Spring MVC with Boot

Tuesday, February 18, 2020 10:14 AM

MVC

By manual approach we can use the pattern but we have to use handler mapping class, abstract controller and lots of configuration file. So we can use frameworks for that.

Libraries :
- Struts from apache
- JSF from oracle
- Spring MVC from pivotal

RULES :
- Every layer in MVC has designated specific logic .
- All operation must take place under the control of controller servlet.
- View layer(JSP) resource should not talk with model layer directly, they must interact via the controller.
- There can be multiple resource program in view layer and also in business logic/Data Access layer but must only have one controller.
- Client cannot access view resources directly.

SPRING MVC

- With respect the MVC architecture browser gives the request to the web application
- As a front controller it traps the request(set url pattern as "/" to make as a front controller)
- Front controller **HandlerMapping** component, to decide a helper controller class, to process the request for the use case related request uri.
- Front controller passes the control to the helper controller class, this should be annotated with **@Controller.**
- This helper controller should have request processing methods for each request uri and should be annotated with **@RequestMapping("/uri").**
- Helper controller class method will process the request and by taking support of service layer or data access layer classes.
- Helper Controller class returns the processed logic and the view logical name in the form of ModelAndView object.
- Front controller uses the ViewController to get the View object. Having view layer technology and resource name keeps the processed data in request scope.
- Front controller uses the View object and render the control to the appropriate view resource, And include the view given response.
- Front controller gives the data.

NOTE -	Since dispatcher servlet will act as a front controller this class uses some predefined spring bean classes. • HandlerMapping • ViewResolver implementation class • HelperController class • Some other specific spring bean classes

NOTE	- In order to convert the class into a spring bean, internally create spring container of type WebApplicationContext. By loading the spring configutaion inside the init method of the Dispatcher Servlet class. We need to pass the configuration file to the dispatcher servlet using init param tag. Generally we place spring configuration file inside WEB-INF folder.

web.xml

```
<servlet>
    <servlet-name>myds</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcheServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/beans.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>myds</servlet-name>
    <url-patter>/</url-pattern>
</servlet-mapping>
```

NOTE - If you follow the naming rule of the spring configuration as **WEB-INF/servletlogicalname-servlet.xml** then using init-param tag is optional

RequestMappingHandlerMapping

- This checks if the uri is matching with value of @RequestMapping annotation applied on a method of @Controller annotated class
- If it matches it returns the object of helper controller class to instruct the Dispatch servlet to execute the method.
- This class need not configure in spring configuration class.

ViewResolver

- Maps the view resolvers to physical view page.
- View resolver is an interface with method *public View resolveViewname(String viewName, java.util.Locale locale)*
- Provides multiple view resolvers implementation classes. In that InternalResourceViewResolver class takes the logical view name given by helper controller class and resolve the physical view name by adding prefixes and suffice then it returns the view object to the dispatcher servlet.
- We must configure this view resolver class inside spring configuration file

```
<context:component-scan base-package="com.cts"/>
<beans id = "vr" class="org.springframework.web.servlet.view.InternalResource">
    <property name = "prefix" value="/WEB-INF/pages"/>
    <property name = "suffix" value=".jsp"/>
</beans>
```

STEPS TO CREATE SPRING MVC USING WEB.XML

1. Create a dynamic web project and include the maven dependency for spring-webmvc
2. Create a WEB-INF/web.xml file and register the dispatcher servlet class with load on start and as a front controller("/")
3. Create a spring configuration file with the convention servletlogicalname-servlet.xml and register some spring bean. And enable the component scanning feature.
4. Create a helper controller class and annotate it with @Controller
5. In this class define the request Handler method and it should be annotated with @RequestMapping annotation

```
@Controller
public class MyController{
    @RequestMapping("/hello")
    public String sayHello(){
        return "home"; // returns jsp
    }
}
```

6. Create a home.jsp file in WEB-INF/jsp folder
7. Enable the context:component-scan in web.xml

REQUEST HANDLER

Request handling method of controller class is a flexible method

- We can give any name
- Any number of parameters
- Return various types of data

Parameter types :

