**LEARNING SPRING CORE CONCEPTS :**

1. p schema namespace -> easy-short property injection using it, p:studentId=”121” (also need to include the xmlns in top of file for p : like context but not location of it ie last two lines of context.)

<**bean** class=*"com.springcore.Student"* name=*"student3"* p:studentId=*"111"* p:studentName=*" Shantanu "* p:studentAddress=*"manpur"* />

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Autowire attribute in bean tag : Spring will itself inject object / ref variable (no need of ref=”” tag)

using 3 ways : byName, byType(class name) ,constructor

<**bean** class=*"com.springcore.auto.wire.Emp"* name=*"emp1"* autowire=*"constructor"* />

KNOWLEDGE:

<!--

Auto wiring work on Object type .

Here we have used autowiring , no need to pass reference of the object as we have done com.springcore.ref section using <ref bean= "" />

1.

"byName" matches the name of beans with other beans declare at the top . (\* name with camelcase of class name)

To make their object and pass its reference here.

2.

"byType" mathes with class name, but error can come when more beans created of same class;

The setter fun of Emp class is used to set reference og address var.

3."constructor" mathes with parameter constructor then it searched for name .

-->

<!-- <bean class="com.springcore.auto.wire.Emp" name="emp1" autowire="byName" />

-->

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[3.@Autowired](mailto:3.@Autowired)

This tag in .xml enable all @ eg Qualifier, @Autowired

<**context:annotation-config** />

Used this to show the @Component Scan working in .xml file, when no <bean> tags are used for declaration

<context:component-scan base-package="com.springcore.spel" />

@Autowired //This autowired can be used at also before following to bind the object.

a). normal field ,

b). setter method ,

c). before constructor

*@Qualifier*("address") //When multiple bean are present use this to show the name of bean, to autowire the object.

private Address address; //this name “address” will be searched in autowiring , to find

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4 .Constructor Injection :

If want to make ci more short , can use c-schema namespace to given value to attribute in bean tag itself without extra <construcror-arg value=”” /> tags

<!-- Also include the namespace of c-schema for directly passing val in bean tag : here to use it (constructor injection) -->

<!-- Method 1. Normally ::::::

<bean class="com.springcore.ci.Certi" name="cer" >

<constructor-arg value="Java Certification" />

</bean>

-->

<!-- MEthod 2 .

🡪Or can use c-schema for declaration , interface/ namespace schema - already included at top.

<bean class="com.springcore.ci.Certi" name="cer" c: name="Java Cetificates" >

</bean>

-->

<**bean** class=*"com.springcore.ci.Certi"* name=*"cer"* c:name=*"Java certificates"*></**bean**>

<**bean** class=*"com.springcore.ci.Person"* name=*"person"*>

<**constructor-arg** value=*"Shivam"* />

<**constructor-arg** value=*"99"* type=*"int"* />

<**constructor-arg** ref=*"cer"* />

<!-- Or Here also directly can use the using c-schema : to avoid all these 3 constructor args tags.

Assignment 1:

<bean class="com.springcore.ci.Person" name="person" c:name="Shivam Rai" c:personId="99" c:certi-ref="cer"></bean>

Already defined certi as object of (independent class ) to take reference here.

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

If refereced type used in property / setter based injection. Then

< property name=”certi” ref=”cer” />

-->

<**constructor-arg** >

<**list** >

<**value**>Delhi</**value**>

<**value**>Bambai</**value**>

<**value**>Calcutta</**value**>

</**list** >

</**constructor-arg** >

</**bean**>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5 .Collections : Injection of various collections in bean

<**bean** name=*"emp1"* class=*"com.springcore.collections.Emp"* >

<!-- Implemented simple attribute \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<**property** name=*"name"* value=*"shivam"* />

<!-- Implemented list \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<**property** name=*"phones"*>

<**list**>

<**value**>123</**value**>

<**value**>456</**value**>

<**value**>789</**value**>

<**null**/>

</**list**>

</**property**>

<!-- Implemented set \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<**property** name=*"addresses"*>

<**set**>

<**value**>Delhi</**value**>

<**value**>Lucknow</**value**>

<**value**>Raisen</**value**>

<**value**>Indore</**value**>

</**set**>

</**property**>

<!-- Implemented map \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* the name="courses" should match with the var "courses" declared in Emp class ie correct spelling -->

<**property** name=*"courses"*>

<**map**>

<**entry** key=*"java"* value=*"2 Month"* />

<**entry** key=*"Python"* value=*"1 Month"* />

<**entry** key=*"C++"* value=*"5 Month"* />

</**map**>

</**property**>

<!-- Implemented Properties \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<**property** name=*"props"*>

<**props**>

<**prop** key=*"name"*>Shivam</**prop**>

<**prop** key=*"status"*>SDE-2</**prop**>

<**prop** key=*"salary"*>90k</**prop**>

<**prop** key=*"Time"*>8pm</**prop**>

<**prop** key=*"comp"*>sleepwell</**prop**>

</**props**>

</**property**>

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

</**bean**>

OUTPUT :

Shivam 🡪simple

[123, 456, 789, null] 🡪List

[Delhi, Lucknow, Raisen, Indore] 🡪set

{java=2 Month, Python=1 Month, C++=5 Month} 🡪Map

{comp=sleepwell, name=Shivam, Time=8pm, salary=90k, status=SDE-2 } 🡪Properties

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6.Using JavaConfig as @Configuration class

com.spring.practicejdbc project under practice JDBC -> springjdbc (Core Sprig)

Refer project of jdbc under practice jdbc : all 3 methods discussed.

Only using the JavaConfig class for configuration.

@Component – used above the class name so , spring can create obj of class. And get the bean of it in APP.java using the same class name in camel case (student) , but want an explicit name then use

@Component(“stud”) now can get bean using “stud” name.

Used @Autowired to bind “book” object in student class, spring created the object of book using @componet placed on book class and autowired it here.

To give the values to the attribute used @Value(“1234”) annotation , in book class field on setter methods , and student class fields on student setter methods , And the book type varible takes value itself from book obj when autowired.

For all these only this need to be done in JavaConfig.class

@Configuration

@ComponentScan(basePackages = "com.spring.practicejdbc")

public class JavaConfig {

}

Method b) No use of @Component in JavaConfig

To ease the process of value passing to fiels , used the

@Bean tags in javaConfig file and make the method which returns the object of the class.

So that use of @Component and @componentscan is eleiminated.

Just comment out all the @Component , and @Value tags and uncomment the @Bean tags , which are used for together initializing the bean and setting the value to the field

/\*@Bean(name={"boook" ,"makebook"})

public Book makebook()

{

Book b= new Book();

b.setName("Leanign is continous");

b.setPrice(99.00);

return b;

}

@Bean(name={"stud","makestudent"})

public Student makestudent()

{

Student s = new Student();

s.setBook(makebook());

s.setId(121);

s.setName("Durgesh");

s.setRollno(167);

return s;

}

\*/

Also get the bean using the different name provide the name in @Bena{name=”stud”, “makestudent”} :

The default name to get the bean is method name itself context.getBean(“makestudent”);

Also at the same time the @Value tags are used , then it will put the @Value over default value given in @Bean .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7.Bean Life-Cycle

* METHOD 1 (in Samosa class ): using the init-method attribute in bean tags and defining function (init ,destroy ) in class.

Init() and Destroy methods : init method is called after the object setter are done, and Destory method work before object deleted.

<bean class=*"com.springcore.lifecycle.Samosa"* name=*"s1"* init-method=*"init"* destroy-method=*"destroy"* >

<**property** name=*"price"* value=*"50"* />

</bean>

//definition in Samosa class for method

public void init()

{{

System.***out***.println("Running inti after obj creation of Samosa ");

}

public void destroy()

{ {

System.***out***.println("Running before ending Somosa obj :::It also need AbstractApplicationContext to use register shutdownhook");

}

AbstractApplicationContext context = new ClassPathXmlApplicationContext("com/springcore/lifecycle/config.xml" );

context.registerShutdownHook(); //it is always required to call destroy method , either use init in arguments or by implementing interfaces ,or by passing the @PostConstruct annotaion.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Method 2: (in Pepsi class ): Using Interface : these method can be used. (do work of init and destroy methods)

public class Pepsi implements InitializingBean , DisposableBean

{

…

,

.

public void afterPropertiesSet() throws Exception

{

// Fom interfACE DECLARED

System.out.println(" Init Methiod Pepsi: using interface Initializaiton bean");

}

public void destroy() throws Exception {

// From the interface declared

System.out.println(" Destroy Methiod Pepsi: using interface Disposable bean");

}

}

Method 3: (in Example class ): using the @PostConstruct @PreDestroy

U can use these method directly in class, just include the depencdency of javax.annotation 1.3.2 in **pom.xml**

import javax.annotation.PostConstruct;

import javax.annotation.PreDestroy;

public class Example {

‘

‘

@PostConstruct

public void start()

{System.out.println("starting method");}

@PreDestroy

public void end()

{System.out.println("ending method");}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8.Passing reference object :

a).using ref tag <ref bean="bref" />

<bean class="com.springcore.ref.B" name="bref" >

<property name="y" value="90"/>

</bean>

<bean class="com.springcore.ref.A" name="aref" >

<property name="x" value="50"/>

<property name="ob">

<ref bean="bref" />

</property>

b).using ref attribute

<property name ="ob" ref="bref"></property>

c) using p:schema

<bean class="com.springcore.ref.A" name="aref" p:x="50" p:ob-ref="bref"/>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9. Spring expression Language (SPEL)

When we use the @Component before the class name : its has to enabled from .xml by

<context:component-scan base-package="com.springcore.spel" />

But when we are using annotations like @Configuration, @ComponetScan , to enable them pass

<context:annotation-config /> in .xml file

Using @Value for passing values in spring expression language.

