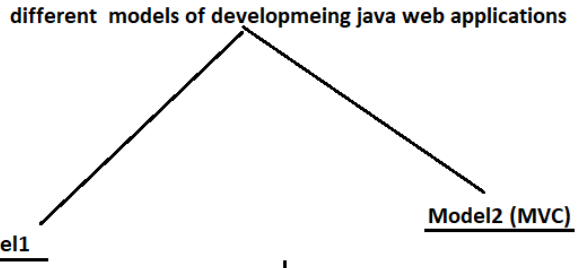
Spring MVC /spring boot MVC
==========



=> Here all logics will be placed either in servlet comp or in jsp comp

=> In this model of application development if servlet comps are used as main comps then the jsp comps will not used and vice-versa.

=> Every main web comp like servlet /jsp comp contains mix of multiple logics .. So we can say there is no clean seperate b/w logics.

=> This model1 architecture is suitable only for small scale web application.

=> Here the multiple logics of web application will be developed using multiple technologies by placing them in multiple layers.

=> Since multiple layers are there we can say there is clean seperate b/w logics

=>M for Model Layer -->contains b.logic + persistence data [Service class] [DAO class] (Basically represents data)

=>V for View Layer --> contains presentation logic (UI logics)

=>C for Controller layer ---> contains Monitoring Logics (tracing the activies)

note: Prefer this architecture to develop medium scale and large projects..

=>suitable technologies for Model layer :: java classes or spring Beans or spring with ORM code or EJB comps beans

=> suitable technologies for Controller Layer :: Servlet or Servlet Filter

=> Suitable technologies for View layer :: jsp , html , thymeleaf , angular, react js and etc...

=>Instead of developing controller comp as normal servlet comp or servlet filter comp ..it is recomanded to develop FrontController Servlet (servlet comp based FrontController DP)

=> All MVC projects of current thread are MVC with FrontController.

=>MVC is not design Pattern .. It is an architecture to devleop Layered Applications in Java .. while developing various layers and comps in that archicture we use multiple design patterns as best practies

=> Generally in the impl of MVC architecture based web application we can see the following Desing Patterns implementation

DAO , SErvice /BusinessDelegate , Factory , FrontController , View Helper , CompositeView and etc..

(required in Model layer) (required in controller layer) (Required in view layer)

==========

FrontController

==========

=>The special web comp of web application that acts as entry and exit point for all or multiple http requests is called FrontController web comp.

=> Generally one web application contains only one FrontController Web comp ..

=> we can take either Servlet comp or servlet filter comp or jsp comp as FrontContrlller comp .. but Servlet comp is recomanded.

=> A Front Controller Web comp traps either all requests or multiple requests ---> applies common system services ---> performs

navigation managment, Data Management , View Management and lastly sends the response to browser (Client)

System services:: The Minimum services that should be applied after trapping requests are called System services .
eg: security , logging, auditing and etc..

navigation managent:: taking the requests by trapping the requests and delegating/passing appropriate uri/url requests to appropriate Handler comps/classes is called Navigation managmeent . (In simple front controller right request to right handler class)

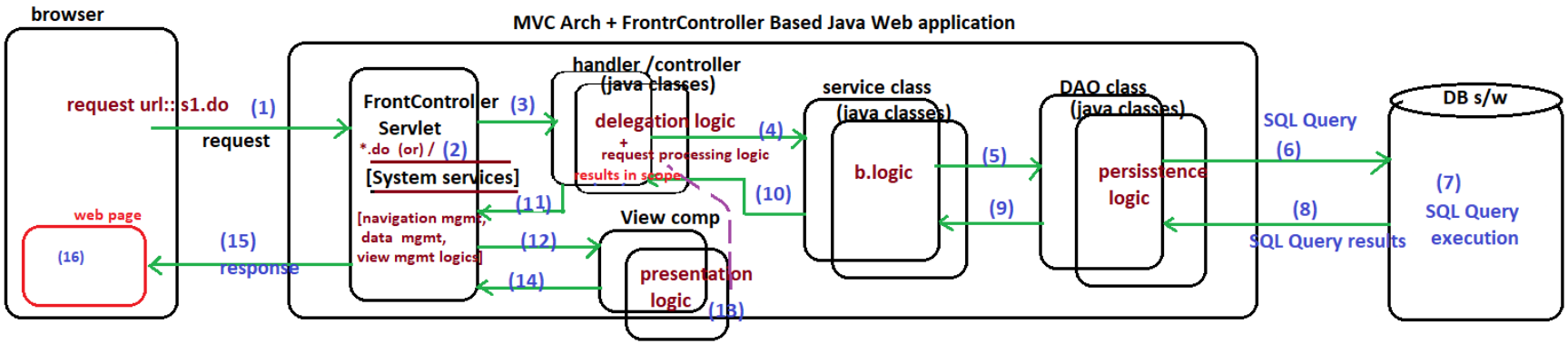
(Model Management)

Data Management :: Passing the data(inputs) that is coming along with request to appropriate handler class and passing data(results) to appropriate view comp from handler class by keeping required scope (generally request scope or session scope) is called Data Management/ Model Management

View Managent :: sending the results/outputs generated by Handler class by selecting approprivate view comp having loose coupling is called View comp

note1:: System services, Data Management, View Management, Navigation management logics will be placed in FrontController i.e entire flow of the Application request to response will be under control and monitoring of FrontController Servlet comp.

note2:: Handler class is java class and also called as controller class either can process the recieved request directly or can take the support of Service, DAO cclasses for the same (taking service,DAO classes is good pratice)



=>Since Front Controller servlet comp should trap and either all or multiple requests , so we need to configure FrontController Servlet comp having extension match url pattern (eg: *.do) or directory match url pattern(/x/y/*) or with "/" (for All requests)

(for multiple requets)

(for multiple requests)

=> with respect to each MVC archicture web application

Frontcontroller servlet :: 1
Controller/Handler classes :: 1 per module
Service classes :: 1 per module
DAO classes :: 1 per db table

=> What is the famous architeture to devleop the Java web applications
MVC + Front Controller archictture.

Limitation of MVC + Front Controller ARchicture when it is implemented with the support of serlvet ,jsp technologies directly with out using frameworks
==========

a) All the logics of the all layers must be developed manually .. This extra burden to programmer
b) We need to understand and implement entire FrontController DP manually. which is quiet complex process to perform
c) All rules of MVC architecture should remembered and implemented manually
The rules are a) every layer is designed to develop to specific logics, So place only those logics and do not place addtional logics
b) There can be multiple view comps , multiple model comps but there should only one front controller
c) All operations and flow of executions must take place through FrontController Servlet.

To overcome these problems take the support MVC frameworks to develop MVC+ FrontController based java applications..

eg:: struts -----> from apache
JSF ------> from sun MS /oracle corp (2)
Spring MVC ----> from interface21 (pivotal team)
spring boot MVC ---> from interface21 (Extension of spring MVC) (1)
Tapstrey -----> from Adobe
ADF ----------> from oracle corp

[All these are java based MVC + frontController Frameworks to develop java web applications)

Advantages of developing MVC web applications using JAva MVC frameworks
==========

a) These MVC frameworks give ready made FrontControllerServlet taking care multiple logics
In Struts ----> ActionServlet is built-in FrontController Servlet.
In JSF ----> FacesServlet is built-in FrontController Servlet.
In SpringMVC --> DispatcherSErvlet is built-in FrontController Servlet.
In SpringBootMVC --> DispatcherSErvlet is built-in FrontController Servlet.

spring Boot MVC = spring MVC++

b) Provides the abstraction on servlet ,jsp technologies and simplfies to MVC architecture based web application development

c) Flow of execution (Navigation mgmt) is fixed according to logics placed in FrontControllerServlet
So Developers need not bother about flow of executions.

d) The pre-defined FrontController contains logics by already satisfying MVC rules and guidelines
So Programmer need not to remember and implement those rules manually

e) Gives good productivtiy in application devleopment (More work in less time

and etc..

DispatcherServlet of spring MVC /spring Boot MVC
==========

=> it is spring MVC supplied readymade servlet comp

org.springframework.web.servlet.DispatcherServlet
(spring-web-<version>.jar)

=> It is given as FrontController Servlet having ready made navigation logics(control the flow), view management logics (controls the view comps) and Data mangement logics (decides the data flow)

=> In spring MVC application we need to cfg explicitly with SErvletcontainer having either directory match url pattern (like /x/y/*) or extenstion match url pattern (like *.do)
(manuall work) (To take multiple requests)

=> Spring Boot MVC application it is automatically registered with servlet Container having url pattern "/" and having load on startup enabling
(To take all requests)

=> On the deployment spring MVC /spring Boot MVC App that following operations takes place

All these are server startup or deployment activities in spring MVC or spring Boot MVC applications.

a) Becoz of "load-on-startup" that is enabled the servlet container creates DispatcherServlet class obj either on the deployment of the web application (for hot deployment) or during the server startup (for cold deployment)

hot deployment :: deploying the web application when the server is in running mode
cold deployment :: deploying the web application when the server is in stopped mode

b) The init() method DispatcherServlet(DS) executes which creates IOC container of type WebApplicationContext (ApplicationContext type)

c) This Web applicationContext performs pre-instantiation of singleton scope spring beans like controller/handler classes, service classes, DAO clases , DataSources, Handler Mappings ,

ViewResolver and etc.. perfoms the necessary Dependency Injections and keeps them in the Internal cache of IOC container.

note:: only standalone App contains main(-) method... So in standlone Apps we create IOC container in the main(-) method.. web applicaitons (spring/boot MVC Apps) does not contain main(-) method

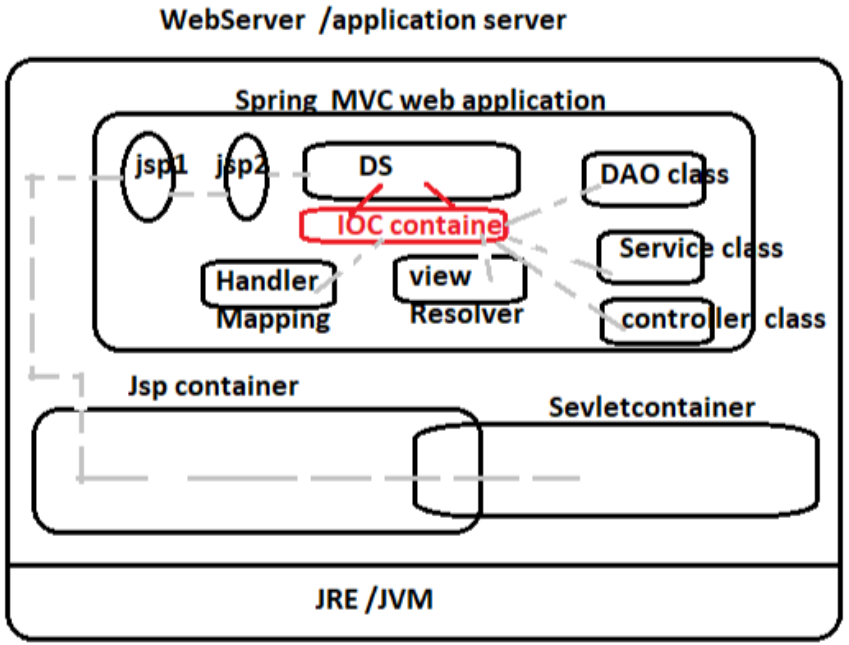
The DispatcherServlet takes are creating IOC container in those web applications.

---

note:: In spring MVC applications, DispatcherServlet , view comps(jsps) are taken care by Servlet,jsp containers .. where as controller/handler classes, service classes, DAO classes, Handler mappings , view resolvers and etc,, are java classes acting as spring beans who will be taken care by the DispatcherServlet created IOC container.

note:: all web applications must be deployed in web server /application server like tomcat , weblogic ,wildfly and etc.. to make them accessible for 24/7 .

WebServer or Application listens client requests continuosly , takes the requets , maps the requests to the web comps of web application , executes the web comps dynamically , gathers the results and sends them browser responses.



Spring MVC /spring Boot MVC flow (Diagramatic Flow)

==========

=>Spring mvc or spring boot mvc are designed around the front controller DispatcherServlet comp

=> The comps of spring mvc/spring boot mvc app are

a) DispatcherSevlet (Front Controller)

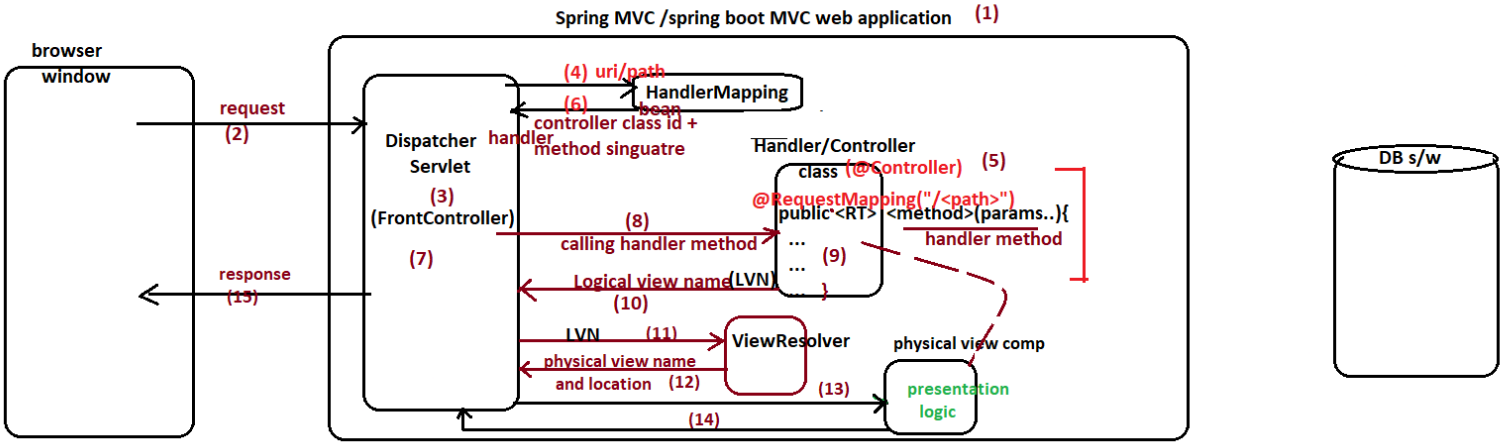
b) View comps (can be from any view technology like html ,jsp , thymeleaf, volocity, freemaker and etc..)

c) HandlerMapping comp (To map incoming request urls with Handler/controller clases)

d) Handler /Controller classes (To delegat the request to service, DAO classes)

e) ViewResolver (To resolve/identity View comps name and location)

and etc..



(1) Programmer deploy spring mvc /spring boot mvc application in web server or application server ... In the process DispatcherServlet is pre-instantiated (DS class obj is created) --> init() method DS executes performing IOC containe creattion which leads to singletom scope spring beans pre-instantations , completing dependency injections and keeping spring bean class objs in in the internal cache of IOC container

Here singleton scope spring beans are :: HandlerMaping, controller classes, View resolver , serivce classes , DAO clases and etc..

(2) enduser gives request to Spring MVC /spring boot mvc web application

(3) As FrontController , the DispatcherServlet comp traps takes the request and applies common system services.

(4) DS (DispatcherServlet) handovers the request to HandlerMapping component

(5) HandlerMapping comp uses Reflection api support to search and get @Controller class and handler method whose request path of @RequestMapping matches with incoming request uri

(6) Handler mapping component gives @Controller class spring bean id and handler method signature to DS

(7) DS takes @Controller class bean id and gets its spring bean class object from the internal cache of IOC container

(8) DS prepares the required arguments that required to call the handler method of @Controller class and calls the handler method having those arguments.

(9) Handler method of @Controller class either process the request by executing b.logic or delegates the request to service,DAO classes to process the requests. Generates results keeps in request scope

(10) Handler method of @Controller class returns LVN back to DispatcherServlet

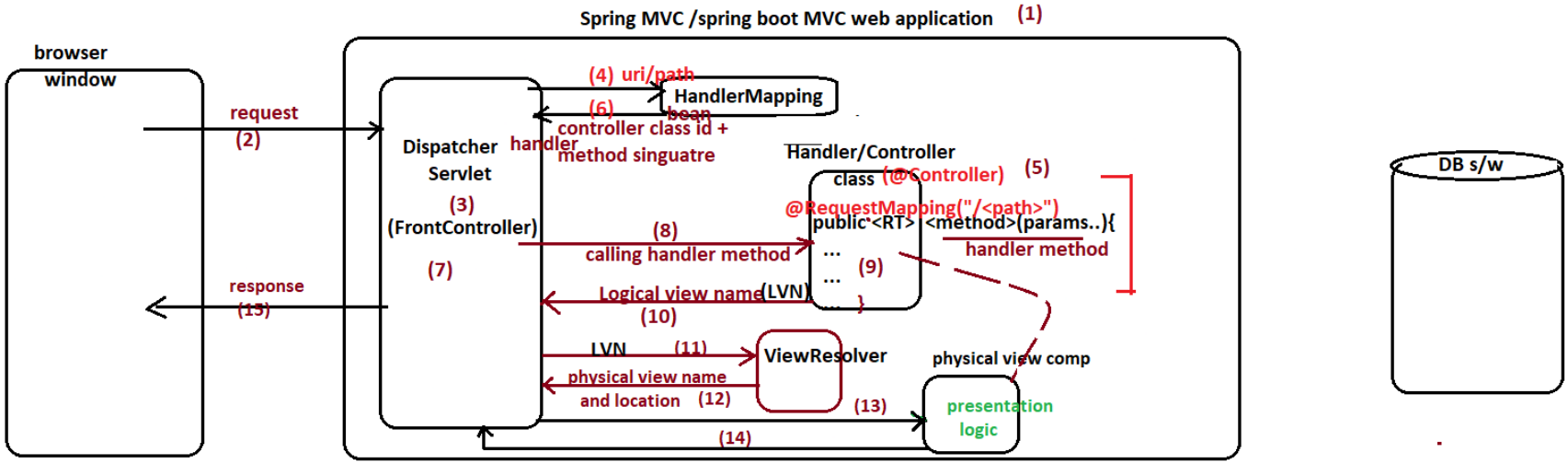
11) DS gives the recieved LVN to View Resolver comp

12) View Resolver comp resolves/identifies the name and location of physical view comp based on the recieved logic view name (LVN)

13) DS passes the control flow to the recieved physical view comp from

14) The physical view comp collectes results request scope , formats the results using presentation logics and sends the formatted results back to DS

15) DS gives the formatted results to browser window as response.



=>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(-,-) method)

Spring Boot MVC in the following operations takes place automatically

a) DispatcherServlet registration enabling load-on startup
(this uses programatic registration of Servlet comp techinue)

b) DispatcherServlet created IOC container
(This takes place from the init() of DispatchServlet comp)

c) HandlerMApping component configuration
(RequestMappingHandlerMapping class becomes spring bean through AutoConfiguration)

d) ViewResolver Configuration
(InternalResourceViewResolver class becomes spring bean through AutoConfiguration)

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 -starter-web represents spring boot statter in spring boot MVC

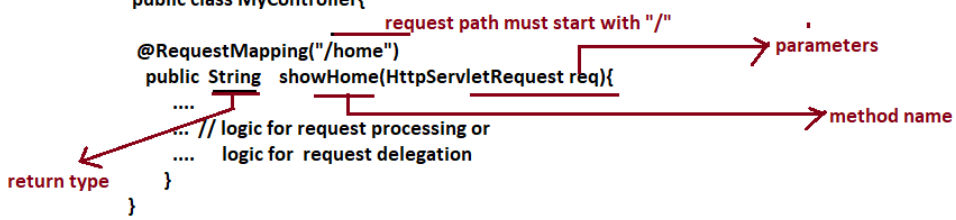
Controller class /Handler class

=> It is java class annotated with @Controller
=> we generally take this calss on 1 per module basis
=> This class can contain multiple hander methods annotation with @RequestMapping having request path (it is /<path>)

=> The sample methods can have flexible signature i.e method name , retruns type , param types ,params count is completely our choice.

//sample controller class

```
@Controller
public class MyController{
```



Allowed return types are

=>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.

Controller method return value	Description
`@ResponseBody`	The return value is converted through `HttpMessageConverter` implementations and written to the response. See `@ResponseBody`.
`HttpEntity<B>`, `ResponseEntity<B>`	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.
`HttpHeaders`	For returning a response with headers and no body.
`String` [The best in spring mvc]	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).
`View`	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).
`java.util.Map`, `org.springframework.ui.Model`	Attributes to be added to the implicit model, with the view name implicitly determined through a `RequestToViewNameTranslator`.
`@ModelAttribute`	An attribute to be added to the model, with the view name implicitly determined through a `RequestToViewNameTranslator`. Note that `@ModelAttribute` is optional. See "Any other return value" at the end of this table.
`ModelAndView` object	The view and model attributes to use and, optionally, a response status.
`void`	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 `lastModified` 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.
`DeferredResult<V>`	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 `DeferredResult`.
`Callable<V>`	Produce any of the above return values asynchronously in a Spring MVC-managed thread. See Asynchronous Requests and `Callable`.
`ListenableFuture<V>`, `java.util.concurrent.CompletionStage<V>`, `java.util.concurrent.CompletableFuture<V>`	Alternative to `DeferredResult`, as a convenience (for example, when an underlying service returns one of those).
`ResponseBodyEmitter`, `SseEmitter`	Emit a stream of objects asynchronously to be written to the response with `HttpMessageConverter` implementations. Also supported as the body of a `ResponseEntity`. See Asynchronous Requests and HTTP Streaming.
`StreamingResponseBody`	Write to the response `OutputStream` asynchronously. Also supported as the body

	of a `ResponseEntity`. See Asynchronous Requests and HTTP Streaming.
Reactive types — Reactor, RxJava, or others through `ReactiveAdapterRegistry`	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 `ResponseBodyEmitter` 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 other return value	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
==========

Controller method argument	Description
`WebRequest`, `NativeWebRequest`	Generic access to request parameters and request and session attributes, without direct use of the Servlet API.
`javax.servlet.ServletRequest`, `javax.servlet.ServletResponse`	Choose any specific request or response type — for example, `ServletRequest`, `HttpServletRequest`, or Spring's `MultipartRequest`, `MultipartHttpServletRequest`.
`javax.servlet.http.HttpSession`	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.
`javax.servlet.http.PushBuilder`	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.
`java.security.Principal`	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`.
`HttpMethod`	The HTTP method of the request.
`java.util.Locale`	The current request locale, determined by the most specific `LocaleResolver` available (in effect, the configured `LocaleResolver` or `LocaleContextResolver`).
`java.util.TimeZone` + `java.time.ZoneId`	The time zone associated with the current request, as determined by a `LocaleContextResolver`.
`java.io.InputStream`, `java.io.Reader`	For access to the raw request body as exposed by the Servlet API.
`java.io.OutputStream`, `java.io.Writer`	For access to the raw response body as exposed by the Servlet API.
`@PathVariable`	For access to URI template variables. See URI patterns.
`@MatrixVariable`	For access to name-value pairs in URI path segments. See Matrix Variables.
`@RequestParam`	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.

`@RequestHeader`	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`	For access to the HTTP request body. Body content is converted to the declared method argument type by using `HttpMessageConverter` implementations. See `@RequestBody`.
`HttpEntity<B>`	For access to request headers and body. The body is converted with an `HttpMessageConverter`. See HttpEntity.
`@RequestPart`	For access to a part in a `multipart/form-data` request, converting the part's body with an `HttpMessageConverter`. See Multipart.
`java.util.Map`, `org.springframework.ui.Model`, `org.springframework.ui.ModelMap` (Best)	For access to the model that is used in HTML controllers and exposed to templates as part of view rendering.
`RedirectAttributes`	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.
`@ModelAttribute`	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).
`Errors`, `BindingResult`	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.
`SessionStatus` + class-level `@SessionAttributes`	For marking form processing complete, which triggers cleanup of session attributes declared through a class-level `@SessionAttributes` annotation. See `@SessionAttributes` for more details.
`UriComponentsBuilder`	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.
`@SessionAttribute`	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.
`@RequestAttribute`	For access to request attributes. See `@RequestAttribute` for more details.
Any other argument	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`.

=>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.

==========
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 Dispatcher calling hanlder method on @Controller class object.

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

=> spring MVC api gives lots of pre-defined HandlerMapping classes for different situations.. Mostly there will not any need of developing custom Handler Mapping classes.

=>BeanNameUrlHandlerMapping
=>ControllerClassNameHandlerMapping
=>SimpleUrlHandlerMapping

=>useful in xml driven cfgs

=>DefaultAnnotationHandlerMapping
(Defult in annotation driven cfg upto 4.x)

Useful annotaion driven cfgs

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

=>In Spring boot MVC application , the RequestMappingHandlerMapping classes comes automatically through autoconfiguration..and will be pre-instantiation automatically using becoz the DispatcherServlet created IOC container.

ViewResolver
==========
=>This comp takes LVN (Logical view name) given by Controller throuhg DS maps/links physical view comp name and location returns 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 comp gives those details to DispatcherServlet comp.

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

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

=>InternalResourceViewResolver (default in spring MVC application)
=>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
---
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)

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

=>This physical view name and location goes to DS from ViewResolver in the form of View obj (View (I) impl class obj)
=> DS gathers physical view name and location from the View object uses rd.foward(-) or some other technique internally to execute physical view comp.

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 use the same comp then we need give special instructions container.

=>Public are web comps can be used by outsider and container/server directly with out having any permissions

=> 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)

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

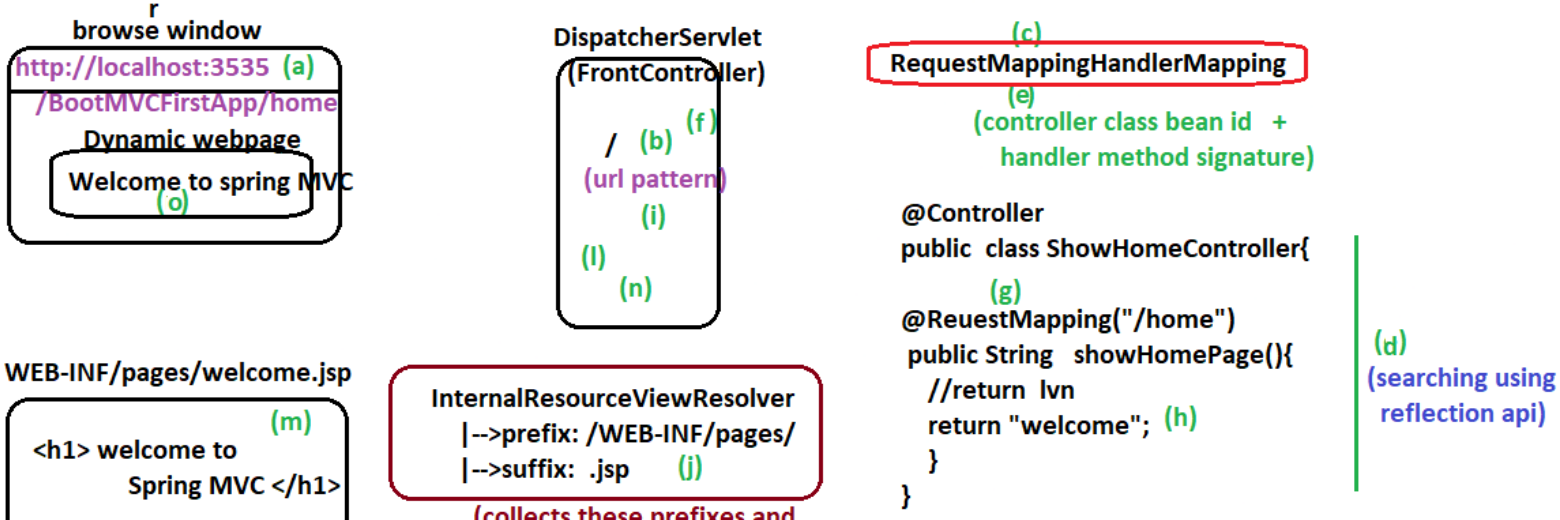
note:: Generally if jsp page is there in private then its cfg in web.xml file in mandatory having url pattern.. But the InternalResourceViewResolver of spring MVC can locate/identity the jsp pages/files of private area directly with out mapping the jsp pages/files with url pattern.

we place jsp pages in spring MVC/spring Boot MVC application
WEB-INF/pages(prefer this) fodler or WEB-INF/jsp folder or WEB-INF/view folder

=>Here "pages" , "jsp", "view" folders are user-defined folder.. we can take any name.

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)

==========



suffixes from application.properties)

View object (View(I) Impl class obj)

/WEB-INF/pages/welcome.jsp
(k)

Spring Boot MVC allows to use web application in two ways

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
   => 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..

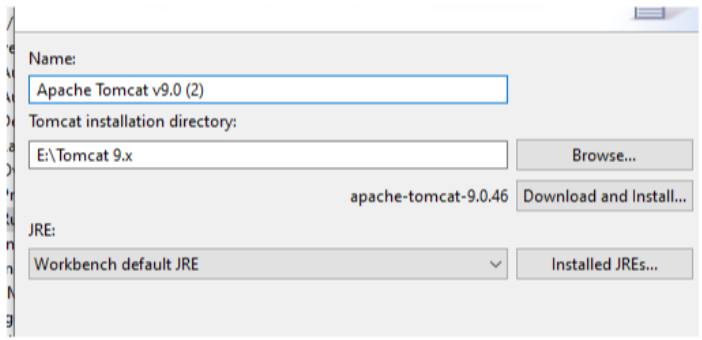
=> In cloud based application development and deployment the companies are preffering to use EmbeddServer in the dev, test , mode/env.. of the Project and the same companies are using external servers for uat , prod mode/env.. of the Project.

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

==========

step1) make sure that Tomcat 9.x is configured with Eclipse IDE.

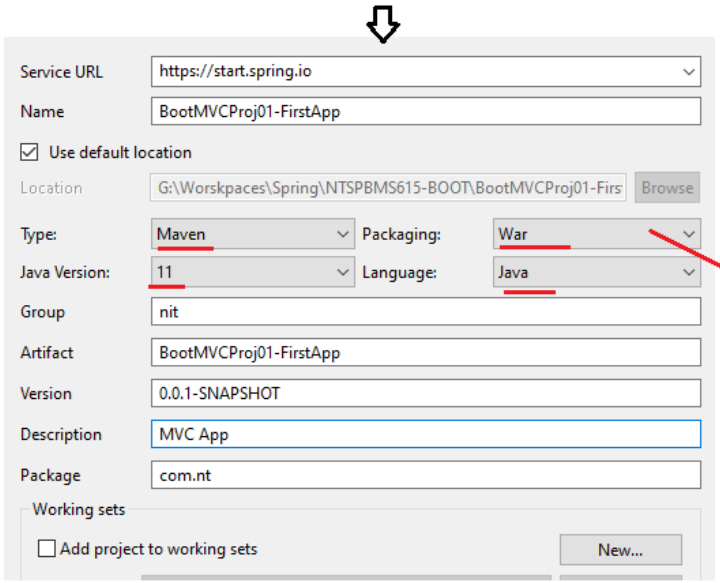
window menu ---> preferences ---> servers ---->Runtime env ----> add --->select apache Tomcat 9.x --->



----> Finish.

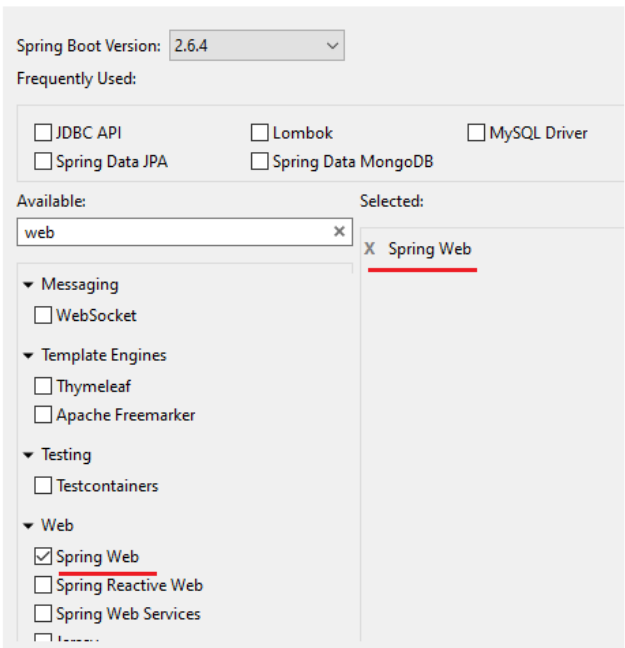
Go to servers tab ----> click on hyperlink ---> select apache Tomcat tomcat 9.x --->next -->next --> finish.

step2) Create spring boot starter Project as shown below



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



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

step3) Add the following entries in application.properties

application.properties

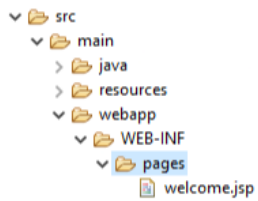
```
#Embedded server port (default is 8080)
server.port=4041

#ViewResolver inputs (prefix,suffix) for default InternalResourceViewResolver
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";
	}

}
```

step5) develop physical view comp



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 --- ---- ----

request url in the browser window :: http://localhost:2020/BootMVCProj01-FirstApp/home

Running first spring boot MVC App as standalone App using embedded Tomcat server

==========

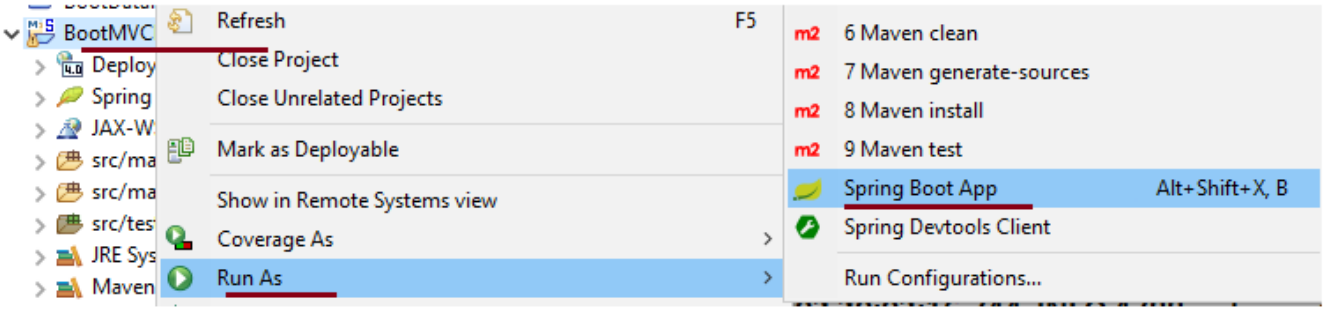
step1) add Tomcat embedded jasper dependency in pom.xml file

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

collect from mvnrepository.com

=>Some how Embeded Tomcat server does not come with built -in jsp container (jasper) .. So we need to add it explicity..

step2) Run web application as the spring boot application



step3) Test the application by giving url

http ://localhost:4041/home

=> When we run the spring boot mvc web application as the standalone app there will not be any context path for it by default .. To provide that add following entry in application.properties file

In application.properties

---

server.servlet.context-path=/FirstBootMVCApp

request url after this change :: http://localhost:4041/FirstBootMVCApp/home

=>if we are running spring boot MVC App as the normal deployable web application in Tomcat server then there is no need of placing main(-) in the @SpringBootApplication class i.e DispatcherServlet registration is happening by using the automatically generated ServletInitializer class.

war file deployment ----> ServletInitializer is very important

=>if we are running spring boot MVC App as the standalone web application .. then there is no need of ServletInitializer class becoz the SpringApplication.run(-) placed in main class will take care of creating IOC container , Embedded Tomcat server ,DispatcherServlet registation and etc..

jar file deployment ----> main(-) of main class is very important

DispatcherServlet Registration internals

==========

=> In spring Boot MVC DispatcherServlet comp is automatically registered with Servlet container having "/" url pattern and the enabled Load on startup .That internally uses Programatic approach nothig but sc.addServlet(-,-) method.

A servlet comp can be registered with ServletCotnainer in 3 ways

a) using declarative approach (web.xml entries)
[For pre-defined servlet comp where xml cfgs are allowed]

b) using annotation driven approach (@WebServlet)
[For user-defined servlet comp where xml cfgs are not allowed]

c) Programatic approach / Dynamic registration of servlet
[For pre-defined servlet comp where xml cfgs are not allowed]

note:: In spring Boot MVC apps DispatcherServlet is pre-defined servlet comp and xml cfgs are not allowed

To make Spring Boot MVC Application working as standalone we bapplication , deployable web application we need to place the following tags pom.xml as dependency representing tocmat jasper.

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <scope>provided</scope>
</dependency>
```

mandatory

Indicates the underlying Runtime env.. or server or container provides the jar file no need of collecting from repository.

Scopes in Maven dependency

- compile :: Dependency is avaiable only during the compilation of source code
- provided :: refer above
- runtime :: dependency is avaiable only during the execution of the compiled code
- system :: collect the dependency from the specified location of the system
- test :: dependency is avaiable during the test cases compilation and execution

if no scope is spcified the dependency is avaiable during compilation , execution of source code and test cases code.

DispatcherServlet Registration internals

=> In spring Boot MVC DispatcherServlet comp is automatically registered with Servlet container having "/" url pattern and the enabled Load on startup .That internally uses Programatic approach nothig but sc.addServlet(-,-) method.

A servlet comp can be registered with ServletCotnainer in 3 ways

a) using declarative approach (web.xml entries)
   [For pre-defined servlet comp where xml cfgs are allowed]

b) using annotation driven approach (@WebServlet)
   [For user-defined servlet comp where xml cfgs are not allowed]

c) Programatic approach / Dynamic registration of servlet
   [For pre-defined servlet comp where xml cfgs are not allowed]

note:: In spring Boot MVC apps DispatcherServlet is pre-defined servlet comp and xml cfgs are not allowed

Dynamic registration of DispatcherServlet comp in programatic approach with respect to spring web MVC App running in external server

spring-web-<ver>.jar
|--->META-INF
|-->services
|----->javax.servlet.ServletInitializer (file) ---> Container
org.springframework.web.SpringServletContainerInitializer

javax.servlet.ServletContainerInitializer(I)
↑ implements
org.springframework.web.SpringServletContainerInitializer (c)

org.springframework.web.WebApplicationInitializer (I)
↑ implements
org.springframework.boot.web.servlet.support.SpringBootServletInitializer (AC)
↑ extends
com.nt.ServletInitializer(c)
[IDE generated class having support logics for ondeployment configurations]

=>Deployment of spring boot mvc web application in external server like tomcat

=> Servletcontainer verifies the deployment directory structure and notices that there is no web.xml file in the web application

=>Servletcontainer creates SerlvetContext obj

=> Since the ServletContaienr is 3.0+ it looks for javax.servlet.ServletContainerInitializer file in all the jar files that are kept in WEB-INF\lib folder ..but finds in spring-web-<ver>.jar file ---> loads that file file and collects that file content org.springframework.web.SpringServletContainerInitializer class,

=> SErvletContainer loads org.springframework.web.SpringServletContainerInitializer class , creates the obj and calls onStartup(-,-) on that object having Sevletcontext obj and empty set collection obj as the arguments.

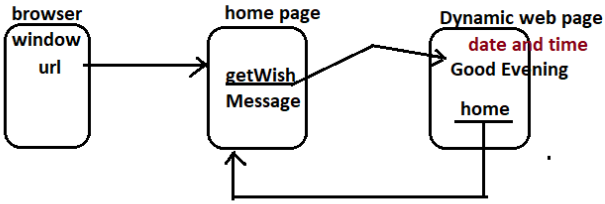
=> onStartup(-,-) of org.springframework.web.SpringServletContainerInitializer class searches forWebApplicationInitialzer(I) impl class that is present in our current web application that is com.nt ServletInitializer class , then it loads that class , creates the object for that class and calls onstartup(-) having ServletContext obj .. since not available in ServletInitializer class .. the super class (SpringBootServletIntializer class) onStartup(-) excutes ..

This onStartup(-) method contains the following logics

a) IOC container creation by taking our main class @SpringBootApplication class as configuration class

b) DispatcherServlet object creation and registering that object with ServletContainer using Servletcontext obj by calling sc.addServlet(-,-) havng url pattern "/" and enabling load on startup by calling setLoadonStartup(-) method.

Second Example App on spring boot MVC
==========



Storyboard
==========

#1 index.jsp (default welcome)

```
<jsp:forward  #2
   page="home"/>
```

DispatcherServlet
(FrontControllerServlet)

#3 #16

[/] -->url pattern

#7 #20

#10 #13

#28 #31

#4 #17

RequestMappingHandlerMapping

#6 #19

#11 #29

InternalResourceViewResolver
|--->prefix: /WEB-INF/pages/
|--->suffix: .jsp
(comes becoz of application.properties)

View object #12

/WEB-INF/pages/welcome.jsp

/WEB-INF/pages/show_result.jsp #30

welcome.jsp (WEB-INF/pages/welcome.jsp)

#14

```
get Wish message #15
( <a href="wish"> get wish message </a>
```

#32

show_result.jsp (WEB-INF/pages/show_result.jsp)

```
<h1> result page <h1>
  wish message:: ${wMsg}  <br>
  sys date :: ${sysDate}  <br>
home
<a href="home">home </a>
```

```
//HAndler or Controller class
@Controller
public class WishMessageController{
  @Autowired
  private IWishService service;
@RequestMapping("/home")
public String displayHomePage(){ #5?
   //return lvn #8
   return "welcome";
}  #9

@RequestMapping("/wish") #21
public  ModelAndView  fetchWishMessage(){ #18?
   //use service
   #22
#25 String msg= service.generateWishMessage();
  //keep data /results in MAV(ModelAndView)
  ModelAndView mav=new ModelAndView();
#26 mav.addAttribute("wMsg",msg);
  mav.addAttribute("sysDate",new Date());
  mav.setViewName("show_result");
  return mav; #27
}
}
```

=>The model attributes added to the MAV(ModelAndView) object goes to DS and become request scope data

=>The LVN added to the MAV object goes to DS and will be given ViewResolver to get View object having physical view name and location

```
Service Interface and  ServiceImpl class
----------
//serivce Interface
public  interface IWishServIce{
  public String  generateWishMessage();
}

//service Impl class
@Service("wishService")
public  class WishServiceImpl implements IWishService{
                                        #23
  public  String generateWishService(){
     ....
     ....    //logic to generate wish message
     ...
     return   msg;  #24
  }
}
```

In most of servers ..the default welcome page.. when no welcome page is cfg is index.jsp or index.html .. if both are placed then it will take index.html

---

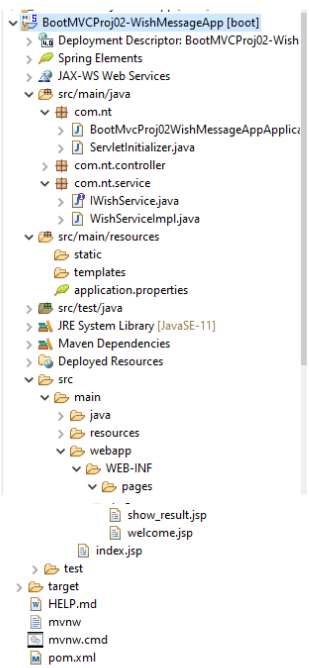
application.properties

```
#View Resolver inputs cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp

# server port cfg  (only for Embedeed server)
server.port=4041

#Context path for web application(only for Embedded Server)
server.servlet.context-path=/WishMessageApp
```

//Controller class
==========

```
package com.nt.controller;
```

```java
import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import com.nt.service.IWishService;

@Controller
public class WishMessageOperationsController {
	@Autowired
	private IWishService  service;

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

	@RequestMapping("/wish")
	public  ModelAndView   fetchWishMessage() {
		//use service
		String  msg=service.generateWishMessage();
		//keep results and other data as Model attributes in MAV object
		ModelAndView mav=new ModelAndView();
		mav.addObject("wMsg", msg);  //attr name ,  value
		mav.addObject("sysDate", new Date());  //atrt name, value
		mav.setViewName("show_result");
		//return MAV
		return mav;
	}


}
```



Data rendering and different signature of handler methods

==========

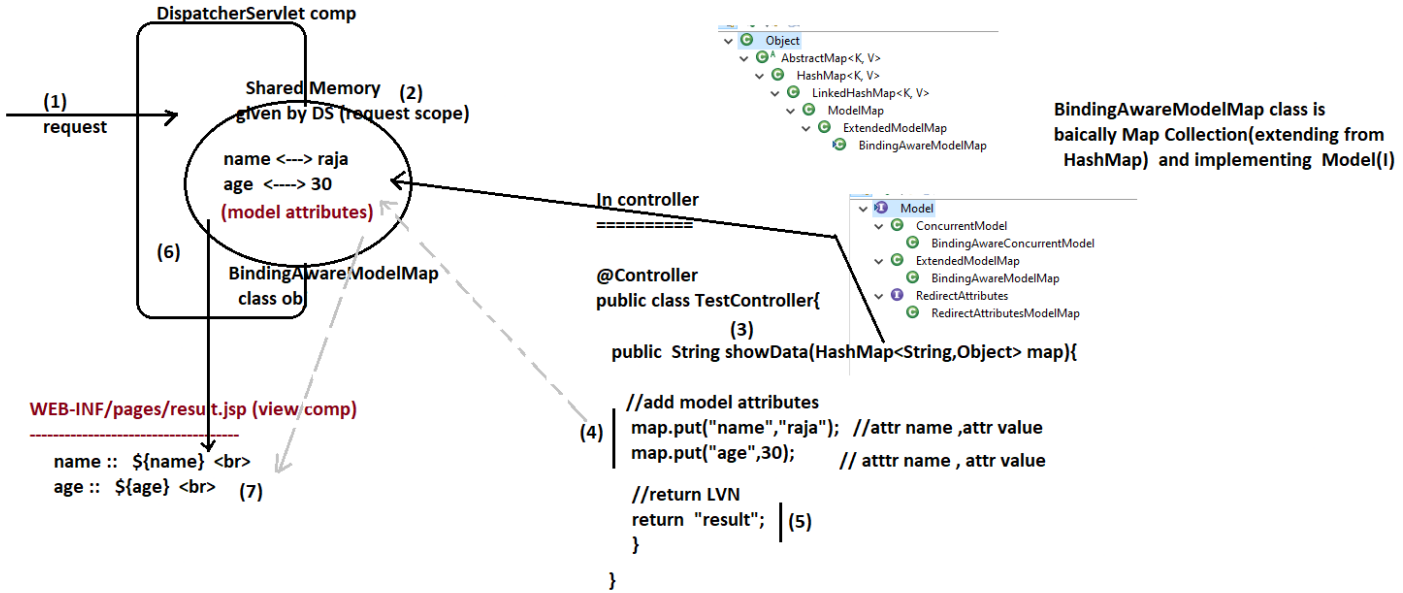
Data rendering:: The process of passing data controller class to view comp through Dispatcherservlet is called Data Rendering..

