# SPRING MVC FRAMEWORK COMPONENTS

Instructor:

# Table Content

# Learning Goals

- **After the course, attendees will be able to:**

| |
|---|
| Understand Spring Web MVC Framework and its core technologies. |
| Know how to write a Web application with Spring MVC Framework. |

Section 1

# SPRING @REQUESTPARAM ANNOTATION

# A Simple Mapping

- We can use *@RequestParam* to extract query parameters, form parameters, and even files from the request.
- In this example, we used *@RequestParam* to extract the *id* query parameter.

```
@GetMapping("/api/foos")
public String getFoos(@RequestParam String id) {
        return "ID: " + id;
}
```

- A simple GET request would invoke *getFoos*:

```
http://localhost:8080/api/foos?id=abc
----ID: abc
```

# Specifying the Request Parameter Name

- **Sometimes we want these to be different, though.**
- Fortunately, **we can configure the *@RequestParam* name using the *name* attribute**:

```
@PostMapping("/api/foos")
public String addFoo( @RequestParam(name = "id") String fooId,
                      @RequestParam String name) {
        return "ID: " + fooId + " Name: " + name;
}
```

- We can also do *@RequestParam(value = "id")* or just *@RequestParam("id")*.

# Optional Request Parameters

- Method parameters annotated with *@RequestParam* are required by default.
- This means that if the parameter isn't present in the request, we'll get an error:

```
GET /api/foos HTTP/1.1
-----
400 Bad Request
Required String parameter 'id' is not present
```

- **We can configure our *@RequestParam* to be optional, though, with the *required* attribute:**

```
@GetMapping("/api/foos")
public String getFoos(@RequestParam(required = false)
                                                String id) {
        return "ID: " + id;
}
```

# Optional Request Parameters

- **In this case, both:**

```
http://localhost:8080/api/foos?id=abc
----ID: abc
And
http://localhost:8080/api/foos
----ID: null
```

- *will correctly invoke the method.*
- When the parameter isn't specified, the method parameter is bound to null.

# Using Java 8 *Optional*

- Alternatively, we can wrap the parameter in **Optional**:

```
@GetMapping("/api/foos")
public String getFoos(@RequestParam Optional<String> id) {
        return "ID: " + id.orElseGet(() -> "not provided");
}
```

- In this case, **we don't need to specify the *required* attribute.**
- And the default value will be used if the request parameter is not provided:

```
http://localhost:8080/api/foos
---- ID: not provided
```

# Mapping a Multi-Value Parameter

- A single *@RequestParam* can have multiple values:

```
@GetMapping("/api/foos")
public String getFoos(@RequestParam List<String> id) {
        return "IDs are " + id;
}
```

- **And Spring MVC will map a comma-delimited *id* parameter:**

```
http://localhost:8080/api/foos?id=1,2,3
----IDs are [1,2,3]
```

- **or a list of separate *id* parameters**:

```
http://localhost:8080/api/foos?id=1&id=2
----IDs are [1,2]
```

Section 2

# SPRING MVC JSP FORM TAGS

# Introduction

- Spring provides a comprehensive set of data binding-aware tags for handling form elements when using JSP and Spring Web MVC, such as *form tag*, *text fields tag*, *select tag*, *check-box(s)*, *radio box(s)*, *password tag*, *button tag*, *errors tag etc*.
- **Spring's form tag library**

| | |
|---|---|
| •The form tag | •The select tag |
| •The input tag | •The option tag |
| •The checkbox tag | •The options tag |
| •The checkboxes tag | •The textarea tag |
| •The radiobutton tag | •The hidden tag |
| •The radiobuttons tag | •The errors tag |
| •The password tag | •HTML5 tags |

# Spring MVC Form Tags

- To use the tags from this library, add the following directive to the top of your JSP page:

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

- **The form tag**
  - ✓ This tag renders an HTML 'form' tag and exposes a binding path to inner tags for binding.
  - ✓ It puts the command object in the PageContext so that the command object can be accessed by inner tags. All the other tags in this library are nested tags of the form tag.
  - ✓ When the **form is loaded**, spring MVC will class **user.getFirstName() and getLastName()** (getter methods).
  - ✓ When the **form is submitted**, Spring MVC will call **user.setFirstName()** and **user.setLastName()** methods.