@Value("#{15+1}") // How to work with Spring expression

private int x;

// including static methods.

@Value("#{T(java.lang.Math).sqrt(25)}")

private double z;

@Value("#{T(java.lang.Math).E}") // syntax : @Value(" { classname.method/var }")

private double e;

@Value("#{ new java.lang.String('Durgesh Tiwari') }")

private String name;

@Value("#{8>1} ")

private boolean isactive

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10. StandAlone Collections (globally declared and use by ref id )

In .xml file the collection(map,list,set etc) are declared using the util schema namespace , and can be used universally by other by taking refernce of their names

Need to include the util-schema just like the context-scheman ,also location of it

standalone list , attach the util schema by coping and replace p , and also copy paste the last 2 line with util replacing context-->

<**util:list** list-class=*"java.util.LinkedList"*

id=*"mybestfriends"* >

<**value**>Aman</**value**>

<**value**>Shantanu</**value**>

<**value**>Aditya</**value**>

<**value**>Rohan</**value**>

<**value**>Prateek</**value**>

</**util:list**>

<!-- standalone Map ,-->

<**util:map** map-class=*"java.util.TreeMap"* id=*"fees"*>

<**entry** key=*"Spring Framework"* value=*"8000"* />

<**entry** key=*"Django Framework"* value=*"5000"* />

<**entry** key=*"hibernate"* value=*"2000"* />

</**util:map**>

<!-- standalone Properties -->

<**util:properties** id=*"dbconfig"* >

<**prop** key=*"driver"* > com.mysql.cj.jdbc.driver </**prop**>

<**prop** key=*"username"* > root </**prop**>

<**prop** key=*"pass"* > root </**prop**>

<**prop** key=*"url"* > mysql:jdbc://localhost:3360/database </**prop**>

</**util:properties** >

<**bean** class=*"com.springcore.standalone.collections.Person"* name=*"person1"*>

<!--

<property name="friends">

<ref bean="mybestfriends" />

</property>

Or simply use 1 line of property as done below:

Here we have used the property injection via standalone ref passing.

-->

<**property** name=*"friends"* ref=*"mybestfriends"* />

<**property** name=*"feestructure"* ref=*"fees"* />

<**property** name=*"properties"* ref=*"dbconfig"* /></**bean**>

To display the class it belong can use this getClass() method also :

System.***out***.println(person1.getFriends().getClass().getName());

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11. Stereotype :

Basically defining the scope of object creation : 1.Singleton 2.Prototype 3.4.5

s1.getAddresses()

s1.getAddresses().getClass().getName()

s1.hashCode()

Can use function like to check the object created are same hashcode or not.

Method 1: Use @Scope(“prototype”) with class declaration

Method 2 : give scope attribute in bean tag:<bean … scope=”singleton”>

//streotyoe annotations

//also can give obj name here @Component("ob") then no need to take camel case obj in test.java, also if ("") not given🡪 then use Camel case to use getBean()

Using the @Scope annotation: also enable it in the .xml file

<**context:component-scan** base-package=*"com.springcore.streotype"* />

Method 1

*Component*("ob")

*@Scope*("prototype")

public class Student

{

*@Value*("Shivam") //@Value can also used above the setter injection to assign the values

private String studName;

*@Value*("Lucknow")

private String city;

*@Value*("#{temp}") // temp is the id of the standalone list created in the .xml

private List<String> addresses; //To insert the list / collection type using annotaion 1. make the standalone coll. in .xml -> then use @value in .java file

‘

‘

}

<!-- Make a standalone collec. here and use annotaiontion @Value(#{temp}) in .java file -->

<util:list list-class="java.util.Vector" id="temp" >

<value>Raisen</value>

<value>Bhopal</value>

<value>Bina</value>

<value>Zebrra</value>

</util:list >

Student s2= context.getBean("ob", Student.class);

System.out.println(s2.hashCode());

System.out.println(" Singleton scope :By default spring container given the same object: ie hashcode will be same "+ "\nBut if we want to change can use M2. use <bean class= name= scope= "

+ "\nM1. @Component @Scope() " );

Sd In .xml file :

Method 2 :

<!-- Used the scope to create obj differently : by defaout it is Singleton (1 obj given) , Prototype means each time new memory obj will be created -->

<bean class="com.springcore.streotype.Teacher" name="teacher"

scope="prototype" />

@Component("ob") // ob is varible which can be used to get bean . also enable the componetscan tag base package in .xml file to make it work.

<**context:component-scan** base-package=*"com.springcore.streotype"* />

Student s2= context.getBean("ob", Student.class);

Teacher t1= context.getBean("teacher",Teacher.class);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ASPECT Oriented PROGRAMMIng ( AOP )**

Designing the class which contains the method which run with java methods either before or after to perform validation etc eg @Before , @After etc

Also declare the aop schema namespace at top of xml file just like the context xmlns: also location using last to line of context.

Make the respective bean of main class, and aspect class then to use them.

<**aop:aspectj-autoproxy** /> need to be declared in .xml file to enable the @Aspect in MyAspect class.

Or can @EnableAutoJProxy in class, if no xml is used.

Make a sepereate class MyAspect which contain method , it will not disturb the normal flow of program ,

@Aspect is provided on declared Aspect class which contains method to support validation in main class

@Aspect

public class MyAspect {…….}

@Before( execution (\* full path of function in main java class )

Eg. This method will run before the makePayment method .

@Before("execution(\* com.aop.services.PaymentServiceImpl.makePayment(..))")

public void printBefore()

{

System.out.println("Payment started...");

}

Important Learning:

\*When using the abstract class : in bean tag give abstract = true; also mention the parent attribute in subclasses bean tag

\*Made a self testing process under practice jdbc (IMP CONCLUSIONS)

/\*

//METHOD 1 🡪 EASIEST METHOD :

USING .XML configuration , given the declaration of bean for each class : no annations needed, simple and easiest.

// No need of any Component or @Qualifier etc

/\*

<bean id="bk" class="com.spring.practicejdbc.Book">

<!-- //Property : injection : eitehr use ci or pi but do not intermix them -->

<property name="name" value="Shavam kalar" />

<property name="price" value="1099.03" />

</bean>

<!-- //constuctor injection -->

<bean id="stud" class="com.spring.practicejdbc.Student" >

<constructor-arg ref="bk" />

<constructor-arg value="121" />

<constructor-arg value="Shantanu" />

<constructor-arg value="167" />

</bean>

1(b)

To set the values in the field of Classes directly use the property injection / or use there setter method to assign

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

\*

METHOD 2: JavaConfig.java

All though null values are passed , but working fine all.

// App.java file to get bean from the JavaConfig class

ApplicationContext context= new AnnotationConfigApplicationContext(JavaConfig.class);

Student s=(Student)context.getBean("makestudent");

System.out.println(s);

Use the JavaConfig.java file to configure and used the

@Configuration

@ComponentScan(basepackges="com.spring.practicejdbc") : to undestart that IOC container make obj for classes where they have used the @componenet.

public class App

{

public static void main( String[] args )

{

System.out.println( "Learning the jdbc using the commands of query database." );

ApplicationContext context= new AnnotationConfigApplicationContext(JavaConfig.class); //when JAvaConfig file is used.

Student s=(Student)context.getBean("student");

System.out.println(s);

}

}

//////////////

JavaConfig.java

@Configuration

@ComponentScan(basePackages = "com.spring.practicejdbc")

public class JavaConfig {

}

//////////////

Student.java

@Component

public class Student {

@Autowired

private Book book; //dependecy injection need to be done

private int rollno;

private String name;

private long id;

.

.

.

}

///////////////

Book.java

@Component

public class Book {

private String name

.

.

.

}

???\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

METHOD 2 (b): To pass the values in the beans

JavaConfig class -

@Configuration

//@ComponentScan(basePackages = "com.spring.practicejdbc")

public class JavaConfig {

@Bean

public Book makebook()

{

Book b= new Book();

b.setName("Leanign is continous");

b.setPrice(99.00);

return b;

}

@Bean

public Student makestudent()

{

Student s = new Student();

s.setBook(makebook()); // pass the method of(object\_creation) in the dependend object.

s.setId(121);

s.setName("Durgesh");

s.setRollno(167);

return s;

}

// @Bean(name={“stud” ,”makestudent”})

public Student makestudent()

{

}

\*\* //if other name are provided , then the bydeafult camel case name of bean diables .: used for getting bean in context.

//TO get the bean in App.java file just use the mehtod name , or u can give the other names in @Bean("stud") , to get the bean ,

\* Also when autowired the object , use the @Qualifier("") to tell the name of bean , to be injected .

\*

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

// METHOD 3 : Using @Component annotations

🡪. Also need to enable this @Component in

a) .xml file by using -->

<context:component-scan base-package="com.spring.practicejdbc" />

b) if using JavaConfig.java then use

@ComponentScan at top of class,below @Configuration , giving path is optional in @ComSc.

Basically declare the @Component over all classes : spring will create the obj for them

3 (b). Now to set the value need to use the

@Value() above each field in respective class

//

student.java

@Autowired // The object created of book will be autowired here

private Book book; //dependecy injection need to be done

private int rollno;

private String name;

private long id;

.

.

.

@Value("121")

public void setRollno(int rollno) {

this.rollno = rollno;

}

public String getName() {

return name;

}

@Value("Rajan")

public void setName(String name) {

this.name = name;

}

public long getId() {

return id;

}

@Value("1111111")

public void setId(long id) {

this.id = id;

//////////////

book.java

@Component

public class Book {

private String name;

private Double price;

.