```
@RequestMapping("/wish")
    public ModelAndView fetchWishMessage() {
        //use service
        String msg=service.generateWishMessage();
        //keep results and other data as Model attributes in MAV object
        ModelAndView mav=new ModelAndView();
        mav.addObject("wMsg", msg);  //attr name ,  value
        mav.addObject("sysDate", new Date());  //atrt name, value
        mav.setViewName("show_result");
        //return MAV
        return mav;
    }
```

Working with ModelAndView as the return type of handler method is bad pratice and legacy style becoz ModelAndView is spring api specific class name , more over we need to create ModelAndView class obj manually

=> Actually to pass data from controller to view comp through DispatcherServlet it is recomanded to use the shared memory given by DisatcherServlet

=>For Every request the DispatcherServlet object creates one Shared Memory in Dispatcher Servlet itself to carry data from Controller comp to View comp.



BindingAwareModelMap class is baically Map Collection(extending from HashMap) and implementing Model(I)

examples on Data Rendering

==========

example1:: (good to use)

```
@RequestMapping("/wish")
public String fetchWishMessage(HashMap<String,Object> map) {
    System.out.println("shared Memory obj class name::"+map.getClass());
    //use service
    String msg=service.generateWishMessage();      =>super class reference variable can refer one of its sub class object
     // keep data in model attributes                => similarly, Interface reference variable can refer one its impl class obj
    map.put("wMsg", msg);
    map.put("sysDate", new Date());
    //return LVN
    return "show_result";
}
```

```
        map.put("wMsg",msg);
        map.put("sysDate",new Date());
        //return MAV
        return "show_result";
    }
```

example2 (Not good becoz Model(I) makes the method invasive method-spring specific)

```
@RequestMapping("/wish")
    public String fetchWishMessage(Model model) {
        System.out.println("shared Memory obj class name::"+model.getClass());
        //use service
        String msg=service.generateWishMessage();
        // keep data in model attributes
        model.addAttribute("wMsg", msg);
        model.addAttribute("sysDate",new Date());
        //return MAV
        return "show_result";
    }
```

example3:: (Not good becoz ModelMap(C) makes the method as invasive method-spring specific)

```
@RequestMapping("/wish")
    public String fetchWishMessage(ModelMap map) {
        System.out.println("shared Memory obj class name::"+map.getClass());
        //use service
        String msg=service.generateWishMessage();
        // keep data in model attributes
        map.addAttribute("wMsg", msg);
        map.addAttribute("sysDate",new Date());
        //return MAV
        return "show_result";
    }
```

example4: (best) (good becoz Map(I) makes the method as non-invasive method-nonspring specific)

```
@RequestMapping("/wish")
    public String fetchWishMessage(Map<String,Object> map) {
        System.out.println("shared Memory obj class name::"+map.getClass());
        //use service
        String msg=service.generateWishMessage();
        // keep data in model attributes
        map.put("wMsg", msg);
        map.put("sysDate",new Date());
        //return MAV
        return "show_result";
    }
```

example5:
=========
(Bad)

```
@RequestMapping("/wish")
    public ModelMap fetchWishMessage() {
        //use service
        String msg=service.generateWishMessage();
        // keep data in model attributes
        ModelMap map=new BindingAwareModelMap();
        map.put("wMsg", msg);
        map.put("sysDate",new Date());
        //return MAV
        return map;
    }
```

=>if no logical view name is taken taken then request path of the handler method by exclusing "/" becomes logical view name .. For example if the request path is "/wish" .. the logical view name will be "wish" .For this

It internally uses RequestToViewNameTranslator comp.

Example6::
(Bad)

```
@RequestMapping("/wish")
    public Model fetchWishMessage() {
        //use service
        String msg=service.generateWishMessage();
        // keep data in model attributes
        Model model=new BindingAwareModelMap();
        model.addAttribute("wMsg", msg);
        model.addAttribute("sysDate", new Date());
        //return MAV
        return model;
    }
```

Example7:: (Bad)

```
@RequestMapping("/wish")
    public Map<String,Object> fetchWishMessage() {
        //use service
        String msg=service.generateWishMessage();
        // keep data in model attributes
        Map<String,Object> map=new HashMap();
        map.put("wMsg", msg);
        map.put("sysDate", new Date());
        //return MAV
        return map;
    }
```

Limitations of taking Map<-,-> or ModelMap or Model(I) as return type of handler method

(a) we should create shared memory explicitly which gives burdern to programmer and memory utialization and cpu utilization
(b) The automatically created SharedMemory In DS (created for BindingAwareModelMap obj) will be wasted

(c) No control on Logical view NAme (LVN) we we need to take requeest path as the logical name

Advantages of taking Map <-,->(I) or ModelMap(c), Model(I) as the paameter types of handler method

(a) we can use the DispatcherServlet created and refered SharedMemory Object to add model attributes
(b) Full control on Logical view name becoz we take java.lang.String as the return type
(c) No need of creating SharedMemories explicitly .. So burdern on the Programmer no

Best signature for handler methods in most of the situations
==========

```
public String <method name> (Map<String,Object> map , other params ...){
    .... //logic to invoke service class methods
    ...

    ... //logic to results to model attributes
    ...

    return "<lvn>";

}
```

Q) what happens if we take handler method return as "void"?
Q) what happens if we return null from handler method irrespective of its return type?

Q) what happens if we take handler method return as "void"?

Ans) It will take request path as the logical view name by excluding the slash.

```
@RequestMapping("/wish")
public void fetchWishMessage(Map<String,Object> map) {
    //use service
    String msg=service.generateWishMessage();
    // keep data in model attributes
    map.put("wMsg", msg);
    map.put("sysDate", new Date());
}
```

It takes the request path "wish" of "/wish" as the logical view name

Q) what happens if we return null from handler method irrespective of its return type?

Ans) It also takes requeat path as the logical view name.

```
@RequestMapping("/wish")
public String fetchWishMessage(Map<String,Object> map) {
    //use service
    String msg=service.generateWishMessage();
    // keep data in model attributes
    map.put("wMsg", msg);
    map.put("sysDate", new Date());
    return null;
}
```

Q) can we send response directly to browser through DispatcherServlet from Controller comp with out involving ViewResolver and view comps ?

Ans) Make HttpServletResponse as the parameter of the handler method .. get PrintWriter from that Object and write response to browser directly.. From this handler method we can return null (if we take some return type) or nothing (if we do not take any return type)

```
@RequestMapping("/wish")
public String fetchWishMessage(HttpServletResponse res) throws Exception{
    //use service
    String msg=service.generateWishMessage();

    //get PrintWriter from response obj
    PrintWriter pw=res.getWriter();
    //write output to response object
    pw.println("<b> wish message is ::"+msg+"</b><br>");
    pw.println("<br> sys date and time ::"+new Date()+"</b>");
```

```
            return null;
        }
```

=>Sending output to browser (client) directly from controller class handler method is very useful while performing file downloading activties.
(nothing but sending the file content of server machine file system to client machine file system)

==========
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 WishMessageController{

@RequestMapping("/report")
    public String  showReport() throws Exception{
        return "show_report";
    }

    @RequestMapping("/REPORT")
    public String  showReport1() throws Exception{
        return "show_report1";
    }
}
```

http://localhost:2525/WishMessageApp/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  showReport() throws Exception{
            System.out.println("WishMessageOperationsController.showReport()");
            return "show_report";
        }
```

if any data is given with out annotation param name then that data goes to param whose name is "value"

=> 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

d) if the request path is "/" for the handler method then it becomes default handler method in controller class i.e when no matching requestpath found then this method executes automatically

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 not 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

http://localhost:2020/BootMVCProj02-WishMessageApp (with respect to external tomcat server) → executes showHomePage() method

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/WishMessageApp (with respect to embedded tomcat) → executes showHomePage() method

http://localhost:2020/BootMVCProj02-WishMessageApp (with respect to external tomcat server) → executes showHomePage() method

Q) what happens if we take handler method return as "void"?

Ans) It will take request path as the logical view name by excluding the slash.

```
@RequestMapping("/wish")
public void fetchWishMessage(Map<String,Object> map) {
    //use service
    String msg=service.generateWishMessage();
    // keep data in model attributes
    map.put("wMsg", msg);
    map.put("sysDate", new Date());
}
```

It takes the request path "wish" of "/wish" as the logical view name

Q) what happens if we return null from handler method irrespective of its return type?

Ans) It also takes requeat path as the logical view name.

```
@RequestMapping("/wish")
public String fetchWishMessage(Map<String,Object> map) {
    //use service
    String msg=service.generateWishMessage();
    // keep data in model attributes
    map.put("wMsg", msg);
    map.put("sysDate", new Date());
    return null;
}
```

Q) can we send response directly to browser through DispatcherServlet from Controller comp with out involving ViewResolver and view comps ?

Ans) Make HttpServletResponse as the parameter of the handler method .. get PrintWriter from that Object and write response to browser directly.. From this handler method we can return null (if we take some return type) or nothing (if we do not take any return type)

```
@RequestMapping("/wish")
public String fetchWishMessage(HttpServletResponse res) throws Exception{
    //use service
    String msg=service.generateWishMessage();

    //get PrintWriter from response obj
    PrintWriter pw=res.getWriter();
    //write output to response object
    pw.println("<b> wish message is ::"+msg+"</b><br>");
    pw.println("<br> sys date and time ::"+new Date()+"</b>");
    return null;
}
```

=>Sending output to browser (client) directly from controller class handler method is very useful while performing file downloading activties.
(nothing but sending the file content of server machine file system to client machine file system)

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 WishMessageController{

@RequestMapping("/report")
    public String showReport() throws Exception{
        return "show_report";
    }

    @RequestMapping("/REPORT")
    public String showReport1() throws Exception{
        return "show_report1";
    }
}
```

http://localhost:2525/WishMessageApp/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 showReport() throws Exception{
    System.out.println("WishMessageOperationsController.showReport()");
```

if any data is given with out annotation param name then that data goes to param whose name is "value"

```
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

d) if the request path is "/" for the handler method then it becomes default handler method in controller class i.e when no matching requestpath found then this method executes automatically

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 not 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

http://localhost:2020/BootMVCProj02-WishMessageApp (with respect to external tomcat server) → executes showHomePage() method

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/WishMessageApp (with respect to embedded tomcat) → executes showHomePage() method

http://localhost:2020/BootMVCProj02-WishMessageApp (with respect to external tomcat server) → executes showHomePage() method

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 .

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";
 }
```

f) Two handler methods of controller class can have same request path having two different request modes like GET ,POST

```
@Controller
public class WishMessageController{

 //@RequestMapping(value="/report",method=RequestMethod.GET) (or)
    @GetMapping("/report")
    public String showReport() throws Exception{
        System.out.println("WishMessageOperationsController.showReport()");
        return "show_report";
    }
```

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

}
```

GET mode request

POST mode request

To send requests to the above handler methods
==========

welcome.jsp

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

note1:: Spring boot MVC/spring MVC application are web application and they can take requests only from browwer and browser can given 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 browser window 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
@PostMapping	can use obly these two @XxxMapping annotations
@PutMapping	
@DeleteMapping	
@Patch Mapping	
and etc..	

Spring Rest (extension of Spring MVC) can use all the five @XxxMapping annotations

Spring MVC/spring Boot MVC/sprWeb for developing web applications

Spring Rest (extension of spring MVC) is for develop Restful webServices (Distributed Applicaitons)

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.WishMessageOperationsController#showReport1() to {GET [/report]}: There is already 'wishMessageOperationsController' bean method com.nt.controller.WishMessageOperationsController#showReport() mapped.

Throws the exception

```
@Controller
public class WishMEssageController{

    @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";
    }
}
```

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"

refer the above example

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

```
@Controller
public class WishMessageOperationsController {

    @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() {
        //return LVN
        return "welcome";
    }
}
```

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

```
<br><br>
 <a href="./"> Get -Home page</a>
 <br>
 <form action="./"  method="POST">
   <input type="submit"  value="POst -Home page"/>
 </form>
```

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

Problem::

```
@Controller
public class WishMessageOperationsController {

    @GetMapping("/all")
    public   String  showAllData() {
      return "show_report";
    }

}
```

```
@Controller
public class TestController {

    @GetMapping("/all")
    public String  getAllTestsData() {
        return "show_report1";
    }

}
```

Caused by: java.lang.IllegalStateException: Ambiguous mapping. Cannot map 'wishMessageOperationsController' method com.nt.controller.WishMessageOperationsController#showAllData() to {GET [/all]}: There is already 'testController' bean method com.nt.controller.TestController#getAllTestsData() mapped.

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 WishMessageOperationsController {

    @GetMapping("/all")
    public   String  showAllData() {
      return "show_report";
    }

}
```

method path (or) method request path

global path or global request path

```
@RequestMapping("/test-operations")
@Controller
public class TestController {

    @GetMapping("/all")
    public String  getAllTestsData() {
        return "show_report1";
    }
}
```

method path or method request path

http://localhost:2020/BootMVCProj02-WishMessageApp/wish-operations/all --> sends request to showAllData() method of WishMessageOperationsController class

http://localhost:2020/BootMVCProj02-WishMessageApp/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 WishMessageOperationsController {

    @GetMapping("/all")
    public  String  showAllData() {
        System.out.println("WishMessageOperationsController.showAllData()");
     return "show_report";
    }

    @GetMapping
    public  String showHomePage1() {
        System.out.println("WishMessageOperationsController.showHomePage1()");
        //return LVN
        //return "forward:wish-operations/all";
        return "forward:all";
    }
```

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

request goes to showHomePage1() method ---> from ther
it goes showAllData() becoz "forward:all"

scenario2:

```
@Controller
public class WishMessageOperationsController {

    @GetMapping
    public  String showHomePage1() {
        System.out.println("WishMessageOperationsController.showHomePage1()");
        //return LVN
        return  "forward:test-operations/all";
    }
}
```

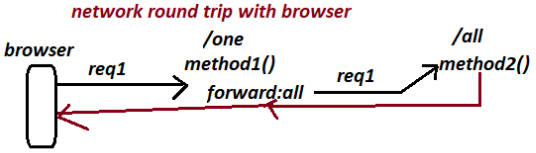
```
@Controller
@RequestMapping("/test-operations")
public class TestController {

    @GetMapping("/all")
    public String  getAllTestsData() {
        System.out.println("TestController.getAllTestsData()");
        return "show_report1";
    }

}
```

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

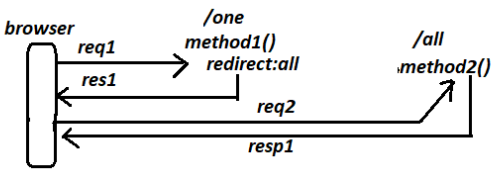
request goes to showHomePage1() method ---> from ther
it goes to getAllTestsData() method of TestController class becoz of "forward:test-operations/all" statement

l) *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



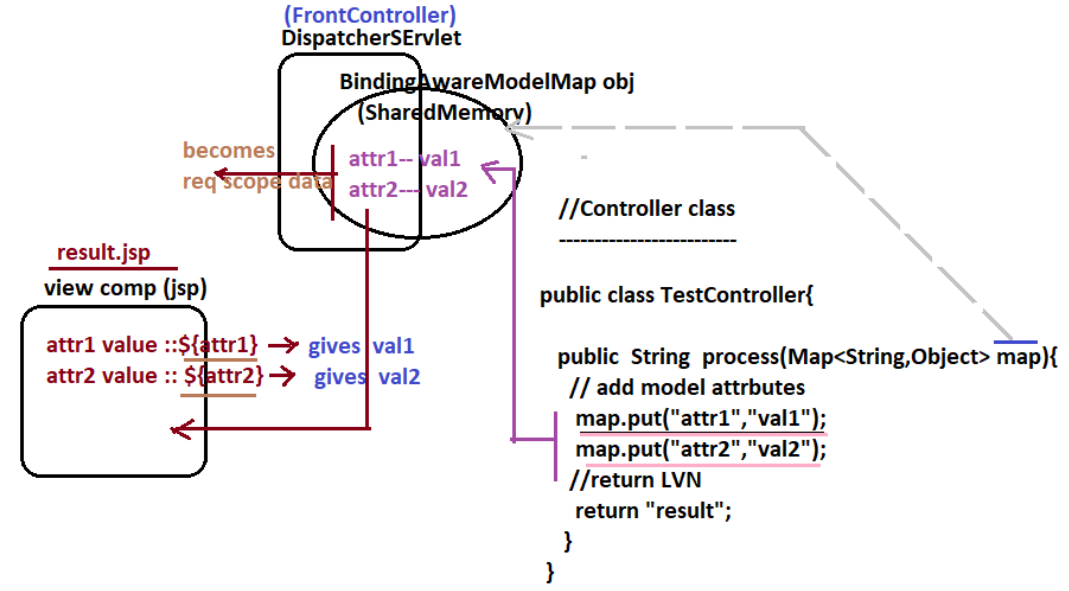
Here method2 can be either there same controller or different controller of same MVC application



Here method2 can be there either same controller or different controller of same MVC web application or different MVC web application of same server or different server belonging same machine or different machine

Data Rendering
==========
It is the process passing data from Controller class to view comp as request scope data through DispatcherServlet comp by keeping the data in shared memory created in DispatcherServlet comp

=>This Shared Memory in DispatcherSErvlet will be created as Map object for the class BindingAwareModelMap class object .. The controller keeps data in this shared memory as Model attributes and the view comp reads data from the same shared memory in the form model attributes.



Different scenarios of Data Rendering
==========
a)Passing simple values from controller class to View comps
b)Passing arrays,collection values from controller class to View comps
c)Passing Model class obj from controller class to view comps
d)Passing Collection of Model class objs from controller class to view comps

a)Passing simple values from controller class to View comps

In controller class

```
@Controller
public class DataRenderingController {

	@GetMapping("/report")
	public String showReportData(Map<String,Object> map) {
		//add simple values as the model attributes
		map.put("name", "raja");
		map.put("age",30);
		map.put("addrs", "hyd");
		//return LVN
		return "show_report";
	}

}
```

In vie comp (show_report.jsp)

```
<%@page isELIgnored="false" %>

 <b>model is is </b><br>

   <b> name:: ${name}</b> <br>
   <b> age:: ${age}</b> <br>
   <b> addrs:: ${addrs}</b> <br>
<br><br>
```

b)Passing arrays,collection values from controller class to View comps

code in controller class

```
@Controller
public class DataRenderingController {
	@GetMapping("/report")
	public  String   showReportData(Map<String,Object> map) {
		//add simple values as the model attributes  (Generally these values  are not hardcode
		 //static values - these values will  come from DB s/w through DAO,Service class)
		map.put("favColors",new String[] {"red","green","yellow","white"});
		map.put("nickNames",List.of("raja","maharaja","king","prince"));
		map.put("phoneNumbers", Set.of(999999L,888888L,7777L,666666L));
		map.put("idDetails",Map.of("aadhar",4545353L,"voterId",45345353L,"panNo","453H545"));
		//return LVN
		return "show_report";
	}
}
```

java9 features

show_report.jsp (view comp)

```
<%@page  isELIgnored="false" %>
<%@taglib  uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<b>Model data(Array,collections)  is :: </b><br>
  favourite colors(arary)::
  <c:if test="${!empty  favColors}">
    <c:forEach var="color" items="${favColors}">
        ${color},
    </c:forEach>
  </c:if>
 <br>
 Nicknames(List Collection)::
  <c:if test="${!empty  nickNames}">
    <c:forEach var="name" items="${nickNames}">
        ${name},
    </c:forEach>
  </c:if>
 <br>
  PhoneNumber(Set Collection)::
  <c:if test="${!empty  phoneNumbers}">
    <c:forEach var="phone" items="${phoneNumbers}">
        ${phone},
    </c:forEach>
  </c:if>

    <br>
  IdDetails(Map Collection -->gives only values by taking keys)::
  <c:if test="${!empty  idDetails}">
        ${idDetails.aadhar},
        ${idDetails.panNo},
        ${idDetails.voterId}<br>
  </c:if>
  <br>
  IdDetails(Map Collection (gives both keys and values))::
  <c:if test="${!empty  idDetails}">
     <c:forEach  var="id"  items="${idDetails}">
        ${id.key}<----> ${id.value} <br>
     </c:forEach>
  </c:if>
```

importing JSTL core tag library

For this we need to add JSTL taglibrary in pom.xml file

```
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>jstl</artifactId>
   <version>1.2</version>
</dependency>
```

c)Passing Model class obj from controller class to view comps

```
@Controller
public class DataRenderingController {


	@GetMapping("/report")
	public  String   showReportData(Map<String,Object> map) {
```

```
        // create Model class obn having data (Generally this object comes from DAO, service classes
         //having db table record
        Product prod=new Product(1001, "sofa",56789.0 , 1.0);
        //make  model class obj as the model attribute
        map.put("prodData",prod);
        //return LVN
        return "show_report";
    }
}
```

View comp (show_report.jsp)

```
<b>Model  data  is </b> <br>
  <c:if test="${!empty prodData}">
     product id ::  ${prodData.pid} <br>
     product name ::  ${prodData.pname} <br>
     product price ::  ${prodData.price} <br>
     product qty ::  ${prodData.qty} <br>
  </c:if>
```

Model class

```
package com.nt.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Product {
      private Integer pid;
      private String pname;
      private  Double price;
      private Double qty;

}
```

d)Passing Collection of Model class objs from controller class to view comps

In controller class

```
@GetMapping("/report")
      public  String  showReportData(Map<String,Object> map) {
            //create List of Model class objs as  the model attribute
            List<Product>  list=List.of(new Product(1001, "sofa", 897688.0, 1.0),
                                        new Product(1002, "chair",554555.0, 2.0));
            map.put("prodList",list);
            //rerturn lvn
            return "show_report";
      }
```

show_report.jsp

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


 <c:choose>
    <c:when test="${!empty prodList}">
      <table align="center" bgcolor="cyan" border="1">
        <tr>
          <th>PID </th> <th>PNAME </th> <th>PRICE </th> <th>QTY</th>
        </tr>
        <c:forEach var="prod"  items="${prodList}">
          <tr>
            <td>${prod.pid} </td>
            <td>${prod.pname} </td>
            <td>${prod.price} </td>
            <td>${prod.qty} </td>
          </tr>
        </c:forEach>
      </table>
    </c:when>
    <c:otherwise>
          <h1 style="color:red;text-align:center"> No Records found </h1>
    </c:otherwise>
  </c:choose>
```

=>we can do Handler method chaining either through forwarding request operation and send redirection opeartion

=>For forwaring request based handler method chaining use "forward: <path>"
=>For redirection based based handler method chaining use "redirct: <path>"

```
@Controller
public class WishMessageOperationsController {

    @GetMapping("/all")
    public  String  showAllData() {
        System.out.println("WishMessageOperationsController.showAllData()");
     return "show_report";
    }

    @GetMapping
    public  String showHomePage1() {
        System.out.println("WishMessageOperationsController.showHomePage1()");
        //return LVN
        return  "redirect:all";
    }
}
```

redirects request to showAllData() handler method whose request path "/all" by having network rountrip with browser window.

=> http://localhost:2020/BootMVCProj02-WishMessageApp/ url based request first goes showHomePage1() method .. from there it goes to showAllData() method.

=>we can inject the following objects created by the Servletcontainer to the @Controller class through @Autowiring process.

a) ServletContext obj (1 per web application)
b) ServletConfig object (1 per each Servlet comp/jsp comp)
c) HttpSession object (1 per browser s/w of each client machine)

In controller class
==========

```
@Controller
public class WishMessageOperationsController {

    @Autowired
    private ServletContext sc;
    @Autowired
    private  ServletConfig cg;
    @Autowired
    private  HttpSession ses;

    @GetMapping("/all")
    public  String  showAllData() {
        System.out.println("WishMessageOperationsController.showAllData()");
        System.out.println(" web application's name::"+sc.getContextPath());
        System.out.println("session id::"+ses.getId());
        System.out.println("DS logical name::"+cg.getServletName());
```

Spring's IOC container is getting the ServletContainer created Servletcontext, ServletConfig ,HttpSession object to the controller class

```
    return "show_report";
}
```

```
@GetMapping
public  String showHomePage1(ServletContext sc, ServletConfig cg, HttpSession ses) {
      System.out.println("WishMessageOperationsController.showHomePage1()");

      System.out.println(" web application's name::"+sc.getContextPath());
       System.out.println("session id::"+ses.getId());
       System.out.println("DS logical name::"+cg.getServletName());

        return "result";
}
```

invalid handler method designing..

=>This handler method throws exception becoz of two reasons

(a) ServletContext , ServletConfig are not the valid parameter types in the handler methods i.e they are not in the list of possible parameter types.

(b) ServletContext obj is one per web application and ServletConfig object is one per ServletComp .. So it is not allowed to make them specific to one request of one handler method

=>we can take handler method having HttpSession as the method parameter type becoz HttpSession object will be created using req object .. and handler method execution takes place on 1 per request basis.

```
@GetMapping
public  String showHomePage1(HttpSession ses) {
      System.out.println("WishMessageOperationsController.showHomePage1()");

       System.out.println("session id::"+ses.getId());      // valid method..
      //return LVN
      return  "redirect:all";
}
```

What is the difference b/w Frontcontroller and Controller/Handler

FrontController	controller /Handler
a) It is Servlet comp or Servlet Filter comp	a) It is java class
b) It is one per project or web application	b) It is generally taken on 1 per module basis
c) contains ServletLife cycle methods like init() ,service(-,-),destroy() directly or indirectly	c) contains user-defined methods as handler methods
d) It acts as entry and exit point for all requests	d) It contains request delegation logic for different url requests based to send the delegate reuqets to service ,DAO classes.
e) Will be instantiated and managed by ServletCotnainer	e) Will be instantiated and managed by IOC container

f) Identified with url pattern like "/" which traps and takes all requests

g) Entire spring mvc designed aroung DispatcherServlet i.e it controls the entire navigation and also takes care of navigation mgmt, view mgmt, model mgmt

f) Indenfied with global path given in @RequestMapping and the handler methods are idenfied with request path again given using @RequestMapping

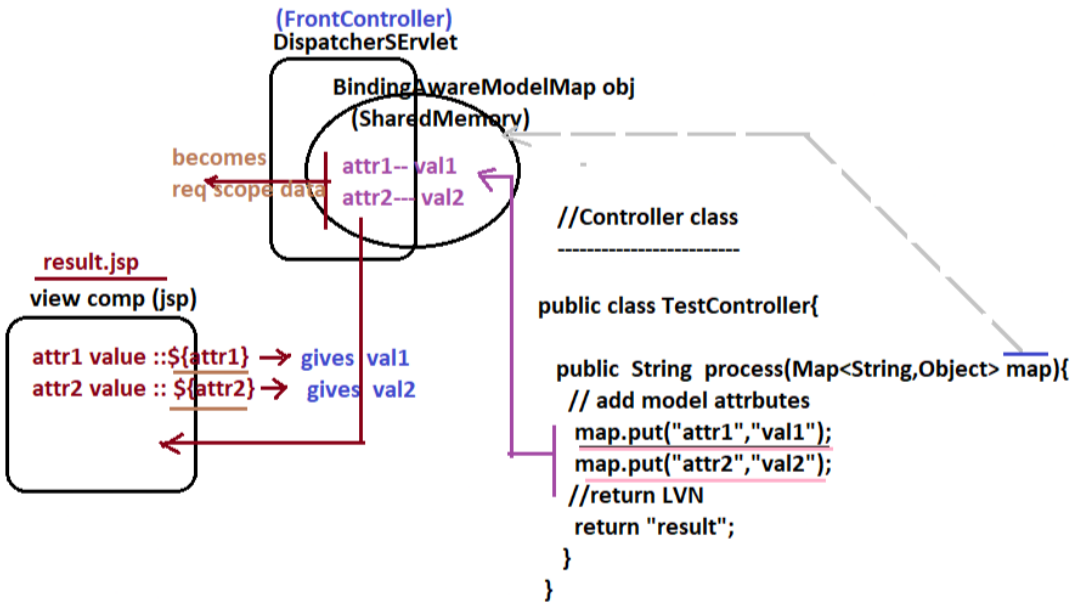
g) HandlerMapping , viewResolver , controller classes are hepler classes for FrontrController comp.

Data Rendering

==========

It is the process passing data from Controller class to view comp as request scope data through DispatcherServlet comp by keeping the data in shared memory created in DispatcherServlet comp

=>This Shared Memory in DispatcherSErvlet will be created as Map object for the class BindingAwareModelMap class object .. The controller keeps data in this shared memory as Model attributes and the view comp reads data from the same shared memory in the form model attributes.



Different scenarios of Data Rendering

==========

a)Passing simple values from controller class to View comps

b)Passing arrays,collection values from controller class to View comps

c)Passing Model class obj from controller class to view comps

d)Passing Collection of Model class objs from controller class to view comps

Data Rendering :: Passing data from controller comp to view comp through DispatcherServlet using the SharedMemory Support

Data Binding :: Passing data from View comp to Controller comp through DispatcherServlet

Two ways of Data Binding

(a) Passing Form data Controller class handler method as Model class obj using @ModelAttribute(-) annotation. (Form binding /Form Data binding)

(b) Passing hyperlink supplied addtional request params to handler method params through DispatcherServlet using @RequestParam annotation (param binding /param data binding)

(a) Passing Form data Controller class handler method as Model class obj using @ModelAttribute(-) annotation. (Form binding /Form Data binding)

To bind form data (form page form comp values) to Model class /Command class obj we need to do following operations

(a) count form page form comps and take same number of properties of required type in Model class

(b) Match Form page comp names with Model class property names

(c) add getter setter methods those properties in in Model class

(d) In Controller class Handler method take Model class type parameter having @ModelAttribute(-) annotation

Form page using html tags

student_register.jsp

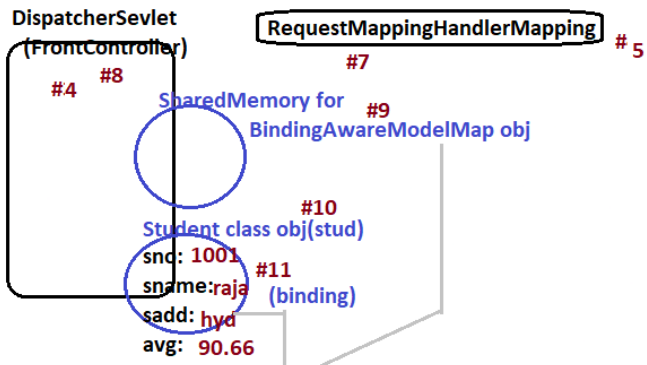
#3

```
<form action="register" method="POST">
  student id :: <input type="text" name="sno"/> <br>
  student name :: <input type="text" name="sname"/> <br>
  student address:: <input type="text" name="sadd"/> <br>
  student avg:: <input type="text" name="avg"/> <br>
  <input type="submit" value="register">
</form>
```

#1 fill form page

#2

1001 raja hyd 90.66




```
//Model class
@Data
public class Student{
  private Integer sno;
  private String sname;
  private Float avg;
  private Stirng sadd;
}
```

```
@Controller
public class StudentRegistrationController{
  @PostMapping("/register")
  public String registerStudent(Map<String,Object> map,
                                #12
                                @ModelAttribute("stud") Student st){
    ...
    ... //pass model class obj to service ,DAO classes
    ...   for registration and get results

    // keep results a model attributes #13
      ...
      ...
    //return LVN
     return "result";
  }
```

#6? (verifies for /register request path handler method)

Internal activities end to end

#1 & #2) Enduser fills up the form page and submits the form page

#3 &4) based on the action path kept in "action" attribute of <form> tag the DS traps and takes the request

#5) DS hand overs the request to Handler Mapping comp

#6 &#7) Handler mapping comp searches form Handler method whose in controller classes whose request path is "/regiser" and gets

registerStudent(-,-) method signature and relevant controller bean id

#8) HandlerMapping passes controller class bean id and handler method signature back to DS

#9) Based on Map<String ,Object> of HandlerMethod signature the SharedMemory will be reffered

#10) Based on @ModelAttribute("stud") Student st of handler method signature it creates Student class obj as the model class obj (Best pratice)

```
Student stud =new Student();
```

note:: if @ModelAttribute is taking with out attribute name like "stud" then the Model class obj will be created having the class name as the attribute name (student)

```
Student student=new Student();      (model class name as object name)
```

#11) DS reads form data as request param values and binds them to the matched Model class obj properties as shown below (it is called form data binding (or) request wrapping)

```
stud.setSno(Integer.parseInt(req.getParameter("sno")));
stud.setSname(req.getParameter("sname"));
stud.setSadd(req.getParameter("sadd");
stud.setAvg(Float.parseFloat(req.getParameter("avg")));
```

Data binding from data to Model class obj

#12) DS calls handler method having Map object pointing SharedMemory and Model class obj representing from data as the argument values

#13) Handler method uses the service,DAO classes support to process the request and to generate the response.

=> While dealing with form page submission related request processing we generally two handler methods in the controller class

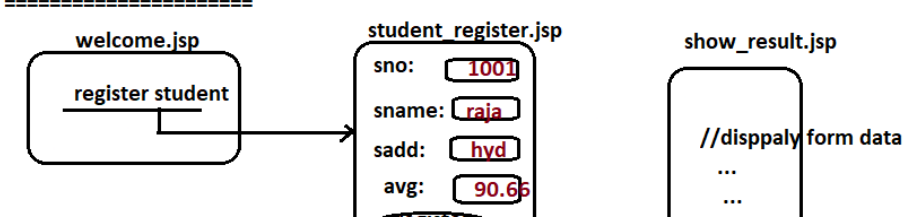
a) GET mode handler method to launch the form page (displaying form page)
b) POST mode handler method to process form page sumbisssion request (Processing form page)

note:: Both these methods can have same request path becoz the request modes are different

```
@Controller
public class StudentRegistrationController{

  @GetMapping("/register")
  public String showForm(){
    //return LVN
     return "student_register";
   }

  @PostMapping("/register")
  public String registerStudent(Map<String,Object> map,
                                @ModelAttribute("stud") Student st){
    ...
    ...
   return "show_result";
  }

}//class
```

=>Model class obj default scope is request scope

Example app Story board



#1) create spring boot starter project having the following depdencices
a) web b) lombok c) tomcat-embedded-jasper with scope provided

#2) Develop model class

```
package com.nt.model;
import lombok.Data;
@Data
public class Student {
	private Integer sno;
	private String sname;
	private String sadd;
	private Float avg;


}
```

#3) Add the following entries in application.properties

application.properties

```
#For view resolver
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp

# for embedded server
server.port=4041
```

#4) DEVELOP CONTROLLER CLASS

```
package com.nt.controller;

import java.util.Map;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import com.nt.model.Student;

@Controller
public class StudentOperationsController {

	@GetMapping("/")
	public  String showHomePage() {
		//return LVN (home page --welcome.jsp)
		return "welcome";
	}

	@GetMapping("/register")
	public   String  showStudentFormPage() {
		//return LVN (form page --> student_register.jsp
		return "student_register";
	}

	@PostMapping("/register")
	public  String  registerStudent(Map<String,Object> map,
	                                @ModelAttribute("stud") Student st) {
		 System.out.println(st);
		//return lvn
		return "show_result";

	}
```

```
}
```

#5) DEVELOP THE VIEW COMPS (JSP PAGES)

welcome.jsp

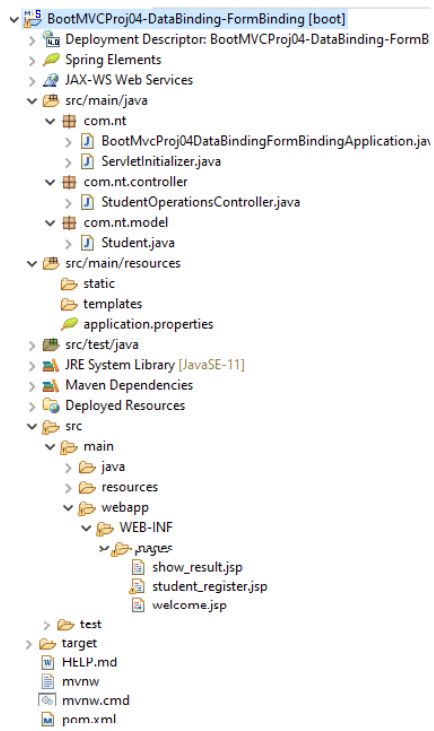
```
<%@ page isELIgnored="false" %>
  <h1 style="color:red;text-align:center">Home page</h1>
  <br><br>
   <h2 style="color:red;text-align:center"><a href="register">register student</a>  </h2>
```

student_register.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>

   <h1 style="color:red;text-aling:center">student registration page</h1>

    <form action="register" method="POST">
      <table align="center" bgcolor="cyan">
        <tr>
          <td>student number  </td>
          <td><input type="text"  name="sno">  </td>
        </tr>
        <tr>
          <td>student name  </td>
          <td><input type="text"  name="sname">  </td>
        </tr>
        <tr>
          <td>student address  </td>
          <td><input type="text"  name="sadd">  </td>
        </tr>
        <tr>
          <td>student avg  </td>
          <td><input type="text"  name="avg">  </td>
        </tr>
        <tr>
          <td colspan="2"><input type="submit" value="Register"></td>
        </tr>
      </table>
    </form>
```

show_result.jsp

```
    <%@page isELIgnored="false"  %>

     <br>
      <b> student  data is :: ${stud} </b>

      <br>
      <a href="./">home</a>
```



While making java bean class as the model class to hold form data if @ModelAttribute with logical name then the Model class obj will be created with that logical name .. if same @ModelAttrbute is taken with out logical name then the Model class obj will be created having the class name as Model class object name.

Case1:

```
@PostMapping("/register")
    public String registerStudent(Map<String,Object> map,
                                  @ModelAttribute Student st) {
        System.out.println(st);
        //return lvn
        return "show_result";
    }
```

Model class object name internally is :: student

Student student=new Student();

In show_result.jsp

```
<b> student data is :: ${student} </b>
```

Case2:

In controller class

```
@PostMapping("/register")
public String registerStudent(Map<String,Object> map,
                              @ModelAttribute("stud") Student st) {
    System.out.println(st);
    //return lvn
    return "show_result";
}
```

=>The Model class object name intenrally is "stud"

Student stud=new Student();

In show_result.jsp

```
<b> student data is :: ${stud}
```

@ModelAttribute is multi-purpose annotation

a) To make java bean class as model class to bind form data into model class obj

b) To prepare hold reference data /addtional data for the comps

Select box , List box and grouped checkboxes. (will be discussed later)

---

if i take <form> with out action attriubte or <a> tag with out href attribute then what happens?

=> Form submission / hyperlink submission goes to that url using which the form page is launched or the hyperlink is launched.

=> In spring /spring boot MVC Applications if the GET mode request path that lanuches form page is same as POST mode request path that process the form submission request then there is no need of giving "action" attribute in the <form> tag.

controller class

```
@Controller
public class StudentOperationsController {

    @GetMapping("/")
    public String showHomePage() {
        //return LVN (home page --welcome.jsp)
        return "welcome";
    }

    @GetMapping("/register")
    public String showStudentFormPage() {
        //return LVN (form page --> student_register.jsp
        return "student_register";
    }

    @PostMapping("/register")
    public String registerStudent(Map<String,Object> map,
                                  @ModelAttribute Student st) {
        System.out.println(st);
        //return lvn
        return "show_result";

    }
```

Form page

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<h1 style="color:red;text-aling:center">student registration page</h1>

    no action attribute is required
<form method="POST">
  <table align="center" bgcolor="cyan">
    <tr>
      <td>student number </td>
      <td><input type="text" name="sno"> </td>
    </tr>
    <tr>
      <td>student name </td>
      <td><input type="text" name="sname"> </td>
    </tr>
    <tr>
      <td>student address </td>
      <td><input type="text" name="sadd"> </td>
    </tr>
    <tr>
      <td>student avg </td>
```

```
    }
```

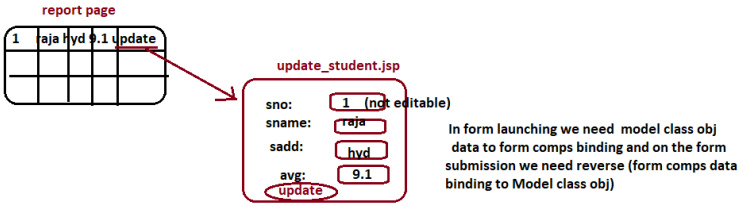
```
        <td><input type="text"  name="avg"> </td>
      </tr>
      <tr>
        <td colspan="2"><input type="submit" value="Register"></td>
      </tr>
    </table>
  </form>
```

=>For every submission of form page the DispatcherServlet creates one object of model class object binds the form data and keeps in request scope i.e the objects for model class will be created on 1 per form submission request.

we can design form page in spring MVC /spring boot MVC applications in two ways

a) using tradtional html tags (old -- not recomanded)
(it supports one way binding i.e we can bind form data into Model class obj but reverse is not possible)

b) using spring MVC supplied jsp taglibrary form tags (recomanded)
(It supports two way binding i.e we can bind form data into Model class obj and Model class obj data to form comps in the form launch)



In form launching we need model class obj data to form comps binding and on the form submission we need reverse (form comps data binding to Model class obj)

note: binding Model class object data into form comps on the lanuch form page is very much required in the edit usecase as shown above

Spring MVC supplied "form" jsp taglibrary tags are

```
<form:form>
<form:input>
<form:checkbox>
<form:checkboxes>
<form:option>
<form:options>
<form:select>
<form:radiobutto>
<form:password> ,<form:errors> ,<form:textarea>
and etc..
```

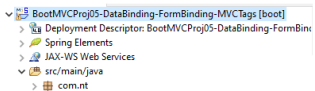
For submit button we still need to use <input type="submit" >

Tags	HTML	Spring
Input	<input type = "text">	<form:input>
Radiobutton	<input type="radiobutton">	<form:radiobutton> <form:radiobuttons> Specify list of values for radiobuttons in one shot
Checkbox	<input type="checkbox">	<form:checkbox> <form:checkboxes> Specify list of values for checkboxes in one shot
Password	<input type="password">	<form:password>
Select and option	<select name="course"> <option value="java">java</option> <option value="spring">spring</option> </select>	<form:select name="course"> <form:option value="java" label="java"/> <form:option value="spring" label="spring"/> </form:select> <form:options> Specify the list of options in one shot
Text area	<input type="textarea">	<form:textarea>
Hidden	<input type="hidden">	<form:hidden>

To this "form" jsp taglibrary of spring /spring boot MVC we need to import taglib uri using <%@taglib uri="....." prefix=".."%>

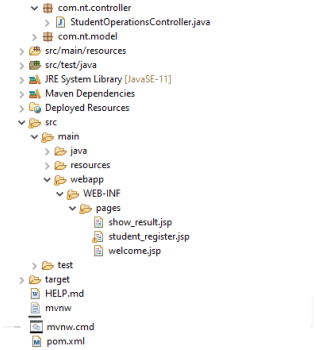
```
<%@taglib uri="http://www.springframework.org/tags/form"  prefix="form"%>
```
(fixed) (programmer choice)

example app
==========



StudentOpertaionsController.java

```
package com.nt.controller;

import java.util.Map;
```




```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import com.nt.model.Student;

@Controller
public class StudentOperationsController {

    @GetMapping("/")   (f)                    (d?)
    public  String showHomePage() {
        //return LVN (home page --welcome.jsp)
        return "welcome";   (g)
    }

(p?)
    @GetMapping("/register")    (s)
    public  String  showStudentFormPage(@ModelAttribute("stud") Student st) {
        st.setAvg(1.0f);  //generally comes  form DB in case edit usecase
(t)     //return LVN (form page --> student_register.jsp
        return "student_register";
    }

(f1?)
    @PostMapping("/register")    (j1)
    public  String  registerStudent(Map<String,Object> map,
                                    @ModelAttribute Student st) {
        System.out.println(st);
        //return lvn                    (k1)
        return "show_result";

    }

}
```

student_register.jsp *(WEB-INF/pages)* (y)

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
    <%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

    <h1 style="color:red;text-align:center">student registration page</h1>
        ( c1) no action , so last /register with POST
    <form:form  method="POST"  modelAttribute="stud">          (z)1
      <table align="center" bgcolor="cyan">
        <tr>
          <td>student number  </td>
          <td><form:input path="sno"/> </td>
        </tr>   1001                  null- empty
        <tr>
          <td>student name  </td>
          <td><form:input path="sname"/>  </td>
        </tr>   rajesh                null -empty
        <tr>
          <td>student address  </td>
          <td><form:input path="sadd"/>  </td>
        </tr>  hyd                    hyd
        <tr>
          <td>student avg  </td>
          <td><form:input path="avg"/>  </td>
        </tr>  90.0                   1.0
        <tr>
          <td colspan="2"><input type="submit" value="Register"></td>
        </tr>                                                  ( b1)
      </table>
    </form:form>
```

(z)2

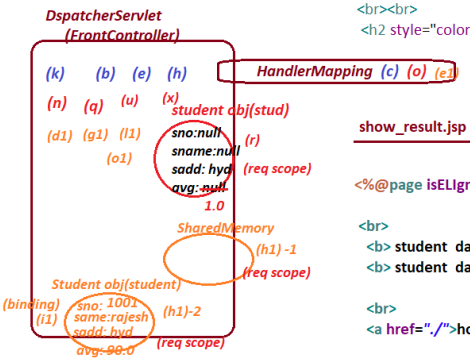
fills the form (a1)

welcome.jsp *(WEB-INF/pages)* (l)

```jsp
<%@ page isELIgnored="false" %>

  <h1 style="color:red;text-align:center">Home page</h1>

  <br><br>
   <h2 style="color:red;text-align:center"><a href="register">register student</a> </h2>
                                                                    (m)
```

show_result.jsp *(WEB-INF/pages)*
(p1)

```jsp
<%@page isELIgnored="false"  %>

<br>
  <b> student  data is :: ${stud} </b> <br>
  <b> student  data is :: ${student} </b>

  <br>
  <a href="./">home</a>
```



(i) (v) (m1)
InternalResourceViewResolver
|--->prefix:/WEB-INF/pages/
|--->suffix: .jsp

View object (j)
/WEB-INF/pages/welcome.jsp
/WEB-INF/pages/student_register.jsp (w)
/WEB-INF/pages/show_result.jsp (n1)

from browser
http://localhost:2020/BootMVCProj05-DataBinding-FormBinding-MVCTags/register
(a)

what is the differnce desing form pages using html form tags and spring mvc supplied jsp form tags?

