

LESSON 4

SPRING MVC TAG LIBRARY

Do Less and Accomplish More

Spring MVC Form Tag Library

Facilitates the development of JSP pages

Integrated with Spring MVC data binding features

Each tag provides support for the set of attributes of its corresponding HTML tag counterpart, making the tags familiar and intuitive to use.

Access Form Tag Library:

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


Spring MVC data binding

Built-in Data Binding handles simple String to data type conversions
HTTP request parameters [String types] are converted to model object properties of varying data types.

As a result:

Data binding makes form beans obsolete

Any object can be a command AKA a form-backing object

Can bind directly to **business domain objects**.

Domain objects exist as POJOs

Makes it EASIER –
As Binding function is ALREADY there.

The Spring MVC Form tag library makes it easy to bind form elements to model data with JSP pages.

Form Tag Library

Tag Name	Description
form	Renders an HTML form tag and exposes the binding path to the other form tags.
input	Renders an input element with the default type of text. The type of the input element can be (optionally) specified (like email, date etc.) . Note that you can't use radiobutton or checkbox for those types.
password	Renders an input element of type password.
hidden	Renders an input element of type hidden.
select	Renders an HTML select element. The option and/or options tag are used to specify the options to render.
option	Renders a single HTML option element inside a select element.
options	Renders a collection of HTML option elements inside a select element.
radiobutton	Renders an HTML input element of the type radio button.
radiobuttons	Renders multiple HTML input elements of the type radio button.
checkbox	Renders an HTML input element of the type checkbox.
checkboxes	Renders multiple HTML input elements of the type checkbox.
textarea	Renders an HTML Textarea element.
errors	Displays binding and/or validation errors to the user. It can be used to either specify the path for field-specific error messages or to specify an * to display all error messages.
label	Renders a HTML Label and associate it with an input element.
button	Renders an HTML Button element.

[Spring Form Tag Library](#)

Form Tag

This tag renders an HTML 'form' tag and exposes a **binding path** to inner tags for binding. *All the other tags in this library are nested tags of the form tag.*

```
@RequestMapping(value = "/addBook", method = RequestMethod.GET)  
public String inputBook(@ModelAttribute("newBook") Book book) {
```

```
<form:form modelAttribute="newBook" action="addBook" method="post">
```

OR

```
<form:form commandName="newBook" action="addBook" method="post">
```

```
@RequestMapping(value = "/addBook", method = RequestMethod.POST)  
public String saveBook(@ModelAttribute("newBook") Book newBook) {
```


Form Tag

A “different” look for the previous Form Tag example.

Which do you prefer?

```
@RequestMapping(value = "/addBook", method = RequestMethod.GET)  
public String inputBook(Model model) {  
    model.addAttribute("book", new Book());  
}
```

```
<form:form modelAttribute="book" action="addBook" method="post">
```

OR

```
<form:form commandName="book" action="addBook" method="post">
```

```
@RequestMapping(value = "/addBook", method = RequestMethod.POST)  
public String saveBook(@ModelAttribute Book newBook) {
```


Form Tag

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

```
public String getAddNewProductForm(@ModelAttribute("newProduct") Product product) {
```

No need for Action as long as GET & POST have same URI [HTML 4]

```
<form:form modelAttribute="newProduct" >
```

No need for Method as Post is "assumed" [Spring]

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
```

```
public String processAddNewProductForm(@ModelAttribute("newProduct") Product productToBeAdded  
) {
```

ALSO, by default, Spring IGNORES Suffixes

/add.do will match to

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
```


Form Tag Binding Algorithm

```
<form:form modelAttribute="newBook" action="addBook" method="post">
```

For “Get” Method, Looks in model map for key: “newBook”

If it doesn’t find the key – error is thrown.

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

```
public String inputBook(@ModelAttribute("newBook")Book book,Model model)
```

For “Post” Method, Looks in model map for key: “newBook”

If it doesn’t find the key, does “best match” on other attributes...

```
@RequestMapping(value = "/addBook", method = RequestMethod.POST)
```

```
public String saveBook(@ModelAttribute("newBook")Book book,Model model)
```

OR

```
public String saveBook(Book book,Model model)
```


@ModelAttribute

Can be placed on a method parameter:

```
@RequestMapping(value = "/addBook", method = RequestMethod.POST)  
public String saveBook(@ModelAttribute("newBook") Book book) {}
```

The Object should be retrieved from the model or instantiated if doesn't exist. The Object fields should be populated from all request parameters that have matching names.

