Developing spring boot MVC app having Service class with B.method

==========

home page

season page

find season

Welcome to Winter Season

GET

home

Story board

(a)

(FrontController)

=>Instead of writing b.logic in the handler method of the controller class, It is recomanded to write in seperate

service class b.method and delegate request to

service class b.method from the controller class handler method

by injecting service class obj to controller class obj using @Autowired annotation

(c) (q)

RequestMappingHandlerMapping

http://localhost:2525/MVCProj2-DataRendering

context path/WA name

(i) (z)

(b) (e) (h) DispatcherServlet

[/]

//serivce interface

public interface ISeasonFinderService{

public String findSeason();

@Controller

}

InternalResourceViewResolver

|--->prefix: /WEB-INF/pages/

|--->suffix: .jsp

View object (i)

/WEB-INF/pages/welcome.jsp

(k) (m) (p) (s)

public class Season FinderOperationsController{ @Autowired

private ISeasonFinderService seasonService;

(y) (b1) (d1)

(d?)

@Service("seasonService")

@RequestMapping("/") (f)

//service Impl class

public class Season FinderServiceImpl

public String showHome(){

View obj (a1) /WEB-INF/pages/display.jsp

resultMsg shared Memory -> summer (for S) (BindingAvare ModelMap object)

return "welcome"; (g)

implements ISeason FinderService{

}

(r?)

public String findSeason(){

@RequestMapping("/season")

...

(v)

welcome.jsp(WEB-INF/pages folder)

(!十

 get sesson

get season href="season

webpage (n) (on the browser) (0)

public String showSeason (Map<String, Object> map){ (t)

b.logic

//use service

(w)

(u) String msg=seasonService.findSeason(); //kee the result in model attribute

map.put("resultMsg", msg);

}

}

display.jsp(WEB-INF/pages folder) (< 1)

<%@page isELIgnored="false"%>

Season name :: ${resultMsg}

home

MVCBootProj2-DataRendering [boot]

> Deployment Descriptor: MVCBootProj2-DataRendering

>Spring Elements

> JAX-WS Web Services

#src/main/java

✓com.nt

>MvcBootProj2DataRenderingApplication.java

>ServletInitializer.java

com.nt.controller

> SeasonOperationsController.java

com.nt.service

>ISeasonFinderService.java

> SeasonFinderServiceImpl.java

#src/main/resources

static

templates

application.properties

src/test/java

//return LVN

webpage (e1) (on the browser)

return "display"; (x)

}

season name: rainy season home

>

>

JRE System Library [JavaSE-16]

>

Maven Dependencies

Deployed Resources

✓ src

main

> java

>

resources

✓ webapp

WEB-INF

✓ pages

application.properties

#Embedded Tomcat server port number server.port=4041

#InternalResourceViewResolver cfgs spring.mvc.view.prefix=/WEB-INF/pages/

spring.mvc.view.suffix=.jsp

//Serivce Interface

package com.nt.service;

public interface ISeasonFinderService {

}

public String findSeason();

//SeasonFinderServiceImpl.java (Service Impl class)

package com.nt.service;

//SeasonOperationsController.java

package com.nt.controller;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Controller; import org.springframework.web.bind.annotation.RequestMapping; import com.nt.service.Season FinderService;

@Controller

public class SeasonOperationsController { @Autowired

private ISeasonFinderService service;

@RequestMapping("/")

public String showHome() {

}

return "welcome";

> test

> target

WHELP.md

mvnw

mvnw.cmd Mpom.x

display.jsp

welcome.jsp

import java.time.LocalDate;