- Primitive type
- Http :
  - HttpServletRequest
  - HttpServletResponse
  - HttpSession
- Object : (set attribute to request scope)
  - Model
  - Map
  - ModelMap
- Form bean class obj
  It is a class that is mapped with form page data. `public String fun1(StudentBean st){}// mapped from form tag`
- BindingResult
  It gives the result of the form and should come after form bean class;

Possible Return Types:

- Void
  If will take the name of the function as the view name.
- String
  Returns the view model name and use the model to send any data. `model.addAttribute()`
- ModelAndView
  We can represent both view name and model;

```
mv.setViewName("view"); mv.addObject("key","value");
ModelAndView mv = new ModelAndView("view","key","value");
```

NOTE - Processed data can be either inserted by http request or session manually by method parameters. Except the data kept in the session object remaining all will be in request scope.

STEPS TO CREATE SPRING MVC USING ANNOTATION

1. We need to register Dispatcher Servler by using Dynamic Servlet Registration with help of **ServletContainerInitializer**
2. Spring container internally uses a class SpringServletContainerInitializer which is an implementation **ServletContainerInitializer**

   NOTE - This class is already added in the jar file which search for implementation of WebApplicationInitializer interface.
   For web.xml based spring application dispatcher servlet create WebApplicationContext object inside the init method by loading the spring configuration file and register that container with it.
   But by using the spring mvc application without xml dispatch servlet creates **AnnotationConfigWebApplicationContext** container by taking support of spring configuration class.

3. After that this container we need to suppy servlet context by this Dispatcher Servlet will location and start Spring Container.

```
servletlogicalname-servlet.xml alternative

@EnableWebMvc
@Configuration
@ComponentScan(basePackages="com.cts")
public class SpringAppConfig{
    @Bean
    public ViewResolver createVR(){
        InternalResouceViewResolver vr = new InternalResouceViewResolver();
        vr.setPrefix("/WEB-INT/pages");
        vr.setSuffix(".jsp");
        return vr;
    }
}
```

4. In this class we have to perform following task
   a. Create the object of Dispatcher servlet manually
   b. Create spring container object of type AnnotationConfigWebApplicationContext by supplying the spring configuration class.
   c. This spring container need to register with dispatcher servlet.
   d. Provide the url pattern to loadOnStartUp.

```
              alternative for web.xml for registering frontcontroller

public class DSInitazer implements WebApplicationInitializer{
    public void onStartup(ServletContext ctx) throws ServletException{
        AnnotationConfigWebApplciationContext springContainer = AnnotationConfigWebApplciationContext();
        springContainer.register(SpringAppConfig.class);

        DispatcherServlet ds = new DispatcherServlet(springContainer);
        ServletRegistration.Dynamic reg = ctx.addServlet("myds",ds);
        reg.setLoadOnStartup(1);
        reg.addMapping("/");
    }
}
```

SPRING BOOT

Spring boot is a spring platform that mainly focus convention over the configuration. It is not a software, it is a spring development platform. It provides Rapid Application Development for spring. With this we can get rid of all the xml related configuration. Spring boot supports auto configuration. With the help of spring boot we can avoid lots of boilerplate code, so that we can focus of business logic. It provides cloud support and third party support to develop micro-services.

NOTE - Spring boot provides integrated tomcat server. Spring boot makes it easy to create a spring application as a standalone application that are ready to run.

Ways to develop spring boot application:
- By installing SpringToolSuite plugin from eclipse marketplace
- By installing entire STS software(Eclipse with STS plugin configured).
- By using spring initializer(start.spring.io)
- By using maven to install spring boot dependency.

