desg1,desg2,desg3

Story board of Mini Project (Layered Application)

Task: Get employees details from emp db table of oracle based on the given 3 desgs

inputs

RealtimeDITest

(Client App)

[presentation logic]

PayrollOperationsController (controller class) [Monitoring Logic]

desg1,desg2 desgs >

List<Employee> obj

IEmployeeService(1) ↑ implements EmployeeSericeImpl(c) [Serivce class] (B.logic/service logic)

javax.sql.DataSource(1)

(upper case) IEmployee DAO (1) SQL Query

desg1,desg2,desg3

implements

HikariDataSource (c)

oracle DB s/w

emp(db table)

implements with desg1.desg2 desgs EmployeeDAOImpl(c)

convert RS records to List<Employee> obj Model class (JavaBean)

gives bunch of Employee records as ResultSet obj

...

...

....

...

....

(sorted)

butputs

Oracle Db s/w

emp(db table)

List<Employee> obj (sorted) (gross Salary and net

=>DataSource class obj will be injected to DAO class obj =>DAO class obj will be injected to Service class obj =>Service class obj will be injected to Controller class obj

Salary calcuations)

Dependnecy Injections using @Autowired

=> ClientApp gets Controller class obj using ctx.getBean(-,-) bean method Dependency lookup

bo

=> when we add spring-boot-starter-jdbc-<ver>.jar

file as dependency we get following classes as

spring bean through autoConfiguration

-> HikariDataSource obj

->JdbcTemplate obj

->NamedParameterJdbcTemplate obj

and etc..

are

These two unused objects in this project

=>we can give inputs to any Autoconfiguraion activity using application.properties file entries.. we can make HikariDataSource object pointing to jdbc con pool for oracle by providing oracle jdbc driver details from application.properties file

represnets

HikariDataSource obj

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system spring.datasource.password=manager

keys are fixed

DAO Interface and DAO Impl class

IEmployeeDAO.java (DAO Interface)

public interface IEmployeeDAO{

}

values are

programmer choice

=>In Layered Apps of the Spring and spring boot Apps we make the classes of different layers (like service class, DAO class, Controller class) and etc.. as different spring beans to make IOC container taking care of their Life cycle management and Dependency Management activities

jdbc con pool for oracle

@Service----> To make the java class as spring bean cum service class @Repository ---> To make the java class as spring bean cum DAO class @Controller ---> To make the java class as spring bean cum controller class

The HikariDataSource obj represented jdbc con pool

We can collect the fixed keys of application.properties file from the

will have set of readily avaiale jdbc con objs pointing to that DB s/w whose jdbc driver details are specified in the application.properties file

eg1:: if application.propperties file contains oracle jdbc driver details then we get jdbc con pool having jdbc con objs of oracle

eg2:: if application.propperties file contains MySQL jdbc driver details then we get jdbc con pool having jdbc con objs of mysql

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

Employee.java (model class- java bean)

String String

public List<Employee> getEmployeesByDesgs(String desg1,desg2, desg3) throws Exception;

EmployeeDAOImpl.java (DAO Impl class)

@Repository("empDAO")

public class Employee DAOImpl implements EmployeeDAO{

private static final String GET_EMPS_QUERY="SELECT EMPNO,ENAME,JOB,SAL, DEPTNO

#7

@Data --> gives settes,getters, toString, hashCode, equals and etc.. method public class Employee{

prviate Integer eno;

private String ename;

private String job; private Double salary; private Integer deptno;

@Data annotation of the Lombok api gives multiple common code to the Java class

private Double grossSalary; FROM EMP WHERE JOB IN(?,?,?) ORDER BY JOB"; private Double netSalary;

@Autowired

}

private DataSource ds;

The IOC container injects HikariDataSource obj that comes through AutoConfiguration

(19)

outer

try with

public List<Employee> getEmpsByDesgs(String desg1,String desg2,String desg3)throws Exception{

resource

nested

List<Emloyee> list=null;

try(Connection con=ds.getConnection(); //gives Pooled jdbc con object

PreparedStatement ps=con.prepareStatement(GET_EMPS_QUERY);){

//set values to query params

ps.setString(1,desg1); ps.setString(2,desg2); ps.setString(3,desg);

// execute the SQL Query

try (ResultSet rs=ps.executeQuery()){

try with resource

//copy each record of the ResultSEt obj to Model class obj

while(rs.next()){

//copy each record to Employee class obj Employee emp=new Employee(); emp.setEid(rs.getInt(1));

BFR

=> Setter methods,

=>Getter methods

=> ToString()

=> hashCode()

=>equals() methods

and etc...

