[Bài đọc] Vòng lặp trong Thymeleaf

1. Cơ bản về vòng lặp

Để hiển thị các sản phẩm trong trang /WEB-INF/templates/product/list.html, chúng ta sẽ sử dụng các phần tử <div>. Mỗi sản phẩm của chúng ta sẽ được hiển thị trong một phần tử <div>, và như vậy, chúng ta sẽ cần tạo một template <div> (một mẫu sẽ minh họa cách chúng ta muốn mỗi sản phẩm được hiển thị), và sau đó hướng dẫn Thymeleaf lặp lại nó, một lần cho mỗi sản phẩm.

Standard Dialect cung cấp cho chúng ta một thuộc tính để làm điều này: th: each.

**Sử dụng th:each**

Đối với trang danh sách sản phẩm, chúng ta sẽ cần một phương thức controller lấy danh sách các sản phẩm từ tầng service và thêm nó vào template:

@RequestMapping("/products")  
public ModelAndView listProducts() {  
 List<Product> products = productService.findAll();  
 ModelAndView modelAndView = new ModelAndView("/product/list", "products", products);  
 return modelAndView;  
}

Và sau đó, chúng ta sẽ sử dụng th:each trong template để lặp qua danh sách sản phẩm:

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>Product List</title>  
 <link href="/styles/style.css" rel="stylesheet">  
   
</head>  
<body>  
<a href="/create-product">Create new product</a>  
<h1 class="title">Products List</h1>  
<div>  
 <div th:each="product,iterStat : ${products}" th:class="${iterStat.odd}? 'odd'">  
 <img width="150px" height="150px" th:src="@{'/images/' + ${product.image}}">  
 <p th:text="${product.name}"></p>  
 <p th:text="${product.price}"></p>  
 <p><a th:href="@{/edit-product/\_\_${product.id}\_\_ }">Edit</a></p>  
 <div class="clear"></div>  
 </div>  
</div>  
  
</body>  
</html>

Giá trị thuộc tính product : ${products} bạn thấy ở trên có nghĩa là: “cho mỗi phần tử trong kết quả của đánh giá ${products}, lặp lại đoạn template này, sử dụng phần tử hiện tại trong một biến được gọi là product”. Hãy đặt tên cho mỗi thứ chúng ta nhìn thấy:

* Chúng ta sẽ gọi ${products} là *iterated expression* (biểu thức được lặp lại) hoặc *iterated variable* (biến lặp).
* Chúng ta sẽ gọi product là *iteration variable* (biến lặp) hoặc đơn giản là *iter variable*.

Lưu ý rằng, biến lặp product được bao phủ cả phần tử <div>

**Iterable values**

Lớp java.util.List không phải là giá trị duy nhất có thể được sử dụng để lặp trong Thymeleaf. Có một tập hợp khá hoàn chỉnh của các đối tượng được coi là có thể lặp lại theo thuộc tính th:each:

* Bất kỳ đối tượng nào đang triển khai java.util.Iterable
* Bất kỳ đối tượng nào đang triển khai java.util.Enumeration.
* Bất kỳ đối tượng nào đang triển khai java.util.Iterator, các giá trị của nó sẽ được sử dụng khi chúng được trả về bởi trình lặp, mà không cần phải cache tất cả các giá trị trong bộ nhớ.
* Bất kỳ đối tượng nào đang triển khai java.util.Map. Khi lặp lại các map, các biến lặp sẽ là của lớp java.util.Map.Entry.
* Bất kỳ mảng nào.
* Bất kỳ đối tượng nào khác sẽ được coi như là một danh sách có giá trị duy nhất chứa chính đối tượng đó.

2. Giữ trạng thái lặp

Khi sử dụng th:each, Thymeleaf cung cấp một cơ chế hữu ích cho việc theo dõi trạng thái của sự lặp lại: biến trạng thái (status variable).