# Spring MVC Form Tags

## ❖ Example:

```
<form:form modelAttribute = "user">
    <table>
        <tr>
            <td>First Name:</td>
            <td>
                <form:input path="firstName" />
            </td>
        </tr>
        <tr>
            <td>Last Name:</td>
            <td>
                <form:input path="lastName" />
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="submit" value="Save Changes" />
            </td>
        </tr>
    </table>
</form:form>
```

# The checkbox tag

- This tag renders an HTML 'input' tag with type 'checkbox'.
- **Give an entity class:**

```
Entity
@Table(name = "Preferences")
public class Preferences {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(name = "receive_news_letter")
    private boolean receiveNewsletter;
    private String[] interests;
    @Column(name = "favourite_word")
    private String favouriteWord;

    // constructors and getter/setter methods
}
```

# The checkbox tag

- **Form:**

```
<form:form   method="POST" action="${pageContext.request.contextPath}/subscription"
             modelAttribute="preferences">
<table>
<tr>
<td>Subscribe to newsletter?:</td>
<%-- Approach 1: Property is of type java.lang.Boolean --%>
<td><form:checkbox path="receiveNewsletter" /></td>
</tr>

<tr>
<td>Interests:</td>
<%-- Approach 2: Property is of an array or of type java.util.Collection --%>
<td>Quidditch:
        <form:checkbox path="interests" value="Quidditch" />
    Herbology:
        <form:checkbox path="interests" value="Herbology" />
        Defence Against the Dark Arts:
        <form:checkbox path="interests" value="Defence Against the Dark Arts" />
</td>
</tr>

<tr>
<td>Favourite Word:</td>
<%-- Approach 3: Property is of type java.lang.Object --%>
<td>Magic:
    <form:checkbox path="favouriteWord" value="Magic" />
</td>
</tr>

</table>
</form:form>
```

# Controller

## Controller class:

```
@Controller
public class PreferencesController {

    @PostMapping("/subscription")
    public String subcription(
            @ModelAttribute("preferences") Preferences preferences,
            Model model) {
        LogUtils.getLogger().info(preferences);

        return "newsletter_subscription";
    }

}
```

## Results:

| | |
|---|---|
| Subscribe to newsletter?: | ☑ |
| Interests: | ☑ Quidditch<br>☐ Herbology<br>☐ Defence Against the Dark Arts |
| Favourite Word: | ☑ Magic |
| Submit | |

# The checkboxes tag

- This tag renders multiple HTML 'input' tags with type 'checkbox'.

```
<form:form>
<table>
     <tr>
     <td>Interests:</td>
     <td>
     <%-- Property is of an array or of type java.util.Collection --%>
     <form:checkboxes path="preferences.interests" items="${interestList}" />
     </td>
     </tr>
</table>
</form:form>
```

# The radiobutton and radiobuttons tags

- **radiobutton** tag renders an HTML 'input' tag with type 'radio'.

```
<tr>
<td>Gender:</td>
<td>Male: <form:radiobutton path="gender" value="M" /> <br />
    Female: <form:radiobutton path="gender" value="F" />
</td>
</tr>
```

- **radiobuttons** tag renders multiple HTML 'input' tags with type 'radio'. For example:

```
<tr>
<td>Gender:</td>
<td>
    <form:radiobuttons path="gender" items="${genderOptions}" />
</td>
</tr>
```

# The select tag

- This tag renders an HTML 'select' element.
- It supports data binding to the selected option as well as the use of nested option and options tags.

```
<tr>
    <td>Skills:</td>
    <td><form:select path="skills" items="${skills}" /></td>
</tr>
```

Section 3

# @SESSIONATTRIBUTES OR @SESSIONATTRIBUTE

# Introduction

- When developing web applications, we often need to refer to the same attributes in several views.
- **For example**, we may have shopping cart contents that need to be displayed on multiple pages.
- A good location to *store those attributes is in the user's session*.
- Have **2 different strategies for working with a session attribute**:
  - ✓ **Using a scoped proxy**
  - ✓ **Using the @*SessionAttributes* annotation**

# @SessionAttributes

- **@SessionAttributes** annotation is used to store the model attribute in the session. This annotation is used at controller class level.