=>In spring boot web mvc application

we place images, java script files,

css files and etc. in public area by taking separate folders in "webapp" folder

✓ webapp

CSS

images

home.jpg

season_icon.jpg

is

```
import org.springframework.stereotype.Service;
@Service("seasonService")
public class Season FinderServiceImpl implements ISeasonFinderService {
@Override
public String findSeason() {
//get System date
LocalDate ld=LocalDate.now();
// get current month
int month=ld.getMonthValue();
//find season
if(month>=7 && month<=9)
return "Rainy Season"; else if(month>=3 && month<=6) return "Summer Season";
else
return "Winter Season";
@RequestMapping("/season")
public String showSeason (Map<String,Object> map) {
//use service
String msg=service.findSeason(); //keep results in Model Attribute map.put("resultMsg", msg);
//return LVN
return "display";
}
}
```

welcpome.jsp

```
<h1 style="color:red;text-align:center"> welcome.jsp(Home page)</h1>
<h3 style="color:green;text-align:center"><a href="season">find season</a></h3>
```

The images kept in webapp/images

can used in the <img> tag as shown below

in display.jsp

```
}
}
```

note: images, js, css of webapp folder are non-standard names

```
href="./">home   
<img src="images/home.jpg" width="80" height="100"/></a>
```

in welcome.jsp

```
"href="season"> show Season   
<img src="images/season_icon.jpg" width="80" height="100"/></a></b>
```

display.jsp

<%@ page isELIgnored="false" %>

<h1 style="color:green;text-align:center"> Season Name::${resultMsg}</h1>

<h3 style="text-align:center">home</h3>

You are screen snaring

■ Stop Share

To get current contextpath url + "/" we need to use this "./"

localhost:2525/MVCBootProj2-D X +

localhost:2525/MVCBootProj2-DataRendering/

welcome.jsp(Home page)

find season

=> if we deploy spring web mvc application in the external server then the Project name becomes the context path or name of the web application automatically

=> if we deploy spring web mvc application in the Embedded server then the application runs with out context path by default. if want to give some context path for this application then use "server.servlet.context-path" property of the application.properties

server.servlet.context-path=/Season FinderApp

localhost:2525/MVCBootProj2-D X

+ localhost:2525/MVCBootProj2-DataRendering/season

Season Name:: Rainy Season

home

Assignment :: Develop the spring boot mvc App that displays wish Message based on the current hour of the day for the hyperlink clicked in the home page

show wish message

home page

Good Morning/Good After noon/Good Evening/Good Night home dynamic webpage

Understanding end to end points of request paths

==========

a) request path of handler method must start with "/"

========

b) request path is case-sensitive in the handler methods of one or more controller classes

@Controller

public class Wish MessageController{

@RequestMapping("/report")

public String show Report() throws Exception{

return "show_report";

}

@RequestMapping("/REPORT")

public String showReport1() throws Exception{

return "show_report1";

}

}

d)

http://localhost:2525/Wish MessageApp/report ---> executes showReport() method
http://localhost:2525/WishMessageApp/REPORT ---> executes showReport1() method

c) One handler method can be mapped with multiple request paths

@RequestMapping({"/report1","/report3","/report2"})

public String show Report() throws Exception{

if any data is given with out annotation

param name then that data goes to param whose name is "value"

System.out.println("Wish MessageOperationsController.show Report()"); note:: The value param in @RequestMapping annotation is array type

return "show_report";

}

=> if param is array and u r intrested to maintain only one value

in that array then u can place that value with out { }.

http://localhost:2525/WishMessageApp/report1 ---> executes showReport() method
http://localhost:2525/WishMessageApp/report2 ---> executes showReport() method
http://localhost:2525/WishMessageApp/report3 ---> executes showReport() method

if the request path is "/" for the handler method then it becomes default handler method

in controller class i.e when no requestpath is given then this method executes automatically

are

το

This kind of methods veryful take the request and to display home page through handler method

@RequestMapping("/")

public String showHomePage() {

//return LVN

return "welcome";

=>if we take handler method with "/" request path to lanuch the home page ..then there is no need of taking index.jsp seperate to send the implicit request, More over this technique works

in both extenral tomcat server deployment and embedded tomcat server deployment of spring boot app.

http://localhost:4041/WishMessageApp. (with respect to embedded tomcat)

executes showHomePage() method

executes showHomePage() method

http://localhost:2020/BootMVCProj02-Wish MessageApp (with respect to external tomcat server)

e) if no request path is given for handler method .. the default request path will be "/"

```
@RequestMapping
public String showHomePage() {
//return LVN
return "welcome";
}
```

http://localhost:4041/Wish MessageApp.