Các biến trạng thái được định nghĩa trong thuộc tính th:each và chứa các dữ liệu sau:

* Chỉ số lặp hiện tại (*iteration index*), bắt đầu bằng 0. Đây là thuộc tính chỉ mục (index).
* Chỉ số lặp hiện tại(*iteration index*),  bắt đầu bằng 1. Đây là thuộc tính đếm (count).
* Tổng số phần tử trong biến lặp. Đây là thuộc tính kích thước (size).
* Biến lặp cho mỗi lần lặp. Đây là thuộc tính hiện tại (current).
* Cho dù lặp hiện tại (current iteration) là chẵn hay lẻ. Đây là những thuộc tính boolean even / odd.
* Cho dù lặp hiện tại (current iteration) là lần đầu tiên. Đây là thuộc tính boolean first.
* Cho dù lặp hiện tại (current iteration) là lần cuối cùng. Đây là thuộc tính boolean last.

Hãy xem cách chúng ta có thể sử dụng nó với ví dụ trước:

<div th:each="product,iterStat : ${products}" th:class="${iterStat.odd}? 'odd'">  
 <img width="150px" height="150px" th:src="@{'/images/' + ${product.image}}">  
 <p th:text="${product.name}"></p>  
 <p th:text="${product.price}"></p>  
 <p><a th:href="@{/edit-product/\_\_${product.id}\_\_ }">Edit</a></p>  
 <div class="clear"></div>  
</div>

Biến trạng thái (iterStat trong ví dụ này) được định nghĩa trong thuộc tính th:each bằng cách viết tên của nó sau biến lặp, được phân tách bằng dấu phẩy. Cũng giống như biến lặp, biến trạng thái cũng được bao phủ theo đoạn mã được xác định bởi thẻ đang giữ thuộc tính th:each.

Chúng ta hãy xem kết quả của việc xử lý template:

<div>  
 <div class="odd">  
 <img width="150px" height="150px" src="/images/samsung.jpeg">  
 <p>Samsung</p>  
 <p>900.0</p>  
 </div>  
 <div>  
 <img width="150px" height="150px" src="/images/Iphone.png">  
 <p>Iphone</p>  
 <p>900.0</p>  
 </div>  
 <div class="odd">  
 <img width="150px" height="150px" src="/images/Bphone.png">  
 <p>Bphone</p>  
 <p>200.0</p>  
 </div>  
 <div>  
 <img width="150px" height="150px" src="/images/Macbook.png">  
 <p>Macbook</p>  
 <p>100.0</p>  
 </div>  
</div>

Lưu ý rằng, biến trạng thái lặp của chúng ta đã hoạt động hoàn hảo, chỉ thiết lập CSS class là odd tới các hàng lẻ.

Nếu bạn không đặt rõ ràng biến trạng thái, Thymeleaf sẽ luôn tạo một biến cho bạn bằng cách thêm Stat vào tên của biến lặp.

3. Tối ưu hóa thông qua lazy retrieval của dữ liệu

Đôi khi, chúng ta có thể muốn tối ưu hóa việc truy xuất tập dữ liệu (ví dụ: từ cơ sở dữ liệu) để các tập dữ liệu này chỉ được truy xuất nếu chúng thực sự được sử dụng.  
Trên thực tế, đây là thứ có thể được áp dụng cho bất kỳ phần dữ liệu nào, nhưng với kích thước mà tập dữ liệu trong bộ nhớ có thể có, việc lấy các tập dữ liệu có nghĩa là được lặp lại là trường hợp phổ biến nhất.

Để hỗ trợ điều này, Thymeleaf cung cấp một cơ chế để tải các biến ngữ cảnh một cách lười biếng (lazily load context variables). Các biến ngữ cảnh thi hành giao diện ILazyContextVariable - bằng cách mở rộng triển khai mặc định LazyContextVariable của nó - sẽ được giải quyết trong thời điểm thực thi. Ví dụ:

context.setVariable(  
 "users",  
 new LazyContextVariable<List<User>>() {  
 @Override  
 protected List<User> loadValue() {  
 return databaseRepository.findAllUsers();  
 }  
});

Biến này có thể được sử dụng mà không cần có kiến ​​thức về *lazyness*, như sau:

<ul>  
 <li th:each="u : ${users}" th:text="${u.name}">user name</li>  
</ul>

Nhưng tại cùng thời điểm, sẽ không bao giờ được khởi tạo (phương thức loadValue () của nó sẽ không bao giờ được gọi) nếu condition  được đánh giá là false trong đoạn mã như sau:

<ul th:if="${condition}">  
 <li th:each="u : ${users}" th:text="${u.name}">user name</li>  
</ul>