java class is a spring container input.

\*\* Here @Bean indicates creating one object in container.

\*)Employee class create object using Spring java Config Code

@Configuration

public class AppConfig {

@Bean

public Employee eob(){

return new Employee();

}

}

\*) task : Student (sob)

------------------------------------------------------------

Date : 19/07/2020

Spring boot 7:50PM

(Backup-session-II)

Mr. RAGHU

---------------------------------------------------

Email: hr@naresit.com

Admin: srikanth (+91 6302 968 665)

Email: javabyraghu@gmail.com

FB: https://www.facebook.com/groups/thejavatemple/

Youtube PlayList:

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

Lombok/Project Lombok:-

Annotations to generate code in class like

@NoArgsConstrcutor : Default constrcutor

@AllAragsConstrcutor : All param constructor

@RequiredArgsConstrcutor : Specific constrcutor

@Setter : set methods

@Getter : get methods

@ToString : toString method

@EqualsAndHashCode : hashcode() and equals()

---Example ---------------------------

1. Create one Spring Starter Project

> File > new > Spring Starter Project > Enter details > Next

> Search Lombok > Choose same > next > Finish

pom.xml

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

2. Create a class and apply lombok annotations:

\*\*\* Annotations never follows order.

package in.nareshit.raghu.model;

import lombok.AllArgsConstructor;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

import lombok.ToString;

//ctrl+shift+O (imports)

@NoArgsConstructor

@AllArgsConstructor

@ToString

@Setter

@Getter

@EqualsAndHashCode

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

--Ex#2------

@RequiredArgsConstructor //selected variable params const

public class Employee {

@NonNull //selects variables for constrcutor params

private Integer empId;

private String empName;

private Double empSal;

}

=> Code Generated is:

public Employee(Integer empId) {

this.empId=empId;

}

--------------------------

\*\*\* If we apply @RequiredArgsConstructor without @NonNull annotation over any variable

then it generates default constructor.

@RequiredArgsConstructor //selected variable params const

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

Generated Code:

public Employee(){

}

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

\*) @Data = @Getter + @Setter + @RequiredArgsConstructor + @ToString + @EqualsAndHashCode

If we apply one @Data annotation , then it generates above all.

Ex:

package in.nareshit.raghu.model;

import lombok.Data;

//ctrl+shift+O (imports)

@Data

public class Employee {

private Integer empId;

private String empName;

private Double empSal;

}

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

Note: If Lombok is not working even after creatig Project then follow below steps:

#1. Goto Location:

C:\Users\<Your PC SystemName>\.m2\repository\org\projectlombok\lombok\1.18.12

#2. Double Click on JAR : lombok-1.18.12.jar

#3. It will open install

(wait for some time to detect STS IDE or choose manually/specify location of STS in your system)

Ex: F:/sts-4.7.0.RELEASE/

#4. Click on Install/Update button

#5. Open STS and work now.

----------------------------------------------------

\*\* If double click is not working (cmd - command prompt)

cmd>cd C:\Users\User\.m2\repository\org\projectlombok\lombok\1.18.12

cmd>java -jar lombok-1.18.12.jar

-----------------------------------------------------

@ConfigurationProperties : Bulk Loading/Bulk Reading

This concept is used to read multiple keys at a time into our code as variables data.

=> Key must look like (prefix=any common word or words all keys)

prefix.key=value

ex:

my.app.title=NareshIT

my.app.code=NIT

my.app.loc=HYD

my.app.ver=3.3

my.app.status=false

=> key in properties file must be variable(variable name) in code.

String title;

String code;

String loc;

Double ver;

Boolean status;

=> Finally, on top of your class (where wrote variables) appy

@ConfigurationProperties(prefix="my.app")

=> variables must have set/get methods (else data can not be given from properties to code)

\*\* If variable name and key name not matched then NO DATA, NO ERROR, NO EXCEPTION,

variable holds default value like null.

But @Value will give Exception.

\*\* @Setter/@Getter annotations are must, else NO DATA, NO ERROR, NO EXCEPTION,

variable holds default value like null.

\*\* If same key is provided multiple times in properties then

Spring boot shows Error, but it is allowed.

Last key-val pair is considered into application.

---Example#1 ----------------------

#1. Create Spring Boot Starter Project with Lombok

#2. application.properties

my.app.title=NareshIT-101

my.app.title=NareshIT-102

my.app.title=NareshIT-103

my.app.code=NIT

my.app.loc=HYD

my.app.ver=3.3

my.app.status=false

#3. Runner class

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import lombok.Getter;

import lombok.Setter;

import lombok.ToString;

@Component

@ConfigurationProperties(prefix = "my.app")

@Getter

@Setter

@ToString

public class DataReadRunner implements CommandLineRunner {

private String title;

private String code;

private String loc;

private Double ver;

private Boolean status;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

}

}

=============Working with multiple prefix==============================

\*\* For one prefix we need to write one class with variables.

Ex: my.app.--- => one class

my.db.----=> one class

my.email.---> one class

=> one class can bulk load at a time one prefix data only.

a. application.properties

my.db.driver=Oracle

my.db.url=jdbc-orcl

my.email.host=gmail-server

my.email.port=998

b. Runner class#1

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import lombok.Getter;

import lombok.Setter;

import lombok.ToString;

@Component

@ConfigurationProperties(prefix = "my.db")

@ToString

@Getter

@Setter

public class DatabaseRunner

implements CommandLineRunner {

private String driver;

private String url;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

}

}

c. Runnerclass#2

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import lombok.Getter;

import lombok.Setter;

import lombok.ToString;

@Component

@ConfigurationProperties(prefix = "my.email")

@ToString

@Setter

@Getter

public class EmailRunner

implements CommandLineRunner {

private String host;

private String port;

@Override

public void run(String... args)

throws Exception {

System.out.println(this);

}

}

Q) If we have multiple runners, then which one is executed first?

A) They are executing in class name sorted order (ex: alphabeticalorder)

Ex: EmailRunner--1

JdbcRunner--2

SecurityRunner--3

Q) Can we change order of runners?

A) YES. Use @Order(int value)

Lowest number is executed first.

--Ex--

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.core.annotation.Order;

import org.springframework.stereotype.Component;

@Component

@Order(32)

public class EmailRunner implements CommandLineRunner {

@Override

public void run(String... args) throws Exception {

System.out.println("EMAIL RUNNER");

}

}

---

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.core.annotation.Order;

import org.springframework.stereotype.Component;

@Component

@Order(-54)

public class JdbcRunner implements CommandLineRunner {

@Override

public void run(String... args) throws Exception {

System.out.println("JDBC RUNNER");

}

}

---

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.core.annotation.Order;

import org.springframework.stereotype.Component;

@Component

@Order(-6)

public class SecurityRunner implements CommandLineRunner {

@Override

public void run(String... args) throws Exception {

System.out.println("SECURITY RUNNER");

}

}

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

Q) If two runners (or more) having same order then?

A) Again Sorting order(alphabetic order).

Q) If few Runners are having order and few are not having order then ?

A)

If any runner is not having order then - default order value is : Integer.MAX\_VALUE (2147483647)

In simple they are executed last with name soring order.

First @Order Runners executed

NExt No Order Runners executed.

--Ex--

RunnerA - No Order

RunnerB - @Order(-6)

RunnerC - No Order

RunnerD - @Order(-6)

RunnerE - @Order(45)

RunnerM - @Order(2)

RunnerZ - @Order(12)

Order of Runners Execution: B D M Z E A C

BackupSession-III - Yaml/Profiles/Main class

Date : 20/07/2020

Spring Boot 7:50PM

Mr. RAGHU

----------------------------------------------------

Maven Session#1

https://www.youtube.com/watch?v=9eJAvApRAI0&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=8

Maven Session#2

https://www.youtube.com/watch?v=jOykBjSl1X0&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=7

Maven Session#3

https://www.youtube.com/watch?v=iLwuhbnFPqo&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=14

@SpringBootApplication =

@ComponentScan + @Configuration (@Component) + @PropertySource + @EnableAutoConfiguration

a. @ComponentScan : Spring container will scan/find/detect class from basePackage(common package name)

This basePackage details can be provided using Component-Scan only.

=> In Spring boot by default starter class package is set to basePackage using @SpringBootApplication.

Ex: Starter class is:

package in.nareshit.raghu;

//ctrl+shift+O

@SpringBootApplication

public class SpringBoot2MultiRunnerApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBoot2MultiRunnerApplication.class, args);

}

}

=> In above example starter class package name 'in.nareshit.raghu'.

So, classes which are inside 'in.nareshit.raghu' package ot its sub package are

scanned by spring container.

=> We can even modify basePackage from starter class package to any other (one or more)

use code at starter class:

@ComponentScan({ "pack1","pack2","pack3",.... })

Ex:

@ComponentScan({"app.test.one","in.nareshit.raghu","my.app.sample"})

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

\*\* In Spring we need to define 'Spring Java Config File' for creating objects

if the class is pre-defined in that case

Step#1 rite one public class with any name

Step#2 Apply @Configuration over class

Step#3 Define one method for one object

public <class/Interface> <objName>() {

//logic

}

Step#4 Apply @Bean (create object) over method

\*\* Incase of Spring Boot Step#1, Step#2 are not required that is handled using

Spring Boot Starter class.

So, we can directly write Step#3 and Step#4 inside main() class

===Ex#1 Spring Core Based Example =============

1. Bean class

package in.nareshit.raghu.db;

import lombok.Data;

@Data

public class DbConnection {

private String driver;

private String url;

public void getConn(){

System.out.println(driver+"-"+url);

}

}

2. Spring Java Config Class \*\*\*\*\*\*\*\*\*\*

package in.nareshit.raghu.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import in.nareshit.raghu.db.DbConnection;

@Configuration

public class AppConfig {

//one method = one object

@Bean //create one object

public DbConnection con() {

DbConnection con = new DbConnection();

con.setDriver("OracleDriver");

con.setUrl("jdbc:oracle");

return con;

}

}

3. Runner class

package in.nareshit.raghu.abcd;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.db.DbConnection;

@Component

public class MyMsgRunner implements CommandLineRunner {

@Autowired

private DbConnection con; //HAS-A

@Override

public void run(String... args) throws Exception {

System.out.println("HELLO");

con.getConn();

}

}

4. Starter class

package in.nareshit.raghu;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBoot2StarterWorkApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBoot2StarterWorkApplication.class, args);

}

}

======Ex#2 Spring Boot Based Example =============

1. Bean class

package in.nareshit.raghu.db;

import lombok.Data;

@Data

public class DbConnection {

private String driver;

private String url;

public void getConn(){

System.out.println(driver+"-"+url);

}

}

2. Runner class

package in.nareshit.raghu.abcd;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.db.DbConnection;

@Component

public class MyMsgRunner implements CommandLineRunner {

@Autowired

private DbConnection con; //HAS-A

@Override

public void run(String... args) throws Exception {

System.out.println("HELLO");

con.getConn();

}

}

3. Spring Starter class + Config Code

package in.nareshit.raghu;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import in.nareshit.raghu.db.DbConnection;

@SpringBootApplication

public class SpringBoot2StarterWorkApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBoot2StarterWorkApplication.class, args);

}

@Bean //create one object

public DbConnection con() {

DbConnection con = new DbConnection();

con.setDriver("MySQLDriver");

con.setUrl("jdbc:mysql");

return con;

}

}

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

Q) When to use @Component and @Configuration + @Bean ?

A) If class is pre-defined then (we do not have .java file)

Use Spring java based Configuration : @Configuration + @Bean

if class is defined by programmer (user defined class, we have .java code)

use @Component.

Q) Cant we use @Configuration + @Bean for Programmer defined class?

A) YES, we can use, but lengthy code.

---------------------------------------------------------------

\*\* @PropertySource : To load properties file in Spring application.

Ex: @PropertySource("classpath:jdbc.properties")

\*\* But in Spring Boot, by default name : application.properties is loaded by Starter.

internal code: @PropertySource("classpath:application.properties")

Q) Can we load other programer defined Proerties in Spring Boot?

A) YES. We can write @PropertySource(\_\_) at starter class, but not a recomanded standard.

Q) If we define @PropertySource(\_\_) at starter class, will spring boot load

application.properties?

A) YES. By Default application.properties is loaded. Even we can load other properties.

Q) We can define both application.properties and manual (other) proeprties in Spring boot?

A) YES. By Default application.properties is loaded. Even we can load other properties.

Ex: @PropertySource("classpath:my.properties")

\*\* it wil not override as like component scan.

--Ex# Spring Boot Properties -----

1. application.properties

ver=3.3

2. my.properties

code=NIT

3. Runner class

package in.nareshit.raghu.abcd;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

@Component

public class MyMsgRunner implements CommandLineRunner {

@Value("${code}")

private String code;

@Value("${ver}")

private String ver;

@Override

public void run(String... args) throws Exception {

System.out.println("HELLO::"+code+"-"+ver);

}

}

4. Main/Starter class

package in.nareshit.raghu;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.PropertySource;

import in.nareshit.raghu.db.DbConnection;

@SpringBootApplication

@PropertySource({"classpath:my.properties"})

public class SpringBoot2StarterWorkApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBoot2StarterWorkApplication.class, args);

}

}

Q) Can we load multiple properties file in Spring Boot?

A) YES.

@PropertySource({"classpath:my.properties" ,"\_\_\_","\_\_\_","\_\_\_" ,...})

Q) If one key is present with different value in both application.properties

<any-name>.properties (Non-spring Boot File) then which one is loaded into Container?

A) First priority order is taken by Spring boot with application.properties only.

If it is not present then <any-name>.properties (Non-spring Boot File) is loaded.

-\_Ex\_-

a. application.properties

code=ABC

b. my.properties

code=NIT

c. Runner class code

package in.nareshit.raghu.abcd;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

@Component

public class MyMsgRunner implements CommandLineRunner {

@Value("${code}")

private String code;

@Override

public void run(String... args) throws Exception {

System.out.println("HELLO::"+code);

}

}

==============================Spring Core=======================================

Annotation Configuration v/s Java Configuration

=> Spring Config (Anno/Java) is used to specify container "CREATE OBJECT TO GIVEN TYPE".

=> Annotation Configuration can be used along with our source code ie .java file is required

(Programmer defined classes only)

=> If a class is pre-defined then we can use only Java Configuration.

Q) If class is pre-defined , use Direct object creation where it is needed then what is

use of Spring Java config file?

A) If we define Student s = new Student() at one method then it behaves like local variable

even though object creation. it is not re-usable.