Trandtional html form tags

(a) These tags are given by w3c (world wide web consortium)

(b) These tags support one way data binding when they are used spring mvc apps (form data to Model class obj)

(c) default request method for the form page is GET

(d) These tags not strictly typed

(e) These tags not case-sensitive

(f) can be used in html programming and also in jsp programming

spring MVC supplied jsp form tags

(a) These tags are given spring MVC framework (pivotal team)

b)These tags support two way data binding when they are used spring mvc apps (form data to Model class obj and vice-versa)

(c) default request method for the form page is POST

(d) these tags strictly typed

(e) These tags are very much case-sensitive

(f) can be used only in jsp programming

b) Data binding using @RequestParam

(Binding request param values to handler method param values)

=>sometimes hyperlinks cary addtional data along with the generated request in the form of request params which will be appended to the requst url as part of query String

```
<a href="editstudent?sno=101"> edit student </a>
<a href="editstudent?sno=102"> edit student </a>
        url having query String (request params)
```

=>These request params can be bound with Handler method params using the support of @RequestParam annotations

ex1:: request url :: http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1001&sname=rajesh

In controller class

```
@GetMapping("/data")
public String process(@RequestParam("sno") int no, @RequestParam("sname") String name) {
    System.out.println(no+"  "+name);
    //return LVN
    return "show_result";
}
```

request params are bound to handler method params

=> if request param names are matching with handler method param name then there is no need of giving request param names in @RequestParam annotation

ex2: http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=9001&sname=rakesh

Handler method in controller class

```
@GetMapping("/data")
public String process(@RequestParam int sno, @RequestParam String sname) {
    System.out.println(sno+"  "+sname);
    //return LVN
    return "show_result";
}
```

}

If any request param name is not matching with @RequestParam attribute name or method param name of handler method then we get exception

[org.springframework.web.bind.MissingServletRequestParameterException: Required request parameter 'sname' for method parameter type String is not present]

ex3:: http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=6001&name=mahesh

Handler method of controller class

```
@GetMapping("/data")
public String process(@RequestParam int sno, @RequestParam String sname) {
    System.out.println(sno+"  "+sname);
    //return LVN
    return "show_result";
}
```

macthing so binding takes place

not matching so 400 error status will be raised.

throws Exception

By default @RequestParam based method param value should be filledup becoz the default value for required annotation in @RequestParam is true.

Solution 1 for ex3 problem (take *required=false* in @RequestParam annotation)

optional to place

request url :: http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1021&name=raju

Handler method of controller class

```
@GetMapping("/data")
public String process(@RequestParam int sno, @RequestParam(required = false) String sname) {
    System.out.println(sno+"  "+sname);
    //return LVN
    return "show_result";
}
```

matching

not matching so null will be taken

gives 1021 null

solution2: (take defauilt value for the request param in @RequestParam)

http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1022&name=rajesh
http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1022
http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1022&sname=king

handler method of controller class

```
@GetMapping("/data")
public String process(@RequestParam int sno,
                      @RequestParam(defaultValue = "rrr") String sname) {
    System.out.println(sno+"  "+sname);
    //return LVN
    return "show_result";
}
```

1022 rrr
1022 rrr
1022 king

What the difference b/w required="false" and defaultValue of @RequestParam annotation?

=> required=false says though certain req param value is not given in the request url then it should not give any exception rather assigns null value method param
=>defaultValue="xxxx" assigns the given default value method param if the req param is got given or request param name is not matching.

what is the differnce desing form pages using html form tags and spring mvc supplied jsp form tags?

Trandtional html form tags

(a) These tags are given by w3c (world wide web consortium)

(b) These tags support one way data binding when they are used spring mvc apps (form data to Model class obj)

(c) default request method for the form page is GET

(d) These tags not strictly typed

(e) These tags not case-sensitive

(f) can be used in html programming and also in jsp programming

spring MVC supplied jsp form tags

(a) These tags are given spring MVC framework (pivotal team)

b)These tags support two way data binding when they are used spring mvc apps (form data to Model class obj and vice-versa)

(c) default request method for the form page is POST

(d) these tags strictly typed

(e) These tags are very much case-sensitive

(f) can be used only in jsp programming

b) Data binding using @RequestParam
(Binding request param values to handler method param values)

=>sometimes hyperlinks cary addtional data along with the generated request in the form of request params which will be appended to the requst url as part of query String

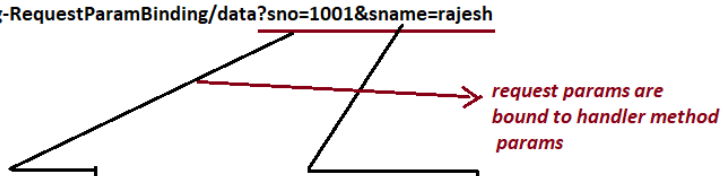
```
<a href="editstudent?sno=101"> edit student </a>
<a href="editstudent?sno=102"> edit student </a>
        url having query String (request params)
```

=>These request params can be bound with Handler method params using the support of @RequestParam annotations

ex1:: request url :: http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1001&sname=rajesh

In controller class

```
@GetMapping("/data")
public String process(@RequestParam("sno") int no, @RequestParam("sname") String name) {
    System.out.println(no+"  "+name);
    //return LVN
    return "show_result";
}
```

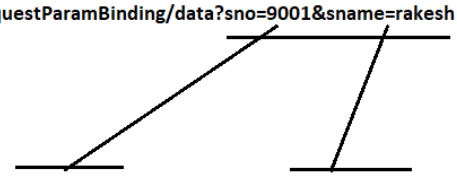


=> if request param names are matching with handler method param name then there is no need of giving request param names in @RequestParam annotation

ex2: http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=9001&sname=rakesh

Handler method in controller class

```
@GetMapping("/data")
public String process(@RequestParam int sno, @RequestParam String sname) {
    System.out.println(sno+"  "+sname);
    //return LVN
    return "show_result";
}
```



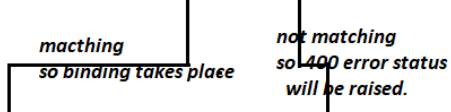
If any request param name is not matching with @RequestParam attribute name or method param name of handler method then we get exception

[org.springframework.web.bind.MissingServletRequestParameterException: Required request parameter 'sname' for method parameter type String is not present]

ex3:: http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=6001&name=mahesh

Handler method of controller class

```
@GetMapping("/data")
```



```
public String process(@RequestParam int sno, @RequestParam String sname) {
    System.out.println(sno+"  "+sname);        throws Exception
    //return LVN
    return "show_result";
}
```

By default @RequestParam based method param value should be filledup becoz the default value for required annotation in @RequestParam is true.

Solution 1 for ex3 problem (take *required=false* in @RequestParam annotation)

---

optional to place

request url :: http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1021&name=raju

Handler method of controller class

matching

not matching so null will be taken

```
@GetMapping("/data")
public String process(@RequestParam int sno, @RequestParam(required = false) String sname) {
    System.out.println(sno+"  "+sname);   gives 1021  null
    //return LVN
    return "show_result";
}
```

solution2: (take defauilt value for the request param in @RequestParam)

http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1022&name=rajesh
http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1022
http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1022&sname=king

handler method of controller class

```
@GetMapping("/data")
public String process(@RequestParam int sno,
                      @RequestParam(defaultValue = "rrr") String sname) {
    System.out.println(sno+"  "+sname);
    //return LVN                          1022  rrr
    return "show_result";                 1022  rrr
}                                         1022  king
```

What the difference b/w required="false" and defaultValue of @RequestParam annotation?

=> required=false says though certain req param value is not given in the request url then it should not give any exception rather assigns null value method param
=>defaultValue="xxxx" assigns the given default value method param if the req param is got given or request param name is not matching.

For numeric type method param if we try to pass string type value from request param then we get Number Format Exception .. No Solution for this problem exception passing correct format value.

case1:: http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=101f&sname=raja

In controller class

```
@GetMapping("/data")
public String process(@RequestParam("sno") int no,
                      @RequestParam("sname") String name) {
    System.out.println(no+"  "+name);
    //return LVN
    return "show_result";
}
```

raises :: 2022-03-18 19:37:46.115 WARN 6872 --- [nio-2020-exec-5] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.method.annotation.MethodArgumentTypeMismatchException: Failed to convert value of type 'java.lang.String' to required type 'int'; nested exception is java.lang.NumberFormatException: For input string: "102ff"]

=> for number type method param of handler method ,if u try to pass no value or empty string "" then also we get NumberFormatException .. This problem can solved using "defaultValue" or require=false with

Wrapper type params.

http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=&sname=raja

In controller class

```
@GetMapping("/data")
public String process(@RequestParam("sno") int no, @RequestParam("sname") String name) {
    System.out.println(no+"  "+name);
    //return LVN
    return "show_result";
}
```

raises :: 2022-03-18 19:38:36.517 WARN 6872 --- [io-2020-exec-11] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.method.annotation.MethodArgumentTypeMismatchException: Failed to convert value of type 'java.lang.String' to required type 'int'; nested exception is java.lang.NumberFormatException: For input string: ""]

for

Soultion1: Assign default value handler method parameter

http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=&sname=raja

Handler method of controller class

```
@GetMapping("/data")
public String process( @RequestParam(name="sno",defaultVlaue="2001") int no,
                       @RequestParam("sname") String name) {
    System.out.println(no+"  "+name);
    //return LVN
    return "show_result";
}
```

gives 2001 raja

solution2: take required=false having wrapper type method param

http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=&sname=raja

Handler method in controller class

```
@GetMapping("/data")
public String process(@RequestParam(name = "sno",required = false) Integer no,
                      @RequestParam("sname") String name) {
    System.out.println(no+"  "+name);
    //return LVN
    return "show_result";
}
```

gives null raja

=>if we pass same request param with multiple values in the query String to bind them for handler method parameter we can the parameter type as the array or List or Set type as shown below

request url :: http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=1022&sname=raja&sadd=hyd&sadd=vizag&sadd=delhi

In the handler method controller class

```
@GetMapping("/data")
public String process(@RequestParam(name = "sno",required = false) Integer no,
                      @RequestParam("sname") String name,
                      @RequestParam("sadd") String addrs[],
                      @RequestParam("sadd") List<String> saddList,
                      @RequestParam("sadd") Set<String> saddSet) {
    System.out.println(no+"  "+name+"  "+Arrays.toString(addrs)+"  "+saddList+"  "+saddSet);
    //return LVN
    return "show_result";
}
```

gives 1022 raja [hyd, vizag, delhi] [hyd, vizag, delhi] [hyd, vizag, delhi]

if the handler method param type is simple string and we are trying to give multiple values through request param the simple stirng param holds the multiple values as the comma seperated list of values.

request url ::
http://localhost:2020/BootMVCProj06-DataBinding-RequestParamBinding/data?sno=101&sname=raja&sadd=hyd&sadd=vizag

handler method of controller class

```
@GetMapping("/data")
public  String   process(@RequestParam(name = "sno",required = false) Integer no,
                         @RequestParam("sname") String name,
                         @RequestParam("sadd") String addrs) {
    System.out.println(no+"   "+name+"  "+addrs);
    //return  LVN
    return "show_result";
}
```

101 raja hyd,vizag

note:: if form page is having group of checkboxes or List box which allows to select multiple items at a time then we need to take array or List of Set type properties in Model class that will be used in form data binding.

Usecase of @Requestparam in real applications

usecase1::

webpage

```
generateExcel report
    href= report?type=excel
generate pdf report
    href=report?type=pdf
```

```
@Controller
public class ReportGenerationController{
   @Autowired
   private ReportDataService service;

public   String   generateReport(Map<String,Object> map,
                                 @RequestParam(name="type") String type){
    List<Product> list=service.getProductsInfo();
     map.put("reportData",list);
      if(type.equalsIgnoreCase("excel")){
          return "excel_report";
       }
      else if(type.equalsIgnoreCase("pdf")){
          return "pdf_report";
       }
     }
}
```

ReportPage (jsp page)

PID	PNAME	PRICE	QTY	operations	
101	table	90.44	10	edit	delete
102	chair	45.77	19	edit	delete

edit?no=101 delete?no=101

edit?no=102 delete?no=102

```
@Controller
public class CURDOperationsContrIller{

  @GetMapping("/edit")
  public String  editProduct(Map<String,Object> map,
                             @RequestParam("no") int no){
     ...
     ... //invoke service logic to modify the record
     ..
  }

  @GetMapping("/delete")
  public String   deleteProduct(Map<String,Object> map,
                                @RequestParam("no") int no){
     ...
     .. //invoke service logic to delete the record
     ..
  }

}//class
```

Boot

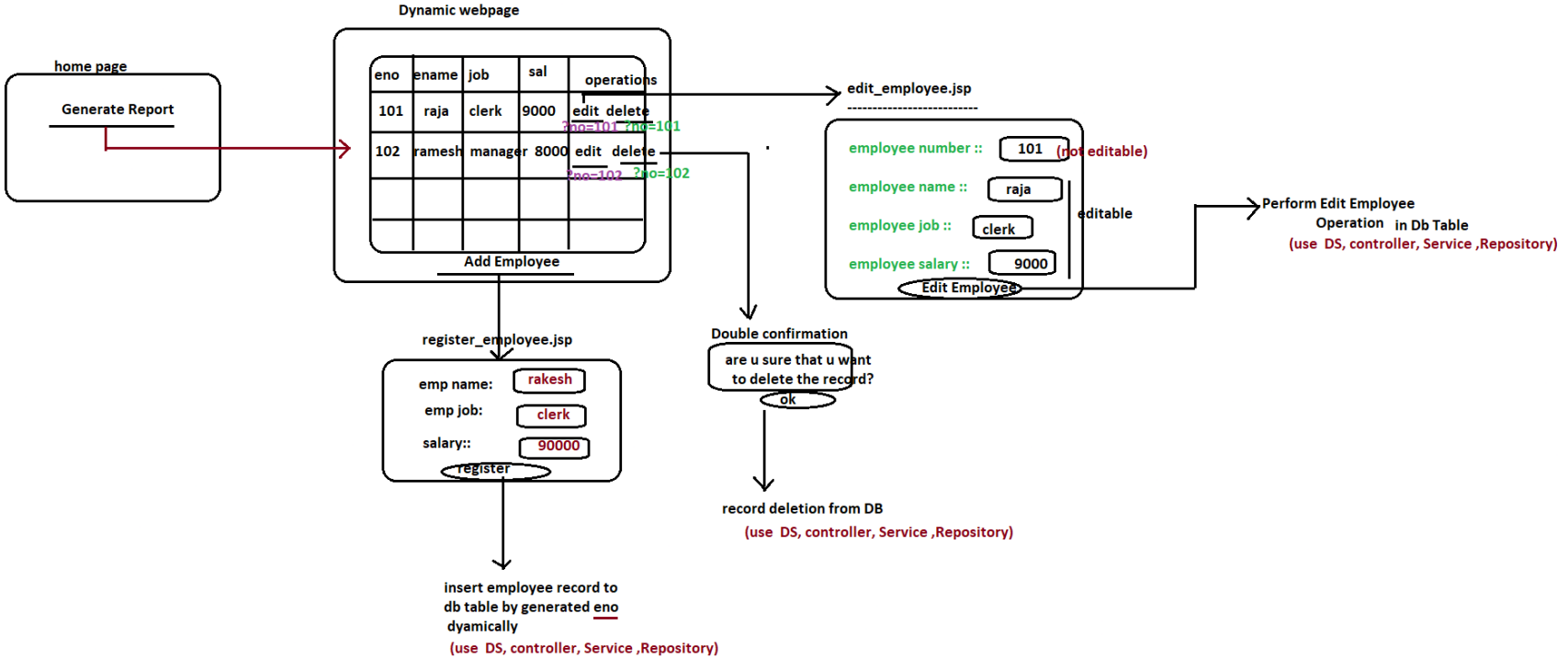
Spring MVC based Layered Application

==========

jsp --------> DispatcherServlet ---------> controller ----------> service class ----------> Repository----> Db s/w

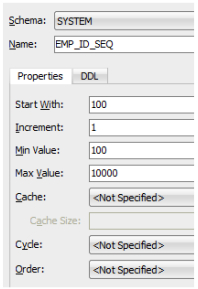
(view) (FrontController) (controller) (Mode layer) (Model layer)

(presenntation logic) navigation logic (delegation logic +exception handling) (b.logic) (persinstence logic)



step1) make sure that "emp" db table in oracle Db s/w

step2) make sure that "emp_id_seq" sequence is created in oracle Db s/w having ability to generate emp no dynamically

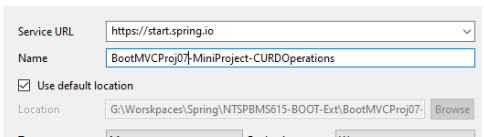


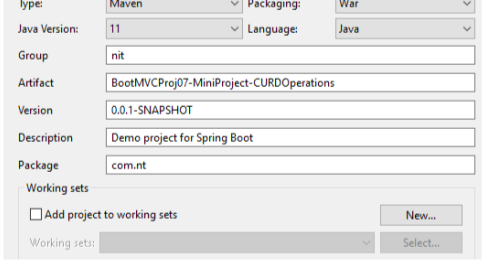
=> CREATE SEQUENCE "SYSTEM"."EMP_ID_SEQ" MINVALUE 100 MAXVALUE 10000 INCREMENT BY 1 START WITH 100 CACHE 20 NOORDER NOCYCLE ;

step3) create spring boot starter Project having the following dependencies

web , data jpa, lombok api , tomcat-embeeded-jasper , ojdbc8 ,jstl

File menu --->new ---> Project ---> spring startter





-->next ---> select the following dependenices
web, lombok, spring data jpa, oracle driver

add extra dependencies in pom.xml by collecting them from mvnrepoistory.com

```
<!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-
jasper -->
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
```

step4) create the following packages in the Project

step5) add the following entries in application.properties

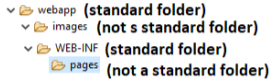
=>For data source cfg
=>For jpa-hibernate properties
=>For view resovlers
=>For Embedded Ports
=>For Context path of web application
while runing in Embedded Tomcat server

application.properties

```
#View Resolver cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp

#Embedded server port number
server.port=4041
#Context path
server.servlet.context-path=/Employee-CURDOperations

#DataSource cfg
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager

#Hibernate -JPA properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect

step6) creating the following folders in src/main/webapp folder



step7) Write the neccessary code to display the home page

i) place report icon as png or jpeg image in images folder

ii) Develop controller class having handler method with request path "/"

```
@Controller
public class EmployeeOperationsController {     (partial code)

    @GetMapping("/")
  public        String showHome() {
            return "home";
        }

    }
```

iii) place home.jsp in WEB-INF/pages folder

home.jsp (WEB-INF/pages)

```
<%@ page isELIgnored="false" %>

 <h1 style="color:red;text-align:center"><a href="report">Get Employee Data</a></h1>
   <br><br>
<h1 style="color:red;text-align:center"><a href="report"><img src="images/report1.png"
          alt="image not found"/></a></h1>
```

step8) Develop Model /Entity class

Employee.java

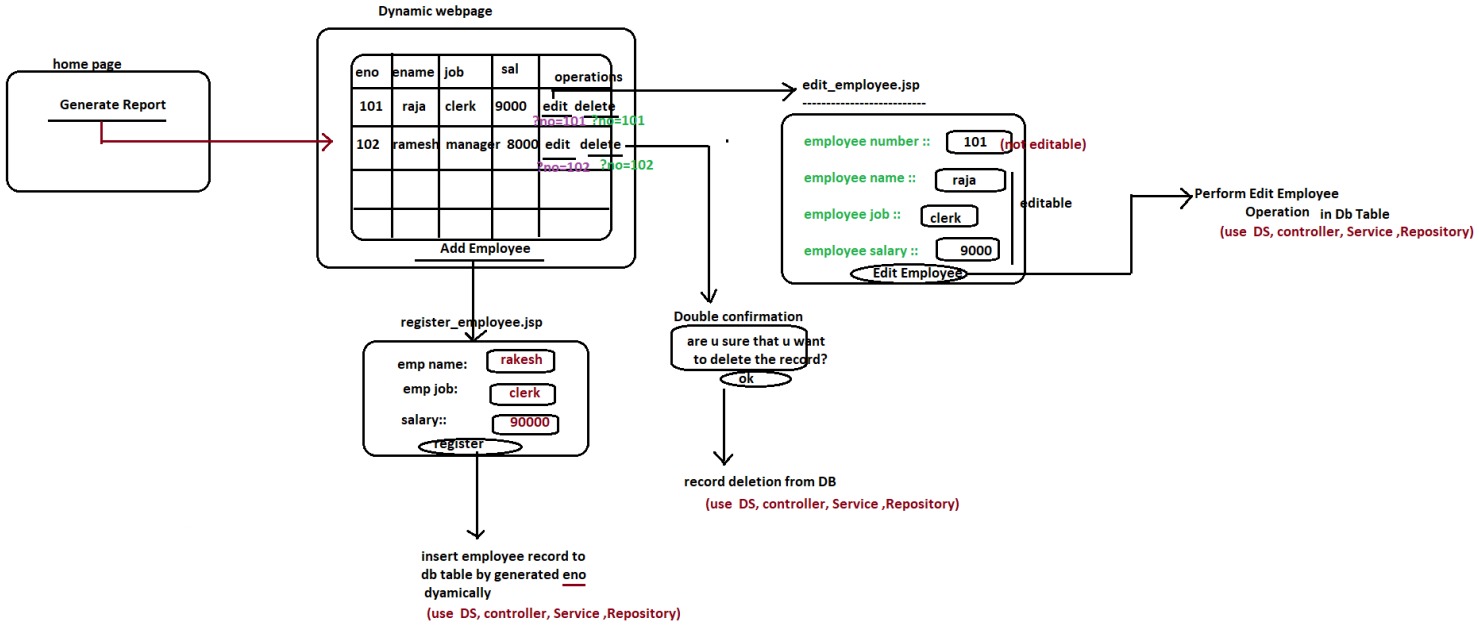
```
package com.nt.model;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import com.sun.source.doctree.SerialDataTree;

import lombok.Data;

@Entity
@Table(name="emp")
@Data
public class Employee  implements Serializable {
    @Id
    @SequenceGenerator(name = "gen1",sequenceName = "emp_id_seq")
    @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
    private Integer empno;
    @Column(length = 20)
    private  String ename;
    @Column(length = 20)
    private  String  job;
    private   Float sal;
}
```

Spring Boot MVC based Layered Application

jsp --------> DispatcherServlet ---------> controller ----------> service class ----------> Repository----> Db s/w

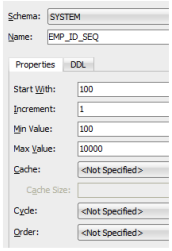
(view) (FrontController) (controller) (Mode layer) (Model layer)

(presenntation logic) navigation logic (delegation logic +exception handling) (b.logic) (persinstence logic)



step1) make sure that "emp" db table in oracle Db s/w

step2) make sure that "emp_id_seq" sequence is created in oracle Db s/w having ability to generate emp no dynamically

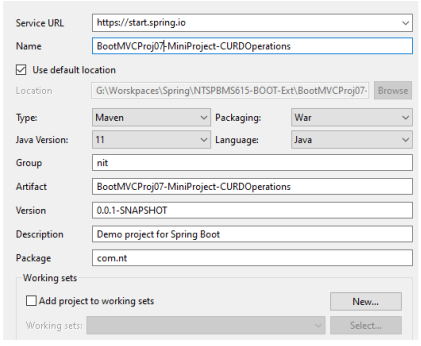


=> CREATE SEQUENCE "SYSTEM"."EMP_ID_SEQ" MINVALUE
100 MAXVALUE 10000 INCREMENT BY 1 START WITH 100 CACHE 20 NOORDER NOCYCLE ;

step3) create spring boot starter Project having the following dependencies

web , data jpa, lombok api , tomcat-embeeded-jasper , ojdbc8 ,jstl

File menu --->new ---> Project ---> spring startter



--->next ---> select the following dependenices
web, lombok, spring data jpa, oracle driver

add extra dependencies in pom.xml by collecting them from mvnrepoistory.com

```
        <!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-
    jasper -->
    <dependency>
      <groupId>org.apache.tomcat.embed</groupId>
      <artifactId>tomcat-embed-jasper</artifactId>
      <scope>provided</scope>
    </dependency>

            <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
    </dependency>
```

step4) create the following packages in the Project

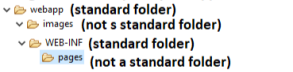
step5) add the following entries in application.properties

=>For data source cfg
=>For jpa-hibernate properties
=>For view resovlers
=>For Embedded Ports
=>For Context path of web application
while runing in Embedded Tomcat server

application.properties

```
#View Resolver cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp

#Embedded server port number
server.port=4041
#Context path
server.servlet.context-path=/Employee-CURDOperations

#DataSource cfg
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager

#Hibernate -JPA properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
```

step6) creating the following folders in src/main/webapp folder



step7) Write the neccessary code to display the home page

i) place report icon as png or jpeg image in images folder
ii) Develop controller class having handler method with request path "/"

```
@Controller
public class EmployeeOperationsController {      (partial code)

    @GetMapping("/")
  public        String showHome() {
            return "home";
      }

}
```

iii) place home.jsp in WEB-INF/pages folder

home.jsp (WEB-INF/pages)

```
<%@ page isELIgnored="false" %>

  <h1 style="color:red;text-align:center"><a href="report">Get Employee Data</a></h1>
    <br><br>
<h1 style="color:red;text-align:center"><a href="report"><img src="images/report1.png"
          alt="image not found"/></a></h1>
```

step8) Develop Model /Entity class

Employee.java

```
package com.nt.model;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import com.sun.source.doctree.SerialDataTree;

import lombok.Data;

@Entity
@Table(name="emp")
@Data
public class Employee  implements Serializable {
    @Id
    @SequenceGenerator(name = "gen1",sequenceName = "emp_id_seq")
    @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
    private Integer empno;
    @Column(length = 20)
    private  String ename;
    @Column(length = 20)
    private  String  job;
    private   Float sal;
}
```

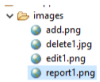
step9) Develop Repository Interface for Employee class

IEmployeeRepo.java

```
package com.nt.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import com.nt.model.Employee;

public interface IEmployeeRepo extends JpaRepository<Employee, Integer> {

}
```

step10) Perform all the activities in in service class, Controller class and in the view comps to perform Report Generation operation

i) gather add icon images in src/main/webapp/images folder related edit , delte and add operations



ii) Develop service interfce and service Impl class having logics for getting all records for report generation

Service Interface

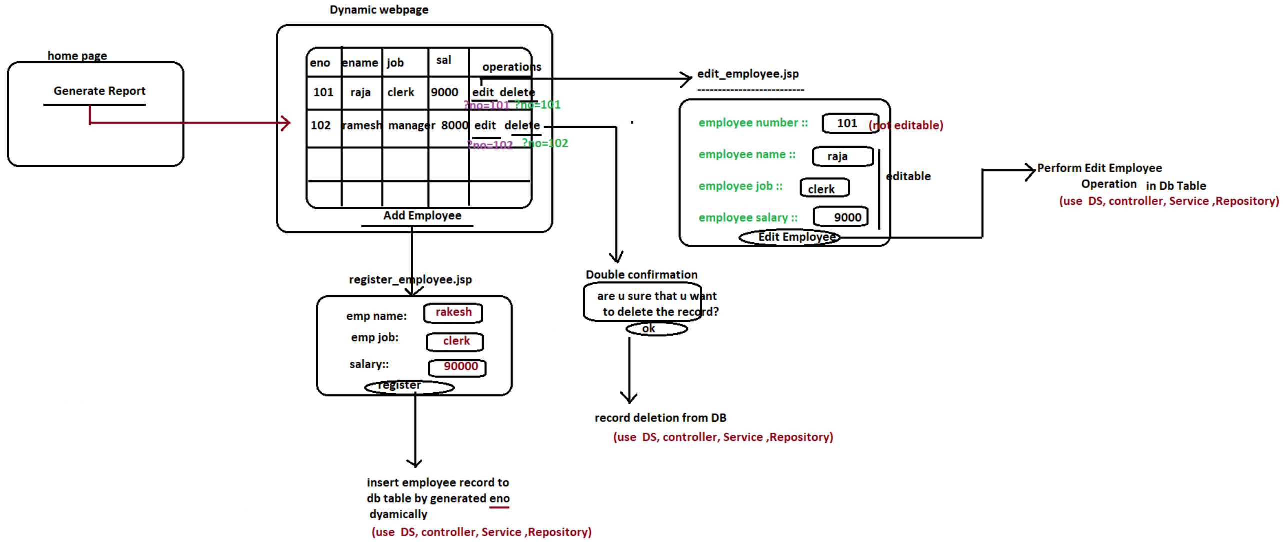
```
public interface IEmployeeMgmtService {
    public  List<Employee>  getAllEmployees();
}
```

//service impl class

```
@Service("empService")
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService {
    @Autowired
    private IEmployeeRepo empRepo;

    @Override
    public List<Employee> getAllEmployees() {
        return empRepo.findAll();
    }

}
```

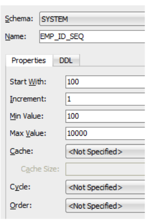
Spring Boot MVC based Layered Application
================================

jsp --------> DispatcherServlet ---------> controller -----------> service class ----------> Repository----> Db s/w
(view) (FrontController) (controller) (Mode layer) (Model layer)
(presenntation logic) navigation logic (delegation logic +exception handling) (b.logic) (persinstence logic)



step1) make sure that "emp" db table in oracle Db s/w

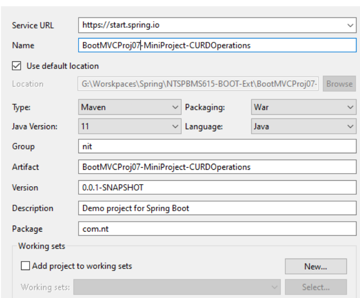
step2) make sure that "emp_id_seq" sequence is created in oracle Db s/w
having ability to generate emp no dynamically



=> CREATE SEQUENCE "SYSTEM"."EMP_ID_SEQ" MINVALUE 100 MAXVALUE 10000 INCREMENT BY 1 START WITH 100 CACHE 20 NOORDER NOCYCLE ;

step3) create spring boot starter Project having the following dependencies
web , data jpa, lombok api , tomcat-embeeded-jasper , ojdbc8 ,jstl

File menu --->new ---> Project ---> spring startter



-->next ---> select the following dependencies
web, lombok, spring data jpa, oracle driver

add extra dependencies in pom.xml by collecting them from mvnrepoistory.com

```
<!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-jasper -->
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
```

```
        <artifactId>jstl</artifactId>
    </dependency>
```

step4) create the following packages in the Project

- src/main/java
  - com.nt
  - com.nt.controller
  - com.nt.model
  - com.nt.repo
  - com.nt.service
- src/main/resources
  - static
  - templates
  - application.properties

packages to be created

step5) add the following entries in application.properties

=>For data source cfg
=>For jpa-hibernate properties
=>For view resovlers
=>For Embedded Ports
=>For Context path of web application while runing in Embedded Tomcat server

application.properties

```
#View Resolver cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp

#Embedded server port number
server.port=4041
#Context path
server.servlet.context-path=/Employee-CURDOperations

#DataSource  cfg
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager

#Hibernate -JPA properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
```

step6) creating the following folders in src/main/webapp folder

- webapp (standard folder)
  - images (not s standard folder)
  - WEB-INF (standard folder)
    - pages (not a standard folder)

step7) Write the neccessary code to display the home page

i) place report icon as png or jpeg image in images folder
ii) Develop controller class having handler method with request path "/"

```
@Controller
public class EmployeeOperationsController {      (partial code)

    @GetMapping("/")
  public      String showHome() {
          return "home";
    }

}
```

iii) place home.jsp in WEB-INF/pages folder

home.jsp (WEB-INF/pages)

```
<%@ page isELIgnored="false" %>

 <h1 style="color:red;text-align:center"><a href="report">Get Employee Data</a></h1>
   <br><br>
<h1 style="color:red;text-align:center"><a href="report"><img src="images/report1.png"
          alt="image not found"/></a></h1>
```

step8) *Develop Model /Entity class*

Employee.java

```
package com.nt.model;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import com.sun.source.doctree.SerialDataTree;

import lombok.Data;

@Entity
@Table(name="emp")
```

```
@Data
public class Employee  implements Serializable {
    @Id
    @SequenceGenerator(name = "gen1",sequenceName = "emp_id_seq")
    @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
    private Integer empno;
    @Column(length = 20)
    private  String ename;
    @Column(length = 20)
    private  String  job;
    private   Float sal;
}
```

step9) Develop Repository Interface for Employee class

IEmployeeRepo.java

```
package com.nt.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import com.nt.model.Employee;

public interface IEmployeeRepo extends JpaRepository<Employee, Integer> {

}
```

step10) Perform all the activities in in service class, Controller class and in the view comps to perform Report Generation operation

i) gather add icon images in src/main/webapp/images folder related edit , delte and add operations

ii) Develop service interfce and service Impl class having logics for getting all records for report generation

Service Interface

```
public interface IEmployeeMgmtService {
    public  List<Employee>  getAllEmployees();
}
```

```
//service impl class
@Service("empService")
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService {
    @Autowired
    private IEmployeeRepo empRepo;

    @Override
    public List<Employee> getAllEmployees() {
        return empRepo.findAll();
    }

}
```

iii) Add the following handler method in controller class

```
@GetMapping("/report")
    public  String  showEmploeeReport(Map<String,Object> map) {
        //use serivce
        List<Employee> list=service.getAllEmployees();
        // put the results in model attributes
        map.put("empsData",list);
        //return LVN
        return "employee_report";
    }//method
```

iv) Develop employee_report.jsp page aa shown below

employee_report.jsp

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

<c:choose>
  <c:when test="${!empty empsData}">
    <table border="1" bgcolor="cyan"  align="center">
     <tr>
       <th>EmpNo  </th>
       <th>EmpName </th>
       <th>Job </th>
       <th>Salary</th>
       <th> Operations </th>
     </tr>
     <c:forEach var="emp" items="${empsData}">
        <tr>
          <td>${emp.empno} </td>
          <td>${emp.ename} </td>
          <td>${emp.job} </td>
```

```
                <td>${emp.sal}</td>
                <td><a href="edit?no=${emp.empno}"><img src="images/edit1.png" width="50" height="50"/></a>    <a href="delete?no=${emp.empno}"><img src="images/delete1.jpg" width="50" height="50"></a> </td>
              </tr>
            </c:forEach>
            </table>
          </c:when>
            <c:otherwise>
              <h1 style="color:red;text-align:center"> Records not found </h1>
            </c:otherwise>
        </c:choose>
        <br><br>
        <hr>
         <h1 style="text-align:center"><a href><img src="images/add.png" width="50" height="50"> Add Employee</a> </h1>
```

step11) Perform the following operations to complete Add Employee operation

(i) write the following logics in Serivce Interface and Serivce Impl class

In service Interface

```
public String registerEmployee(Employee emp);
```

Serivce Impl class

```
@Override
    public String registerEmployee(Employee emp) {
   int idVal=empRepo.save(emp).getEmpno();
        return "Employee is saved with the id value ::"+idVal;
    }
```

ii) In controller class add two handler methods
one for showing addEmployee formpage (form launching)
another for processing of the form page (form submission)

```
@GetMapping("/add")
    public String showAddEmployeeForm(@ModelAttribute("emp") Employee emp) {
        emp.setJob("CLERK");  //intial value to display in form comp as initial value
        //return lvn
        return "employee_register";
    }
```

For fom launching

```
    @PostMapping("/add")
    public String addEmployee(Map<String,Object> map,
                              @ModelAttribute("emp") Employee emp) {
        //use service
      String result=service.registerEmployee(emp);
      List<Employee> list=service.getAllEmployees();
      //keep results in model attributes
      map.put("resultMsg", result);
      map.put("empsData",list);
      //return LVN
      return "employee_report";

    }
```

For form submission

iii) develop the employee_register.jsp page

```
<%@ page isELIgnored="false" %>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
 <h1 style="color:red;text-align:center"> Register Employee </h1>
 <form:form modelAttribute="emp">
   <table border="1" bgcolor="cyan" align="center">
     <tr>
       <td> employee name :: </td>
       <td> <form:input path="ename"/> </td>
     </tr>


     <tr>
       <td> employee desg :: </td>
       <td> <form:input path="job"/> </td>
     </tr>

     <tr>
       <td> employee salary :: </td>
       <td> <form:input path="sal"/> </td>
     </tr>

     <tr>
       <td colspan="2" align="center"><input type="submit" value="register Employee"/></td>
     </tr>
   </table>
 </form:form>
```

iv) add the following addtional code in employee_report.jsp

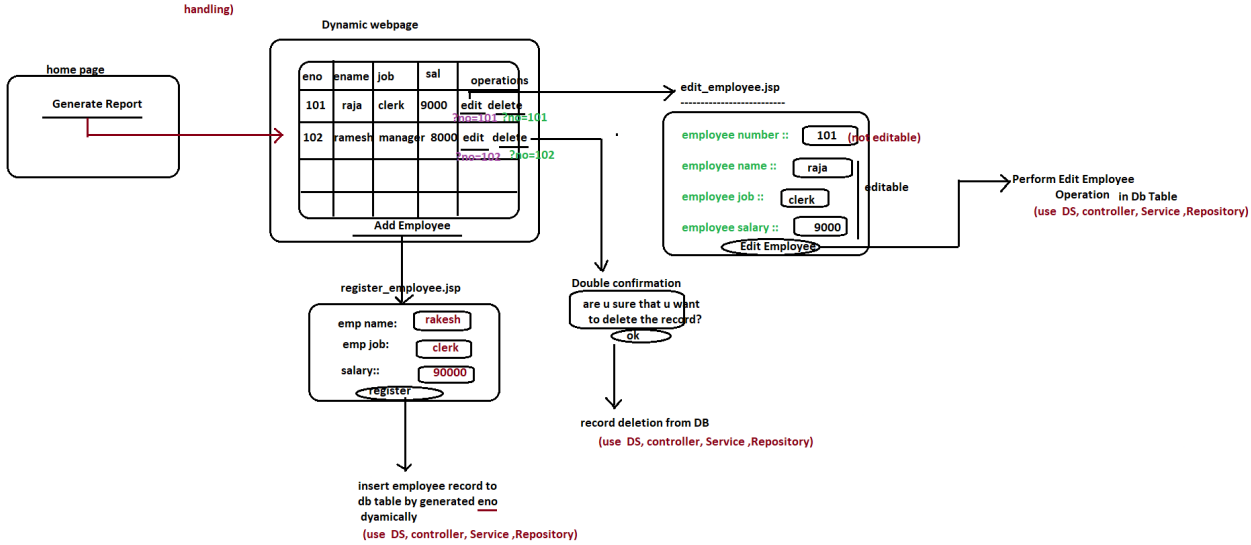
```
    <c:if test="${!empty resultMsg}">
        <h3 style="color:green;text-align:center"> ${resultMsg }</h3>
    </c:if>
```

Boot

Spring MVC based Layered Application
==========

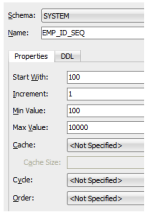
jsp --------> DispatcherServlet ---------> controller ----------> service class ----------> Repository----> Db s/w
(view) (FrontController) (controller) (Mode layer) (Model layer)

(presenntation logic) navigation logic (delegation logic +exception handling) (b.logic) (persinstence logic)



step1) make sure that "emp" db table in oracle Db s/w

step2) make sure that "emp_id_seq" sequence is created in oracle Db s/w having ability to generate emp no dynamically

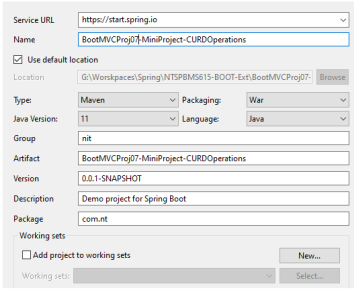


=> CREATE SEQUENCE "SYSTEM"."EMP_ID_SEQ" MINVALUE
100 MAXVALUE 10000 INCREMENT BY 1 START WITH 100 CACHE 20 NOORDER NOCYCLE ;

step3) create spring boot starter Project having the following dependencies

web , data jpa, lombok api , tomcat-embeeded-jasper , ojdbc8 ,jstl

File menu --->new ---> Project ---> spring startter

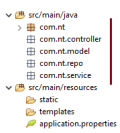


--->next ---> select the following dependenices
web, lombok, spring data jpa, oracle driver

add extra dependencies in pom.xml by collecting them from mvnrepoistory.com

```
<!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-
jasper -->
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
```

step4) create the following packages in the Project



packages to be created

step5) add the following entries in application.properties

=>For data source cfg
=>For jpa-hibernate properties
=>For view resovlers
=>For Embedded Ports
=>For Context path of web application
while runing in Embedded Tomcat server

application.properties

```
#View Resolver cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp

#Embedded server port number
server.port=4041
#Context path
server.servlet.context-path=/Employee-CURDOperations

#DataSource cfg
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
```

```
#Hibernate -JPA properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
```

step6) creating the following folders in src/main/webapp folder

- webapp (standard folder)
  - images (not s standard folder)
  - WEB-INF (standard folder)
    - pages (not a standard folder)

step7) Write the neccessary code to display the home page

i) place report icon as png or jpeg image in images folder

ii) Develop controller class having handler method with request path "/"

```
@Controller
public class EmployeeOperationsController {     (partial code)

    @GetMapping("/")
  public        String showHome() {
              return "home";
        }

    }
```

iii) place home.jsp in WEB-INF/pages folder

home.jsp (WEB-INF/pages)

```
<%@ page isELIgnored="false" %>

  <h1 style="color:red;text-align:center"><a href="report">Get Employee Data</a></h1>
    <br><br>
 <h1 style="color:red;text-align:center"><a href="report"><img src="images/report1.png"
          alt="image not found"/></a></h1>
```

step8) *Develop Model /Entity class*

Employee.java

```
package com.nt.model;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import com.sun.source.doctree.SerialDataTree;

import lombok.Data;

@Entity
@Table(name="emp")
@Data
public class Employee  implements Serializable {
    @Id
    @SequenceGenerator(name = "gen1",sequenceName = "emp_id_seq")
    @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
    private Integer empno;
    @Column(length = 20)
    private  String ename;
    @Column(length = 20)
    private  String  job;
    private   Float sal;
}
```

step9) Develop Repository Interface for Employee class

IEmployeeRepo.java

```
package com.nt.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import com.nt.model.Employee;

public interface IEmployeeRepo extends JpaRepository<Employee, Integer> {

}
```

step10) Perform all the activities in in service class, Controller class and in the view comps to perform Report Generation operation

i) gather add icon images in src/main/webapp/images folder related edit , delte and add operations

- images
  - add.png
  - delete1.jpg
  - edit1.png
  - report1.png

ii) Develop service interfce and service Impl class having logics for getting all records for report generation

