**Spring MVC**

1 Framework building web aplication in java

2 based on model view controller DP

3 leverage features of Core spring framework(IoC,DI)

**MVC**

Web browswer -> Front Controller /Dispatcher servelt (model)->Controller (model)-> view template ->Web browswer

**Benfits**

1 building eb app UI s in java

2 reusable UI componenetes

3 manage application state of web request

4 process form data: validaion,conversion

5 felxible config for view layer

**Compenents of a Spring MVC application**

1 set of web pages to layout a UI compenets (web pages)

2 collection of Spring beans (controller,service,etc)

2 spring Configuration (xml,anoation,java)

**SpringMVC front Controller**

Front Controller/Dipacther deveopled by Spring delgetes incoming req

We only create model,view ,controller

**Controller**

1 handel the request

2 place data in model

3 send to appriate view

**Model**

Contains data

Bean can be used as model

Place data in model

Data can be anu java object/collection

**View**

Display data

Jsp+jstl

**Spring MVC Configuration Process**

**Part 1**

adding a config file to WEB-INF/web.xml

1 config Spring Dispatcher SErvelt

2 setup URL apping to Spring Dispacher SErvlet

**Part 2**

adding a config file to WEB-INF/SpringDemo-servelt.xml

3 add support of Compnet Scanning

4 add supprot for conversion,fomartiong ,validation mvc annoation

5 config Spring View resvoler

**Build**

1 Java EE prespective

2 dynamic project

3 add all spring jars to lib and JSTL libs also

Config Requiement Changes

In web.xml

<!-- Spring MVC -->

<!-- step 1 : Config SpringMVC Dispatcher SErvelt -->

<servlet>

<servlet-name>dispatcher-servlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/SpringDispatcherServelt</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!-- step 2 : set up url mapping for Spring MVc dispatcher Servelt -->

<servlet-mapping>

<servlet-name>dispatcher-servlet</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

In Dispatcher Servelt basic required

<!-- Step 3 add Support for componenet Scanning -->

<context:component-scan base-package=*"com.sample.controller"*></context:component-scan>

<!-- Step 4 Add support for Conversion , formatting and validation and support -->

<mvc:annotation-driven/>

<!-- Step 5 Define Spring MVC view Resolver -->

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

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

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

</bean>

**Deveolping Controller and View**

1 Request mapping for request path

2 Controller to handle the request

3 froward to view template

**Deveolpment Process**

1 Create contrller

2 Define Controller method

3 Add request mapping to controller method

4 retuen view Name

5 deveolp View page

**Step 1 create class and annotate with @Controller**

@ controller inherits from @component support scanning

@RequestMapping actual path to map

Annotaions maps a path to a method name

**Form data in Spring**

**To read html feild data in jsp ${param.feildname}**

**Adding data to spring model**

1 model is container of our applicaton data

2 in controller can add anytinh in model string, object, db info

3 view page can access data from model

If need to read form data in controller then requst has to parameter of method

**Model is container holds form data**

Here are the steps on how to access static resources in a Spring MVC. For example, you can use this to access images, css, JavaScript files etc.

Any static resource is processed as a URL Mapping in Spring MVC. You can configure references to static resources in the spring-mvc-demo-servlet.xml.

**Create “resources” directory in web content**

**Step 1:** Add the following entry to your Spring MVC configuration file: **spring-mvc-demo-servlet.xml**

You can place this entry anywhere in your Spring MVC config file.

<mvc:resources mapping="/resources/\*\*" location="/resources/"></mvc:resources>

**Step 2:** Now in your view pages, you can access the static files using this syntax:

<img src="${pageContext.request.contextPath}/resources/images/spring-logo.png">

You need to use the JSP expression **${pageContext.request.contextPath}** to access the correct root directory for your web application.

Apply the same technique for reading CSS and JavaScript

**Reading HTMl data using @RequestParam annotaion**

**@RequestParam help to read form data from request and automaaticaly bind it to variable coming to your method**

**Add Contrller @RequestMapping**

1 Adding Request mapping to controller serves as parent mapping for controller

2 all request mapping on methods in the controller are realative

3 similar to folder directive structure

**Advantge ambigouity over come**

**Spring Form tags**

1 build blocks of web page

2 form tags are configurable and resuable for web page

**Binding Data**

1 Spring forms make use of data binding

2 Automaticcaly setting/retriving data from java object / bean

Spring form tags will geneatre html code

To refer Spring form tags in jsp

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

**Form with input field**

**Sample 1**

Spring from with input field send to controller from their to success page

Showing Form

In spring controller

\* befor showing from must add a model attirbute(bean)