Core Java example

Ex:

class A {

void m1(){

Student s = new Student();

}

}

class B{

void m2(){

Student s = new Student();

}

}

A a= new A();

a.m1();

a.m1();

a.m1();

B b = new B();

b. m2();

b. m2();

b. m2();

Object is created 6 times for Student class

---------Spring core Style-----------------------------

@Bean

public Student sob() {

return new Student();

}

class A{

@Autowired

Student s;

void m1(){ // use s object here }

}

class B{

@Autowired

Student s;

void m1(){ // use s object here }

}

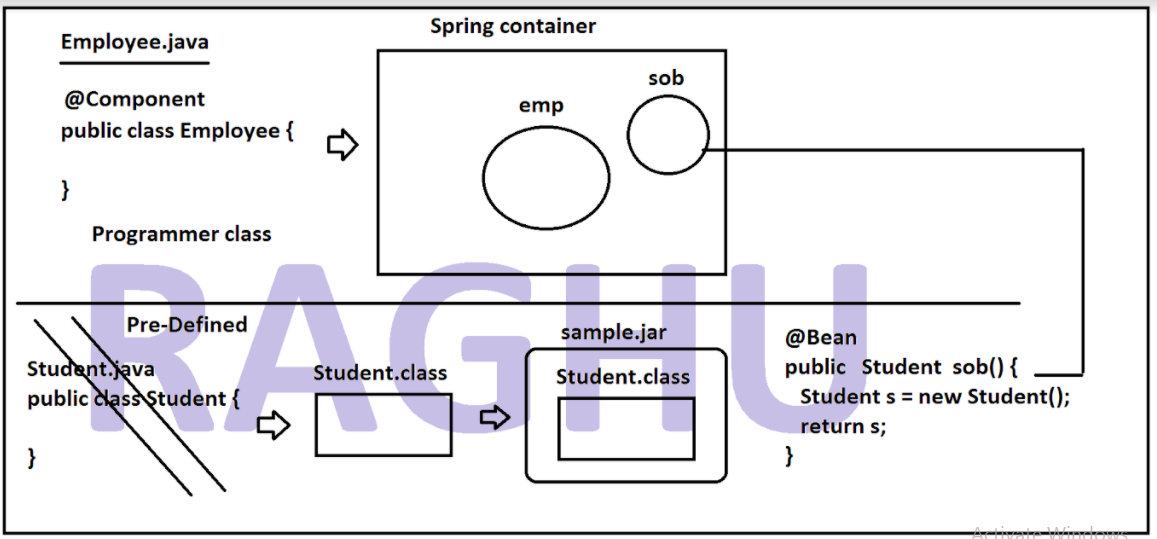
A a= new A();

a.m1(); a.m1(); a.m1();

B b = new B();

b. m2(); b. m2(); b. m2();

Object is created 1 time for Student class and used at multiple places.

****

Date : 21/07/2020

Spring Boot 7:50PM

Mr. RAGHU

-----------------------------------------------------------

Maven Session#1

https://www.youtube.com/watch?v=9eJAvApRAI0&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=8

Maven Session#2

https://www.youtube.com/watch?v=jOykBjSl1X0&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=7

Maven Session#3

https://www.youtube.com/watch?v=iLwuhbnFPqo&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=14

Q) in Spring Boot do we need to add version for a Dependency?

A) No. We are not required add version manully.

Spring Boot Parent Maven Project is taking care of Handling versions for

our dependencies which are mostly used in realtime.

Versions Like MySQL Database, Tomcat server, ServletsAPI, Lombok, DevTools, Cloud version..etc

Such concept is called as dependencyManagement using Parent Project.

Q) Can we write Spring boot application without parent?

How can we get Parent Project details?

A) Every Spring Boot application must be connected to Parent project

we can get details from pom.xml using tag <parent>

Q)\*\*\* How can we convert our project into jar/war format?

A) Execute Maven goals :

clean package (better one) (or) Maven install

> While creating Project Choose Package : JAR/WAR

> After coding complete > Right click on Project

> Run As > Maven Build ... > Enter Goals like : clean package

> Apply and Run

> Right click on Project > Refresh > open target folder > find JAR/WAR

Q) What are different scopes used in Maven/Spring Boot application for dependencies?

A) 5 Scope mainly exist:

compile (default)

runtime

system

test

provided

Even maven contains import scope that is applied only for DepenencyManagement tag.

Ex:

compile scope

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

runtime scope

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

provided scope

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<scope>provided</scope>

</dependency>

test scope

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

System Scope:

if jar in not present in maven , that exist in local Drive.

<dependency>

<groupId>com.app.one</groupId>

<artifactId>my-sample</artifactId>

<scope>system</scope>

<systemPath>D:/abcd/my-sample.jar</systemPath>

</dependency>

----------------------------------------------------------------------------------

Q) What is exclusions ? why should we use this?

A) If we add one dependency it may give multiple jars.

One Jar is actually called as Parent jars

Other jars are called child jars (Dependency chain jars).

<dependency>

<groupId>\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_</groupId>

<artifactId>\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_</artifactId>

<exclusions>

<exclusion>

<groupId>\_\_\_\_\_\_\_\_\_\_</groupId>

<artifactId>\_\_\_\_\_\_\_\_</artifactId>

</exclusion>

</exclusions>

</dependnecy>

Ex:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

==WAR FILE CREATION in SPRING BOOT===================================================

1. Create Spring Starter Project

Name : SpringBoot2WebTest

Package: WAR

> Next > Search and select Spring Web

> Finish

2. Write some code..\*\*\*(no code i wrote)

3. Right click on Project > Run as > maven install

4. Refresh proejct

5. Open target folder

Q) How can delete content in target folder

A) Maven clean (Run as > Maven Clean) Goal.

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

Date : 22/07/2020

Spring Boot 7:50 PM

Mr. Raghu

-----------------------------------------------

Tools Session:

GitHub Session-1 @ 9.00 AM

| by Mr. Raghu | 23rd July

Link: www.zoom.us/j/98331093737

------------------------------------------

Scheduling using Spring boot:-

Scheduling is a process of

"Executing a task in a loop" based on

PERIOD OF TIME or POINT OF TIME.

=> These are executed without human interaction program based.

Ex:

Bank Monthly Statements

Salary Playslips

Credit Card Bill/invoice

Stock Reports

Insurance Payment

Electricity Bill

Mobile Recharge

Daily StandUP meeting in office everyday.(Scrum meetings)

LICENSE RENIVAL

Sprint Planning

Alaram

Every day our class

Exams and Results

Festivals

----------------------------------------------------------

PERIOD OF TIME :- TIME GAP , NO STARTING DATE/TIME

4 hrs,

25 mins,

3 sec,

8 days,

3 months,

..etc

POINT OF TIME :- STARTING DATE/TIME

11AM

4th JAN

9PM - 22nd DEC

------------------------------------------------------

For every Spring boot basic application , scheduling is supported.

in pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

spring-boot-starter = Scheduling + Logging + AutoConfiguration + YAML Process.

----Coding Steps---------------------------------------------------

1. Create Spring Boot Starter project

2. At starter class level add below annotation:

@EnableScheduling

3. Define one class and apply @Component over class

4. Define one (or more method) with below syntax that executs task in scheduling

@Scheduled(\_\_\_\_\_\_\_\_)

public void <methodName> () {

//logic..

}

@Scheduled:

This annotation indicates to Spring container , execute below method in a loop.

Until application/server Stopped.

It supports scheduling using below concepts.

a. fixedDelay

b. fixedRate

c. cron expression

a. fixedDelay: This is used to specify exact time gap between

last method execution complete time and next method start time.

=> On Application startup, even scheduling also started without time gap.

We can give gap time for first call in scheduling using: initialDelay

Ex:

package in.nareshit.raghu;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class ReportGeneratorService {

//@Scheduled

// public void <methodName>() {...}

@Scheduled(initialDelay = 2000,fixedDelay = 5000) // 1000 mill sec = 1sec

public void generateReport() {

System.out.println("HELLO:"+new Date());

}

}

=> To stop execution, Goto Console and click STOP button (RED COLOR BOX TYPE BUTTON)

Q) Can we write only @Scheduled without any Input?

A) NOT POSSIBLE. Spring container throws Exception like:

IllegalStateException: Encountered invalid @Scheduled method 'generateReport': Exactly one of the 'cron', 'fixedDelay(String)', or 'fixedRate(String)' attributes is required

Q) Who will call @Scheduled method?

A) Spring container will create object to given class and executes.

Q) What is another way of providing fixedDelay?

A) we can provide as int type or String type

int type value : @Scheduled(fixedDelay = 1000)

String type value: @Scheduled(fixedDelayString = "1000")

Q) Can we define multiple @Scheduled methods ?

A) YES. Our application behaves like main thread. All @Scheduled methods behaves like

child threads. Which runs in background (Demon Thread).

Ex:

package in.nareshit.raghu;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class ReportGeneratorService {

//@Scheduled

// public void <methodName>() {...}

@Scheduled(fixedDelayString = "1000")

public void generateReportA() {

System.out.println("AA:"+new Date());

}

@Scheduled(fixedDelayString = "2000")

public void generateReportB() {

System.out.println("BB:"+new Date());

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

b. fixedRate:- The max time gap between method call.

@Component

public class ReportGeneratorService {

//@Scheduled

// public void <methodName>() {...}

@Scheduled(fixedRate=1000)

public void generateReportForMonthly() {

System.out.println("AA:"+new Date());

}

@Scheduled(fixedRateString = "2000")

public void generateReportForWeekly() {

System.out.println("BB:"+new Date());

}

}

Ex:

I gave one Task => 30 mins

fixedDelay - For sure 1 hr break

Boy-task-3 hrs

take rest 1 hrs

Boy-task-3hrs min

take rest 1 hrs

fixedRate - max time is 1 hr (after 1 start next task)

Boy-task-30 min (Reaming time = 1hr - 30 min = 30 min)

take rest 30 mins

Boy-task-30 min (Reaming time = 1hr - 30 min = 30 min)

take rest 30 mins

fixedRate - max time is 1 hr (after 1 start next task)

Boy-task-2 hr

NO WAITING

Boy-task-2 hr

NO WAITING

Boy-task-2 hr

\*\*\* Note: both fixedRate and fixedDely comes under PERIOD OF TIME.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

c. cron expression : \*\*\* It supports both POINT OF TIME and PERIOD OF TIME.

=> In Unix/Linux OS , to indicate date and time generally this is USED.

Same thing also added in Spring Boot.

CRON SYNTAX:

\* \* \* \* \* \*

SEC MIN HRS DAY MON WEEKDAY

0-59 0-59 0-23 1-31 1-12 SUN-MON

Allowed Symbols are: \* - , / ?

--Examples----------------------------------------------------------

Ex#1

0 0 10 \* \* \*

=> Execute given task every day 10:00:00 AM

Ex#2 comma(,) indicates possible values

0 0 8,10 \* \* \*

=> Execute given task every day 8:00:00AM and 10:00:00 AM

Ex#3 0 0 8-10 \* \* \*

Dash(-) indicates rages (take all possible values in between)

=> Execute given task every day (3 times)

8:00:00 AM

9:00:00 AM

10:00:00 AM

\*\* If our task is taking execution time "any value", this is executed 3 times here.

ob.m1();

ob.m1();

ob.m1();

Ex: 0 15 \* \* \* \*

=> Execute Given task at every hr 15th min.

It is not 15 min gap.

9:00:10-started

9:15:00 - TASK EXECUTED

10:15:00- ..

11:15:00 -...

Ex: 10 \* \* \* \* \*

=> Execute task for every minute 10th sec.

9:00:10 - TASK EXECUTED

9:01:10

9:02:10

--Example code---------------------------

package in.nareshit.raghu;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class ReportGeneratorService {

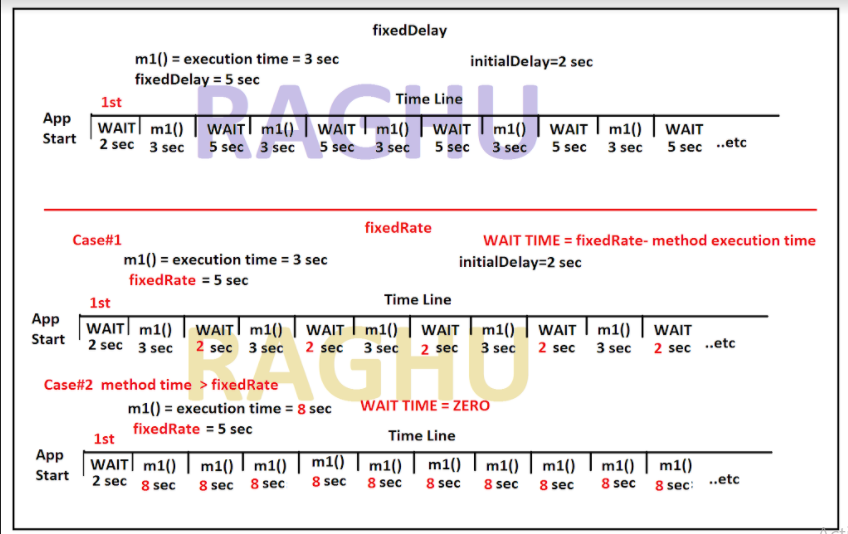
@Scheduled(cron = "10 \* \* \* \* \*")

public void generateReportB() {

System.out.println("HELLO:"+new Date());

}

}

****

Date : 23/07/2020

Spring Boot 7:50PM

Mr. RAGHU

-----------------------------------------------------------------

Github:

https://www.youtube.com/watch?v=T2UHpsxJ-2o&fbclid=IwAR3DcDd8mVVzpZ5QvjaofEvZgXOawF9Tnsaqz9xXbB3oI6nP38pGAETvUao

Log4J

https://www.youtube.com/watch?v=mblGoKU1aKo&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=6

https://www.youtube.com/watch?v=5oLfHiP\_iJc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=5

https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/scheduling/support/CronSequenceGenerator.html

cron :- This is expression used to indicate date and time

It is actually given by Unix/Linux OS.

Syntax:

SEC MIN HRS DAY MONTH WEEKDAY

0-59 0-59 0-23 1-31 1-12 SUN-SAT

Possible Symobols in CRON: \* - / , ?

\* = any/all/every

- = range (between value)

/ = Period of time

, = possible set of values

? = any [but used only at DAY and WEEK DAY when MONTH is provided]

---Examples-----------------------------

Ex#1: 0 0 5 \* \* \*

Task is executed every day 5:00:00 AM

Ex#2: 0 0 18 \* \* \*

Task is executed every day 6:00:00 PM

Ex#3 Write one expression that executs task at 4PM every day

Hint PM = add +12 (or) -12

0 0 16 \* \* \*

Ex#3 0 0 8,10 \* \* \*

Task is executed every day 8:00:00 AM and

10:00:00 AM

Ex#4 Write one expression that executs task at 6AM and 6PM every day

0 0 6,18 \* \* \*

Ex#5 0 0 8-11 \* \* \* [Dash indicates Range]

Task is executed every day

8AM, 9AM, 10AM, 11AM

Ex#6 Write one expression that executs task at 5PM and 3 more times after every hour.

0 0 17-20 \* \* \* (best)

0 0 17,18,19,20 \* \* \*

Ex#7 Execute task Morning 11:30 AM and 11:30 PM every day

SEC MIN HRS DAY MONTH WEEKDAY

0 30 11,23 \* \* \*

Ex#8 0 0 \* \* \* \*

Task is executed every hr 0thMin and 0th Sec

9:00:00 AM

10:00:00 AM

Ex#9 20 \* \* \* \* \*

Task executed every minute 20th sec

6:00:20

Next 6:01:20

Next 6:02:20

Ex#10 Execute a task on every minute 45th sec

45 \* \* \* \* \*

Q) What is the difference?

45 0 0 \* \* \* => Every day midnight 12O'clock 45th sec : 12:00:45 AM

45 \* \* \* \* \* => Every minute 45th sec.

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

Ex#11

0 0 9 12 3 \*

Execut task on every Year on March 12th 9:00:00 AM

Ex#12

0 0 10 10 \* \*

Execut task on every Month 10th Day 10:00:00AM

Ex#13 Execute task on Dec 25th 7AM

0 0 7 25 12 \*

Ex#14

0 0 9 10 5 SUN,FRI

Given task is executed every year on May 10th 9:00:00 AM

if given day(10th) is SUN or FRI DAY only.

