

Working with Forms in Thymeleaf

📅 January 04, 2020 2 Comments 🏷️ Thymeleaf 🏷️ Forms

1. Introduction

In this article, we are going to show how to create HTML **Forms in Thymeleaf** with all necessary implementation on the backend side to handle POST requests. Thymeleaf completely supports HTML5, so there is almost no limitation in building complex forms that can be easily handled by the Spring Boot application. This article will cover all common fields used in forms such as **inputs, checkboxes, radio buttons, and dropdowns**.

2. Creating a Form

Thymeleaf comes with several special attributes used for building and handling forms:

- **th:field** - used for binding inputs with properties on the form-backing bean,
- **th:errors** - attribute that holds form validation errors,
- **th:errorclass** - CSS class that will be added to a form input if a specific field has validation errors,
- **th:object** - an attribute that holds a **command object** (main form bean object) that is a form representation on the backend side.

2.1. Command object

Command object is a base bean object attached to the Form which contains all attributes related to input fields. This is the main POJO class with declared setter and getter methods. **Command objects** shouldn't contain any business logic.

The following example shows how to use `th:object` attribute that holds a reference to the **Command object**:

```
<form th:action="@{/registration}"
      th:object="${registration}" method="post">
    ...
</form>
```

Copy

Note that:

- inside one form element there can be only one `th:object` attribute,
- the expression used to define bean object should point to bean object directly. Property navigation will cause errors.

The following example will work just fine:

```
<form th:object="${customer}"></form>
```

Copy

Here we will get an error (`base.customer` - property navigation are not allowed):

```
<form th:object="${base.customer}"></form>
```

Copy

2.2. Inputs

Thymeleaf provides a special attribute `th:field` responsible for binding input fields with a property in the bean class. This attribute behaves differently depending on whether it is attached to. Thymeleaf supports all-new input types introduced in HTML5 such as `type="color"` or `type="datetime"`.

In the following simple example HTML `text` input element was bind with property `username`:

```
<input type="text" th:field="*{username}" />
```

Copy

Let's build more complex form with new HTML5 input types like **number**, **datetime** and **color**:

Copy

```
<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>Spring Boot Thymeleaf Application - Form
Samples - Inputs</title>
</head>
<body>

    <form th:action="@{/sampleInputs}"
th:object="${sampleInputs}" method="post">

        <div>
            <label>Date:</label>
            <input type="text" th:field="*{dateField}"
placeholder="yyyy-MM-dd" />
        </div>
        <div>
            <label>Integer:</label>
            <input type="number" th:field="*{numberField}"
placeholder="integer" />
        </div>
        <div>
            <label>Double:</label>
            <input type="text" th:field="*{doubleField}"
placeholder="double" />
        </div>
        <div>
            <label>String:</label>
            <input type="text" th:field="*{textField}"
placeholder="string" />
        </div>
        <div>
            <label>Color:</label>
            <input type="color" th:field="*{colorField}"
placeholder="color" />
        </div>
        <div>
            <label>Date time:</label>
            <input type="datetime-local" th:field="*
{dateTimeField}" placeholder="date" />
        </div>

        <input type="submit" value="Submit"/>
    </form>
```

```
</body>
</html>
```

Command object attached to the form will have the following structure:

```
package com.frontbackend.thymeleaf.forms.sample.model;

import lombok.*;
import org.springframework.format.annotation.DateTimeFormat;

import java.util.Date;

@ToString
@NoArgsConstructor
@Setter
@Getter
public class SampleInputs {

    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date dateField;

    private Double doubleField;
    private Integer numberField;
    private String textField;
    private String colorField;

    @DateTimeFormat(pattern = "yyyy-MM-dd'T'HH:mm")
    private Date dateTimeField;

}
```

Copy

After filling out form with sample data, the result **command object** will have the following values:

Date: 2019-10-10

Integer: 21

Double: 123.12

String: This is simple text

Color:

Date time: 10.10.2019, 23:00

Submit

```
SampleInputs(  
    dateField=Thu Oct 10 00:00:00  
    CEST 2019,  
    doubleField=123.12,  
    numberField=21,  
    textField=This is simple text,  
    colorField=#d91111,  
    dateTimeField=Thu Oct 10  
    23:00:00 CEST 2019  
)
```

2.3. Checkboxes

In Thymeleaf templates, we can also define **checkboxes**.

