TechMaster
Học là có việc

SPRING MVC VÀ THYMELEAF

Trần Văn Thịnh

Spring MVC

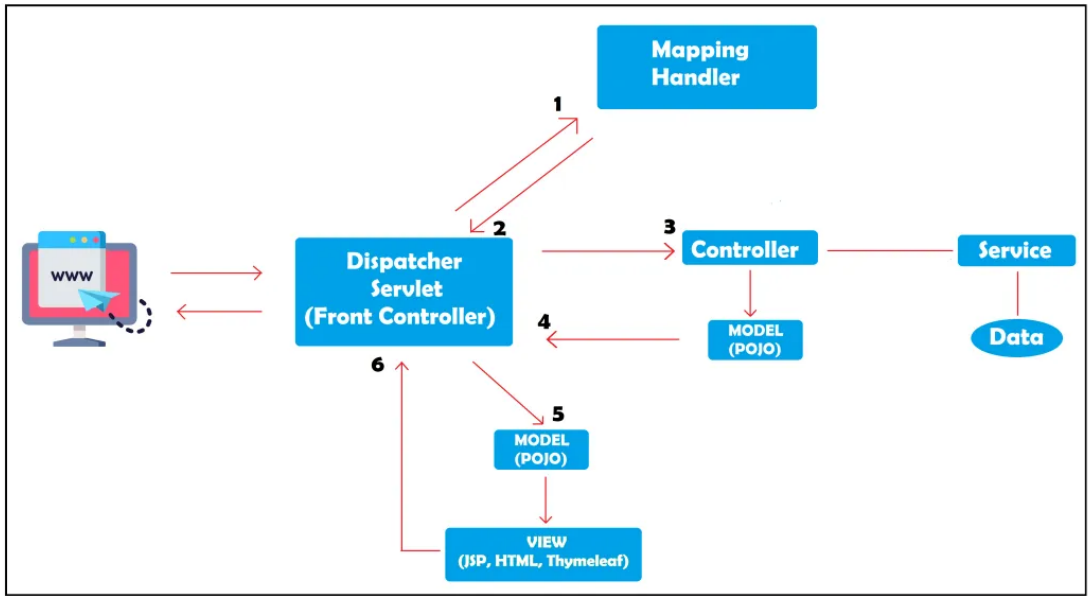
ThymeLeaf

- Giới thiệu ThymeLeaf
- Message(thông báo) và Variable(Biến)
- Vòng lặp
- Điều kiện
- Các biểu thức

Spring MVC
Và
ThymeLeaf

SPRING MVC

Cơ chế xử lý Request và Response trong Spring MVC



Cơ chế xử lý Request và Response trong Spring MVC

- Spring Font Controller (DispatcherServlet) sau khi nhận yêu cầu sẽ tìm đến Mapping Handler thích hợp(có thể hiểu một địa chỉ(url) sẽ maping với 1 controller tương ứng). Mapping Handler sẽ ánh xạ yêu cầu của client đến controller thích hợp.
- DispatcherServlet gửi yêu cầu đến Controller thích hợp
- Sau khi thực hiện tiến trình từ yêu cầu của client, nó thực thi các logic được xác định trong controller và cuối cùng trả về đối tượng Model và View.
- Dựa trên Model và View, Spring MVC sẽ sử dụng View Resolver được cấu hình để tìm View tương ứng (**JSP, FreeMaker, Thymeleaf…)**
- View sẽ được generate và trả về kết quả trong response cho client

Spring Boot hỗ trợ cấu hình ViewResolver

- Khi phát triển một dự án Spring Boot, thường chỉ sử dụng duy nhất một công nghệ cho tầng View (JSP, Thymeleaf, ..), Spring Boot sẽ tự động cấu hình một ViewResolver để làm việc với công nghệ đó. Tuy nhiên trong trường hợp sử dụng nhiều công nghệ cho tầng View, phải tự cấu hình tất cả các ViewResolver cần thiết.
- Trong trường hợp sử dụng nhiều công nghệ cho tầng View, sẽ có nhiều ViewResolver tham gia vào luồng đi (flow) của ứng dụng. Các ViewResolver được sắp xếp theo thứ tự ưu tiên (0, 1, 2, ..). Nếu ViewResolver (0) không tìm thấy "View Name" cần thiết, ViewResolver (1) sẽ được sử dụng, ...