Ex#15

0 0 10 \* 1 \*

=> When month is provided and you want to specify any day and any week

(or) every day and every week use ? inplace of \*

0 0 10 ? 1 ?

=> Symbol ? is applied only for day and week day.

task executed every year JAN every day 10AM

Ex#16 0 \* 10 \* \* \*

Invaild

=> Once Symbol \* is used then next position values

are not allowed that means wrong expression

Ex#17

\* 0 9 \* \* \*

(invaild)

Ex#18 HAPPY BIRTHDAY TO ME

59 59 23 20 7 \*

Every year July 20th midnight 1sec before 12 NOON

23:59:59 (if we add 1 sec next day 21st 12AM)

Ex#19 HAPPY NEW YEAR

59 59 23 31 12 \*

if we add +1

0 0 0 1 1 \*

Ex#20

0 30 9 12 6 \*

Add 12 days 6hrs and 20 mins

0 50 15 24 6 \*

========Period of time=============================================

Use Symbol / at any position (not in week)

Ex#21 \*/10 \* \* \* \* \*

For every 10 sec gap (Here symbol / indicates time gap)

Ex#22 0/10 \* \* \* \* \*

0 \* \* \* \* \*

+

\*/10 \* \* \* \* \*

Starting of execution at 0th sec of every minute

and

also execute with 10 sec gap

Ex#23 0 0/15 9 \* \* \*

Every day 9AM and repeate with gap of 15 min

9:00:00 AM

next 9:15:00 AM

next 9:30:00 AM

next 9:45:00 AM

next day

Ex#24 0 0/10 \* \* \* \*

Stating: Every hr 0th min and 0th sec

Gap: 10 mins

9:00:00 - start

Next 9:10:00

Next 9:20:00

9:30:00

9:40:00

9:50:00

next 10:00:00

Ex#25 0/10 30/10 9 \* \* \*

Every Day

Start - 9:30:00 AM

Next - 9:40:10 AM

Next - 9:50:20

Next - Next DAY

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

Date : 24/07/2020

Spring Boot 8PM

Mr. RAGHU

-------------------------------------------------------

Chapter#2 Spring Boot Email

( AutoConfiguration )

\*) Message : (Email) Message is collection of below details

To, Cc, Bcc, Subject, Text, Attachments.

=> In Programming Message is called MimeMessage

(MIME=Multipurpose Internet Mail Extension)

=> In simple, MIME indicates sending any type data over email

Text, Audio, Video, PDF, Document, Images..etc

=> Once we create MimeMessage, we can perform send operation.

=> Todo Above process we should login into Mail Account,

In programming it is called as Session (Email Session).

--Task In Order--

1. Login into Email account

2. Create Message object

3. Send Operation

----------------

=> Todo Above tasks in order Sun Microsystems(Now Oracle) has provided JAVA MAIL API

[Jakarta Mail API] But code is very lengthy. It is simplified by Spring Boot

using one Starter (Dependency)

spring-boot-starter-mail

-> Above Starter provides requires jars

i. Spring Context Support Jars [Email Coding]

ii. Jakarta Mail API

iii. Spring Boot Starter Jar

//ctrl+shift+T (open pre-defined .class file)

=> JavaMailSender(I) which is given by Spring Context Support, this is used to

create Message(MimeMessage) and perform send() operation.

=> Here JavaMailSender is interface, So use Impl class ie JavaMailSenderImpl

which is given by Spring Context Support.

\*\*\* JavaMailSenderImpl will login into Email account.

---Spring Java Configuration for JavaMailSenderImpl object creation---

@Configuration

public class AppConfig {

//1 method = 1 object

@Bean

public JavaMailSender mailSender(){

JavaMailSenderImpl impl = new JavaMailSenderImpl();

impl.setHost(\_\_);

impl.setPort(\_\_);

impl.setUsername(\_\_);

impl.setPassword(\_\_);

impl.setJavaMailProperties(\_\_);

return impl;

}

}

--------------------------------

=> Incase of Spring f/w Above code(Configuration) must be given by Programmer,

coming to Spring boot this is given by spring-boot-starter-mail (AutoConfiguration).

=> But programmer has to given inputs using either properties or yml.

--application.properties--

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=paramesh.nit.12

spring.mail.password=boot123456

#Support Login Enable

spring.mail.properties.mail.smtp.auth=true

#Support Security for TLS (Transport Layer Security)

spring.mail.properties.mail.smtp.starttls.enable=true

--application.yml--

spring:

mail:

host: smtp.gmail.com

port: 587

username: paramesh.nit.12

password: boot123456

properties:

mail:

smtp:

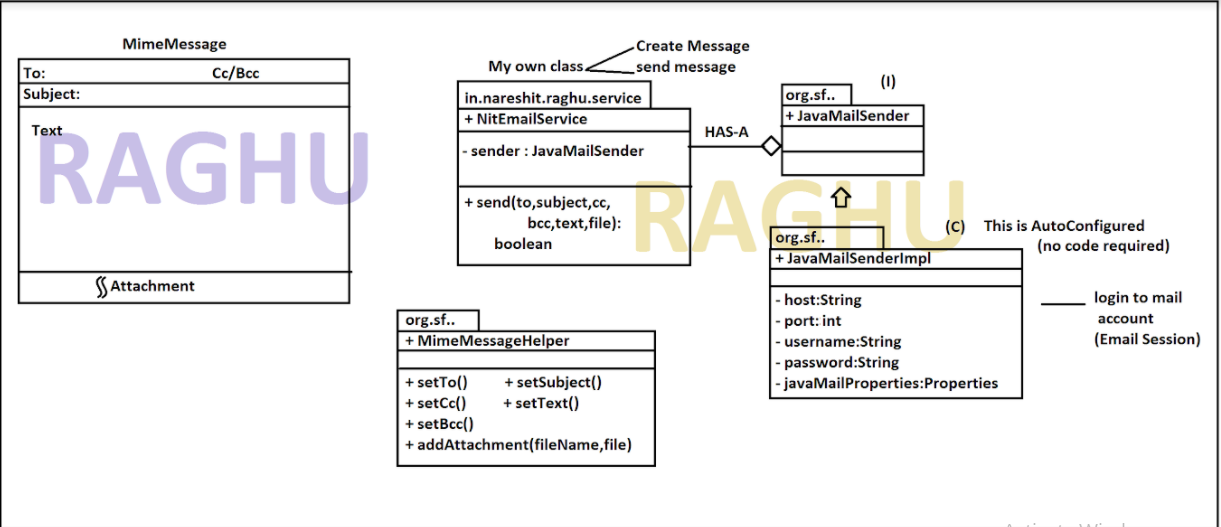
auth: true

starttls:

enable: true

=> MimeMessageHelper(C) this class is used to reduce lines of code to constrcut

(or create) MimeMessage object with details, like to,cc,bcc,subject,text,files..etc

****

Date : 25/07/2020

Spring Boot 8PM

Mr. RAGHU

--------------------------------------------------------------

Spring Boot Email

=> Sun Microsystem has provided API 'Java Mail API' (Jakarta Mail API).

=> Code is very lengthy using Java Mail.

=> Simplified by Spring boot by using AutoConfiguration

i. Common Jars (Spring Context, Java Mail, Spring Core Jars..)

ii. Common classes and interface (JavaMailSender and Impl)

iii. Configuration code ( @Bean -> JavaMailSenderImpl)

--3 Steps--

1. Login To Mail Account

2. Create Message(MimeMessage) [MIME=Miltipurpose Internet Mail Extension)

3. Send Operation

---------------

Here Step#1 can be done if we create object to JavaMailSenderImpl (Bean),

But this work is done by Spring boot only , we need to give inputs

Like host,port, username, password and properties.

Next, Step#2,Step#3 should be done programmer only , for that we can use

interface JavaMailSender(I). This interface is providing methods like

a) createMimeMessage() that creates Empty Message

b) send(MimeMessage) it will send message to given To,CC,BCC address.

We can use one pre-defined class ie 'MimeMessageHelper', as name indicates

it is used to set details like To,Cc,Bcc,Subject,Text and Files to MimeMessage

using less lines of code. It is optional, if we don not use this Lines of Code

will be more.

--Extra work-----------------

1. Enable Less secure apps your sender email account (it is free service)

>> Login to Sender Gmail account

> Click on Top Right corner on name/Letter Symbol

>> Manage your google accounts

>> Click on Security

>> Goto Less Secure Apps

>> Click on enable less secure apps

>> Click ON

2. Disable anti-virus and firewalls for 10mins

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

Q) What is FileSystemResource ?

A) To send any file which actually exist in Drives (D:/, E:/...) or folder System

to select that file in core Java concept

File f = new File("");

But to get its data like filename and converting byte[] format should be done manually

that is coded as FileSystemResource class.

------------code steps---------------------------

#1. Create Spring Starter Project with Dep: Java Mail Sender

#2. application.properties

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=paramesh.nit.12

spring.mail.password=boot123456

#Support Login Enable

spring.mail.properties.mail.smtp.auth=true

#Support Security for TLS (Transport Layer Security)

spring.mail.properties.mail.smtp.starttls.enable=true

#3. Email Service

package in.nareshit.raghu.service;

import javax.mail.internet.MimeMessage;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.core.io.FileSystemResource;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.mail.javamail.MimeMessageHelper;

import org.springframework.stereotype.Component;

@Component

public class NitEmailService {

@Autowired

private JavaMailSender sender; //HAS-A

public boolean send(

String to,

String[] cc,

String[] bcc,

String subject,

String text,

FileSystemResource file

)

{

boolean status = false;

try {

//1. create empty message / new message

MimeMessage message = sender.createMimeMessage();

//2. take helper object

//true-allocate memory for attachment

//false-do not waste memory, no attachment

//if file!=null - means file exist

MimeMessageHelper helper = new MimeMessageHelper(

message, file!=null?true:false);

//3. provide details to message

helper.setTo(to);

helper.setSubject(subject);

if(cc!=null) //cc is optional in email

helper.setCc(cc);

if(bcc!=null)

helper.setBcc(bcc); //it is also optional

helper.setText(text);

if(file!=null) {

helper.addAttachment(file.getFilename(), file);

}

//4. send message

sender.send(message);

status = true;

} catch (Exception e) {

status=false;

e.printStackTrace();

}

return status;

}

}

#4. Test it

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.core.io.FileSystemResource;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.service.NitEmailService;

@Component

public class MailTestRunner implements CommandLineRunner {

@Autowired

private NitEmailService nes;

@Override

public void run(String... args) throws Exception {

FileSystemResource file =

new FileSystemResource("F:\\Images\\MONO-SOA-MS.png");

boolean sent = nes.send(

"asjadshahab@gmail.com",

new String[] {

"amittestmail02@gmail.com",

"talwar.aa@gmail.com",

"kruti.kruti16@gmail.com",

},

new String[] {

"prateekgupta.mittal@gmail.com",

"abdulbaseer481@gmail.com",

"mlpmukesh@gmail.com"

},

"HELLO",

"WELCOME",

file);

if(sent) {

System.out.println("Email sent succesfully");

}else {

System.out.println("Email not sent succesfully");

}

}

}

Date : 27/07/2020

Spring Boot 7:50PM

Mr. RAGHU

------------------------------------------------------

Chapter # Spring Boot using Spring Data JPA

---------------------------------------------------

in pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

JPA is a Sun/Oracle Specification that follows ORM concept

and this simplified by Spring Data JPA to work with Database.

=> Compared to other concepts (F/w and Technologies) Sprig Data JPA is faster

and code is less lines.

--------------------------------------------------------------------

ORM - JPA

ORM - Object Relation Mapping - Java Persistency API [javax.persistence]

(Theory) (Programming)

--------------------------------------------------------------------

ORM - Mapping : One class (Model/Entity) will be mapped with one Database Table

=> Mapping Means link one class with one table using JPA Annotations given by Sun/Oracle

=> Mapping should follow rules like

a) class name must be mapped with table name

b) variable name must be mapped with column

c) Every table/class must have one PK column/variable.

=> Programmer should do MAPPING, then ORM F/w will generate SQL when we call operations

(methods given by ORM F/w)

---Example--------------

package in.nareshit.raghu.model;

//ctrl+shift+O

import javax.persistence.\*;

@Data

@Entity

@Table(name="emptab)

public class Employee {

@Id //PrimaryKey

@Column(name="eid")

private Integer empId;

@Column(name="eid")

private String empName;

@Column(name="eid")

private Double empSal;

}

Here, @Entity and @Id are required, @Table, @Column are optional.

@Entity : It maps class with table and variables with clumns.

@Id : It indicates PrimaryKey

@Table : It is optional, If we do not provide table name

then by default className is taken as tableName.

@Column: It is optional, If we do not provide column name

then by default variablename is taken as column.

-------------------------------------------------------------

Connection with Database:-

spring.datasource.driver-class-name=<DriverClassName>

spring.datasource.url=<DB URL>

spring.datasource.username=<DB UserName>

spring.datasource.password=<DB PWD>

-Ex- MySQL

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/<DBName>

spring.datasource.username=root

spring.datasource.password=root

-Ex- Oracle

<dependency>

<groupId>com.jslsolucoes</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0.1.0</version>

</dependency>

<dependency>

<groupId>com.oracle.database.jdbc</groupId>

<artifactId>ojdbc8</artifactId>

<scope>runtime</scope>

</dependency>

--Example Properties--

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:ORCL

spring.datasource.username=system

spring.datasource.password=nit

-ORM Properties-

dialect : Dialect is pre-defined class given by ORM f/w that will generate SQL query

when we perform operation. All are defined in package: org.hibernate.dialect

Ex: MySQLDialect, Oracle10gDialect,..etc

https://docs.jboss.org/hibernate/orm/5.2/javadocs/org/hibernate/dialect/package-summary.html

show-sql : It will display GENERATED SQL on console/log file

if value is true , default is false- DO NOT DISPLAY GENERATED SQL

ddl-auto : 4 Possible values

a) create : Every time creates new tables created (even if exist or not)

If old tables are there with data, then drop them and create new

on app starterup.

b)\*\*update :

If table exist - use same

Table not exist - create new

=> Alter table if we modify/add variables in model class

c)\*validate: [Default value]

ORM F/w does nothing, Programmer has to create/alter/drop tables manually

d) create-drop :

Creates new tables on app starterup (by Dropping old tables if exist)

and after performing operation, while Stopping application

again tables are dropped.

--Example Properties--

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

---application.yml--------------

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

password: root

url: jdbc:mysql://localhost:3306/boot8pm

username: root

jpa:

database-platform: org.hibernate.dialect.MySQL8Dialect

hibernate:

ddl-auto: create

show-sql: true

-------------------------------

Q) If we want to move from one Database to another Database, where do we need to

change code in Spring Data?

A) only properties/yml file and pom.xml dependencies.

------------------------------------------------------------

Spring Boot Embedded Database [No Download + No Install]

These databases also called as InMemoryDatabases,

these are exist in RAM, when start app they are created

and when we stop applocation, data is gone.

=> For Embeeded Database, internally hibernate.ddl-auto: create-drop

=> There are 3 embedded Dbs are exist

H2

HyperSQL (HSQL)

Derby

=> \*\*\* Use embedded databases only for DevEnvironment,

to avoid database download and install, only work on actual coding.

