Spring MVC using Spring Boot

Spring MVC

DEFINITION/WHAT

MVC is Software architectural design pattern

A Spring MVC is a Java Framework which is used to develop dynamic web applications

NEED/WHY

Need 1 : Separates the functionality of an application into three interconnected parts - Model, View, and Controller .

Need 2 : Re-usability of the code and parallel development

Need 3 : It uses light-weight servlet container to develop and deploy your application.

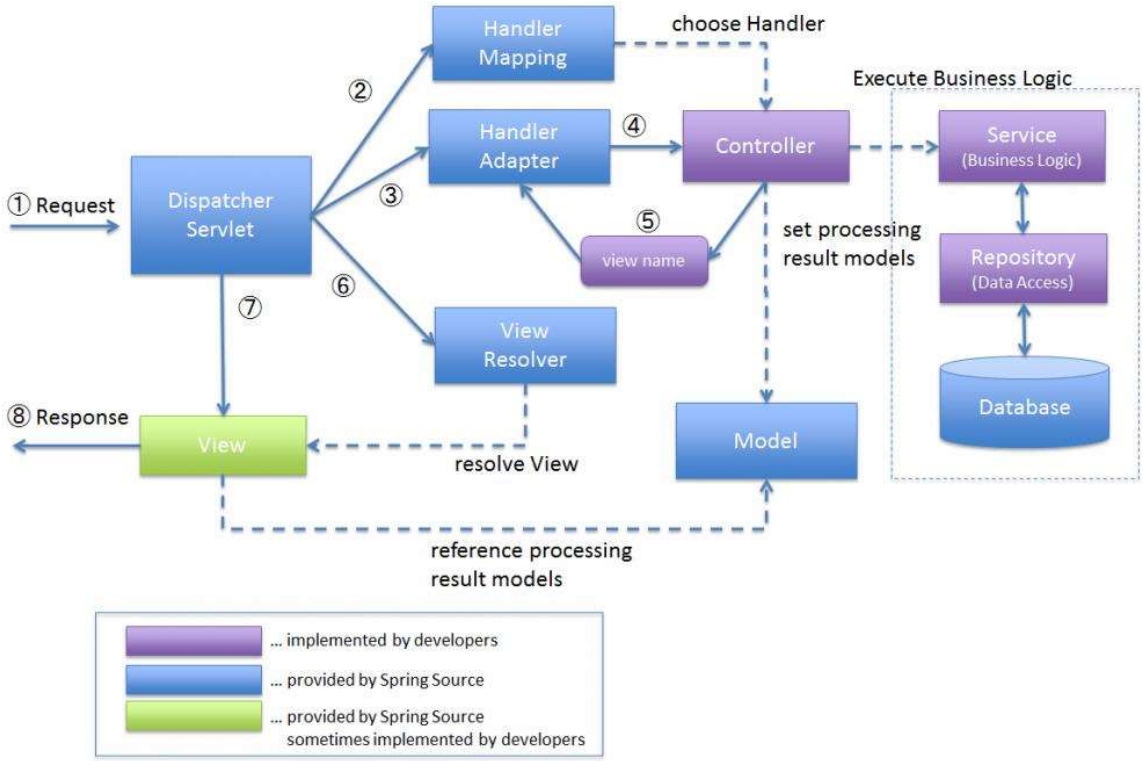
Need 4 : Flexible Mapping - It provides the specific annotations that easily redirect the page.

Need 5 :Not tied to a specific view technology (for example, JSP), but we have the option to choose from the ones we like the most

IMPLEMENTATION/HOW IT INTERNALLY WORKS

To create and run a Spring MVC web application in spring boot, you need to add the spring-boot-starter dependency in your pom. xml file

Spring MVC Request flow:


Handler Mapping
choose Handler
Execute Business Logic
②
Handler Adapter
④
Controller
Service (Business Logic)
① Request
Dispatcher Servlet
③
⑤
view name
set processing result models
Repository (Data Access)
⑥
⑦
View Resolver
Database
⑧ Response
View
Model
resolve View
reference processing result models
... implemented by developers
... provided by Spring Source
... provided by Spring Source sometimes implemented by developers