+

- Giới thiệu ThymeLeaf
- Message(thông báo) và Variable(Biến)
- Vòng lặp
- Điều kiện
- Các biểu thức

Giới thiệu ThymeLeaf

- Thymeleaf là một Java Template Engine phía server (Tầng View trong Spring MVC)
- Có nhiệm vụ xử lý và generate ra các file HTML, XML, v.v..
- Các file HMTL do Thymeleaf tạo ra là nhờ kết hợp dữ liệu và template + quy tắc để sinh ra một file HTML chứa đầy đủ thông tin.
- Sử dụng trong Spring Boot
  - Dependency

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

  - Các file html đặt trong thư mục **src/main/resources/templates**
- Tham khảo : https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html

GIỚI THIỆU THYMELEAF

Model

```
public class Person {

    private String firstName;
    private String lastName;

}
```

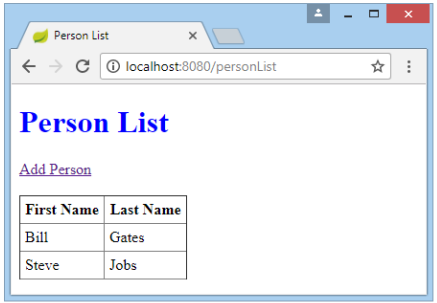
List<Person> persons

+

View (Thymeleaf Template)

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8" />
    <title>Person List</title>
    <link rel="stylesheet" type="text/css"
          th:href="@{/css/style.css}"/>
  </head>
  <body>
    <h1>Person List</h1>
    <a href="addPerson">Add Person</a>
    <br/><br/>
    <div>
      <table border="1">
        <tr>
          <th>First Name</th>
          <th>Last Name</th>
        </tr>
        <tr th:each ="person : ${persons}">
          <td th:utext="${person.firstName}">...</td>
          <td th:utext="${person.lastName}">...</td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

Thymeleaf Engine



- Nếu ứng dụng chỉ sử dụng duy nhất 1 công nghệ cho tầng View, Sping Boot sẽ tự cấu hình ViewResolver
- Trường hợp sử dụng nhiều công nghệ View, thì phải tự cấu hình ViewResolver cho thymeleaf

```
@Configuration
public class ThymeleafViewResolverConfig {
    @Bean
    public ViewResolver thymeleafViewResolver() {
        ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
        viewResolver.setTemplateEngine(thymeleafTemplateEngine());
        viewResolver.setCharacterEncoding("UTF-8");
        viewResolver.setOrder(0);
        // Important!!
        // th_page1.html, th_page2.html, ...
        viewResolver.setViewNames(new String[] { "th_*" });
        return viewResolver;
    }
    // Thymeleaf template engine with Spring integration
    @Bean
    public SpringTemplateEngine thymeleafTemplateEngine() {
        SpringTemplateEngine templateEngine = new SpringTemplateEngine();
        templateEngine.setTemplateResolver(thymeleafTemplateResolver());
        templateEngine.setEnableSpringELCompiler(true);
        return templateEngine;
    }

    @Bean
    public SpringResourceTemplateResolver springResourceTemplateResolver() {
        return new SpringResourceTemplateResolver();
    }

    // Thymeleaf template resolver serving HTML 5
    @Bean
    public ITemplateResolver thymeleafTemplateResolver() {
        ClassLoaderTemplateResolver templateResolver = new ClassLoaderTemplateResolver();
        templateResolver.setPrefix("templates/");
        templateResolver.setCacheable(false);
        templateResolver.setSuffix(".html");
        templateResolver.setTemplateMode(TemplateMode.HTML);
        templateResolver.setCharacterEncoding("UTF-8");
        return templateResolver;
    }
}
```

Giới thiệu ThymeLeaf

- HTML Template Mode
  - Chế độ HTML Template cho phép bất kỳ một loại đầu vào HTML nào, bao gồm HTML5, HTML4, XHTML. Khi xử lý các tài liệu này Thymeleaf Engine sẽ không kiểm tra well-formed của tài liệu, đồng thời nó cũng không xác thực (validate) tài liệu này.
  - Trong chế độ HTML Template, Thymeleaf không tạo ra các thẻ (tag) mới (Ngoại trừ thẻ <th:block>), nó thêm các đánh dấu (markup) của nó vào các thẻ có sẵn của HTML. Bạn có thể hình dung được giao diện của trang khi bạn mở trực tiếp HTML Thymeleaf Template trên trình duyệt. Vì vậy Thymeleaf giúp thu hẹp đáng kể khoảng cách giữa các nhóm thiết kế và nhóm phát triển ứng dụng
  - Ví dụ: thì content trong th:src sẽ thay thế content trong src