=> \*\* These are never used in Production.

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

Q) What is Proxy pattern? Or proxy class generation using interface?

A) class is generated at runtime with full logic by using interface Design,

that uses Reflection API.

https://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Proxy.html

https://docs.oracle.com/javase/8/docs/technotes/guides/reflection/proxy.html

https://blogs.oracle.com/javamagazine/the-proxy-pattern

=> In Spring, for our Module Database operations, we do not write class.

Just Define interface that extends (any one below)

CrudRepository<T,ID>

PagingAndSortingRepository<T,ID>

JpaRepository<T,ID>

\*\*One module one interface.

Ex: Employee module

package in.narseshit.raghu.repo;

//ctrl+shift+O

public interface EmployeeRepository

extends CrudRepository<Employee,Integer> {

}

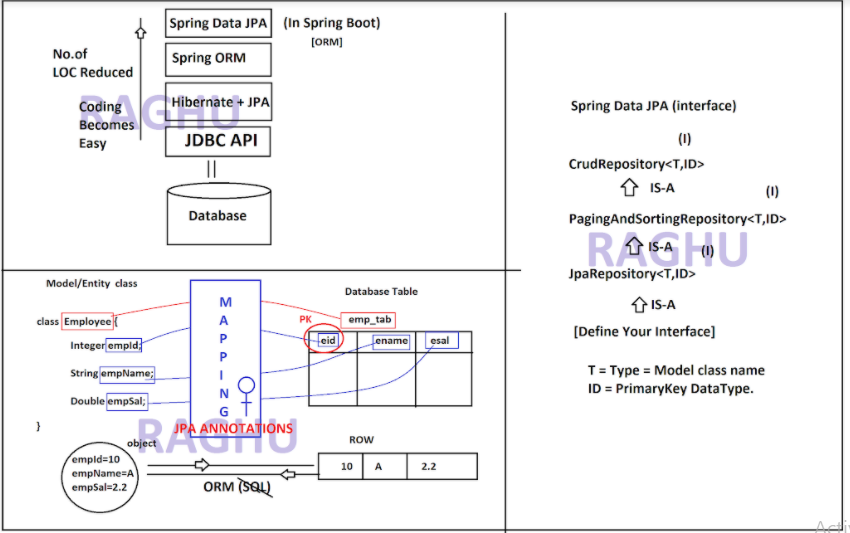
=> For above interface 'EmployeeRepository' a class is generated at runtime

by writing code for Data JPA with all methods.

=> \*\*\* Do not write any Impl class in Spring Data JPA.

=> \*\*\* We can even provide custom Queries (SQL,JPQL/HQL,Joins,procedure calls..etc)

[All are abstract methods only-Code given by Spring Data JPA]

****

Date : 30/07/2020

Spring Boot 7:50PM

Mr. RAGHU

----------------------------------------------

Github: https://github.com/javabyraghu

Working with Proxy using Core Java Concept like Spring Data JPA

\*\* https://github.com/javabyraghu/DynamicProxyExample

==================== Spring Boot Data JPA + H2 Database =============================

Steup:-

#1. Create one Starter Project

Name : SpringBoot2DataJPAH2Ex

Dep : Spring Data JPA, Lombok, H2, Spring Web.

\*\*\* Here we are adding Spring web to see UI for H2.

#2 application.properties

#Default port number 8080

server.port=9898

#Enable Browser view for data base

spring.h2.console.enabled=true

#Give database name

spring.datasource.url=jdbc:h2:mem:testdb

#Display Query at console

spring.jpa.show-sql=true

#3. Run Starter class and goto browser, enter url:

http://localhost:9898/h2-console

#4 Provide JDBC Url : jdbc:h2:mem:testdb

#5. Click on Connect

#6. Come back to STS console and click on STOP button (RED COLOR)

\*\*\* 3 Repository interfaces given by Data JPA

=> CrudRepository(I)

=> PagingAndSortingRepository(I)

=> JpaRepository(I)

\*\*\* For embeed database ddl-auto=create-drop

[Creates new table on startup and drops current table while stoping]

----------Ex#code--------------------------------------------------

Code Files

1. Model class

2. Repository interface

3. Runner class

1.model class

package in.nareshit.raghu.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

//ctrl+shift+O

@Data

@NoArgsConstructor

@AllArgsConstructor

@Entity

@Table(name="emp\_tab")

public class Employee {

@Id

@Column(name="eid")

private Integer empId;

@Column(name="ename")

private String empName;

@Column(name="esal")

private Double empSal;

}

2. Repository interface

T= Model class name

ID=PrimaryKey DataType

package in.nareshit.raghu.repo;

import org.springframework.data.repository.CrudRepository;

import in.nareshit.raghu.model.Employee;

//ctrl+shift+O

public interface EmployeeRepository

extends CrudRepository<Employee, Integer>

{

}

3. Runner class

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmployeeTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

@Override

public void run(String... args) throws Exception {

//insert rows

repo.save(new Employee(100,"SAM",2500.0));

repo.save(new Employee(101,"SYED",3500.0));

repo.save(new Employee(102,"RAM",4500.0));

repo.save(new Employee(104,"KHAN",9500.0));

System.out.println("DONE");

}

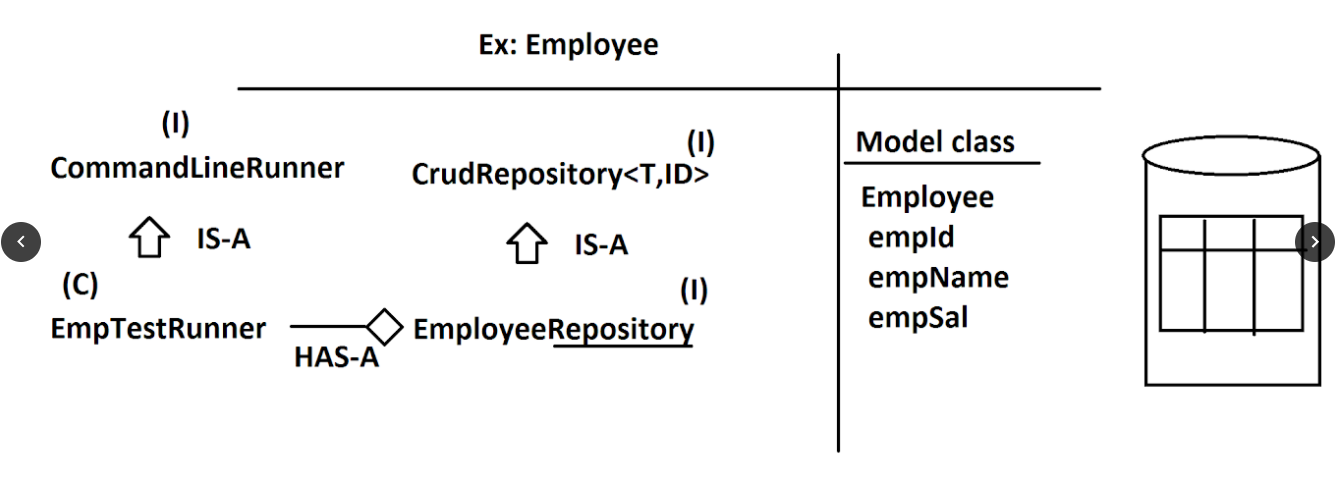
}

> Run this starter class > goto browser > enter URL

http://localhost:9898/h2-console/

> click on connect > click on table name > Run

> Come back to STS > goto console > STOP button (red color)

****

Date : 31/07/2020

Spring Boot 7:50PM

Mr. RAGHU

--------------------------------------------------------------

Task:

1. Write one Runner class that print

'product.code' taken from properties file

2. Write Profiles example using YAML concept.

3. Write Example using properties for @Value and

@Configuration properties for List,Set.

4. Write Spring Java based configuration using

Employee class.

Date : 03/08/2020

Spring Boot 7:50 PM

Mr. RAGHU

------------------------------------------------------

Spring Data JPA : Connecting with External Database

Database Connection Properties:

driver-class-name

url

username

password

ORM Properties:

dialect

show-sql

ddl-auto

\*) Model class

\*) Repository interface

\*) Runner class (TEST)

--MySQL Commands-------------

> Login using Pwd: \*\*\*\*

> create new database

mysql> create database boot8pm;

> Use database

mysql> use boot8pm;

> view tables

mysql> show tables;

---------Example---------------

#1. Create new Starter project with Dep: Spring Data JPA, Lombok, MySQL Driver

#2. Write application.properties or yml

--application.properties--

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/boot8pm

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

--application.yml--

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

password: root

url: jdbc:mysql://localhost:3306/boot8pm

username: root

jpa:

database-platform: org.hibernate.dialect.MySQL8Dialect

hibernate:

ddl-auto: create

show-sql: true

=> Here Driver class and database-platform (dialect) are auto-detected based on URL

those are optional.

#3. Model class

package in.nareshit.raghu.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Entity

public class Employee {

@Id

private Integer empId;

private String empName;

private Double empSal;

private String empDept;

}

#4. Repository Interface

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

}

#5 Runner classes

--a---------

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

repo.save(new Employee(101, "AJAY", 250.0, "DEV"));

repo.save(new Employee(102, "SAM", 350.0, "DEV"));

repo.save(new Employee(103, "SYED", 280.0, "BA"));

repo.save(new Employee(104, "VIJAY", 390.0, "QA"));

}

}

\*) save(obj) : This method is given by CrudRepository which behaves like either save

or update.

First it will read id from object, if id exist in database then

update, else insert query is generated and executed.

Q) What is the difference between persist(),merge() and save() methods?

A) persist() : it is insert operation, returns void [given by Sun JPA]

merge() : it will update record, that returns same object [given by Sun JPA]

save() : either persist() or merge() [given by Spring boot Data JPA]

\*\*) To see pre-defined code for data JPA, open class: SimpleJpaRepository

ctrl+shift+T > SimpleJpaRepository > double click.

ctrl+O (to see all method/variables)

Q) What is Batch Insert/Update in JDBC?

A) Executing multiple insert/update queries at a time using one set of inputs

with one network call between app and database.

-----------------------------------------------------

\*) saveAll(Iterable<S> objects) :-

this method is used perform JDBC Batch update internally.

ie insert/update at a time multiple objects in database using single network call.

--Runner class code---

package in.nareshit.raghu.runner;

import java.util.Arrays;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

//pass list of objects to saveAll() method

repo.saveAll(

Arrays.asList(

new Employee(101, "AJAY", 250.0, "DEV"),

new Employee(102, "SAM", 350.0, "DEV"),

new Employee(103, "SYED", 280.0, "BA"),

new Employee(104, "VIJAY", 390.0, "QA")

)

);

}

}

---------------------------------------------------

\*) Task: [Use Apache POI/ https://poi.apache.org ]

> Create one Excel Sheet with Students data

> Read excel Sheet data as List<Student>

> saveAll() into database using Data JPA.

=====select operations=======================================

findAll(): Iterable<T>

[Returns all rows from DB]

findAllById(Iterable<ID> ids): Iterable<T>

[Multiple selected rows by id using in operator internally]

findById(ID id):Optional<T>

[Returns only one row-> object]

existsById(ID id):boolean

true if given id exist as row in db, else false.

Java 8:

https://www.youtube.com/c/NareshIT/search?query=java+8

https://www.youtube.com/watch?v=YXJw3Zq0GjI&list=PLVlQHNRLflP\_VZ0e3G7BUhjn11sMbiXz\_

\*) operator in is used to select random rows from database

Ex: Select 34,22,18,56,92 rows from employee

SQL: select \* from emptab where id in(34,22,18,56,92)

---Runner class code--

package in.nareshit.raghu.runner;

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

repo.saveAll(

Arrays.asList(

new Employee(101, "AJAY", 250.0, "DEV"),

new Employee(102, "SAM", 350.0, "DEV"),

new Employee(103, "SYED", 280.0, "BA"),

new Employee(104, "VIJAY", 390.0, "QA")

)

);

System.out.println("------SELECT ALL--------");

repo.findAll().forEach(System.out::println);

System.out.println("------SELECT FEW--------");

//pass list input

repo.findAllById(List.of(101,103)).forEach(System.out::println);

System.out.println("------SELECT one--------");

//JDK 1.8 Optional , to make null check, when data got from DB

Optional<Employee> opt = repo.findById(104);

if(opt.isPresent()) {

System.out.println(opt.get());

} else {

System.out.println("NO DATA FOUND");

}

System.out.println("------CHECK EXIST--------");

boolean exist1 = repo.existsById(101);

boolean exist2 = repo.existsById(109);

System.out.println(exist1 +"--"+ exist2);

}

}

Task:

Work with Oracle DB: Properies??

Work with PostgresDB : Properies??

MySQL Installer

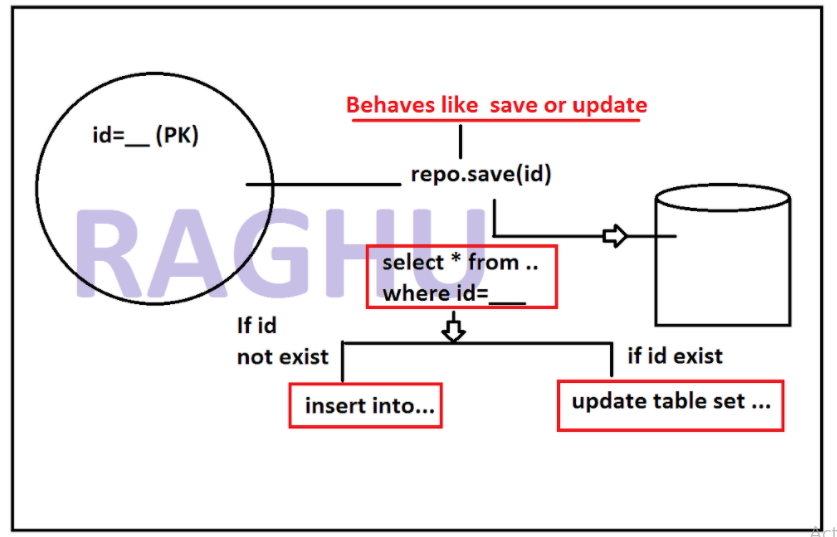
https://dev.mysql.com/downloads/installer/

Download > Next > Next > Finish

PostgressDB:

https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

JDK 14: https://www.oracle.com/in/java/technologies/javase-downloads.html

****

Date : 04/08/2020

Spring Boot 7:50

Mr. RAGHU

-----------------------------------------------------------

CrudRepository Operations:-

count():long

=> it returns no.of rows exist in database table as long type

SQL: select count(\*) from emptab;

\*) Deleting Rows in table:

a. deleteById(id):void

This method is used to delete one row from db table, based on PK.

ex: repo.deleteById(40);

b. delete(object):void [Sun-JPA Spec]

This method takes object that has primary key.