Service Interface

```
public interface IEmployeeMgmtService {
    public  List<Employee>  getAllEmployees();
    }
```

```
//service impl class
@Service("empService")
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService {
    @Autowired
    private IEmployeeRepo empRepo;

    @Override
    public List<Employee> getAllEmployees() {
        return empRepo.findAll();
    }

}
```

iii) Add the following handler method in controller class

```
@GetMapping("/report")
    public  String  showEmploeeReport(Map<String,Object> map) {
        //use serivce
        List<Employee> list=service.getAllEmployees();
        // put the results in model attributes
        map.put("empsData",list);
        //return LVN
        return "employee_report";
    }//method
```

iv) Develop employee_report.jsp page aa shown below

employee_report.jsp

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

<c:choose>
  <c:when test="${!empty empsData}">
   <table border="1" bgcolor="cyan" align="center">
    <tr>
      <th>EmpNo </th>
      <th>EmpName </th>
      <th>Job </th>
      <th>Salary</th>
      <th> Operations </th>
    </tr>
    <c:forEach var="emp" items="${empsData}">
       <tr>
         <td>${emp.empno} </td>
         <td>${emp.ename} </td>
         <td>${emp.job} </td>
        <td>${emp.sal} </td>
        <td><a href="edit?no=${emp.empno}"><img src="images/edit1.png" width="50" height="50"/></a>    <a href="delete?no=${emp.empno}"><img src="images/delete1.jpg" width="50" height="50"></a> </td>
       </tr>
    </c:forEach>
    </table>
  </c:when>
    <c:otherwise>
       <h1 style="color:red;text-align:center"> Records not found </h1>
    </c:otherwise>
</c:choose>
 <br><br>
 <hr>
  <h1 style="text-align:center"><a href><img src="images/add.png" width="50" height="50"> Add Employee</a> </h1>
```

step11) Perform the following operations to complete Add Employee operation

(i) write the following logics in Serivce Interface and Serivce Impl class

In service Interface

```
public String registerEmployee(Employee emp);
```

Serivce Impl class

```
@Override
    public String registerEmployee(Employee emp) {
   int idVal=empRepo.save(emp).getEmpno();
         return "Employee is saved with the id value ::"+idVal;
    }
```

ii) In controller class add two handler methods
one for showing addEmployee formpage (form launching)
another for processing of the form page (form submission)

```
@GetMapping("/add")
public String showAddEmployeeForm(@ModelAttribute("emp") Employee emp) {
        emp.setJob("CLERK");  //intial value to display in form comp as initial value
        //return lvn
        return "employee_register";
}
```

For fom launching

```
@PostMapping("/add")
public String addEmployee(Map<String,Object> map,
                          @ModelAttribute("emp") Employee emp) {
     //use service
  String result=service.registerEmployee(emp);
  List<Employee> list=service.getAllEmployees();
  //keep results in model attributes
  map.put("resultMsg", result);
  map.put("empsData",list);
  //return LVN
  return "employee_report";

}
```

For form submission

iii) develop the employee_register.jsp page

```
<%@ page isELIgnored="false" %>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
 <h1 style="color:red;text-align:center"> Register Employee </h1>
 <form:form modelAttribute="emp">
   <table border="1" bgcolor="cyan" align="center">
     <tr>
       <td> employee name :: </td>
       <td> <form:input path="ename"/> </td>
     </tr>


     <tr>
       <td> employee desg :: </td>
       <td> <form:input path="job"/> </td>
     </tr>

     <tr>
       <td> employee salary :: </td>
       <td> <form:input path="sal"/> </td>
     </tr>

     <tr>
       <td colspan="2" align="center"><input type="submit" value="register Employee"/></td>
     </tr>
   </table>
 </form:form>
```

iv) add the following addtional code in employee_report.jsp

```
<c:if test="${!empty resultMsg}">
        <h3 style="color:green;text-align:center"> ${resultMsg }</h3>
    </c:if>
```

step12) Perform the following operations to edit product job

Two phases of edit operation
a) Launching selected Employee record for editing (for edit hyperlink request) GET
b) perform edit employee operation (form submission related POST request)

a) Launching selected Employee record for editing

i) add the following code in service interface ,service impl class
to get Employee details based on the given employee number

In service Interface

```
public Employee getEmployeeByNo(int no);
```

In service Impl class

```
@Override
    public Employee getEmployeeByNo(int no) {
        Employee emp=empRepo.findById(no).get();
        return emp;
    }
```

ii) Add the following code in Controller class to launch edit_employee form page having dyamically selected emp no based Emloyee details in Model class obj

```
@GetMapping("/edit")        for edit hyperlink request
public String showEditEmployeeForm(@RequestParam("no") int no,
                                   @ModelAttribute("emp") Employee emp) {
    //get Employee details dyamically based on the given emp no
    Employee emp1=service.getEmployeeByNo(no);
    //emp=emp1;
    BeanUtils.copyProperties(emp1, emp);
    //return lvn
    return "employee_edit";
}
```

iii) develop WEB-INF/pages/employee_edit.jsp page specifying the model attribute "emp"

employee_edit.jsp

```
<%@ page isELIgnored="false" %>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<h1 style="color:red;text-align:center"> Employee Employee </h1>
<form:form modelAttribute="emp">
  <table border="1" bgcolor="cyan" align="center">
  <tr>
      <td> employee no :: </td>
      <td> <form:input path="empno" readonly="true"/> </td>
    </tr>

    <tr>
      <td> employee name :: </td>
      <td> <form:input path="ename"/> </td>
    </tr>


    <tr>
      <td> employee desg :: </td>
      <td> <form:input path="job"/> </td>
    </tr>

    <tr>
      <td> employee salary :: </td>
      <td> <form:input path="sal"/> </td>
    </tr>

    <tr>
      <td colspan="2" align="center"><input type="submit" value="Edit Employee"/></td>
    </tr>
  </table>
</form:form>
```

b) perform edit employee operation (form submission related POST request)

i) add the following code in serivce interface and service Impl class to perform edit employee operation

In service Interface

```
public String editEmployee(Employee emp);
```

In service Impl class

```
@Override
    public String editEmployee(Employee emp) {
        int idVal=empRepo.save(emp).getEmpno();  //save(-) method can perform both save /edit operations
            return idVal+" Employee is updated ";

    }
```

ii) add the following @PostMapping("/edit") handler method in controller class to complte edit operation by processing the POST mode form submission request

step13) Perform the following operations to complete delete employee activities

i) Add the following code in service interface and service Impl class

In service interface

```
public String deleteEmployee(int no);
```

In service impl class

```
@Override
    public String deleteEmployee(int no) {
        empRepo.deleteById(no);
        return no+" emp no Employee is deleted";
    }
```

ii) Write @GetMapping("/delete") in controller class as shown below

```
@GetMapping("/delete")
public String deleteEmployee(@RequestParam("no") int no,
                             Map<String,Object> map) {
    //use service
    String msg=service.deleteEmployee(no);
    List<Employee> list=service.getAllEmployees();
    //keep results in model attributes
    map.put("resultMsg", msg);
    map.put("empsData",list);
    //return lvn
    return "employee_report";
}
```

ii) Add onclick based javascript confirm(' ___') function call to get double confirmation from enduser towards the deletion of record.

In employee_report.jsp

```
<td>
  <a onclick="return confirm('Do you want to delete')"
     href="delete?no=${emp.empno}"><img src="images/delete1.jpg" width="50" height="50">
  </a>
</td>
```

What is Double posting problem and how to solve it?

Ans) Repeating the request of form submission for multiple times becoz the clicks on refresh of button for multiple times and getting side effects in bad manner is called Duplicate form sumbission or double posting problem .. Generally form submission requests are POST mode requests carrying data .. The repetation of the same request definetely leads to problems.

=>POST is idempotent i.e not safe to repeat the request becoz it carries data and updates/modifies the data of server

non

=>GET is idempotent i.e safe to repeate the request becoz the does not carry data along with the request, in fact it reads the data from the server if repeat the reading data .. not a problem

Double posting problem use-cases are

==========

(a) Form filled with credit/debit card details .. for payment.
On the result page of this form submission if u press refresh button for multiple times then payment will be taken/deducted for multiple times.

(b) Form filled for employee/student/... registration
On the result page of this form submission ,if u press refresh button for multiple times then same student will be studnet/employee/... for multiple times with different ids .. if the app is generating id dynamically

The solutions for Double Posting Problem are

(a) Disable refresh Button using Java Script (Very complex - may not work or many give other side effeccts)
(b) use Serlvet Filters Session tokens logics (Complext implement using Servlet Filter)
(c) use PRG Pattern (PRG-POST REDIRECT -GET) (Easy to implement every where)

Legacy techniques

Modren technique

(c) use PRG Pattern (PRG-POST REDIRECT -GET) (Easy to implement every where)

=>This pattern says do not send response to browser from the POST mode handler method of the controller class becoz repeating the post mode request is always costly side effect as disscussed above..

So make the POST mode handler method redirecting the request to GET Mode handler method and send response to browser from there,becoz the repeation of GET mode request is not having any side effects.

Basic solution with PRG patttern

==========

In controller class

```
@PostMapping("/add")
    public String addEmployee(Map<String,Object> map,
                              @ModelAttribute("emp") Employee emp) {
        System.out.println("EmployeeOperationsController.addEmployee()");
        //use service
      String result=service.registerEmployee(emp);
      //keep results in model attributes
       map.put("resultMsg", result);
      //return LVN
      return "redirect:report";

    }

  @GetMapping("/report")
      public  String  showEmployeeReport(Map<String,Object> map) {
            System.out.println("EmployeeOperationsController.showEmployeeReport()");
             //use serivce
            List<Employee> list=service.getAllEmployees();
            // put the results in model attributes
            map.put("empsData",list);
            //return LVN
            return "employee_report";
      }//method
```

Limitation :: The model atttribute give POST mode hander method can nit be accessed in the view comp becoz that model attribute scope requst scope ..But GET mode handler method gets new request becoz redirection happend with browser

handler method gets new request becoz redirection happend with browser.
Final Impact :: Report data comes with add operation messages out

Improved Solution1: (PRG Pattern with Redriect Scope Flash Attributes)

In controller class

```
@PostMapping("/add")
    public String addEmployee(RedirectAttributes attrs,
                              @ModelAttribute("emp") Employee emp) {
        System.out.println("EmployeeOperationsController.addEmployee()");
        //use service
      String result=service.registerEmployee(emp);
      //keep results in model attributes (RedirectAttributes) .
      attrs.addFlashAttribute("resultMsg", result);
      //return LVN
      return "redirect:report";

     }

    public  String  showEmployeeReport(Map<String,Object> map) {
            System.out.println("EmployeeOperationsController.showEmployeeReport()");
             //use serivce
            List<Employee> list=service.getAllEmployees();
            // put the results in model attributes
            map.put("empsData",list);
            //return LVN
            return "employee_report";
        }//method
```

pro: RedirectAttributes scope is maintaindata b/w source request and redirected requested (bigger than request scope and smaller than session scope)

final impact :: Report data will be placed flash attribute data given POST mode but that will go off from second refresh of report page onwords.

cons

Improved Solution2: (PRG Pattern with Redriect Scope Flash Attributes)

==========

In controller class

```
@PostMapping("/add")
    public String addEmployee(HttpSession ses,
                              @ModelAttribute("emp") Employee emp) {
        System.out.println("EmployeeOperationsController.addEmployee()");
        //use service
      String result=service.registerEmployee(emp);
      //keep results in session attributes
       ses.setAttribute("resultMsg", result);
      //return LVN
      return "redirect:report";

     }

    @PostMapping("/add")
        public String addEmployee(HttpSession ses,
                                  @ModelAttribute("emp") Employee emp) {
            System.out.println("EmployeeOperationsController.addEmployee()");
            //use service
          String result=service.registerEmployee(emp);
          //keep results in session attributes
           ses.setAttribute("resultMsg", result);
          //return LVN
          return "redirect:report";

        }
```

pros: The Session atttributes data given by Post mapping method will be displayed on the result page will remain on it across the multiple requests becoz its session scope.

What is Double posting problem and how to solve it?

Ans) Repeating the request of form submission for multiple times becoz the clicks on refresh of button for multiple times and getting side effects in bad manner is called Duplicate form sumbission or double posting problem .. Generally form submission requests are POST mode requests carrying data .. The repetation of the same request definetely leads to problems.

=>POST is non idempotent i.e not safe to repeat the request becoz it carries data and updates/modifies the data of server

=>GET is idempotent i.e safe to repeate the request becoz the does not carry data along with the request, in fact it reads the data from the server if repeat the reading data .. not a problem

Double posting problem use-cases are

==========

(a) Form filled with credit/debit card details .. for payment.
On the result page of this form submission if u press refresh button for multiple times then payment will be taken/deducted for multiple times.

(b) Form filled for employee/student/... registration
On the result page of this form submission ,if u press refresh button for multiple times then same student will be studnet/employee/... for multiple times with different ids .. if the app is generating id dynamically

The solutions for Double Posting Problem are

(a) Disable refresh Button using Java Script (Very complex - may not work or many give other side effeccts) | Legacy techniques
(b) use Serlvet Filters Session tokens logics (Complext implement using Servlet Filter)
(c) use PRG Pattern (PRG-POST REDIRECT -GET) (Easy to implement every where) | Modren technique

(c) use PRG Pattern (PRG-POST REDIRECT -GET) (Easy to implement every where)

=>This pattern says do not send response to browser from the POST mode handler method of the controller class becoz repeating the post mode request is always costly side effect as disscussed above.. So make the POST mode handler method redirecting the request to GET Mode handler method and send response to browser from there,becoz the repeation of GET mode request is not having any side effects.

Basic solution with PRG patttern

==========

In controller class

```
@PostMapping("/add")
	public String addEmployee(Map<String,Object> map,
	                          @ModelAttribute("emp") Employee emp) {
	    System.out.println("EmployeeOperationsController.addEmployee()");
	    //use service
	  String result=service.registerEmployee(emp);
	  //keep results in model attributes
	   map.put("resultMsg", result);
	  //return LVN
	  return "redirect:report";

	}

   @GetMapping("/report")
	public  String  showEmployeeReport(Map<String,Object> map) {
		System.out.println("EmployeeOperationsController.showEmployeeReport()");
		  //use serivce
		List<Employee> list=service.getAllEmployees();
		// put the results in model attributes
		map.put("empsData",list);
		//return LVN
		return "employee_report";
	  }//method
```

Limitation :: The model atttribute give POST mode hander method can nit be accessed in the view comp becoz that model attribute scope requst scope ..But GET mode handler method gets new request becoz redirection happend with browser.
Final Impact :: Report data comes with out add operation messages

Improved Solution1: (PRG Pattern with Redriect Scope Flash Attributes)

==========

In controller class

```
---------------------------
@PostMapping("/add")
      public String addEmployee(RedirectAttributes attrs,
                                @ModelAttribute("emp") Employee emp) {
           System.out.println("EmployeeOperationsController.addEmployee()");
           //use service
        String result=service.registerEmployee(emp);
        //keep results in model attributes (RedirectAttributes) .
        attrs.addFlashAttribute("resultMsg", result);
        //return LVN
        return "redirect:report";

      }

     public  String  showEmployeeReport(Map<String,Object> map) {
                System.out.println("EmployeeOperationsController.showEmployeeReport()");
                 //use serivce
                List<Employee> list=service.getAllEmployees();
                // put the results in model attributes
                map.put("empsData",list);
                //return LVN
                return "employee_report";
            }//method
```

pro: RedirectAttributes scope is maintaindata b/w source request and redirected requested (bigger than request scope and smaller than session scope)

final impact :: Report data will be placed flash attribute data given POST mode but that will go off from second refresh of report page onwords.

cons

Improved Solution2: (PRG Pattern with Redriect Scope Flash Attributes)

==========

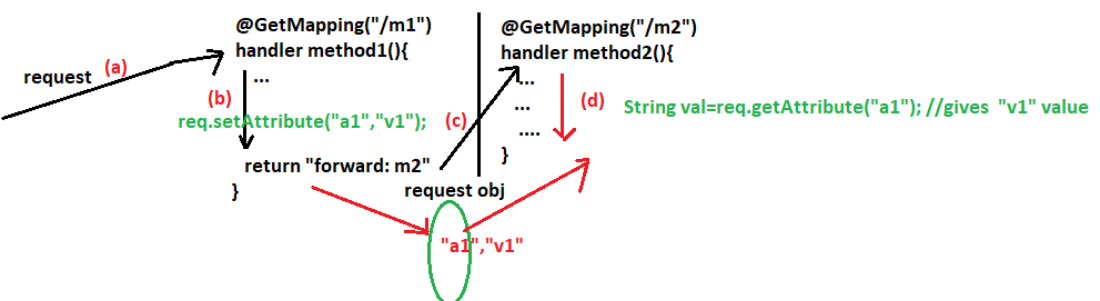
In controller class

```
------------------
@PostMapping("/add")
      public String addEmployee(HttpSession ses,
                                @ModelAttribute("emp") Employee emp) {
           System.out.println("EmployeeOperationsController.addEmployee()");
           //use service
        String result=service.registerEmployee(emp);
        //keep results in session attributes
         ses.setAttribute("resultMsg", result);
        //return LVN
        return "redirect:report";

      }

     @PostMapping("/add")
          public String addEmployee(HttpSession ses,
                                    @ModelAttribute("emp") Employee emp) {
               System.out.println("EmployeeOperationsController.addEmployee()");
               //use service
            String result=service.registerEmployee(emp);
            //keep results in session attributes
             ses.setAttribute("resultMsg", result);
            //return LVN
            return "redirect:report";

          }
```

pros: The Session atttributes data given by Post mapping method will be displayed on the result page will remain on it across the multiple requests becoz its session scope.

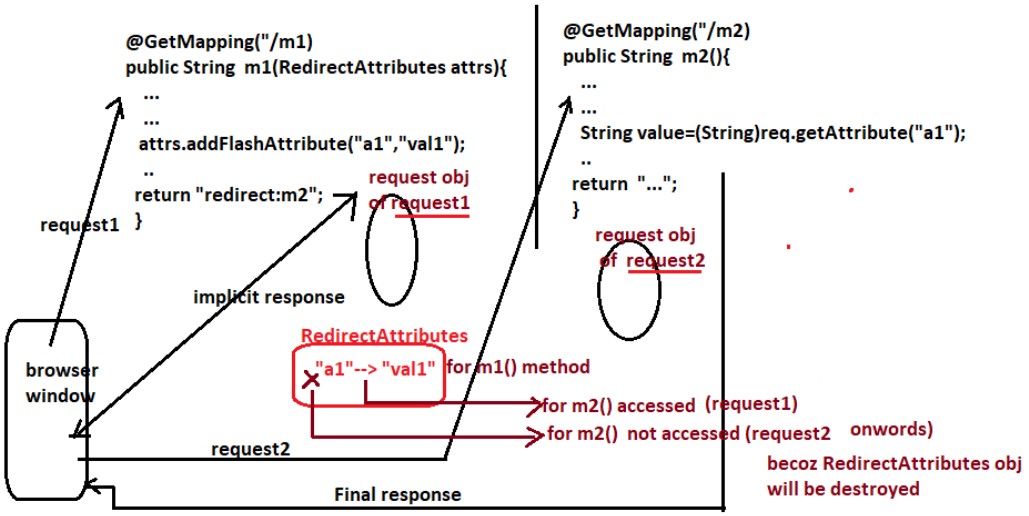
Q)What is difference among the request attributes scope , Redirect attributes scope and Sesssion Attributes Scope?

=>request attributes scope is specific to each request i.e they are visible through out request



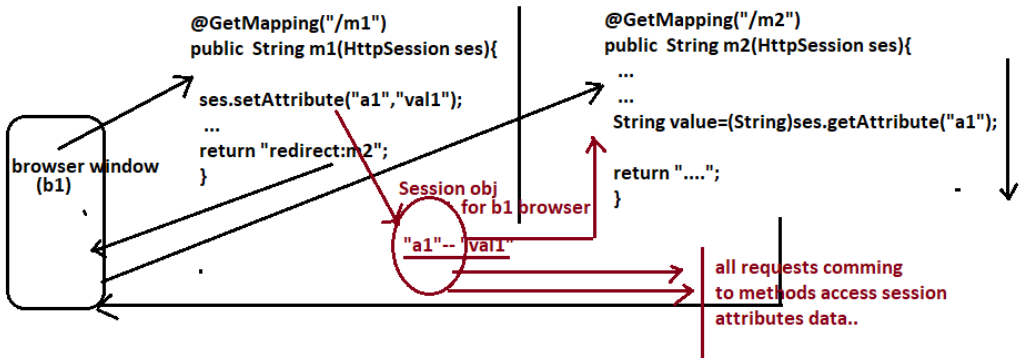
Both handler methods use same req,res objs .. So the data kept in one req object is visible and accessible in another req object.

=>RedirectAtrtibutes Scope Ridrection scope i.e source comp request data can be aaccessed in destination comp after redirecting request



=> Session attributes scope is session scope i.e specific to each browser s/w

=> Session atrributes data is visible and accessible across the multiple requests that are coming from a browser s/w



Conclusion:: a) request scope :: specific to each request
b) session scope :: specific to each browser s/w
c) Redirection scope :: specifc each redirection activity right from source request to dest request

---

EmployeeController.java

```
package com.nt.controller;

import java.util.List;
import java.util.Map;

import javax.servlet.http.HttpSession;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.nt.model.Employee;
import com.nt.service.IEmployeeMgmtService;

@Controller
public class EmployeeOperationsController {
	@Autowired
	private IEmployeeMgmtService service;

	@GetMapping("/")
  public  String showHome() {
		return "home";
	}
```

```
@GetMapping("/report")
public  String  showEmployeeReport(Map<String,Object> map) {
        System.out.println("EmployeeOperationsController.showEmployeeReport()");
         //use serivce
        List<Employee> list=service.getAllEmployees();
        // put the results in model attributes
        map.put("empsData",list);
        //return LVN
        return "employee_report";
}//method

@GetMapping("/add")
public String showAddEmployeeForm(@ModelAttribute("emp") Employee emp) {
        System.out.println("EmployeeOperationsController.showAddEmployeeForm()");
          emp.setJob("CLERK");  //intial value to display in form comp as initial value
          //return lvn
          return "employee_register";
}



@PostMapping("/add")
public String addEmployee(RedirectAttributes attrs,
                          @ModelAttribute("emp") Employee emp) {
        System.out.println("EmployeeOperationsController.addEmployee()");
        //use service
  String result=service.registerEmployee(emp);
  //keep results in model attributes (RedirectAttributes)
  attrs.addFlashAttribute("resultMsg", result);
  //return LVN
  return "redirect:report";

}


@GetMapping("/edit")
public String  showEditEmployeeForm(@RequestParam("no") int no,
                                    @ModelAttribute("emp") Employee emp) {
        //get Employee details dyamically based on the given emp no
         Employee emp1=service.getEmployeeByNo(no);
         //emp=emp1;
         BeanUtils.copyProperties(emp1, emp);
         //return lvn
         return "employee_edit";
}


@PostMapping("/edit")
public String  EditEmployee(@ModelAttribute("emp") Employee emp,
                            RedirectAttributes attrs) {
        //use  service
         String msg=service.editEmployee(emp);
         //keep results  as flashAttributes attributes in Redirect scope
          attrs.addFlashAttribute("resultMsg", msg);
        //return lvn
          return "redirect:report";
}


@GetMapping("/delete")
public String  deleteEmployee(@RequestParam("no") int no,
                              RedirectAttributes attrs) {
        //use service
        String msg=service.deleteEmployee(no);
         //keep results in model attributes
          attrs.addFlashAttribute("resultMsg", msg);
        //return lvn
          return "redirect:report";
}


}//class
```

Adding Bootstrap support to Project

==========

=>Bootstrap is collection ready made css styles ... which will be used to improve presentation part of the view comps..

=>By importing bootstrap readymade CSS Libraries we can apply ready made css styles to magnify responsiness in the application..

types of CSS (Cascading Style sheets)

a) inline styles (can be applied on every tag seperately using style attribute)

b) Embedded styles (we can prepre sepreate style classes to use them in the page)

c) exteranl styles (we can prepare seperate css files and we and link them with .html , .jsp files

to apply the styles)

=>Bootstrap lots extenal styles to apply on our jsp or html tags of view comps.

Procedure to apply bootrap styles on employee_report.jsp

step1) import boostrap css link to employee_report.jsp

in employee_report.jsp

```
<link rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"/>
```

collect this from www.bootstrapcdn.com
(https://www.bootstrapcdn.com/)

step2) collect container styles and apply on whole page using div tag

```
<div class="container">
  ...
  ....
  ....
  ....
  ....
</div>
```

step3) apply style classes like "table" ,"table-stripped", "table-hover" and etc..
on <table> tag.

```
<div class="container">

<c:choose>
  <c:when test="${!empty empsData}">
    <!-- <table border="1"  class="table table-striped" > -->
    <!--  <table border="1"  class="table table-hover" > -->
    <table border="1"  class="table" >
    <tr class="table-danger">
      <th>EmpNo  </th>
      <th>EmpName </th>
      <th>Job </th>
      <th>Salary</th>
      <th> Operations </th>
    </tr>
    <c:forEach var="emp" items="${empsData}">
       <tr  class="table-success">
         <td>${emp.empno} </td>
         <td>${emp.ename} </td>
         <td>${emp.job} </td>
        <td>${emp.sal} </td>
        <td><a href="edit?no=${emp.empno}"><img src="images/edit1.png" width="50" height="50"/></a>
                <a onclick="return confirm('Do you want to delete')" href="delete?no=${emp.empno}">
<img src="images/delete1.jpg" width="50" height="50"></a>  </td>
       </tr>
    </c:forEach>
    </table>
  </c:when>
    <c:otherwise>
      <h1 style="color:red;text-align:center"> Records not found </h1>
    </c:otherwise>
</c:choose>

                    <c:if test="${!empty resultMsg}">
                       <h3 style="color:green;text-align:center"> ${resultMsg }</h3>
                    </c:if>

 <br><br>
 <hr>
       <h1 style="text-align:center"><a href="./"><img src="images/home1.png" width="50" height="50">Home</a> </h1>
 <hr>
  <h1 style="text-align:center"><a  href="add"><img src="images/add.png" width="50" height="50"> Add Employee</a>
</h1>
```

Form validations in spring boot mvc

=> The process of verifying the pattern and format of the form data is called form validations

=> The difference b/w form validation logic and b.logic is

In form validation we verify pattern and format of the data .. where as in b.logic we use form data as inputs to perform calculations and to generate the results

examples for form validation logics

a) name is required
b) address should have minimum of 10 chars
c) age must be numeric value and should be there in the range of 1 through 100 and etc..

examples for b.logics

a) given name is already registered or not ?
b) credit number is valid or not
c) for the given address , product delivery is possible or not

Two types of form validations

1.Client Side Form validations

=> this form valiation logic executes in browser by going to browser along with the html
=>generally written in java script

2.Server side Form validations

=> These form validation logics are in java code.. in controller or service class before using form data in b.logic

=> The Best pratice for form validations is "write both Client side and server side form validations but enable server side form validations only when Cient side form validations are not done"

Different approaches of form validations implementation in spring boot mvc application

a) Using Programatic approach (Best)
(we can enable server side form validations only when client side form validations not done)
(Here we take seperate Validator class for validations)

b) Using Annotation driven approach (Not that much recomanded)
(Here Server side form validation will always be applied irrespective of the Client side form validations are performed or not)
Hibernate
(Here we add validation annotation in the Entity /Model class)

Procedure to apply Programatic form validations on spring boot mvc mini Project

==========

step1) prepare seperate properties file having form validation error messages

validation.properties (com.nt.validations pkg)

```
# Form validation Error message
empname.required=employee name is required
empname.maxlength= employee name can have max of 10 chars
empdesg.required=employee desg is required
empdesg.maxlength= employee desg can have max of 9 chars
empsal.required=employee salary is required
empsal.range= employee salary must be in 1 to 100000 range
```

note:: here keys and values are programmer choice

step2) Configure this properties file using @PropertySource

```
@SpringBootApplication
@PropertySource(value = "com/nt/validations/validation.properties")
public class BootMvcProj07MiniProjectCurdOperationsApplication {

	public static void main(String[] args) {
		SpringApplication.run(BootMvcProj07MiniProjectCurdOperationsApplication.class, args);

	}

}
```

step3) Develope Validator class implementing spring api's Validator(I) and provide form validation logic by overriding validate(-) method

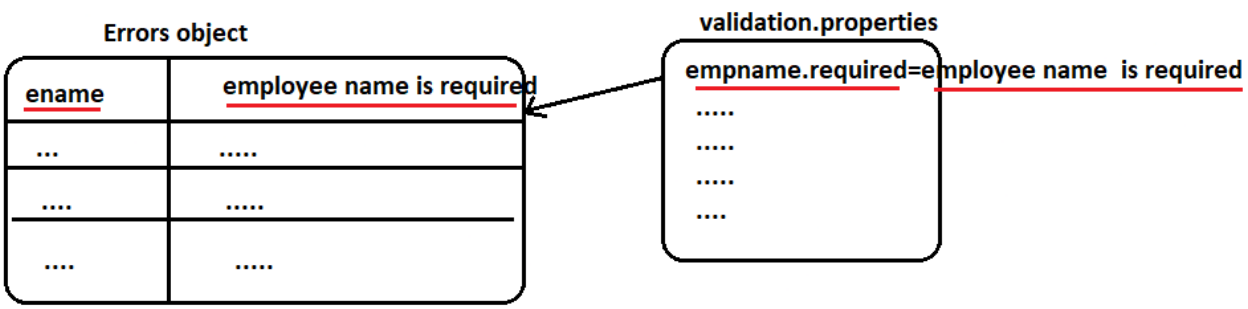
EmployeeValidator.java

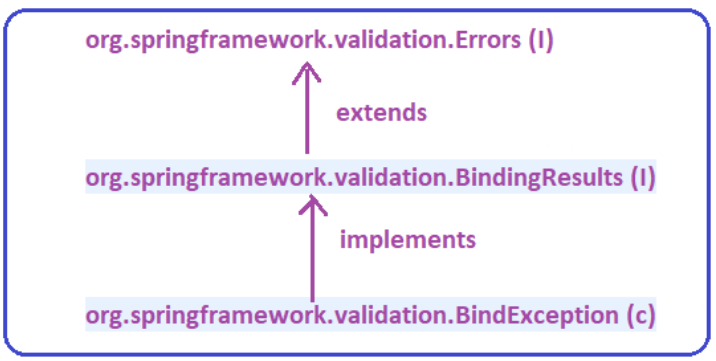
```
package com.nt.validations;

import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

import com.nt.model.Employee;

public class EmployeeValidator implements Validator {

	@Override
	public boolean supports(Class<?> clazz) {
		return clazz.isAssignableFrom(Employee.class);  //checks wheather we are passing
                                                        //correct model class or not
	}

	@Override
	public void validate(Object target, Errors errors) {
		System.out.println("EmployeeValidator.validate()");
		//type casting
```

if the supports(-) method returns false then validate(-) will not executed otherwise will be executed

```
                    //type casting
                    Employee emp=(Employee)target;
                    //form validation logics
                     //required rule  ename
                    if(emp.getEname()==null || emp.getEname().length()==0 || emp.getEname().equals(""))
                       errors.rejectValue("ename","empname.required");
                     //max length ename                                    key in the properites file
                    else if(emp.getEname().length()>10)
                           errors.rejectValue("ename","empname.maxlength");

                     //required rule  job
                    if(emp.getJob()==null || emp.getJob().length()==0 || emp.getJob().equals(""))
                       errors.rejectValue("job","empdesg.required");
                     //max length  job
                    else if(emp.getJob().length()>9)
                           errors.rejectValue("job","empdesg.maxlength");

                     //required rule  sal
                    if(emp.getSal()==null )
                       errors.rejectValue("sal","empsal.required");
                    else if(emp.getSal()<1 || emp.getSal()>100000)
                           errors.rejectValue("sal","empsal.range");
               }

          }
```

=>It is recomanded to write seperate validator class for every Entity class





step4) Inject EmployeeValidator class obj to Controller class and call supports(-) and validate(-,-) method in @PostMapping handlermethods related to addEmployee , Edit Employee operations

In controller class

==========

```
@Controller
public class EmployeeOperationsController {
	@Autowired
	private IEmployeeMgmtService service;
	@Autowired
	private EmployeeValidator  empValidator;

	@GetMapping("/")
  public  String showHome() {
		return "home";
	}

	@GetMapping("/report")
	public  String  showEmployeeReport(Map<String,Object> map) {
		System.out.println("EmployeeOperationsController.showEmployeeReport()");
		 //use serivce
		List<Employee> list=service.getAllEmployees();
		// put the results in model attributes
		map.put("empsData",list);
		//return LVN
		return "employee_report";
	}//method

	@GetMapping("/add")
	public String showAddEmployeeForm(@ModelAttribute("emp") Employee emp) {
		System.out.println("EmployeeOperationsController.showAddEmployeeForm()");
		  emp.setJob("CLERK");  //intial value to display in form comp as initial value
		  //return lvn
		  return "employee_register";
	}



	@PostMapping("/add")
	public String addEmployee(RedirectAttributes attrs,
					@ModelAttribute("emp") Employee emp,
					BindingResult  errors) {
		System.out.println("EmployeeOperationsController.addEmployee()");
		if(empValidator.supports(emp.getClass())) {
			empValidator.validate(emp, errors); //T
			if(errors.hasErrors())
				return "employee_register";
		}
		//use service
	  String result=service.registerEmployee(emp);
	  //keep results in model attributes (RedirectAttributes)
	  attrs.addFlashAttribute("resultMsg", result);
```

```
    //return LVN
    return "redirect:report";

  }


  @GetMapping("/edit")
  public String  showEditEmployeeForm(@RequestParam("no") int no,
                                      @ModelAttribute("emp") Employee emp) {
      //get Employee details dyamically based on the given emp no
       Employee emp1=service.getEmployeeByNo(no);
       //emp=emp1;
       BeanUtils.copyProperties(emp1, emp);
       //return lvn
       return "employee_edit";
  }


  @PostMapping("/edit")
  public String  EditEmployee(@ModelAttribute("emp") Employee emp,
                              RedirectAttributes attrs,
                              BindingResult errors) {
      if(empValidator.supports(emp.getClass())) {
            empValidator.validate(emp, errors); //T
            if(errors.hasErrors())
                  return "employee_edit";
      }
      //use  service
       String msg=service.editEmployee(emp);
       //keep results  as flashAttributes attributes in Redirect scope
        attrs.addFlashAttribute("resultMsg", msg);
      //return lvn
        return "redirect:report";
  }


  @GetMapping("/delete")
  public String  deleteEmployee(@RequestParam("no") int no,
                                RedirectAttributes attrs) {
      //use service
      String msg=service.deleteEmployee(no);
       //keep results in model attributes
        attrs.addFlashAttribute("resultMsg", msg);
      //return lvn
        return "redirect:report";
  }


}//class
```

Form validations in spring boot mvc

=> The process of verifying the pattern and format of the form data is called form validations

=> The difference b/w form validation logic and b.logic is

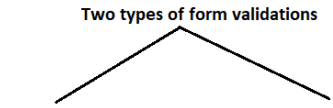
In form validation we verify pattern and format of the data .. where as in b.logic we use form data as inputs to perform calculations and to generate the results

examples for form validation logics

a) name is required
b) address should have minimum of 10 chars
c) age must be numeric value and should be there in the range of 1 through 100 and etc..

examples for b.logics

a) given name is already registered or not ?
b) credit number is valid or not
c) for the given address , product delivery is possible or not



1.Client Side Form validations

=> this form valiation logic executes in browser by going to browser along with the html
=>generally written in java script

2.Server side Form validations

=> These form validation logics are in java code.. in controller or service class before using form data in b.logic

for
=> The Best pratice form validations is "write both Client side and server side form validations but enable server side form validations only when Cient side form validations are not done"

Different approaches of form validations implementation in spring boot mvc application

a) Using Programatic approach (Best)
(we can enable server side form validations only when client side form validations not done)
(Here we take seperate Validator class for validations)

b) Using Annotation driven approach (Not that much recomanded)
(Here Server side form validation will always be applied irrespective of the Client side form validations are performed or not)
Hibernate
(Here we add validation annotation in the Entity /Model class)

Procedure to apply Programatic form validations on spring boot mvc mini Project

step1) prepare seperate properties file having form validation error messages

validation.properties (com.nt.validations pkg)

```
# Form validation Error message
empname.required=employee name is required
empname.maxlength= employee name can have max of 10 chars
empdesg.required=employee desg is required
empdesg.maxlength= employee desg can have max of 9 chars
empsal.required=employee salary is required
empsal.range= employee salary must be in 1 to 100000 range
```

note:: here keys and values are programmer choice

step2) Configure this properties file in application.properties

In application.properties

```
spring.messages.basename=com/nt/validations/validation
spring.messages.encoding=UTF-8
```

step3) Develope Validator class implementing spring api's Validator(I) and provide form validation logic by overriding validate(-) method

EmployeeValidator.java

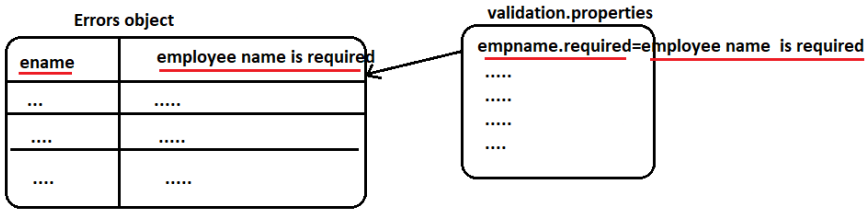
```
package com.nt.validations;

import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

import com.nt.model.Employee;
 @Component
public class EmployeeValidator implements Validator {
```

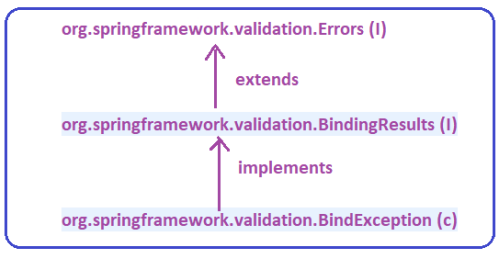
if the supports(-) method returns false then validate(-) will not executed otherwise will be executed

```
@Override
public boolean supports(Class<?> clazz) {
      return clazz.isAssignableFrom(Employee.class);  //checks wheather we are passing
                                                      //correct model class or not
}
```

```
@Override
public void validate(Object target, Errors errors) {
      System.out.println("EmployeeValidator.validate()");
      //type casting
      Employee emp=(Employee)target;
      //form validation logics
       //required rule  ename
      if(emp.getEname()==null || emp.getEname().length()==0 || emp.getEname().equals(""))
         errors.rejectValue("ename","empname.required");
       //max length ename                                  key in the properites file
      else if(emp.getEname().length()>10)
            errors.rejectValue("ename","empname.maxlength");

       //required rule  job
      if(emp.getJob()==null || emp.getJob().length()==0 || emp.getJob().equals(""))
         errors.rejectValue("job","empdesg.required");
       //max length  job
      else if(emp.getJob().length()>9)
            errors.rejectValue("job","empdesg.maxlength");

       //required rule  sal
      if(emp.getSal()==null )
         errors.rejectValue("sal","empsal.required");
      else if(emp.getSal()<1 || emp.getSal()>100000)
            errors.rejectValue("sal","empsal.range");
}

}
```

=>It is recomanded to write seperate validator class for every Entity class





step4) Inject EmployeeValidator class obj to Controller class and call supports(-) and validate(-,-) method in @PostMapping handlermethods related to addEmployee , Edit Employee operations

In controller class
==========

```
@Controller
public class EmployeeOperationsController {
    @Autowired
    private IEmployeeMgmtService service;
    @Autowired
    private EmployeeValidator  empValidator;

    @GetMapping("/")
  public  String showHome() {
          return "home";
    }

    @GetMapping("/report")
    public  String  showEmployeeReport(Map<String,Object> map) {
          System.out.println("EmployeeOperationsController.showEmployeeReport()");
           //use serivce
          List<Employee> list=service.getAllEmployees();
          // put the results in model attributes
          map.put("empsData",list);
          //return LVN
          return "employee_report";
    }//method

    @GetMapping("/add")
    public String showAddEmployeeForm(@ModelAttribute("emp") Employee emp) {
          System.out.println("EmployeeOperationsController.showAddEmployeeForm()");
           emp.setJob("CLERK");  //intial value to display in form comp as initial value
           //return lvn
           return "employee_register";
```

```
    }


    @PostMapping("/add")
    public String addEmployee(RedirectAttributes attrs,
                              @ModelAttribute("emp") Employee emp,
                              BindingResult  errors) {
        System.out.println("EmployeeOperationsController.addEmployee()");
        if(empValidator.supports(emp.getClass())) {
            empValidator.validate(emp, errors); //T
            if(errors.hasErrors())
                return "employee_register";
        }
        //use service
      String result=service.registerEmployee(emp);
      //keep results in model attributes (RedirectAttributes)
      attrs.addFlashAttribute("resultMsg", result);
      //return LVN
      return "redirect:report";

    }


    @GetMapping("/edit")
    public String  showEditEmployeeForm(@RequestParam("no") int no,
                                        @ModelAttribute("emp") Employee emp) {
        //get Employee details dyamically based on the given emp no
         Employee emp1=service.getEmployeeByNo(no);
         //emp=emp1;
         BeanUtils.copyProperties(emp1, emp);
         //return lvn
         return "employee_edit";
    }


    @PostMapping("/edit")
    public String  EditEmployee(@ModelAttribute("emp") Employee emp,
                                RedirectAttributes attrs,
                                BindingResult errors) {
        if(empValidator.supports(emp.getClass())) {
            empValidator.validate(emp, errors); //T
            if(errors.hasErrors())
                return "employee_edit";
        }
        //use  service
         String msg=service.editEmployee(emp);
         //keep results  as flashAttributes attributes in Redirect scope
          attrs.addFlashAttribute("resultMsg", msg);
        //return lvn
          return "redirect:report";
    }


    @GetMapping("/delete")
    public String  deleteEmployee(@RequestParam("no") int no,
                                  RedirectAttributes attrs) {
        //use service
        String msg=service.deleteEmployee(no);
         //keep results in model attributes
          attrs.addFlashAttribute("resultMsg", msg);
        //return lvn
          return "redirect:report";
    }


}//class
```

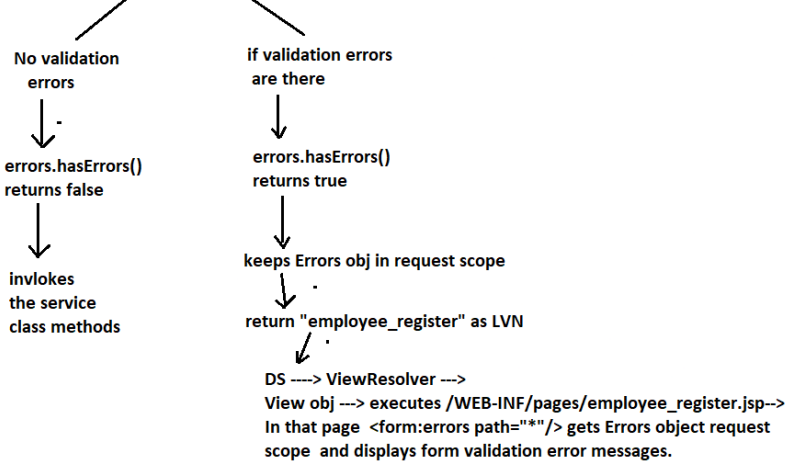
step5) *place <form:errors> tag in employee_register.jsp , employee_edit.jsp pages to display form validation error messages*

In employee_register.jsp and employee_edit.jsp

```
<form:form modelAttribute="emp">
    <p style="  color:red;text-align:center">

    <form:errors  path="*"/>

    </p>
```

Flow of execution w.r.t form validations

employee_register.jsp (form page) submission ----> DS ----> HadlerMapping --> gets
addEmployee(RedirectAttributes attrs,
@ModelAttribute("emp") Employee emp,
BindingResult errors) method signature which is havig @PostMapping("/register")-->
DS prepare empty RedirectAttributes obj, writes form data to Employee class obj , create
BindException obj and calls addEmployee(-,-,-) having those objs as the arguments --->
calls vaidator.suppors(-) having Model class name as argument --->
calls validator.validate(-,-) having model class obj and BindException obj(errors) as the argiments



How can we display form validation error messages beside the form comps?

=>use <form:errors> by specifying form comp name in "path" attribute

```
<form:form modelAttribute="emp">

  <table border="1" bgcolor="cyan" align="center">
    <tr>
      <td> employee name :: </td>
      <td> <form:input path="ename"/> <form:errors cssStyle="color:red" path="ename"/> </td>
    </tr>


    <tr>
      <td> employee desg :: </td>
      <td> <form:input path="job"/> <form:errors cssStyle="color:red" path="job"/> </td>
    </tr>

    <tr>
      <td> employee salary :: </td>
      <td> <form:input path="sal"/> <form:errors cssStyle="color:red" path="sal"/> </td>
    </tr>

    <tr>
      <td colspan="2" align="center"><input type="submit" value="register Employee"/></td>
    </tr>
  </table>
</form:form>
```

(or)

```
<style media = "all">
    body {
      background-color: pink;
    }
    span {
      color: red;
    }
  </style>

<h1 style="color:red;text-align:center"> Register Employee </h1>
<form:form modelAttribute="emp">