SPRING BOOT SETUP

Using Spring tool suite to create spring boot

- Click on the create new Spring starter project and give the project name, artifact name and then add Spring web in the package and click finish.
- Add a controller class by annotating **@Controller** inside **src/main/java** folder
- All the ***.jsp** file should be under **src/main/webapp.** This is a public folder and can be accessed from outside. In order to make it private we need to create a folder and put it in the new folder like **src/main/webapp/pages/*.jsp.**
- Add the jsp related capabilities. For that we need jasper related dependency in maven. In order to verify the version we need to check the tomcat version in maven dependency.
- In **src/main/application.properties** we need to add

```
spring.mvc.view.prefix=/pages/
spring.mvc.view.suffix=.jsp
```

- To change port under application.properties

```
server.port=8081
```

- When we create a spring boot application there should be a main method with annotation **@SpringBootApplication** and it acts as a Configuration file and register the controller in the same package.
- If the controller is in different package use **@ComponentScan**

```
@SpringBootApplication
@ComponentScan(basePackages = "p2") // if the controller is in another package
public class MySpringbootApplication{
    public static void main(String[] args){
        SpringAplication.run(MySpringbootApplication.class,args);
    }
}
```

Using start.spring.io to create spring boot

- Go to the website and give the project name and artifact id.
- And click download to get the project in a zip file.
- Extract the file and open it in eclipse using import command.

Using maven to create boot

- Create a new maven simple project(Select checkbox for skip the arc type) and give the project name, artifact Id and other details.
- In the generated pom.xml we need to add one parent tag and two dependencies.
  pom.xml

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.4.RELEASE</version>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groudId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>2.2.4.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.apache.tomcat</groudId>
        <artifactId>tomcat-jasper</artifactId>
        <version>9.0.30</version> <!-- As per tomcat jar file version -->
    </dependency>
</dependencies>
```

- Place an application.properties under src/main/resource folder and mention the prefix and suffix.
- Create a controller class inside **src/main/java**
- Create a class with main method in the same package and annotate this class with @SpringBootApplication like this snippet.
- Create .jsp file inside **src/main/webApp**.

HANDLING CLIENT DATA IN SPRING MVC

- **@RequestMapping** - If you want to specify the type of method for the request under RequiredMapping annotation use the function method to specify it.

```
@RequestMapping(value="/hello",method=RequestMethod.GET)
```

- **@RequestParam** -
  - We can take paramters in RequestHandling method by just give the **same name** of the DOM element as the paramater name.
  - If you want to use **different name** from the DOM element's name then use the annotation @RequiredParam("name") and if the DOM element name is the same as the Request name then the it will be injected.
  - It can also be used to insert a whole **bean class**.

```
@RequestMapping(value="/hello",method=RequestMethod.GET)
public String helloStudent( @RequestParam("name") String studentName, int rollNo, MarkList marks){

}
```

- **@PathVariable** - From client if we send the data without the logical name and with the help of request uri then it's called PathVariable.

```
href="addstudent/100" instead of href="addstudent?marks=100"
```

There are not effective methods in spring boot to read the path variable. Through the RESTful webservices and spring mvc framework we can inject pathvariable to the request handling method using @PathVarialble.

```
@RequestMapping(value="/hello/{name}")
    public String helloStudent( @PathVariable("name") String studentName){
}
```

SPRING MVC FORM VALIDATION

We need to validate a form page inside the client application as well as server side programming. When a valid value comes from the client then only service layer method should be called by the controller.

We can perform the validation by 3 ways

1. custom validation class
2. annotation
3. by using both

Validation by using custom validation class

- First we need to create a *form backing class.* It is a normal java class with variables for those many input field we require in our web page.

- In the controller class we define a method that will launch a form page. Here we need to bind form backing class object with the form page by putting form backing class object inside model object and giving a name to that object.

```
@RequestMapping("/")
public String newStudent(Model m){
    StudentBean stu = new StudentBean();
    m.addAttribute("student",stu);
    return "home"
}
```

NOTE - The data that is kept in model object with a name is known as model attribute.

- Develop a form page and we should use **spring mvc jsp form** tag to bind the model attribute with the input field. It is inject data to jsp from java class.
  - Spring mvc given jsp tags we need to import in our form page jsp taglib directive.