ViewResolver

Definition/What

Provides a mapping between view names and actual views

Need/Why

Need : Enable you to render models in a browser without tying the implementation to a specific view technology

Implementation/How it internally works

The default view resolver is an automatically-registered InternalResourceViewResolver

This defines prefix and suffix to the view name

Real Time Example

Example

```
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">

    <property name="prefix" value="/WEB-INF/jsp/"></property>

    <property name="suffix" value=".jsp"></property>

  </bean>
```

Additional Information

The Spring framework comes with quite a few view resolvers
e.g. InternalResourceViewResolver, BeanNameViewResolver, and a few others.

Spring MVC Tag Libraries

Definition/What

Spring provides a form tag library to implement view pages using the JSP.

These tags are data binding-aware tags that can automatically set data to Java object/bean and also retrieve from it.

Need/Why

Need 1 : It provides tags for creating HTML components, error messages, themes, and internationalized messages

Need 2: These tags are the configurable and reusable building blocks for a web page

Implementation/How it internally works

It is a built-in library so we can use it on the JSP page by using this tag

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

Real Time Example

After adding above tag, we can create an HTML form by using the form prefix

For example , to create an input text, we can use it like :

<form:input type="text" path="name" />

BindingResult

Definition/What

Is Spring's object that holds the result of the validation and binding and contains errors that may have occurred

Need/Why

Need : Useful in error handling

Real Time Example

Ex:

```
        @RequestMapping("/helloagain")

  public String submitForm( @Valid @ModelAttribute("emp") Employee e,
        BindingResult br)    {

    if(br.hasErrors())       {

      return "viewpage";

    }     else       {

    return "final";

    }    }
```

Additional Information

The BindingResult must come right after the model object that is validated or else Spring will fail to validate the object and throw an exception.

Validations in Spring MVC

Definition/What

Used to restrict the input provided by the user.

Need/Why

Need : Input data is validated/verified before it is used

Implementation/How it internally works

Bean validation API is used to apply constraints on object model via annotations

Some of the validation annotations are @NotNull, @Min, @Max, @Size

In the controller class, the @Valid annotation applies validation rules on the provided object.

Real Time Example

```
public class Employee {

   private String name;

   @Size(min=1,message="required")

   private String pass;

          //...

}

@RequestMapping("/helloagain")

  public String submitForm( @Valid @ModelAttribute("emp") Employee e,
          BindingResult br)

   {   //....

}
```

ADDITIONAL INFORMATION

To validate the user's input, it is required to use the Spring 4 or higher version and Bean Validation API

Spring validations can validate both server-side as well as client-side applications

Controller

Definition/What

A controller is responsible for controlling the way that a user interacts with an MVC application.

Implementation/How it internally works

The Controller is responsible for controlling the application logic and acts as the coordinator between the View and the Model

The controller class is responsible for processing incoming REST API requests, preparing a model, and returning the view to be rendered as a response

@Controller annotation is used for Spring controllers

Front Controller

Definition/What

A Front Controller is a common design pattern in web applications

Is used to receive requests and delegate to other components in the application for actual processing

Dispatcher Servlet

Definition/What

Acts as a front controller

It provides a single entry point for a client request to Spring MVC web application and forwards request to Spring MVC controllers for processing.

Need/Why

Need 1: It provides a mechanism for request processing

Need 2 : DispatcherServlet also plays an important role in view resolution, error handling, locale resolution, theme resolution, etc.

Implementation/How it internally works

DispatcherServlet does following things in Spring MVC:

a. Receives all requests as Front Controller and provides a single entry point to the application
b. Mapping requests to correct Spring MVC controller (using handler mapping)
c. Consulting ViewResolvers to find correct View
d. Forwarding request to chosen View for rendering
e. Returning the response to the client
f. Creates web-context to initialize the web-specific beans like controllers, view resolvers, and handler mapping

Exception Handling in Spring MVC

Definition/What

Is a mechanism where any error raised from any part of the application is sent to the browser in the form of proper response

Need/Why

Need 1: To make sure you are not sending server exceptions to client

Implementation/How it internally works

Developers can write the exception handling code at one single location so that exception raised from any part of the application will arrive at this central location.