Let's see an example of an HTML form that contains checkbox inputs.

Copy

```

<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>Spring Boot Thymeleaf Application - Form
Samples - Checkboxes</title>
</head>
<body>

<form th:action="@{/sampleCheckboxes}"
th:object="${sampleCheckboxes}" method="post">

    <ul>
        <li th:each="color : ${allSampleColors}">
            <input type="checkbox" th:field="*{colors}"
th:value="${color}"/>
            <label th:for="${#ids.prev('colors')}"
th:text="${color}">RED</label>
        </li>
    </ul>

    <div>
        <label th:for="${#ids.next('first')}">First:
</label>
        <input type="checkbox" th:field="*{first}" />
    </div>
    <div>
        <label th:for="${#ids.next('second')}">Second:
</label>
        <input type="checkbox" th:field="*{second}"/>
    </div>

    <input type="submit" value="Submit"/>
</form>

</body>
</html>

```

This example form starts with a list of checkboxes for multi-selecting colors, and below that we have two checkboxes with **First** and **Second** flag. Note that we have to use Thymeleaf utility class for generating ids: `#ids.prev('colors')` and `#ids.next('second')`, because checkboxes are potentially multi-valued, and their ids must be unique.

To handle that form we created **command object** with the following structure:

```
package com.frontbackend.thymeleaf.forms.sample.model;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

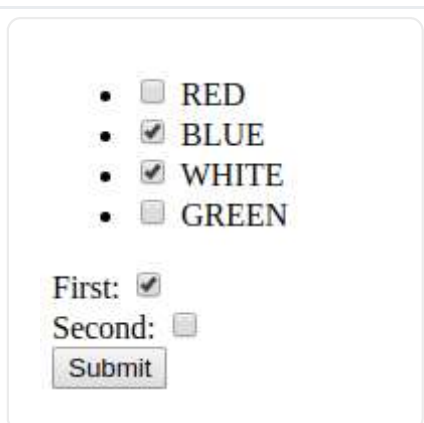
@ToString
@NoArgsConstructor
@Setter
@Getter
public class SampleCheckboxes {

    private SampleColor[] colors;

    private Boolean first;
    private Boolean second;

}
```

Example values in form will give the following result in **command object**:

	<pre>SampleCheckboxes(colors=BLUE, WHITE, first=true, second=false)</pre>
---	---

2.4. Radio Buttons

Radio button fields are similar to checkboxes except that they are not multivalued.

In the following example we give user the opportunity to choose one color:

[Copy](#)

```
<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>Spring Boot Thymeleaf Application - Form
Samples - Checkboxes</title>
</head>
<body>

<form th:action="@{/sampleRadioButtons}"
th:object="${sampleRadioButtons}" method="post">

    <ul>
        <li th:each="color : ${allSampleColors}">
            <input type="radio" th:field="*{color}"
th:value="${color}" />
            <label th:for="${#ids.prev('color')}}"
th:text="${color}">RED</label>
        </li>
    </ul>

    <input type="submit" value="Submit"/>
</form>

</body>
</html>
```

Our command object will look like the following:

[Copy](#)

```

package com.frontbackend.thymeleaf.forms.sample.model;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@ToString
@NoArgsConstructor
@Setter
@Getter
public class SampleRadioButtons {

    private SampleColor color;
}

```

Example values in form will give the following result:

- ☐ RED
- ☐ BLUE
- ☒ WHITE
- ☐ GREEN

SampleRadioButtons(
color=WHITE
)

2.5. Dropdown/List selectors

In order to create HTML dropdown we need to define **select** element and nested **option** tags. The **select** element has to include a **th:field** attribute.

The following example shows how to use the dropdown in single and multi-select mode.

Copy

```

<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>Spring Boot Thymeleaf Application - Form
Samples - Checkboxes</title>
</head>
<body>

    <form th:action="@{/sampleDropdowns}"
th:object="${sampleDropdowns}" method="post">

        <div>
            <label>Select one color:</label>
            <select th:field="*{color}">
                <option th:each="color : ${allSampleColors}"
                    th:value="${color}"
                    th:text="${color}">RED
            </option>
            </select>
        </div>

        <div>
            <label>Select colors:</label>
            <select th:field="*{colors}" multiple>
                <option th:each="color : ${allSampleColors}"
                    th:value="${color}"
                    th:text="${color}">RED
            </option>
            </select>
        </div>

        <input type="submit" value="Submit"/>
    </form>

</body>
</html>

