[Bài đọc] Tạo Form

1. Xử lý command object

Command Object là tên gọi cho đối tượng đứng sau mỗi form trong Spring MVC, các đối tượng này lưu giữ giá trị của các trường của một form, các thuộc tính của các đối tượng này được get và set bởi framework dựa theo các input của form.

 Thymeleaf yêu cầu bạn xác định command object bằng cách sử dụng thuộc tính th:object trong thẻ <form>:

<form action="#" th:action="@{/seedstartermng}" th:object="${seedStarter}" method="post">  
 ...  
</form>

 Có một số ràng buộc sau đây:

* Các giá trị cho thuộc tính th:object trong các thẻ form phải là biểu thức ($ {...}) chỉ định theo tên của model attribute, ${seedStarter} là hợp lệ, nhưng $ {seedStarter.data} sẽ không hợp lệ.
* Thực tế là các form HTML không thể lồng nhau, do đó trong thẻ <form>, cũng không thể được đặt thêm th:object khác.

2. Inputs

Để bind giá trị của một input với một trường dữ liệu trong command object, dùng thuộc tính th:field theo cú pháp như sau:

<input type="text" th:field="\*{datePlanted}" />

Bạn có thể xem th:field tương đương với thuộc tính "path" trong thư viện thẻ JSP.

Thuộc tính th:field hoạt động khác nhau tùy thuộc vào việc nó được gắn vào một thẻ <input>, <select> hay <textarea>> (và cũng phụ thuộc vào loại thẻ <input> cụ thể). Trong trường hợp này (input [type = text]), dòng mã trên tương đương với:

<input type="text" id="datePlanted" name="datePlanted" th:value="\*{datePlanted}" />

và hơn thế nữa, trong trường hợp trên, một trường text đang được bind với một trường dữ liệu Date, nên th:field cũng sẽ đăng ký với bộ chuyển đổi kiểu tự động của Spring MVC, nhờ đó, ngày sẽ được hiển thị đúng định dạng.

Giá trị cho thuộc tính th:field phải là biểu thức chọn (dấu \* trong \* {...}), có nghĩa là chúng sẽ được bind với thuộc tính của command object, chứ không phải là bất kỳ biến hay model attribute nào khác.

Trái với các đối tượng trong th:object, các biểu thức này có thể là dẫn xuất từ thuộc tính, ví dụ \*{datePlanted.day}.

Lưu ý rằng th:field cũng hiểu các kiểu phần tử <input> mới được HTML5 giới thiệu như <input type = "datetime" ... />, <input type = "color" ... />, v.v. một cách hiệu quả.

3. Checkbox fields

th:field cũng cho phép chúng ta xác định đầu vào checkbox. Hãy xem ví dụ từ trang HTML sau:

<div>  
 <label th:for="${#ids.next('covered')}" th:text="#{seedstarter.covered}">Covered</label>  
 <input type="checkbox" th:field="\*{covered}"/>  
</div>

Lưu ý, có một số nội dung cần quan tâm ở đây bên cạnh checkbox, giống như label được gắn ngoài và cũng sử dụng hàm #ids.next('covered') để lấy giá trị, sẽ được áp dụng cho thuộc tính id của checkbox input.

Tại sao chúng ta cần sinh tự động id cho trường này? Bởi vì các checkbox có nhiều giá trị và do đó các id của chúng sẽ luôn được thêm một số thứ tự (sequence), bằng cách sử dụng hàm # ids.seq (...) để đảm bảo rằng mỗi checkbox input cho cùng một thuộc tính có một giá trị id khác nhau.

Chúng ta có thể thấy điều này dễ dàng hơn nếu chúng ta nhìn vào trường checkbox có nhiều giá trị:

<ul>  
 <li th:each="feat : ${allFeatures}">  
 <input type="checkbox" th:field="\*{features}" th:value="${feat}"/>  
 <label th:for="${#ids.prev('features')}"  
 th:text="#{${'seedstarter.feature.' + feat}}">Heating</label>  
 </li>  
</ul>

Lưu ý rằng, chúng ta đã thêm thuộc tính th:value tại thời điểm này, vì trường features không phải là một kiểu boolean (true/false), mà thay vào đó là một mảng các giá trị.

Hãy xem kết quả đầu ra HTML được tạo bởi mã như sau:

<ul>  
 <li>  
 <input id="features1" name="features" type="checkbox" value="SEEDSTARTER\_SPECIFIC\_SUBSTRATE"/>  
 <input name="\_features" type="hidden" value="on"/>  
 <label for="features1">Seed starter-specific substrate</label>  
 </li>  
 <li>  
 <input id="features2" name="features" type="checkbox" value="FERTILIZER"/>  
 <input name="\_features" type="hidden" value="on"/>  
 <label for="features2">Fertilizer used</label>  
 </li>  
 <li>  
 <input id="features3" name="features" type="checkbox" value="PH\_CORRECTOR"/>  
 <input name="\_features" type="hidden" value="on"/>  
 <label for="features3">PH Corrector used</label>  
 </li>  
</ul>

Chúng ta có thể thấy ở đây cách một hậu tố (thứ tự) được thêm vào thuộc tính id của mỗi input và cách hàm # ids.prev (...) cho phép chúng ta truy tìm giá trị cuối cùng được tạo cho một input id cụ thể.

Đừng lo lắng về những thông tin đầu vào bị ẩn có name="\_features": chúng được tự động thêm vào để tránh các vấn đề với trình duyệt không gửi giá trị checkbox được bỏ chọn tới máy chủ khi gửi form.

Lưu ý rằng, nếu thuộc tính của chúng ta có chứa một số giá trị được chọn trong bean form-backing, th:field sẽ thêm thuộc tính checked = "checked" vào các thẻ input tương ứng.

4. Radio Button fields

Các trường radio được xác định theo cách tương tự với các checkbox không phải là giá trị boolean, xác nhận rằng chúng không được đa giá trị, tất nhiên:

<ul>  
 <li th:each="ty : ${allTypes}">  
 <input type="radio" th:field="\*{type}" th:value="${ty}"/>  
 <label th:for="${#ids.prev('type')}" th:text="#{${'seedstarter.type.' + ty}}">Wireframe</label>  
 </li>  
</ul>

5. Dropdown/List

Các trường Select có hai phần: thẻ <select> và các thẻ <option> lồng nhau của nó. . Khi tạo loại trường này, chỉ thẻ <select> phải include thuộc tính th:field,nhưng thuộc tính th:value trong các thẻ <option> lồng nhau sẽ rất quan trọng vì chúng sẽ cung cấp phương tiện để biết đó là tùy chọn được chọn hiện tại (tương tự như các checkbox không phải là boolean và nút radio).

Hãy xây dựng lại trường type dưới dạng dropdown select:

<select th:field="\*{type}">  
 <option th:each="type : ${allTypes}"  
 th:value="${type}"  
 th:text="#{${'seedstarter.type.' + type}}">Wireframe  
 </option>  
</select>

Tại thời điểm này, hiểu được đoạn mã này khá dễ dàng. Chỉ cần chú ý cách thiết lập thuộc tính th:each trong thẻ <option>.

Last modified: Wednesday, 27 November 2019, 10:22 AM