Ex:

Employee emp = new Employee();

emp.setEmpId(10);

repo.delete(emp);

c. deleteAll(Iterable<T> objects):void [Sun-JPA Spec]

This method is used to delete multiple rows at a time using

objects having PK

Ex:

repo.deleteAll(

List.of( //object having PK

new Employee(105),

new Employee(101)

)

);

d. deleteAll():void

This method is used to delete all rows from DB table

Ex: repo.deleteAll();

--code-----------

1. Model class

package in.nareshit.raghu.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.NonNull;

import lombok.RequiredArgsConstructor;

@Data

@NoArgsConstructor

@RequiredArgsConstructor

@AllArgsConstructor

@Entity

public class Employee {

@Id

@NonNull

private Integer empId;

private String empName;

private Double empSal;

private String empDept;

}

2. Repository interface

package in.nareshit.raghu.repo;

import org.springframework.data.repository.CrudRepository;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends CrudRepository<Employee, Integer> {

}

3. Runner class

package in.nareshit.raghu.runner;

import java.util.Arrays;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

repo.saveAll(

Arrays.asList(

new Employee(101, "AJAY", 250.0, "DEV"),

new Employee(102, "SAM", 350.0, "DEV"),

new Employee(103, "SYED", 280.0, "BA"),

new Employee(104, "VIJAY", 390.0, "QA"),

new Employee(105, "AHMED", 890.0, "BA"),

new Employee(106, "JAI", 360.0, "QA")

)

);

System.out.println("------COUNT--------");

System.out.println(repo.count());

System.out.println("------DELETE--------");

//Data JPA

repo.deleteById(103);

//Sun-JPA-Spec

Employee emp = new Employee();

emp.setEmpId(104);

repo.delete(emp); //object having PK

//Sun-JPA-Spec - multiple delete

repo.deleteAll(

List.of( //object having PK

new Employee(105),

new Employee(101)

)

);

repo.deleteAll();

repo.findAll().forEach(System.out::println);

}

}

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

Working with PagingAndSortingRepository

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

PagingAndSortingRepository interface extends internally CrudRepository.

PagingAndSortingRepository provides 2 methods:

a. findAll(Sort):Iterable<T>

b. findAll(Pageable):Page<T>

a. findAll(Sort):Iterable<T>

=> Here Sort is a class that is used to fetch data based on given property /column

in default ASC order.

=> To Change Order to DESC, use enum Direction having two values (ASC,DESC).

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.data.domain.Sort;

import org.springframework.data.domain.Sort.Direction;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

/\*

repo.saveAll(

Arrays.asList(

new Employee(101, "AJAY", 250.0, "DEV"),

new Employee(102, "SAM", 350.0, "DEV"),

new Employee(103, "SYED", 280.0, "BA"),

new Employee(104, "VIJAY", 390.0, "QA"),

new Employee(105, "AHMED", 890.0, "BA"),

new Employee(106, "JAI", 360.0, "QA")

)

);\*/

System.out.println("------COUNT--------");

repo.findAll(

Sort.by("empName")

).forEach(System.out::println);;

repo.findAll(

Sort.by(Direction.DESC,"empName")

).forEach(System.out::println);

}

}

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

Pagination: It is a process of fetching few rows from DB table using Pagination inputs.

Inputs:

page size (int) [Starts from Zero]

page number

//Page Number, page Size

Pageable pageable = PageRequest.of(2, 4);

=> here Pageable is interface that is implemented using PageRequest(C)

Output:

Data(List) and other details (like isFirst,isLast,hasNext..)

--Example#1----

package in.nareshit.raghu.runner;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

repo.findAll().forEach(System.out::println);

System.out.println("-------------");

//input for Pagination

//pageNum,pageSize

Pageable pageable = PageRequest.of(6, 4);

//execute and get output

Page<Employee> page = repo.findAll(pageable);

//print data

List<Employee> list = page.getContent();

list.forEach(System.out::println);

}

}

---Example#2---------

package in.nareshit.raghu.runner;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

repo.findAll().forEach(System.out::println);

System.out.println("-------------");

//input for Pagination

//pageNum,pageSize

Pageable pageable = PageRequest.of(2, 3);

//execute and get output

Page<Employee> page = repo.findAll(pageable);

//print data

List<Employee> list = page.getContent(); //page data

list.forEach(System.out::println);

//---output details-------

System.out.println("Page Has Data?"+page.hasContent()); //boolean

System.out.println("First?"+page.isFirst()); //boolean

System.out.println("Last?"+page.isLast()); //boolean

System.out.println("Next?"+page.hasNext()); //boolean

System.out.println("Prev?"+page.hasPrevious()); //boolean

System.out.println("PageNumber?"+page.getNumber());

System.out.println("Size?"+page.getSize()) ; //page size

System.out.println("Total Pages?"+page.getTotalPages()) ; //Total Pages

System.out.println("Total rows?"+page.getTotalElements()) ; //Total Rows

System.out.println("Page Count?"+page.getContent().size()) ; //Total Rows

}

}

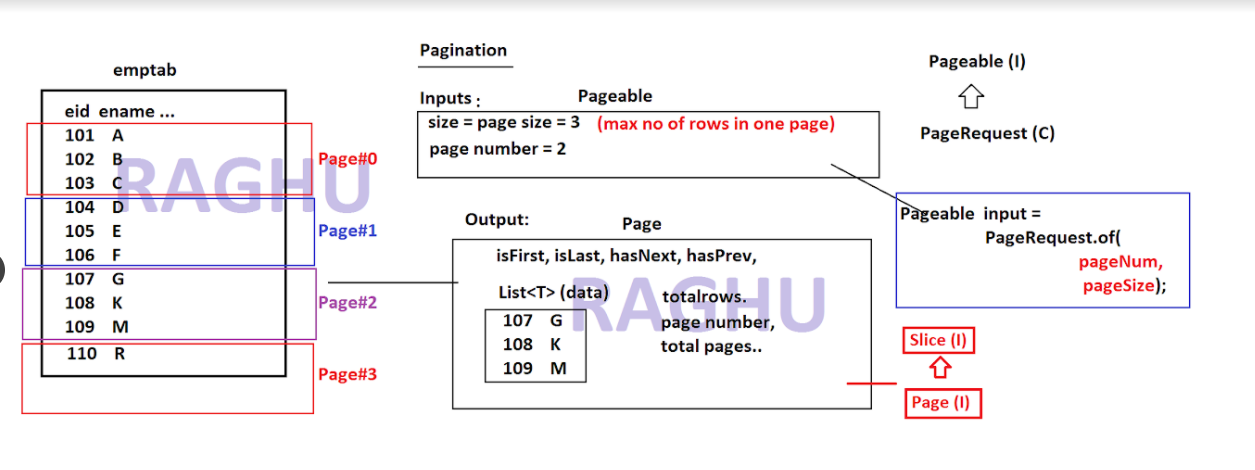
Q) What is the difference between findAll() and

findAll(PageRequest.of(pageNum, pageSize))?

A) findAll() - select all rows

findAll(PageRequest.of(pageNum, pageSize))

It will select few rows based on pagesize given.

****

Date : 05/08/2020

Spring Boot 7:50PM

Mr. RAGHU

------------------------------------------------------------

Spring Boot Data JPA :

1. JpaRepoitory

findAll():List<T>

This method returns all rows from DB table as List Format.

findAll(Example<T>):List<T>

This method is used to fetch rows by generating dynamic SQL

with no-null object input.

=> It will compare non-null values(given in object) with every row in DB table

---Runner class---

package in.nareshit.raghu.runner;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.data.domain.Example;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

repo.findAll().forEach(System.out::println);

System.out.println("-------------");

//----------------------------------------

//1. create Employe with empDept

Employee emp = new Employee();

emp.setEmpDept("Dev");

//2. convert to Example object using static method of(prob)

Example<Employee> example = Example.of(emp);

//3. execute using findAll

List<Employee> list = repo.findAll(example);

//4. print data

list.forEach(System.out::println);

}

}

------------------------------------------------------------------

2. Custom Query Method

So,far we used pre-defined methods given by Repository interface.

This time we want to create our own SQL queries for Customize.

So, use below concepts.

a) findBy [abstract method converted SQL]

b) @Query (HQL/JPQL/SQL) [Manual Query]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

a) findBy [abstract method converted SQL]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This is one abstract method , must be defined in Our Repository interface

it will be converted into one equal SQL query at runtime.

Basic Syntax:

<RT> findBy<VariablesAndConditions>(<Params>);

=> This concept supports only select operation.

=> Non-select operations like DELETE/INSERT/UPDATE not supported by findBy

=> findBy Variable name must match with model class variable name

else : PropertyReferenceException:

No property eSal found for type Employee! Did you mean 'empSal'?

=> If we did not specify any keyword for condition then default is = (Is,Equals)

SQL: Select ename which has 3 chars exactly

\_ =(one underscore) = 1 char

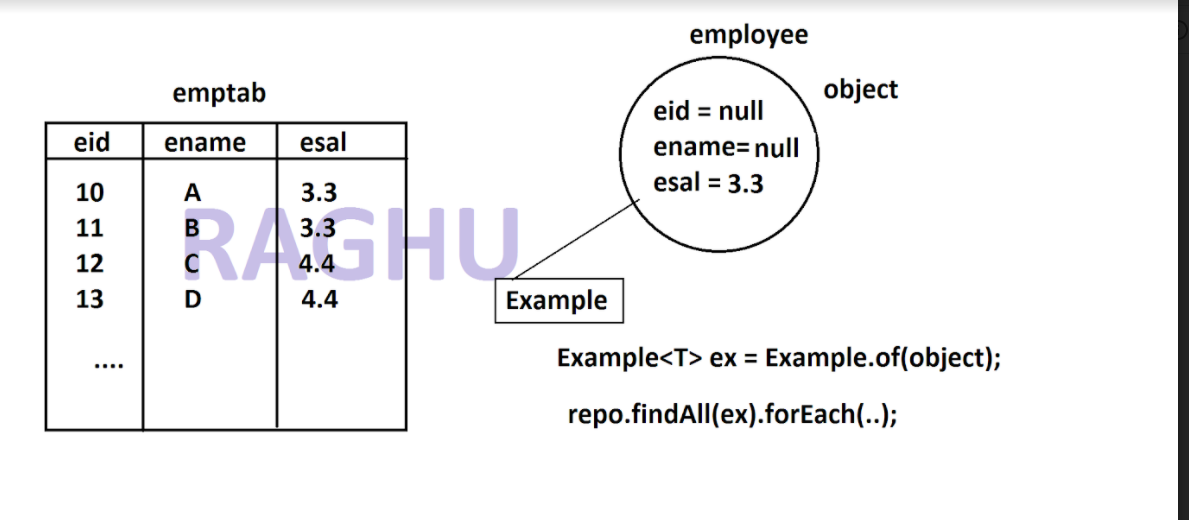
SELECT \* FROM EMPTAB WHERE ename like '\_\_\_';

SQL: select ename , that starts with A later any no.of chars

ELECT \* FROM EMPTAB WHERE ename like 'A%';

SQL: select ename, that contains at least 3 chars (3 or more)

SELECT \* FROM EMPTAB WHERE ename like '\_\_\_%';

****

Date : 06/08/2020

Spring Boot 7:50PM

Mr. RAGHU

---------------------------------------------------

findBy Examples:-

Dep: Spring Data Jpa, Lombok, MySQL.

1. Model class

package in.nareshit.raghu.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.NonNull;

import lombok.RequiredArgsConstructor;

@Data

@NoArgsConstructor

@RequiredArgsConstructor

@AllArgsConstructor

@Entity

public class Employee {

@Id

@NonNull

private Integer empId;

private String empName;

private Double empSal;

private String empDept;

}

2. application.yml

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

password: root

url: jdbc:mysql://localhost:3306/boot8pm

username: root

jpa:

database-platform: org.hibernate.dialect.MySQL8Dialect

hibernate:

ddl-auto: update

show-sql: true

3. Repository

package in.nareshit.raghu.repo;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

//SQL: select \* from emptab where dept=?

List<Employee> findByEmpDeptEquals(String empDept);

//List<Employee> findByEmpDeptIs(String empDept);

//List<T> findByVariable(DataType param)

//SQL: select \* from emptab where name=?

List<Employee> findByEmpName(String empName);

//SQL: select \* from emptab where esal < ?

// findBy<VariableName><KeyWord>

List<Employee> findByEmpSalLessThan(Double empSal);

//SQL: select \* from emptab where esal <= ?

List<Employee> findByEmpSalLessThanEqual(Double empSal);

//SQL: select \* from emptab where esal > ?

List<Employee> findByEmpSalGreaterThan(Double empSal);

//SQL: select \* from emptab where dept<>? //!=

List<Employee> findByEmpDeptNot(String empDept);

//SQL: select \* from emptab where ename is null

List<Employee> findByEmpNameIsNull();

//SQL: select \* from emptab where ename is not null

List<Employee> findByEmpNameNotNull();

//List<Employee> findByEmpNameIsNotNull();

//---------------------------------------------------

//SQL: select \* from emptab where ename like ?

List<Employee> findByEmpNameLike(String empName);

//SQL: select \* from emptab where ename not like ?

List<Employee> findByEmpNameNotLike(String empName);

//-------------2nd part-----------------------------

//SQL: select \* from emptab where ename like ?% (starts with) ex: AB%,MN%

//For your input % Symbol is added at the end

//Ex input AB -> AB% (Starter with your input)

List<Employee> findByEmpNameStartingWith(String empName);

//SQL: select \* from emptab where ename like %? (ends with) ex: %AB,%MN

//For your input % Symbol is added at the starting

//Ex input AB -> %AB (ends with your input)

List<Employee> findByEmpNameEndingWith(String empName);

//SQL: select \* from emptab where ename like %?%

//Input : AB -> %AB% (Symbol added before and after)

List<Employee> findByEmpNameContaining(String empName);

}

4. Runner class

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

//repo.findByEmpDept("QA").forEach(System.out::println);

//repo.findByEmpName("JAI").forEach(System.out::println);

//repo.findByEmpSalLessThanEqual(9000.9).forEach(System.out::println);

//repo.findByEmpSalGreaterThan(250.9).forEach(System.out::println);

//repo.findByEmpDeptEquals("QA").forEach(System.out::println);

//repo.findByEmpDeptNot("QA").forEach(System.out::println);

//repo.findByEmpNameIsNull().forEach(System.out::println);

//repo.findByEmpNameNotNull().forEach(System.out::println);

//repo.findByEmpNameLike("A%").forEach(System.out::println);

//repo.findByEmpNameNotLike("A%").forEach(System.out::println);

//repo.findByEmpNameStartingWith("A").forEach(System.out::println);

//repo.findByEmpNameEndingWith("C").forEach(System.out::println);

repo.findByEmpNameContaining("A").forEach(System.out::println);

}

}

===========Part-2====================================================

\*) Repository

package in.nareshit.raghu.repo;

import java.util.Collection;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

//SQL: select \* from emptab where ename=? and esal=?

List<Employee> findByEmpNameAndEmpSal(String empName,Double empSal);

//SQL: select \* from emptab where ename=? or esal<?

List<Employee> findByEmpNameOrEmpSalLessThan(String empName,Double empSal);

//SQL: select \* from emptab where ename is not null or esal>=?

List<Employee> findByEmpNameIsNotNullOrEmpSalGreaterThanEqual(Double empSal);

//SQL: select \* from emptab where ename is not null or esal>=? order by dept desc

List<Employee> findByEmpNameIsNotNullOrEmpSalGreaterThanEqualOrderByEmpDeptDesc(Double empSal);

List<Employee> findByEmpNameIsNotNullOrEmpSalGreaterThanEqualOrderByEmpDept(Double empSal); //default is ASC

//SQL: select \* from emptab where eid between ? and ?

List<Employee> findByEmpIdBetween(Integer id1,Integer id2);

//SQL: select \* from emptab where eid in (?,?,?)

//Pass input for this using Collection Type (List,Set..)