=>All DataSource classes are the impl classes of the javax.sql.DataSource(1),So to inject DataSource object to DAO class object we need to take javax.sql.DataSource (1) type HAS-A property having @Autowired Annotation So that we can inject our choice DataSource object time to time to the DAO class object

(Java bean obj) Employee class obj

0

List<Employee> obj

rs(ResultSet obj)

=> In DAO class,we always sepertae Query from the Code => we place SQL Queries at top of the class in upper case Latters

101 raja CLERK 9000 456

102 suresh MANAGER 19000156

0

Employee class obj

1

*

104 ramesh SALESMAN 7000 256、

ALR #

Employee class obj

2

as the values of String Constant Variable (private static final) for easy modification purpose

=> All datasources classes implement javax.sql.DataSource(I) inject

So we can datasource object to DAO class obj by placing

@Autowired on the top javax.sql.DataSource(1) type HAS-A property

emp.setEname(rs.getString(2));

... ...

emp.setDesg(rs.getString(3));

emp.setDeptno(rs.getInt(5)); emp.setSalary(rs.getFloat(!4));

//add each Model class object to rs

}

}//try2

}//try1

list.add(emp);

catch(SQLException se){

se.printStrackTrace();

throw se; //exception rethrowing for exception propagation catch(Exception e){

e.printStackTrace();

throw e; //exception rethrowing for exception propagation

return list; (20)

} //method

Y/class

Service Interface

public interface IEmployee Service{

public List<Employee>fetchAllEmployeesByDesgs(String desg1,

=> In Layered app development, we need catch and rethrow the exeption

in the following layers to pass/propagate the exception to caller /previous

Layer

=>DAO layer

=> Service Layer

=> Controller Layer

to

And we need catch and handle the exception in the Client App to

present exception to end user in the form of non-technical guiding message becoz all inputs and all outputs/errors should taken or given from/to enduser only from Client App

=> It is recommended to use separate jdbc con for every persistence operation we do on the DB s/w becoz the rollback persistence operation should effect other persistence operations that are happening

String desg2,

In Layered Application

String desg3) throws Exception;

}

Service Impl class

@Service("empService")

#7

public class EmployeeServiceImpl implements IEmployeeService{

@Autowired

private IEmployeeDAO dao;

public List<Employee> fetchAllEmployeesByDesgs(String desg1,

==========

Client App ===> Presentation logics (To give inputs and to show outputs)

Controller class ===> Monitoring logics

Service class =====> b.logics

DAO class =====> Persistence logics

DAO Impl class obj is injected

(17)

hean

ton

String desg2,

...

//convert desg to uppercase letters //use dao

String desg3) throws Exception{

(9) All single scope spring class obj refs will be kept

in the internal cache of IOC contianer

(18)

(21) List<Employee> list-dao.. getEmployeesByDesgs(desg1,desg2,desg3);

return list; (22)

note:: Here u can add b.logics calculating grossSalary and netsalary

empDAO empService

EmployeeDAOImpl class obj ref

}//method

payroll

(12?)

}//class

EmployeeServiceImpl class obj ref

PayrollOperationscontroller obj ref

hikariDataSource HikariDataSource class obj ref

controller class

@Controller("payroll") #7

public class Payroll OperationsController {

@Autowired

private IEmployeeService service;

Service Impl class obj

is injected

(15)

public List<Employee> showAllEmployeesByDesgs(String desg1,

String desg2,

String desg3) throws Exception{

//use service

(16)

(23) List<Employee> list-service.fetchEmployeesByDesgs(desg1,desg2,desg3);

return list; (24) }//method

}//class

Client App/main class

---

(7) @ComponentScan of @SpringBootApplication

scan the current pkg (com.nt) and its sub pkgs for

with

java classes that are annoted streotype annotations

and also gets their scope and also notices that there are no @Bean methods

(8) pre-instatiation of singleton scope spring beans

and necessary dependency Injections using @Autowired takes place =>HikariDataSource object creation

=> JdbcTemplate class obj injected with DataSource obj =>NamedParameterJdbcTemplate class obj injected with DataSource obj =>EmployeeDAOImpl class obj creation injected with HikariDataSource obj =>EmployeeSerivceImpl class obj injeccted with EmployeeDAOImpl clss obj creation =>PayrollOperationsController class obj injeccted with EmployeeServiceImpl class

creation

(6) The @EnableAutoConfiguration of @SpringBootApplication

@SpringBootApplication (5) loads and creates

(1) run the App

public class BootProj03_LayeredApp{ this Configuration class obj)