```
<%@ taglib prefix="f" uri="http://www.springframework.org/tags/form" %>
<f:form action="addstudent" modelAttribute="student" > <!-- name of the attribute of the request method -->
<f:input path="roll"/>
```

- In order to display form *validation error message* beside all the input controls errors in f.

```
<f:input path="roll"/> <f:errors path="roll"/>
```

- Create a custom validator class by implementing **Validator** interface and write all the validation related logic in this class. This class should be annoted with **@Component** which will make it as a spring bean class.
  - support() - validates the form backing class
  - validate() - logic for validation

```
@Component
public class MyValidator implements Validator{
    public boolean supports(Class<?> cl){
           return StudentBean.class.isAssignableFrom(cl);
    }

    public void validate(Object target, Errors errors){
         //validation login
         errors.rejectValue("roll(path)","resoucebundle-key","defaultmessage");
         //for rejecting
    }
}
```

rejectValue() -

1. first parameter is the field name.
2. key of the resource bundle file if applicable.
3. default message if not using resource bundle file.

- Define the variable fo **MyValidator** class inside controller class and annotate it **@Autowired**. Use this variable inside the request method to validate the input fields. **BindingResult** is an interface that contains all the errors from the form.

```
@Autowired
MyValidator val;

@RequestMapping("/addStudent")
public String newStudent( ModelAttribute("student") StudentBean student,BindingResult br){
    val.validate(student,br);
    if(br.hasErrors()){
        return "home";
    }
    return "success"
}
```

ResourceBundle file

To reuse the validation form we use resource bundle and we also use it for spring internationalization. It is a property file with the name **src/main/resources/messages.properties .** In this file we keep the error message code and it's description in the form of key value pair.
We can pass dynamic value using placeholder **{index}.** The placeholder should be filled in the third argument as an object array.

```
roll.invalid=The roll should contain {0} digits
errors.rejectValue("roll","roll.invalid",{"6"}","defaultmessage");
```

Validation by using annotation

We use hypernate based annotation to validate the form field. To use this we need to add hibernate-validator in pom.xml. The annotation belonging to this api need to applied on the top of form backing class field.

Some of the validator annotation are
- @AssertTrue
- @AssertFalse
- @DecimalMax("<double>") - for a double datatype field and the value must be <= given value.
- @DecimalMean()
- @Min()
- @Max()
- @NotNull
- @Null
- @Size(min=5,max=10) - length of string
- @Pattern("regex")
- @Past - for Date variable that should be in the past
- @Future - for Date variable that should be in the future
- @CreditCardNumber - to check valid credit card no
- @Email - validate email

We can supply error messages to all these annotation if the validation fails such as @AssertTrue(messages="it should be true")
Inorder to activate it annotaion based be need to apply @Valid.

Type Mismatch**

If you wanna give a custom message when the input type is mismatch you can give the attribute **typeMismatch in the messages.properties.**

```
typeMismatch.roll=message
typeMismatch.student.roll=message
```

Date Object**
Inside **controller class** to parse the data from jsp input

```
@InitBinder
```

```
public void func(WebDataBinder wdb){
    SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yy");
    CustomDateEditor cde = new CustomDateEditor(sdf,true);// predefined property editor by spring
    wdb.registerCustomEditor(Date.class(Type),"dob",cde);
}
```

SPRING I18

- i18 is used to develop application so that it va be adapted to various languages and region withough chagning source code.
- Localization is adpating software for specific language by translating text and adding locale specific component.
- Using i18 user preferred lanuage is set for the view of the web application.
- By using i18 we render labels in webpages in different languages.
- It improves the experience of customers and vistors.
- While working with i18 we need to deal with multiple resource bundle file.(messages.properties) one for each language including a base language. This file name must follow one of the following rules
    1. messages_languagecode_countrycode.properties
    2. messages_languagecode.properties
- In this property file the keys will be same but the value will be as per the selected locale

Locale

It is a geographic or political region that share some language and other information. It can be either *language_code* or a combination of *language_code+country_code* or a combination of some more information.

NOTE - i18 always deals with presentation part of our application. It is no way related with business layer logic and data access layer logic.

- When request sent to change language at server side a ResourceBundle file of the selected lanuage will be loaded and value of the key will be take then and will be replaced.
- We need to register LocaleResolver interface and **LocaleChangeInterceptor** in our application.
- **LocaleChangeInterceptor** allows to change the locale for every request with the respect to the controller based on additional request parameter that is configured.
- It reads the additional request parameter and creates the object java.util.Locale and adds it to the session.

