browser window

request uri (3)

request

network

(2) [FrontController Servlet]

(4) Dispatcher Servlet *

Spring MVC/ Spring Boot MVC Application

(1)

<uri,path Handler Mapping (5)

(7)

controller class bean id, handler method signature

(8) call to handler method

(6?)

controller/Handler class (@Controller)

@RequestMapping("/<path>") public <RT> method('<params>{

...

(9 processing logic

(or reg delegation logic to service,DAO

(/-url pattern [System services, Navigatin logics

(10) LVN (Logical View name)

ViewResolver

(11) LVN

(12) physical view comp details (View obj)

physical view comp

note: The process of keeping results in one or another

scope is called working with model attribute s

dynamic webpage

(15) response

...

]

(16)

(14)

(13 prefentation

logic

app,

In the Spring Boot MVC the following operations takes place automatically a) DispatcherServlet registration enabling load-on startup on it (this uses programatic registration of Servlet comp technique)

b) DispatcherServlet created IOC container of type ApplicationContext (This takes place from the init() of

DispatchServlet comp)

c) HandlerMapping component configuration

(RequestMappingHandlerMapping class becomes

spring bean through AutoConfiguration process)

d) ViewResolver Configuration

(InternalResourceViewResolver class becomes

spring bean through AutoConfiguration process)

n

=>3 types of registerting servlet comp with

Servlet container

a) Declarative approach (using web.xml)

b) Annotation approach(using @WebServlet)

c) Progamatic approach (using ServletContext.addServlet(-))

in

*** symbol diagram indicates that they**

are automatically generated or configured

classes of spring boot MVC web application.

**** symbol indicates that they are programmer**

(Spring boot MVC app uses the third approach nothing but programmatic approach to register DispatcherServlet comp with SErvletContainer)

developed user-defined comps.. even service, DAO classes are are user-defined classes

e) DispatcherServlet created IOC container performs pre-instatiation of singleto scope spring beans (service classes, controller classes, view resolver, HandlerMapping, DAO clases/Repository Impl classes and etc..)

and etc..

=>spring-web-<ver>.jar,spring-webmvc-<ver>.jar files represent spring web mvc framework/spring mvc framework (For spring mvc App) represents spring boot statter forspring boot MVC app (For spring boot mvc App)

=>spring-starter-web-<ver>.jar file

(this starter gives the above two jar files and other supporting jar files)

Controller class/Handler class

======

=> It is java class annotated with @Controller

=> we generally take this calss on 1 per module basis

spring web mvc=spring mvc

spring boot mvc =spring mvc ++

=>This class can contain multiple handler method annotatedwith

@RequestMapping having request path (it is /<path>)

=>The

handler

=>In small web application the handler class handler methods directly process the request having b.logic where as in medium and large scale apps the handler methods contain delegation logic to interact with service,dao classes

methods can have flexible signature i.e method name, retrun: type, param types,params count is completely our choice.

=> Every handler method of handler class is identified with its request path ("/<path>) that is given

in @RequestMapping annotation... we generate request to handler methods by specifying their request paths //sample controller class

==========

@Controller

in the request urls

public class MyController{

request path must start with "/"

parameters

@RequestMapping("/home")

public String showHome(HttpServletRequest req){

University Project

note::

DispatcherServlet is readymade servlet comp given by Spring/spring boot MVC API which will be configured /register with ServletContainer automatically having the url pattern "/" by enabling <load-on-startup> in Programatic approach of the servlet registration

|---> Adminissions module |----> Academics module

note:: HandlerMapping component (REquestMappingHandlerMapping), ViewResolver component(Internal ResourceViewResolver) will become spring beans automatically through AutoConfiguration process done by IOC container which DispatcherServlet

|----> Examinations module |----> Sports Module

is given by

=>DispatcherServlet is front controller servlet which is 1 per Project (1 per spring/spring boot mvc app)
=>Controller/Handler is java class which is 1 per module

=>service class is java class which is 1 per module

=>DAO class is java class which is 1 per db table

http://localhost:3131/Proj01/home (Incomding request URL)

....

method name

protocol

logic for request processing or

web server host & port number

project

Based on this

name

....

logic for request delegation

request goes to handler method

return type

}

note:: one controller class can have any no.of handler methods.