  <table border="1" bgcolor="cyan" align="center">
    <tr>
      <td> employee name :: </td>
      <td> <form:input path="ename"/> <form:errors path="ename"/> </td>
    </tr>


    <tr>
      <td> employee desg :: </td>
      <td> <form:input path="job"/> <form:errors path="job"/> </td>
    </tr>

    <tr>
      <td> employee salary :: </td>
      <td> <form:input path="sal"/> <form:errors path="sal"/> </td>
    </tr>

    <tr>
      <td colspan="2" align="center"><input type="submit" value="register Employee"/></td>
    </tr>
  </table>
</form:form>
```

3 types of validation errors in spring mvc /spring boot mvc web application
==========

(a) form validation errors (discussed above)
(b) typeMimatch errors
(c) Application logic errors

Procedure to place TypeMist match errors and application logic errors in existing project
==========

=>typeMisatch errors will be raised if form data binding to Model class obj gives problem like if we try to store string value(form data) to numeric property of model class then we get NumberFormat Exception with other messges which is basically the typemismatch error .. To replce technical message with our choice custom message take the support of "typeMistmatch.<property>" key in the properties file

in validation.properties

```
#typeMismatch error messages   (typeMismatch.<property>= message )
typeMismatch.sal=salary must be numeric value
```

=>small b.logic related rejection error are called application logic errors .. they can be placed in properties with our choice keys and values.

in validation.properties

```
# application logic-error messsages
job.reject=hackers are not allowed to register
```

Code in Controller class Handler methods

```
@PostMapping("/add")
    public String addEmployee(RedirectAttributes attrs,
                              @ModelAttribute("emp") Employee emp,
                              BindingResult  errors) {

        System.out.println("EmployeeOperationsController.addEmployee()");
    //checking for type mismatch errors
        if(errors.hasFieldErrors())
            return "employee_register";

        //checking form validation errors
    if(empValidator.supports(emp.getClass())) {
            empValidator.validate(emp, errors); //T
            if(errors.hasErrors())
                return "employee_register";
        }
      //application logic errors
    if(emp.getJob().equalsIgnoreCase("hacker")) {
     errors.rejectValue("job", "job.reject");
     return "employee_register";
    }

        //use service
      String result=service.registerEmployee(emp);
      //keep results in model attributes (RedirectAttributes)
      attrs.addFlashAttribute("resultMsg", result);
      //return LVN
      return "redirect:report";

    }

    @PostMapping("/edit")
        public String  EditEmployee(@ModelAttribute("emp") Employee emp,
                                    RedirectAttributes attrs,
                                    BindingResult errors) {
            //checking for type mismatch errors
            if(errors.hasFieldErrors())
                return "employee_edit";

            //checking form validation errors
     if(empValidator.supports(emp.getClass())) {
                empValidator.validate(emp, errors); //T
                if(errors.hasErrors())
                    return "employee_edit";
            }
        //application logic errors
      if(emp.getJob().equalsIgnoreCase("hacker")) {
        errors.rejectValue("job", "job.reject");
        return "employee_edit";
      }
            //use  service
            String msg=service.editEmployee(emp);
            //keep results  as flashAttributes attributes in Redirect scope
             attrs.addFlashAttribute("resultMsg", msg);
            //return lvn
              return "redirect:report";
        }
```

Writing Client side form vlaidation logics in the form pages of the Mini Project

=>For this it is better to write Java Script code in seperate .js file and link the file in multiple form pages using <script> tag.

step1) write JS form validation logics in seperate .js file

validations.js (in src/main/webapp/js folder)

```
function validation(frm){
        //empty the error messages
    document.getElementById("enameErr").innerHTML="";
    document.getElementById("jobErr").innerHTML="";
    document.getElementById("salErr").innerHTML="";
  //read form data
   let ename=frm.ename.value;
   let job=frm.job.value;
   let sal=frm.sal.value;
     let flag=true;
  //form validations (client side)
  if(ename==""){  //ename required rule
        document.getElementById("enameErr").innerHTML="employee name is mandatory(cs)";
     flag=false;
   }
 else if(ename.length>10){   //ename -max length rule
        document.getElementById("enameErr").innerHTML="employee name must have max of 10 chars(cs)";
    flag=false;
  }


 if(job==""){   //job -required rule
        document.getElementById("jobErr").innerHTML="employee desg is required(cs)";
    flag=false;
  }
 else if(job.length>9){  //job -max length rule
        document.getElementById("jobErr").innerHTML="employee desg  can hav max 9 characters(cs)";
   flag=false;
 }


 if(sal==""){  //sal required
        document.getElementById("salErr").innerHTML="employee salary is required(cs)";
   flag=false;
```

```
 }
 else if(isNaN(sal)){  //sal must be numeric vlaue
        document.getElementById("salErr").innerHTML="employee salary must numberic value(cs)";
    flag=false;
 }
 else if(sal<0 || sal>100000){  //sal -range rule
      document.getElementById("salErr").innerHTML="employee salary must be in the range 1 through 1000000(cs)";
      flag=false;
}


return flag;

}
```

step2) write the following code in form pages (employee_register.jsp and employee_edit.jsp)

employee_register.jsp

```
<%@ page isELIgnored="false"  %>
<%@taglib uri="http://www.springframework.org/tags/form"  prefix="form" %>

<style media = "all">
      body {
        background-color: pink;
      }
      span {
        color: red;
      }
    </style>

    <script language="JavaScript"  src="js/validations.js">
    </script>

<h1 style="color:red;text-align:center"> Register Employee </h1>
<form:form modelAttribute="emp" onsubmit="return validation(this)">
  <%-- <p style="color:red;text-align:center">
    <form:errors  path="*"/>
  </p>
  --%>
  <table border="1" bgcolor="cyan"  align="center">

    <tr>
      <td> employee name ::  </td>
      <td> <form:input path="ename"/>   <form:errors  path="ename"/>  <span id="enameErr"></span></td>
    </tr>
```

```
    <tr>
      <td> employee desg :: </td>
      <td> <form:input path="job"/>  <form:errors  path="job"/> <span id="jobErr"></span></td>
    </tr>

    <tr>
      <td> employee salary :: </td>
      <td> <form:input path="sal"/>   <form:errors  path="sal"/><span id="salErr"></span></td>
    </tr>

    <tr>
      <td colspan="2" align="center"><input type="submit" value="register Employee"/></td>
    </tr>
  </table>
 </form:form>
```

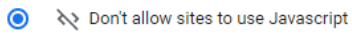
employee_edit.jsp

```
<%@ page isELIgnored="false"  %>
<%@taglib uri="http://www.springframework.org/tags/form"  prefix="form" %>
 <h1 style="color:red;text-align:center"> Employee Employee </h1>

  <script language="JavaScript"  src="js/validations.js">
    </script>

 <form:form modelAttribute="emp"  onsubmit="return validation(this)">
 <%-- <p style="color:red;text-align:center">
    <form:errors  path="*"/>
  </p> --%>

  <table border="1" bgcolor="cyan"  align="center">
  <tr>
      <td> employee no ::  </td>
      <td> <form:input path="empno"  readonly="true"/>  </td>
    </tr>

    <tr>
      <td> employee name ::  </td>
      <td> <form:input path="ename"/>  <span id="enameErr"></span></td>
    </tr>
```

```
    <tr>
      <td> employee desg ::  </td>
      <td> <form:input path="job"/>  <span id="jobErr"></span></td>
    </tr>

    <tr>
      <td> employee salary ::  </td>
      <td> <form:input path="sal"/>  <span id="salErr"></span></td>
    </tr>

    <tr>
      <td colspan="2" align="center"><input type="submit" value="Edit Employee"/></td>
    </tr>
  </table>
</form:form>
```

How to disable java script through chrome settings

chrome menu button : (three dots) ---> settings ---> search for javascript ---> site settings --->



How can we diaply fallback message when the java script is disabled using chrome settings?

Ans) place <noscript> tag having fallback message

```
<noscript>
    <h1 style="color:red;text-align:center">Please enable java script </h1>
</noscript>
```

=> "Only Server side form validations" improves network round trips browser and server if the form page is rejected for multiple times

=> "Only Client side form validations" reduces network round trips browser and server becoz browser submits request to server from form page only after having valid form data.. But there is possibility of disabling java script execution in the browser through browser setttings

=> Writing both server side and client side form validations gives that advantage of executing server side form validation logics though client side form validations are disabled.. when clietn side form validation not diasabled then both client side and server side form validations executes which is performence issue.

=>The Best solution is write both client side and server side form validations but enable server side form validations only when client side form validations are not done.

Enablind Server Side form Validations only when Client Side Form validations are not done

==========

=> Send flag to server side form client side indicating wheather client side form validations are done or not .. if done ignore execution of server form validation logics otherwise execute the server side form validation logics.
=>we can send the above flag with the support of hidden box having intial value "no" asssuming client side form validations are not done.. for onsubmit event if java script code executed then change that value to "yes" ... Based this hidden box value "yes" or "no" take the decision of executing server side form validation logics.

step1) place hidden box in both employee_register.jsp and employee_edit.jsp pages

```
<form:hidden path="vflag" />
```

step2) Take property in Model class having name "vflag" with initial value "no"

In Employee.java

```
private String vflag="no";
```

step3) Write code in Java script file /code to change vflag to "yes" indication the client side java script code is executed.

In validation.js file

```
// change vflag value to "yes" indicating client side form validations are done
frm.vflag.value="yes";
```

step4) Add the following code in @PostMapping addEmployee(-,-,-) , editEmployee(-,-,-) methods to enable server side form validations only when client side form validations are not done.

In Controller class

```
@PostMapping("/add")
    public String addEmployee(RedirectAttributes attrs,
                              @ModelAttribute("emp") Employee emp,
                              BindingResult errors) {

        System.out.println("EmployeeOperationsController.addEmployee()");
        //enable Server side form validations only when client side form validations are not done
      if(emp.getVflag().equalsIgnoreCase("no")) {
        //checking for type mismatch errors
         if(errors.hasFieldErrors())
            return "employee_register";
```

```
                //checking form validation errors
                if(empValidator.supports(emp.getClass())) {
                        empValidator.validate(emp, errors); //T
                        if(errors.hasErrors())
                                return "employee_register";
                }
                }

        //application logic errors
         if(emp.getJob().equalsIgnoreCase("hacker")) {
          errors.rejectValue("job", "job.reject");
          return "employee_register";
         }


              //use service
            String result=service.registerEmployee(emp);
            //keep results in model attributes (RedirectAttributes)
            attrs.addFlashAttribute("resultMsg", result);
            //return LVN
            return "redirect:report";

         }

@PostMapping("/edit")
      public String  EditEmployee(@ModelAttribute("emp") Employee emp,
                                          RedirectAttributes attrs,
                                          BindingResult errors) {
          if(emp.getVflag().equalsIgnoreCase("no")) {
                  //checking for type mismatch errors
                  if(errors.hasFieldErrors())
                          return "employee_edit";

                  //checking form validation errors
              if(empValidator.supports(emp.getClass())) {
                          empValidator.validate(emp, errors); //T
                          if(errors.hasErrors())
                                  return "employee_edit";
                  }
          }//if
     //application logic errors
    if(emp.getJob().equalsIgnoreCase("hacker")) {
      errors.rejectValue("job", "job.reject");
      return "employee_edit";
    }
            //use  service
             String msg=service.editEmployee(emp);
             //keep results  as flashAttributes attributes in Redirect scope
              attrs.addFlashAttribute("resultMsg", msg);
```

To make vflag property of Entity class not participating in any persistence activitu use @TRansient on the top of property

In Employee.java

```
---------------------------
        @Transient
      private  String vflag="no";
```

```
        //return lvn
          return "redirect:report";
    }
```

---

soft deletion vs hard deletion

Hard deletion :: Deleting the record of the db table perminently is called hard deletion
-> removing product from inventory/catalog
-> removing items added to Shoppring cart
-> Removing cards added for the payment

Soft deletion ::Not Physically deleting the record from db table .. but marking the record as not avaiable record or not active record is called soft deletion

=> closing bank account , employee regignation , patient discharge and etc..

=>For this in JPA , we have @SQLXxx annotation are given to execute custom query for standard persistence operations.. The annotations are

@SQLInsert :: we can place custom query form standard save(-) method (insert operation)

@SQLDelete :: we can place custom query from standard delete(-) method

@SQLUpdate :: we can place custom query for standard save(-) method (update operation)

and etc..

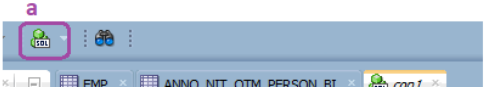
=>using @SQLDelete(-) we execute update query for repo.delete(-) method which changes the status of record to inactive.. This is nothing but soft deletion
=>But inacive status records should not participate in report generation select operation for that we need need @Where to form global implicit condition to filter softetly deleted records from queries exeution.

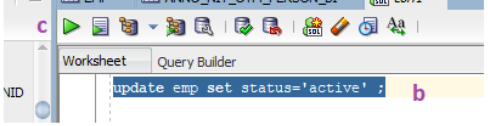
Example App
==========

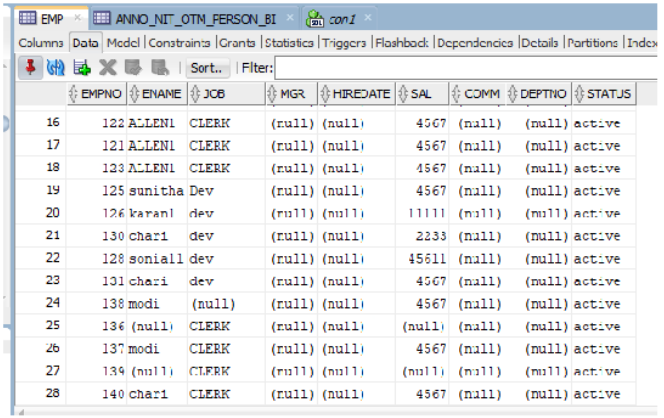
step1) add addtional "status" column in emp db table and place default "active" for all records

PK	Name	Data Type	Size	Not Null	Default
	EMPNO	NUMBER	4	☑	
	ENAME	VARCHAR2	10	☐	
	JOB	VARCHAR2	9	☐	
	MGR	NUMBER	4	☐	
	HIREDATE	DATE		☐	
	SAL	NUMBER	7	☐	
	COMM	NUMBER	7	☐	
	DEPTNO	NUMBER	2	☐	
	STATUS	VARCHAR2	10	☐	

a







EMP | ANNO_NIT_OTM_PERSON_BI | con1

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Index

Sort.. | Filter:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	STATUS
16	122	ALLEN1	CLERK	(null)	(null)	4567	(null)	(null)	active
17	121	ALLEN1	CLERK	(null)	(null)	4567	(null)	(null)	active
18	123	ALLEN1	CLERK	(null)	(null)	4567	(null)	(null)	active
19	125	sunitha	Dev	(null)	(null)	4567	(null)	(null)	active
20	126	karan1	dev	(null)	(null)	11111	(null)	(null)	active
21	130	chari	dev	(null)	(null)	2233	(null)	(null)	active
22	128	soniall	dev	(null)	(null)	45611	(null)	(null)	active
23	131	chari	dev	(null)	(null)	4567	(null)	(null)	active
24	138	modi	(null)	(null)	(null)	4567	(null)	(null)	active
25	136	(null)	CLERK	(null)	(null)	(null)	(null)	(null)	active
26	137	modi	CLERK	(null)	(null)	4567	(null)	(null)	active
27	139	(null)	CLERK	(null)	(null)	(null)	(null)	(null)	active
28	140	chari	CLERK	(null)	(null)	4567	(null)	(null)	active

step2) place @SQLDelete having update query that makes status col value as inactive value for delete operation

Also palce @Where annotation having status (which is global implicit condition)

SQL Queries

```
@Entity
@Table(name="emp")
@SQLDelete(sql = "UPDATE EMP SET STATUS='inactive' WHERE EMPNO=?")
@Where(clause = "STATUS <> 'inactive' ")
@Data
public class Employee  implements Serializable {
	@Id
	@SequenceGenerator(name = "gen1",sequenceName = "emp_id_seq",allocationSize = 1, initialValue = 1)
	@GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
	private Integer empno;
	@Column(length = 20)
	private  String ename;
	@Column(length = 20)
	private  String  job;
	private   Float sal;
	@Transient
	private  String vflag="no";
}
```

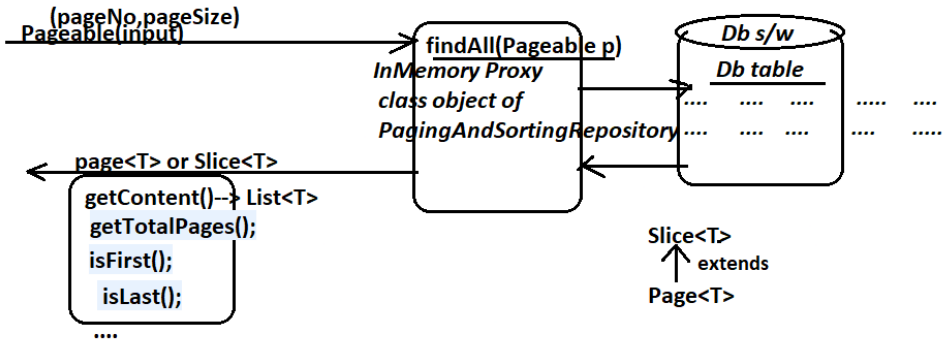
step3) execute the application.

Spring Boot MVC with spring data jpa Pagination

NOTE:: if one Project demands both soft deletion and hard deletion then use @SQLDelete(-) , deletById(-) methods of for softdeletion by having update query. So go for HQL/JPQL delete for hard deletion operation

Page<T> findAll(Pageable pageable); (PagingAndSortingRepository)

=> This methods takes pageNo(0 based),pageSize as inputs in the form of Pageable obj and returns output as Page<T> /Slice<T> obj having requested page records , pages count , current numbers ,total records and etc..



if total records count is 20 .. and pagesize is 5 then it gives 20 records in to 4 pages(5,5,5,5).. if we ask for 3 page (indirectly 4 page) records then it gives 16 to 20 records. if ask for 2nd page (indirectly 3 page) recors then it gives 11 to15 records.

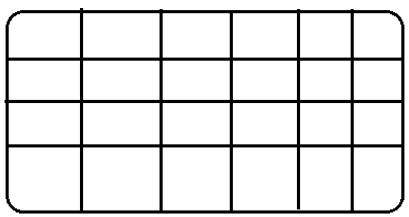
```
Pageable pageable=PageRequest.of(2,5);
```

note:: Pageable object can have both Paging and Sorting info

=> DispatcherServlet can prepare Pageable object and can give as handler method arg value if the parameter type is Pageable ... In this process it prepares that Pageable object having pageNo , pageSize gathered from the request parameters "page" ,"size".
=>We can give default pageNo, pageSize to the Pageable object of Handler method parameter using @PageableDefault annotation as shown below

```
@GetMapping("/emp_report")
    public String showEmployeeReport(
        @PageableDefault(page=0,size=3,sort="job",direction=Sort.Direction.ASC) Pageable pageable,
        Map<String,Object> map) {


    }
```

Report havin pagination..



first next [1] [2] [3] [4] previous last

EmpNo	EmpName	Job	Salary	Operations
120	ALLEN1	CLERK	4567.0	
7934	MILLER	CLERK	1300.0	
7876	ADAMS	CLERK	1100.0	

previous first [1] [2] [3] [4] [5] [6] [7] [8] [9] Last next

steps for coding

==========

step1) Add pagingation related report method in service interface and ins service impl class

In service Interface

```
public Page<Employee> getEmployeesPageData(Pageable pageable);
```

In service Impl class

```
@Override
    public Page<Employee> getEmployeesPageData(Pageable pageable) {
        Page<Employee> page=empRepo.findAll(pageable);
         return page;
    }
```

step3) place the following handler method in controller class

```
@GetMapping("/report")
    public String showEmployeeReport(@PageableDefault(page=0,size=3,sort = "job",direction =Sort.Direction.ASC ) Pageable pageable,
                                     Map<String,Object> map) {
        System.out.println("EmployeeOperationsController.showEmployeeReport()");
         //use serivce
        Page<Employee> page=service.getEmployeesPageData(pageable);
        // put the results in model attributes
        map.put("empsData",page);
        //return LVN
        return "employee_report";
    }//method
```

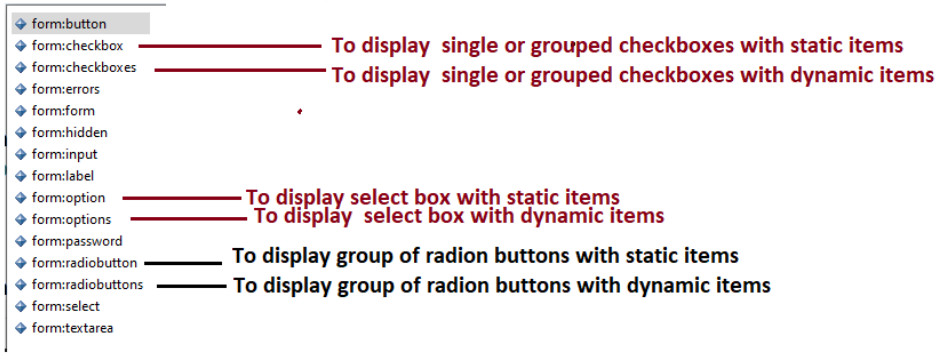
step3) Write the following code in employee_report.jsp

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

<link rel="stylesheet"
     href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"/>

<div class="container">

<c:choose>
  <c:when test="${!empty empsData.getContent()}">
    <!-- <table border="1" class="table table-striped" > -->
     <!-- <table border="1" class="table table-hover" > -->
     <table border="1" class="table" >
     <tr class="table-danger">
```

```
      <th>EmpNo  </th>
      <th>EmpName </th>
      <th>Job </th>
      <th>Salary</th>
      <th> Operations </th>
    </tr>
    <c:forEach var="emp" items="${empsData.getContent()}">
       <tr  class="table-success">
         <td>${emp.empno} </td>
         <td>${emp.ename} </td>
         <td>${emp.job} </td>
        <td>${emp.sal} </td>
        <td><a href="edit?no=${emp.empno}"><img src="images/edit1.png" width="50" height="50"/></a>
                <a onclick="return confirm('Do you want to delete')" href="delete?no=${emp.empno}"><img src="images/delete1.jpg" width="50" height="50"></a>  </td>
       </tr>
    </c:forEach>
    </table>

     <p  style="text-align:center">
    <c:if test="${empsData.hasPrevious()}">
      <a href="report?page=${empsData.getPageable().getPageNumber()-1}"> previous </a>     
     </c:if>
     <c:if test="${!empsData.isFirst()}">
      <a href="report?page=0">first</a>     
     </c:if>

          <c:forEach var="i" begin="1" end="${empsData.getTotalPages()}" step="1">
              [<a href="report?page=${i-1}">${i}</a>]    
          </c:forEach>

     <c:if test="${!empsData.isLast()}">
      <a href="report?page=${empsData.getTotalPages()-1}">Last</a>     
     </c:if>


      <c:if test="${empsData.hasNext()}">
        <a href="report?page=${empsData.getPageable().getPageNumber()+1}"> next </a>
      </c:if>
     </p>

  </c:when>
    <c:otherwise>
      <h1 style="color:red;text-align:center"> Records not found </h1>
    </c:otherwise>
</c:choose>

                  <c:if test="${!empty resultMsg}">
                     <h3 style="color:green;text-align:center"> ${resultMsg }</h3>
                  </c:if>

 <br><br>
 <hr>
       <h1 style="text-align:center"><a href="./"><img src="images/home1.png" width="50" height="50">Home</a> </h1>
 <hr>
  <h1 style="text-align:center"><a  href="add"><img src="images/add.png" width="50" height="50"> Add Employee</a> </h1>
</div>
```

Reference Data generation using @ModelAttribute(-) for select box , list box , group checkboxes and grouped radion buttons

spring mvc form tag library tags



Component in form page	property type in Model class
1) select box (allows single item selection)	String
2) List box (Allows to select multiple items)	String[] or java.util.List or java.util.Set
3) Single checkboxed (allows to select or deselect)	String
4) Group checkboxes (allow to select/deselect bunch of items)	String[] or java.util.List or java.util.Set
5) Grouped Radio Buttons (allows to select one radion button at a time)	String

=> if want to get initial value for text box or password box we can do that work from Model class properties by assinging initial values.

name :: <form:input path="name"/>

name :: raja

In Model class

```
private String name="raja";
```

=>For List Box ,select box ,grouped checkboxes, grouped radio buttons if want to get their all items dynamically from Db s/w then we can not dependent on putting those values to the model class properties.

In form page

```
<form:select path="language">

</form>
```

In Model class

```
String language="hindi";
```

=>For these special 4 comps we need to take the support of reference Data constructed by @ModelAttribute(-) to put data into elements dynamically

@ModelAttribute(-) is multiple pupose annotations

a) To bind from data into Model class object (data binding)

b) To create dyanmic data as the reference data required for

the "special 4 comps (making reference data as "special 4 comps
dynamic element values)

use-cases :: a) displaying all countries to select box by collecting from DB m service class
b) displaying all languages to select box by collecting from DB or service class
c) displaying all hobies to grouped checkboxes by collecting from DB or service class
d) displaying all IT courses to the ListBox by collecting them from DB or service class

To consturct addtional reference required for 4 comps we need to use
@ModelAttribute(-) method in controller class having the following syntax

Giving addtional reference data required for the select box

In controller class

```
@ModelAttribute("countriesInfo")
public List<String> populateCountries(){
   ...
   ... //logic to get countries dyamically from DB or service class
   ..
}
```

must match

In form page

```
<form:select path="country">
  <form:options items="${countriesInfo}"/>
</form:select>
```

In Model class

```
private String country="india"; //becomes default item that is selected.
```

Locale means language +country
eg:: en-US
fr-FR
de-DE
fr-CA (french as it speaks in candata)
hi-IN
en-IN (english as it speaks in India)

Example App displaying select box with dyamic countries with the support of reference data using @ModelAttribute(-)

step1) Add the following code in service Inteface , service Impl class to get all countries dyamically with the support of Locale API (java.util pkg)

In service Interface

```
public Set<String> getAllCountries();
```

In service Impl class

```
@Override
    public Set<String> getAllCountries() {
        // get All Locales of the world
        Locale locales[]=Locale.getAvailableLocales();
        Set<String> countrySet=new TreeSet();
        for(Locale l:locales) {
            if(l!=null)
              countrySet.add(l.getDisplayCountry());
        }
        return countrySet;
    }//method
```

}//method

step3) place @ModelAttribute(-) method in Controller class invoking the service class method

```
@ModelAttribute("countriesInfo")
public Set<String> populateCountries(){
    System.out.println("EmployeeOperationsController.populateCountries()");
    //use service
    Set<String> countrySet=service.getAllCountries();
    return countrySet;
}
```

This method will be called by for every GET ,POST mode request and makes the returned Collection data as the reference data in request scope having given name "countriesInfo" as the attribute name

step4) add addtional comp in the form page of type select box by making reference data as select box items data..

In employee_register.jsp

```
<tr>
    <td> select counstry :: </td>
    <td><form:select path="country">
            <form:options items="${countriesInfo}"/>
        </form:select>
    </td>
</tr>
```

form comp name

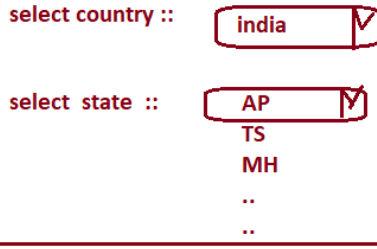
Model attribute name (reference attribute name)

step5) take addtional attribute in model class to hold default item for the the above select box and also hold the selected item as part of data binding.

```
@Entity
@Table(name="emp")
@SQLDelete(sql = "UPDATE EMP SET STATUS='inactive' WHERE EMPNO=?")
@Where(clause = "STATUS <> 'inactive' ")
@Data
public class Employee  implements Serializable {
    @Id
    @SequenceGenerator(name = "gen1",sequenceName = "emp_id_seq",allocationSize = 1, initialValue = 1)
    @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
    private Integer empno;
    @Column(length = 20)
    private  String ename;
    @Column(length = 20)
    private  String  job;
    private   Float sal;
    private  String status="active";
    @Transient
    private  String vflag="no";
    private String country="India";

}
```

Q) In Spring MVC tags , what is difference b/w <form:option> tag and <form:options> tag?

Getting states into one SelectBox based on country from another SelectBox Box?



=>There is no pre-defined class to get states based on the select country .. So we need to maintain either in properties file or DB or RestAPI comp.
(eg: ApiGee)

step1) add states of India , UnitedStates in properties file manually

In application.properties

```
----------------------------------------
# Add Sates for countries
India=AP,TS,MH,OD,TN,KR,MP,UP,BR,DL,WB,RJ,JH,AS,PB,KL,CH,GA,JK,NL,UK,MZ,MN,LD,HR,HP,DD,GJ,PY,AN
United\u0020States=LA,KY,NY,KS,LA,ME,MD,MA,MI,MN,MS,AS,WS,CA,WD,AK,GA,TS,AR,OR,AL,VT,WY,DE,HI,ID,IA,NV,OK,SC,SD,RI,PA,WI
```

space in unicode char set

step2) add the following code in serivce Interface and service Impl class

In service Interface

```
public List<String> getStatesByCountry(String country);
```

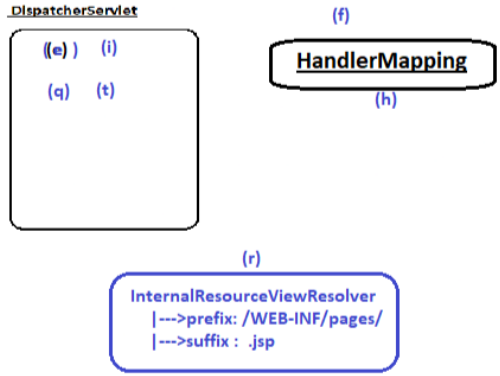
In service Impl class

```
@Override                                   (I)
    public List<String> getStatesByCountry(String country) {
        // get states of a country through Environment obj
        String statesInfo=env.getRequiredProperty(country);
        //convert comma seperated values into List colelction using "," as delimeter
        List<String> statesList=Arrays.asList(statesInfo.split(","));
        // sort collection  (natural sorting)
        Collections.sort(statesList);
        return statesList;   (m)
    }
```

step3) Add "state" property in Model class

```
@Entity
@Table(name="emp")
@SQLDelete(sql = "UPDATE EMP SET STATUS='inactive' WHERE EMPNO=?")
@Where(clause = "STATUS <> 'inactive' ")
@Data
public class Employee  implements Serializable {
    @Id
    @SequenceGenerator(name = "gen1",sequenceName = "emp_id_seq",allocationSize = 1, initialValue = 1)
    @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
```

```
    private Integer empno;
    @Column(length = 20)
    private  String ename;
    @Column(length = 20)
    private  String  job;
    private   Float sal;
    private  String status="active";
    @Transient
    private  String vflag="no";
    private String country="India";
    private String state;

}
```



step4) add the following employee_register.jsp (java script code + spring mvc tags code)

In *employee* register.jsp

```
<script language="JavaScript">   (c)
     function sendReqForStates(frm){
       frm.action="statesurl"  //request path for handler method
       frm.submit(); //submits the request
     }          (d)

 </script>

<tr>
  <td> select country :: </td>
  <td><form:select path="country" onchange="sendReqForStates()">
          <form:options items="${countriesInfo}"/>
     </form:select>
   </td>
</tr>

<tr>
  <td> select state :: </td>
  <td><form:select path="state">
          <form:options items="${statesInfo}"/>
     </form:select>
   </td>
</tr>
```

(b)
select UnitedStates (a)
(u)
Model attribute name having states Info

step4) add the following addtional handler method in controller class

```
@PostMapping("/statesurl")      (j)
    public  String   showStatesByCountry(@RequestParam("country") String country,
                                         @ModelAttribute("emp") Employee emp,
                                         Map<String,Object> map ) {
        //use service
(n)     List<String> statesList=service.getStatesByCountry(country);   (k)
```

```
        // put statesList in model attribute
        map.put("statesInfo", statesList);   (o)
        //return LVN of form page
        return "employee_register";   (p)

    }
```

(g)

What is the difference between <form:options> and <form:option>?

Ans) <form:option> for aadding static items select box or list box

```
<form:select path="country">
   <form:option value="india">India</form:option>
   <form:option value="China">China</form:option>
   ....
   ...
</form:select>
```

<form:options> tag is given to add items to select box /list box dyamically by getting them from Model attributes of any form

In form page

```
<tr>
    <td> select country ::  </td>
    <td><form:select path="country" onchange="sendReqForStates()">
            <form:options items="${countriesInfo}"/>
        </form:select>
    </td>
</tr>
```

In controller class

```
@ModelAttribute("countriesInfo")
    public Set<String>  populateCountries(){
        System.out.println("EmployeeOperationsController.populateCountries()");
        //use service
        Set<String> countrySet=service.getAllCountries();
        return countrySet;
    }
```

==========
Assignment
==========

add following comps with dynamic data using reference data concept

(a) grouped radio buttons having gender info
(b) grouped checkboxes having hobies info
(c) List box having courses to select

InitBinder in spring MVC /Spring boot MVC

==========

=>In spring we use the support of PropertyEditors to convert given String values to different types of required value before performing injection or binding.

=> All PropertyEditors are classes implementing PropertyEditor(I) directtly or indirectly

=> Spring gives multiple Built-in PropertyEditors

- ByteArrayPropertyEditor
- CharacterEditor
- CharArrayPropertyEditor
- CharsetEditor
- ClassArrayEditor
- ClassEditor
- CurrencyEditor
- CustomBooleanEditor
- CustomCollectionEditor
- CustomDateEditor
- CustomMapEditor
- CustomNumberEditor
- FileEditor
- InputSourceEditor
- LocaleEditor
- PathEditor
- PatternEditor
- PropertiesEditor
- ReaderEditor
- ResourceBundleEditor
- StringArrayPropertyEditor
- StringTrimmerEditor
- TimeZoneEditor
- URIEditor
- URLEditor
- UUIDEditor
- ZoneIdEditor

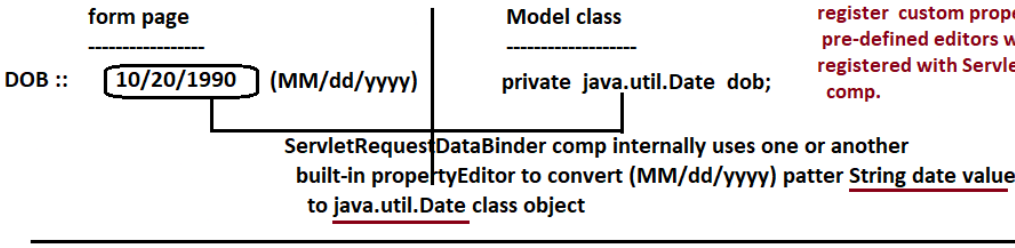
s are

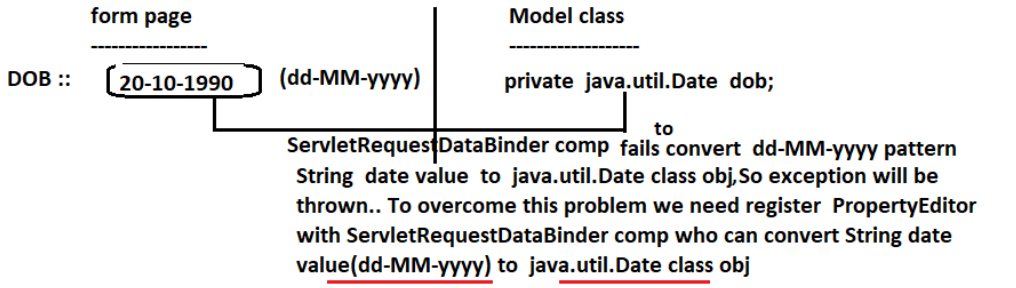
=>Most of these propertyeditor automatically registered with IOC container and will be used internally towards data injection or data binding process.

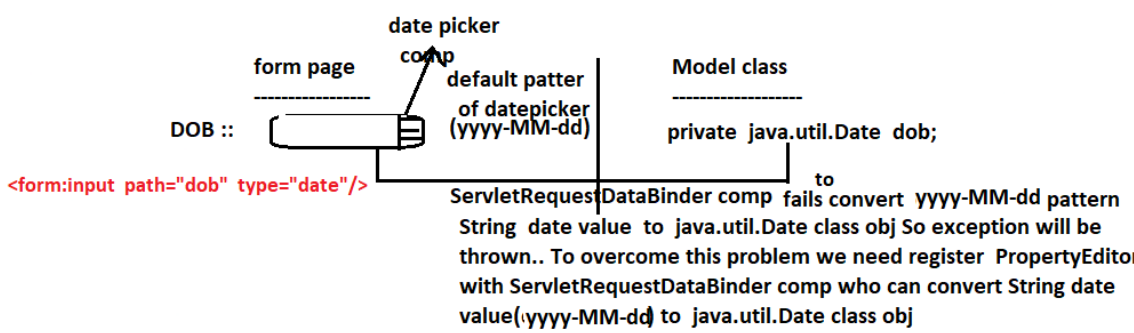
java.lang.Object
  org.springframework.validation.DataBinder
    org.springframework.web.bind.WebDataBinder
      org.springframework.web.bind.ServletRequestDataBinder

=> DispatcherServlet intenrally uses ServletRequestDataBinder comp to perform data binding activity that is writing form data to Model class object properties. This binder comp internally uses multiple registered Property editor to convert form comps supplied string input values to different types values as required for the Model class object proeprties.

DS uses ServeltRequestDataBinder for data binding activity.. So we need register custom propertyeditors or pre-defined editors which are not already registered with ServletRequestDataBinder comp.







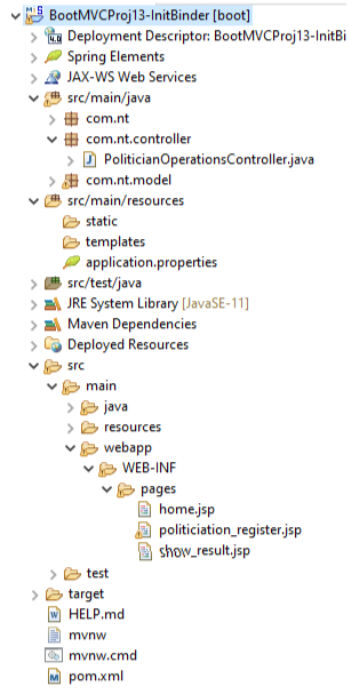
note:: To register PropertyEditors (ready made or custom) with ServletRquestDataBinder comp we need to use @InitBinder methods..

```
@InitBinder
public   void   myBinder(WebDataBinder  binder){

binder.register customEditor(-,-,-);

}
```

note:: @ModelAttribute(-) method (reference data method) executes every time form launch
note: @InitBinder method executes for each form lanuch and form sumission

=>CustomDateEditor is pre-defined PropertyEditor that takes given SimpleDateFromat obj's date pattern ..converts String date value that is given in the specified pattern to java.util.Date class obj

---



home.jsp

```
<h1 style="color:red;text-align:center"><a href="register"> Register Politician</a></h1>
```

Model class

```
package com.nt.model;

import java.util.Date;

import lombok.Data;

@Data
public class PoliticianProfile {
        private Integer pid;
        private String pname;
        private  String party;
        private  Date  dob=new Date(100,0,01);
        private  Date  doj;
        private  boolean consPost=false;

}
```

```
//controleIr clss

package com.nt.controller;

import java.text.SimpleDateFormat;

import org.springframework.beans.propertyeditors.CustomDateEditor;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import com.nt.model.PoliticianProfile;

@Controller
public class PoliticianOperationsController {

        @GetMapping("/")
        public  String  showHome() {
                System.out.println("PoliticianOperationsController.showHome()");
                 //return lvn
                return "home";
        }
```

```
	@GetMapping("/register")
	public  String showFormPage(@ModelAttribute("pp") PoliticianProfile profile) {
		System.out.println("PoliticianOperationsController.showFormPage()");
		//return LVN
		return  "politiciation_register";
	}

	@PostMapping("/register")
	public  String  registerPolitician(@ModelAttribute("pp") PoliticianProfile profile) {
		System.out.println("PoliticianOperationsController.registerPolitician()");
		System.out.println("model class obj data::"+profile);
		//by invoking service class .. u can execute b.logic

		return "show_result";
	}

	@InitBinder
	public  void  myDateBinder(WebDataBinder binder) {
		System.out.println("PoliticianOperationsController.myDateBinder()");
		SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
		binder.registerCustomEditor(java.util.Date.class, new CustomDateEditor(sdf, true));
	}

}
```

form page

```
<%@ page isELIgnored="false"%>
<%@taglib uri="http://www.springframework.org/tags/form"  prefix="form"%>

<h1 style="color:red;text-align:center">Politician Registration</h1>

<form:form method="POST" modelAttribute="pp">
 <table border="0" bgcolor="cyan" align="center" >
   <tr>
    <td> Policitiiation name::  </td>
    <td><form:input path="pname"/> </td>
   </tr>

   <tr>
    <td> Policitiiation Party Name::  </td>
    <td><form:input path="party"/> </td>
   </tr>

   <tr>
    <td> Policitiiation DOB::  </td>
    <td><form:input path="dob"  type="date"/> </td>
   </tr>
   <tr>
    <td> Policitiiation DOJ::  </td>
    <td><form:input path="doj"  type="date"/> </td>
   </tr>
    <tr>
    <td>Has ConstitutionPost?::  </td>
    <td>  <form:radiobutton  path="consPost"  value="true" /> yes   
          <form:radiobutton  path="consPost"  value="false" /> no
     </td>
   </tr>
   <tr>
    <td colspan="2"> <input type="submit"  value="register"> </td>
   </tr>
 </table>

</form:form>
```

show_result.jsp

```
<%@page isELIgnored="false" %>

 <b><i> Model class obj data is :: ${pp}</i></b>
 <br>
 <a href="./">home</a>
```

Handler Interceptor

==========

=> This comp logic can be added to controller comp as the extension hook logic ... which can be executed as pre , post and after logics with respect to controller classes execution

=> These are like servlet filter comps .. but servlet filter comps allows us to add only pre ,post request processisng logics with respect to other web comps of the web application where as HandlerInterceptor allows us to add pre,post, after logics with respect to controller classes.

=> To develop Handler Interterceptor for Controller class we need to take a class implementing HandlerInteceptor (I) which is having 3 methods (java 8 interface having all the 3 methods as the default methods)

`default void`	`postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView)` Interception point after successful execution of a handler.

usecase :: sending addional model data having system date,time and etc..

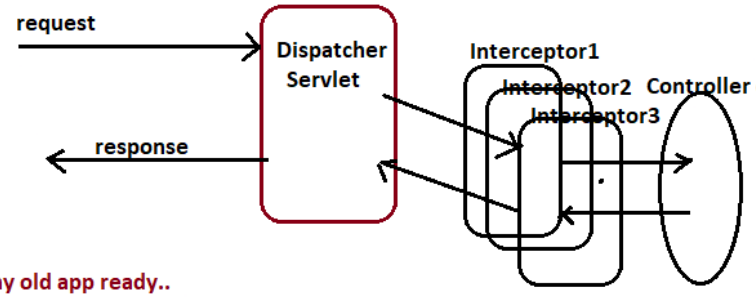
`default boolean`	`preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)` Interception point before the execution of a handler. **(controller)**

usecase:: time checking (trading b/w 9am to 3pm) , allowing requests only from certain browser and etc..

`default void`	`afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)` Callback after completion of request processing, that is, after rendering the view.

usecase : Nullifying request attributes ,session attributes and etc..

=>Since all the 3 methods are default methods.. So we can overide only those methods in which we are intrested in



if multiple interceptors are applied on the same controller class then trap request in certain order before going to controller.. but they trap response in revers order while leaving the controller

(1,2,3)

(3,2,1)

Example App

==========

step1) Keep any old app ready..

(BootMVCProj14-WishMessageApp-HandlerInterceptor)

step2) Develop Handler Interceptor class.