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

- **@SessionAttribute** annotation is used to retrieve the existing attribute from session that is managed globally and it is used at method parameter as shown follows.

```
@GetMapping("/info")
public String userInfo(@SessionAttribute("user") User user) {
    //... //... return "user";
}
```

# Controller

```java
@Controller
@SessionAttributes("user")
public class UserController {

    /**
     * Add user in model attribute.
    */
    @ModelAttribute("user")
    public User setUpUserForm() {
        return new User();
    }

    @PostMapping("/dologin")
    public String doLogin(@ModelAttribute("user") User user, Model model) {

        // Implement your business logic
        if ("admin".equals(user.getUsername())
                && "admin".equals(user.getPassword())) {

            return "index";
        } else {
            model.addAttribute("message", "Login failed. Try again.");
            return "login";
        }
    }
```

# Controller

```
    /*
     * Get user from session attribute
     */
    @GetMapping("/info")
    public String userInfo(@SessionAttribute("user") User user) {

        System.out.println("User Name: " + user.getUsername());

        return "index";
    }
}
```

# View

## ❖ Login.jsp

```
<form:form action="${pageContext.request.contextPath}/dologin" method="post" modelAttribute="user">
        <h2 class="text-center">Log in</h2>
        <label style="color: red">${errorMessage}</label><!-- JSP Expression -->
        <div class="form-group">
            <input type="text" name="username" class="form-control"
                                                placeholder="Username" required="required">
        </div>
        <div class="form-group">
            <input type="password" name="password" class="form-control"
                                                placeholder="Password" required="required">
        </div>
        <div class="form-group">
            <button type="submit" class="btn btn-primary btn-block">Log in</button>
        </div>
        <div class="clearfix">
            <label class="float-left form-check-label"><input type="checkbox"> Remember me</label>
            <a href="#" class="float-right">Forgot Password?</a>
        </div>
  </form:form>
```

Section 4

# SPRING EXPRESSION LANGUAGE (SpEL)

# Introduction

- **Spring Expression Language** (SpEL) is a powerful expression language, which can be used for querying and manipulating an object graph at runtime.
  - ✓ **SpEL** supports standard *mathematical operators*, *relational operators*, *logical operators*, *conditional operators*, *collections* and *regular expressions*, etc.
  - ✓ It can be used to *inject a bean* or *a bean property* into another bean.
  - ✓ *Method invocation of a bean* is also supported.

# SpEL Examples

- **Example 1**: The logical operators, (&&) or (||) and not (!), are supported. The textual equivalents can also be used.
  - ✓ (1) First let's define the **MyOtherGlass** POJO:

```
package fa.training.entities;

public class MyOtherGlass {
    private boolean empty;
    private boolean halfEmpty;
    private int volume;
    private int maxVolume;
    private boolean largeGlass;

    public MyOtherGlass() {

    }
    // getter and setter moethod
}
```

# SpEL Examples

- (2) Let's now create our spring configuration file where we define the **smallGlass** bean and the **largeGlass** bean.

```
<bean id="smallGlass" class="fa.training.entities.MyOtherGlass">
    <constructor-arg name="volume" value="5" />
    <constructor-arg name="maxVolume" value="10" />
    <property name="largeGlass"
        value="#{smallGlass.maxVolume ge 20 and
                                smallGlass.maxVolume le 30}" />
</bean>

<bean id="largeGlass" class="fa.training.entities.MyOtherGlass">
    <constructor-arg name="volume" value="5" />
    <constructor-arg name="maxVolume" value="30" />
    <property name="largeGlass"
        value="#{largeGlass.maxVolume ge 20 and
                                largeGlass.maxVolume le 30}" />
</bean>
```