}

Allowed return types are

=====

=========

note: Every handler method is indentified with its request path

given in @RequestMapping annotation. each incoming request will mapped to handler method based on this request path

=>nearly 20+ return types are allowed for handler method .. but most of them are given by keeping spring rest(required for Restfull webservices) in mind.. In spring MVC we use very little number of return types.

In

Controller method return value

@ResponseBody

HttpEntity<B>, ResponseEntity<B>

HttpHeaders

String The best in spring mvc]

View

java.util.Map,

org.springframework.ui.Model

@ModelAttribute

ModelAndView object (Legacy)

void

DeferredResult<V>

Description

The return value is converted through HttpMessageConverter implementations and written to the response. See @ResponseBody.

The return value that specifies the full response (including HTTP headers and body) is to be converted through HttpMessageConverter implementations and written to the response. See ResponseEntity.

For returning a response with headers and no body.

A view name to be resolved with ViewResolver implementations and used together with the implicit model-determined through command objects and @ModelAttribute methods. The handler method can also programmatically enrich the model by declaring a Model argument (see Explicit Registrations).

A view instance to use for rendering together with the implicit model- determined through command objects and @ModelAttribute methods. The handler method can also programmatically enrich the model by declaring a Model argument (see Explicit Registrations).

Attributes to be added to the implicit model, with the view name implicitly determined through a RequestToViewNameTranslator.

`An attribute to be added to the model, with the view name implicitly determined through a RequestToViewNameTranslator.`

Activate Windows

Note that @ModelAttribute is optional. See "Any other return value" at the end this table

Go to Settings to activa

The view and model attributes to use and, optionally, a response status.

A method with a void return type (or null return value) is considered to have fully handled the response if it also has a ServletResponse, an OutputStream argument, or an @ResponseStatus annotation. The same is also true if the controller has made a positive ETag or last Modified timestamp check (see Controllers for details).

If none of the above is true, a void return type can also indicate "no response body" for REST controllers or a default view name selection for HTML controllers.

Produce any of the preceding return values asynchronously from any thread-for example, as a result of some event or callback. See Asynchronous Requests and

`esult.`

Spring Rest = spring mvc++

spring boot Rest = sprng boot mvc++

note: officially there is no module called spring rest/spring boot rest.. the spring mvc/spring boot mvc taken for Restful web services programming is called spring Rest | Spring boo rest

restful web service app=Distributed App

note: spring mvc/spring boot mvc module can be used to develop the web applications and also to develop the Restful web services (Distributed Applications)

Restful webservice app = MVC web application + +

`>`

`Callable<V>`

ListenableFuture<V>,

Produce any of the above return values asynchronously in a Spring MVC-managed thread. See Asynchronous Requests and Callable.

`Alternative to DeferredResult, as a convenience (for example, when an underlying java.util.concurrent.CompletionStage<V service returns one of those).`

`java.util.concurrent.CompletableFuture`

`<V>`

`ResponseBodyEmitter, SseEmitter`

StreamingResponseBody

Reactive types - Reactor, RxJava, or others through

`ReactiveAdapterRegistry`

Any other return value

Emit a stream of objects asynchronously to be written to the response with dows HttpMessageConverter implementations. Also supported as the body of as to activa ResponseEntity. See Asynchronous Requests and HTTP Streaming.

`gs`

Write to the response OutputStream asynchronously. Also supported as the body of a ResponseEntity. See Asynchronous Requests and HTTP Streaming. Alternative to DeferredResult with multi-value streams (for example, Flux, Observable) collected to a List.

For streaming scenarios (for example, text/event-stream, application/json+stream), SseEmitter and ResponseBody Emitter are used instead, where ServletOutputStream blocking I/O is performed on a Spring MVC-managed thread and back pressure is applied against the completion of each write. See Asynchronous Requests and Reactive Types.

Any return value that does not match any of the earlier values in this table and that is a String or void is treated as a view name (default view name selection through RequestToViewNameTranslator applies), provided it is not a simple type, as determined by BeanUtils#isSimpleProperty. Values that are simple types remain unresolved.