.

.

@Value("Novel")

public void setName(String name) {

this.name = name;

}

public Double getPrice() {

return price;

}

@Value("2000.00")

public void setPrice(Double price) {

this.price = price;

}

/////////

\* configannoation.xml file

.

.

<context:component-scan base-package="com.spring.practicejdbc" />

.

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

Some confusion left in this 3 method : how to set the properties directly for ref objects Book just like we use @Value for other fieds

\* \*/

1. LEARNING SPRING JDBC :

@Configuration

public class JdbcConfig {

@Bean( name={"ds"})

public DataSource getDataSource() // we can also use DataSource return type of function: inplace of DriverManager as its parent class

{

DriverManagerDataSource ds = new DriverManagerDataSource();

ds.setDriverClassName("com.mysql.cj.jdbc.Driver");

ds.setUrl("jdbc:mysql://localhost:3306/springjdbc"); //"springjdbc" is the name of database

ds.setUsername("root");

ds.setPassword("root");

return ds;

}

**@Bean( name={"JdbcTemplate"}) //Or use @Bean("JdbcTemplate")**

**public JdbcTemplate getTemplate()**

**{**

**JdbcTemplate jdbcTemplate= new JdbcTemplate();**

**jdbcTemplate.setDataSource(getDataSource());**

**return jdbcTemplate;**

**}**

@Bean(name={"studentDao"})

public StudentDao getStudentDao()

{

StudentDaoImpl studentDao= new StudentDaoImpl(); //Obj of class StudentDaoImp is made

studentDao.setJdbcTemplate(getTemplate());

return studentDao;

}

}

// WORKING WITH JDBC SPRING

A). Import the 4 dependecies in pom.xml file : spring core, springcontext, jdbc , j connector .

B).save it and update project, see Maven Dep folder for following dep , if not present then update proj with "offline mode" or "force import"

C). To connect to Data base Structure is such that

three packages made :

1.Package "com.spring.practicejdbc " -> App.java , config.xml , JdbcConfig.java

2.Package "com.spring.practicejdbc.entites" -> Student class

3.Package "com.spring.practicejdbc.dao" -> StudentDao interface, StudentDaoImpl class ,RowMapperImpl class

Make respective classes & interfaces inside each packages

D). Declare the class Employee as made in Mysql database.

E). Now in XML file

first make a bean class to make connetion to database : using DriverManagerdataSource class

and provide properties like driverclassname, url,username, password

F). Now we need to make connection with database :

-> JdbcTemplate has dataSource prop: is capable of making conn with database bean of DriverMAnagerDAtaSource , at same time we need a source to handle jdbcTemplate class obj.

-> So we make a class : StudentDaoImp implements StudentDao interface , which provides varios method to work in database.

-> Declare a bean of StudentDaoImp, and declare a obj of jdbctemplate in it as property. (as a reference object variable)

->Before using jdbctemplate obj , its bean must be declare above this bean , so declare the bean and make the property dataSource as its reference (which make the conn using url, username, pass etc)

->The class name should be written properly .

G). Now next work is declare the various method (insert, update, delete) in StudentDao interface and override them in StudentDaoImpl class

Also in StudentDaoImpl class declare the obj of JdbcTemplate

Implement the getters, setters for obj

----> ALso ALL the query are written here and implemeted used this.JdbcTemplate.update fun

H). Next is to work in driver class , use ClassPathApplicationContext to getBean

// getBean take obj of EmployeeDao class in obj employeeDao

// Next is to make a object of Employee class, by putting id, name, city using setter function

//Now use The obj of EmployeeDao class to insert this employee obj created before.

I). Now run app.java and see the changes update in sql table.

J).Similary can implent the method of change , delete ope

Eg --> First declare method in interface EmployeeDao.java

--> Then implement/ override the method in EmployeeDaoImpl

--> Then in app.java using EmployeeDao obj perform the required query.

K). To perform the select query 2 ways

1.To fetch the single row :

Declare the select row in EmployeeDao interface.

Overwrite in EmployeeDaoImpl class

Now task is to Use the RowMapper obj , to preform select query.

Here we use the queryForobject

(basically RowMapperImpl class implements the rowmapper interface which has implemeted method for getting the data using ResultSet and returning the obj )

Student student =this.jdbcTemplate.queryForObject(query,rowMapper,studentId); // this row mapper is basically an interface which helps to convert the result set to Class objects to display them , here its class rowmapperclass will implement the method.

return student;

L).Select all TO Fecth the all Data

1.Declare the method getAllEmployee, which return list.

2.Implemetent the method in EmployeeDaoImpl

3.Write the query in overwrite method and use jdbcTemplate.query( pass query, and RowMapperImpl())

1. LEARNING SPRING ORM :

Mydreamservice@ias2025.com

ORM object relational mapping.

Spring + ORM frameworks(Hibernate,Toplink ,iBATIS)

ADV:

-> less coding,

-> east to test,

-> better exception handling,

-> interated transaction management;

Here we need to make the table explicity and mapp it with database using ORM , so ORM now will return the object.

To work with Spring ORM , we need HibernateTemplate object.

The Structre : refer Video 1 : code with durgesh diagram .

The Strcuture says

1.Make Product Dao class(data access object ) , which will contain the Object of HibernateTemplate

2.But HibernateTemplate depends on SessionFactory interface which further depend child class LocakSessionFactoryBean .

3.LocalSessionFactoryBean need 3 things

a). DataSource (url, username)

b)Hibernate Properties

c).AnnotatedClass (eg Student)

// Instructions to work with ORM by SHIVAM

A). Make a new Package : com.spring.orm -> app.class and config.xml

Make subpackages for Dao --> StudentDao.java

Make subpackage for Entites --> Student.java

B). Make ready the Student.java class

-> Here we are using @Entity,@Table(name of table), @Id ,@Column(name="student\_id")

to make our own table.

-> Take necessary getters, setters , constructor.

C). Now we have Entity (StudentDao) class , we use it to add this table to database .

but insetion is done by obj of hibernateTemplate , so delclare it as data member with getters, setters , contrcutors etc

D). Now we need to tell the spring continer to make objects of respective classes. by making beans in .xml file.

1. Make bean of StudentDao class and in property take reference of hibernateTemplate

2. Declare hibernateTemplate bean which takes reference of sessionFactory interface

3. Declare bean of childclass of sessionFactory : LocalSessionFactoyBean

4.LocalSessionFactoyBean is declared as bean with 3 properties

a) dataSource -> Need to declared as bean

b)hibernateProperties -> dilect ,show\_sql,hbm2ddl etc

c)annotatedClasses -> entites eg StudentDao.java

5.Declare bean of dataSource and set name of class : shortcut (press ctrl + shift+ T) search DriverManagerDataSource open it , and copy the interface.Classname correctly.

Maitain the required connection in dataSource(driverClassName, url, username, password)

E). Next Step is use this in app.java

Delare the AppliacationContext = new ClassPathXml...

and get the required Bean()

Now perform insert operation on Student obj .

Error comes access denied to write

F). Using the @Transactional , apply the namespace : copy and replace p:schema using tx , also last two lines of contxt are copied and tx is put in place of context.

Also enalbe the transational by using

<tx : annotation-driver /> in xml file

G). Perform CRUD operations

C -> create

R ->read (select queries)

U ->update

D ->Delete

**\*\*\* Very Important Notes self for Spring MVC :::**

1. LEARNING SPRING MVC FROM YOUTUBE CODE WITH DURGESH LECTURES

1.)Create the tomcat server download tomcat 9 zip file. and extract it to the practice eclipse folder in e drive

2.) configure : eclipse window -> preference ->server ->runtime environment --------- select(add) Apache version 9 ->next and in browse : select folder of parent directory of bin. then finish .

And try your server. window ->show view ->server

(if 1 time user) : then choose the new server by right click and select tomcat 9 and also start your server and on google type localhost:8080

STARTING MVC PROJECT:

3).make a new maven project and choose the archtype-webapp in categlog internal. and create springmvc project . and keep internet on and done.

4).

Configure Spring with Tomcat :