\* that will be the bean to hold form data from data binding

**Deveolpement process**

1 create Student Class

2 Create Student Controller class

3 create HTML class

4 create form processing codes

5 Create confirmation page

<form:form action=*"processform"* modelAttribute=*"student"*> // student is the same as model attribute

First Name:<form:input path=*"firstName"*/>

<br><br>

Last Name: <form:input path=*"lastName"*/>

<br><br>

<input type=*"submit"* value=*"submit"*/>

</form:form>

path=*"firstName" binds form filed propery of an entity When form loaded it uses getFirstname(); When it is submitted it uses setFirstName();*

@RequestMapping("/showform")

**public** String showForm(Model model)

{

//create object

Student theStudent=**new** Student();

model.addAttribute("student",theStudent);

**return** "student-form";

}

**- model is used to pass data btw controller and view**

- Student theStudent=**new** Student(); model.addAttribute("student",theStudent); student => attribute name same as to be used in form for binding

The student value of attribute

**-@ModelAttribute bind form data to Object**

@RequestMapping("/processform")

**public** String processForm(@ModelAttribute("student") Student student) {

System.***err***.println(student.getFirstName());

**return** "student-confirmation";

}

In jsp conirmation we can data in this manner

**${student.firstName } --- ${student.lastName }**

Drop Down List Spring MVC

<form:select> is the tag for Spring Form

<form:select path=*"country"*>

<form:option value=*"IND"* label=*"India"*/>

<form:option value=*"PAK"* label=*"Pakistan"*/>

<form:option value=*"USA"* label=*"America"*/>

<form:option value=*"UK"* label=*"London"*/>

</form:select>

**This will bind .If i want read it from java side for options Create a has map n intialize it during object creation**

Call it in form like below

<form:options items=*"*${student.stateoption}*"* />

**In case I want to read data from properties files Changes required so it cant be hardcoded in java file**

**1 Create a properties file to hold the countries. It will be a name value pair. Â Country code is name. Country name is the value.**

**countries.properties**

Put data like

BR=Brazil FR=France CO=Colombia IN=India

Note the location of the properties file is very important. It must be stored in WEB-INF/countries.properties

1. **Update header section for Spring config file**

We are going to use a new set of Spring tags for <util>. As a result, you need to update the header information in the Spring config file.

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:xsi = *"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:util=*"http://www.springframework.org/schema/util"*

xsi:schemaLocation = *"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc.xsd*

*http://www.springframework.org/schema/util*

[*http://www.springframework.org/schema/util/spring-util.xsd"*>](http://www.springframework.org/schema/util/spring-util.xsd%22%3e)

**3. Load the country options properties file in the Spring config file. Bean id: countryOptions**

<util:properties id="citeOptions" location="classpath:../countries.properties" />

**4. Inject the properties values into your Spring Controller: StudentController.java**

@Value("#{citeOptions}")

private Map<String, String> cityOptions;

1. **Add the country options to the Spring MVC model. Attribute name: theCountryOptions**

@RequestMapping("/showForm") public String showForm(Model theModel) { // create a student object Student Student theStudent = new Student(); // add student object to the model theModel.addAttribute("student", theStudent); // add the country options to the model theModel.addAttribute("theCountryOptions", citeOptions); return "student-form"; }

**6. Update the JSP page, student-form.jsp, to use the new model attribute for the drop-down list: theCountryOptions**

<form:select path="country">

<form:options items="${theCountryOptions}" />

</form:select>

**Spring Form with Radio Buutons**

Java<form:radiobutton path=*"lanuage"* value=*"Java"*/>

Ruby<form:radiobutton path=*"lanuage"* value=*"Ruby"*/>

Python<form:radiobutton path=*"lanuage"* value=*"Python"*/>

Sql<form:radiobutton path=*"lanuage"* value=*"Sql"*/>

**If we want populate radio button like drop down fron java side**

Create an hash map

**private** LinkedHashMap<String, String> domianOption;

Put vaules in contructor creat getter method

Use this to iterate in front end

<form:radiobuttons path=*"domain"* items=*"*${student.domianOption}*"* />

**Spring Form with Check Box**

Windows <form:checkbox path=*"oprSystem"* value=*"windows"*/>

Linux <form:checkbox path=*"oprSystem"* value=*"linux"*/>

Ubuntu <form:checkbox path=*"oprSystem"* value=*"ubnutu"*/>

In java it as select mulptiple option if has selected multiple check box So array of String add approaiate get/ set method

More then single field acttuly a collection

While printing result we need to use loop over for this kind case to show multpile data Use JSTL c tag to use iteration code