The possible parameter types are

(25+ are there)

Controller method argument

WebRequest, NativewebRequest

javax.servlet.ServletRequest, javax.servlet.ServletResponse

`javax.servlet.http.HttpSession`

javax.servlet.http.PushBuilder

`java.security.Principal`

HttpMethod

`java.util.Locale`

java.util.Timezone + java.time.ZoneId

java.io.InputStream, java.io.Reader

java.io.Outputstream, java.io.writer

`@PathVariable`

`@MatrixVariable`

@RequestParam

@RequestHeader

Description

Generic access to request parameters and request and session attributes, without direct use of the Servlet API.

Choose any specific request or response type-for example, servletRequest, HttpServletRequest, or Spring's MultipartRequest, MultipartHttpServletRequest.

Enforces the presence of a session. As a consequence, such an argument is never null. Note that session access is not thread-safe. Consider setting the RequestMappingHandlerAdapter instance's synchronizeonSession flag to true if multiple requests are allowed to concurrently access a session.

Servlet 4.0 push builder API for programmatic HTTP/2 resource pushes. Note that, per the Servlet specification, the injected PushBuilder instance can be null if the client does not support that HTTP/2 feature.

Currently authenticated user-possibly a specific principal implementation class if known.

Note that this argument is not resolved eagerly, if it is annotated in order to allow a custom resolver to resolve it before falling back on default resolution via HttpServletRequest#getUserPrincipal. For example, the Spring Security Authentication implements Principal and would be injected as such via HttpServletRequest#getUserPrincipal, unless it is also annotated with @AuthenticationPrincipal in which case it is resolved by a custom Spring Security resolver through Authentication#getPrincipal. Activate Windows The HTTP method of the request.

The current request locale, determined by the most specific LocaleResolver available (in effect, the configured LocaleResolver or LocaleContextResolver). The time zone associated with the current request, as determined by a LocaleContextResolver.

For access to the raw request body as exposed by the Servlet API. For access to the raw response body as exposed by the Servlet API.

For access to URI template variables. See URI patterns.

For access to name-value pairs in URI path segments. See Matrix Variables.

For access to the Servlet request parameters, including multipart files. Parameter values are converted to the declared method argument type. See @RequestParam as well as Multipart.

Note that use of @RequestParam is optional for simple parameter values. See "Any other argument", at the end of this table.

For access to request headers. Header values are converted to the declared method argument type. See @RequestHeader .

`@CookieValue`

For access to cookies. Cookies values are converted to the declared method argument type. See @cookieValue.

@RequestBody

`HttpEntity<B>`

@RequestPart

`java.util.Map (best of best)`

org.springframework.ui.Model, org.springframework.ui.ModelMap

RedirectAttributes

@ModelAttribute

`Errors, BindingResult`

(Best)

Good in special cases

SessionStatus + class-level @SessionAttributes

`UriComponentsBuilder`

@SessionAttribute

@RequestAttribute

Any other argument

For errors

generation

For access to the HTTP request body. Body content is converted to the declared method argument type by using HttpMessageConverter implementations. See @RequestBody.

For access to request headers and body. The body is converted with an HttpMessageConverter. See HttpEntity.

For access to a part in a multipart/form-data request, converting the part's body

with an HttpMessageConverter. See Multipart.

For access to the model that is used in HTML controllers and exposed to templates as part of view rendering.

Specify attributes to use in case of a redirect (that is, to be appended to the query string) and flash attributes to be stored temporarily until the request after redirect. See Redirect Attributes and Flash Attributes.

For access to an existing attribute in the model (instantiated if not present) with data binding and validation applied. See @ModelAttribute as well as Model and DataBinder.

Note that use of @ModelAttribute is optional (for example to set its attributes).

For access to errors from validation and data binding for a command object (that is, a @ModelAttribute argument) or errors from the validation of a @RequestBody or @RequestPart arguments. You must declare an Errors, or BindingResult argument immediately after the validated method argument.

For marking form processing complete, which triggers cleanup of session attributes declared through a class-level @sessionAttributes annotation. See @SessionAttributes for more details.

For preparing a URL relative to the current request's host, port, scheme, context path, and the literal part of the servlet mapping. See URI Links.