```
LocaleChangeInterceptor lci = new LocaleChangeInterceptor();
lci.setParamName("lang");
```

- Based on the lang value only locale changed for every request.
- **SessionLocalResolver** is an implementation of **LocalResolver** that reads the Locale object from the session object and loads the appropriate resource bundle file.
- If there is no any addition parameters found in session it will choose the default locale configured or http request header *accept-language* header value.

```
SessionLocaleResolver slr = new SessionLocaleResolver();
slr.setDefaultLocale(Locale.US);
```

- We need to register above both classes as spring bean.

Step to implement i18

- Register the **LocalChangeInterceptor** and **SessionLocaleResolver** class as a spring bean.
- Define the two methods in the spring configuration class(@SpringBootApplication annotated class or @Configuration annotated class)
- Annotate both method with @Bean and return LocalChangeIntercepter and SessionLocaleResolver objects.
- For SessionLocalResolver set the Bean Id to be localeResolver **@Bean("localeResolver").**
- Register the **LocalChangeInterceptor** object with the **InterceptorRegistry** for this we need to implement **WebMvcConfigurer** interface inside the spring configuration class and override **addIntercepters()** method

    AppConfig.java

```
@Configuration
public class AppConfig implements WebMvcConfigurer { //or in the springbootapp class
    @Override
    public void addInterceptors(InterceptorRegistry registry){
        registry.addInterceptor(getLci());
    }
}
```

NOTE - In spring mvc interceptors are like servlet filters which will intercept each request and response to a request handling method

- Create multiple resource bundle files (messages.properties) for each locale including a base file under **src/main/resources** folder.
- Inside the form page display all the labels using spring supplied tag for that we need to import spring tag libraries using tag lib directive in **jsp file**.

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags/" %>
<h1><spring:message code="label.welcome"/><h1>
```

- In order to load the appropriate resource bundle we need to send a request along with an additional request paramters like ***lang="fr"*** in the localeResolver bean method.

JPA

JPA is an api given by the sun microsystem to develop data access layer in ORM approach. It is a data access layer framework that works on the top of jdbc and provides us an object oriented way to perform CRUD operation with the database.

Java Persistence

Three common things in any application
- Storing data
- Processing data &
- Presenting data

  For processing and presenting data we represent that data in object oriented fashion and for storing and persisting the data we represent that in relational fashion.

JSTL

Jsp tag lib which provides us with useful java functionality. We need **jstl1.2.jar** to use the JSTL file using maven**.**

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

Library url	prefix
/core	c
/sql	sql
/xml	xml
/fmt (format)	fmt
/functions	fn

Steps to use JSTL tags in jsp:
1. Add the JSTL related jar file.
2. Import the tag library using the tag lib directives with a prefix
3. By the help of Expression language use the tag.

Core tag lib:

It is a collection of some general purpose jsp tags. These are used to create, get, remove scoped data. This tag can also be used to take a decision, iterate o items etc.

```
<c:set var="x" value="100"/>
<c:when test="${x>10}" ></c:when>
```

REST API

Web service is an implementation of Service oriented architecture. A service oriented architecture is an architectural pattern in computer software design in which application components provide services to other components via a communication protocol mostly over a network.

Web applications can also be accessed over the web but there are some differences between web applications and web services.