List<Employee> findByEmpIdIn(Collection<Integer> input);

//SQL: select \* from emptab where eid not in (?,?,?)

List<Employee> findByEmpIdNotIn(Collection<Integer> input);

}

\*) Runner

package in.nareshit.raghu.runner;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

//repo.findByEmpNameAndEmpSal("ABC", 5580.0).forEach(System.out::println);

//repo.findByEmpNameOrEmpSalLessThan("ABC", 5580.0).forEach(System.out::println);

//repo.findByEmpNameIsNotNullOrEmpSalGreaterThanEqual(5580.0).forEach(System.out::println);

//repo.findByEmpNameIsNotNullOrEmpSalGreaterThanEqualOrderByEmpDeptDesc(5580.0).forEach(System.out::println);

//repo.findByEmpNameIsNotNullOrEmpSalGreaterThanEqualOrderByEmpDept(5580.0).forEach(System.out::println);

//repo.findByEmpIdBetween(103,107).forEach(System.out::println);

//repo.findByEmpIdIn(List.of(101,103,105,107)).forEach(System.out::println);

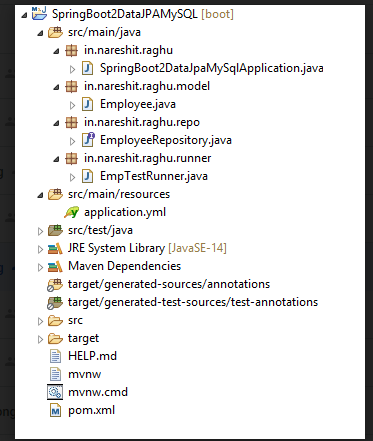
repo.findByEmpIdNotIn(List.of(101,103,105,107)).forEach(System.out::println);

}

}

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

Task : Sequences, Procedures , Joins Examples

****

Date : 07/08/2020

Spring Boot 7:50PM

Mr. RAGHU

---------------------------------------------

findBy : It is used to define one abstract method that is converted into one SQL

query with restrictions(where condition) and projections(select clause).

findBy - Projections : [selecting required columns, not all]

-> Projections are two types:

a) static projections : Every time returns same Type View interface

b) dynamic projection : It returns columns/variables with selected view while calling method

#1. Define one interface with any name inside Repository interface

Syntax:

interface \_\_\_\_{

}

#2. select variables from model classes which you want to display

#3. Write selected variables get methods inside new interface

Syntax:

DT getVariableName();

#4. use that interface as findBy method Return Type

---------Core Java-------------------------------------

\*) Generics

\*) Reflection API

\*) Proxy Pattern

-------------------------------------------------------

--Example--

Repository interface:

--------------------------

package in.nareshit.raghu.repo;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

//SQL: select ename,esal from emptab where dept=?

//List<MyViewOne> findByEmpDept(String empDept);

interface MyViewOne {

String getEmpName();

String getEmpDept();

}

//SQL: select eid,esal from emptab where dept=?

List<MyViewTwo> findByEmpDept(String empDept);

interface MyViewTwo {

Integer getEmpId();

Double getEmpSal();

}

}

Runner class:-

----------

package in.nareshit.raghu.runner;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

import in.nareshit.raghu.repo.EmployeeRepository.MyViewTwo;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

/\*

//repo.findByEmpDept("DEV").forEach(System.out::println);

List<MyViewOne> list = repo.findByEmpDept("DEV");

for(MyViewOne m: list) {

System.out.println(m.getEmpName()+"-"+m.getEmpDept());

}

\*/

List<MyViewTwo> list = repo.findByEmpDept("DEV");

for(MyViewTwo m: list) {

//System.out.println(m.getClass());

System.out.println(m.getEmpId()+"-"+m.getEmpSal());

}

}

}

---------------------Dynamic Projection-----------------

\*) Every time while calling method we can specify what type of output view should come.

Must be provided as \_\_.class param.

First Define Generic method using below syntax inside Repository interface

<T> List<T> findBy<VariablesAndConditions>(<Params>, Class<T> cls);

Ex:

//SQL: select eid,esal from emptab where dept=?

<T> List<T> findByEmpDept(String empDept,Class<T> clz);

--Repository interface--

package in.nareshit.raghu.repo;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

//SQL: select ename,esal from emptab where dept=?

//List<MyViewOne> findByEmpDept(String empDept);

interface MyViewOne {

String getEmpName();

String getEmpDept();

}

//SQL: select eid,esal from emptab where dept=?

<T> List<T> findByEmpDept(String empDept,Class<T> clz);

interface MyViewTwo {

Integer getEmpId();

Double getEmpSal();

}

}

---Runner class--

package in.nareshit.raghu.runner;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

import in.nareshit.raghu.repo.EmployeeRepository.MyViewOne;

import in.nareshit.raghu.repo.EmployeeRepository.MyViewTwo;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

//while method calling provide class input type

List<MyViewOne> list = repo.findByEmpDept("DEV", MyViewOne.class);

for(MyViewOne one :list) {

System.out.println( one.getEmpName() +" - " + one.getEmpDept());

}

//-------------------------

List<MyViewTwo> list2 = repo.findByEmpDept("DEV",MyViewTwo.class);

for(MyViewTwo two:list2) {

System.out.println(two.getEmpId() + " - " + two.getEmpSal());

}

}

}

--------------Core----------------------------------------------

String s = "HI";

int m =10;

Object ob = new <anyType>();

Class clz = \_\_\_.class;

Class clz = Class.forName("package.className");

--Core java Concept#2 Methods--

public void show(Product a){ }

public void show(Employee e){ }

public void show(Admin a){ }

--Sol#1 (Not a Good Approch, Upasting/Downcasting required)

public void show(Object ob){ }

--Sol#2 (Create one Super Type and provide,Upasting/Downcasting required)

public void show(Data ob){ }

Product extends Data

Employee extends Data

Admin extends Data

--Sol#3 Generic Method (JDK1.5)

public <T> void show(Any Param,Class<T> clz){ }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Query : Custom SQL/HQL/JPQL queries

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Query is used to execute programmer specifi manual SQL/HQL-JPQL queries.

-> It support Native SQL(SQL related to Database)

-> It supports HQL/JPQL (Hibernate Query Language, Java Persistency Query Language)

-> Tt supports JOINS (fetch data from multiple table using single SQL)

-> It supports Non-select (and select) operations update and delete.

-> It support Parameters(input) (runtime inputs)

a) Positional parameters ?1,?2,?3...etc (Old Style)

b) Named Parameters :a,;xyz,:empId, :helloOne..etc (new Style)

-> Supoprts Projections (3 types 1-columns, multi-columns, all-columns).

-> It supports calling Procedures/Functions.

SQL -> constrcuted using Table and Column names.

-> It may be/is Database Dependent

-> Case-insensitive (SELECT,select,Select,sElECt are same)

HQL/JPQL -> constrcuted using ClassName and Variable names

-> It is Database Independent (Bcoz of Dialect HQL/JPQL ->SQL)

-> Partially case-sensitive (SQL words are Case-insensitive,

but java words :className,variable name are case-sensitive)

=> To declare parameters use either Positional (?1,?2,?3...etc) Paramter (or)

Named Parameters (:a,:xyz,:mno..etc)

=> Positional Params are not allowed with out position number:

Like ?, ? , ? are not allowed in Data JPA, they are allowed in JDBC.

Then Data JPA throws Exception like:

IllegalArgumentException: JDBC style parameters (?) are not supported for JPA queries.

=> params can be provided in any order (?3, ?4, ?2, ?1)

without any gap. If gap numbers are provided Ex: ?1, ?3 (we missed ?2)

then :

QueryException: Unexpected gap in ordinal parameter labels [1 -> 3] : [1,3]

-------------named params-----------------------------------------------------------

=> Inplace of ?1, ?2..., we can provide names for positions using :<name> (colon-name)

=> This NamedParameter must match with method parameter, else

IllegalStateException: Using named parameters for method

public abstract java.util.List in.nareshit.raghu.repo.EmployeeRepository.getEmpBySal(java.lang.Double) but parameter 'Optional[b]' not found in annotated query

'SELECT e FROM Employee e WHERE e.empSal>:a'!

Ex:#1 (Vaild)

@Query("SELECT e FROM Employee e WHERE e.empSal>:sal")

List<Employee> getEmpBySal(Double sal);

Ex:#2 (Invaild) -> Named Param :sal not matched with input

@Query("SELECT e FROM Employee e WHERE e.empSal>:sal")

List<Employee> getEmpBySal(Double input);

Ex:#3 To make Ex#2 Valid use one Annotation @Param

that binds methodParam with NamedParam

@Query("SELECT e FROM Employee e WHERE e.empSal>:sal")

List<Employee> getEmpBySal(@Param("sal")Double input);

Ex#4 it is vaild

@Query("SELECT e FROM Employee e WHERE e.empSal>:sal")

List<Employee> getEmpBySal(@Param("sal")Double sal);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

---------------Code------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

===Repository Interface==

package in.nareshit.raghu.repo;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

//SQL->HQL/JPQL

//tableName->className, columnName->variableName

@Query("SELECT e FROM Employee e")

List<Employee> getAllEmps();

//-------------Positional Params-------------------

@Query("SELECT e FROM Employee e WHERE e.empDept=?1")

List<Employee> getEmpByDept(String a);

//?1 -1st Param(input), ?2-2nd param(input)

@Query("SELECT e FROM Employee e WHERE e.empDept=?1 or e.empSal<?2")

List<Employee> getEmpByDept(String a,Double b);

//Vaild one -> b input is given to empDept, a-> input is given to empSal

@Query("SELECT e FROM Employee e WHERE e.empDept=?2 or e.empSal<?1")

List<Employee> getEmpDept(Double b,String a);

//-------------Named Params-------------------

//----- :<anyName> //even variable name also valid

@Query("SELECT e FROM Employee e WHERE e.empSal>:sal")

List<Employee> getEmpBySal(@Param("sal") Double esal);

}

----Runner class-----------

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

//repo.getAllEmps().forEach(System.out::println);

//repo.getEmpByDept("DEV").forEach(System.out::println);

//repo.getEmpByDept("DEV",900.0).forEach(System.out::println);

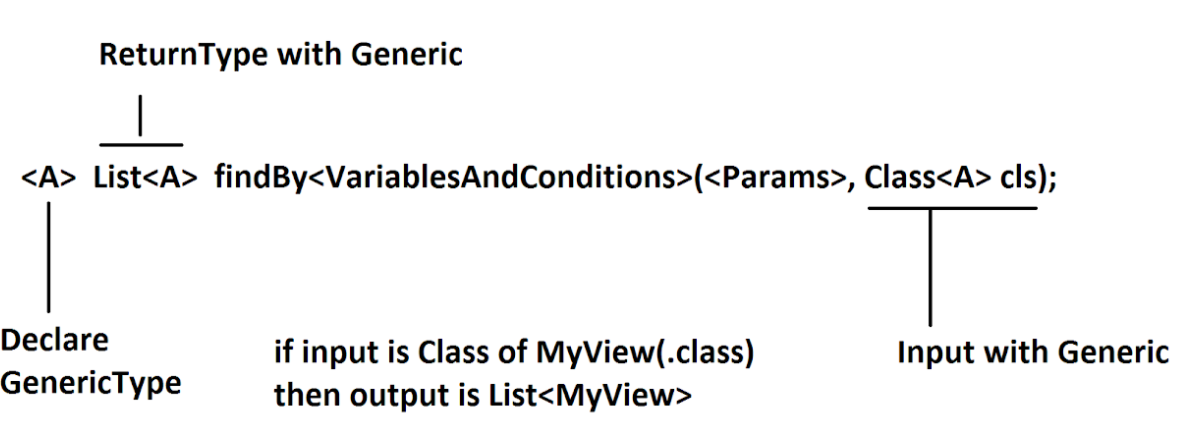
//repo.getEmpDept(900.0,"DEV").forEach(System.out::println);

repo.getEmpBySal(900.0).forEach(System.out::println);

}

}

------------------------------------

****

Date : 08/08/2020

Spring Boot 7:50PM

Mr. RAGHU

----------------------------------------------------------

https://www.youtube.com/c/NareshIT/search?query=murali

FB:

https://www.facebook.com/groups/thejavatemple/

Email:

javabyraghu@gmail.com

Github:

http://github.com/javabyraghu

Youtube Playlist:

https://www.youtube.com/watch?v=EA43S5R8LSc&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr

--------------------------------------------------------------

Spring Boot with Docker - 10th Aug - 7AM

https://www.zoom.us/j/96480571571

Agile (Theory) - 10th Aug - 1PM

https://www.zoom.us/j/98674341611

@Query("") :-

Positional Params ?1,?2...

Named Params : :a,:xyz,:empId..etc

=> While writing query using @Query annotation

we need to provide fully Qualified Model classname

ie Package + ClassName.

But Providing package Name is optional

-> Both Examples are vaild. Packagename is scaned by Repository interface using

Generic type param.

Ex#1

SELECT e FROM Employee e

Ex#2

SELECT e FROM in.nareshit.raghu.model.Employee e

------------------------

=> If we did not specify SELECT CLAUSE then it indicates select all columns

--Both are same--

Ex#1

SELECT e FROM Employee e

Ex#2

FROM Employee e

----------------------

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

@Query Projections

-> If we try to write SELECT clause with below options, output types are given as

a. SELECT all columns -> output List<T> , T= Model class

b. SELECT one column -> Output List<DT> , DT = DataType of selected variable/column

c. SELECT multiple columns -> Output List<Object[]>

SQL:

#1 select \* from student -> Output: List<Student>

#2 select sid from student -> Output: List<Integer>

#3 select sname,sfee from student -> Output: List<Object[]>

----Code------

\*)Model class

package in.nareshit.raghu.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.NonNull;

import lombok.RequiredArgsConstructor;

@Data

@NoArgsConstructor

@RequiredArgsConstructor

@AllArgsConstructor

@Entity

public class Employee {

@Id

@NonNull

private Integer empId;

private String empName;

private Double empSal;

private String empDept;

}

\*) application.yml

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

password: root

url: jdbc:mysql://localhost:3306/boot8pm

username: root

jpa:

database-platform: org.hibernate.dialect.MySQL8Dialect

hibernate:

ddl-auto: update

show-sql: true

\*)Repository Interface

package in.nareshit.raghu.repo;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

@Query(" FROM Employee e ")

List<Employee> getMyData();

@Query("SELECT e.empName,e.empId FROM Employee e")

List<Object[]> getVals();

@Query("SELECT e.empSal FROM Employee e")

List<Double> getDetails();

}

\*)Runner class:

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

//repo.getMyData().forEach(System.out::println);

//JDK 1.8 Print using Lambda

//repo.getVals().forEach(ob-> System.out.println(ob[0] + "-" +ob[1]));

//JDK 1.8 Stream API

/\*

repo.getVals()

.stream()

.map(ob-> ob[0] + "-" +ob[1])

.forEach(System.out::println);

\*/

repo.getDetails().forEach(System.out::println);

}

}

\_\_Note\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\* Use List<> Type if query returns more then 1 row is returning.

It means, if Query returns only single row data(Unique Result) then

Do not use List.

\*\* If we are not using List and query returns multi-rows data

then exception:

NonUniqueResultException: query did not return a unique result: 9

\*\* If Singl row result is given then Output Types are:

a. all columns : T (Type)

b. One column : DT (DataType)

c. multiple columns: Object (Object[] casted to Object)

----Core Java--------------------

Q) Is this code valid?