(2)

public static void main(String args[]){

checks for starter jar files to make

certain java class as spring beans

=>finds spring-boot-starter-jdbc jar make:

the following classes as spring beans

(10) //get IOC container (3) does many things including IOC container creation ApplicationContext ctx-SpringApplicaton.run(BootProj03_LayeredApp.class,args);

by collecting inputs from application.properties file

->HikariDataSource

singleton scope

->NamedParameterTemplate class

->JdbcTemplate class

//get Controller class obj ref (4) collects configuration class name (13) PayrollOperationsController controller = ctx.getBean("payroll", PayrollOperationsController.class);

controller

(11)

Service

//invoke the b.method

bean id

try{

(14)

(25)

List<Employee> list-controller.showAllEmployeesByDesgs("CLERK","MANGER","SALESMAN"); //proess the results

list.forEach(emp->{

(26)

System.out.println(emp);

}

}//try

catch(Exception e){

e.printStackTrace();

s.o.p(" internal problem .---try again");

}

//close the container

}//main (28)

}//class

ctx.close(); (27) close the IOC container

and vanishes all the Objects

(It is end of the Application)

Procedure to develope spring boot Layered Application

====

step1) make sure that oracle Db s/w is installed and the default db table "emp" is avaiable

step2) make sure that STS plugin is installed in Eclipse IDE

step3) create spring boot starter Projects adding the following starters

a) jdbc api c b) lombok c) oracle driver

File menu---> new ---> others --->search spring ---> spring starter --->

Service URL

https://start.spring.io

Name

BootProj03-LayeredApp

Use default location

Location

G:\Worskpaces\Spring\NTSPBMS715-Boot\BootProj03-LayeredA Browse

Type:

Maven Project

Packaging:

Jar

Java Version:

18

✓ Language:

Java

Group

nit

Artifact

BootProj03-LayeredApp

Version

0.0.1-SNAPSHOT

Description Package

First spring boot App

com.nt

Working sets

Add project to working sets

Working sets:

New...

Select...

step4) add the following entries in application.properties file

application.properties

#DataSource cfg

<

=>select the following starters =>

X Lombok

X JDBC API

X Oracle Driver

bj

note: In autoconfiguration the java

classes automatically become spring beans though they are not configured with streo type annotations and @Bean methods

=>next => next ==>finsih

=>if u want to add new starters in the middle of the Project Development

then u can use the following procedure

=> Right click on the Project --->spring---> Add starters ---> select new starters ----> select pom.xml file---> .....

spring.datasource.driver-class-name-oracle.jdbc.driver.Oracle Driver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=manager

To get all keys of application.properties file refer
https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html

How to configure Lombok api plugin in Eclipse IDE?

Ans)

Go to help menu --->install new software -----> search and add url

Work with: lombok api - https://projectlombok.org/p2

--->select all checkboxes -----> next --> next ---> ..... -> asks to restart IDE ... ...

(Its new and latest approach, So old approach of running Lombok api jar file option is not required)

While developing layered apps using spring/spring boot framework we need to make the following classes

implementing interfaces

(Strategy DP Implementation)

a) All DataSource classes should implemeng jl.DataSource(1)

(luckily the Datasource classes given by HikariCP, C3PO, Vibur and etc..

are already implementing javax.sql.DataSource (I))

jakarta

b) All DAO classes should implement common DAO Interface

usecase1 :: DAO class1---> oracle persistence logic DAO class2 ---> mysql persistence logic

usecase2 :: DAO class1---> plain jdbc persistence logic DAO class2 ---> spring jdbc persistence logic DAO class3 ---> spring ORM persistence logic

c) All service classes should implement common Service (1) usecase:: Service class1

Service class2

--->b.logic using formulae1

--->b.logic using formulae2

note:: Since controller class always acts as target class and it is not dependent

not

to other spring beans So we do implement any interfaces on Controller class

How do u propagate/pass Exception raised in DAO class to Client App/enduser in Layered archictecture based Project?