For access to any session attribute, in contrast to model attributes stored in the session as a result of a class-level @sessionAttributes declaration. See @SessionAttribute for more details.

For access to request attributes. See @RequestAttribute for more details.

If a method argument is not matched to any of the earlier values in this table and

it is a simple type (as determined by BeanUtils#isSimpleProperty, it is resolved as

a @RequestParam. Otherwise, it is resolved as a @ModelAttribute.

Handler method of the controller or Handler class can have 0 or more parameters, we generally take them as the per the need and as needed.

es

=>Spring Boot MVC web application, if we place controller class under root pkg where @SpringBootApplication class

is placed then the @Controller classes will be scanned automatically by IOC container to make them as spring beans (requal practice)

HandlerMapping component

=>It is the component with whom DispatcherServlet handovers the request.. This component uses the reflection api support interally to search for handler method in all @Controller classes finding the matched handler method and gives that @Controller class bean id and handler method signature back to DispagcherServlet to make Dispatched hanlder method on @Controller class object.

are

that

=> All HandlerMapping clases ready made classes implementing org.springframework.web.servlet.HandlerMapping(1) directly or indirectly..

=> spring MVC api gives lots of pre-defined HandlerMapping classes for different situations.. Mostly there will not be

any need of developing custom Handler Mapping classes.

=>BeanNameUrlHandlerMapping

=>ControllerClass NameHandlerMapping

=>SimpleUrlHandlerMapping

=>DefaultAnnotationHandlerMapping

(Defult in annotation driven cfg upto 4.x)

=>RequestMappingHandlerMapping

(Default in annotation driven cfgs from spring 5.x)

=>useful in xml driven cfgs

拳

of spring mvc apps

in

Useful annotaion driven cfgs

of spring mvc apps

It is also default in spring boot MVC Apps

of

conclusion:: In latest version spring mvc apps and in spring boot mvc Apps directly or indirectly we use RequestMapping

=>In Spring boot MVC application, the RequestMappingHandlerMapping class comes automatically

(as Spring bear through autoconfiguration.. and will be pre-instantiated automatically by using the DispatcherServlet

created IOC container.

ViewResolver

==========

HandlerMapping

note: spring mvc apps can be deployed only in external servers like tomcat,wildfly,.. note: spring boot mvc Apps can be deployed either in external servers or in Embedded servers of app itself. and

with

=>This comp takes LVN (Logical view name) given by Controller throuhg DS maps/links physical view comp name and location retake the View object (View Interface impl class obj) having that physical view comp name

and location..

=> ViewResolver does not execute View comp.. ViewResolver indentifies the name and location of physical view compnaves those details to DispatcherServlet in the form View object.

=> All ViewResolver comps are the classes implementing org.springframework.web.servlet.ViewResolver(1) directly

or indirectly..

=> Most of the times we work with ready made ViewResolvers i.e there is no need of developing user-defined ViewResolvers..

The spring boot mvc gives

Embedded Tomat (default) Embedded Jetty Embedded undertow

servers

=>InternalResourceViewResolver (default in spring boot mvc applications)

=>UrlBasedViewResolver

=>ResouceBundlerViewResolver

=>XmlViewResolver

=>TileViewResolver

=>BeanNameViewReoslver

and etc..

=> InternalResourceViewResolver will become spring bean automatically through auto configuration ..To suply inputs

to this ViewResolver take the support application.properties or yml file.

In application.properties

is

spring.mvc.view.prefix=/WEB-INF/pages/

(This location of physical view comp)

spring.mvc.view.suffix=.jsp

(This extension or type of view comp)

([prefix+ lvn+ suffix)

if the controller supplied LVN view name is "home" then the physical view comp name will be

"/WEB-INF/pages/home.jsp" (prefix + lvn + suffix)

prefix

[Location]

suffix

[extension]

LVN :: Logical View name

comp

View obj /WEB-INF/pages/home.jsp

prefix

suffix

lvn

=>This physical view name and location goes to DS from ViewResolver in the form of View obj (View (I) impl class obj)

and

=> DS gathers physical view name and location from the View object uses rd.foward(-) or some other technique internally to execute physical view comp.