These are couple of ways for exception handling in spring MVC.

A. @ExceptionHandler annotation over the exception handler method
B. Global Exception Handler using @ControllerAdvice annotation

@ExceptionHandler

Definition/What

This annotation is applied over a method which acts as an exception handler

Need/Why

Need 1: To annotate the method(s) in the controller class for handling the exceptions raised during the execution of the controller methods

Implementation/How it internally works

Specify an exception handler method for a class whose methods can throw an exception

Any uncaught exception thrown by the methods of this class will be directed to this exception handler

It is achieved by using @ExceptionHandler annotation over the exception handler method

Real Time Example

```
@ExceptionHandler(IOException.class)
public void exceptionHandler(){ }
```

Here , the method exceptionHandler will catch any uncaught exceptions of type java.io.IOException from the class in which this method is declared

Return type of method can be anything
(void, String, ModelAndView object, java.lang.Map or a custom JSON object) , based on requirement

Additional Information

This approach applies to only the controller in which the handler method is declared.

@ControllerAdvice

Definition/What

This annotation is applied over a class which contains exception handler methods which are annotated with @ExceptionHandler.

Need/Why

Need 1: Centralized exception handling mechanism so that exceptions thrown across the application are handled at one place

Implementation/How it internally works

A separate class containing exception handler is created

This class is annotated with @ControllerAdvice and contains exception handler methods which are annotated with @ExceptionHandler annotation.

Whenever any uncaught exception is thrown from any controller, it arrives to the appropriate error handler methods in this class

Real Time Example

```
@ControllerAdvice
public class GlobalErrorHandler {

@ExceptionHandler(IOException.class)
 public ModelAndView inputOutputError(Exception e) {
    //...code
 }

@ExceptionHandler(AppException.class)
  public ModelAndView applicationError(AppException e) {
   //..code
  }
}
```

Additional Information

Any class annotated with @ControllerAdvice becomes a controller-advice and three types of method are supported:

1. Exception handling methods annotated with @ExceptionHandler.
2. Model enhancement methods (for adding additional data to the model) annotated with @ModelAttribute. Note that these attributes are not available to the exception handling views
3. Binder initialization methods (used for configuring form-handling) annotated with @InitBinder.

@InitBinder

Definition/What

Used to customize the request being sent to the controller

Need/Why

Need 1: It helps to control and format requests that come to the controller as it is defined in the controller.

Implementation/How it internally works

The @InitBinder annotated methods will get called on each HTTP request if we don't specify the 'value' element of this annotation

Each time this method is called a new instance of WebDataBinder is passed to it

To be more specific about which objects our InitBinder method applies to, we can supply 'value' element of the annotation @InitBinder.

The 'value' element is a single or multiple names of command/form attributes and/or request parameters that this init-binder method is supposed to apply to.

Real Time Example

```
@InitBinder
public void initBinder(WebDataBinder binder) {
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

        binder.registerCustomEditor(Date.class, new
CustomDateEditor(dateFormat, true));
  }
```

Additional Information

The methods annotated with @InitBinder support all arguments types that handler methods supports, except for command/form objects and corresponding validation result objects

One of the argument should be WebDataBinder

Return type should be void.

WebDataBinder

Definition/What

Is a DataBinder that binds request parameter to JavaBean objects.

Need/Why

Need : Used to register custom formatter, validators and PropertyEditors.

Real Time Example

```
WebDataBinder.addCustomFormatter(..);
WebDataBinder.addValidators(..);
WebDataBinder.registerCustomEditor(..);
```

Model

Definition/What

In Spring MVC, The model represents a Java object carrying data

Model defines a holder for model attributes and is primarily designed for adding attributes to the model

Real Time Example

```
@GetMapping("/getMessage")
public String getMessage(Model model) {
                model.addAttribute("message", message);
                 return "show";
}
```

ModelMap

Definition/What

ModelMap is an extension of Model with the ability to store attributes in a map and chain method calls

Need/Why

Need 1 :ModelMap object is used to pass multiple values from Spring MVC controller to view

Real Time Example

```
@GetMapping("/printViewPage")
public String passParametersWithModelMap(ModelMap map) {
   map.addAttribute("welcomeMessage", "welcome");
   map.addAttribute("message", "Baeldung");
   return "viewPage";
}
```

ModelAndView

Definition/What

Is an object that holds both the model and view.