Can be placed on method. Method invoked before methods annotated with @RequestMapping

```
@ModelAttribute("books")  
    List<Book> addBookList(Model model) {  
        return bookService.getAllBooks();  
    }
```

Object is added to Model – in this example the List of books is added

Form examples with HTML output

```
<form:input id="title" path="title"/>
```

Generated HTML:

```
<input id="title" name="title" type="text" value=""/>
```

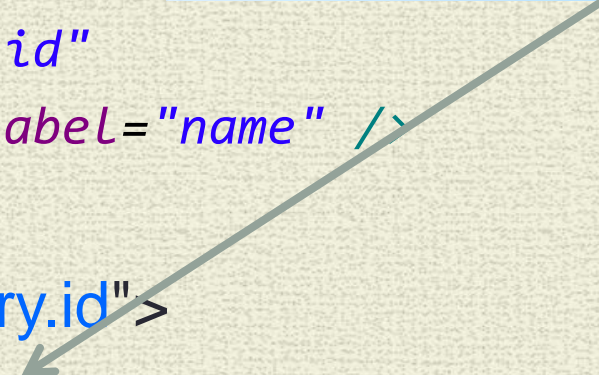
- When **categories** is a **LIST**

```
<form:select id="category" path="category.id"
items="${categories}" itemValue="id" itemLabel="name" />
```

Generated HTML:

```
<select id="category" name="category.id">
  <option value="1" selected="selected">Computing</option>
  <option value="2">Travel</option>
  <option value="3">Health</option>
</select>
```

If category.id already has a value,
Then it will be marked as selected



NOTE: *path* is the “binding Path” defined previously

Form example with HTML output [Cont.]

```
<form:select id="category" path="category.id">
  <form:option value="0" label="--Select Category"/>
  <form:options items="{categories}" itemLabel="name" itemValue="id"/>
</form:select>
```

```
<select id="category" name="category.id">
  <option value="0" selected="selected">--Select Category</option>
  <option value="1">Computing</option>
  <option value="2">Travel</option>
  <option value="3">Health</option>
</select>
```

When **categories** is a MAP

```
<form:select path="category.id"
  items="{categoriesMap}" itemLabel="name"/>
```

Automatically Yields:

```
<option value="1">Computing</option>
```

Explicit itemLabel
Otherwise defaults to Map value
In this case category Object

Where "1" is the Map key

General Purpose Spring Tag Library

Not Spring MVC specific

Available to any Java Server Page used in the Spring Framework

Tags for evaluating errors, setting themes and outputting internationalized messages.

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


Spring Tag Library

Tag Name	Description
<code>htmlEscape</code>	Sets the default HTML escape value for the current page. If set, this tag overrides the value set in the <code>defaultHtmlEscape</code> context-parameter.
<code>escapeBody</code>	Escapes the enclosed body.
<code>message</code>	Displays a message with the given code and for the selected locale. (See the section about internationalization (I18N) later in this chapter for more information.)
<code>theme</code>	Uses the variable from the currently selected theme. (See Chapter 9 for more information.)
<code>hasBindErrors</code>	Shows or hides part of the page (or element) based on whether an model attribute has bind (or validation) errors.
<code>nestedPath</code>	Selects a nested path to be used by the <code>bind</code> tag's path. For example, it can be used to bind <code>customer.address</code> to <code>street</code> instead of to <code>customer.address.street</code> .
<code>bind</code>	Binds an attribute of the model (attribute) to the enclosed input element.
<code>transform</code>	Transforms the bound attribute using Spring's type-conversion system. This tag must be used inside a <code>spring:bind</code> tag.
<code>url</code>	Similar to the <code>jstl core URL</code> tag. It is enhanced so the URL can contain URL template parameters.
<code>param</code>	Specifies a parameter and value to be assigned to the URL. This tag is used inside an <code>url</code> tag.
<code>eval</code>	Evaluates a SpEL expression and prints the result or assigns it to a variable.

Message Tag

Message tag

Internationalization support through externalization of messages

Text from MessageSource configured in DispatcherServlet

```
<spring:message code="greeting" text ="Hi" />
```

“code” isn’t set or cannot be resolved, “text” will be used as default message.