(forward(-,-) on RequestDispatcher obj)

if u want to configure other ViewResolver classes as additional classes then we need to use @Bean methods

support in @Configuration class or @SpringBootApplication class.

=>spring MVC/spring boot mvc recomands to keep jsp files /pages in the

private area of the web application.

=> WEB-INF folder and its sub folders are called private area of the web application

=> Outside WEB-INF area folder and sub folders are called public area of the web application

=> Only the underlying server/container can use private area comps directly.. if any outside request wants to

to get Permission

use the same comp then we need give special instructions container. (eg: servlet comps) =>Public areaweb comps can be used by outsider and container/server directly with out having any permissions (eg: html,jsp files placed outside of WEB-INF folder)

=> so for we have placed html,jsp files in the public area of the web application.. but it is

recomanded to place in private area of the web application to get the following advantages.

a) No outsider can give direct request to jsp pages .. if jsp page is reading and displaying reuest scope

data given by servlet comp.. since there is no possibility of giving direct request..so the jsp page will never display null values.

(ugly values)

browser rea

resisp

servlet

DB S

b) we are able hide jsp file/page from outside!.. i.e we can hide techology of the website from outsiders. (protection from hackers)

area

note:: Generally if jsp page is there in private then its cfg in web.xml file is mandatory having url pattern..

But the Internal ResourceVie solver of spring MVC can locate/identity the jsp pages/files of priva

area directly with out mapping the jsp page /file with url pattern.in web.xml

we place jsp pages in spring MVC/spring Boot MVC application in the following folders WEB-INF/pages(prefer this) fodler or WEB-INF/jsp folder or WEB-INF/view folder =>Here "pages", "jsp", "view" folders are user-defined folders. we can take any name. (non-standard folders) =>WEB-INF,classes, lib folders and web.xml file namesare standard names in web application development (fixed names) Story Board of First spring Boot MVC Web application to display the home page (jsp of private area) for given implicit request. (Code based flow through story board) (MVCBootApp1)

(0) Deployment activities

browser window

http://localhost:3031/Proj01/home

Dynamic webpage

(15)

welcome to spring boot MVC

(1)

DispatcherServlet (FrontContrlier)

path

(2) (5) (8)

url pattern:/

(11)

(14)

(3) (default) REquestMÀppingHandlerMapping

@Controller

(web app root folder)

FirstApp

>WEB-INF

-->*.html |-->*.jpg public area

lib

|-->*.jar

classes

private area

|--->*.java |---> *.class

pages

|-->*.jsp

-->web.xml

=>In Eclipse IDE only webapp folder (src/main/webapp folder) content becomes publc area content on the deployment

=> the src/main/java content, WEB-INF folder content internally becomes private area content on the deployment

MVC-FrontController-JEE

Deployment Descriptor: Archetype Created Web Application #src/main/java

com.nt.controller

> LinksHandler.java

com.nt.frontcontroller >NitFrontControllerServlet.java

com.nt.serivce

> ILinksService.java

goes to WEB-INF/classes folder internally (becomes private area content)

> LinksServiceImpl.java

>

src/test/java

> JRE System Library [JavaSE-16]

> Maven Dependencies

> Deployed Resources

src

main

> java

webapp

goes to web application's root

WEB-INF

becomes private area content

> test

intenrally

target

M pom.xml

public class ShowHomeController{

(6)

request path

@RequestMapping("/home")

public String showHomePage(){

//return LVN

return "welcome"; (7)

(4?)

(searches for the handler method

in @Controller class whose request path' "/home"

Xweb.xml index.jsp

show_languages.jsp show_wish.jsp

folder (becomes public area content

internally)

(1)

(9)

InternalResourceViewResolver (default) |--->prefix: /WEB-INF/pages/ |---->suffix: .jsp

View Object (10)

/WEB-INF/pages/welcome.jsp

prefix

LVN

suffix

}

WEB-INF/pages/welcome.jsp

et3's

<h1>welcome to 'spring Boot MVC </h1>

Proj01

|--->src/main/java]

--->com.nt.controller

|-->ShowHomeController.java

|---->rc/main/webapp]

Spring Boot MVC allows to use web application in two ways

jetty or undertow

Spring boot

1) as standalone web application =>This application runs in the Embedded Tomcat server of the Application itself

=> No need of arranging seperate server

=>Very useful in dev, test env.. where we are not ready to spend seperate

money/memory for installation of webserver

=> This application can be packed as jar file which contains

web application + Embedded Tomcat server.