TimeCheckInterceptor.java

```
import java.time.LocalDateTime;

import javax.servlet.RequestDispatcher;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import org.springframework.web.servlet.HandlerInterceptor;

public class TimeCheckInterceptor implements HandlerInterceptor {

	@Override
	public boolean preHandle(HttpServletRequest req, HttpServletResponse res, Object handler)
			throws Exception {
		System.out.println("TimeCheckInterceptor.preHandle()");
		// get System date and time
		LocalDateTime  ldt=LocalDateTime.now();
		//get  current hour of the day
		int hour=ldt.getHour();
		if(hour<9 || hour>17 ) {
			RequestDispatcher rd=req.getRequestDispatcher("/timeout.jsp");
			rd.forward(req, res);
			return false;
		}
		return true;

	}

}
```

step4) place timeout.jsp page in src/main/web app folder

timeout.jsp

```
<%@ page isELIgnored="false" %>

<h1 style="color:red;text-align:center"> Trading timing are 9am to  5pm</h1>

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

step5) Register Interceptor with Inteceptor registry by Web MVC configurer class as spring bean

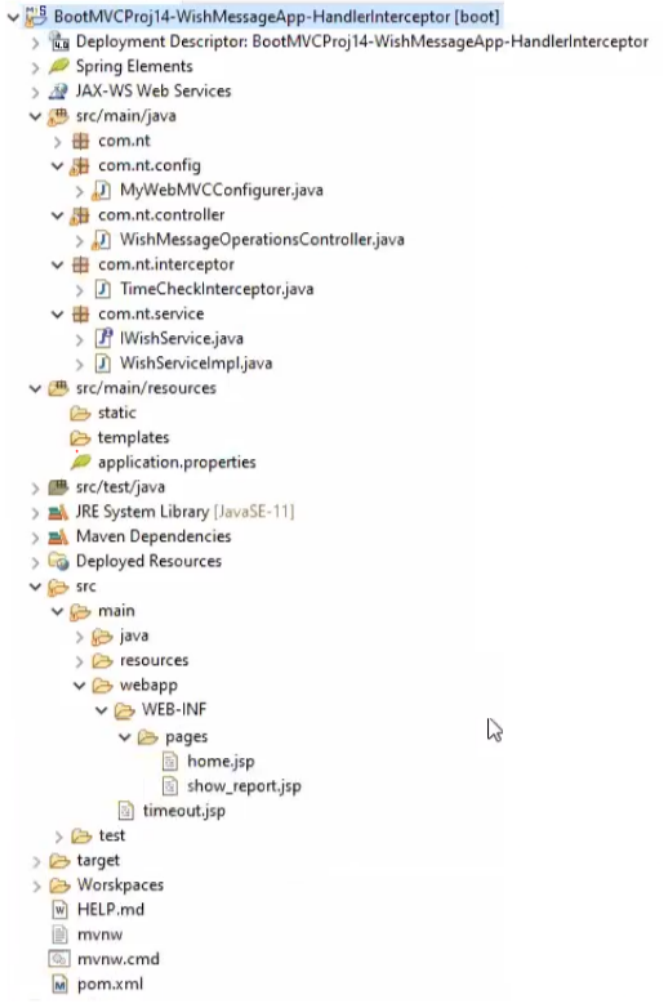
MyWebMVCConfigurer.java

```
package com.nt.config;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import com.nt.interceptor.TimeCheckInterceptor;


@Component
```

```java
public class MyWebMVCConfigurer implements WebMvcConfigurer {

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        System.out.println("MyWebMVCConfigurer.addInterceptors()");
        registry.addInterceptor(new TimeCheckInterceptor());
    }

}
```

step6) Run the application by changing the timings

1 execution having time b/w 9 am to 5pm
2 execution having time before 9am and after 5pm



=>In servlet ,jsp based web application we take the support servlet filters to add pre-request processing and post response generation logics

=>In Spring MVC,spring boot MVC web applications we take the support of Handler interfaceptors to pre ,post after completion logics.

==========

I18n in spring boot MVC application

==========

I18n :: Internationlization (I 18 letters n)

=> Making our app working for different Locales is called enabling I18n on the application

=> Locale means language + country

eg:: en-US (english as it speaks in USA)
fr-FR (french as it speaks in France)
hi -IN (hindi as it speaks in India)
fr- CA (french as it speaks in Canada)
and etc..

=>By enabling I18n on the web application we can change the following presentations according to the Locale that is chosen

a) Presentation Lables
b) Date patterns
c) Time patterns
d) Number formats
e) Currency Symbols
f) Content indentation (most of languages left to right but urdu,arabic and etc.. right to left)
and etc..

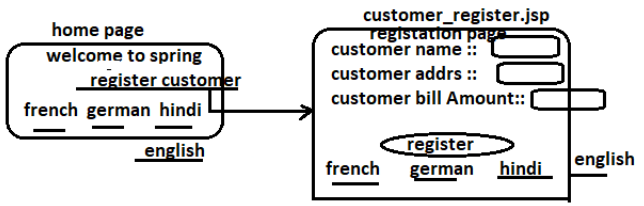
eg: google home page , gmail inbox page , youtube , facebook,amazon and etc..

=> By adding I18n support we can sell our software products to more clients .. if they are web applications then we can attract more customer belonging different coutries and localaties.

=> For presentation labels of I18n we need to multiple properties files for multiple locales on 1 per Locale basis

App.properties (base properties file -- english lables)
App_fr_FR.properties (for french --> french labels)
App_de_DE.properties (for german --> german labels)
App_hi_IN.properties (for hindi --> hindi labels)
and etc..

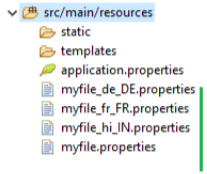
All these properties file must have same key but different values collected from google translator



step1) create spring starter project adding the following dependencies

a) lombok b) spring web c) jstl

step2) prepare multiple properties for multiple locales as shown above having same keys with different values collected from google translator.



for I18n

myfile.properties

```
#Base file (english lables)
home.title=Welcome to Spring boot MVC
home.link= register customer

cust.registration.title=Customer Registration Page
cust.registration.name=Customer name
cust.registration.addrs=Customer Address
cust.registration.billAmt=Customer BillAmount
cust.btn.register= register
```

myfile_hi_IN.properties

```
#Hindi Hocale (hindi lables)
home.title= \u090F\u092E\u0935\u0940\u0938\u0940 \u092E\u0947\u0902 \u0906\u092A\u0915\u093E \u0938\u094D\u0935\u093E\u0917\u0924 \u0939\u0948
home.link= \u0917\u094D\u0930\u093E\u0939\u0915 \u092A\u0902\u091C\u0940\u0915\u0943\u0924 \u0915\u0930\u0947\u0902

cust.registration.title=\u0917\u094D\u0930\u093E\u0939\u0915 \u092A\u0902\u091C\u0940\u0915\u0943\u0924 \u092A\u0943\u0937\u094D\u0920
cust.registration.name=\u0917\u094D\u0930\u093E\u0939\u0915 \u0915\u093E \u0928\u093E\u092E
cust.registration.addrs=\u0917\u094D\u0930\u093E\u0939\u0915 \u0915\u093E \u092A\u0924\u093E
cust.registration.billAmt=\u0917\u094D\u0930\u093E\u0939\u0915 \u092C\u093F\u0932 \u0930\u093E\u0936\u093F
cust.btn.register=\u092A\u0902\u091C\u0940\u0915\u0943\u0924 \u0915\u0930\u0947\u0902
```

myfile_de_DE.properties

```
#german Locale
home.title=Willkommen bei Spring Boot mvc
home.link= Kunde registrieren

cust.registration.title=Kundenregistrierungsseite
cust.registration.name=Kundenname
cust.registration.addrs=Rechnungsbetrag des Kunden
cust.registration.billAmt=Montant de la facture client
cust.btn.register=registrieren
```

myfile_fr_FR.properties

```
#French Locale (french lables)
home.title=Bienvenue sur Spring Boot MVC
home.link= enregistrer le client

cust.registration.title=Page d'enregistrement client
cust.registration.name=Nom du client
cust.registration.addrs=Adresse du client
cust.registration.billAmt=Rechnungsbetrag des Kunden
cust.btn.register=S'inscrire
```

step3) Cfg base properties file in application.properties

In application.properties

```
# configure base properties file
spring.messages.basename=myfile
```

step3) Activate I18n in spring boot MVC Application by configuring SessionLocaleResolver as the spring bean

In main class

```
@Bean(name="localeResolver") //fixed bean id
public SessionLocaleResolver createSLResolver() {
```

Resolvers are given to activate

```
        SessionLocaleResolver resolver=new SessionLocaleResolver();
        resolver.setDefaultLocale(new Locale("en","US"));
        return resolver;
    }
```

The moment this spring bean class obj is created.. it makes the underlying spring mvc or spring boot mvc app to activate the I18n on the application.

certain facility in spring mvc or spring boot mvc appliation which will not automatically come

eg: TilesResolver (To activate tile framework)
SessionLocaleResolver (To activate I18n)
CosMultipartResolver (To activate file uploading)

step4) Configure LocalChangeInterceptor as spring bean in main class using @Bean method

This interceptor takes the locale value from specified request param and changes locale of every request by trapping the request.

```
@Bean
public LocaleChangeInterceptor createLCInterceptor() {
    LocaleChangeInterceptor interceptor=new LocaleChangeInterceptor();
    interceptor.setParamName("lang");   //default is locale
    return interceptor;
}
```

=>This Interceptor that allows for changing the current locale on every request, via a configurable request parameter (default parameter name: "locale").

step5) Develop Custom Configiurer class as spring bean to register the above interceptor with InterceptorRegistry

```
//MyWebMVCConfigurer.java
package com.nt.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.i18n.LocaleChangeInterceptor;

@Component
public class MyWebMVCConfigurer implements WebMvcConfigurer {
    @Autowired
    private LocaleChangeInterceptor interceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
      registry.addInterceptor(interceptor);
    }

}
```

The interceptors in spring boot mvc web application will be activated only after registering with InterceptorRegistry

=>In servlet ,jsp based web application we take the support servlet filters to add pre-request processing and post response generation logics

=>In Spring MVC,spring boot MVC web applications we take the support of Handler interfaceptors to pre ,post after completion logics.

==========

I18n in spring boot MVC application

==========

I18n :: Internationlization (I 18 letters n)

=> Making our app working for different Locales is called enabling I18n on the application

=> Locale means language + country

eg:: en-US (english as it speaks in USA)
fr-FR (french as it speaks in France)
hi -IN (hindi as it speaks in India)
fr- CA (french as it speaks in Canada)
and etc..

=>By enabling I18n on the web application we can change the fllowing presentations according to the Locale that is chosen

a) Presentation Lables
b) Date patterns
c) Time patterns
d) Number formats
e) Currency Symbols
f) Content indentation (most of languages left to right
but urdu,arabic and etc.. right to left)
and etc..

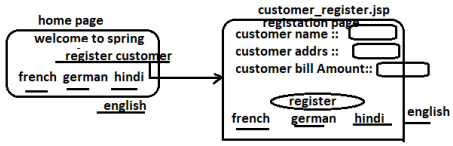
eg: google home page , gmail inbox page , youtube , facebook,amazon and etc..

=> By adding I18n support we can sell our software products to more clients .. if they are web applications then we can attract more customer belonging different coutries and localaties.

=> For presentation labels of I18n we need to multiple properties files for multiple locales on 1 per Locale basis

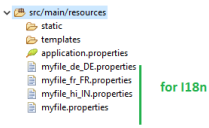
App.properties (base properties file -- english lables)
App_fr_FR.properties (for french --> french labels)
App_de_DE.properties (for german --> german labels)
App_hi_IN.properties (for hindi --> hindi labels)
and etc..

All these properties file must have same key but different values collected from google translator



step1) create spring starter project adding the following dependencies
a) lombok b) spring web c) jstl

step2) prepare multiple properties for multiple locales as shown above having same keys with different values collected from google translator.



for I18n

myfile.properties

```
#Base file (english lables)
home.title=Welcome to Spring boot MVC
home.link= register customer

cust.registration.title=Customer Registration Page
cust.registration.name=Customer name
cust.registration.addrs=Customer Address
cust.registration.billAmt=Customer BillAmount
cust.btn.register= register
```

myfile_hi_IN.properties

```
#Hindi Locale (hindi lables)
home.title= \u090F\u092E\u0935\u0940\u0938\u0940 \u092E\u0947\u0902 \u0906\u092A\u0915\u093E \u0938\u094D\u0935\u093E\u0917\u0924 \u0939\u0948
home.link= \u0917\u094D\u0930\u093E\u0939\u0915 \u092A\u0902\u091C\u0940\u0915\u0943\u0924 \u0915\u0930\u0947\u0902

cust.registration.title=\u0917\u094D\u0930\u093E\u0939\u0915 \u092A\u0902\u091C\u0940\u0915\u0943\u0924\u093F \u092A\u0943\u0937\u094D\u0920
cust.registration.name=\u0917\u094D\u0930\u093E\u0939\u0915 \u0915\u093E \u0928\u093E\u092E
cust.registration.addrs=\u0917\u094D\u0930\u093E\u0939\u0915 \u0915\u093E \u092A\u0924\u093E
cust.registration.billAmt=\u0917\u094D\u0930\u093E\u0939\u0915 \u092C\u093F\u0932 \u0930\u093E\u0936\u093F
cust.btn.register=\u092A\u0902\u091C\u0940\u0915\u0943\u0924 \u0915\u0930\u0947\u0902
```

myfile_de_DE.properties

```
#german Locale
home.title=Willkommen bei Spring Boot mvc
home.link= Kunde registrieren

cust.registration.title=Kundenregistrierungsseite
cust.registration.name=Kundenname
cust.registration.addrs=Rechnungsbetrag des Kunden
cust.registration.billAmt=Montant de la facture client
cust.btn.register=registrieren
```

myfile_fr_FR.properties

```
#French Locale (french lables)
home.title=Bienvenue sur Spring Boot MVC
home.link= enregistrer le client

cust.registration.title=Page d'enregistrement client
cust.registration.name=Nom du client
cust.registration.addrs=Adresse du client
cust.registration.billAmt=Rechnungsbetrag des Kunden
cust.btn.register=S'inscrire
```

step3) Cfg base properties file in application.properties

In application.properties

```
# configure base properties file
spring.messages.basename=myfile
```

step3) Activate I18n in spring boot MVC Application by configuring SessionLocaleResolver as the spring bean

In main class

```
@Bean(name="localeResolver") //fixed bean id
public SessionLocaleResolver createSLResolver() {

    SessionLocaleResolver resolver=new SessionLocaleResolver();
    resolver.setDefaultLocale(new Locale("en","US"));
    return resolver;
}
```

The moment this spring bean class obj is created.. it makes the underlying spring mvc or spring boot mvc app to activate the I18n on the application.

Resolvers are given to activate certain facility in spring mvc or spring boot mvc appliation which will not automatically come

eg: TilesResolver (To activate tile framework)
SessionLocaleResolver (To activate I18n)
CosMultipartResolver (To activate file uploading)

step4) Configure LocalChangeInterceptor as spring bean in main class using @Bean method

This interceptor takes the locale value from specified request param and changes locale of every request by trapping the request.

```
@Bean
public LocaleChangeInterceptor createLCInterceptor() {
    LocaleChangeInterceptor interceptor=new LocaleChangeInterceptor();
    interceptor.setParamName("lang");  //default is locale
    return interceptor;
}
```

=>This Interceptor that allows for changing the current locale on every request, via a configurable request parameter (default parameter name: "locale").

step5) Develop Custom Configiurer class as spring bean to register the above interceptor with InterceptorRegistry

```
//MyWebMVCConfigurer.java
package com.nt.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.i18n.LocaleChangeInterceptor;

@Component
public class MyWebMVCConfigurer implements WebMvcConfigurer {
    @Autowired
    private LocaleChangeInterceptor interceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
      registry.addInterceptor(interceptor);
    }

}
```

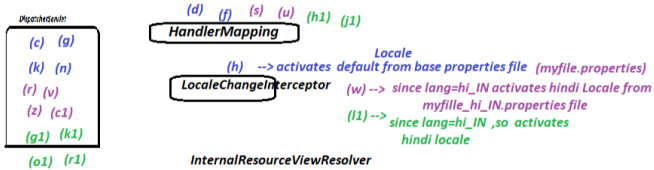
The interceptors in spring boot mvc web application will be activated only after registering with InterceptorRegistry

step6) Develop the Model class

```
package com.nt.model;
import lombok.Data;
@Data
public class Customer {
    private Integer cno;
    private String cname;
    private String caddrs;
    private Float billAmount;
}
```

step7) develop the Controller class having handler methods to lanunch home pag, form page

```
//controller class
package com.nt.controller;
import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;

import com.nt.model.Customer;

@Controller
public class CustomerOperationsController {

  @GetMapping("/")
  public String showHome() {
      //return LVN
```



```
        return "welcome";  (k)
    }

    @GetMapping("/register")                    (i1?)
(m1)public String showCustomerFormPage(@ModelAttribute("cust") Customer cust,
                                       Map<String,Object> map) {
        //return LVN
        return "customer_register";   (n1)


    }

}
```

|-->prefix: /WEB-INF/pages/
|--->suffix : .jsp (l) (a1) (p1)

(m) (b1)
/WEB-INF/pages/welcome.jsp
/WEB-INF/pages/customer_register.jsp
(q1)

step8) Develop jsp pages enabling UTF-8 content type and reading locale specific messges from properties files.

WEB-INF/pages/welcome.jsp (o) (d1)

```
<%@page isELIgnored="false" contentType="text/html; charset=UTF-8" %>
<%@taglib uri="http://www.springframework.org/tags"  prefix="sp"%>

<h1 style="color:blue;text-align:center"> <sp:message code="home.title"/></h1>
<br><br>
 <a href="register"><h2 style="color:red;text-align:center"><sp:message code="home.link"/></h2> </a>
                                                    (f1)
 <br><br>

 <p align="center">
   <a href="?lang=fr_FR">French</a>    
   <a href="?lang=de_DE">German</a>    
   <a href="?lang=hi_IN">  हिन्दी (q) </a>    
   <a href="?lang=en_US">English</a>

 </p>
```

(p) collects messages from default properties file (english labels)

(e1) collects the messages from myfile_hi_IN.properties file

=>if <a> tag href not having url or href itself not placed then the generated request goes to that url using which the web page is launched.

WEB-INF/pages/customer_register.jsp (s1)

```
<%@ page isELIgnored="false"  contentType="text/html; charset=UTF-8" %>
<%@taglib uri="http://www.springframework.org/tags/form"  prefix="form"%>
<%@taglib uri="http://www.springframework.org/tags"  prefix="sp"%>

 <h1 style="color:red;text-align:center"><sp:message code="cust.registration.title"/></h1>

 <form:form modelAttribute="cust">
  <table border="1" align="center" bgcolor="cyan">
    <tr>
      <td><sp:message code="cust.registration.name"/> </td>
      <td><form:input path="cname"/> </td>
    </tr>
    <tr>
      <td><sp:message code="cust.registration.addrs"/> </td>
      <td><form:input path="caddrs"/> </td>
    </tr>
    <tr>
      <td><sp:message code="cust.registration.billAmt"/> </td>
      <td><form:input path="billAmount"/> </td>
    </tr>
    <tr>
      <td><input type="submit"  value="<sp:message code="cust.btn.register"/>"/> </td>
    </tr>
  </table>
 </form:form>

  <br><br>

 <p align="center">
   <a href="?lang=fr_FR">French</a>    
   <a href="?lang=de_DE">German</a>    
   <a href="?lang=hi_IN">  हिन्दी  </a>    
   <a href="?lang=en_US">English</a>

 </p>
```

(t1)
collects the message from myfile_hi_IN.properties file

(a)During the deployment of web application all singleton scope spring beans like SessionLocaleResolver, LocaleChangeInterceptor , CustomerOperationsController , MyWebMVCConfigurer classes will be pre-instantiated and the the injections takes place

=>SessionLocaleResolver obj creation activates the I18n

=>LocaleChnageInterceptor will be activated becoz it is registered with InterceptorRegistry

(b) http://localhost:2020/BootMVCProj15-I18n

Applying Date ,time , number ,currency formts in the I18n App

=>We can use JSTL core, formatting tag libraries together to format date,time, number and currency values as show below.

step1) add JSTL libaries to build path

```
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

step2) Import JSTL core , formatting tag libaries

welcome.jsp

```
<%@page isELIgnored="false" contentType="text/html; charset=UTF-8" %>
<%@taglib uri="http://www.springframework.org/tags"  prefix="sp"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt"  prefix="fmt" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core"  prefix="c" %>

<h1 style="color:blue;text-align:center"> <sp:message code="home.title"/></h1>
<br><br>
<a href="register"><h2 style="color:red;text-align:center"><sp:message code="home.link"/></h2> </a>
<br>

<h1>Current active Locale is :: ${pageContext.response.locale} </h1>
<fmt:setLocale value="${pageContext.response.locale}"/>

<jsp:useBean id="dt" class="java.util.Date"/>
<fmt:formatDate var="fdt" value="${dt}" type="date" dateStyle="SHORT" />
<b>formatted date :: ${fdt} </b><br>
<fmt:formatDate var="ftime" value="${dt}" type="time" timeStyle="FULL" />
<b>formatted time :: ${ftime} </b>

<fmt:formatNumber var="fnumber" value="1000000" type="number"/> <br>
<b>formatted number :: ${fnumber}</b>

<fmt:formatNumber var="fcurrency" value="1000000" type="currency"/> <br>
<b>formatted currency :: ${fcurrency}</b>

<fmt:formatNumber var="fpercentage" value="0.211" type="PERCENT"/> <br>
<b>formatted percentage :: ${fpercentage}</b>
```

```
<br><br>

<p align="center">
  <a href="?lang=fr_FR">French</a>    
  <a href="?lang=de_DE">German</a>    
  <a href="?lang=hi_IN">□□□□□□</a>    
  <a href="?lang=en_US">English</a>

</p>
```

=> pageContext,response are implicit objs from 9 implcit objs provided by the jsp page
=> ${pageContet.response.locale} gives curreent active locale.
x

The LocaleInterceptor keeps
the activate Locale in the "locale"
property of "response" obj .From theere
this EL expression gets the current active locale.

File Uploading and File downloading spring boot MVC
==========

Microservices arch impl in spring = spring boot + spring rest/web + spring clould

Spring Rest (logical name) =spring web++

File uploading

The process of selecting files from Client machien file sytem and sending them server file system
is file uploading

n
File dowloanding

=>The getting server machine file system files content to client machine file system is called
file donwloading

usecases:: job portal apps, matrimony apps, social working netwokings apps, video sharing apps,
profile mgmt apps, our classcontent sharing and etc..

=>Only in standalone apps , the files content (LOBs like CLOB,BLOB) will be stored directly db table
cols... in other apps like distributed apps, web applications and etc.. the LOBs (files) will saved
in server machine file system or in special cloud softwares like EDN (Eletronic document network),
s3Bucket(given by aws) ,DMS(document mangement system), cloud front ,google bucket and etc..
but the ticket/tocken or path of file will be saved in db table cols as String values

some link to cloud storage

link to file system

File Uploading and File downloading spring boot MVC
==========

Microservices arch impl in spring = spring boot + spring rest/web + spring clould

Spring Rest (logical name) =spring web++

File uploading

The process of selecting files from Client machien file sytem and sending them server file system is file uploading

n

File dowloanding

=>The getting server machine file system files content to client machine file system is called file donwloading

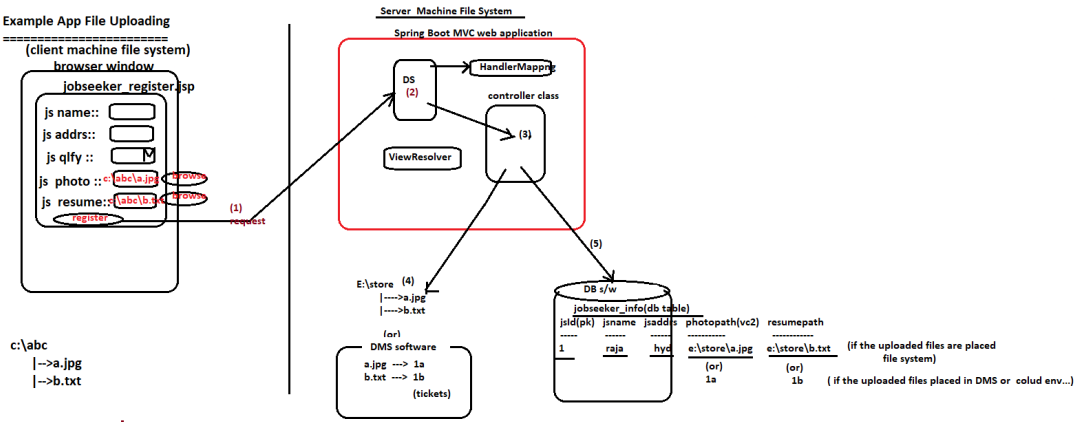
usecases:: job portal apps, matrimony apps, social working netwokings apps, video sharing apps, profile mgmt apps, our classcontent sharing and etc..

=>Only in standalone apps , the files content (LOBs like CLOB,BLOB) will be stored directly db table cols... in other apps like distributed apps, web applications and etc.. the LOBs (files) will saved in server machine file system or in special cloud softwares like EDN (Eletronic document network), s3Bucket(given by aws) ,DMS(document mangement system), cloud front ,google bucket and etc.. but the ticket/tocken or path of file will be saved in db table cols as String values

some link to cloud storage

link to file system

Example App File Uploading
==========



=> To represent the content of uploaded files of form page in Model class obj , we take the support of MultipartFile type properties in Model class .. which are input streams representing the content of the uploaded files .. So we can take the support of outstreams to write the content to server machine file system

step1) create spring starter Project adding spring web , jstl , spring data jpa , lombok, common-io , ojdbc8 , tomcat-embedded jasper jar files.

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
	<groupId>com.oracle.database.jdbc</groupId>
	<artifactId>ojdbc8</artifactId>
	<scope>runtime</scope>
</dependency>
<dependency>
	<groupId>org.projectlombok</groupId>
	<artifactId>lombok</artifactId>
	<optional>true</optional>
</dependency>
```

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-tomcat</artifactId>
	<scope>provided</scope>
</dependency>
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-test</artifactId>
	<scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
	<!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-jasper -->
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.11.0</version>
</dependency>
```

step2) develop the following model class for Persistence

(Entity class for Persistence)

```
//JobSeekerInfo.java
package com.nt.entity;

import java.io.Serializable;
```

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Entity
@Table(name="BOOT_JS_INFO")
@Data
public class JobSeekerInfo implements Serializable {
	@Id
	@GeneratedValue
	private Integer jsId;
	@Column(length = 15)
	private String jsName;
	@Column(length = 15)
	private String jsAddrs;
	@Column(length = 20)
	private  String resumePath;
	@Column(length = 20)
	private  String photoPath;


}
```

step3) Develop seperate Model class for formPage data binding

```
package com.nt.entity;

import java.io.Serializable;

import org.springframework.web.multipart.MultipartFile;

import lombok.Data;


@Data
public class JobSeekerData implements Serializable {
	private Integer jsId;
	private String jsName;
	private String jsAddrs;
	private  MultipartFile resume;  //Input stream to hold content of uploded file
	private  MultipartFile photo;


}
```

step4) Develop the Repository Interface

```
package com.nt.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.nt.entity.JobSeekerInfo;

public interface IJobSeekerRepo extends JpaRepository<JobSeekerInfo, Integer> {


}
```

step5) Develope service interface and service Impl class

```
service Interface
---------------------
public interface IJobSeekerMgmtService {
    public  String registerJobSeeker(JobSeekerInfo info);
}

service impl class
---------------------------
@Service
public class JobSeekerMgmtServiceImpl implements IJobSeekerMgmtService {
	@Autowired
	private IJobSeekerRepo  jsRepo;

	@Override
	public String registerJobSeeker(JobSeekerInfo info) {

		return "Job seeker is saved with "+jsRepo.save(info).getJsId()+"  id value";
	}

}
```

step6) add entries in application.properties file having datasource cfg and jpa-hibernate entries

In application.properties

```
#Datasource cfg,jpa-hibernate cfg
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
#based in these details the HikariCP DataSource object will be created pointing to
#jdbc con pool for oracle as part autoconfiguration activity

#JPA-hibernate cfgs
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

step7) Cfg ViewResolver in application.properties file

In application.properites

```
#view Resolver cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

step7) add entries for embedded Tomcat server

```
#For Embedded server
server.port=4041
server.servlet.context-path=/JobSeekersApp
```

step8) Cfg CommonsMultipartResolver in main class as spring bean using @Bean method to activate file upload mechasim in spring mvc or spring boot mvc application,

```
@Bean(name="multipartResolver")   fixed bean id
public CommonsMultipartResolver createCMResolver() {
        CommonsMultipartResolver resolver=new CommonsMultipartResolver();
        resolver.setMaxUploadSizePerFile(50*1024*1024);
        resolver.setMaxUploadSize(-1);  //all files togather how much size is allowed  -1 indicates no limit
        return resolver;
}
```

=>Resolvers are internally used to activate certain certain facilities in spring boot MVC application.. They are internally gathered by calling ctx.getBean(-) method with fixed bean ids .. So we must provide same bean ids while cfg them

For the following spring beans cfg , we need the fixed bean ids

class	fixed bean id
ResourceBundleMessageSource ---------->	messageSource ,
SessionLocaleResolver ---------->	localeResolver
CommonsMultiPartResolver ---------->	multipartResolver

step9)
Develop handler method in controller class to take implicit reuquest to return LVN for home page

```
@Controller
public class JobSeekerOperationsController {
      @Autowired
      private IJobSeekerMgmtService service;

      @GetMapping("/")
      public  String showHomePage() {
            return  "welcome";
      }

}
```

WEB-INF/pages/welcome.jsp

```
<%@ page  isELIgnored="false" %>


<h1 style="text-align:center"><a href="register">Register JobSeeker</a></h1>


<h1 style="text-align:center"><a href="list_js">List Jobseekers</a></h1>
                                        (For future  file downloading operation)
```

step10) place handler method in controller class to show Jobseeke registration form page

In controller class

```
@GetMapping("/register")
      public   String  showJSRegistrationForm(@ModelAttribute("js") JobSeekerData jsData) {
            //return LVN
            return "jobseeker_register";
      }
```

File Uploading and File downloading spring boot MVC
==========

Microservices arch impl in spring = spring boot + spring rest/web + spring clould

Spring Rest (logical name) =spring web++

File uploading

The process of selecting files from Client machien file sytem and sending them server file system is file uploading

n

File dowloanding

=>The getting server machine file system files content to client machine file system is called file donwloading

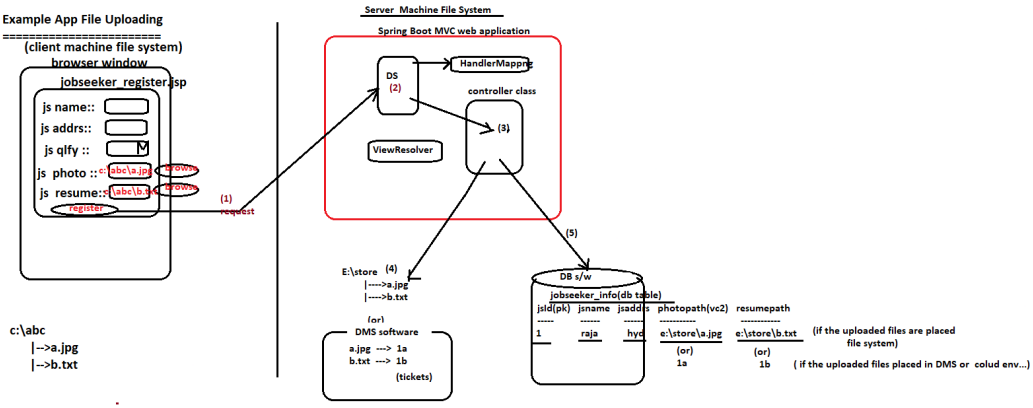
usecases:: job portal apps, matrimony apps, social working netwokings apps, video sharing apps, profile mgmt apps, our classcontent sharing and etc..

=>Only in standalone apps , the files content (LOBs like CLOB,BLOB) will be stored directly db table cols... in other apps like distributed apps, web applications and etc.. the LOBs (files) will saved in server machine file system or in special cloud softwares like EDN (Eletronic document network), s3Bucket(given by aws) ,DMS(document mangement system), cloud front ,google bucket and etc.. but the ticket/tocken or path of file will be saved in db table cols as String values

some link to cloud storage

link to file system

Example App File Uploading
==========



=> To represent the content of uploaded files of form page in Model class obj , we take the support of MultipartFile type properties in Model class .. which are input streams representing the content of the uploaded files .. So we can take the support of outstreams to write the content to server machine file system

step1) create spring starter Project adding spring web , jstl , spring data jpa , lombok, common-io , ojdbc8 , tomcat-embedded jasper jar files.

```
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-jpa</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
	</dependency>

	<dependency>
		<groupId>com.oracle.database.jdbc</groupId>
		<artifactId>ojdbc8</artifactId>
		<scope>runtime</scope>
	</dependency>
	<dependency>
		<groupId>org.projectlombok</groupId>
		<artifactId>lombok</artifactId>
		<optional>true</optional>
	</dependency>
```

```
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-tomcat</artifactId>
		<scope>provided</scope>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-test</artifactId>
		<scope>test</scope>
	</dependency>

	<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
	<!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-jasper -->
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.11.0</version>
</dependency>
```

step2) develop the following model class for Persistence (Entity class for Persistence)

```
//JobSeekerInfo.java
package com.nt.entity;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Entity
```

```
@Table(name="BOOT_JS_INFO")
@Data
public class JobSeekerInfo implements Serializable {
	@Id
	@GeneratedValue
	private Integer jsId;
	@Column(length = 15)
	private String jsName;
	@Column(length = 15)
	private String jsAddrs;
	@Column(length = 20)
	private  String resumePath;
	@Column(length = 20)
	private  String photoPath;


}
```

step3) Develop seperate Model class for formPage data binding

```
package com.nt.entity;

import java.io.Serializable;

import org.springframework.web.multipart.MultipartFile;

import lombok.Data;


@Data
public class JobSeekerData implements Serializable {
	private Integer jsId;
	private String jsName;
	private String jsAddrs;
	private  MultipartFile resume;  //Input stream to hold content of uploded file
	private  MultipartFile photo;


}
```

step4) Develop the Repository Interface

```
package com.nt.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.nt.entity.JobSeekerInfo;

public interface IJobSeekerRepo extends JpaRepository<JobSeekerInfo, Integer> {


}
```

step5) Develope service interface and service Impl class

service Interface

```
public interface IJobSeekerMgmtService {
    public  String registerJobSeeker(JobSeekerInfo info);
}
```

service impl class

```
@Service
public class JobSeekerMgmtServiceImpl implements IJobSeekerMgmtService {
	@Autowired
	private IJobSeekerRepo  jsRepo;

	@Override
	public String registerJobSeeker(JobSeekerInfo info) {

		return "Job seeker is saved with "+jsRepo.save(info).getJsId()+"  id value";
	}

}
```

step6) add entries in application.properties file having datasource cfg and jpa-hibernate entries

In application.properties

```
#Datasource cfg,jpa-hibernate cfg
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
#based in these details the HikariCP DataSource object will be created pointing to
#jdbc con pool for oracle as part autoconfiguration activity

#JPA-hibernate cfgs
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

step7) Cfg ViewResolver in application.properties file

In application.properites

```
```

```
#view Resolver cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

step7) add entries for embedded Tomcat server

```
#For Embedded server
server.port=4041
server.servlet.context-path=/JobSeekersApp
```

step8) Cfg CommonsMultipartResolver in main class as spring bean using @Bean method to activate file upload mechasim in spring mvc or spring boot mvc application,

```
@Bean(name="multipartResolver")          fixed bean id
public CommonsMultipartResolver createCMResolver() {
        CommonsMultipartResolver resolver=new CommonsMultipartResolver();
        resolver.setMaxUploadSizePerFile(50*1024*1024);
        resolver.setMaxUploadSize(-1);  //all files togather how much size is allowed  -1 indicates no limit
        return resolver;
}
```

=>Resolvers are internally used to activate certain certain facilities in spring boot MVC application.. They are internally gathered by calling ctx.getBean(-) method with fixed bean ids .. So we must provide same bean ids while cfg them

For the following spring beans cfg , we need the fixed bean ids

class	fixed bean id
ResourceBundleMessageSource ---------->	messageSource
SessionLocaleResolver ---------->	localeResolver
CommonsMultiPartResolver ---------->	multipartResolver

step9)
Develop handler method in controller class to take implicit reuquest to return LVN for home page

```
@Controller
public class JobSeekerOperationsController {
      @Autowired
      private IJobSeekerMgmtService service;

      @GetMapping("/")
      public  String showHomePage() {
            return  "welcome";
      }

}
```

WEB-INF/pages/welcome.jsp

```
<%@ page  isELIgnored="false" %>


<h1 style="text-align:center"><a href="register">Register JobSeeker</a></h1>


<h1 style="text-align:center"><a href="list_js">List Jobseekers</a></h1>
                                        (For future  file downloading operation)
```

step10) place handler method in controller class to show Jobseeke registration form page

In controller class

```
@GetMapping("/register")
      public   String  showJSRegistrationForm(@ModelAttribute("js") JobSeekerData jsData) {
            //return LVN
            return "jobseeker_register";
      }
```

step 11) Develop form page

jobseeker_register.jsp

```
<%@ page language="java"  isELIgnored="false" %>
<%@taglib uri="http://www.springframework.org/tags/form"  prefix="frm"%>


<frm:form modelAttribute="js" enctype="multipart/form-data">
  <table border="0"  bgcolor="cyan" align="center">
    <tr>
      <td> Name :: </td>
      <td><frm:input path="jsName"/>  </td>
    </tr>
    <tr>
      <td> Address :: </td>
      <td><frm:input path="jsAddrs"/>  </td>
    </tr>
    <tr>
      <td> Select Resume :: </td>
      <td><frm:input type="file" path="resume"/>  </td>
    </tr>
    <tr>
      <td> Select  Photo :: </td>
      <td><frm:input type="file" path="photo"/>  </td>
    </tr>
    <tr>
      <td colspan="2"><input type="submit" value="register">  </td>
    </tr>
  </table>
</frm:form>
```

File Uploading and File downloading spring boot MVC
==========

Microservices arch impl in spring = spring boot + spring rest/web + spring clould

Spring Rest (logical name) =spring web++

File uploading

The process of selecting files from Client machien file sytem and sending them server file system is file uploading

n

File dowloanding

=>The getting server machine file system files content to client machine file system is called file donwloading

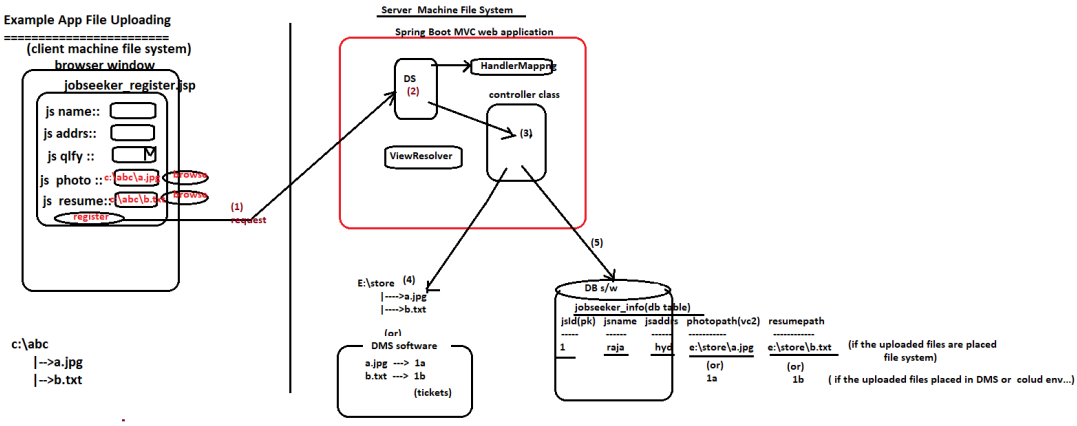
usecases:: job portal apps, matrimony apps, social working netwokings apps, video sharing apps, profile mgmt apps, our classcontent sharing and etc..

=>Only in standalone apps , the files content (LOBs like CLOB,BLOB) will be stored directly db table cols... in other apps like distributed apps, web applications and etc.. the LOBs (files) will saved in server machine file system or in special cloud softwares like EDN (Eletronic document network), s3Bucket(given by aws) ,DMS(document mangement system), cloud front ,google bucket and etc.. but the ticket/tocken or path of file will be saved in db table cols as String values

some link to cloud storage

link to file system

Example App File Uploading
==========



=> To represent the content of uploaded files of form page in Model class obj , we take the support of MultipartFile type properties in Model class .. which are input streams representing the content of the uploaded files .. So we can take the support of outstreams to write the content to server machine file system

step1) create spring starter Project adding spring web , jstl , spring data jpa , lombok, common-io , ojdbc8 , tomcat-embedded jasper jar files.

