Date : 28/09/2020

Spring Boot 7AM

Mr. RAGHU

**-------------------------------------------------**

**ORM : Object Relational Mapping (Theory)**

=> ORM says 'Perform Database Operations in Object Format'.

save(object), update(object),delete(object), get(id):object

all operations are executed using objects (input/output).

=> We need to follow Mapping Rule.

class -----mapped with ----- table

variable -----mapped with ----- column

then

object <<---->> Row (No SQL query given by programmer).

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

**Spring Data using JPA with Spring Boot**

Database : 2 types SQL Based(Oracle,MySQL..) , NoSQL Based(MongoDB, Redis, Casandra..etc)

=> Spring Data supports performing database operations with both types of databases.

(SQL Based and NoSQLbased).

=> For SQL based Database operations are done using JPA(Java Persistency API).

Q) What is the difference between below concepts?

JDBC --- Perform Database operations using Java Database Connectivity API with SQL

ORM --- Theory , that says do DB operations in Object format

JPA -- Specification (Interfaces, Annotations,...) given by SunMicrosystem/Oracle

Hibernate -- Implementation of JPA.

Spring JDBC -- Perform Database operations using JDBC + SQL with JdbcTemplate.

Spring ORM --- do DB operations using JPA with Hibernate using Template programming

But Configuration is required.

Spring Data using Boot: do DB operations using JPA with any vendor(3rd party).

AutoConfiguration, Code Generated for basic operations.

RAD - Rapid Application Development. (Less Lines of Code, faster coding).

\*) As of now, Any Database programming finally executing as JDBC(SQL) logic only,

even any webapplication finally executed as Servlets concept only.

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

**ORM/JPA Properties**

a) dialect : It is a class, that generates SQL query when we perform database operation.

SQL queries are database dependent. So, do not use a direct SQL query in application

ie one SQL working fine with Oracle, may or may not work with SQLServer,MySQL..etc

Use Dialect in applications, that generates SQL based on database.

Few Dialects: Oracle10gDialect, MySQL8Dialect, ..etc

\*) For every database one dialect class is provided by JPA/Vendors..

Link:

https://docs.jboss.org/hibernate/orm/5.2/javadocs/org/hibernate/dialect/package-summary.html

b) show-sql : It is boolean property. Do display/print generated SQL on console/log file

provide its value as true, default is false.

c) hibernate-ddl.auto : Tables/Sequences..etc DB Schema can be created manually by

programmer or even ORM can create same. It has possible values

-> hibernate-ddl.auto = create : Hibernate create new table every time. If old table

exist then drop old table and create new table.

-> hibernate-ddl.auto = update : Hibernate create new table if table not exist,

else use same.

-> hibernate-ddl.auto = validate : Hibernate does nothing, all schema design must be

handled by programmer manually.

-> hibernate-ddl.auto = create-drop : Hibernate create new table on application startup

and drops same table when application is stopped.

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

**Database connection Properties**

a) driver-class-name : Driver class given by Vendor

b) URL : Location (IP/PORT) of Database usig protocol

c) username : Database username

d) password : Database password

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

**Embedded Databases using Spring Boot**

=> Spring Boot Application works as Database Independent. ie any database if we use

code never gets modified, may be some keys required to change.

=> In that case, We no need to download and install one database for Development.

We can use Embedded (InMemory/RAM) Database given by Spring Boot(3).

=> They are: HSQL (HyperSQL), Apache Derby, H2\*\*. These are Embedded Database,

(NO DOWNLOAD + NO INSTALL). But use only for Dev/Test only. \*\*\*Not in Production.

=> Download and Installed Databases may effect system/Dev Machine performance.

So, Embedded Database are good to use.

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

**JPA Basic Annotations**

a. @Entity : This annotation must be applied at Model/Entity class level.

b. @Id : It indicates Primary Key. Every DB table must have PrimaryKey.

c. @Table: If we do not provide table name, then class name is taken as table name.

To avoid that, we can provide our own table name. (It is optional)

d. @Column: If we do not provide column name, then variable name is taken as column name.

To avoid that, we can provide our own column name. (It is optional)

-> These annotations are exist in package: javax.persistence

-ex--

package in.nareshit.raghu.model;

//JPA Annotation

//ctrl+shift+O

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

import lombok.Data;

@Data

@Entity

@Table(name="emptab")

public class Employee {

@Id //PK

@Column(name="eid")

private Integer empId;

@Column(name="ename")

private String empName;

@Column(name="esal")

private Double empSal;

}

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

\*) Entity class : A class mapped with Database table is called as Entity.

E-R Diagram : Entity(Table) - Relation.

\*) Model class: A class has no logics/claculations methods, contains only variables

with set/get methods, their objects are used to store data and transfer from

UI to Database and database to UI.

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

**Spring Boot : Spring Data using H2 Configuration only**

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

S#1 Create one Starter Project

Name : SprngBoot2DataJpaH2Ex

Dep : Spring Data JPA, H2, Lombok, Spring Web.

S#2 Write 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.Data;

@Data

@Entity

@Table(name="emptab")

public class Employee {

@Id //PK

@Column(name="eid")

private Integer empId;

@Column(name="ename")

private String empName;

@Column(name="esal")

private Double empSal;

}

S#3 application.properties

# Default port is 8080

server.port=9090

# To view Datbase inside browser

spring.h2.console.enabled=true

# To display SQL at console

spring.jpa.show-sql=true

# Database name generated at runtime, to avoid that

spring.datasource.url=jdbc:h2:mem:testdb

S#4 Run main class (Ctrl+F11)

S#5 Goto browser and check with URL:http://localhost:9090/h2-console

> Modify JDBC URL : jdbc:h2:mem:testdb

> Click on Connect

> Click on table name

> Run SQL query

S#6 back to STS IDE and Stop Application

> Goto Console Option

> Click on RED COLOR BOX (Stop Button)

\*)If we forgot to stop and trying to start one more time, then It says

PORT Number was already in use error.

> Then Goto Console click on x symbol (Close Console)

> Then Click on STOP button.

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

\*) Spring Data has provided APIs using Interfaces and Impls, they says:

'DO NOT WRITE CODE FOR BASIC OPERATIONS, ONLY WRITE INTERFACES WITH DETAILS

MODEL CLASS AND PRIMARYKEY TYPE'. Code Generated for you inputs at runtime,

using SimpleJpaRepository(Impl class). [Proxy Pattern].

Proxy using Java:

https://docs.oracle.com/javase/8/docs/technotes/guides/reflection/proxy.html

Example: Github link

https://github.com/javabyraghu/DynamicProxyExample

--Core Java FAQs--

Q) What is Reflection API?

Q) How can we load a class, create object and call method using Reflection?

Q) What is Proxy pattern in Java?

Q) How to generate one class for an interface given with methods logic?