2) as Deployable web application

jettty /undertow

=> This application runs in External server of the application

=> It is seperate war file to manage

=> This application can be deployed any web server or applications server

=> This is every useful in Production env..

=> In cloud based application development and deployment the companies are preffering

--->WEB-INF

--{pages] (12) |--->welcome.jsp

In Spring boot mvc app the default Embedded server is Tomcat, But we can also configure jetty,undertow as the embedded servers

spring mvc supports only deployable application developoment where as spring boot mvc supports both standalone web app development and deployable web application development

=>The server s/w that is installed separately and configured for our web application is called External server

eg: Installed tomcat, jetty or underow and etc.. servers =>The server s/w that comes as InMemory server in the execution of Spring boot mvc app is called Embedded server

to use EmbeddServer in the dev, test, mode/env.. of the Project and the same companies are using external servers for uat, prod mode/env.. of the Project.

(in cloud)

Procedure to develop First Spring boot MVC App that shows the private area jsp file as the home page of

eg:: Embedded Tomcat, Embedded Jetty and etc..

the web application

step1) make sure that Tomcat 10.xis configured with Eclipse IDE.

window menu ---> preferences ---> servers ---->Runtime env ----> add --->select apache Tomcat

Name:

Apache Tomcat 10.x

Tomcat installation directory:

E:\Tomcat 10.X

JRE:

Workbench default JRE

10.x

--->

=>The latest tomcat version is 10.x which deals with

servlet 5.x api having jakarta.servlet,jakarta.servlet.http

and etc.. pkgs ... Spring boot mvc latest version

jakarta.servlet pkgs.

use toicmat 10.1

Browse...

apache-tomcat-9.0.46 Download and Install...

is supporting

Installed JREs...

----> Finish.

spring boot 3.x is compitible with

servlet 5.x and tomcat 10.x

Go to servers tab ----> click on hyperlink ---> select apache Tomcat tomcat 10.x->next -->next --> finish.

step2) Create spring boot starter Project as shown below

↓

Service URL

https://start.spring.io

Name

BootMVCProj01-FirstApp

Use default location

Location

G:\Worskpaces\Spring\NTSPBMS615-BOOT\BootMVCProj01-Firs Browse

Type:

Maven

Packaging:

War

Java Version:

11

✓ Language:

Javal

Group

nit

Artifact

BootMVCProj01-FirstApp

Version

0.0.1-SNAPSHOT

Description

MVC App

Package

com.nt

Working sets

Add project to working sets

New...

Spring Boot Version: 3.x

Frequently Used:

JDBC API

Lombok

MySQL Driver

Spring Data JPA

Spring Data MongoDB

Available:

Selected:

web

X Spring Web

▾ Messaging

WebSocket

▾ Template Engines

Thymeleaf

Apache Freemarker

▾ Testing

Testcontainers

▾ Web

Spring

Web

Spring Reactive Web

D.....

next ---> next ---> finish.

step3) Add the following entries in application.properties application.properties

=>if jar is selected here we can run the

web application only as the standalone web App

that contains embedded tomcat server

=>if war is selected here we can run the

as

web application as the standalone web App that contains embedded tomcat server and also normal deployable web application in external server.