```
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-jpa</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
	</dependency>

	<dependency>
		<groupId>com.oracle.database.jdbc</groupId>
		<artifactId>ojdbc8</artifactId>
		<scope>runtime</scope>
	</dependency>
	<dependency>
		<groupId>org.projectlombok</groupId>
		<artifactId>lombok</artifactId>
		<optional>true</optional>
	</dependency>
```

```
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-tomcat</artifactId>
		<scope>provided</scope>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-test</artifactId>
		<scope>test</scope>
	</dependency>

	<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
	<!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-jasper -->
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.11.0</version>
</dependency>
```

step2) develop the following model class for Persistence

(Entity class for Persistence)

```
//JobSeekerInfo.java
package com.nt.entity;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;
```

```
@Entity
@Table(name="BOOT_JS_INFO")
@Data
public class JobSeekerInfo implements Serializable {
	@Id
	@GeneratedValue
	private Integer jsId;
	@Column(length = 15)
	private String jsName;
	@Column(length = 15)
	private String jsAddrs;
	@Column(length = 20)
	private  String resumePath;
	@Column(length = 20)
	private  String photoPath;


}
```

step3) Develop seperate Model class for formPage data binding

```
package com.nt.entity;

import java.io.Serializable;

import org.springframework.web.multipart.MultipartFile;

import lombok.Data;


@Data
public class JobSeekerData implements Serializable {
	private Integer jsId;
	private String jsName;
	private String jsAddrs;
	private  MultipartFile resume;  //Input stream to hold content of uploded file
	private  MultipartFile photo;


}
```

step4) Develop the Repository Interface

```
package com.nt.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.nt.entity.JobSeekerInfo;

public interface IJobSeekerRepo extends JpaRepository<JobSeekerInfo, Integer> {


}
```

step5) Develope service interface and service Impl class

service Interface

```
public interface IJobSeekerMgmtService {
    public  String registerJobSeeker(JobSeekerInfo info);
}
```

service impl class

```
@Service
public class JobSeekerMgmtServiceImpl implements IJobSeekerMgmtService {
	@Autowired
	private IJobSeekerRepo  jsRepo;

	@Override
	public String registerJobSeeker(JobSeekerInfo info) {

		return "Job seeker is saved with "+jsRepo.save(info).getJsId()+"  id value";
	}

}
```

step6) add entries in application.properties file having datasource cfg and jpa-hibernate entries

In application.properties

```
#Datasource cfg,jpa-hibernate cfg
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
#based in these details the HikariCP DataSource object will be created pointing to
#jdbc con pool for oracle as part autoconfiguration activity

#JPA-hibernate cfgs
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

step7) Cfg ViewResolver in application.properties file

In application.properites

```
#view Resolver cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

step7) add entries for embedded Tomcat server

```
#For Embedded server
server.port=4041
server.servlet.context-path=/JobSeekersApp
```

step8) Cfg CommonsMultipartResolver in main class as spring bean using @Bean method to activate file upload mechasim in spring mvc or spring boot mvc application,

```
@Bean(name="multipartResolver")   fixed bean id
public CommonsMultipartResolver createCMResolver() {
        CommonsMultipartResolver resolver=new CommonsMultipartResolver();
        resolver.setMaxUploadSizePerFile(50*1024*1024);
        resolver.setMaxUploadSize(-1);  //all files togather how much size is allowed  -1 indicates no limit
        return resolver;
}
```

=>Resolvers are internally used to activate certain certain facilities in spring boot MVC application.. They are internally gathered by calling ctx.getBean(-) method with fixed bean ids .. So we must provide same bean ids while cfg them

For the following spring beans cfg , we need the fixed bean ids

class	fixed bean id
ResourceBundleMessageSource ---------->	messageSource
SessionLocaleResolver ---------->	localeResolver
CommonsMultiPartResolver ---------->	multipartResolver

step9)
Develop handler method in controller class to take implicit reuquest to return LVN for home page

```
@Controller
public class JobSeekerOperationsController {
      @Autowired
      private IJobSeekerMgmtService service;

      @GetMapping("/")
      public  String showHomePage() {
            return  "welcome";
      }

}
```

WEB-INF/pages/welcome.jsp

```
<%@ page  isELIgnored="false" %>


<h1 style="text-align:center"><a href="register">Register JobSeeker</a></h1>


<h1 style="text-align:center"><a href="list_js">List Jobseekers</a></h1>
                                        (For future  file downloading operation)
```

step10) place handler method in controller class to show Jobseeke registration form page

In controller class

```
@GetMapping("/register")
      public   String  showJSRegistrationForm(@ModelAttribute("js") JobSeekerData jsData) {
            //return LVN
            return "jobseeker_register";
      }
```

step 11) Develop form page

jobseeker_register.jsp

```
<%@ page language="java"  isELIgnored="false" %>
<%@taglib uri="http://www.springframework.org/tags/form"  prefix="frm"%>


<frm:form modelAttribute="js" enctype="multipart/form-data">
  <table border="0"  bgcolor="cyan" align="center">
    <tr>
      <td> Name :: </td>
      <td><frm:input path="jsName"/>  </td>
    </tr>
    <tr>
      <td> Address :: </td>
      <td><frm:input path="jsAddrs"/>  </td>
    </tr>
    <tr>
      <td> Select Resume :: </td>
      <td><frm:input type="file" path="resume"/>  </td>
    </tr>
    <tr>
      <td> Select  Photo :: </td>
      <td><frm:input type="file" path="photo"/>  </td>
    </tr>
    <tr>
      <td colspan="2"><input type="submit" value="register">  </td>
    </tr>
```

```
        </table>
       </frm:form>
```

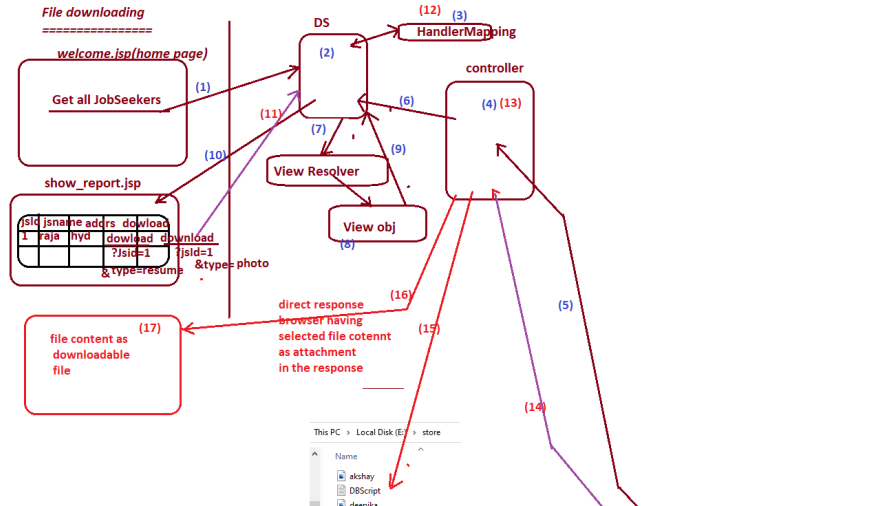
step12) Develop handler method to process the form submission and to complete file upload activity

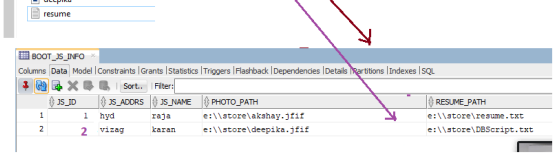
```
    private Environment env;

        @PostMapping("/register")
        public  String  registerJSByUploadingFiles(@ModelAttribute("js") JobSeekerData jsData,
                                                   Map<String,Object> map)throws Exception {
            //get Upload folder location from properties file
            String storeLocation=env.getRequiredProperty("upload.store");
            // if that not avaiable then create it
            File file=new File(storeLocation);
             if(!file.exists())
                    file.mkdir();

             // get InputStreams representin the upload files content
             MultipartFile  resumeFile=jsData.getResume();
             MultipartFile  photoFile=jsData.getPhoto();
             InputStream isResume=resumeFile.getInputStream();
             InputStream isPhoto=photoFile.getInputStream();
             //get the names of the uploaded files
             String resumeFileName=resumeFile.getOriginalFilename();
             String photoFileName=photoFile.getOriginalFilename();
             //create outstreams reprsenting empty destination files
             OutputStream osResume=new FileOutputStream(file.getAbsolutePath()+"\\"+resumeFileName);
             OutputStream osPhoto=new FileOutputStream(file.getAbsolutePath()+"\\"+photoFileName);
             // perform file copy operation
             IOUtils.copy(isResume,osResume);
             IOUtils.copy(isPhoto,osPhoto);
             //close stream
             isResume.close();
             osResume.close();
             isPhoto.close();
             osPhoto.close();
            //prepare  Entity class obj from Model class obj
             JobSeekerInfo jsInfo=new JobSeekerInfo();
              jsInfo.setJsName(jsData.getJsName());
              jsInfo.setJsAddrs(jsData.getJsAddrs());
              jsInfo.setResumePath(file.getAbsolutePath()+"/"+resumeFileName);
              jsInfo.setPhotoPath(file.getAbsolutePath()+"/"+photoFileName);
            //use  Service
              String msg=service.registerJobSeeker(jsInfo);
            //keep the uploaded file names and  location in model attributes
              map.put("file1", resumeFileName);
              map.put("file2", photoFileName);
              map.put("resultMsg",msg);
              //return LVN
            return "show_result";
        }
```

step13) develop the result page

show_result.jsp

```
<%@page isELIgnored="false" %>

<h1 style="color:red;text-align:center">Result page</h1>

<b>
      Result is :: ${resultMsg} <br>
      The uploaded file names are ::  ${file1} , ${file2}
</b>

<br><br>
 <a href="./">home</a>
```





Part1 Development steps (Displaying having donwload hyperlinks for files)

step1) add methods in service Interface and in service Impl class

In service Interface

```
public List<JobSeekerInfo> fetchAllJobseekers();
```

In service Impl class

```
@Override
    public List<JobSeekerInfo> fetchAllJobseekers() {
        return jsRepo.findAll();
    }
```

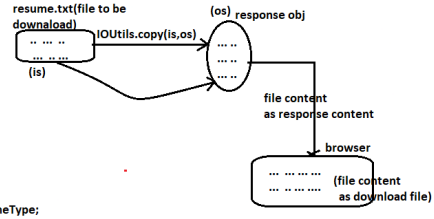
step3) Add the following handler method in controller class

```
@GetMapping("/list_js")
public String showReport(Map<String,Object> map) {
    System.out.println("JobSeekerOperationsController.showReport()");
    //use service
    List<JobSeekerInfo> list=service.fetchAllJobseekers();
    System.out.println(list.size());
    //add result to model attrbutes
    map.put("jsList", list);
    //return LVN
    return "show_report";
}
```

step4) **develope show_report.jsp page in WEB-INF/pages folder as shown below**

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

<c:choose>
  <c:when test="${!empty jsList}">
    <table border="1" align="center" bgcolor="cyan">
      <tr>
        <th>JsId </th><th>JsName </th><th>JsAddrs </th><th>resume </th><th>photo </th>
      </tr>
      <tr>
       <c:forEach var="info" items="${jsList}">
         <tr>
          <td>${info.jsId } </td>
          <td>${info.jsName } </td>
          <td>${info.jsAddrs } </td>
          <td><a href="download?jsId=${info.jsId}&type=resume">download resume</a> </td>
          <td><a href="download?jsId=${info.jsId}&type=photo">download photo</a> </td>
         </tr>
       </c:forEach>
      </tr>
    </table>
  </c:when>
  <c:otherwise>
     <h1 style="color:red;text-align:center">Records not found </h1>
  </c:otherwise>
</c:choose>
```

Standard process for file download activity



a) create File obj having path of the file to be downloaded
```
File file=new File("E:/store/resume.txt");
```
b) Get the length file to be downloaded
```
long length =file.length();
```
c) make file content length as response content length
```
response.setContentLength(length);
```

d) set file MIMEtype as response content type
```
ServletContext sc=req.getServletContext();
String mimeType=sc.getMimeType("E:/store/resume.txt");
mimeType=mimeType==null?"application/octet-stream":mimeType;
res.setContentType(mimeType);
```
generic /universal mimetype

e) create InputStream representing file to be downloded
```
InputStream is=new FileInputStream(file);
```

f) create outputStream representing the response obj
```
ServletOutputStream os=res.getOutputStream();
```

g) Give instruction to browser to make the recieved as the downloadable file
```
res.setHeader("Content-Disposition","attachment;fileName="+file.getName());
```

note:: The default value of "Content-Disposition" header is "inline" i.e the recieved content will be displayed on the browser directly ..where as "attachment" instructs the browser to make recieved response content as the downloadable file with given name.

h) copy File content to response as the response content
```
IOUtils.copy(is,os);
```
i) close streams
```
is.close();
os.close();
```

File Uploading and File downloading spring boot MVC
==========

Microservices arch impl in spring = spring boot + spring rest/web + spring clould

Spring Rest (logical name) =spring web++

File uploading

The process of selecting files from Client machien file sytem and sending them server file system is file uploading

n

File dowloanding

=>The getting server machine file system files content to client machine file system is called file donwloading

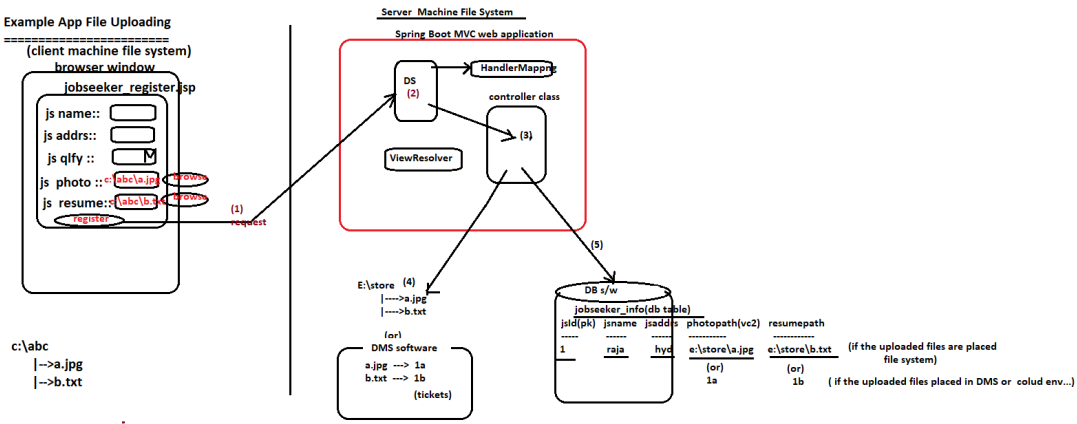
usecases:: job portal apps, matrimony apps, social working netwokings apps, video sharing apps, profile mgmt apps, our classcontent sharing and etc..

=>Only in standalone apps , the files content (LOBs like CLOB,BLOB) will be stored directly db table cols... in other apps like distributed apps, web applications and etc.. the LOBs (files) will saved in server machine file system or in special cloud softwares like EDN (Eletronic document network), s3Bucket(given by aws) ,DMS(document mangement system), cloud front ,google bucket and etc.. but the ticket/tocken or path of file will be saved in db table cols as String values

some link to cloud storage

link to file system

Example App File Uploading
==========



=> To represent the content of uploaded files of form page in Model class obj , we take the support of MultipartFile type properties in Model class .. which are input streams representing the content of the uploaded files .. So we can take the support of outstreams to write the content to server machine file system

step1) create spring starter Project adding spring web , jstl , spring data jpa , lombok, common-io , ojdbc8 , tomcat-embedded jasper jar files.

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
	<groupId>com.oracle.database.jdbc</groupId>
	<artifactId>ojdbc8</artifactId>
	<scope>runtime</scope>
</dependency>
<dependency>
	<groupId>org.projectlombok</groupId>
	<artifactId>lombok</artifactId>
	<optional>true</optional>
</dependency>
```

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-tomcat</artifactId>
	<scope>provided</scope>
</dependency>
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-test</artifactId>
	<scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
	<!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-jasper -->
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.11.0</version>
</dependency>
```

step2) develop the following model class for Persistence

(Entity class for Persistence)

```
//JobSeekerInfo.java
package com.nt.entity;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Entity
@Table(name="BOOT_JS_INFO")
@Data
public class JobSeekerInfo implements Serializable {
	@Id
	@GeneratedValue
	private Integer jsId;
	@Column(length = 15)
	private String jsName;
	@Column(length = 15)
	private String jsAddrs;
	@Column(length = 20)
	private  String resumePath;
	@Column(length = 20)
	private  String photoPath;


}
```

step3) Develop seperate Model class for formPage data binding

```
package com.nt.entity;

import java.io.Serializable;

import org.springframework.web.multipart.MultipartFile;
```

```
import lombok.Data;


@Data
public class JobSeekerData implements Serializable {
      private Integer jsId;
      private String jsName;
      private String jsAddrs;
      private  MultipartFile resume;  //Input stream to hold content of uploded file
      private  MultipartFile photo;


}
```

step4) Develop the Repository Interface

```
package com.nt.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.nt.entity.JobSeekerInfo;

public interface IJobSeekerRepo extends JpaRepository<JobSeekerInfo, Integer> {


}
```

step5) Develope service interface and service Impl class

service Interface

```
public interface IJobSeekerMgmtService {
    public  String registerJobSeeker(JobSeekerInfo info);
}
```

service impl class

```
@Service
public class JobSeekerMgmtServiceImpl implements IJobSeekerMgmtService {
      @Autowired
      private IJobSeekerRepo  jsRepo;

      @Override
      public String registerJobSeeker(JobSeekerInfo info) {

            return "Job seeker is saved with "+jsRepo.save(info).getJsId()+"  id value";
      }

}
```

step6) add entries in application.properties file having datasource cfg and jpa-hibernate entries

In application.properties

```
#Datasource cfg,jpa-hibernate cfg
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
#based in these details the HikariCP DataSource object will be created pointing to
#jdbc con pool for oracle as part autoconfiguration activity

#JPA-hibernate cfgs
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

step7) Cfg ViewResolver in application.properties file

In application.properites

```
#view Resolver cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

step7) add entries for embedded Tomcat server

```
#For Embedded server
server.port=4041
server.servlet.context-path=/JobSeekersApp
```

step8) Cfg CommonsMultipartResolver in main class as spring bean using @Bean method to activate file upload mechasim in spring mvc or spring boot mvc application,

```
@Bean(name="multipartResolver")        fixed bean id
public CommonsMultipartResolver  createCMResolver() {
      CommonsMultipartResolver resolver=new CommonsMultipartResolver();
      resolver.setMaxUploadSizePerFile(50*1024*1024);
      resolver.setMaxUploadSize(-1);  //all files togather how much size is allowed  -1 indicates no limit
      return resolver;
}
```

=>Resolvers are internally used to activate certain certain facilities in spring boot MVC application.. They are internally gathered by calling ctx.getBean(-) method with fixed bean ids .. So we must provide same bean ids while cfg them

For the following spring beans cfg , we need the fixed bean ids

class	fixed bean id
ResourceBundleMessageSource ---------->	messageSource
SessionLocaleResolver ---------->	localeResolver
CommonsMultiPartResolver ---------->	multipartResolver

step9)
Develop handler method in controller class to take implicit reuquest to return LVN for home page

```
@Controller
public class JobSeekerOperationsController {
      @Autowired
      private IJobSeekerMgmtService service;

      @GetMapping("/")
      public  String showHomePage() {
            return  "welcome";
      }

}
```

WEB-INF/pages/welcome.jsp

```
<%@ page  isELIgnored="false" %>


<h1 style="text-align:center"><a href="register">Register JobSeeker</a></h1>
```

```
<h1 style="text-align:center"><a href="list_js">List Jobseekers</a></h1>
```

(For future file downloading operation)

step10) place handler method in controller class to show Jobseeke registration form page

In controller class

```
@GetMapping("/register")
      public  String  showJSRegistrationForm(@ModelAttribute("js") JobSeekerData jsData) {
            //return LVN
            return "jobseeker_register";
      }
```

step 11) Develop form page

jobseeker_register.jsp

```
<%@ page language="java"  isELIgnored="false" %>
<%@taglib uri="http://www.springframework.org/tags/form"  prefix="frm"%>


<frm:form modelAttribute="js" enctype="multipart/form-data">
  <table border="0"  bgcolor="cyan" align="center">
    <tr>
      <td> Name :: </td>
     <td><frm:input path="jsName"/>  </td>
    </tr>
    <tr>
      <td> Address :: </td>
     <td><frm:input path="jsAddrs"/>  </td>
    </tr>
    <tr>
      <td> Select Resume :: </td>
     <td><frm:input type="file" path="resume"/>  </td>
    </tr>
    <tr>
      <td> Select  Photo :: </td>
     <td><frm:input type="file" path="photo"/>  </td>
    </tr>
    <tr>
      <td colspan="2"><input type="submit" value="register">  </td>
    </tr>
  </table>
</frm:form>
```

step12) Develop handler method to process the form submission and to complete file upload activity

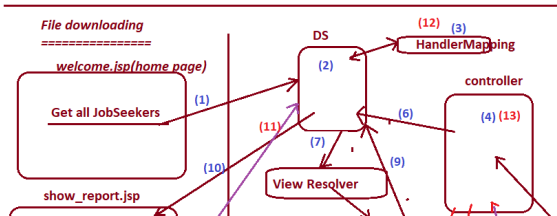
```
private Environment env;

    @PostMapping("/register")
    public  String  registerJSByUploadingFiles(@ModelAttribute("js") JobSeekerData jsData,
                                               Map<String,Object> map)throws Exception {
          //get Upload folder location from properties file
          String storeLocation=env.getRequiredProperty("upload.store");
          // if that not avaiable then create it
          File file=new File(storeLocation);
           if(!file.exists())
                 file.mkdir();

           // get InputStreams representin the upload files content
           MultipartFile  resumeFile=jsData.getResume();
           MultipartFile  photoFile=jsData.getPhoto();
           InputStream isResume=resumeFile.getInputStream();
           InputStream isPhoto=photoFile.getInputStream();
           //get the names of the uploaded files
           String resumeFileName=resumeFile.getOriginalFilename();
           String photoFileName=photoFile.getOriginalFilename();
           //create outstreams reprsenting empty destination files
           OutputStream osResume=new FileOutputStream(file.getAbsolutePath()+"\\"+resumeFileName);
           OutputStream osPhoto=new FileOutputStream(file.getAbsolutePath()+"\\"+photoFileName);
           // perform file copy operation
           IOUtils.copy(isResume,osResume);
           IOUtils.copy(isPhoto,osPhoto);
           //close stream
           isResume.close();
           osResume.close();
           isPhoto.close();
           osPhoto.close();
          //prepare  Entity class obj from Model class obj
           JobSeekerInfo jsInfo=new JobSeekerInfo();
            jsInfo.setJsName(jsData.getJsName());
            jsInfo.setJsAddrs(jsData.getJsAddrs());
            jsInfo.setResumePath(file.getAbsolutePath()+"/"+resumeFileName);
            jsInfo.setPhotoPath(file.getAbsolutePath()+"/"+photoFileName);
          //use  Service
            String msg=service.registerJobSeeker(jsInfo);
          //keep the uploaded file names and  location in model attributes
            map.put("file1", resumeFileName);
            map.put("file2", photoFileName);
            map.put("resultMsg",msg);
            //return LVN
          return "show_result";
    }
```

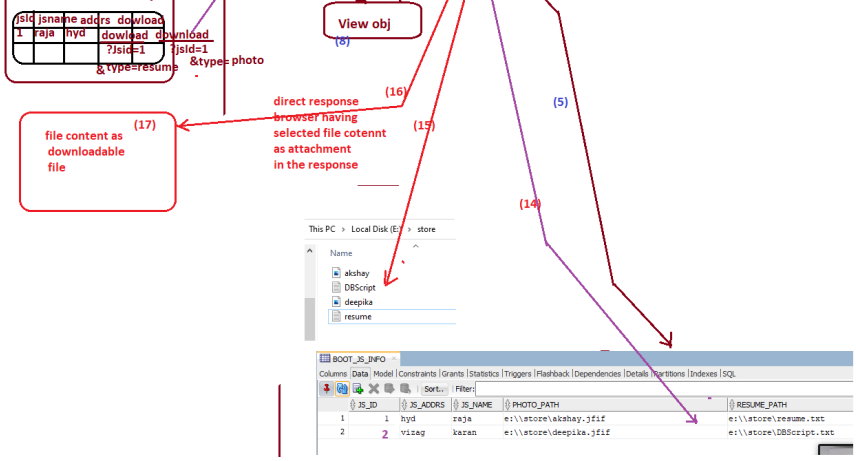
step13) develop the result page

show_result.jsp

```
<%@page isELIgnored="false" %>

<h1 style="color:red;text-align:center">Result page</h1>

<b>
      Result is :: ${resultMsg} <br>
      The uploaded file names are ::  ${file1} , ${file2}
</b>

<br><br>
 <a href="./">home</a>
```

File downloading





Part1 Development steps (Displaying having donwload hyperlinks for files)

==========

step1) add methods in service Interface and in service Impl class

In service Interface

```
public List<JobSeekerInfo> fetchAllJobseekers();
```

In service Impl class

```
@Override
    public List<JobSeekerInfo> fetchAllJobseekers() {
        return jsRepo.findAll();
    }
```

step3) Add the following handler method in controller class

```
@GetMapping("/list_js")
public String showReport(Map<String,Object> map) {
    System.out.println("JobSeekerOperationsController.showReport()");
    //use service
    List<JobSeekerInfo> list=service.fetchAllJobseekers();
    System.out.println(list.size());
    //add result to model attrbutes
    map.put("jsList", list);
    //return LVN
    return "show_report";
}
```

step4) **develope show_report.jsp page in WEB-INF/pages folder as shown below**

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

<c:choose>
  <c:when test="${!empty jsList}">
    <table border="1" align="center" bgcolor="cyan">
      <tr>
        <th>JsId </th><th>JsName </th><th>JsAddrs </th><th>resume </th><th>photo </th>
      </tr>
      <tr>
        <c:forEach var="info" items="${jsList}">
          <tr>
            <td>${info.jsId } </td>
            <td>${info.jsName } </td>
            <td>${info.jsAddrs } </td>
            <td><a href="download?jsId=${info.jsId}&type=resume">download resume</a> </td>
            <td><a href="download?jsId=${info.jsId}&type=photo">download photo</a> </td>
          </tr>
        </c:forEach>
      </tr>
    </table>
  </c:when>
  <c:otherwise>
    <h1 style="color:red;text-align:center">Records not found </h1>
  </c:otherwise>
</c:choose>
```

Standard process for file download activity

a) create File obj having path of the file to be downloaded
File file=new File("E:/store/resume.txt");

b) Get the length file to be downloaded
long length =file.length();

c) make file content length as response content length
response.setContentLength(length);

d) set file MIMEtype as response content type
ServletContext sc=req.getServletContext();
String mimeType=sc.getMimeType("E:/store/resume.txt");
mimeType=mimeType==null?"application/octet-stream":mimeType;
res.setContentType(mimeType); generic /universal mimetype

e) create InputStream representing file to be downloded
InputStream is=new FileInputStream(file);

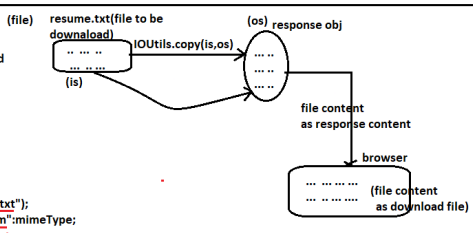
f) create outputStream representing the response obj
ServletOutputStream os=res.getOutputStream();

g) Give instruction to browser to make the recieved as the downloadable file
res.setHeader("Content-Disposition","attachment;fileName="+file.getName());

note:: The default value of "Content-Disposition" header is "inline" i.e the recieved content will be displayed on the browser directly ..where as "attachment" instructs the browser to make recieved response content as the downloadable file with given name.

h) copy File content to response as the response content
IOUtils.copy(is,os);

i) close streams
is.close();
os.close();

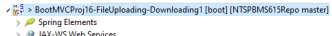


Procedure to perform file donwloading using spring boot MVC App

==========

step1) Write two @Query methods in Repository Interface to get resume path or photoPath based on given JobSeeker Id.

```
public interface IJobSeekerRepo extends JpaRepository<JobSeekerInfo, Integer> {
    @Query("select resumePath from JobSeekerInfo where jsId=:id")
    public String getResumePathByJsId(Integer id);
    @Query("select photoPath from JobSeekerInfo where jsId=:id")
    public String getPhotoPathByJsId(Integer id);
```



```
    }
```

step2) write suporting methods repository methods in Service Interface and Service Impl class

In service Interface

```
public  String   fetchResumePathByJsId(Integer jsId);
   public  String   fetchPhotoPathByJsId(Integer jsId);
```

In service Impl class

```
@Override
    public String fetchResumePathByJsId(Integer jsId) {
        return jsRepo.getResumePathByJsId(jsId);
    }

    @Override
    public String fetchPhotoPathByJsId(Integer jsId) {
        return jsRepo.getPhotoPathByJsId(jsId);
    }
```

step3) Write Handler method in controller class to complete file downloaing activity
note:: if the handler method return type is "null" then it sends response directly to browser.

In controller class

```
@Autowired
    private ServletContext sc;
    @GetMapping("/download")
    public  String   fileDownload(HttpServletResponse  res,
                                  @RequestParam("jsId") Integer id,
                                  @RequestParam("type") String type) throws Exception{
        // get path of the file to be downloaded
        String filePath=null;
        if(type.equalsIgnoreCase("resume"))
         filePath=service.fetchResumePathByJsId(id);
        else
            filePath=service.fetchPhotoPathByJsId(id);
        System.out.println(filePath);
        //create File object representing file to be downloaded
         File file=new File(filePath);
         //get the length of the file and make it as the response content length
         res.setContentLengthLong(file.length());
         //get MIME of the file and make it as the resposen content type
         String mimeType=sc.getMimeType(filePath);
         mimeType=mimeType==null?"application/octet-stream":mimeType;
         res.setContentType(mimeType);
         //create InputStream pointing to the file
         InputStream is=new FileInputStream(file);
         //create OutputStream pointing to response obj
         OutputStream os=res.getOutputStream();
         //instruct the browser to give file content as downloadble file
         res.setHeader("Content-Disposition","attachment;fileName="+file.getName());
         // write  file content to  response obj
          IOUtils.copy(is, os);
         //close streams
          is.close();
          os.close();
        return null;  //makes the handler method to send response directly to browser
    }
```

Different types ViewResolvers in spring boot MVC

=>ViewResolver resolves /identifies physical view comp name , location and returns   View obj having that name and location To DS.

=>All ViewResolvers are the classes implementing org.sf.web.servlet.ViewResolver(I).. The imp view resolvers are

a) InternalResourceViewResolver
b) UrlBasedViewResolver
c) ResourceBundleViewResolver
d) XmlViewResolver
e) TilesViewResolver
f) BeanNameViewResolver
and etc..

note:: The default View resolver that will be activated in spring boot is InternalResourceViewResolver

Diffent ways activating   ViewResolver in spring boot mvc
---

a) using application.properties

```
#view Resolver cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

=>Only for InternalResourceViewResolver

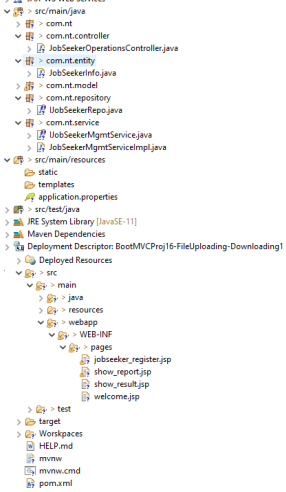
b) By placing   @Bean method in main class or @Configuration class.
(Works for any ViewResolver)

```
@Bean
    public  ViewResolver  createIRVR() {
        InternalResourceViewResolver resolver=new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/pages/");
        resolver.setSuffix(".jsp");
        return resolver;
    }
```

c) Using MVCConfigurer class having the support of ViewResolverRegistry
(Works for all kinds of ViewResolvers)

```
// MyMVCConfigurer.java
package com.nt.config;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Component
public class MyMVCConfigurer implements WebMvcConfigurer {

    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {

        InternalResourceViewResolver resolver=new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/pages/");
        resolver.setSuffix(".jsp");
    registry.viewResolver(resolver);

    }

}
```

if all 3 approaches are used with different settings to register the same ViewResolver which settings will be taken?



Different types ViewResolvers in spring boot MVC

=>ViewResolver resolves /identifies physical view comp name , location and returns View obj having that name and location To DS.

=>All ViewResolvers are the classes implementing org.sf.web.servlet.ViewResolver(I).. The imp view resolvers are

a) InternalResourceViewResolver
b) UrlBasedViewResolver
c) ResourceBundleViewResolver
d) XmlViewResolver
e) TilesViewResolver
f) BeanNameViewResolver
and etc..

note:: The default View resolver that will be activated in spring boot is InternalResourceViewResolver

Diffent ways activating ViewResolver in spring boot mvc

---

a) using application.properties

```
#view Resolver cfg
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

=>Only for InternalResourceViewResolver
(For this entries InternalResourceViewResolver will be activated by taking InternalResourceView (or) JstlView as the default View class)

b) By placing @Bean method in main class or @Configuration class. (Works for any ViewResolver)

```
@Bean
public ViewResolver createIRVR() {
    InternalResourceViewResolver resolver=new InternalResourceViewResolver();
    resolver.setPrefix("/WEB-INF/pages/");
    resolver.setSuffix(".jsp");
    return resolver;
}
```

c) Using MVCConfigurer class having the support of ViewResolverRegistry (Works for all kinds of ViewResolvers)

```
// MyMVCConfigurer.java
package com.nt.config;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Component
public class MyMVCConfigurer implements WebMvcConfigurer {

    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {

        InternalResourceViewResolver resolver=new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/pages/");
        resolver.setSuffix(".jsp");
```

```
    registry.viewResolver(resolver);

    }

}
```

if all 3 approaches are used with different settings to register the same ViewResolver which settings will be taken?

Ans) The settings taken in the @Bean method approach takes place

---

UrlBasedViewResolver

=> It is super class for InternalResourceViewResolver

=> Allows to configure any View class i.e View class configuration is mandatory

=> The physical View comp can be there in any technology like jsp , volocity ,freeemarket, tiles and etc.. becoz we need to cfg View class explicitly according to the View Technology we are using.

=>This ViewResolver gives View comp having prefix+LVN+suffix together as Physical view comp name and location.

```
AbstractAtomFeedView, AbstractFeedView, AbstractJackson2View, AbstractPdfStamperView, AbstractPdfView, AbstractRssFeedView, AbstractTemplateView,
AbstractUrlBasedView, AbstractView, AbstractXlsView, AbstractXlsxStreamingView, AbstractXlsxView, FreeMarkerView, GroovyMarkupView, InternalResourceView,
JstlView, MappingJackson2JsonView, MappingJackson2XmlView, MarshallingView, RedirectView, ScriptTemplateView, TilesView, XsltView
```

(Possible View classes implementing View(I))

Example

In main class or @Configuration class

```
@Bean
    public ViewResolver createUBVResolver() {
        System.out.println("BootMvcProj02WishMessageAppApplication.createUBVResolver()");
        UrlBasedViewResolver resolver=new UrlBasedViewResolver();
        resolver.setPrefix("/WEB-INF/pages/");
        resolver.setSuffix(".jsp");
        resolver.setViewClass(InternalResourceView.class);
        return resolver;
    }
```

=> In UrlBasedViewResolver we need to use view classes as show below

=> if the view comps are servlet,jsp comps of private area then use JstlView or InternalResourceView

=> if the view comps are Tiles then use TilesView

=> if the view comps are Freemarker then use FreeMarkerView

=> if the view comps are GroovyMarkup then use GroovyMarkupView

and etc..

What is the difference b/w InternalResorceView and JstlView?

Ans) Both View classes are given to take the private area servlet,jsp comps as the view comps

=> if the View class InternalResourceView (Best), then we need to add jstl jar files only when jstl tags are used in the jsp pages otherwise not required.

=> if the View class JstlView , then we need to add jstl jar files in the jsp pages irrespective of the jstl tags that are used in the jsp pages.

InternalResourceViewResolver

=>Sub class for UrlBasedViewResolver having InternalResorceView or JstlView as the default view class .. if jstl tags are used takes JstlView otherwise takes InternalResourceView as the default View class

=> Given only for taking servlet,jsp comps as the view comps that are there in private area.

=> if view comps are private area then prefer using this ViewResolver.

=>To activate this ViewResolver we can use application.properties entries(best) or @Bean method or MVCConfigurer class

In main class or @Configuration class

```
@Bean
public ViewResolver createIRVResolver() {
    System.out.println("BootMvcProj02WishMessageAppApplication.createUBVResolver()");
    InternalResourceViewResolver resolver=new InternalResourceViewResolver();
    resolver.setPrefix("/WEB-INF/pages/");
    resolver.setSuffix(".jsp");
    return resolver;
}
```

Takes InteralResoruceView or JstlView as the default View class

What is difference b/w UrlBasedViewResolver and InternalResourceViewResolver?

UrlBasedViewResolver	*InternalResourceViewResolver*
a) Super class to InternalResoruceViewResolver	*a) sub class to UrlBasedViewResolver*
b) Allows to take any view comps of any technology	*b) Allows to take view comps of servlet,jsp technologies*
c)Does not have default View class	*c) It is having default View class (InternalResourceView or JstlView)*
d) In spring boot MVC app we can activate this ViewResolver either either using @Bean method or using MVCConfigurer class	*d) In spring boot MVC app we can activate this ViewResolver either using @Bean method or using MVCConfigurer class or application.properites file entries*
e) Prefere this ViewResolver if u have other than servlet,jsp comps as the view comps	*e) Prefere this ViewResolver if u have servlet,jsp comps as the view comps*

note:: Both ViewResolvers are giving tight coupling b/w LVN and Physical View name

LVN	physcial View comp name
display	/WEB-INF/pages/display.jsp (prefix: /WEB-INF/pages/, suffix: .jsp)
welcome	/WEB-INF/pages/welcome.jsp (prefix: /WEB-INF/pages/, suffix: .jsp)

To achieve loosecoupling use other viewResolvers like ResourceBundleViewResolver or XmlViewResolver

ResourceBundleViewResolver

==========

=> Allows to use properties file to write view comps configurations nothing but mapping logical view name with physical view comp name and location.

=> Gives loose coupling between lvn and physical view comp name

=> We must take seperate properties file views.properties in classpath folder .. if file name is changed or location is changed it must be cfg while creating the obj for ResourceBundleViewResolver class using setBaseName(-) method

example

---

note:: While working with this ViewResolver we can take our choice view class name , phsyical view comp name and location for every logical view name (lvn) i.e for each lvn we can have our choice technology view comp name and location.

step1)

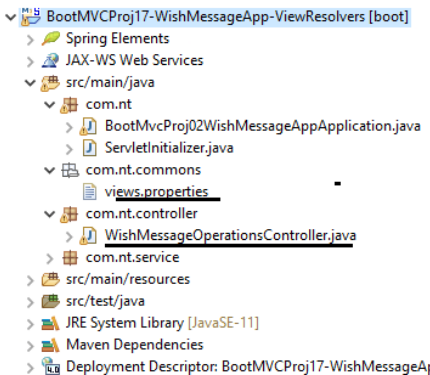
In views.properties (com/nt/commons folder of src/main/java folder)

```
-----------------------------
#lvn.(class)=<pkg>.<view class>
welcome.(class)=org.springframework.web.servlet.view.JstlView
#lvn.url= <name of location of physical view comp)
welcome.url=/WEB-INF/pages/namaskar.jsp

#lvn.(class)=<pkg>.<view class>
show_result.(class)=org.springframework.web.servlet.view.InternalResourceView
#lvn.url= <name of location of physical view comp)
show_result.url=/WEB-INF/pages/display.jsp
```

step2) Configure ResourceBundlerViewResolver as the spring bean in the @Bean method

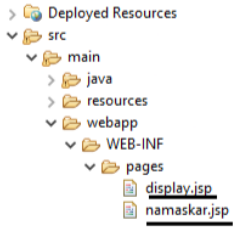
In main class or @Configuration class

```
-----------------------------------------------------------
@Bean
      public  ViewResolver  createRBVResolver() {
    System.out.println("BootMvcProj02WishMessageAppApplication.createRBVResolver()");
        ResourceBundleViewResolver resolver=new ResourceBundleViewResolver();
          resolver.setBasename("com/nt/commons/views");
          return resolver;
      }
```



Controller class

```
@Controller
public class WishMessageOperationsController {
      @Autowired
      private IWishService  service;


      @GetMapping("/")
      public  String  showHomePage() {
      System.out.println("WishMessageOperationsController.showHomePage()");
       return "welcome";
```




```
}


@GetMapping("/wish")
public  String   fetchWishMessage(Map<String,Object> map) {
        //use service
        String  msg=service.generateWishMessage();
         //  keep  data in model attributes
        map.put("wMsg", msg);
        map.put("sysDate",new Date());
        //return MAV
        return "show_result";
}


}
```

ResourceBundleViewResolver and XmlViewResolver classes are deprecated from version 5.3
in favor of Spring's common view resolver variants * and/or custom resolver implementations

99% times we use application.properties file entries based
InternalResourceViewResolver for normal jsp pages as the view comps

XmlViewResolver

=>Allows us take seperate xml file (spring bean cfg file) whose default name "views.xml" in classpath folders to configure view class, url for every logical view name as spring bean cfg details.

=> if file name or location is different we need to confugure while creating object for XmlViewResolver class.

step1) develop spring bean cfg file having lvns mapped with view classes and physical view comp name and locations

=>LVN as the bean id
=>View class as spring bean class name
=> url property Injection :: having the name and location of physical view comp

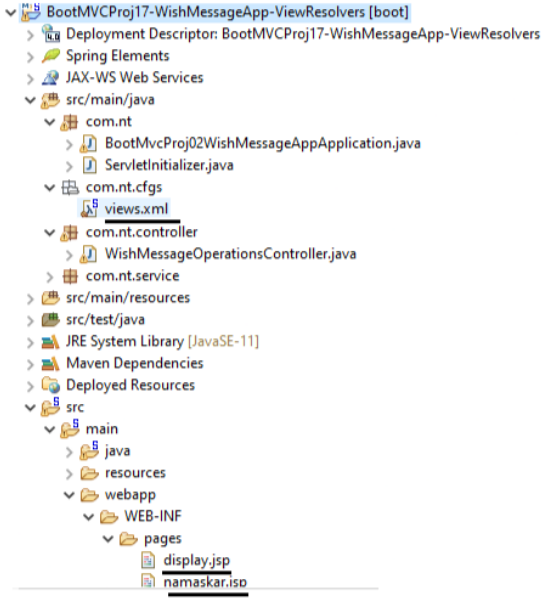
views.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean id="welcome"  class="org.springframework.web.servlet.view.InternalResourceView">
    <property name="url" value="/WEB-INF/pages/namaskar.jsp"/>
  </bean>
            LVN                          View class
  <bean id="show_result"  class="org.springframework.web.servlet.view.JstlView">
    <property name="url" value="/WEB-INF/pages/display.jsp"/>
  </bean>          property
                                physical view comp name and location
```

```
</beans>
```

step2) Cfg XmlViewResolver as the spring bean class using @Bean method

```
@Bean
    public  ViewResolver  createXVResolver() {
 System.out.println("BootMvcProj02WishMessageAppApplication.createXVResolver()");
      XmlViewResolver resolver=new XmlViewResolver();
         resolver.setLocation(new ClassPathResource("com/nt/cfgs/views.xml"));
         return resolver;
     }
```




```
@Controller
public class WishMessageOperationsController {
      @Autowired
      private IWishService  service;


      @GetMapping("/")
      public  String  showHomePage() {
      System.out.println("WishMessageOperationsController.showHomePage()");
       return "welcome";
      }


      @GetMapping("/wish")
      public  String   fetchWishMessage(Map<String,Object> map) {
            //use service
            String  msg=service.generateWishMessage();
             //  keep  data in model attributes
            map.put("wMsg", msg);
            map.put("sysDate",new Date());
            //return MAV
            return "show_result";
      }



}
```

ViewResolver Chaining

=>when multiple viewResolvers are configured if first view resolver fails to locate physical view comp and control going to next view resolver in the chain is called ViewResolver chaining.

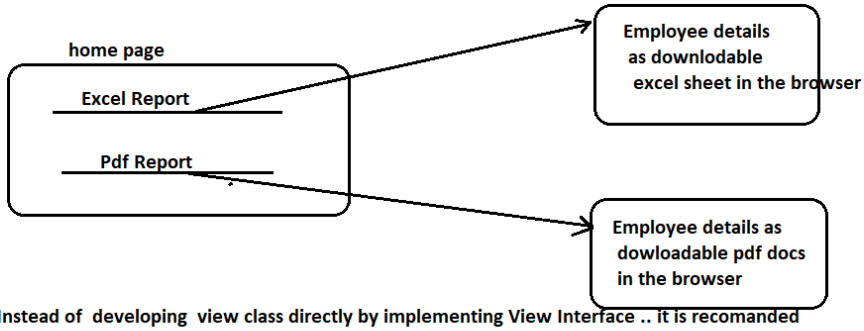
=>InternalResourceViewResolver,UrlBasedViewResolver , XmlViewResolver , ResourceBundlerViewResolver do not support any kind of ViewReoslver chaining.. The TilesviewResolver ,BeanNameViewResolver supports view Resolver chaining.

==========
BeanNameViewResolver and taking JAva calsses as view comps
==========

=>some times we need generate Excel reports, pdf reports having model data . For this we need to use thrid party apis like iText api (for pdf reports) , POI api for Excel reports

=>Writing java code in jsp pages is not recomanded .. Since spring MVC allows us to take different technology view comps.. So we think about take java classes as the View comps (classes implementing View(I)) and we can add these third party apis in those View classes.

=>if want to take spring bean classes (view classes) as the view comps ..it is recomanded to Seperate ViewResolver called "BeanNameViewResolver".



=>Instead of developing view class directly by implementing View Interface .. it is recomanded to take the same class by extening AbstractXxxView class to simplify process like AbstractPdfView , AbstractExcelView and etc..

```
@Component("excel_report")
public class EmployeeExcelReport extends AbstractExcelView{

    public void buildExcelDocument(-,-,-,-,-){
      ...
      .. use poi api here to build excel sheet
      .. by getting data from model atttributes
      ..
    }

}

@Component(" pdf._report")
public class Employee Pdf Report extends Abstract Pdf View{

    public void build Pdf Document(-,-,-,-,-){
      ...
      .. use iText api here to build pdf
      .. by getting data from model atttributes
      ..
    }

}
```

To use java classes as the Views .. we need to take the support of BeanNameViewResolver by confiuring it in the main class using @BeanMethod

```
@Bean
public ViewResolver BeanNameViewResolver(){

 BeanNameViewResolver resolver=new
              BeanNameViewResolver();
 resolver.setOrder(Ordered.HIGHEST_PRECEDENCE);
 return resolver;
}
```

Becoz of this hight precence this resolver gets high priority in the view resolver chaining

Integer MAX value it holds

note:: The default priority vlaue is :: Ordered.LOWEST_PRECEDENCE (It holds Integer MIN value)

Example App
==========

step1) create Project adding the follwing dependencies

a) lombok api b) web c) spring data jpa d) oracle driver (e) itext 2.1.7 (f) poi 3.17

```
<!-- https://mvnrepository.com/artifact/com.lowagie/itext -->
<dependency>
  <groupId>com.lowagie</groupId>
  <artifactId>itext</artifactId>
  <version>2.1.7</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>3.17</version>
</dependency>
</dependencies>
```

collect from mvnrepository.com

step2) create Model class

Employee.java

```
package com.nt.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Entity
@Table(name="emp")
@Data
public class Employee {
	@Id
	@GeneratedValue
	private  Integer empno;
	private String ename;
	private String  job;
	private  Long deptNo;
	private  Double sal;


}
```

step3) Repository Interface

```
package com.nt.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.nt.model.Employee;
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

step4) Develop service Interface and service Impl class

Service interface

```
package com.nt.service;
import java.util.List;
import com.nt.model.Employee;
public interface IEmployeeMgmtService {
	public  List<Employee> getAllEmployees();
}
```

Service Impl class

```
//EmployeeServiceImpl.java
package com.nt.service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.nt.model.Employee;
import com.nt.repo.IEmployeeRepository;

@Service
public class EmployeeServiceImpl implements IEmployeeMgmtService {
	@Autowired
	private  IEmployeeRepository  empRepo;

	@Override
	public List<Employee> getAllEmployees() {
		return  empRepo.findAll();
	}

}
```

step4) add method in controller to reuturn lvn of home page

In controller class

```
-------------------------
@Controller
public class EmployeeOperationsController {
	@Autowired
	private IEmployeeMgmtService service;

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

step5) add viewReolvers(InternalResourceviewResolver) , DataSource cfg in application.propperties file (server managed jdbc con pool)

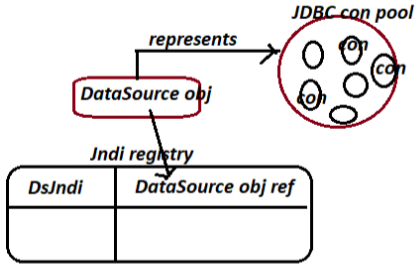
In application.properties

```
#View Resolvers configuration (IRVR)
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

To create jdbc con pool for oracle in tomcat server add the following <Resource> tag in Context.xml file under <Context> tag.

```
<Resource name="DsJndi" auth="Container" type="javax.sql.DataSource"
	maxTotal="100" maxIdle="30" maxWaitMillis="10000"
	username="system" password="manager" driverClassName="oracle.jdbc.driver.OracleDriver"
	url="jdbc:oracle:thin:@localhost:1521:xe"/>
```

(Collect from <Tomcat_home>/webapps/docs/jndi-datasource-examples-howto.html)



In appplication.properties

```
-------------------------------------
#Server managed jdbc con pool configuration
spring.datasource.jndi-name=java:/comp/env/DsJndi
```

fixed prefix — jndi name given in <Resource>

step6) develop home page having hyperlinks as WEB-INF/pages/welcome.jsp

welcome.jsp

```
---------------
<%@ page isELIgnored="false"  %>


<h1 style="text-align:center"><a href="report?type=excel">Excel Report</a></h1>
<br><br>
<h1 style="text-align:center"><a href="report?type=pdf">PDF Report</a></h1>
```

step6) Configure BeanNameViewResolver as spring bean using @Bean method in main class

In main class

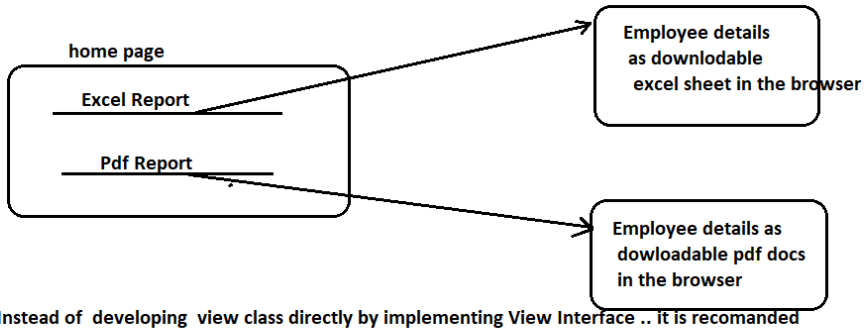
```
@Bean
public  ViewResolver createBNVResolver() {
	BeanNameViewResolver resolver=new BeanNameViewResolver();
	resolver.setOrder(Ordered.HIGHEST_PRECEDENCE);
	return resolver;
}
```

==========
BeanNameViewResolver and taking JAva calsses as view comps
==========

=>some times we need generate Excel reports, pdf reports having model data . For this we need to use thrid party apis like iText api (for pdf reports) , POI api for Excel reports

=>Writing java code in jsp pages is not recomanded .. Since spring MVC allows us to take different technology view comps.. So we think about take java classes as the View comps (classes implementing View(I)) and we can add these third party apis in those View classes.

=>if want to take spring bean classes (view classes) as the view comps ..it is recomanded to Seperate ViewResolver called "BeanNameViewResolver".



=>Instead of developing view class directly by implementing View Interface .. it is recomanded to take the same class by extening AbstractXxxView class to simplify process like AbstractPdfView , AbstractExcelView and etc..