Thông báo(Message)

- Message có thể được khai báo trong các file properties và được load lên view sử dụng #{...}

```
<p th:utext="#{home.welcome}">Welcome to our grocery store!</p>
```

```
home.welcome=¡Bienvenido a nuestra tienda de comestibles!
```

- Sử dụng cùng variable:

```
<p th:utext="#{home.welcome(${session.user.name})}">
  Welcome to our grocery store, Sebastian Pepper!
</p>
```

```
home.welcome=¡Bienvenido a nuestra tienda de comestibles!
```

Biến(Variable)

- Biến trong ThymeLeaf có thể là:
  - Một thuộc tính (attribute) org.springframework.ui.Model
  - Một thuộc tính (attribute) HttpServletRequest
  - Một thuộc tính (attribute) HttpSession
  - …
- Biến có thể sử dụng ở mọi nơi trong Template, sử dụng ${…}
- Ví dụ: Today is: 13 february 2011. thì thực tế nó được thực hiện như sau: ctx.getVariables().get("today");

Biến(Variable)

```
@RequestMapping("/variable-example1")
public String variableExample1(Model model, HttpServletRequest request) {
    // variable1
    model.addAttribute("variable1", "Value of variable1!");
    // variable2
    request.setAttribute("variable2", "Value of variable2!");
    return "variable-example1";
}
```


```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8" />
    <title>Variable</title>
  </head>
  <body>
     <h1>Variables</h1>


     <h4>${variable1}</h4>
     <span th:utext="${variable1}"></span>

     <h4>${variable2}</h4>
     <span th:utext="${variable2}"></span>


  </body>

</html>
```

Biến(Variable)

- Các đối tượng cơ bản của biểu thức (Expression Basic Objects) trong ThymeLeaf
  - #ctx: the context object.
  - #vars: the context variables.
  - #locale: the context locale.
  - #httpServletRequest: (only in Web Contexts) the HttpServletRequest object.
  - #httpSession: (only in Web Contexts) the HttpSession object.
- Ví dụ: Established locale country: US.

- Expression Utility Objects trong ThymeLeaf
  - #dates: utility methods for java.util.Date objects: formatting, component extraction, etc.
  - #calendars: analogous to #dates, but for java.util.Calendar objects.
  - #numbers: utility methods for formatting numeric objects.
  - #strings: utility methods for String objects: contains, startsWith, prepending/appending, etc.
  - #objects: utility methods for objects in general.
  - #bools: utility methods for boolean evaluation.
  - #arrays: utility methods for arrays.
  - #lists: utility methods for lists.
  - #sets: utility methods for sets.
  - #maps: utility methods for maps.
  - #aggregates: utility methods for creating aggregates on arrays or collections.
  - #messages: utility methods for obtaining externalized messages inside variables expressions, in the same way as they would be obtained using #{…} syntax.
  - #ids: utility methods for dealing with id attributes that might be repeated (for example, as a result of an iteration).
- 13 May 2011

Biến(Variable)

- Local Variables: là các biến chỉ định nghĩa trong template (file html), chỉ được sử dụng trong một section của template

```
@RequestMapping("/variable-in-loop")
public String objectServletContext(Model model,
HttpServletRequest request) {
    String[] flowers = new String[] {"Rose","Lily", "Tulip",
"Carnation", "Hyacinth" };


    model.addAttribute("flowers", flowers);
    return "variable-in-loop";
}
```


```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8" />
  <title>Variable</title>
</head>
<body>
    <h1>Variable: flower, iter</h1>
    <table border="1">
        <tr>
            <th>No</th>
            <th>Flower Name</th>
        </tr>

        <!--
             Local Variable: flower
             Local Variable: iter (Iterator).
          -->
        <tr th:each="flower, iter : ${flowers}">
            <td th:utext="${iter.count}">No</td>
            <td th:utext="${flower}">Flower Name</td>
        </tr>
    </table>
</body>
</html>
```

Biến(Variable)