# SpEL Examples

## (3) Test

```
LogUtils.getLogger().info(smallGlass.isLargeGlass());

LogUtils.getLogger().info(largeGlass.isLargeGlass());
```

## Results:

```
[INFO ] 2020-11-14 15:12:47.678 [http-nio-8080-exec-2]
LogUtils - false
[INFO ] 2020-11-14 15:12:47.680 [http-nio-8080-exec-2]
LogUtils - true
```

# SpEL Examples

## ❖ Example 2:

```
<bean id="officeAddress" class="fa.training.entities.Address">
        <property name="number" value = "101" />
        <property name="street" value = "#{'M I Road'}" />
        <property name="city" value = "Jaipur" />
        <property name="state" value = "Rajasthan" />
        <property name="pinCode" value = "#{'302001'}" />
    </bean>

    <bean id="employee" class="fa.training.entities.Employee">
        <property name="empId" value = "1001" />
        <property name="empName" value = "Ram" />
        <!--  Bean reference through SpEL -->
        <property name="officeAddress" value = "#{officeAddress}" />
        <property name="officeLocation" value = "#{officeAddress.city}" />
        <!--  Method invocation through SpEL -->
        <property name="employeeInfo" value = "#{officeAddress.getAddress('Ram')}" />
    </bean>
```

# SpEL - properties files

- Create **DBConfig.properties** file:

```
driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
url=jdbc:sqlserver://localhost:1433;databaseName=DBName
username=sa
password=12345678
```

- **spring-servlet.xml** file:

```
<!- Properties Spring -->
<bean class="org.springframework.beans.factory.config
                                        .PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>classpath:DBConfig.properties</value>
        </list>
   </property>
</bean>
```

# SpEL - properties files

- **spring-servlet.xml** file:
  - ✓ The **context:property-placeholder** tag is used to externalize properties in a separate file.
  - ✓ It automatically configures **PropertyPlaceholderConfigurer**, which replaces the ${} placeholders, which are resolved against a specified properties file (as a Spring resource location).

```
<context:property-placeholder
    location="classpath:data.properties, classpath:DBConfig.properties"
    ignore-unresolvable="true" />
```

  - ✓ Default resource location: src/main/resources.

# SpEL - properties files

- **spring-servlet.xml** file:

```
<!-- DataSourse -->
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.
                          DriverManagerDataSource">
    <property name="driverClassName" value="${driver}" />
    <property name="url" value="${url}" />
    <property name="username" value="${username}" />
    <property name="password" value="${password}" />
</bean>
```

# SpEL Using Annotation

- SpEL expressions begin with the # symbol, and are wrapped in braces: **#{expression}**.
- Properties can be referenced in a similar fashion, starting with a $ symbol, and wrapped in braces: **${property.name}**.

```
@Value("#{19 + 1}") // 20
private double add;

@Value("#{'String1 ' + 'string2'}") // "String1 string2"
private String addString;

@Value("#{20 - 1}") // 19
private double subtract;

@Value("#{10 * 2}") // 20
private double multiply;

@Value("#{36 / 2}") // 19
private double divide;
```

```
@Value("#{36 div 2}") // 18, the same as for / operator
private double divideAlphabetic;

@Value("#{37 % 10}") // 7
private double modulo;

@Value("#{37 mod 10}") // 7, the same as for % operator
private double moduloAlphabetic;

@Value("#{2 ^ 9}") // 512
private double powerOf;

@Value("#{(2 + 2) * 2 + 9}") // 17
private double brackets;
```

# SpEL Using Annotation

- **Example**: The **@Component** annotation for registering the bean and **@Value** for setting values into bean properties.

```java
@Component
public class Address {
    @Value("100")
    private String houseNo;
    @Value("The Mall")
    private String street;
    @Value("Shimla")
    private String city;
    @Value("HP")
    private String state;
    // As SpEL literal
    @Value("#{'171004'}")
    private String pinCode;

    // getter and setter methods
}
```

```java
@Component
public class Person {
    @Value("#{'Suresh'}")
    private String name;
    @Value("34")
    private int age;
    // SpEL Bean reference
    @Value("#{address}")
    private Address address;
    @Value("#{address.city}")
    private String personCity;
    // SpEL Method invocation
    @Value("#{person.getInfo()}")
    private String personInfo;

    // getter and setter methods
}
```