- Web applications are meant for end users and to be accessed by the browser in human readable format. Whereas web service are meant for application to access the data in the form of XML, JSON, plain text etc. format.
- Any application can access the web services to use some data or to perform some tasks whereas web services cannot access the web application in order tc get some data.
- Web application maintains the user session but web services are always stateless.
- A single web service can be reused by different kind of web applications, whereas web applications are not meant for reusing.

NOTE - By using web services we can communicate with heterogeneous web services such as java, python etc.

TYPE OF WEB SERVICE

- SOAP
- RESTful

SOAP Web services

It is an XML based protocol to communicate between the client and server side application. Since it is an XML based protocol it is platform independent and language independent. i.e. our client application can be developed in any language and can run on any platform. Similarly our server side application as well.

Create a web services using SOAP is a heavy weight web service. Because in addition of creating the web service classes there is a need to create some extra bind classes, some extra XML files and to use some third party API to parse the XML to expose and consume the web services.

RESTful Web services

- In order to make the web service to light weight another architectural style is proposed and it is named as REST
- It states that create the web services and expose them through the http protocol and allow the client to consume it without any binding classes and XML fil
- Based on this REST architectural style, Sun Microsystem released JAX-RS API to create RESTful web services in java. It is one of the service API in J2EE platfor

NOTE - In a RESTful web service a class is called as root-resource and its each method of this class(Actual service) is called as resource.

RULES TO MAKE JAVA CLASS TO WEBSERVICE

- Each resource of our web service class should contain an **addressable URI** that can be accessed through http protocol
- While exposing web service we need to take support of http protocol methods
  GET - If you want to create a service that returns some information and some data to the client.
  POST - If you want to create a service for storing/inserting some data given by the client.
  PUT - If you want to update the data in the server side by the new data from the client.
  DELETE - It is used to perform delete operation.
- Every RESTful web service class must be public and it must contain a **default constructor**.
- A web service method should return the data in one of the following format
    - application/json
    - application/xml
    - text/html
    - text/plain
- General to make a java class as RESTful web service we need to use JAX-RS API with one of its implementation **Jersey** or **RestEasy**. Spring framework has itse provided with spring implementation to make a spring controller bean as a RESTful API.
- Spring MVC framework is designed in such a way that we can convert any controller bean as a RESTful web service very easily.

  NOTE - if you make a controller bean as a RESTful controller then the request handling methods returns the restful represented object instead o returning model and view object.
- While returning the response from the controller bean through the dispatcher servlet Spring REST makes use of **http message converter** object for converti Java objects to JSON or XML or vice versa.
- To make a request handling method as a RESTful web service we need to apply **@ResponseBody** annotation on the top of request handling method.
  NOTE - Naming convention for controller is **<name>Resource.java**

- Instead of putting @**ResponseBody** on every method of a controller class we should use **@RestController** instead of @Controller annotation.

```
@RestController
public class MyResource{


    @GetMapping("/getStudents")
    public Student getStudents(){
        // do operations
        return student; // return data
    }
}
```

- Other important annotation
    - @PathVariable - used to inject the URI path parameter to the request method parameter.
    - @RequestBody - used to convert client data in the form of XML or JSON format to the Java object and inject to request handler.

MediaType

This class is used to specify the request handler method that which type the format the request data is coming and to which format the Response needs to generated.

```
@GetMapping("/getStudents", produces = MediaType.APPLICATION_JSON_VALUE, consumes=...)
```

- ResponseEntity<data-type> - To send status code along with data
- HttpHeaders - To send the header information

Spring REST Client

We can develop a RESTful client application to call a web service's API endpoint. Spring framework has given a class RestTemplate to access the API endpoi in our java application in very easy manner.

```
public class MyRestClient{
    public static void main(String[] args){
```

```
    public static void main(String[] args){
        RestTemplate rt = new RestTemplate();
        /* getForObject, postForObject and so on */
        Student result = rt.postForObject("<url>",data,Student.class); // returns T;
        rt.getForEntity(); // returns ReponseEntity
    }
}
```