(with respect to embedded tomcat)

executes showHomePage() method

executes showHomePage() method

http://localhost:2020/BootMVCProj02-Wish MessageApp

(with respect to external tomcat server)

if handler method that shows home page is having request path "/" then we need to give

request to that hanlder method having "./" as the url. (while sending request from hyperlinks and forms)

result page

```
<a href="./">Go to home</a>
```

f) Taking request path as "/" is equal to not taking any request path

```
@RequestMapping("/")
public String showHomePage() {
//return LVN
return "welcome";
}
```

is same as

```
@RequestMapping
public String showHomePage(){
//return LVN
return "welcome";
}
@PostMapping("/report") // first request path will be taken @GetMapping("/report"); //will be ignored
public String showReport1() {
System.out.println
("SeasonFinderOperationsController.show Report1() (GET /POSTmode)");
}
return "show_result1";
```

if we place multiple reuestpaths on the same handler method

ot

having both GET and POST modes then the first request path will be taken and other request paths will be ignored

f) Two handler methods of controller class can have same request path having two different

request modes like GET,POST

```
@Controller
public class Wish MessageController{
//@RequestMapping(value="/report", method=RequestMethod.GET) (or) (old style) @GetMapping("/report")
public String showReport() throws Exception{
System.out.println("Wish MessageOperationsController.showReport()"); return "show_report";
In @RequestMapping annotation
if no mode is specified the default mode is GET
}
```

GET mode request

```
//@RequestMapping(value="/report", method=RequestMethod.POST) (old style)
@PostMapping("/report")
public String showReport1() throws Exception{
System.out.println("WishMessageOperationsController.showReport1()");
return "show_report1";
}
}
```

POST

To send requests to the above handler methods

==========

==========

mode request

welcome.jsp

sends GET mode request

```
<a href="report"> get Report</a>
<br>
<form action="report" method="POST"> <input type="submit" value="Get Report"/>
</form>
```

note:: web applications/websites allow only browser as the client.. and this browser s/w can send only GET,POST mode requests

note:: Distributed Apps (web services) allow different types of clients including the browser So they can send different modes of requests like GET,POST,PUT, DELETE,PATCH and etc..

note1:: Spring boot MVC/spring MVC application are web application and they can take requests only from browwer and browser can give only GET,POST mode requests note2:: the request given from browser window by typing the url is GET mode request the hyperlink generated request from the webpage is GET mode request the form page can generate either GET mode or POST mode request.. default is GET mode note3: spring 4.x onwords @XxxMapping annotations are introduced as alternate for specifying request modes @RequestMapping annotation and these are recomaned to use. The @XxxMapping annotations are

@GetMapping spring MVC/spring BootMVC

annotations

@PostMapping can use obly these two @XxxMapping annotations

Spring Rest

(extension of

Spring MVC)

@PutMapping @DeleteMapping

and etc..

can use all the @Patch Mapping

five

@XxxMapping

annotations

GET Mode request is idempotent (safe to repeat the request)

POST Mode request is not idempotent (not safe to repeat the request)

Spring MVC/spring Boot MVC/sprWeb for developing web applications Spring Rest (extension of spring MVC) is for develop"Restful webServices (Distributed Applicaitons)

ing

g) What happens if two handler methods of a controller class having same request path

and same request mode?

raises exception representing the Problem ::

java.lang.IllegalStateException: Ambiguous mapping. Cannot map 'wishMessageOperationsController' method

com.nt.controller.Wish MessageOperationsController#showReport1()

to {GET [/report]}: There is already 'wishMessageOperationsController' bean method

com.nt.controller.WishMessageOperationsController#showReport() mapped.