```
@Component("excel_report")
public class EmployeeExcelReport extends AbstractExcelView{

    public void buildExcelDocument(-,-,-,-,-){
      ...
      .. use poi api here to build excel sheet
      .. by getting data from model atttributes
      ..
    }

}
```

```
@Component(" pdf._report")
public class Employee Pdf Report extends Abstract Pdf View{

    public void build Pdf Document(-,-,-,-,-){
      ...
      .. use iText api here to build pdf
      .. by getting data from model atttributes
      ..
    }

}
```

To use java classes as the Views .. we need to take the support of BeanNameViewResolver by confiuring it in the main class using @BeanMethod

```
@Bean
public ViewResolver BeanNameViewResolver(){

 BeanNameViewResolver resolver=new
              BeanNameViewResolver();
 resolver.setOrder(Ordered.HIGHEST_PRECEDENCE);
 return resolver;
}
```

Becoz of this hight precence this resolver gets high priority in the view resolver chaining

Integer MAX value it holds

note:: The default priority vlaue is :: Ordered.LOWEST_PRECEDENCE (It holds Integer MIN value)

Example App
==========

step1) create Project adding the follwing dependencies

a) lombok api b) web c) spring data jpa d) oracle driver (e) itext 2.1.7 (pdf docs) (f) poi 3.17 (excel docs)

```
<!-- https://mvnrepository.com/artifact/com.lowagie/itext -->
<dependency>
  <groupId>com.lowagie</groupId>
  <artifactId>itext</artifactId>
  <version>2.1.7</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>3.17</version>
```

collect from mvnrepository.com

```
        </dependency>
      </dependencies>
```

step2) create Model class

Employee.java

```
package com.nt.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Entity
@Table(name="emp")
@Data
public class Employee {
      @Id
      @GeneratedValue
      private  Integer empno;
      private String ename;
      private String  job;
      private  Long deptNo;
      private  Double sal;


}
```

step3) Repository Interface

```
package com.nt.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.nt.model.Employee;
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

step4) Develop service Interface and service Impl class

Service interface

```
package com.nt.service;
import java.util.List;
import com.nt.model.Employee;
public interface IEmployeeMgmtService {
      public  List<Employee> getAllEmployees();
}
```

Service Impl class

```
//EmployeeServiceImpl.java
package com.nt.service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.nt.model.Employee;
import com.nt.repo.IEmployeeRepository;

@Service
public class EmployeeServiceImpl implements IEmployeeMgmtService {
      @Autowired
      private  IEmployeeRepository  empRepo;

      @Override                (w)
      public List<Employee> getAllEmployees() {
            return  empRepo.findAll();   (x)
      }

}
```

step4) add method in controller to reuturn lvn of home page

In controller class

```
@Controller
public class EmployeeOperationsController {
    @Autowired
    private IEmployeeMgmtService service;

(g) @GetMapping("/")    (d?)
    public String showHomePage() {
        //return LVN
        return "welcome";  (h)
    }
}
```

step5) add viewReolvers(InternalResourceviewResolver) , DataSource cfg in application.propperties file (server managed jdbc con pool)

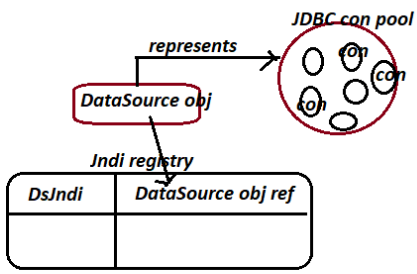
In application.properties

```
#View Resolvers configuration (IRVR)
spring.mvc.view.prefix=/WEB-INF/pages/   (k)
spring.mvc.view.suffix=.jsp
```

To create jdbc con pool for oracle in tomcat server add the following <Resource> tag in Context.xml file under <Context> tag.

```
<Resource name="DsJndi" auth="Container" type="javax.sql.DataSource"
          maxTotal="100" maxIdle="30" maxWaitMillis="10000"
          username="system" password="manager" driverClassName="oracle.jdbc.driver.OracleDriver"
          url="jdbc:oracle:thin:@localhost:1521:xe"/>
```

(Collect from <Tomcat_home>/webapps/docs/jndi-datasource-examples-howto.html)



In appplication.properties

```
#Server managed jdbc con pool configuration
spring.datasource.jndi-name=java:/comp/env/DsJndi
```

fixed prefix — jndi name given in <Resource>

step6) develop home page having hyperlinks as WEB-INF/pages/welcome.jsp

welcome.jsp (n)

```
<%@ page isELIgnored="false" %>

                                                    (o)
<h1 style="text-align:center"><a href="report?type=excel">Excel Report</a></h1>
<br><br>
<h1 style="text-align:center"><a href="report?type=pdf">PDF Report</a></h1>
```

step6) Configure BeanNameViewResolver as spring bean using @Bean method in main class

In main class

```
(j)  @Bean
     public ViewResolver createBNVResolver() {
(b1)     BeanNameViewResolver resolver=new BeanNameViewResolver();
         resolver.setOrder(Ordered.HIGHEST_PRECEDENCE);
         return resolver;
     }
```

}

note:: BeanNameViewResolver Supports ViewResolver Chaining i.e if this ViewResolver fails to locate the view comps the task will be passed next ViewResolver that is there in the chain .. In this example that ViewResolver name is InternalResourceViewResolver.

step7) Add another handler method in controller class to get Employees Data as the report data based on the hyperlink that is clicked.

```
@GetMapping("/report")  (u)                (r?)
    public  String  showReport(Map<String,Object> map,
                               @RequestParam("type") String type) {
        //use  service                              (v)
    (y) List<Employee> empsList=service.getAllEmployees();
        // add results to model attribute
        map.put("empsList", empsList);
        //return lvn based on the hyperlink that is clicked
         if(type.equalsIgnoreCase("excel"))
            return  "excel_report"; (z)
         else
            return "pdf_report";

    }
```

step8) Deveop the java classes as the view Classes having the above LVNS as the bean ids

note:: BeanNameViewResolver looks to map lvn with that spring bean acting view comp having lvn as the bean id.

ExcelReportView.java

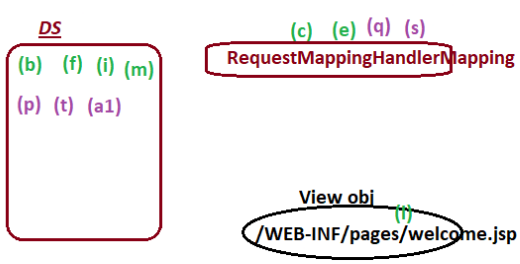
```
package com.nt.view;

import java.util.List;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.view.document.AbstractXlsView;

import com.nt.model.Employee;

@Component("excel_report")  (c1)
public class ExcelReportView extends AbstractXlsView {
   private int i=1;
      @Override                (d1)
      public  void buildExcelDocument(Map<String, Object> map, Workbook workbook,
                                      HttpServletRequest req,HttpServletResponse res) throws Exception {
          //get Model attributes data
          List<Employee> list=(List<Employee>)map.get("empsList");
          //create  excel sheet in work
          Sheet sheet1=workbook.createSheet("Employee");


          Row row1=sheet1.createRow(0);
          row1.createCell(0).setCellValue("EMPNO");
          row1.createCell(1).setCellValue("ENAME");
          row1.createCell(2).setCellValue("JOB");
          row1.createCell(3).setCellValue("SAL");
          row1.createCell(4).setCellValue("DEPTNO");

          list.forEach(emp->{
              // add row to Excel sheet
              Row row=sheet1.createRow(i);
              //add cells to row
              row.createCell(0).setCellValue(emp.getEmpno());
              row.createCell(1).setCellValue(emp.getEname());
```

(e1) :: Sends Model attributes data to browser excel response

```
                    row.createCell(2).setCellValue(emp.getJob());
                    row.createCell(3).setCellValue(emp.getSal());
                    if(emp.getDeptno()!=null)
                      row.createCell(4).setCellValue(emp.getDeptno());
                    i++;
                });//forEach(-)

        }//method
    }//class
```

PdfReportView.java

```
package com.nt.view;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.view.document.AbstractPdfView;

import com.lowagie.text.Document;
import com.lowagie.text.Font;
import com.lowagie.text.Paragraph;
import com.lowagie.text.Table;
import com.lowagie.text.pdf.PdfWriter;
import com.nt.model.Employee;

@Component("pdf_report")
public class PdfReportView  extends AbstractPdfView {

    @Override
    public void buildPdfDocument(Map<String, Object> map, Document doc, PdfWriter writer,
            HttpServletRequest req, HttpServletResponse res) throws Exception {
        //get Model attributes data
            List<Employee> list=(List<Employee>)map.get("empsList");
            // add Paragraph
            Paragraph para=new Paragraph("Employee Report", new Font(Font.TIMES_ROMAN,
                                                                     Font.DEFAULTSIZE,
                                                                     Font.BOLDITALIC));
            doc.add(para);
            //add table content
            Table table=new  Table(5,((ArrayList) list).size());
            for(Employee emp:list) {
                table.addCell(String.valueOf(emp.getEmpno()));
                table.addCell(emp.getEname());
                table.addCell(emp.getJob());
                table.addCell(String.valueOf(emp.getSal()));
                if(emp.getDeptno()!=null)
                  table.addCell(String.valueOf(emp.getDeptno()));
                else
                    table.addCell("_____");

            }
            doc.add(table);
    }


}
```



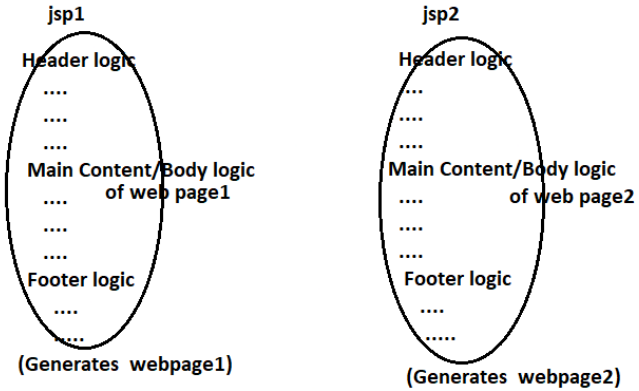
http://localhost:2020/BootMVCProj18-ViewAsClass-BeanNameViewResolver/ (a)

Tiles Framework
==========

=> Tiles framework is a pluggable framework that can linked any other mvc framerwork like struts, spring mvc,jsf and etc... to implement Composite View Design Pattern with Layout control support.

Composite View Design Pattern

Problem::

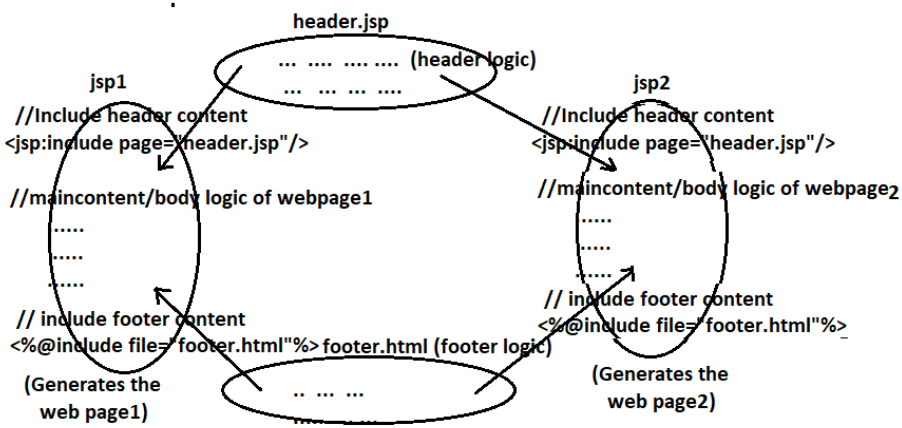


note:: Here header ,footer logics are not reusable logics though they are required in multiple web comps.

note: header,footer logics are looking like boilerplate code.

Solution:: Implement Composite view Design pattern

=>keep different common logics in diffent seperate web comps and include their output in main web comps either using <jsp:include> or <%@include>



note:: Header header ,footer logics have become reusable logics i.e bolerplate code problem is avoided.

=>Singular View :: web page generated by the single jsp page

=>composite view :: the view(web page) generated by multiple jsp comps togather

LayoutControl:: all the web pages of the web application will be developed based on the common template page. So all the web pages of the web application will be having uniform look. The modification done in layout /template page will reflect to all the web pages of the web application

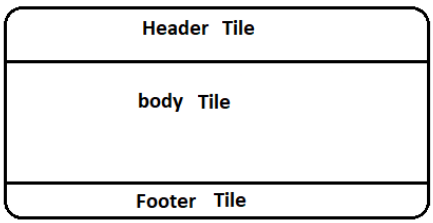
The popular layouts/ templates are

(a) Classic layout
(b) Two-Column Layout
(c) Three -Column Layout
(d) Circular Layout

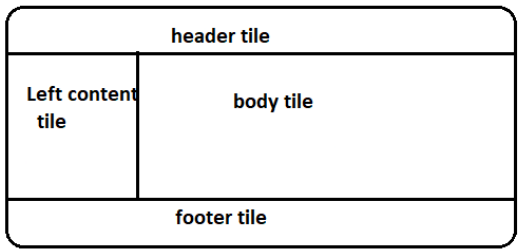
and etc..

=> a Tile is logical portion of web page i.e it represents certain portion of the webpage

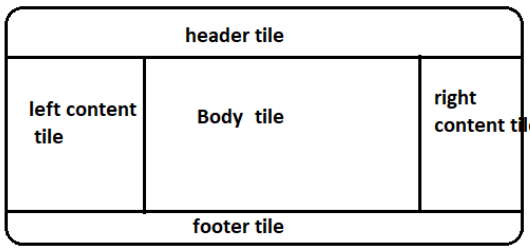
classic Layout



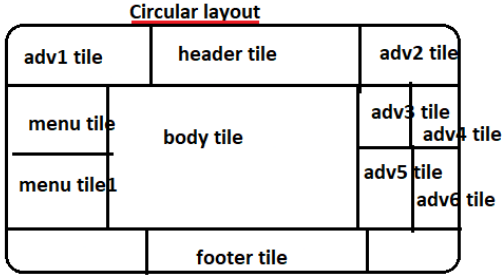
2 column layout



3 column layout



Circular layout



mvnrepository.com is given based three column layout.

By integrating Tiles framework with Spring MVC we can make every web page coming based on layout page having composite view pattern implementation

step1) Activate Tiles framework by creating TilesConfigurer as the spring bean in the main class.

In main class

```
@Bean
public TilesConfigurer createTilesConfigurer(){
   TilesConfigurer configurer=new TilesConfigurer();
   configurer.setDefinitation("/WEB-INF/tiles.xml"); //any xml file name of any location
                                                      can be taken.. the default
                                                     //name is tiles.xml of WEB-INF folder
  return configurer;
}
```

step2) create Layout page as jsp page by importing tiles jsp taglibrary

WEB-INF/pages/layout.jsp (classic layout) (m)

(z)

```
%@ taglib prefix="tiles" uri="http://tiles.apache.org/tags-tiles" %>

 <table border="0" width="100%" heigth="100%">
  <tr height="30%" width="100%">
   <td> <tiles:insertAttribute name="header"/> </td>   tile1
  </tr>
  <tr height="60%" width="100%">
   <td>  <tiles:insertAttribute name="body"/> </td>    tile2
  </tr>
  <tr height="10%" width="100%">
   <td>  <tiles:insertAttribute name="footer"/> </td>   tile3
  </tr>
 </table>
```

displays layout.jsp as the webpage having the following tile values

a) header tile:: /WEB-INF/pages/header.jsp
b) body tile :: /WEB-INF/pages/home.jsp
c) footer tile :: /WEB-INF/pages/footer.jsp

a) header tile:: /WEB-INF/pages/header.jsp
b) body tile :: /WEB-INF/pages/login .jsp
c) footer tile :: /WEB-INF/pages/footer.jsp

home.jsp (WEB-INF/pages/)

```
<a href="login">Login Page </a>
```

(n)

step3) Identity the no.of the webpages that u want to develop based on the layout page

(a) home page(page1) (b) login page (c) registration page

step4) develop WEB-INF/tiles.xml file having tiles definitations on 1 per web page basis

WEB-INF/tiles.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tiles-definitions PUBLIC
      "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
      "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">

<tiles-definitions>
     <!-- tile definitation for page --!>
    <definition name="homePageDef" template="/WEB-INF/pages/layout.jsp">
      <put-attribute name="header" value="/WEB-INF/pages/header.jsp" />
      <put-attribute name="body" value="/WEB-INF/pages/home .jsp" />
      <put-attribute name="footer" value="/WEB-INF/pages/footer.jsp" />
    </definition>
     <!-- tile definitation for page2 --!>
      <definition name="loginPageDef" template="/WEB-INF/pages/layout.jsp">
        <put-attribute name="header" value="/WEB-INF/pages/header.jsp" />
        <put-attribute name="body" value="/WEB-INF/pages/login.jsp" />
        <put-attribute name="footer" value="/WEB-INF/pages/footer.jsp" />
      </definition>

     <!-- tile definitation for page  --!>
      <definition name= " reg PageDef" template="/WEB-INF/pages/layout.jsp">
        <put-attribute name="header" value="/WEB-INF/pages/header.jsp" />
        <put-attribute name="body" value="/WEB-INF/pages/registration.jsp"/>
        <put-attribute name="footer" value="/WEB-INF/pages/footer.jsp" />
      </definition>
 </tile-definitations>
```

WEB-INF/tiles.xml (improved using tiles-inheritence)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tiles-definitions PUBLIC
    "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
```

"http://tiles.apache.org/dtds/tiles-config_2_0.dtd">

```
<tiles-definitions>

  <!-- BaseDef -->
  <definition name="baseDef" template="/WEB-INF/pages/layout.jsp">
    <put-attribute name="header" value="/WEB-INF/  pages /header.jsp" />
    <put-attribute name="body" value="" />
    <put-attribute name="footer" value="/WEB-INF/  pages   /footer.jsp" />
  </definition>
  <definition name="homePageDef" extends="baseDef">
      <put-attribute name="body" value="/WEB-INF/pages/home.jsp" />
   </definition>
  <definition name="loginPageDef" extends="baseDef">
      <put-attribute name="body" value="/WEB-INF/pages/login.jsp" />
  </definition>
   <definition name="regPageDef" extends="baseDef">
      <put-attribute name="body" value="/WEB-INF/pages/registration.jsp" />
   </definition>
</tiles-definitations>
```

(l) (y) (k) (x)

step5) Configure TilesViewResolver as spring bean in main class using @Bean method

in Main class

```
@Bean                    (j) (w)
public  TilesViewResolver  createTVResolver(){
  TilesViewResolver  resolver=new TilesViewResolver();
  return resolver;
 }
```

step 6) Develope controller class having handler methods returning tile definitation names as the LVNs

```
@Controller
public  WebOperationsController{

 @GetMapping("/")   (g)                (d?)
 public  String showHomePage(){
     return "homePageDef";
 }      (h)  ------------------------>Tile definitation name as lvn

        (t)
 @GetMapping("/login")              (q?)
 public String  showLoginPage(){
   return "loginPageDef"; (u)
 }  ------------------------>Tile definitation name as lvn

 @GetMapping("/register")
 public String  showRegisterPage(){
   return "regPageDef";
  }  ------------------------>Tile definitation name as lvn

 }
```

DispatcherServlet

(b) (f) (i)

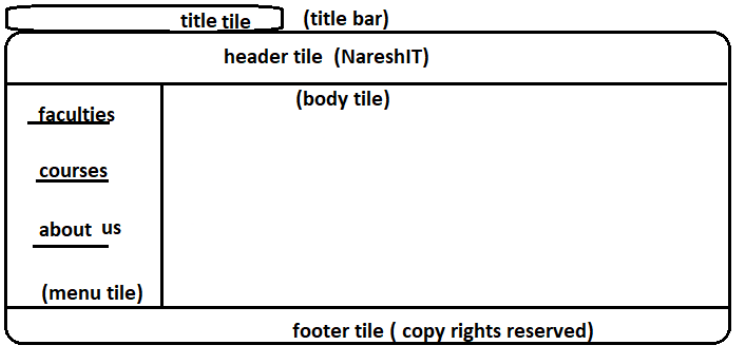
(o) (s) (v)

(c) (e) (p) (r)

RequestMappingHandlerMapping

http://localhost:2525/TilesApp1/ (a)

==========
Tiles framework Implementation
==========



no.of pages :: 4 no.of tiles in web page is :: 5

1) home page (header tile:: header.jsp
footer tile :: footer.jsp
body tile :: home.jsp
menu tile :: menu.jsp)
tiltle tile :: home page

2) faculties page (header tile:: header.jsp
footer tile :: footer.jsp
body tile :: faculties.jsp
menu tile :: menu.jsp)
tiltile tile :: faculties page

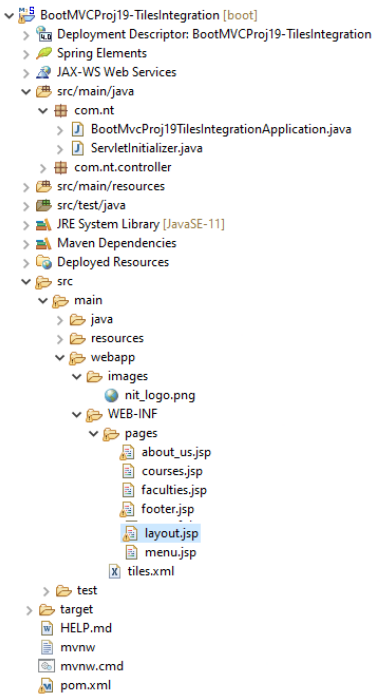
3) courses page (header tile:: header.jsp
footer tile :: footer.jsp
body tile :: courses.jsp
menu tile :: menu.jsp
titile tile :: courses page)

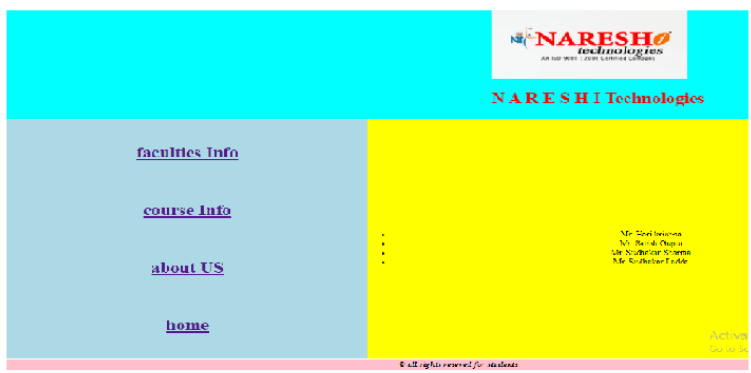
4) about us page (header tile:: header.jsp
footer tile :: footer.jsp
body tile :: aboutUs.jsp
menu tile :: menu.jsp
titile tile :: AboutUs page)

dependecies to add in pom.xml file are

spring web, lombok, tiles-core ,tiles-jsp ,jstl

```
<!-- https://mvnrepository.com/artifact/org.apache.tiles/tiles-core -->
<dependency>
   <groupId>org.apache.tiles</groupId>
   <artifactId>tiles-core</artifactId>
   <version>3.0.8</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.tiles/tiles-jsp -->
<dependency>
   <groupId>org.apache.tiles</groupId>
   <artifactId>tiles-jsp</artifactId>
   <version>3.0.8</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>jstl</artifactId>
   <version>1.2</version>
</dependency>
```





Thymeleaf (As alternate to jsp pages)
==========

=> Another UI Technology that is build on the top of html for dynamic web pages
=> only html gives static web pages .. html tags+ thymeleaf tags gives dynamic webpages . It is light weight alternate to heavy weight jsp pages..
=> In the execution of jsp page lot of memory and lot of cpu time is required becoz interanlly translates jsp to a jsp equalent servlet comp and creates multiple implicit objs (9) if used or not used.. for all these things lot of memory and cpu time required.
=>To overcome the above problems of jsp pages use thymeleaf as lightweight alternate for rendering dynamic webpages...

=> We need to write thymeleaf tags in html tags.. by importing thymeleaf namespace by specifying its namespace uri..

```
<html xmlns:th="https://www.thymeleaf.org">
  ...
</html>
```

=>Thymeleaf can be used only in java env..
=>thymeleaf file extension must be .html file

=>Spring boot gives built-in support of thymeleaf UI by giving
default prefix is <classpath>/templates/ (classpath here is src/main/resources folder)
default suffix is .html

note:: we can not mixup thymeleaf and jsp UI togather in a single spring MVC or spring Boot MVC web application.

=> To use thymeleaf in spring boot add this starter to pom.xml

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-thymeleaf -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
  <version>2.3.4.RELEASE</version>
</dependency>
```

=>To execute thymeleaf code Thymeleaf engine is required which will come becoz this jar file.. This engine converts thymeleaf tags to html code.. and sends to browser as response..

=> Standard jsp tags identified with fixed prefix called <jsp: xxxx> and similary thymeleaf tags are indentified with <th: xxx> prefix

=> Popular symbols in thymeleaf programming
@ --->to specify Location (/<globalpath>/<request path>) (useful in href, action urls)
$ ---> To read data from container managed scopes like model attributes
* ---> To bind/link data to form comps (useful only in thymeleaf forms)
(model properties, ref data)

a. To read data from model attributes/container managed scopes
-> for primitive/wrapper/String type model attrbutes th:text="${<attribute name>}" (eg: <span th:text="${wMsg}"/>)
-> for Object type model attrbutes th:text="${<objectName>}" eg <span th:text="${emp}"/>)
-> for getting property values from Object type model attrbutes th:text="${<objectName>.<property name>}" (eg: <span th:text="${emp.ename}"/>)
-> for looping through arrays/collection th:each="<counter variable>:${<collection/array type attribute>}"

```
(eg: <table>
       <tr th:each="cust:${custList}" >
           <td> <span th:text="${cust.cno}"/> </td>
           <td> <span th:text="${cust.cname}"/> </td>
           .....
       </tr> )
```

b. To display images (To link image file to <img> tag)

```
<img th:src="@{/path}">
```

c. To Link/map with hyperlink

```
<a th:href="@{/path}"> xxxx </a>
```

d.To Link /map css file

```
<link rel="stylesheet" th:href="@{/path}">
```

e. To Link /map java script fle

```
<script type="text/javascript" th:src="@{/path}">
  ....
</script>
```

while working with thymeleaf , we must take controller having class level global path using @RequestMapping("/...") annotation.
(Global path)

f) Thymeleaf forms are birectional forms i.e they support DataBinding(writing form data to model class obj) and DataRendering (writing handler methods supplied model attributes/model class obj data to form comps)

=> To specify action url <form th:action="@{global path/request path}" >
= > For binding model class obj data/model attributes data to form comps (for form backing object operation)
-> <form th:object="${model attribute name/object name of model clas}"> (alternate to <frm:from modelAttribute="....">)
=> For binding model class obj data /model attributes data to form comps
<input type="text" th:field="*{<model class property name/model attrobute name>}"> (alternate to <frm:input path="..."/>)

(ref data)

What is the difference b/w <th:text> and <th:field> tags in thymeleaf

th:text is given to read . data : from different scopes and to display them on browser.. like reading and displaying model class obj data and model attrbutes data..

th:field is given to bind model class obj property values /model attribute values(reference data) to form comps (text boxes)

Converting MiniProject to Thymeleaf UI based Application

step1) add thymeleaf starter to pom.xml file

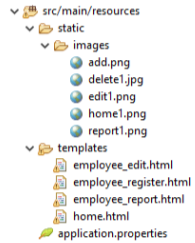
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

* we can remove JSTL jar files from pom.xml

step2) Provide global path to controller class

```
@Controller
@RequestMapping("/employee") //global path
public class EmployeeController {
  ...
  ...
  ...
  }
```

step3) copy js,images folders of webapp/webcontent to "static" folder of src/main/resources folder and WEB-INF/pages folder jsp files to template folder of "src/main/resources" folder , change .jsp extension to .html



we can comment view resolver cfg in application.properties

```
##View Resolver
#spring.mvc.view.prefix=/WEB-INF/pages/
#spring.mvc.view.suffix=.jsp
```

refer BootMVCProj20-MiniProject-CURD-Thymeleaf [boot]

note:: we can delete images, js folders of src/main/webapp folder and src/main/webapp/WEB-INF/pages folder

step4) modify "template" folder .html files code thymeleaf code from jsp code.

step5) Rum the Application...

home.html

```
<html xmlns:th="https://thymeleaf.org">

<h1 style="text-align: center"><a th:href="@{/employee/emp_report}"><img th:src="@{/images/report1.jpeg}"/></a></h1>
```

employee_report.html

```
<html xmlns:th="https://thymeleaf.org">

 <div th:if="${!empsList.empty}">
   <table border="1" bgcolor="cyan" align="center">
       <tr bgcolor="pink">
         <th>eno </th> <th>ename </th> <th>desg</th> <th>salary </th> <th>deptNo </th> <th>operations </th>
       </tr>
       <tr th:each="emp:${empsList}">
           <td><span th:text="${emp.empno}"/> </td>
```

```
            <td><span th:text="${emp.ename}"/> </td>
            <td><span th:text="${emp.job}"/> </td>
            <td><span th:text="${emp.sal}"/> </td>
            <td><span th:text="${emp.deptno}"/> </td>
            <td><a th:href="@{/employee/edit_employee(eno=${emp.empno})}"><img th:src="@{/images/edit1.png}" width="50" height="50">   </a>
                <a th:href="@{/employee/delete_employee(eno=${emp.empno})}" onclick="confirm('Do u want to delete')"><img th:src="@{/images/delete1.jpg}" width="50" height="50"></a>
            </td>
          </tr>
    </table>
    </div>
 <div th:if="${empsList.empty}">
      <h1 style="color:red;text-align:center"> Records  not  found </h1>
 </div>
        <blink><h1 style="color:green;text-align:center" th:text="${resultMsg}"/> </blink>
<br>
 <h1 style="text-align:center"><a th:href="@{/employee/insert_employee}">Add Employee <img th:src="@{/images/add.png}"/></a> </h1>

<br>
<br>
 <h1 style="text-align:center"><a th:href="@{/employee/}">home  <img th:src="@{/images/home1.png}"/> </a></h1>
```

modify_employee.html

```
 <html xmlns:th="https://thymeleaf.org">

 <h1 style="color:blue;text-align:center">Edit Employee </h1>

 <form th:action="@{/employee/edit_employee}"  th:object="${emp}"  method="POST">
   <table border="0" bgcolor="cyan" align="center">
   <tr>
       <td>  Employee eno :: </td>
       <td>  <input type="text" th:field="*{empno}"  readonly="true"/> </td>
     </tr>
     <tr>
       <td>  Employee name :: </td>
       <td>  <input type="text" th:field="*{ename}"/>  </td>
     </tr>
     <tr>
       <td>  Employee Desg :: </td>
       <td>  <input type="text" th:field="*{job}"/> </td>
     </tr>
     <tr>
       <td>  Employee Salary :: </td>
       <td>  <input type="text" th:field="*{sal}"/> </td>
     </tr>
     <tr>
       <td>  Employee Dept no :: </td>
       <td>  <input type="text" th:field="*{deptno}"/> </td>
     </tr>
     <tr>
       <td> <input type="submit"  value="Edit Employee" > </td>
       <td> <button  type="reset" > Cancel</button>  </td>
     </tr>
   </table>
 </form>
```

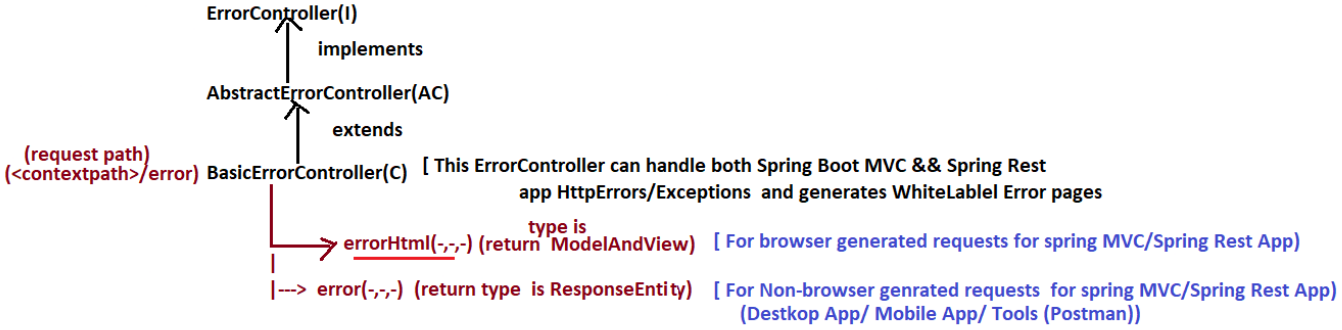
Error or Exception handling in spring MVC/spring Boot MVC

Http status/response status codes

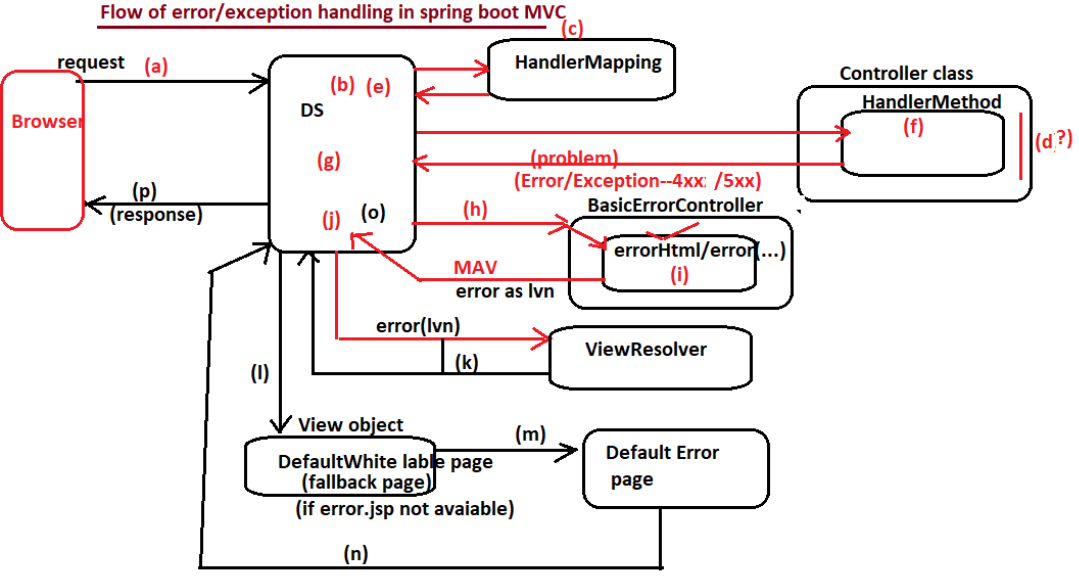
101 -199 / 1xx :: Information (The happenings in the server will be displayed)
200- 299 /2xx :: success (for given reuqest , the response has come successfully with out error /exception)
300-399 / 3xx :: Redirection (The given request to one website will be redirected to another website)
400-499/4xx :: Incomplete (Some problem in the locating and executing web comp) (Client side errors) | (these codes will come when there is a problem in the execution of web applicaiton)
500-599 / 5xx :: Server Error (For exceptions raised in web comps or in underlying server/container)

400-499 /4xx :: Http Errors
500-599/5xx :: Http Errors(becoz of exceptions raised in web comps/ in underlying server/container)

Spring Rest/
=> Spring Boot MVC is having pre-defined Controller to handle these HttpErrors/exceptions and to give default whiteLabel Error pages



Flow of error/exception handling in spring boot MVC



```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h1 style="color:red;text-align:center">Server Internal Problem (5xx)</h1>
    <p style="text-align:center"><img  src="images/500_error.jfif"/> </p>
</body>
```

404.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h1 style="color:red;text-align:center">Requested Resource not found (404)</h1>
    <p style="text-align:center"><img  src="images/404_error.jfif"/> </p>
</body>
</html>
```

We can also make Custom Exceptions generating our choice error codes (4xx or 5xx) by using @ResponseStatus annotation on the top of Custom Exception class

EmployeeNotFoundException.java

```java
package com.nt.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;
                                    502  error
@ResponseStatus(code = HttpStatus.BAD_GATEWAY)
public class EmployeeNotFoundException extends RuntimeException {
      public EmployeeNotFoundException() {
            super();
      }


      public EmployeeNotFoundException(String msg) {
            super(msg);
      }

}
```

In Service class

```java
@Override
    public Employee fetchEmployeeByEno(int eno) {
        //use repo
     Optional<Employee> opt=empRepo.findById(eno);
      if(opt.isPresent())
        return opt.get();   //returns  Employee
      else
        throw new  EmployeeNotFoundException("Problem in getting employee");
      //throw new RuntimeException("Problem in fetching record");
    }
```

alternate code

```java
return empRepo.findById(eno).orElseThrow(()->new
EmployeeNotFoundException("emp not found"));
```

Problem in handler method

↓

specific error code page (like 404.jsp ,500.jsp,501.jsp and etc..)

↓ if not found

specific error series page (4xx.jsp , 5xx.jsp)

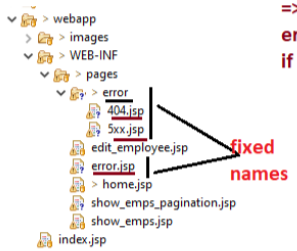
↓ if not found

error.jsp (Glogal error page)

↓ if not found

DefaultWhite Lablel page

=>Instead of whiteLabel error page which is fallback error page for diffent problems we can configure custom Error Pages for different Http Errors/ Exceptions (4xx and 5xx problems)



=>we configure error.jsp in WEB-INF/pages folder as global Error page i.e these error page executes for 4xx and 5xx error/exceptions raised in web application.. if specific error code based error pages not found..

WEB-INF/pages/error.jsp

```
<%@ page isELIgnored="false" %>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
   <h1 style="color:red;text-align:center"> Some Internal problem --Inconvience is regratted </h1>
   <hr>
    <table border="1" bgcolor="cyan" align="center">
     <tr>
      <td>status </td>
      <td>${status}</td>
     </tr>
     <tr>
      <td>timestamp </td>
      <td>${timestamp}</td>
     </tr>
      <tr>
      <td>message </td>
      <td>${message}</td>
     </tr>
     <tr>
      <td>type </td>
      <td>${type}</td>
     </tr>
       <tr>
      <td>path </td>
      <td>${path}</td>
     </tr>

     <tr>
      <td>trace</td>
      <td>${trace}</td>
     </tr>
    </table>
</body>
</html>
```

---

we can cfg error page for each error code like 404,405, 500,501,502 and etc.. for that we need to take <errorcode>.jsp as the file name in WEB-INF/pages/error folder.
or we can also take error page for spefic serries error codes like 4xx.jsp for 400-499 error codes and similarly 5xx.jsp for all 500-599 errors/exceptions..
we should also these pages in WEBE-INF/pages/error folder.

place

5xx.jsp

404.jsp

Servers in spring boot MVC web applications

==========

The spring boot mvc / spring boot rest applications can be deployed and executed in two types of servers

a) spring boot web Embedded servers
   i) apache Tomcat (default) (popular) (1)
   ii) Eclipse Jetty
   iii) Jboss undertow

=>These are generally used in the dev ,test env.. of the Project

b) external servers
   i) apache tomcat (3)
   ii) jetty
   iii) wildfly (1)
   iv) glassfish (2)
   v) undertow
   vi) weblogic
   vii) webshpere
   viii) jboss
   and etc... (In fact any java based web server or application server can be used)

=>These are used in "uat" , "prod" env.. of the Project

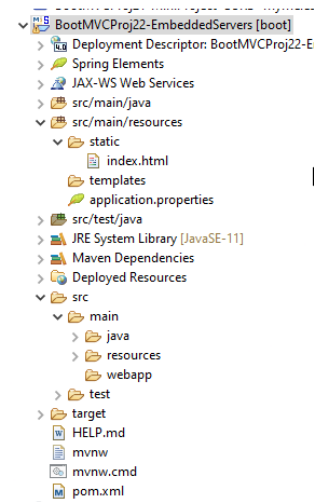
=>web server gives only ServeltContainer ,Jsp Container where as App server gives Servlet container, jsp container and EJB container

=> spring boot is not given to develop EJB comps .. So we do not need EJB container for spring boot applications
=> Spring boot is given to develop standalone Apps, web applications, Distributed Apps (restfull-web based) which are generally packaged as jar files/war files .. So to deploy and execute spring boot MVC Apps/ Spring rest apps there is no need of java based application servers.. we can manage with Java based web servers .. For this reason they have not given any Application server as embedded server in spring boot .

=> EAR file (Enterpirse Application archieve)
ear file = jar file (ejb comp) +war file (web application)

=> spring boot does not support ear files deployment.. generally we need application server to deploy and execute ear files ...
So to deploy and execute spring boot mvc /spring rest app we just need web servers like tomcat, jetty ,undertow and etc..

=>In order to work with other embedded servers in spring boot mvc applications we need to disable spring-boot-tomcat-starter dependency from pom.xml file and we need to add other embedded servers starter dependencies to the pom.xml file



=>we can place static content in spring boot mvc application in "static" folder of src/main/resources folder.
=>static content means "css" ,"js","images", "html","audio","video" and etc.. files.

src/main/resources
  -> static
    |---->images
        |--->*.jpg or *.png or etc..
    |--->html
        |--->*.html
    |--->css
        |--->*.css
    |--->js
        |---->*.js

In Most of servers the index.html placed in "static" folder of "src/main/resources" folder will be taken as default welcome file..

=>if we add spring- boot-web-starter by selecting "spring web" towards creating spring boot project then the spring -boot-tomcat-starter dependency will be coming automatically.. which gives embedded tomcat server.

In recent version of

```
<dependency>
        <groupId>org.springframework.boot</groupId>
```

spring boot .. it is coming an independent and seperate starter.. earlier it used come as dependent jar file for spring-boot-web-starter dependency

```
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
```

To work with jetty server as the embedded server

step1) dissable spring-boot-starter-tomcat dependency from pom.xml file

```
<!--  <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope>
        </dependency> -->
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Go to dependecy hierarchy tab in pom.xml file , search for spring-boot-starter-tomcat jar and execlude that jar file by using right click exclude option.

step2) add spring-boot-starter-jetty dependency to pom.xml

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-jetty -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

step3) Run the application as standalone App

Right click on the Project --->run as ---> spring boot App

step4) Test the Application..

url in browser :: http://localhost:4041/ServersApp/

application.properties

```
server.port=4041
server.servlet.context-path=/ServersApp
```

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
   <h1 style="color:blue;text-align:center"> welcome to spring boot MVC </h1>
</body>
</html>
```

Procedure to use the "JBOSS undertow" server as the embedded server in spring boot MVC

==========

step1) same as above do it even for jetty server

step2) add "spring-boot-starter-underow" to pom.xml

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
starter-undertow -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-undertow</artifactId>
</dependency>
```

step3 & step4) same as above...

---

if we add all the 3 embedded server's dependency starters to pom.xml file then the spring boot mvc app chooses the embedded server in the following order

a) tomcat
b) jetty (if tomcat dependency is not avaialbe)
c) undertow ((if tomcat,jetty dependencies are not avaialbe)

Adding "Dev Tools" support to spring boot web MVC /spring rest Project

=>generally when we do source code changes (especially java code) in spring boot web mvc , spring rest applications we need to restart server to reflect that changes.. Some IDEs like eclipse automatically reloads the web application for code changes but that is not so effective some times.. which again forces us to go for clean project ,restart ide and etc.. addtional activities.

[Some times these IDEs will not recognize the changes done static content especially css files , js files]

=>To overcome this problem and simplify code modifications in the development mode of the project , it is recomanded to add " spring-boot -devtools" dependency to pom.xml for

in pom.xml

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
    </dependency>
```

add

we can this stater as the dependency while creating the project or after creating project using (right click on project --->spring--->add devtools)

devtools :: Developer Tools

note: the moment if we try to save chanages done in .java files using "save" button or ctrl+s key the dev tools recognizes the changes in .class files (byte code) and internally uses some live parellel server to reflect changes .. by restarting servers and reloading the apps..