- th:width tạo một hay nhiều local variable

```
@RequestMapping("/variable-example3")
public String variableExample3(Model model) {
    String[] flowers = new String[] {"Rose","Lily", "Tulip",
"Carnation", "Hyacinth" };


    model.addAttribute("flowers", flowers);

    return "variable-example3";
}
```


```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8" />
  <title>Variable</title>
</head>
<body>
    <h1>th:with</h1>
    <!-- Local variable: flower0 -->
    <div th:with="flower0 = ${flowers[0]}">

      <h4>${flower0}</h4>
      <span th:utext="${flower0}"></span>

    </div>

    <!-- Local variable: flower1, flower2 -->
    <div th:with="flower1 = ${flowers[1]}, flower2 = ${flowers[2]}">

      <h4>${flower1}, ${flower2}</h4>
      <span th:utext="${flower1}"></span>
      <br/>
      <span th:utext="${flower2}"></span>

    </div>

    <hr>

    <!-- Local variable: firstName, lastName, fullName -->
    <div th:with="firstName = 'James', lastName = 'Smith', fullName = ${firstName} +' ' +
${lastName}">

        First Name: <span th:utext="${firstName}"></span>
        <br>
        Last Name: <span th:utext="${lastName}"></span>
        <br>
        Full Name: <span th:utext="${fullName}"></span>


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

Biến(Variable)

- th:width với Array

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8" />
  <title>Variable</title>
</head>
<body>
    <h1>th:with (Array)</h1>
     <!-- Create an Array: -->
    <th:block th:with="flowers = ${ {'Rose', 'Lily', 'Tulip'} }">
        <table border="1">
            <tr>
                <th>No</th>
                <th>Flower</th>
            </tr>
            <tr th:each="flower, state : ${flowers}">
                <td th:utext="${state.count}">No</td>
                <td th:utext="${flower}">Flower</td>
            </tr>
        </table>
    </th:block>
</body>
</html>
```

Biến(Variable)

- Biểu thức trên các lựa chọn với cú pháp dấu hoa thị * (Expressions on selections (asterisk syntax))
  - ${…} và *{…} đều có thể dùng với các biến, chỉ khác nhau khi sử dụng cùng th:object. Nếu dùng *{…} ở phía bên trong của th:object, thì có nghĩa là viết các attribute của object đó. ${…} lấy giá trị trong Context hay Model

Ví dụ này:

```
<div th:object="${session.user}">
  <p>Name: <span th:text="*{firstName}">Sebastian</span>.</p>
  <p>Surname: <span th:text="*{lastName}">Pepper</span>.</p>
  <p>Nationality: <span th:text="*{nationality}">Saturn</span>.</p>
</div>
```

Sẽ tương đương với

```
<div>
  <p>Name: <span th:text="${session.user.firstName}">Sebastian</span>.</p>
  <p>Surname: <span th:text="${session.user.lastName}">Pepper</span>.</p>
  <p>Nationality: <span th:text="${session.user.nationality}">Saturn</span>.</p>
</div>
```

Vòng lặp

- Sử dụng th:each

```
<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
  </tr>
  <tr th:each="prod,iterStat : ${prods}" th:class="${iterStat.odd}?
'odd'">
    <td th:text="${prod.name}">Onions</td>
    <td th:text="${prod.price}">2.41</td>
    <td th:text="${prod.inStock}? #{true} : #{false}">yes</td>
  </tr>
</table>
```

Điều kiện

- Sử dụng th:if và th:unless

```
<a href="comments.html"
   th:href="@{/product/comments(prodId=${prod.id})}"
   th:if="${not #lists.isEmpty(prod.comments)}">view</a>
```

```
<a href="comments.html"
   th:href="@{/product/comments(prodId=${prod.id})}"
   th:unless="${#lists.isEmpty(prod.comments)}">view</a>
```

- Sử dụng th:switch and th:case

```
 <div th:switch="${user.role}">
    <p th:case="'admin'">User is an administrator</p>
    <p th:case="#{roles.manager}">User is a manager</p>
    <p th:case="*">User is some other thing</p>
 </div>
```