Object ob = new Object[]{10,"A",3.3};

A)

---------------------------------

--Ex:--code----------

\*\* Runner class \*\*

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

@Query(" FROM Employee e WHERE e.empId=:empId")

Employee getMyData(Integer empId);

@Query("SELECT e.empSal FROM Employee e WHERE e.empId=:empId")

Double getDetails(Integer empId);

// Object ob = (Object[])

@Query("SELECT e.empName,e.empId FROM Employee e WHERE e.empId=:empId")

Object getVals(Integer empId);

}

\*\* Repository Interface \*\*

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

//Employee e = repo.getMyData(101);

//Double e = repo.getDetails(101);

Object ob = repo.getVals(101);

Object[] e =(Object[])ob;

System.out.println(e[0]+"-"+e[1]);

}

}

----Core Java-------------------------------------------------------

Q)

Boolean b1 = new Boolean("123");

Boolean b2 = new Boolean("ABC");

sysout(b1);

sysout(b2);

sysout(b1.equals(b2));

A)

---------------------------------------------------------------------

Q) How can define Pure SQL(Native SQLs) using @Query in Data JPA?

A) provide attribute nativeQuery = true then given query behaves like SQL query.

Its default value is false (nativeQuery = false). That indicates given one is HQL/JPQL

//SQL: SELECT emp\_name FROM employee WHERE emp\_name = 'AJAY' ;

@Query(nativeQuery = true,value = "SELECT emp\_id FROM employee WHERE emp\_name=:ename")

Integer getMyInfo(String ename);

Q) Is Native query not recommended? Why?

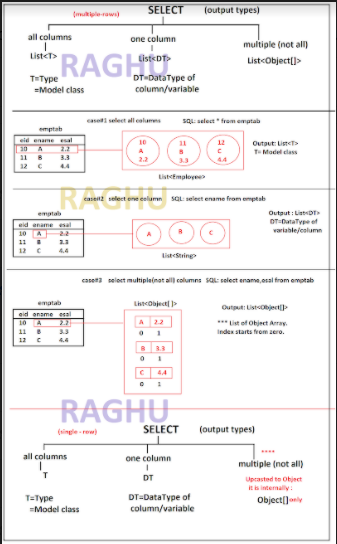
A) In realtime we can not predict which database is selected by

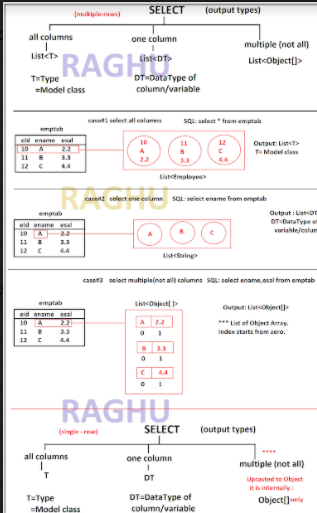
client (or may be database migrations are done)

SO, All sqls in app may not work.

---------------------------------------------------------------------

Task: Define Procedures in MySQLDatabase with examples

****

****

Date : 10/08/2020

Spring Boot 7:50PM

javabyraghu@gmail.com

Mr. RAGHU

----------------------------------------------------

All PDFs:-

https://www.mediafire.com/file/w5x9w5vcmkwkkdv/RaghuSirNareshITJavaPdfs.zip/file

\*) Date and Time

\*) Lob (BLOB and CLOB)

\*) Collections (List,Set, Map)

\*) Association Mapping (HAS-A)

+ Multiplicity [1...1/1...\*/\*...1/\*...\*]

\*) Joins

\*) Procedure Calls

--------------------------------------------------------------

\*) Date and Time :-

Sun JPA has provided [javax.persistence]

Annotation: @Temporal

Enum : TemporalType (DATE, TIME, TIMESTAMP)

--code--

\*\*) application.properties

#datasource

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/boot8pm

spring.datasource.username=root

spring.datasource.password=root

#jpa

spring.jpa.show-sql=true

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=create

-----------------------------

1. Model

package in.nareshit.raghu.model;

import java.util.Date;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Temporal;

import javax.persistence.TemporalType;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@NoArgsConstructor

@Data

@Entity

@AllArgsConstructor

public class Student {

@Id

private Integer stdId;

private String stdCode;

private Double stdFee;

@Temporal(TemporalType.DATE)

private Date dteA;

@Temporal(TemporalType.TIME)

private Date dteB;

@Temporal(TemporalType.TIMESTAMP)

private Date dteC;

}

2. Repository Interface

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Student;

public interface StudentRepository

extends JpaRepository<Student, Integer>

{

}

3. Runner class

package in.nareshit.raghu.runner;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Student;

import in.nareshit.raghu.repo.StudentRepository;

@Component

public class StudentDataRunner implements CommandLineRunner {

@Autowired

private StudentRepository repo;

@Override

public void run(String... args) throws Exception {

repo.save(

new Student(

101, "A", 3.3,

new Date(),

new Date(),

new Date()

)

);

}

}

\*\*\*) If we do not provide any type, then default is taken as TIMESTAMP.

--------------------------------------------------------------------------

Task:

a. LocalDate

b. java.sql types

c. SimpleDateFormat

d. Calendar

----------------------------------------------------------------------

\*) Lob (BLOB and CLOB) [javax.persistence]

BLOB = Binary Large OBject -> To store binary data(images, audio, videos, pdfs..etc)

CLOB = Character Large OBject -> Large Text data.

--Type--

@Lob // byte[] + Lob = BLOB

private byte[] arr;

@Lob // char[] + Lob = CLOB

private char[] arr;

----------Example code---------------

1. Model class

package in.nareshit.raghu.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Lob;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@NoArgsConstructor

@Data

@Entity

@AllArgsConstructor

public class Student {

@Id

private Integer stdId;

private String stdCode;

private Double stdFee;

@Lob

private byte[] stdImg;

@Lob

private char[] stdDesc;

}

2. Repository interface

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Student;

public interface StudentRepository

extends JpaRepository<Student, Integer>

{

}

3. Runner class

package in.nareshit.raghu.runner;

import java.io.FileInputStream;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Student;

import in.nareshit.raghu.repo.StudentRepository;

@Component

public class StudentDataRunner implements CommandLineRunner {

@Autowired

private StudentRepository repo;

@Override

public void run(String... args) throws Exception {

FileInputStream fis = new FileInputStream("F:\\Images\\MultiDb.png");

byte[] img = new byte[fis.available()];

fis.read(img);

String text ="ABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKDABCDSDKLJKJLSHDIHNSIHISHDJSDKD";

char[] desc = text.toCharArray();

repo.save(

new Student(

101, "A", 3.3,

img,

desc

)

);

fis.close();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Collections (List,Set, Map)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-> Collection holds multiple values. Single column can not hold multiple values.

So, JPA says -'ONE CHILD TABLE IS CREATED FOR ONE COLLECTION VARIABLE IN MODEL'.

With min 2-column and max-3 columns (key-column,index column, element column).

--------------------------------------

Non-index Index

--------------------------------------

Set List,Map

--------------------------------------

\*\*\* Index-column is applied only for index type.

=> Note : List created with 3 columns, Key Column, index-column(list-index -starts from zero)

element (list data ) column

Map created with 3 columns, Key Column, index-column(map key) element (map val) column

Set created with 2 columns, Key Column, no index col, element (set values ) column

=> We must provide @ElementCollection annotation over Collection type variable in model class

=> We can even provide optional annotation (tableName, column names)

@CollectionTable(

name="std\_labs" , //tablename

joinColumns = @JoinColumn(name="sid") //key column

)

=> For Element(Data) Column:-

@Column(name="labname") //element column

=> For List index column name use : @OrderColumn(name="pos")

=> For Map index column name use : @MapKeyColumn(name="code")

---code---

1. Model class

package in.nareshit.raghu.model;

import java.util.List;

import java.util.Map;

import java.util.Set;

import javax.persistence.CollectionTable;

import javax.persistence.Column;

import javax.persistence.ElementCollection;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.MapKeyColumn;

import javax.persistence.OrderColumn;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@NoArgsConstructor

@Data

@Entity

@AllArgsConstructor

public class Student {

@Id

private Integer stdId;

private String stdCode;

private Double stdFee;

@ElementCollection

@CollectionTable(

name="std\_subj" , //tablename

joinColumns = @JoinColumn(name="sid") //key column

)

@OrderColumn(name="pos") //index column for List

@Column(name="subject") //element column

private List<String> subj;

@ElementCollection

@CollectionTable(

name="std\_labs" , //tablename

joinColumns = @JoinColumn(name="sid") //key column

)

@Column(name="labname") //element column

private Set<String> labs;

@ElementCollection

@CollectionTable(

name="std\_grades" , //tablename

joinColumns = @JoinColumn(name="sid") //key column

)

@MapKeyColumn(name="code") //index column Map

@Column(name="grade") //element column

private Map<String,String> grades;

}

2. Repository interface

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Student;

public interface StudentRepository

extends JpaRepository<Student, Integer>

{

}

3. Runner class

package in.nareshit.raghu.runner;

import java.util.List;

import java.util.Map;

import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Student;

import in.nareshit.raghu.repo.StudentRepository;

@Component

public class StudentDataRunner implements CommandLineRunner {

@Autowired

private StudentRepository repo;

@Override

public void run(String... args) throws Exception {

repo.save(

new Student(

101, "ABC", 300.0,

//JDK 9 Collections creation

List.of("ENG","MAT","SCI"),

Set.of("BTN","PHY","CHE"),

Map.of(

"ENG","A+",

"MAT","A",

"SCI","B")

)

);

}

}

Q) What are optional annotations in Collection Mapping?

Q) Provide default table,column names if no optional annotaions present?

Q) If model class contains 3 List, 2 Sset variables, then total how many tables created?

Q) Create Employee with eid,ename,esal, projest(set),

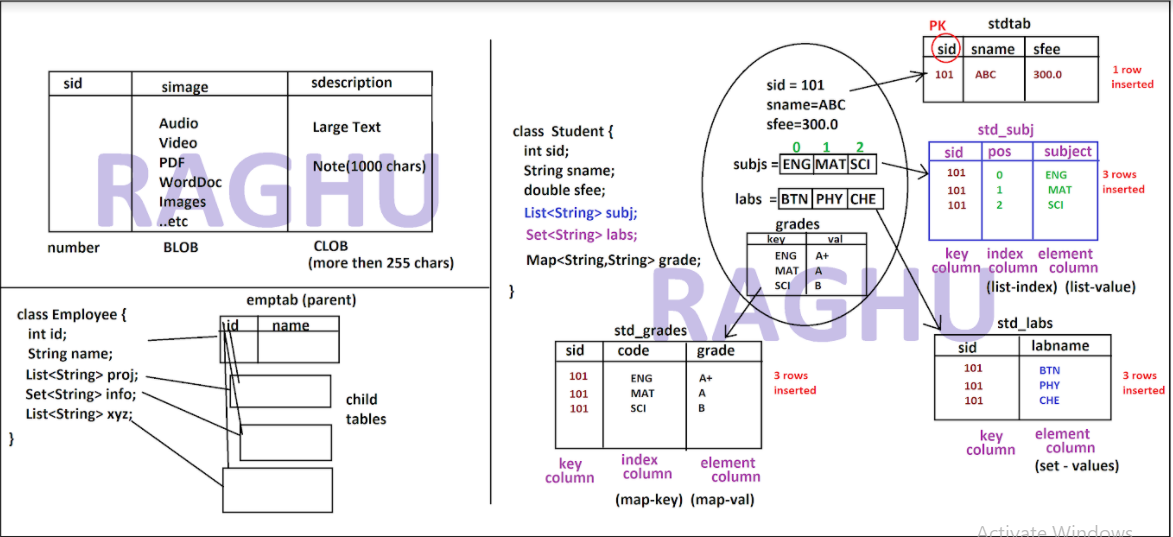
costInfo(List), clients(Map,clientName,clientcode).

Q) What are key constraints? List out them? [Database]

Q) How to link two tables using PK and FK columns?[Database]

Q) What are Joins, write examples? [Database]

----------------------------------------------------------------

****

Date : 11/08/2020

Spring Boot 7:50PM

Mr. RAGHU

---------------------------------------------------

\*) Multiplicity (Database Concept):-

Relation between Two (or more) tables created using one table PrimaryKey

as another table ForeignKey concept.

=> Types(4)

1...1 1...\*

\*...1 \*...\*

Ex:

\*...1 (multiple rows in 1st table are connected to one row in 2nd table)

https://github.com/javabyraghu/HibernateOneToOneFullEx

https://github.com/javabyraghu/HibernateOneToManyFullEx

https://github.com/javabyraghu/HibernateManyToOne

https://github.com/javabyraghu/HibernateManyToManyFullEx

--Note--

=> Sybol \*(many) side FK column is created.

=> Incase of \*...\* one extra table(join table) is created, with 2 Fk Columns

(Join Column, InverseJoin Column).

=> Incase of 1...1 use \*...1 and apply unique=true at many side.

-----------------------------------------------------------------

Non-Collection Based Collection Based

-----------------------------------------------------------------

1...1 1...\*

\*...1 \*...\*

-----------------------------------------------------------------

=> If we are working with Collection type, then use List/Set type variable.

\*\*\* This is not Collection with Pritimive (DO NOT COMPARE WITH LAST SESSION CONCEPT)

--code overview steps--

1. Write two classes

2. Applay HAS-A Relation with multiplicity

3. Apply annotation over HAS-A Relation

4. Apply @JoinColumn or @JoinTable over HAS-A relation

Q) Sir how should we identify parent table and child table?

A)

Car HAS-A Engine

Employee IS-A Person

Company HAS-A Branch

-> Employee, Dept -> Project, Module

-> Admin, Dept -> Product, Brand

-> Customer, Invoice

-> Bird, Animal -> Circkter, Person...etc.

---F/w----------------------------

https://www.mediafire.com/file/w5x9w5vcmkwkkdv/RaghuSirNareshITJavaPdfs.zip/file

JOINS:

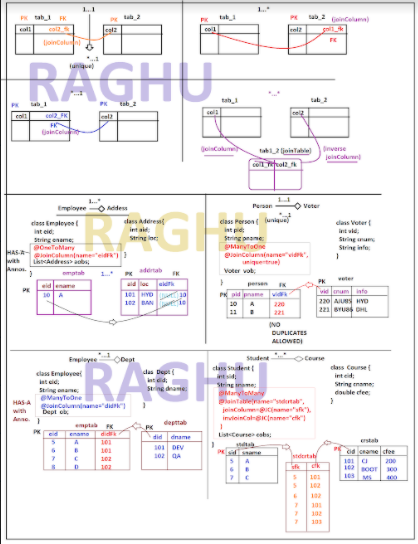
1. Inner JOIN

2. Outer JOIN

a. Left Outer JOIN | Left Join

b. Right Outer JOIN | Right Join

c. Full Outer Join | Full Join

****

Date : 12/08/2020

Spring Boot 7:50PM

Mr. RAGHU

---------------------------------------------------

\*) @Query with non-select operations :-

Here, we need to apply two more annotations, to perform non-select operations

(update/delete), those are: @Transactional and @Modifying

@Modifying : It indicates given Query is non-select, if we did not apply this

then @Query is considered as SELECT Query.

@Transactional : It will commit (modify/update data in db if no mistake in Query and inputs)

or rollback (cancel data modify/update).