In tomcat server, the servlet container logical name

is CATALINA and Jsp Container logical name is JASPER

#Embedded server port (default is 8080)

server.port=4041

note:: Internal ResourceViewResolver is given

to resolve the private area servlet,jsp comps as view

comps by appending prefix, suffix values to the LVN.. It is

#ViewResolver inputs (prefix,suffix) for default Internal ResourceViewResolver not designed for html comps.. So using this ViewResolver

we can not take html files as the view comps.

spring.mvc.view.prefix=/WEB-INF/pages/

spring.mvc.view.suffix=.jsp

step4)

Develop the controller class having handler method

ShowHomeController.java

```
package com.nt.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class ShowHomeController {
@RequestMapping("/home")
public String showHome() {
//retrun LVN
return "welcome";
```

=>One controller class can have 1 or more handler methods

=> Every handler method is identified with its request path given in @RequestMapping(-) annotation

```
}
```

}

step5) develop physical view comp

sro

main

> java

>

resources

✓ webapp

WEB-INF

welcome.jsp

✓ pages

created manually

welcome.jsp

<h1 style="color:red;text-align:center">WElcome to spring boot MVC</h1>

step6) Run the web application as deployable web application in the external Tomcat server..

Right Click on the Project ---> run as ---> run on server ---> select Tomcat ---> next ---

localhost:2525/MVCBootProj1-SX

+

You are screen sharing

■ Stop Share

localhost:2525/MVCBootProj1-Showing HomePage/home

Welcome to Spring boot MVC First App

MVCBootProj1-Showing HomePage [boot]

not

if we get the 404 error [/WEB-INF/pages/welcome.jsp] is found through it is physically avaiable, then perform the following operation

Right click on the Project ---->properties ----> deployment assembly ---->

add --->folder --> select src/main/webapp folder --->next --> next --> finish.

>

u. Deployment Descriptor: MVCBootProj1-Showing HomePage

›

Spring

Elements

> JAX-WS Web Services

#src/main/java

✓com.nt

>ServletInitializer.java,

com.nt.controller

> MvcBootProj1Showing HomePageApplication.java => contains main(-) with the code SpringApplication.run(-) to create Embedded Server and IOC container, Very useful in Standalone App execution which uses embedded Tomcat server

>ShowHomeController.java

#src/main/resources

This class and its super class are useful to register the pre-defined DispatcherServlet dyamically with SErvletContainer using Programtic Approach

`static`

templates

application.properties

>

src/test/java

> JRE System Library [JavaSE-16]

>

Maven Dependencies

>

Deployed Resources

✔ src

main

> java

> resources

webapp

WEB-INF

✓ pages

welcome.jsp

webapp folder's directcontent is called public area content

WEB-INF and its sub folders content is called private area content

even classes kept'src/main/java folder packages goes to WEB-INF/classes folder

>test

i.e they are also going to private area

› target

WHELP.md

mvnw

mvnw.cmd

M pom.xml

step7) Run the spring boot MVC app as standalone App using Embedded Tomcat server

i) add Tomcat embedded jasper dependency to the pom.xml file by collecting it

from mvnrepository.com

<!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-jasper -->

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

note: Some how, the embedded Tomcat server is not comming with Jsp container whose name is Jasper.. So we need to

add the jsp container(jasper) dependency manually through pom.xml as shown above

ii) Run the application as java app or spring boot app

Right click on the Project ---> run as ---> Java App or Spring boot App

Welcome to Spring boot MVC First App

localhost:4041/home

(no context

path

by

default for the

web application that is

deployed in Embedded tomcat server)

How to provide context path to the spring boot MVC web application when it is executed as stadalone app on Embedded Tomcat server?

Ans) provide the context path for the web application in application.properties as shown below

← → C

In application.properties

 # ContextPath for App in Embedded Server

server.servlet.context-path=/MVCFirstApp1

localhost:4041/MVCFirstApp1/home

Welcome to Spring boot MVC First App

Q) we can

not use web.xml file for welcome file cfg in spring boot MVC app becoz spring boot mvc avoids the xml cfgs

So how to make welcome file effect in spring boot mvc

application?

request

path

Ans) configure one of the handler method with "/"

which returns LVN pointing to home page

O

```
//ShowHomeController.java
package com.nt.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class ShowHomeController {
/*@RequestMapping("/home")
public String launchHomePage() {
//return LVN
return "welcome";
}*/
@RequestMapping("/")
public String launchHomePage() {
//return LVN
return "welcome";
}
}
```

c

localhost:4041/MVCFirstApp1/

θ

Welcome to Spring boot MVC First App

Using Embedded

Tomcat server

localhost:2525/MVCBootProj1-Showing HomePage/

Welcome to Spring boot MVC First App

Using External Tomcat server.