Links URL

- Cú pháp @{…}, sử dụng cùng th:href thay cho href của thẻ <a> của html
- Các loại URL
  - Absolute URLs (đường dẫn tuyệt đối): http://www.thymeleaf.org
  - Relative URLs (đường dẫn gián tiếp)
    - Page-relative. Ví dụ: user/login.html
    - Context-relative. Ví dụ: /itemdetails?id=3 (Context name trong server sẽ được tự động thêm vào)
    - Server-relative. Ví dụ ~/billing/processInvoice (Cho phép gọi URL của các context (application) khác trong cùng server. Ví dụ 2 application được deploy lên cùng 1 tomcat server
    - Protocol-relative URLs, như: //code.jquery.com/jquery-2.0.3.min.js

Links URL

```
<!– Biểu diễn 'http://localhost:8080/gtvg/order/details?orderId=3' -->
<a href="details.html"
  th:href="@{http://localhost:8080/gtvg/order/details(orderId=${o.id})}">view</a>

<!– Biểu diễn đường dẫn gián tiếp Context-relative với application context name là ‘example’ và request param oderId :  ‘/example/order/details?orderId=3' -->
<!– Nếu có nhiều request param hoặc path variable thì ngăn cách nhau bởi dấu phẩy . Ví dụ (orderId=${o.id}, orderType=‘car’)
<a href="details.html" th:href="@{/order/details(orderId=${o.id})}">view</a>

<!– Biểu diễn đường dẫn gián tiếp với path variable orderId:  ‘/example/order/3/details' -->
<a href="details.html" th:href="@{/order/{orderId}/details(orderId=${o.id})}">view</a>

<!–Có thể sử dụng các biểu thức để tính toán ra url -->
<a th:href="@{${url}(orderId=${o.id})}">view</a>
<a th:href="@{'/details/'+${user.login}(orderId=${o.id})}">view</a>
```

Links URL

- Menu cho home page

```
<p>Please select an option</p>
<ol>
  <li><a href="product/list.html" th:href="@{/product/list}">Product List</a></li>
  <li><a href="order/list.html" th:href="@{/order/list}">Order List</a></li>
  <li><a href="subscribe.html" th:href="@{/subscribe}">Subscribe to our Newsletter</a></li>
  <li><a href="userprofile.html" th:href="@{/userprofile}">See User Profile</a></li>
</ol>
```

Literals

- Text literals: Đặt trong dấu ‘’ nháy đơn (single quote)

```
<p> Now you are looking at a <span th:text="'working webapplication'">template file</span>. </p>
```

- Number literals

```
<p>The year is <span th:text="2013">1492</span>.</p> <p>In two years, it will be <span th:text="2013 + 2">1494</span>.</p>
```

- Boolean literals

```
<div th:if="${user.isAdmin()} == false"> ...
```

```
<div th:if="${variable.something} == null"> ...
```

- Literal Substitution (thay thế)

  Dùng để thay thế cho việc cộng String với biến.

  Ví dụ `<span th:text="|Welcome to our application, ${user.name}!|">`

  tương đương với `<span th:text="'Welcome to our application, ' + ${user.name} + '!'">`

Các biểu thức

- Các phép tính toán học trong thymeleaf: + - * / và % (chia lấy dư)

  Ví dụ
  ```
  th:with="isEven=(${prodStat.count} % 2 == 0)"
  ```

  hoặc (tính toán với OGNL, Spring EL)
  ```
  th:with="isEven=${prodStat.count % 2 == 0}"
  ```

- Biểu thức so sánh
  - Toán tử: > (gt), < (lt) , >= (ge) , <= (le), == (eq), != (neq/ne), ! (not)
  - Biểu thức:
    - Ví dụ: if-then-else

      ```
      <tr th:class="${row.even}? 'even' : 'odd’”>

      <!– lồng nhau -->
      <tr th:class="${row.even}? (${row.first}? 'first' : 'even') : 'odd’”>

      <!--  Có thể bỏ bớt 1 về nếu trả về null. Ví dụ nếu biểu thức điều kiện là false thì trả về null, thì bỏ về sau dấu : -->
      <tr th:class="${row.even}? 'alt'">
      ```

    - Biểu thức mặc định (Toán tử Elvis): biểu thức không có vế then

      ```
      <p>Age: <span th:text="*{age}?: '(no age specified)'">27</span>.</p>
      <!  Nếu age không bằng null thì trả ra giá trị của age, nếu không thì trả về giá trị mặc định '(no age specified)’ Tương đương với -->
      <p>Age: <span th:text="*{age != null}? *{age} : '(no age specified)'">27</span>.</p>
      ```