Throws the exception

```
@Controller
public class Wish MessageController{
@GetMapping("/report")
public String showReport() throws Exception{
System.out.println("WishMessageOperationsController.showReport()");
return "show_report";
}
@GetMapping("/report")
public String showReport1() throws Exception{
System.out.println("WishMessageOperationsController.showReport1()");
return "show_report1";
}
```

}

=> Two handler methods of same Controller

class or different controller classes of a web application should

have

not same request path and request mode

h) In spring MVC/spring boot MVC maximum two methods can have same request path

one method with "GET" mode and another method with "POST" mode

refer the above example given in (f) point

=> Spring MVC/spring boot MVC apps are web applications So browser can give only GET,POST mode requests

=>Spring Rest/Spring Boot Rest Apps are Restfull apps (Distributed Apps) So the Client Apps can give GET,POST,PUT,PATCH,DELETE mode requests

default

The client for Web application is "browser" where as the clients for Distributed App is browser, desktop app, web application, mobile app, another distributed app and etc...

i) In spring MVC/Spring boot MVC maxiumum two methods of a controller class can be there with out request path by taking "/" as the request path

(One with GET mode and another with POST mode)

```
@Controller
public class Wish MessageOperationsController {
I
@GetMapping //Mapped with default request path "/" with GET mode
public String showHomePage1() {
//return LVN
return "welcome";
}
@PostMapping //Mapped with default request path "/" with POST mode
public String showHomePage2()
F
//return LVN
return "welcome";
}
}
```

To generate requests to the above handler methods of a controller class

j)

```
'roblem::
<br><br>
<a href="./">Get -Home page</a>
```

GET mode request

<form action="./" method="POST">

<input type="submit" value="POst -Home page"/> POST mode request

</form>

What happens of two handler methods of two different controller classes are having same request path?

@Controller

public class Wish MessageOperationsController {

@GetMapping("/all")

public String showAllData() {

return "show_report";

}

with same request mode

@Controller public class TestController {

@GetMapping("Zall") public String getAllTestsData() { return "show_report1";

}

}

The request paths of a controller must be unique either in same controller class or in multiple controller classes belong to a project i.e two handler methiods of one or more controller classes of project should not have same request paths

Caused by: java.lang.IllegalStateException: Ambiguous mapping. Cannot map

'wishMessageOperationsController' method

com.nt.controller.Wish MessageOperationsController#showAllData()

to {GET [/all]}: There is already 'testController' bean method

com.nt.controller.TestController#getAllTestsData() mapped.

controller

Solution:: along with method level request paths provide the class level global path using @RequestMapping annotation as shown

below

global path or global request path

@RequestMapping("/wish-operations")

@Controller public class Wish MessageOperationsController {

@GetMapping("/all")

public String showAllData() {

global path or global request path

@RequestMapping("/test-operations") @Controller public class TestController { @GetMapping("/all") ·

public String getAllTestsData() {

return "show_report";

}

}

return "show_report1";

}

method path (or)

method path or

method request path

method request path http://localhost:2020/BootMVCProj02-Wish MessageApp/wish-operations/all --> sends request to

showAllData() method of WishMessageOperationsController class

http://localhost:2020/BootMVCProj02-Wish MessageApp/test-operations/all --> sends request to

showAllTestsData() method of TestController class

k) The request given to one handler method of one controller can be forwarded to another method of same handler class or different handler class by using "forward: xxxx" concept which internally uses rd.foward(-,-) for forwarding the request. (This concept is called Handler methods chanining)

scenario1:

@Controller

public class Wish MessageOperationsController {

@GetMapping("/all")

public String showAllData() {

System.out.println("WishMessageOperationsController.showAllData()");

return "show_report";

}

@GetMapping("/report")

public String showHomePage1() {

System.out.println("WishMessageOperationsController.showHomePage1()");

//return LVN

return "forward:all";

}

http://localhost:2020/BootMVCProj02-WishMessageApp/report

scenario2:

@Controller

request goes to showHomePage1() method ---> from there

it goes showAllData() becoz "forward:all"