```

command object to store selected values will have the following structure:

Copy

```

package com.frontbackend.thymeleaf.forms.sample.model;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@ToString
@NoArgsConstructor
@Setter
@Getter
public class SampleDropdowns {

    private SampleColor color;
    private SampleColor[] colors;
}

```

Example values in form will give the following result:

Select one color:

GREEN ▼

RED
BLUE
WHITE
GREEN ▼

Select colors:

GREEN ▼

Submit

SampleDropdowns(
color=GREEN,
colors=RED, BLUE, WHITE
)

3. Validation and Error Messages

Thymeleaf allows us to handle errors on the form and field level.

3.1. Field errors

To check if the field contains any error we can use `#fields.hasErrors()` the method that takes the field expression as the parameter.

The following example appends `fieldError` class when error occurs on `dateField` input.

Copy

```
<input type="text" th:field="*{dateField}"
th:class="${#fields.hasErrors('dateField')}? fieldError"
/>
```

We can also list all errors assigned to the specified field:

```
<div>
  <p th:each="error : ${#fields.errors('dateField')}"
th:text="${error}"></p>
</div>
```

Copy

Thymeleaf comes with special attribute `th:errorclass` that appends the specified CSS class to the element if such field has any associated errors.

```
<input type="text" th:field="*{dateField}" class="form-
input" th:errorclass="error" />
```

Copy

In case `dateField` has errors, the input tag will render as:

```
<input type="text" id="dateField" name="dateField"
value="2013-01-01" class="form-input error" />
```

Copy

3.2. All errors

We can show all errors that accrued in the form. We just need to use `#fields.hasErrors()` utility method with `*` or `all` parameter, or `#fields.hasAnyErrors()`.

```
<ul th:if="${#fields.hasErrors('*')}">
  <li th:each="error : ${#fields.errors('*')}"
th:text="${error}">error</li>
</ul>
```

Copy

3.3. Global errors

In Thymeleaf forms, we have also the third type of an error: **global**. These are errors that are not related with any form or field. We can add them programmatically from the backend side of the application.

To access those errors we need to use **global** as a parameter to the **#field.hasErrors()** method.

```
<ul th:if="${#fields.hasErrors('global')}">
  <li th:each="error : ${#fields.errors('global')}"
      th:text="${error}">error</li>
</ul>
```

Copy

4. Conclusion

In this article, we focused on creating forms, binding fields with **command object** attributes and handling validation errors. Thymeleaf supports all new types introduced in HTML5 and comes with several utility methods that could be helpful in creating complex forms with all necessary validations.

As usual, the code used in this tutorial can be found in our [GitHub repository](#).

Comments (2)



Bala

4 years ago | <#>

Team -

Thanks for adding this. Much helpful. Could you also please add selecting checkboxes at different levels and how can we handle it via Springboot + Thymeleaf?

Eg.

1) CheckBox1

a.Checkboxa1
b.Checkboxb1
c.Checkboxc1
2)CheckBox2
a.Checkboxa2
b.Checkboxb2
c.Checkboxc2
3)Checkbox3
a.Checkboxa3
b.Checkboxb3
c.Checkboxc3

Also, if possible keep it under a fieldset enabling Checking and Unchecking.

[Reply](#)



franek

4 years ago | <#>

it wasn't easy, because th:field doesn't work in such complex structures as trees, so I had to try a different approach, and everything seems to be working properly, pls go and check that article: <https://frontbackend.com/thymeleaf/spring-bootstrap-thymeleaf-checkbox-tree>

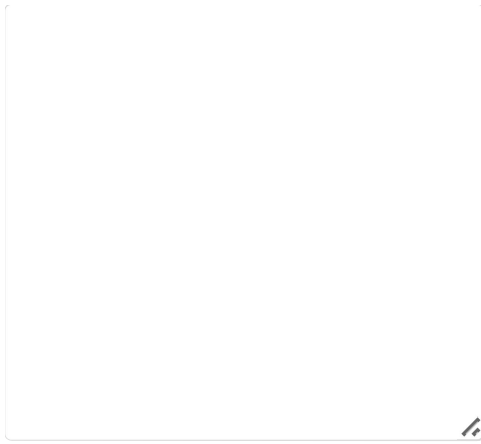
[Reply](#)

Leave a comment

Name

Email

Comment



Submit

2024 FrontBackend