*THIS ISSUE WILL COME EVERY TIME WHEN U CREATE THE NEW PROJECT FOR USING TOMCAT SERVER SO :*

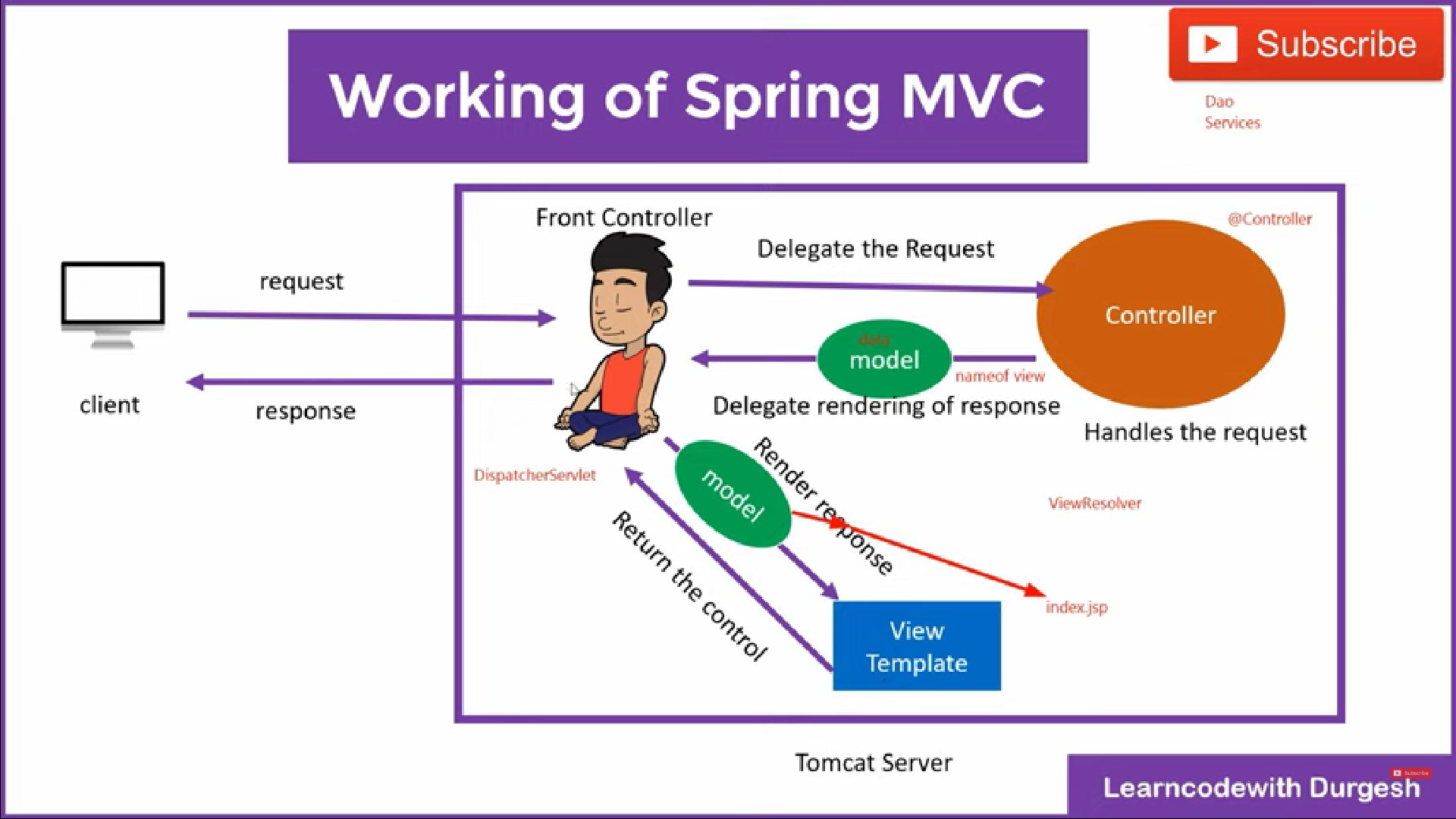
Now error comes because index.jsp has not server runtime libraries . To Solve right click your project and choose build path

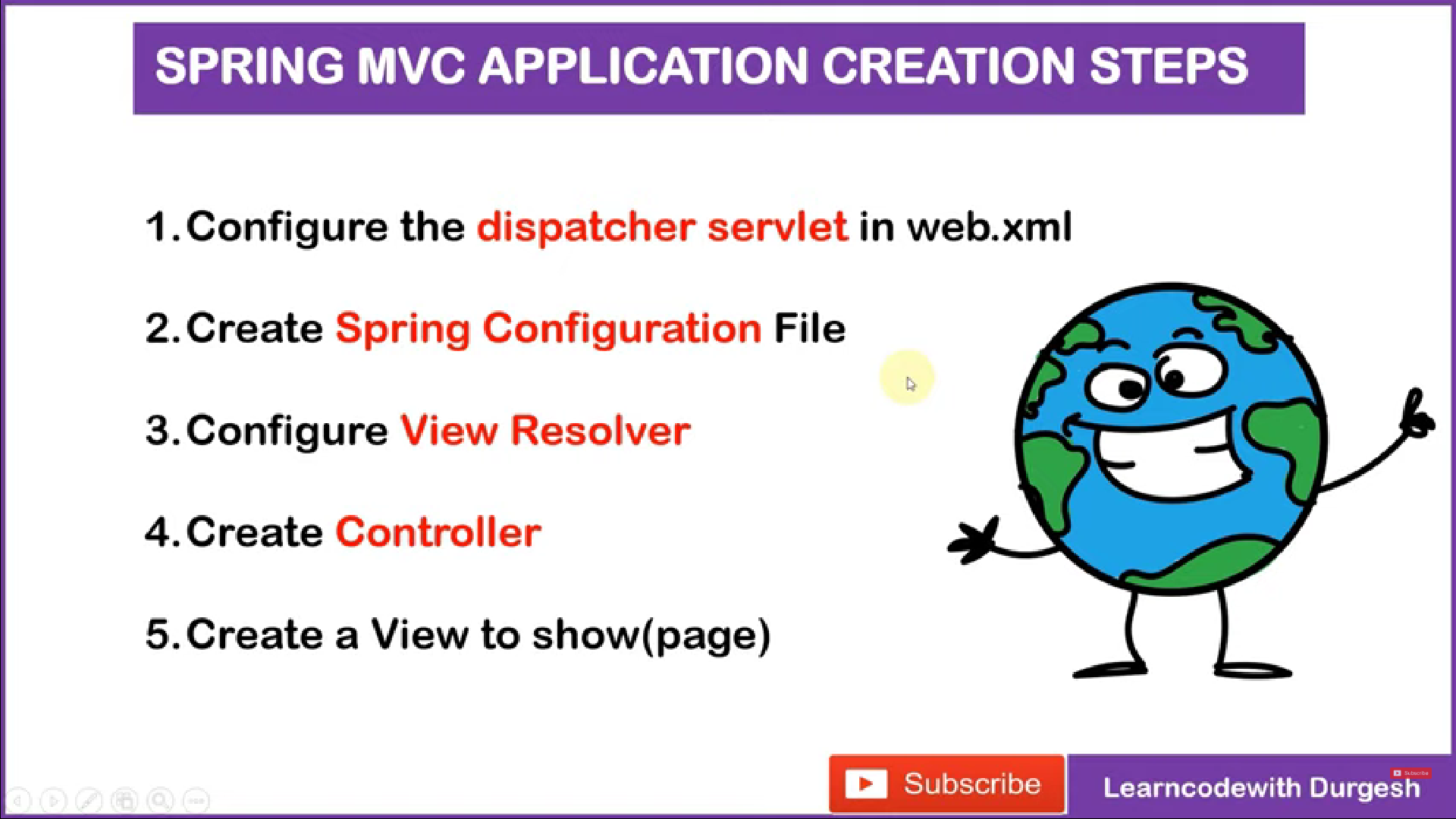
Springmvc -> buildpath -> configbuild\_path -> libraries tab -> addserver runtime -> tomcat 9 libraries .

Inject mvc-dependency from google in pom.xml search ,

(do check the force install option while updating project : if they do not import successfully ,in maven dependencies folder )

MVC itself install other required dependencies like core, context, etc





*5). IMPORTANT 5 STEPS :*

A). CONFIGURE THE DISPATHCHER SERVLET (front controller) IN WEB.XML and also map the servlet. with URL in web.xml file

B). CREATE THE SPRING CONFIGURATION (spring-servlet.xml) FILE in WEB-INF folder

C). CONFIGURE VIEW RESOLVER in .xml file

D). CREATE CONTROLLER in pack controller in src/main/java

E). CREATE A VIEW TO SHOW PAGE create jsp files of methods declare in HomeController

6). In web.xml file create the servlet tag and create it , to search the class name of servlet can use CTRL + SHIFT + T in any java class and then search DispatcherServlet and can copy package and class name from there , no need to by-heart .

And then also map the servlet declared with url pattern:: / to handle all the url requests

7). Create Configuration file :::: ( servletname-servlet.xml ) file : inside web-inf folder ,copy data from previous config.xml file and remove extra beans.

8). create bean view resolver and adject its 2 properties

a-> give the prefix -> containing folder /WEB-INF/views/

b-> give suffix -> ending of the page .jsp

path of page is : /WEB-INF/views/ \_\_\_\_\_\_\_\_ .jsp

9). Make the controller class inside springmvc.controller package

and put annotation @Controller on class

-> also maintain @RequestMapping("/home") to tell when will this function work on writing url or /home is fired .

10). Now we need to create the "index".jsp page in views folder(if not present the create views folder.)

In views folder create new jsp file : index.jsp , and can delete old index present in WEB-INF

(by default project run on index page provided , so your can declare another handler method with Requestmapping (“/”)…. {return index;} , to run your own index page of view folder)

Last step is to enable the @Controller annotation by <context:component-scan base-package="path where u used @Controller" >

Imp\* : you can mention to scan the full project : its also fine

At last save and run the springmvc project (left click and select run with server).