(Forward request handler method chaining can be done across the multiple methods

of different controller classes belonging to a same app)

@RequestMapping(/wish-ops")

public class Wish MessageOperationsController {

```
@GetMapping ("/report")
public String showHomePage1() {
//return LVN
System.out.println("Wish MessageOperationsController.showHomePage1()");
return "forward:/test-operations/all";
@Controller
@RequestMapping("/test-operations")
public class TestController {
@GetMapping("/all") public String getAllTestsData() {
System.out.println("TestController.getAllTestsData()");
return "show_report1";
}
}
}
```

http://localhost:2020/BootMVCProj02-Wish Message App/ wish-ops/report

request goes to showHomePage1() method ---> from ther

it goes to getAllTestsData() method of TestController class

becoz of "forward:test-operations/all" statement

}

1) We can perform handler methods chaining using "redirect:xxx" statement which internally

uses send Redirection with the support of response.sendRedirect(-) method =>forwarding request mode communication takes place directly from method1 to method2

=> send redirection mode communication takes place from method to method2 after having

network round trip with browser

browser

req1

response

/one

method1()

forward:all

either in

same controller or

/all

Here method2 can be

forwards

method2()

there

different controller of same MVC application

req1

(forwarding request)

(Source method and dest method

must be belong to same web application)

/one

browser

req1

method1()

redirect:all

res1

req2

resp 2

(SendRedirection)

Here method2 can be

/all

there either same controller or

method?()

different controller of

same MVC web application or different

MVC web application of same server or

different server belonging same machine or different machine s

(Source method and dest method can belong to same or different classes of same app or different apps running on the same server or different servers belonging to same machine or different machines)

=>while performing the forwarding request mode of handler method chaining the the source handler method and dest handler

method

must be there dealing with same mode of request

=>while performing the redirection mode of handler method chaining the source handler method and dest handler

method can be there dealing with same mode of requests or different modes of requests

What is the difference b/w Frontcontroller

==========

FrontController

==========

=======

and Controller/Handler class

==========

controller/Handler

a) It is Servlet comp or Servlet Filter comp

b) It is one per project or web application

c) contains ServletLife cycle methods like

init(),service(-,-), destroy() directly or indirectly

d) It acts as entry and exit point for all

requests and responses

e) Will be instantiated and managed by

ServletCotnainer (Servlet comp)

f) Identified with url pattern like "/"

which traps and takes all requests

DispatcherServlet's default url pattern is "/"

is

g) Entire spring mvc designed around

DispatcherServlet i.e it controls the entire navigation and also takes care of navigation

mgmt, view mgmt, model mgmt

a

========

a) It is java class

b) It is generally taken on 1 per module basis

c) contains user-defined methods as handler methods

mapped with request paths

d) It contains request delegation logic for different

url based requests to send or delegate reuqets to service,DAO classes.

e) Will be instantiated and managed by IOC container (Spring Bean)

f) Indenfied with global path given in

(class level

@RequestMapping and the handler methods

Controller classes are identified with global path

and the handler methods are identified with request paths

(method level) are idenfied with request path again given using @RequestMapping or @XxxMapping (@GetMapping,@PostMapping)

g) HandlerMapping, viewResolver, controller classes are hepler

classes for FrontrController comp.

h) In most of web frameworks inlcuding

spring MVC or spring boot MVC the

FrontController servlet is pre-defined

Servlet like "DispatcherServlet"

i) it always controller layer comp in MVC

application

Passing different kinds of data/values

from

h) In any web framework based

web application, the Controller/handler

classes are user-defined classes.

place

i) if we b.logics in handler methods of controller class then it falls

under model layer, if we place delegation logic in the same handler methods

then it falls under controller layer

Contrller comps to View Comps using Data Rendering Techniques (as Model attributes)

a) Passing simple values

b)

Passing collections and arrays

c)

Passing Model class obj

d) Passing collection of Model class objects

a) Passing simple values

In Controller class

Data Rendering is the process of passing data from controller class handler

methods to view comps through shared Memory(BindingAwareModelMap obj)

```
@GetMapping("/report")
public String sendData(Map<String, Object> map) {
// put simple values to Model attributes
map.put("name", "raja");
map.put("age", 30);
//return LVN
return "show_report";
}
```

Here the simple value will be

converted into a wrapper obj

Integer value.