Ans) While developing the Layered App, the DAO classes, Service classes, controller classes will do following things to pass/propagate the raised exception to Client App/Enduser

(End user gives inputs to the client app and expects success output /failed error

from Client App itself through error raised in any part of the Project)

a) we need to declare the exception to be thrown using throws statement

in DAO, SErvice,controller classes (Do not catch and handle exceptions using try/catch blocks)

(or)

in

b) We need to catch and handle the exceptions DAO, Service, Controller classes

(best)

using try/catch blocks or try with resource + catch blocks but we need to rethrow the exception in catch block using "throw" statement

Becoz of the above operations

=> The exception raised in DAO class will

Service class

propagate(directly/rethrown process) to

=> Service class the propagates the received exception to Controller

controller class the propagates the received exception to Client App

=> The client App receives the Exception and handles the exception using try/catch block to display exception related technical messages to enduser as non-technical guiding messages to endusers

How to Convert

RS obj

RS obj(ResultSEt) records to List of Java Bean class objs

--> List<Employee> obj

Java Bean class obj

Employee obj

List Collection (ArrayList)

101 aja

BER

1

101

rs(ResultSet obj) raja 9000 CLERK 1001

102

rajesh 8000

2

833

(2)

(3)

MANAGER 1002. (4) (5)

suresh

9000

MARIAZER 1003

note:: In DAO class

ALR

we place all the SQL Queries

at the top of the class

as the values of

Relavent code in DAO Impl class

private static final

==========

String Constant variables that

to in upper case letters

to

differentiate them from the

private static final String

regular java code

@Autowired

private DataSource ds;

9000 clerk 100T

Employee obj

103 rajesh

8000/manager, 1002 Employee obi

103 suresh

9000 MANAGER 1003

GET_EMPS_BY_DESGS="SELECT EMPNO, ENAME, JOB,SAL, DEPTNO FROM EMP

WHERE JOB IN(?,?,?)";

JAva bean class

@Data

public class Employee{

implements Serializab

private Integer empno; private String ename;

private String desg;

private Double salary;

private Integer deptno;

}

public List<Employee> fetch EmployeesByDesgs (String desg1,String desg2,String desg3) throws Exception{

List<Employee> list=null;

try(//get pooled connection

Connection con=ds.getConnection();

//create PreparedStatement by making the Query pre-compiled SQL Query

```
PreparedStatement ps=con.prepareStatement(GET_EMPS_BY_DESG);){
//set values to Query params
ps.setString(1,desg1);
ps.setString(2,desg2);
ps.setString(3,desg3);
try( // execute the select Query
ResultSet rs=ps.executeQuery(); ){
list=new ArrayList();
while(rs.next()){
//copy each record values to the JAva bean classs obj
Employee emp=new Employee();
emp.setEmpno(rs.getInt(1));
emp.setEname(rs.getString(2));
emp.setSalary(rs.getDouble(3));
emp.setDesg(rs.getString(4));
emp.setDeptno(rs.getInt(5));
//add Java bean class obj to List collection
list.add(emp);
}
}//try2
}//try1
catch(SQLException se){
throw se;
// Exception rethrowing for Propagation
}
catch(Exception e){
throw e; // Exception rethrowing for Propagation
}
return list;
}
```

are

=>In spring, we can make java classes as spring beans either using stereo type annotations that placed on top of the user- defined classes or using @Bean methods of @Configuration class

=>In spring boot, we can make java classes as spring beans a) using stereo type annotations that placed on top of the user- defined classes b)using @Bean methods of @Configuration class c) using AutoConfiguration activity based on the starters that

are added to the CLASSPATH/BUILDPATH of the Project

```
package com.nt.service;
```

```
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.nt.dao.IEmployee DAO;
BootIOCProj03-LayeredApp-MiniProject [boot]
src/main/java com.nt
import com.nt.model.Employee;
@Service("empService")
@Autowired
public class Employee ServiceImpl implements IEmployeeMgmtService {
private IEmployee DAO empDAO;
@Override
public List<Employee> fethEmployees ByDesgs(String desg1, String desg2, String desg3) throws Exception
{
//convert the desgs into UPPPERCASE LATTERS
desg1=desg1.toUpperCase();
desg2=desg2.toUpperCase();
desg3=desg3.toUpperCase();
> BootlocProj03 LayeredAppMiniProjectA com.nt.controller
> EmployeeOperationsController.java
com.nt.dao
> Employee DAOImpl.java
>IEmployeeDAO.java
com.nt.model
> Employee.java
#com.nt.service
> EmployeeServiceImpl.java
>IEmployeeMgmtService.java
#src/main/resources
application.properties
src/test/java
//use DAO
List<Employee> list=empDAO.getEmpsByDesgs(desg1, desg2, desg3);
//Sort the object in List Collection
list.sort((t1, t2)->t1.getEmpno().compareTo(t2.getEmpno()));
>
//calculate gross salary and netsalary
```

list.forEach(emp->{

> JRE System Library [JavaSE-17]