Fire /home with project name

localhost:8080/springmvc/home

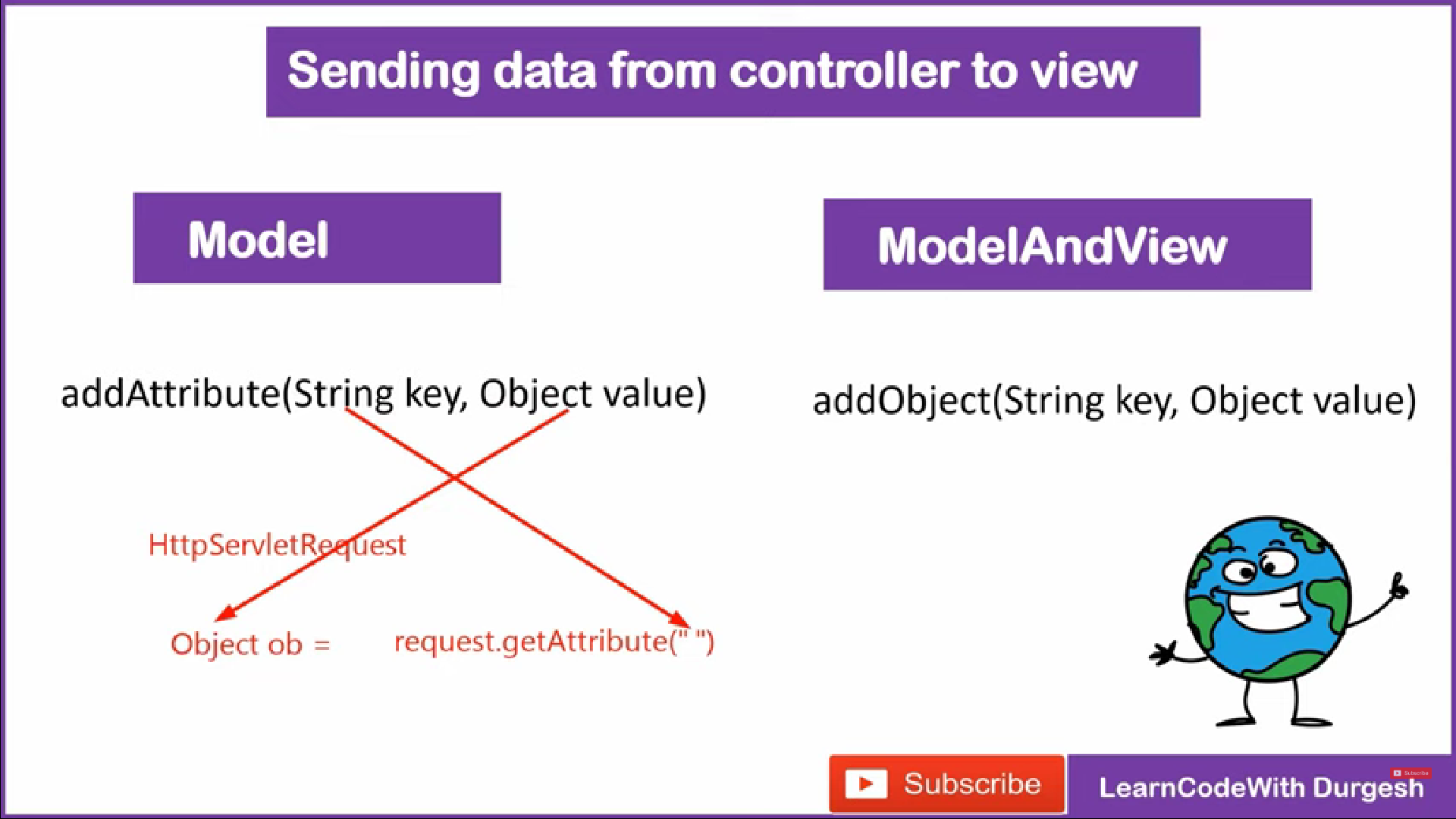
Similarly create other handler method like contacts , etc and run them from HomeController class

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

VIDEO :

\*\*\*\*\*\*\*\*\*\*\*\*\* SENDING THE DATA FROM CONTROLLER TO VIEW () \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ie not need to write static data in view pages, instead supply data from controller , which can be extracted out in views.jsp files.



Use these method to send data

Just pass the Model obj in (argument) of handler method in HomeController class and use addAttribute(“name”,”passedvalue”)

METHOD 1: (mostly used) MODEL OBJECT

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-> model object : First import util.List in index.jsp : paste this in top <%@ page import = "java.util.\*"%>

-> In HomeCntrler :

Take model obj in parameter of method and use

model.addAtribute(String key, Obj val)

1. (Getting value using <% %> tags)

->In JSP file : get data using scripplet tag

<% String name = (String) request.getAttribute("key") %> also typecast it

->And then finally print <%=name %>

-> to print the list: loop is used out.println() ......look code

2. (Getting value using ${name} 🡪 expression language)

Alternate to using scripplet tag to extract data :

is expression lang : which can be ${name} , can be enabled using <%@ page isELIgnored="false" %> at top of jsp file

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

METHOD 2 : Returning obj of ModelAndView

->ModelAndView object 🡪 addObject(String key , Obj val) method

Make new controller to show model view

1. make @requestmapping("/help") and define a method named help

@RequestMapping("/help")

public ModelAndView help()

{

ModelAndView modelAndView= new ModelAndView();

System.out.println("this is help [page] ");

modelAnsView.addObject("name","Rai" );

modelAndview.setViewName("help");

// generally this was return statement in previous method.(ie the help.jsp page will is displayed upon firing in url)

return "modelAndView";

}

2. create view (help.jsp file ) in views folder.

And rest part is same just get.Arribute("key") and display using script tag or use expression language to get send attribute in .jsp page

TIP: WHat ever Additional features you are using like List just , import them at top

<%@ page import="java.util.\*"%>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11). Quiclky access SENT DATA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* :

Video: 57

Using jsp expression and JSTL(used for using for each loop using c: tags ,Need to include the dependency in .xml and taglib of jstl in .jsp file )

Now we will learn : how to direcly use expression lang to avoid repetitive use of getattribute() method.

ALso at top of .jsp file make

<%@ page isELIgnored="false" %> , ie now ${name} will be working.

To access a list directly print it {list} in .jsp or jstl

TO ACCESS LIST ELEMENTS IN FOR EACH LOOP ::::::

Use JSTL :

to access in for-each loop -> for each element access.

12). Applied all these concept in About.jsp

a>

Inject JSTL dependecy in pom.xml

b>

Also insert the <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> at top of the .jsp file.

So use now using this to access ele of list "marks"

<c:forEach var="item" items="${marks}">

<h1> ${item} </h1> or can use here <h1> <c:out value="${item }"><h1> //jstl c:out

</c:forEach>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 58

13). Send data from view(form) to Controller(beckend) \*\*\*\*\*\*\*\*\*\*\*\*\*

View ---------------------> Controller

(Html form) (HttpServletRequest)

Getting the data from Controller

using getParameter("field name") ----> old method

Now in spring use

@RequestParam ---> get one -one field

@ModelAttribute ---> take all object data / full form

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video : 59

@RequestMapping -->>

i) can be applied to map handler method to use url : /home

ii) can be applied to class so that all other method extend that

eg class mapping /appoint

ie now methods url will be relative to class url

eg /appoint / home/

Using RequestMapping we can also give method type (path and value can be used both same meaning )

@RequestMapping(path="/home" , method= RequestMethod.GET)

….

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video : 60

Make New view contact.jsp and design a form using bootstracp

Also make a new controller , in samepackage ContactController to handle the submission of form :

Now take the 3 required field of form Email, UserName , Password.

Now for <form action="processform" method="post"> we have given relative url="processform" not absolute url="/processform" ,(in form action in html form don’t confuse with url handler method “/..”) also method is post so take care while taking out data using @RequestMapping to give value/path and the method type

Now go to HomeController and design the handle method to support the processform url firing

:We have used @RequestParam to fetch the field from "form" submission , And Model Object (model.addAttribute) to send the attributes to another jsp page so that there they can be extracted out using expression language ${name }

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

@RequestMapping(path="/processform" , method = RequestMethod.POST)

public String handleform(@RequestParam("email") String userEmail ,

@RequestParam("userName") String userName ,

@RequestParam("password") String userPassword , Model model)

{

model.addAttribute("name",userName);

model.addAttribute("email",userEmail);

model.addAttribute("password",userPassword);

return "success"; // need to make this .jsp page to handle this /processform firing

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<%@ page isELIgnored="false" %> at top of file

In success.jsp page just fetch the data using expression language.

Then run the project and fire

http://localhost:8080/zpracticemvc/contact

By default the argu of requestParam are true ie valued needed can also make value as optional to enter

@RequestParam(name="email" , required= true) String userEmail

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

// FUll code of Handle method , till we have not used ModelAttribute

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

@RequestMapping(path="/processform" , method = RequestMethod.POST)

public String handleform(@RequestParam(name="email" , required= true) String userEmail ,

@RequestParam("userName") String userName ,

@RequestParam("password") String userPassword , Model model)

{

User user = new User();

user.setEmail(userEmail);

user.setUserName(userName);

user.setPassword(userPassword);

System.out.println("User mail is "+ userEmail );

System.out.println("User Name is "+ userName );

System.out.println("User Password is "+ userPassword );

//process data

//sending data to view using Model

model.addAttribute("name",userName);

model.addAttribute("email",userEmail);

model.addAttribute("password",userPassword);

return "success";

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 62 : VERY IMP @ModelAttribute annotation \*\*\*\*\*\*\*\*\*\*\*

USE 1 ::::::::::::::

@ModelAttribute annotation: VERY IMP it does 3 works

1.It fetches the data from form :( DOING WORK OF @Requestparam to fetch details from the form submission. )

2.It assign Fields to Entity class E.g. username, Password , Email to user data : helpful to display user.

3.It also send the data to next Jsp page: DOING WORK OF MODEL.ADDATTRRIBUTE

IMP: Mapping should be correct ie name used in "form" should be same as name used in User/Entity class

---

Intermeditate practice :using obj of class User to display and set fields instead of individual displaying fields display full user obj:

Now we are making the Object of Model (User class) , which has fieds userName, email,password .