**Appliying Built In Validation Rules for Spring Form**

Need for validation

-Required fileds

-range

-formate

-custom

**We can use Java Standrad Beam Validation API**

**It defines metadata model and API for entity validation**

-it is not tied to either web or presentaion tier

-availbel for server side and also clinet side using JAVAFX

- vaidation only above spring 4 and above

-add jar and can use inbuilt method

**@NotNull,@Min,@Max,@Size,@Pattern,@Future/@Past**

**Set UP Devlopemnt Environment**

-java Bean Vlidation API

- Hibernate not only provide ORM it also provide many things like vlaidation

Downlaod Hibernate Validator

Add to project

**Create form where lastName is required**

Steps

Add Vlaidation Rules to Customers Class

Display error in HTML page

Perform validation in controller class

Upload confirmation page

@NotNull(message="is required")

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

**public** String lastName;

Can’t be null min one char is required

Last Name\*: <form:input path=*"lastName"*/>

<form:errors path=*"lastName"* cssClass=*"error"* />

In jsp along with <form:input> tag mention <form:error> tag so bind error result can be showhed here based on path name cssClass for Ui moidfication to msg

Diplay if error msg is set

@RequestMapping("/processform")

**public** String processForm(@Valid @ModelAttribute("customer") Customer customer, BindingResult bindingResult) {

**if**(bindingResult.hasErrors())

{

**return** "customer-form";

}

**else**

{

**return** "customer-confirmation";

}

}

@Valid annotaions specifies perform validation rulse on Cutomer Object and also its result willl be binded in BindingResult

If any error in binding result return to form if not proceded

We can add seom print debug msgs

If we want remove whitespace Adv spring soloution

Spring Validation along with @InitBinder

Need to trim whitespace from input field

When performing Spring MVC validation, the location of the BindingResult parameter is very important. In the method signature, **the BindingResult parameter must immediately after the model attribute**.

If you place it in any other location, Spring MVC validation will not work as desired. In fact, your validation rules will be ignored.

**Defining @RequestMapping methods**

@RequestMapping handler methods have a flexible signature and can choose from a range of supported controller method arguments and return values.

The Errors or BindingResult parameters have to follow the model object that is being  
bound immediately ...

@InitBinder works as pre-processor

It will pre process each web req coming to our controller it will excute first

Method annoted with @InitBinder is executed 1st

We use it remove leading and trailing white space

If it has only white space trim to null

StringTrimEditor class of Java Remove white space leading and trailing

True constructor means trim to nullif entirly space

Use databinder to register customeditoer for example do for String class

Pre-process every string form data remove leading and trailing space if string only has white space trim it to null

Validation Number ranges @min @max

Add validation fo free pasees

Only range between 0 to 10

@Min(value=0,message="must be grater then or equal to zero")

@Max(value=10,message="must be less then or equal to 10")

**private** **int** freePass;

Regular Expression Validation

RE is a sequnce charcater that defines search pattern

Pattern used to find search string

Postal code

Only 5 char/ digitss

Applya eXpression

Addd RE validation to Customer Class

@Pattern(regexp="^[A-Za-z0-9]{5}",message="only 5 char/ digits allowed")

**private** String postalCode;

How to make integer Filed required Ex free pass

By adding @NotNull wont fix bcz

Its int we will get this error

To resolve it

So instead using int we can use Integer how it will fix means we have an inibinder which trim sapce till make it null so Intreger can have null

@NotNull(message="is required")

@Min(value=0,message="must be grater then or equal to zero")

@Max(value=10,message="must be less then or equal to 10")

**private** int freePass;

Failed to convert property value of type java.lang.String to required type int for property freePass; nested exception is java.lang.NumberFormatException: For input string: ""

@NotNull(message="is required")

@Min(value=0,message="must be grater then or equal to zero")

@Max(value=10,message="must be less then or equal to 10")

**private** Integer freePass;

**Handling String I/P for Integer fields**

1 Create custom error Meassge propertie file

2 load custom proprty in spring config file

**Create properties file under**

Create folder resources in SRC / are create under WEB-INF

ErrorType.SpringModuleAttributeName.Filedname=custom error message

Same format in properti fila also

typeMismatch.customer.freePasses=Invalid Number

**If its under WEB-INF in spring conf: Add**

<bean id=*"messageSource"*

class=*"org.springframework.context.support.ReloadableResourceBundleMessageSource"*>