> Maven Dependencies

emp.setGrossSalary(emp.getSalary()+(emp.getSalary()*0.5));

emp.setNetSalary(emp.getGrossSalary()-(emp.getGrossSalary()*0.2));

target/generated-sources/annotations

target/generated-test-sources/test-annotation

});

return list;

}

}

get

>

>

>

src

>

target

HELP.md

mynw mvnw.cmd pom.xml

Assigment :: Develop the Mini Project as the Layered app to hospitals details based on given 3 locations

In order to work work with Lombok api in our Eclipse IDE Projects we need to perform the following operations

a) Lombok api jar /library in the Project (we can add this as the spring boot starter or as maven/gradle dependency)

b) add ProjectLombok related p2 plugin to the Eclipse IDE

The procedure is

name: Lombok api

Help menu ---->install new software ----> add --->

url: https://projectlombok.org/p2

-->select all check boxes ----> next --> next ---> accept all terms

& conditions ----> next ---> next --> automatically restarts the IDE

In the above Layered App, we taken

interface, impl class model for DataSource, DAO,

service logics becoz they are dependent other different classes where as for controller class such dependent is not taken becoz that is always target class and never going to be dependent class to other class..

=> DAO logics are kept in interface - impl class model becoz in different impl classes we can write the persistence logics using different persistence tehnlogies or frameworks like jdbc,hibernate, spring ORM

spring data jpa and etc...

=> Service logics are kept in interface - impl class model becoz in different impl classes we can write the b.logic logics using different formulas and algorithms.

=> DataSource logics are kept in interface - impl class model that gives flexibility to inject our choice DataSource class object to DAO class object

javax.sql.DataSource(1)

pkg.HikariDataSource

(For HikariCP)

pkg.ComboPooled pkg.Basic

DataSource

(For c3P0)

DataSource

(apacheDBCp2)

pkg.OracleUCPDataSource

(for Oracle UCP)

Improvising the Client App by opening and closing IOC container and Scanner object using try with resource becoz both objects AutoCloseable objects

//Client

App

```
package com.nt;
import java.sql.SQLException;
import java.util.List;
import java.util.Scanner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext; import
org.springframework.context.ConfigurableApplicationContext;
import com.nt.controller.Payroll MgmtOperationsController;
import com.nt.model.Employee;
@SpringBootApplication
public class BootlocProj03 MiniProjectRealtimeDiApplication1 {
args);
public static void main(String[] args) {
try(//get access to IOC contaner
ConfigurableApplicationContext ctx-SpringApplication.run(BootlocProj03 MiniProject
RealtimeDiApplication1.class,
//get Scanner class object
Scanner sc=new Scanner(System.in);
```

```
X
// get Controller Spring bean class obj ref
PayrollMgmtOperationsController controller=ctx.getBean("payroll-
controller", Payroll MgmtOperationsController.class);
// read inputs from the enduser
System.out.println("Enter Desg1 ::");
String desg1=sc.next();
System.out.println("Enter Desg2 ::");
String desg2=sc.next();
System.out.println("Enter Desg3 ::");
String desg3=sc.next();
//invoke the b.methods
List<Employee> list-controller.showEmployeesByDesgs(desg1, desg2,desg3);
//display result
System.out.println("Employees belonging to "+desg1+" "+desg2+" "+desg3+" are ");
list.forEach(emp->{
System.out.println(emp);
At this the objects created in try with resource will be closed automatically
System.out.println("Internal Problem ::"+e.getMessage());
});
}//try
catch (Exception e) {
e.printStackTrace();
}
}//main
}//class
```

How did u use Dependency Injection in u r Project?

Ans) My Project is Layered App i.e different logics are kept in different classes and we are making them communicating with each.. The available layers in the Layered app are

a) Persistence Layer or DAO Layer

b) Service Layer

c) Controller layer

d) DB Layer

e) UI Layer

By making the DataSource, DAO, SErvice, contriller classes as spring beans we perform dependency inejctions among them like

=> DataSource object to DAO class obj

=> DAO class object to Service class obj

=> Service class obj to Controller class obj

Depedency Injections using @Autowired

=>The main class/Client App gets Controller class obj

by using ctx.getBean(-) method call Dependency Lookup

This HikariDataSource obj points jdbc con pool for oracle in which jdbc con objs created for the oracle DB will be decided based on jdbc driver s/w details that are placed in the application.properites file