\*\*\* If we use Service Layer (@Service) that takes care of TransactionManagement,

even this annotation @Transactional is not required.

----code---

1. Model class

package in.nareshit.raghu.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.NonNull;

import lombok.RequiredArgsConstructor;

@Data

@NoArgsConstructor

@RequiredArgsConstructor

@AllArgsConstructor

@Entity

public class Employee {

@Id

@NonNull

private Integer empId;

private String empName;

private Double empSal;

private String empDept;

}

2. Repository

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Modifying;

import org.springframework.data.jpa.repository.Query;

import org.springframework.transaction.annotation.Transactional;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

@Transactional //commit/rollback

@Modifying // update/delete (not select)

@Query("delete from Employee e where e.empName=:empName")

int removeDataByName(String empName);

@Transactional

@Modifying // update/delete (not select)

@Query("DELETE FROM Employee e WHERE e.empName IS NULL")

int removeNameIsNull();

@Transactional

@Modifying // update/delete (not select)

@Query("UPDATE Employee e SET e.empName=:empName WHERE e.empId=:empId")

int upateNameById(String empName,Integer empId);

}

3. Runner class

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmpTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

public void run(String... args) throws Exception {

//int count = repo.removeDataByName("HYJ");

//int count = repo.removeNameIsNull();

int count = repo.upateNameById("ABCD", 101);

System.out.println(count);

}

}

4. yaml file

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

password: root

url: jdbc:mysql://localhost:3306/boot8pm

username: root

jpa:

database-platform: org.hibernate.dialect.MySQL8Dialect

hibernate:

ddl-auto: update

show-sql: true

===========================Association Mapping Example==============================

1 Model class = 1 table

1 Model class -> create 1 Repository interface

1 Row in table -> 1 object in Code

Coding Order

1. Child Model class

Parent Model class

2. Child Repository

Parent Repository

3. Runner class

HAS-A with Two Repositories

create child objects and save

create parent objects (link with child) and save

https://github.com/javabyraghu?tab=repositories

--code--: many-to-one-----

1. Models

package in.nareshit.raghu.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Entity

public class Dept {

@Id

private Integer deptId;

private String deptName;

}

---

package in.nareshit.raghu.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Entity

public class Employee {

@Id

private Integer empId;

private String empName;

//HAS-A variable with Annotations

@ManyToOne

@JoinColumn(name="didFk")

private Dept dob; //HAS-A

}

2. Repository

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Dept;

public interface DeptRepository

extends JpaRepository<Dept, Integer> {

}

--

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

}

3. Runner

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Dept;

import in.nareshit.raghu.model.Employee;

import in.nareshit.raghu.repo.DeptRepository;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class DataRunner implements CommandLineRunner {

@Autowired

private DeptRepository drepo;

@Autowired

private EmployeeRepository erepo;

@Override

public void run(String... args) throws Exception {

//1 row = 1 object

Dept d1 = new Dept(101, "DEV");

Dept d2 = new Dept(102, "QA");

drepo.save(d1);

drepo.save(d2);

Employee e1 = new Employee(5, "A", d1);

Employee e2 = new Employee(6, "B", d1);

Employee e3 = new Employee(7, "C", d2);

Employee e4 = new Employee(8, "D", d2);

erepo.save(e1);

erepo.save(e2);

erepo.save(e3);

erepo.save(e4);

System.out.println("---------------------");

}

}

---code: many-to-many-------

1. Models

package in.nareshit.raghu.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Entity

public class Course {

@Id

private Integer cid;

private String cname;

private Double cfee;

}

--

package in.nareshit.raghu.model;

import java.util.List;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.JoinTable;

import javax.persistence.ManyToMany;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Entity

public class Student {

@Id

private Integer sid;

private String sname;

@ManyToMany

@JoinTable(name="stdcrstab",

joinColumns = @JoinColumn(name="sfk"),

inverseJoinColumns = @JoinColumn(name="cfk")

)

private List<Course> cobs;

}

2. Repository

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Course;

public interface CourseRepository

extends JpaRepository<Course, Integer> {

}

--

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Student;

public interface StudentRepository

extends JpaRepository<Student, Integer> {

}

3. Runner class

package in.nareshit.raghu.runner;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Course;

import in.nareshit.raghu.model.Student;

import in.nareshit.raghu.repo.CourseRepository;

import in.nareshit.raghu.repo.StudentRepository;

@Component

public class DataTestRunner implements CommandLineRunner {

@Autowired

private CourseRepository crepo;

@Autowired

private StudentRepository srepo;

@Override

public void run(String... args) throws Exception {

Course c1 = new Course(101, "CJ", 200.0);

Course c2 = new Course(102, "BOOT", 300.0);

Course c3 = new Course(103, "MS", 400.0);

crepo.save(c1);

crepo.save(c2);

crepo.save(c3);

//parent objects with child obj links

Student s1= new Student(5, "A", List.of(c1,c2));

Student s2= new Student(6, "B", List.of(c1));

Student s3= new Student(7, "C", List.of(c1,c2,c3));

srepo.save(s1);

srepo.save(s2);

srepo.save(s3);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Working with JOINS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Inner JOIN | JOIN

2. Outer JOIN

a. LFET OUTER JOIN | LEFT JOIN

b. RIGHT OUTER JOIN | RIGHT JOIN

c. FULL OUTER JOIN | FULL JOIN

SYNTAX:

SELECT <P>.<variable>, <C>.<childVariable>

FROM <ParentModel> <P>

[JOIN TYPE]

<P>.<HAS-A-VARIABLE> AS <C>

WHERE <condition>;

---Ex---

Product---<>Brand

class Brand {

bid, bcode, btype..

}

class Product{

pid, pcode,pcost

bob(HAS\_A)

}

Ex#1 select bid,bcode,pcode where pcost is more then 400

SELECT b.bid,b.bcode,p.pcode

FROM Product p

JOIN

p.bob as b

WHERE p.pcost>400

Ex#2

SELECT p.pcode, b.bcode

FROM Product p

INNER JOIN

p.bob as b

WHERE b.btype IS NOT NULL;

---Ex---------------

SELECT e.ename, d.dname

FROM EMPLOYEE e

RIGHT JOIN

e.dob as d

WHERE e IS NULL (where parent is null in right join, child is null in left join)

--------Repository code------

package in.nareshit.raghu.repo;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

//@Query("SELECT e.empName,d.deptName FROM Employee e INNER JOIN e.dob as d") //-2- section

//@Query("SELECT e.empName,d.deptName FROM Employee e LEFT JOIN e.dob as d") //-1- + -2- section

//@Query("SELECT e.empName,d.deptName FROM Employee e LEFT JOIN e.dob as d WHERE d IS NULL") //-1- section

@Query("SELECT e.empName,d.deptName FROM Employee e RIGHT JOIN e.dob as d WHERE e IS NULL") //-2- +-3- section

List<Object[]> getDataA();

}

--Runner class code--

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class DataRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

@Override

public void run(String... args) throws Exception {

repo.getDataA()

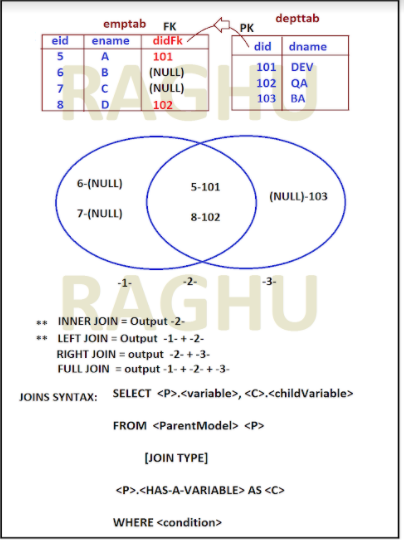
.stream()

.map(ob->ob[0]+"-"+ob[1])

.forEach(System.out::println);

}

}

****

Date : 13/08/2020

Spring Boot 7:50PM

Mr. RAGHU

-------------------------------------------------

\*) Stored Procedures:-

Procedures is a set of Statements those are executed as one block at database side.

[We can compare with our java method]

-Java Method:-

RT MethodName(Parama){

Statements;

}

-Procedure:-

CREATE PROCEDURE <Procedure-Name>(

IN PARAM, OUT PARAM)

BEGIN

STATEMENTS;

END$

=> Procedure supports taking input (IN PARAM) and provides Output even(OUT PARAM).

=> In Procedures , (both IN and OUT) PARAMS are optional.

----------------------------------------------------------------------------------

\*) By using Spring Boot DATA JPA we can execute Stored Procedures using

a. @Query

b.\*\*\* EntityManager

---Code# Insert Data in tables---------

1. Model class

package in.nareshit.raghu.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Entity

@Table(name="employee")

public class Employee {

@Id

@Column(name="eid")

private Integer empId;

@Column(name="ename")

private String empName;

@Column(name="esal")

private Double empSal;

@Column(name="edesig")

private String empDesig;

@Column(name="edept")

private String empDept;

}

2. Repository

package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer>

{

}

3. Runner

package in.nareshit.raghu.runner;

import java.util.Arrays;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import in.nareshit.raghu.model.Employee;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmployeeDataRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

@Override

public void run(String... args) throws Exception {

repo.saveAll(Arrays.asList(

new Employee(101, "ABC", 320.0, "SE", "DEV"),

new Employee(102, "XYZ", 380.0, "SSE", "DEV"),

new Employee(103, "MNO", 450.0, "STM", "QA"),

new Employee(104, "PQR", 160.0, "SAD", "BA")

)

);

}

}

4. application.yml

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

password: root

url: jdbc:mysql://localhost:3306/boot8pm

username: root

jpa:

database-platform: org.hibernate.dialect.MySQL8Dialect

hibernate:

ddl-auto: create

show-sql: true

===========MySQL===========================

PL/SQL : To indicate PL/SQL block we need to write DELIMITER $$ at begining and DELIMITER ; at end.

-Syntax:--

DELIMITER $$

CREATE PROCEDURE <procName>(<IN PARAMS>, <OUT PARAMS>)

BEGIN

END$$

DELIMITER;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ex#1 Fetch all rows (no IN and OUT params)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--MySQL--

DELIMITER $$

CREATE PROCEDURE getAllEmps()

BEGIN

SELECT \* FROM EMPLOYEE;

END$$

DELIMITER;

---CALLING ABOVE PROCEDURE USING DATA JPA--

A) using @Query

S#1

Define one absract method in Repository interface using

below syntax:

@Query(nativeQuery = true, value = "{call <procName>()}")

Ex:

package in.nareshit.raghu.repo;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer>

{

@Query(nativeQuery = true, value = "{call getAllEmps()}")

List<Employee> getDataA();

}

S#2 Execute using Runner class

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmployeeDataRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

@Override

public void run(String... args) throws Exception {

/\*repo.saveAll(Arrays.asList(

new Employee(101, "ABC", 320.0, "SE", "DEV"),

new Employee(102, "XYZ", 380.0, "SSE", "DEV"),

new Employee(103, "MNO", 450.0, "STM", "QA"),

new Employee(104, "PQR", 160.0, "SAD", "BA")

)

);\*/

System.out.println("---USING @QUERY------");

repo.getDataA().forEach(System.out::println);

}

}

B) Using EntityManager (No Method in Repository)

S#1 Runner class code

package in.nareshit.raghu.runner;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.StoredProcedureQuery;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

@Component

public class EmpProcCallRunner implements CommandLineRunner{

@Autowired

private EntityManager em;

@Override

public void run(String... args) throws Exception {

System.out.println("---FROM JPA STYLE---");

//#1. Create StoredProcedure Query

StoredProcedureQuery proc = em.createStoredProcedureQuery("getAllEmps",Employee.class);

//#2. Execute Query

List<Employee> list = proc.getResultList();

//#3. print data

list.forEach(System.out::println);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ex#2 Fetch few rows with Input (IN Used and NO OUT params)

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

DELIMITER $$

CREATE PROCEDURE getEmpByDept(IN emp\_dept VARCHAR(20))

BEGIN

SELECT \* FROM EMPLOYEE e where e.edept = emp\_dept ;

END$$

DELIMITER ;

--code---

A) Using @Query : In this case just add symbol ? inside proc call in Repository

S#1 Repository

package in.nareshit.raghu.repo;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import in.nareshit.raghu.model.Employee;

public interface EmployeeRepository

extends JpaRepository<Employee, Integer> {

@Query(nativeQuery = true, value = "{call getEmpByDept(?)}")

List<Employee> getDataB(String deptName);

}

S#2 Runner class code

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.EmployeeRepository;

@Component

public class EmployeeDataRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

@Override

public void run(String... args) throws Exception {

System.out.println("---USING @QUERY------");

repo.getDataB("DEV").forEach(System.out::println);

}

}

B) Using EntityManager

In this case, we need to add two extra steps, given as

-> Register Param (provide param name,datatype, param type to data JPA)

-> Pass input value for Param to call procedure

---Runner class code---------

package in.nareshit.raghu.runner;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.ParameterMode;

import javax.persistence.StoredProcedureQuery;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Employee;

@Component

public class EmpProcCallRunner implements CommandLineRunner{

@Autowired

private EntityManager em;

@Override

public void run(String... args) throws Exception {

System.out.println("---FROM JPA STYLE---");

//#1. Create StoredProcedure Query

StoredProcedureQuery proc =

em.createStoredProcedureQuery("getEmpByDept",Employee.class);

//#1.(A) register input param

proc.registerStoredProcedureParameter("emp\_dept", String.class, ParameterMode.IN);

//#1.(B) pass input value in param place

proc.setParameter("emp\_dept", "DEV");

//#2. Execute Query

List<Employee> list = proc.getResultList();

//#3. print data

list.forEach(System.out::println);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ex#3 Fetch count of data using IN and OUT params

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DELIMITER $$

CREATE PROCEDURE getEmpByDeptCount(IN emp\_dept VARCHAR(20), OUT ecount INT)

BEGIN

SELECT COUNT(\*) INTO ecount FROM EMPLOYEE e WHERE e.edept = emp\_dept ;

END$$

DELIMITER ;

--code---

-Using EntityManager-

\*) If Output Default is given as select clause a multiple values, that can be accessed using

a method : getResultList();

But this time is not List, it is one value. So, call method :execute(),

then read value using param name: getOutputParameterValue(paramName)

that returns Object data. Cast to our specific type.

package in.nareshit.raghu.runner;

import javax.persistence.EntityManager;

import javax.persistence.ParameterMode;

import javax.persistence.StoredProcedureQuery;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

@Component

public class EmpProcCallRunner implements CommandLineRunner{

@Autowired

private EntityManager em;

@Override

public void run(String... args) throws Exception {

System.out.println("---FROM JPA STYLE---");

//#1. Create StoredProcedure Query

StoredProcedureQuery proc = em.createStoredProcedureQuery("getEmpByDeptCount");

//#1.(A) register input param

proc.registerStoredProcedureParameter("emp\_dept", String.class, ParameterMode.IN);

proc.registerStoredProcedureParameter("ecount", Integer.class, ParameterMode.OUT);

//#1.(B) pass input value in param place

proc.setParameter("emp\_dept", "DEV");

//#2. Execute Query

proc.execute(); //It is not List Result. Single value.

//#3. read and print data

int count = (Integer) proc.getOutputParameterValue("ecount");

System.out.println(count);

}

}

Q) Is using Stored Procdecures recomanded in realtime projects?

A) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.