Now Also : only single object of user will be added using model.addAttribute(user);

And in jsp file also accessing by user.userName etc ;

----

NOW JUST Replace all code using @ModelAttribute in one line code

Use 2 :::::::::

Another use of @ModelAttribute is to keep the repetive code of all jsp pages together,

eg we have used Header and desc in both pages success.jsp and contact .jsp

That is each time when the all jsp pages started first of all : @modelattribute method content will be added there ie before handles method run : see console

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

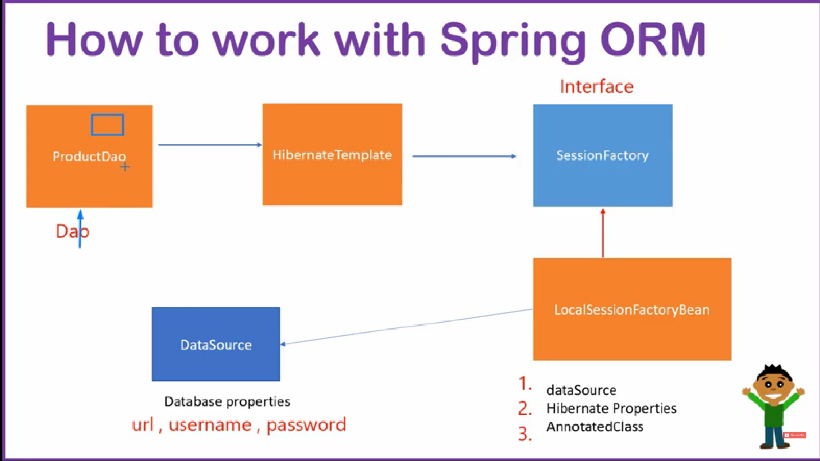
VERY IMPORTANT: SAVING DATA OF USER FROM HTML FORM TO DATABASE TABLE \*\*\*\*\*\*\*\*\*\*\*

Video 63 .....................................................

Spring orm + spring mvc configuration:

Follow layer Structures

**CONTROLLER LAYER -----> SERVICE LAYER -----> DAO LAYER (use hibernate obj to access) -----> DATABASE**



Configure Spring orm

**CONTROLLER LAYER -----> SERVICE LAYER -----> DAO LAYER (use hibernate obj**

**to access) -----> DATABASE**

Configuraation :

**STEP-1:**

add spring orm dependency , hibernate dependecy , mysql connector dependency.

**STEP- 2.**

Make the classes UserDao at DAO package to access Database

Now declare the method to save

//declare save method and object of hibernateTemplate

// @Autowired spring ; automatically inject obj of it , just need to decalre the bean of its class in .xml file

@Autowire

HibernateTemplate hibernateTemplate;

public int saveUser(User user)

{

int id=(Integer)this.hibernateTemplate.save(user) ;

return id; //return id of saved user.

}

the user object,

We need hibernateTemplate obj to access database (@Autowired)

Also enable the writing mode @Transational (wheneven we add/update the data in database this is used): Basically when we do change in database eg add, update, delete need to use @Transactional annotation in method. For this we need to do confifuration in .xml file ie **<tx:annotation -driven />**  schema, etc

\*\* Working discussed in orm and jdbc lectures

**STEP-3.**

Declare the User class as

@Entity so table create for it in database :

Also this time take take another attribute private int id

@id

@GeneratedValue(strategy=GenerationType.AUTO) in user Table

**STEP-4.**

CONTROLLER LAYER -----> SERVICE LAYER -----> DAO LAYER (use hibernate obj to access) -----> DATABASE

Make the service layer class : which access UserDao

Make UserService under springmvc.service package

Here we use the UserDao obj (@autowired) to save the created user

delcare UserDao obj and call its save method from here

eg

@Autowired //means obj will be given by spring

private UserDao userDao;

public int createUser(User user)

{return this.userDao.saveUser(user);

}

**STEP-5.**

CONTROLLER LAYER -----> SERVICE LAYER -----> DAO LAYER (use hibernate obj to access) -----> DATABASE

Now use the controller class

and declare the var of UserService and call the createUser method of it in handleForm method in class.

@Autowired

UserService userService;

public String handleForm()

{

/...

this.userService.createUser();

return "success";

}

So indirectly

Controller has (UserService obj) -----> UserService has(UserDAo obj) --------> UserDao (has hibernateTemplate obj ) ---> Then DATABASE accessed

IMP:

Since we have used the @Autowired for obj creating so ( may be not correct :@Componet used in their classes start , they must be component scan in .xml ) be declared in .xml files

Or use this shortcut Annotation instead of giving xml definition of beans for them

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

New Shortcut Notations:

@Service

---place this at UserService class (SERVICE LAYER),

so spring will understand this is service class and it will inject its obj to @Autowired use for obj creation

@Repository

---place this at UserDao class (DAO LAYER)

so spring will understand this is Database layer / data access obj class , so spring will inject its obj to @Autowired use for obj creation of UserDao

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

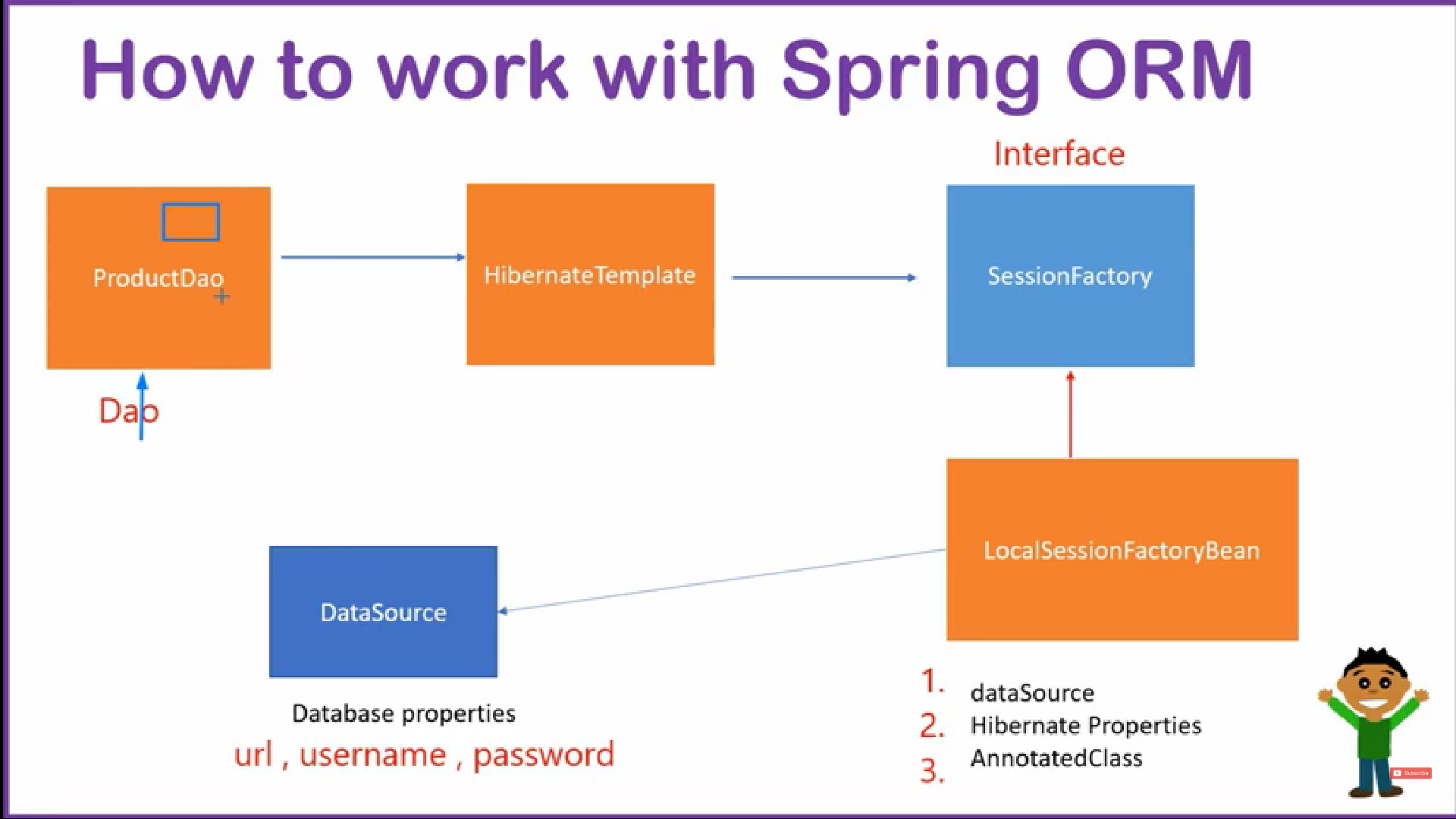
NOW TO MAKE ALL THESE CONFIGURATION WORKING / ANNOTAION WORKING , ENABLE THEM IN SPRING-SERVLER.XML FILE

WE ARE changing the context :component Scan to full project name "springmvc" : ie it will scan all its sub packages and make all annotations fully funtional

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

Also we have Configured every other classes like Dao, services, but not

**the hibernate template. 🡪 So declare this in .xml file**

\*\*\*\*\* Here we imported all configuration from springorm \*\*\*\* we have configured hibernate Template class.(Refer video of Spring orm )

Basically we need

----> HIBERNATE\_TEMPLATE class obj

----> which need LOCAL\_SESSION\_FACTORY\_BEAN class

(modify code of the entity to springmvc.model.User)

----> which further need DATA\_SOURCE (ds)

----> which further need the setting configuration of DRIVER\_MANAGER\_DATA\_SOURCE (ie setting driver, user Name, password of database used)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Wrting mode error --------------- comming because hibernatetempalte obj is not able to open session so

Lastly to do the update/add operation on database we need @Transactional , which need to define bean of

HibernateTransactionManager class in .xml which takes session factory (alredy defined)

\*\*\*\*\*\*\* So first put @Transactional above the saveUser method in UserDao class , or whereever u are calling chages in database like save , update etc

so declare this class ----------------> hibernateTrasactionManager class as a bean in.xml

**Also at top : write the <tx:annotation -driven />**

-->need to define the schema for this tx : so just copy paste p schema and replace the tx inplace of p :

-->Also last two lines copy and replace the "context" with "tx"

in last to line of context .