<property name=*"basename"* value=*"/WEB-INF/messages"* /> // read from WEB-INF

</bean>

**For SRC**

<property name=*"basename"* value=*"classpath:/resources/messages"* />

**For deep error inspect binding result object by adding debug message in Customer Contoller**

Field error in object 'customer' on field 'freePass': rejected value [sd]; codes [typeMismatch.customer.freePass,typeMismatch.freePass,typeMismatch.java.lang.Integer,typeMismatch]; arguments [org.springframework.context.support.DefaultMessageSourceResolvable: codes [customer.freePass,freePas]; arguments []; default message [freePass]]; default message [Failed to convert property value of type 'java.lang.String' to required type 'java.lang.Integer' for property 'freePass'; nested exception is java.lang.NumberFormatException: For input string: "sd"]

**customer.freePass,freePas => we are overidng error code with own msg**

**Custom Vlidation With Spring MVC**

Consider Form Course

With first and Last Name and Course code and it start with LUV

Write our own cuatom Bussiness Rules: Cource code msut start with LUV

Spring MVC call our custom Validation

Custom return boolen results true/false

**We crate Custom Java Annotaion from Scracth**

**Like @CourceCode**

**Development Rules**

1 create custom validation rules

2 add validation to customer class

3 display error msg from html

4 update confirmation page

**1 create custom validation rules**

A create @CourceCode annotaion

B Create CourceCodeConstraintValidator

**Create Java Annotaion**

Ex Annnotaions

@CourceCode(value="TOP",message="Must Start with Top")

**private** String courceCode;

We should applay to our field like this

**Create pkg for validation sepratley**

New -> Annotaion

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

**import** javax.validation.Constraint;

**import** javax.validation.Payload;

// Adding Constraint on top of a contraint which need to validate Rules of Custom class wr bussiness rule written in CourceCodeConstriantValidator

@Constraint(validatedBy=CourceCodeConstriantValidator.**class**)

// wr can we apply we can apply on method and feild

@Target({ElementType.***METHOD***,ElementType.***FIELD***})

// how long annotaion will stored/Used ->reatain this annotaion in java class file process it at runtime by JVM

@Retention(RetentionPolicy.***RUNTIME***)

**public** **@interface** CourceCode { // **@interface** Syntax

//define attributes to pass it our annotaions

// define default cource code

**public** String value() **default** "LUV";

// define default error msg

**public** String message() **default** "Must start With LUV";

// define default groups -> can grp related constraints

**public** Class<?>[] groups() **default** {};

// define default payloads -> provide custom deatils about validation failure(security level,error code )

**public** Class<? **extends** Payload>[] payload() **default** {};

}

**Give Annotaions name syntax and extra addition of syntax on top now define parmeters which annotaion will have**

**Then add method decalrtion for attributes if dnt provide a vaule by user will use default value**

**public** String value() **default** "LUV";

Same for message also

**public** String message() **default** "Must start With LUV";

**Now Annotaion is Completd now CourceCodeConstriantValidator Class which has bussiness ruels:**

**It gives true/ false if satifeis rules**

**import** javax.validation.ConstraintValidator;

**import** javax.validation.ConstraintValidatorContext;

**public** **class** CourceCodeConstriantValidator **implements** ConstraintValidator<CourceCode, String>{

// define CoursePreFix that will validate against it

**private** String coursePrefix;

@Override

**public** **void** initialize(CourceCode theCourceCode) {

coursePrefix=theCourceCode.value(); // assaign the value from passed in form our annotaion

}

// theCode is data entered by user

//ConstraintValidatorContext constraintValidatorContext -> we can add additional error msgs

@Override

**public** **boolean** isValid(String theCode, ConstraintValidatorContext constraintValidatorContext) {

// need to check for null or else we will get null poineer exception during runtime

**boolean** result;

**if**(theCode!=**null**)

{

result= theCode.startsWith(coursePrefix);

}

**else**

{

result= **true**;

}

**return** result;

}

}

Is it possible to integrate multiple validation string in one annotation? For example, validate against both LUV and TOPS.

**Answer:**

Yes, you can do this. In your validation, you will make use of an array of strings.

Here's an overview of the steps.

1. Update CourseCode.java to use an array of strings

2. Update CourseCodeConstraintValidator.java to validate against array of strings

3. Update Customer.java to validate using array of strings

**1. Update CourseCode.java to use an array of strings**

Change the value entry to an array of Strings:

// define default course code

public String[] value() default {"LUV"};

**1. Update CourseCode.java to use an array of strings**

Change the value entry to an array of Strings:

// define default course code

public String[] value() default {"LUV"};

**2. Update CourseCodeConstraintValidator.java to validate against array of strings**

Change the field for coursePrefixes to an array

private String[] coursePrefixes;

**3. Update Customer.java to validate using array of strings**

@CourseCode(value={"TOPS", "LUV"}, message="must start with TOPS or LUV")

private String courseCode;