Need/Why

Need 1:ModelAndView make it possible for a controller to return both model and view in a single return value

Implementation/How it internally works

Inside ModelAndView the view object can be either a view name which is resolved by a ViewResolver or a view object can be specified directly.

The model is a map to add multiple objects.

Real Time Example

```
@RequestMapping(method = RequestMethod.GET)
 public ModelAndView printHello() {
            ModelAndView modelAndView = new ModelAndView("hello");

            modelAndView.addObject("message", "Welcome to Spring MVC
                  framework");
            return modelAndView;
      }
```

Here, printHello' method return a ModelAndView instance and it holds the view 'hello' and a message "Welcome to Spring MVC framework"

@ModelAttribute

Definition/What

An annotation that binds a method parameter or method return value to a named model attribute which further can be used in the view page

Need/Why

Need 1: Used to bind request parameters with model object, validation

Implementation/How it internally works

It can be used either at the method level or method parameter level

Spring MVC will invoke all methods with @ModelAttribute annotation before any handler method executed by the framework

An @ModelAttribute on a method parameter indicates the parameter should be retrieved from the model.

Real Time Example

@ModelAttribute at a Method Level:

```
@ModelAttribute("user")
public String save(User user, Model model) {
        model.addAttribute("user", user);
        return "response";
}
```

@ModelAttribute at a Method Parameter Level:

```
@PostMapping("save")
public String save(@ModelAttribute("user") User user, Model model) {
        model.addAttribute("user", user);
        return "response";
}
```

Additional Information

The annotation works only if the class is a Controller class (i.e. annotated with @Controller)

Alternative /Differences

Method level Vs method parameter level

Method level annotation:-

Is useful when we always like to populated model with certain attributes

Method level ModelAttribute annotation cannot be mapped directly with any request

Developers can add the values in the Model at a global level

@ModelAttribute methods are invoked before the controller methods annotated with @RequestMapping are invoked.

Method Parameter Level:

It binds the form data with a POJO bean

The framework tries to find this argument from the model, in case this is not available, it first creates and adds it to the model

Spring MVC Annotations

@RequestParam

Definition/What

It reads the HTML form data provided by a user and bind it to the request parameter.

Need/Why

Need 1: used to read the form data and bind it automatically to the parameter present in the provided method

Real Time Example

EX.

```
public String display(@RequestParam("name") String name,@RequestParam("pass")
String pass,Model m)
{
        //..code
}
```

@RequestMapping

Definition/What

This annotation maps HTTP requests to handler methods of MVC and REST controllers.

@RequestMapping can be applied to the controller class as well as methods

Need/Why

Need 1: used to map web requests onto specific handler classes and/or handler methods

Real Time Example

EX.

```
@RequestMapping(value = "/ex/foos", method = RequestMethod.GET)
@ResponseBody
public String getFoosBySimplePath() {
        return "Get some Foos";
}
```

@ResponseBody

Definition/What

A spring annotation which binds a method return value to the web response body

Need/Why

Need 1: This annotation tells a controller that the object returned is automatically serialized into JSON and passed back into the HttpResponse object

Implementation/How it internally works

It is not interpreted as a view name

It uses HTTP Message converters to convert the return value to HTTP response body, based on the content-type in the request HTTP header.

@SessionAttributes

Definition/What

This annotation is used at controller class level

Need/Why

Need 1: used to store the model attribute in the session

Implementation/How it internally works

It is not int.

Real Time Example

```
@SessionAttributes("user")
public class LoginController {

        @ModelAttribute("user")
        public User setUpUserForm() {
        return new User();
        }
}
```

Here, the model attribute 'user' will be added to the session if the name attribute of the @ModelAttribute and @SessionAttributes annotations is same