Also, the spring:message- tag, works with the locale support that comes with Spring. [Internationalization]

Spring [externalize] message tag example

Dispatcher Servlet configuration file declares message source bean. “messages” file is messages.properties and must reside in the source class path in order to be discovered.

```
<bean id= "messageSource"  
      class="org.springframework.context.support.ResourceBundleMessageSource">  
    <property name="basenames" value="message2,messages"/>  
</bean>
```

Spring message tag accesses label name from resource file messages.properties

```
<label for="bookList"><spring:message code="book.book" /> </label>
```


Spring URL Tag

Access static resources:

Spring URL tag

Resolves the path from context root - irrespective of current URL.

```
<style type="text/css">@import url("<spring:url  
value="/css/main.css"/>"); </style>
```

Resolves to:

```
<style type="text/css">@import url("/Book05a/css/main.css");</style>
```

Query Parameter passing:

Like JSTL URL

```
<spring:url value="/addBook" var="addBook_url" >  
<spring:param name="ISBN" value="1234"/></spring:url>  
<a href="${addBook_url}">Add Book</a>
```

Resolves to: Add Book

Spring template/@PathVariable:

```
<spring:url value="/book_edit/{id}" var="edit" >  
<spring:param name="id" value="${book.id}" />  
</spring:url>  
<td><a href="${edit}">Edit</a></td>
```

Resolves to: Edit

Main Point

- The Spring MVC tag library facilitates data binding with specialized Form tags.
- *The practice of the TM technique facilitates the strengthening of the bond between the heart and mind*

Excluding Fields from Data Binding

- We don't always want to bind our ENTIRE domain object
- For example, an internal Customer or Product ID
- Need to explicitly restrict binding on specific fields
- WebDataBinder object used by the binding function
- @initBinder annotation identifies Controller method that accesses WebDataBinder
- @InitBinder
- **public void initialiseBinder(WebDataBinder binder) {**
- **binder.setDisallowedFields("id");**
-

Custom Data Binding

- Built-in Data Binding handles simple String to data type conversions
- Custom Binding is needed to handle more complex cases
- Three Options in Spring MVC:
 - **Custom PropertyEditor**
 - Old-style [“retro”] ,heavyweight based on entire java.beans package
 - **Converter**
 - General-purpose type conversion system introduced Spring 3.0. used internally by Spring. One way conversion, locale agnostic
 - **Formatter**
 - Designed for Spring MVC form conversion. 2 way conversion - to & from String. Locale aware. More lightweight than Converter.

Out-of-the-box Spring Formatters

- The Number Package:
 NumberFormatter,
 CurrencyFormatter
 PercentFormatter
- The DateTime Package:
- [DateFormatter API](#)
- Custom Examples [NOT out-of-the-box]:
 ISBN Number
 Multiple Select List – collection of Objects

ISBN Formatter Example

```
• public class ISBNFormatter implements Formatter<ISBNNumber> {  
•     public String print(ISBNNumber isbn, Locale locale) {  
•         return isbn.getStart() + "-" +  
•             isbn.getMiddle() + "-" + isbn.getEnd();  
•     }  
•  
•     public ISBNNumber parse(String source, Locale locale)  
•         throws ParseException {  
•         int start = Integer.parseInt(source.substring(0, 3));  
•         int middle = Integer.parseInt(source.substring(4, 7));  
•         int end = Integer.parseInt(source.substring(8, 11));  
•         return new ISBNNumber(start, middle, end);  
•     }  
• }
```


Formatter Configuration

```
<mvc:annotation-driven conversion-service="conversionService"/>
```

```
<bean id="conversionService" class=
    "org.springframework.format.support.FormattingConversionServiceFactoryBean">
    • <property name="formatters">
        <set>
            <bean class="mum.edu.formatter.ISBNFormatter"> </bean>
        </set>
    </property>
</bean>
```

• JavaConfig Version

```
• public void addFormatters(FormatterRegistry registry) {
•     registry.addFormatter(isbnFormatter);
• }
```


Multiple Object Select Formatter

- `/**`
- `* If a Book referenced in a Form by the String value of its ID`
- `* it is converted into an Object & Vice-versa`
- `*/`
- `@Component`
- `public class BookFormatter implements Formatter<Book> {`
- `@Autowired`
- `private BookService bookService;`
- `public Book parse(String text, Locale locale) throws ParseException {`
- `return bookService.get(Long.valueOf(text));`
- `}`
- `@Override`
- `public String print(Book object, Locale locale) {`
- `return String.valueOf(object.getId());`
- `}`

Main Point

- Spring Data Formatters allows displayed form data to be automatically mapped to the internal domain model
- *The practice of the TM technique, automatically allows for coordination between outer and inner values enlivening all aspects of life.*