# SpEL Using Annotation - properties files

- Create a **data.properties** file:

```
technic_name=Java Web,Android,.Net,C/C++,Angular,React
MSG1=Sorry, your username or password is incorrect. Please try again!
MSG2=Username must be not empty!
MSG3=Password must be not empty!
MSG4=You must input all required fields!
MSG5=Wrong format!
```

- We will add some fields to read the configuration from **employee.properties** using @Value annotation.
- **Example**:

```
@Controller
@SessionAttributes("user")
@PropertySource(value = "classpath:data.properties")
public class UserController {

    @Value("#{'${MSG1}'}")
    private String msg1;
    @Value("#{'${MSG2}'}")
    private String msg2;
    @Value("#{'${MSG3}'}")
    private String msg3;
    @Value("#{'${MSG4}'}")
    private String msg4;
    @Value("#{'${MSG5}'}")
    private String msg5;

    @Value("#{'${technic_name}'.split(',')}")
    private List<String> technics;
}
```

Section 4

# REDIRECTVIEW TO ADD/FETCH FLASH ATTRIBUTES USING REDIRECTATTRIBUTES

# RedirectAttributes class

- A specialization of the Model interface that controllers can use to select attributes for a redirect scenario.
- This interface also provides a way to **add flash attributes** and they will be automatically propagated to the "output" FlashMap of the current request.
- A **RedirectAttributes model is empty** when the method is called and is never used unless the method returns a redirect view name or a RedirectView.
- **After the redirect**, flash attributes are automatically added to the model of the controller that **serves the target URL**.

# Methods

- **addFlashAttribute**:("key", "value")
  - ✓ Flash Attributes are attributes which lives in session for short time.
  - ✓ It is used to propagate values from one request to another request and then automatically removed.
  - ✓ Handling flash attributes are achieved using **FlashMap** and **FlashMapManager**.
  - ✓ But in annotated spring MVC controller, it can be achieved with **RedirectAttributes**.
- **addAttribute**(“attributeName”, “attributeValue”)
  - ✓ Add the supplied attribute under the supplied name.

# Add Flash Attributes

```
@RequestMapping(value = "mybook", method = RequestMethod.GET)
  public ModelAndView book() {
    return new ModelAndView("book", "book", new Book());
  }

  @RequestMapping(value = "/save", method = RequestMethod.POST)
  public RedirectView save(@ModelAttribute("book") Book book,
      RedirectAttributes redirectAttrs) {
    redirectAttrs.addAttribute("msg", "Hello World!");
    redirectAttrs.addFlashAttribute("book", book.getBookName());
    redirectAttrs.addFlashAttribute("writer", book.getWriter());

    RedirectView redirectView = new RedirectView();
    redirectView.setContextRelative(true);
    redirectView.setUrl("/hello/{msg}");
    return redirectView;
  }
```

# Fetch Flash Attributes

- To fetch flash attributes we have two approaches.
  - ✓ The first one is by using **Model** as an argument in the @**RequestMapping** method and fetch the flash attribute as below.

model.asMap().get("key");

```
@RequestMapping(value = "/hello/{msg}", method = RequestMethod.GET)
  public String hello(Model model, RedirectAttributes redirectAttrs,
      @PathVariable("msg") String msg, HttpServletRequest request) {
    System.out.println("Message:" + msg);

    System.out.println("Fetch Flash Attributes By using Model");
    System.out.println("Book Name:" + model.asMap().get("book"));
    System.out.println("Writer:" + model.asMap().get("writer"));

    return "redirect:/success.jsp";
  }
```

# SUMMARY

1. Spring @RequestParam Annotation
2. Spring Form tags
3. @SessionAttributes or @SessionAttribute
4. Spring Expression Language (SpEL)
5. RedirectView and RedirectAttributes

# Thank you