In show_report.jsp (view comp)

```
<%@ page isELIgnored="false" %>
<b> model attributes data is :: ${name},${age}</b>
```

b) Passing collections and arrays

In controller class

```
@GetMapping("/report")
public String sendData(Map<String, Object> map) {
//put arrays,collections as the model attribute values
map.put("favColors", new String[] {"red","green","yellow"});
map.put("nickNames", List.of("raja","raj","maharaj"));
map.put("phoneNumbers", Set.of(999999L,888888L,7777777L));
map.put("idDetails", Map.of("aadhar No",7889999,"voterId",654665464));
7/return LVN
return "show_report";
}
```

In show_report.jsp(view comp)

```
<%@ page isELIgnored="false" import="java.util.*" %>
<h1> model attributes data is </h1>
<b> fav colors are :: <%=Arrays.toString(((String[])request.getAttribute("favColors"))) %></b><br>
<b> nick names are :: ${nickNames}</b><br>
<b>PhoneNumbers are :: ${phoneNumbers}</b> <br>
<b> idDetails are :: ${idDetails}</b><br>
```

use

Writing Java code in jsp pages is not good pratice.. So JSTL tags and EL togather to avoid or minimize java code in Jsp page

To use JSTL in our jsp pages

JSTL tags are useful for the Programmers to make the jsp pages as the java code less jsp pages and to improve readability of the jsp pages

(a) add jstl dependency in pom.xml file

```
<!-- https://mvnrepository.com/artifact/org.eclipse.jetty/apache-jstl -->
<dependency>
<groupId>org.eclipse.jetty</groupId>
<artifactId>apache-jstl</artifactId>
<version>11.0.0</version>
</dependency>
```

b) import jstl core library in jsp page and use its tags

JSTL :: Jsp Standard Tag Library

=>Jsp Tag library is a library where set of jsp tags are placed

|-->It gives 5 taglibaries

=> Jsp tag library is like Java Package

core --> for basic programming

sql

---> for DB connectivity

show_report.jsp

```
<%@ page isELIgnored="false" import="java.util.*" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<h1> model attributes data is </h1>
<b> fav colors are :: </b><br>
<c:forEach var="color" items="${favColors}">
${color}, counter
</c:forEach>
variable
<br>
items to
process
<b> nicknames are :: </b><br>
<c:forEach var="name" items="${nickNames}">
${name},
</c:forEach>
<br>
<b> phoneNumbers are :: </b><br>
<c:forEach var="ph" items="${phoneNumbers}">
${ph},
</c:forEach>
<br>
<b> idDetails :: </b><br>
<c:forEach var="id" items="${idDetails}">
${id.key}, ${id.value} <br>
</c:forEach>
```

formatting --> for formatting numbers, currency and etc..

xml --> for processing xml content

functions

--> for String manipulation

JSTL Tag library

==========

enhanced for loop

core

using tags

fmt

sql

xml

functions

=> Every Jsp tag library is identified with its tag lib uri..

=> In jsp page/file, the jsp tag library can be imported by specifying its uri

with the support of <%@taglib uri="....."%> tag

taglibrary uri

==========

http://java.sun.com/jsp/jstl/core

http://java.sun.com/jsp/jstl/fmt

http://java.sun.com/jsp/jstl/sql

http://java.sun.com/jsp/jstl/xml http://java.sun.com/jsp/jstl/functions

=>In the deployment of spring boot web mvc application if the content of the webapp folder is not moving to root folder

of the web application then we need to perform some explicit cfgs as shown below

Right click on the project ---->properties ---> Deployment and Assembly ---> add --> folder --->

select src/main/webapp ----> observe /src/main/webapp folder mapping to "/" ---> ... --> ....