Now Submit the form : And see the updated table in your database

* Success --

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 64:

How to redirect in spring mvc: That is when we fire one page url it redirect to other page \*\*\*\*\*\*\*\*\*\*\*\*\*

0) HttpServletResponse: we can use this old method : take argument of HttpServletResponse in method and call response.sendRedirect(); //also handle exveption in this case

1)redirect prefix

2)Redirect View

Make a class Recontroller @Controller - this is enabled already in .xml by scanning the component Scan "zpractivemvc" package

Define the methods in ReController : to use redirect just return from method

Method 1: redirect prefix :::::::::::::::

@requestMapping("/one")

public String one()

{ system.out.println("this is one handler");

return "redirect:/enjoy" ; // this is redirect prefix : when one url is fired it internally fire enjoy

}

@RequestMapping("/enjoy")

public String two()

{

System.out.println("this is two method");

return "";

}

Method 2: Redirect View :::::::::::::

Using redirect view : we have to return the object of redirectView

@RequestMapping("/three")

public RedirectView three()

{

System.out.println("This is three handler runnning the redirect view");

RedirectView redirectView= new RedirectView();

redirectView.setUrl("https://www.google.com"); //("contact") // need to given relative path only contact, not “ /contact ” 🡪 it will lost the project name

return redirectView;

}

\*\*\*\*\*\* ZPRACTICEMVC IS FULLY WORKING CONCEPT OF SPRING MVC CONCEPTS \*\*\*\*\*\*\*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 65 :

PROJECT ON REDIRECT VIEW \*\*\*\*\*\*\*\*\* :

Search Engine

Using the concept of Search in google we have used our own search box

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 66 :

Handling the complex form \*\*\*\*\*\*\*\*\*\*\*\* :

Done under folder of Springmvcsearch

Type the form html code in complex\_form.jsp file : imported link form bootstap 4 :

Now made a newController FormController class

-using the handler method to tackle form submission(path,post)

-using ModelAttribute("student") Student student to do 3 things together

-Made a new class for Student in src.main.java

-Assign all attribute with same name that we had taken in html form

-And in success.jsp page we have printed Student

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 67: Binding Result to show errors\*\*\*\*\*\*

WE have just used the BindingResult object as argument to form handler method , and used its obj.hasErrors() method to return a view/jsp

And in our jsp we have placed the error using alert box of bootstrap

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 68-69

Handling the static file resources : ie handle images, adding the js, css files in folders\*\*\*\*

Make seperate folder of the css, js, images under resources folder

Define files and images under specific folders

Now to inlcude the required files in your jsp Configure mvc in xml and include their links in .jsp file also include jstl taglib in respective .jsp page

Here we are showing the static file in SearchController because it contains the home.jsp page where we are linkning the css and js files

In spring-servlet.xml file : we have enabled the mvc annotation

Just declare schemas for mvc by copy pasting p and last two lines of context

<mvc:annotation-driven />

<mvc:resources location="WEB-INF/resources/" mapping="/resources/\*\*" />

we have added the jstl taglib in home.jsp top line for using the jstl

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

Lastly

Apply the link tages in home.jsp

<link href="<c:url value="/resources/css/style.css"/> ">

<script src="<c:url value="/resources/js/script.js"/> "></script>

Also to declare the image in home.jsp

Use \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<img alt="my image" src="<c:url value="/resources/image/pass photo.jpg"/>" />

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 70 :

Basically we need to take the image form "from " submission using the @RequestParam("profile") also obj of CommonsMultipartFile file ,HttpSession s) are taken to store the collected image file and save this file on new folder/or server.

Stepss: Take CommonMultiple part ,configure it in .xml as bean and takes object of CommonMultipart inhandler method

File Uploading in mvc :::::In zpractive mvc project by mistake shifted from searchmvc to zpractive

(Later corrected this , present in both projects)

Using multipart resolver Commmonmultipart resolver

Do configuration in .xml file

Inject common file upload dependency in.xml

Inject common io dependency in .xml

Declare bean on multipart resolver in spring-servlet.xml

Make the form of file submit from bootstrap , then include the method action of form , also give the enctype="multipart/form-data"

Basically we need to take the image form "from " submission using the @RequestParam("profile") also obj of CommonsMultipartFile file ,HttpSession s) are taken to store the collected image file and save this file on new folder/or server.

We have various funtions for file eg file.getSize() , etc . and to save this file in specific folder

1. collet it in byte[]

then take the path wehere u want to save , and finally using fileoutput stream write the data of file and then close it

Also handles try and catch for write operation

@RequestMapping(value="/uploadimage" , method=RequestMethod.POST)

public String fileupload(@RequestParam("profile") CommonsMultipartFile file ,HttpSession s)

{

System.out.println("This is file upload handler");

System.out.println("File size:"+ file.getSize());

System.out.println("File name:"+ file.getOriginalFilename());

byte[] data= file.getBytes();

// we want to save this file on server , but what eclipse do is it save the project on wtpwebapps folder , there spring follow structre of project and save this file : this problem is not present when we deploy project on server

String path= s.getServletContext().getRealPath("/")+ file.getOriginalFilename();

System.out.println(path);

try {

FileOutputStream fos= new FileOutputStream(path);

fos.write(data);

fos.close();

System.out.println("File uploaded in desired folder");

} catch (IOException e) {

e.printStackTrace();

System.out.println("Uploading / file writtning error ");

}

return "filesuccess";

}

This is where our image is saved in : secret folder of eclipse

// we want to save this file on server , but what eclipse do is it save the project on wtpwebapps folder (secret folder) , there spring follow duplicate structure of project and save this file , infact run all projects from this portion only.

All project are run from this folders basically keeps copy of original project and run them form here

: this problem will not come/ present when we deploy project on real server

E:\coding\Practice\_eclipse\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\zpracticemvc\pass photo.jpg

File uploaded in desired folder

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

ALso if u want to display the collected image of form to next jsp page , take Model to send data from handler method and then in .jsp page: use img tag and use c: taglib of jstl to display image (include at top taglib of jstl)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 71 :::::

@Pathvariable annotation :\*\*\*\*\*\*\*\*\*\*

The @PathVariable is used to bind the uri varible to varible

eg http://localhost:8080/springmvcsearch/user/99/Shivam

@RequestMapping("/user/{userId}/{userName}")

public String getUserDetail(@PathVariable("userId") int userId, @PathVariable("userName") String userName)

{

System.out.println("USer ID : "+ userId + " "+ userName);

return "home";

}

hence spring bind int id with 99 and name Shivam

In SearchController , defined a method for it

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_----

Video : 72:

Handling exception in beckend \*\*\*\*\*\*\*\*\*\*\*\*

\*\*We are working in search controller , so disabled the js and img as they disturbing ele form home .jsp by putting comments

Here we used our used defined exception handling page.

we are using a handler method

@ExceptionHandler(value=NullPointerException.class)

public String exceptionHanderNull(Model m)

{

m.addAttribute("msg","Null Pointer exception has occured");

return "null\_page";

}

//similarly we can define the generic expression hander, the main task it to design a html form for error showing in the null\_page.jsp file.

If we want also send the status , include the

@ResponseStatus(value="HttpStatus.INTERNAL\_SERVER\_ERROR")

// this status can be seen in web page by inspecting it and looking console for error

We have not copied code from video so the null\_page code is unmannered view will come

?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 73: Exception Handle for full project \*\*\*\*\*\*\*\*

We make a new class MyExceptionHandler , which hanlde the all errors of Project

:Just paste the methods inside this class

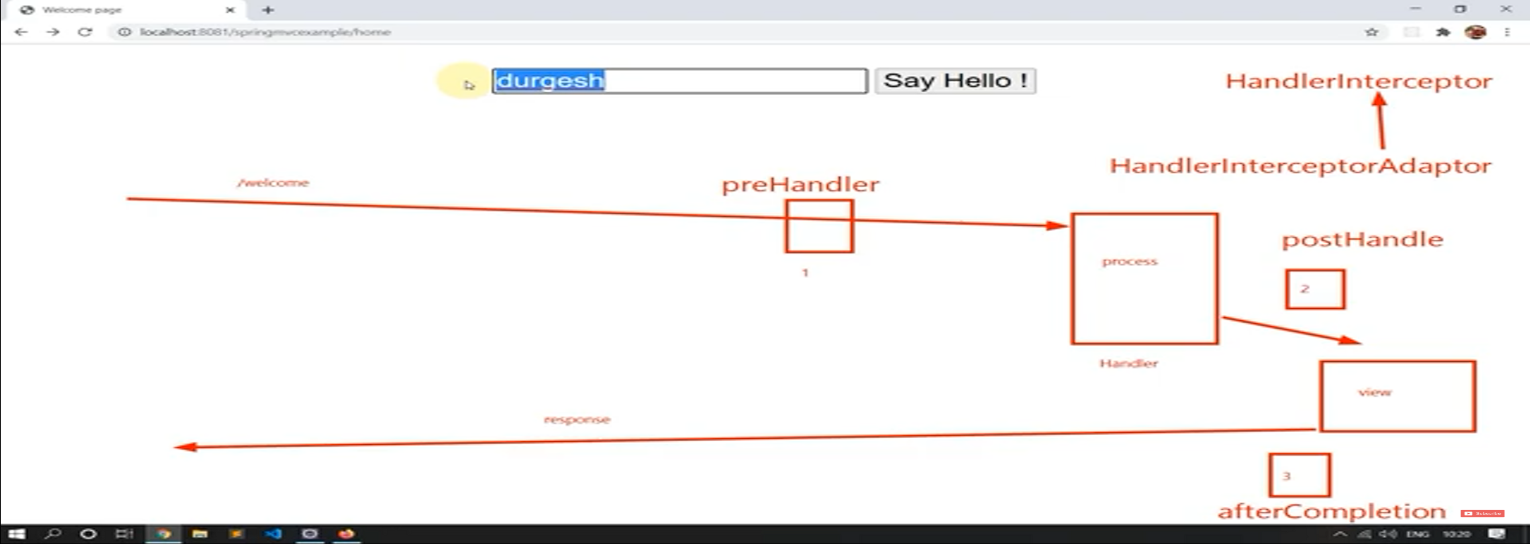
Use anntotation on class

@ControllerAdvice : tell the spring to use it as Exception handler for project and tell the exception according to methods defined inside it.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 74:

SPRING MVC INTERCPTOR \*\*\*\*\*\*\*\*\*\*\*:

refer diagram :

It can be used to check the validation of name data etc

Spring interceptor provides the working , at 3 places

a- before the request is mapped with handler method'

b- after handler method

c-after view is resolved , and readly to display response

Need to HandlerInterceptor interface has class

HanlderInterceptorAdaptor class ,

make our own class and extend this previous class

we have 3 method to override

-preHandler (used can be as filter to return back control if it return false , then request is send back , spirng mvc accepts that request has been handled , and no fulrter processing needed)

-postHandle (after hanlder and before view)

-AfterCompletion (after view)

Make new project as springmvcexample for concept of intercept

::::Configure all the project using the web .xml and spring-servlet.xml file and then inlcude the dependencies in pom.xml

Include the interceptor mvc tag in .xml servlet file and pass the path url , before which u want to use your interceptor pre handler method

<!-- this inereceptor will work beforee fireing welcome -->

<mvc:interceptors>

<mvc:interceptor>

<mvc:mapping path="/welcome" />

<bean class="interceptorExample.MyInterceptor" />

</mvc:interceptor>

</mvc:interceptors>

To apply the validation in pre handler method just use the class MyInterceptor and override its methods . prehandle

get the name used in form field and apply the condition on it. If String not satisy that condition then return the request , wiithout further processing . with the msg can be given by response.\_\_\_\_ methods

eg :

String name=request.getParameter("user");

if(name.startsWith("d"))

{

response.setContentType("text/html");

response.getWriter().println("<h1>Invalid NAme: Name should not Start with d");

return false; //return no need to accept the d strings

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

VIDEO 75 :

PROJECT APP: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Project on CRUD (Create , Retrieve(read) , Update ,Delete)

Usage concepts:

spirng mvc

Hibernate (jpa)

Need : Configure the tomcat, connect the msql database

Make a new Maven project and select the maven archtype-webapp project

package:com.operation

Name: productcrudapp

1. Correct the error in src/main/web-inf index.jsp error beacuse of tomcat absence, so choose project and select buildpath as add library -> server runtime -> tomcat 9 save it .

(if tomcat is downloaded and confiugred else also can add dependecy of tomcat (not used here)) .

2.

Add dependencies in pom.xml:

spring web-mvc

spring orm (object relational mapping )

spring hibernate

my sql connector

update the maven as project .

3.

configure the sping web.xml file

configure dispatcher servler ,and mapp it with url pattern .

4.

Make the spring-servlet.xml file

Declare the schema

And use the component scan to scan base package:

Make new Controller class.

Declare the class of view resolver : prefix and suffix are delcalred

Make the views folder inside the web-inf

Configure the database in spring-servlert.xml file(spring orm tutorial )

-> datasource bean : set database properties

->local session factory bean : set dataSource , hibernate properties

(Change the classes remove annotaed classes fomr list tags )

-> Make hibernate template

-> Also use hiberante transaction manager bean (used for update operation) and also enable tx:annotaion -driven , also declare schema of tx

------Done with Configuration of the project

Now make the DAo class in dao package, which use the obj of (@Autowired) hibernate template to connect with database .

Use @componentScan on Productdao class, so spring will deal with obj of dao class

Basically this obj will do all operartion with the database.

Also define the Product class in model package.

Make the product class as well

use @Entity annotation form product table in database of this class.

also make Id as auto incremented.

@Entity

public class Product

{

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

private int id;

.

.

.

}

make getter setter and toString as well

8888888888888888888888

Now since we have used the @Entity on the Product class

we need to give this class int .xml file under the annotated classes of session Factory.

<!-- c).annotedClasses-->

<property name="annotatedClasses" >

<list>

<value>

productcrudapp.model.Product

</value>

</list>

NEXT PART WE CONFIGURE MVC : and work with controllers

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 76:

we are here using the the new databases product crud which is defined in .xml DriveManagerDataSoucre bean

Inlcude the css, and javascript links from the bootstrap 4

aslo one line of resposive making :

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

Now we included the jstl in pom.xml to write the data in jsp files. Also included the taglib of jstl files in the jsp file , basically using the c: to write the jstl

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

Basically jstl is used to fetch the data at run time , and if data is not provided then it puts the default value.

We have coded the add-product jsp page. and also included the common base.jsp file using

<%@include file="./base.jsp" %>

We will design the basic jsp page of product.

We have take the Back Button functionality.

we have used <**a href="${pageContext.request.contextPath}** / " , this will keep the project url intact otherwise it will not work (back button) here, whereas online itcan work.

So include this : contextPath will become / in hosting it online server

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Now we desing the handler for this submission of form :

Handle\_product

Here we use handler which redirect us to another view. Also use the HttpServletRequest to get the contextPath , so upon redirecting path of project is not vanished. (unless we host it online)

Now we need to save the product ,

So use the product Dao controller , in controller to save the product

Now just use the productDao obj in home controller , and call the create method in the form-hanlder method

See Data will be saved in sql database.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Now

work on index.jsp file / home page file.

Include the base.jsp file

In add\_product handler method : inlcude the base.jsp file so css,javascript wil be included.

Make a form uisng the bootstrap Also design the back and add button.

Using the jstl : using c:for each loop

Also jstl taglib is used to put the the title from runtime to page, otherwise default title is put in it.

In add-product form handler

We use the HomeController method: We have used model to put the tittle at the run time using jstl.

In home handler used productDao object to get all stored record in database and displayed in index.jsp page using table format dynamically using jstl for each loop.

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

To work with delete function : just declare the delete handler and make it as Redirecting view , use product Dao to delete also set the redirecting url as

// here when we provide the url / it will remove the project name, so need to get the context path using httprequest

redirectView.setUrl(request.getContextPath()+"/");

Font awesome ion cdn to get button sumbols : copy the link and past to base.jsp along with css defined there

Now next open the Font Awseome webbsite to get icon , copy the code and paste it in table td tag using anchor tag

https://cdnjs.com/libraries/font-awesome

https://fontawesome.com/v4/icon/trash

now comming to index.jsp and addding another td tag to delete button addition - delete/ id --- this is going to fire the handler method with id at runtime hence ...processing it faster.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Video 77 : **Adding update feature to project**

Same copied the add\_product form , just change it with update , also used value tag to display previous availble values.

We need to declare button , just same as above to update the recored , in index.jsp.

Also make a handler for updating ,we have passed the product via model from index page to update\_form page, here we display old details that need to be filled to update.

Now we can use this form -action to be handled sepeately , but here we use same handler for update also.

Just modify the save product called function of ProductDao object.

Use saveorUpdate funtion

Here we have used the updateform .jsp so in form action give the url using pagecontext /handleproduct otherwise it will not update

<form action="${pageContext.request.contextPath }/handle-product" method="post">

we are using the login , that same handle controller is used for add and update,

while adding we dont pass the id, so new id is assigned and add

while updating we also pass the id , so it update already present id

Also make the type attribute as hidden , in update from .jsp page

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_--

All the changes of the functions are also reflected in my sql database as well

LEARNING AOP :

Video 78 Aspect Oriented Programming:

Its is used to run the segment of code befor and after the selected unit without disturbing the main code.

eg public PaymentService

{

//code before the function

public void makepayment() ---->JOIN POINT

{

}

//code after the function

}

CODE : make the new maven project and type as quick start

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1)add dependeccy of spring core, spring- context,

2)Make the payment Inteface also class

Search spring cre dtd to get xmlns for config.file

simply create the bean in xml ,so that object can be managed by spring.

using contextPath to acceess object direclty

Get Dependencies like

spring aop

aspectjruntime

aspectjweaver

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*IMPORTANT REMOVE THE SCOPE PART FORM THE ALL DEPENDECIEES OTHERWISE IT WILL NOT WORK

Update your project , and chek if dependecies of aspectrt etc are added or not , otherwise just force update them.

all the advices are kept in aspect class

Make new class Aspect , and keep method with annotation like @After @Before etc

If we pass the paremeter in makepayment() function :

then use .. in @before ohterwise patten will not match .

when paymentObject.makePayment(123);

@Before("execution(\* com.aop.services.PaymentServiceImpl.makePayment(..))")

public void printBefore()

{

System.out.println("Payment started...");

}

Refer video for more use Tutorail 78 Spring series Code with Durgesh Youtube