第 2-4 课：模板引擎 Thymeleaf 高阶用法

上一课我们介绍了 Thymeleaf 最常用的使用语法，这一课我们继续学习 Thymeleaf 高阶的使用方式，并对这些使用方式进行总结分类。其实上一课的内容，基本可以满足 Thymeleaf 80% 的使用场景，高阶用法会在某些场景下提供更高效、便捷的使用方式。

内联 [[ ]]

如果不想通过 th 标签而是简单地访问 model 对象数据，或是想在 javascript 代码块里访问 model 中的数据，则要使用内联的方法。

内联文本：[[...]] 内联文本的表示方式，使用时，必须先用在 th:inline="text/javascript/none" 激活，th:inline 可以在父级标签内使用，甚至可以作为 body 的标签。内联文本比 th:text 的代码少，不利于原型显示。

页面 inline.html（文本内联）：

```
<div>
    <h1>内联</h1>
    <div th:inline="text" >
        <p>Hello, [[${userName}]] !</p>
        <br/>
    </div>
</div>
```

以上代码等价于：

```
<div>
    <h1>不使用内联</h1>
    <p th:text="'Hello, ' + ${userName} + ' !'"></p>
    <br/>
</div>
```

通过以上代码可以看出使用内联语法会更简洁一些。

如果想在脚本中使用后端传递的值，则必须使用脚本内联，脚本内联可以在 js 中取到后台传过来的参数：

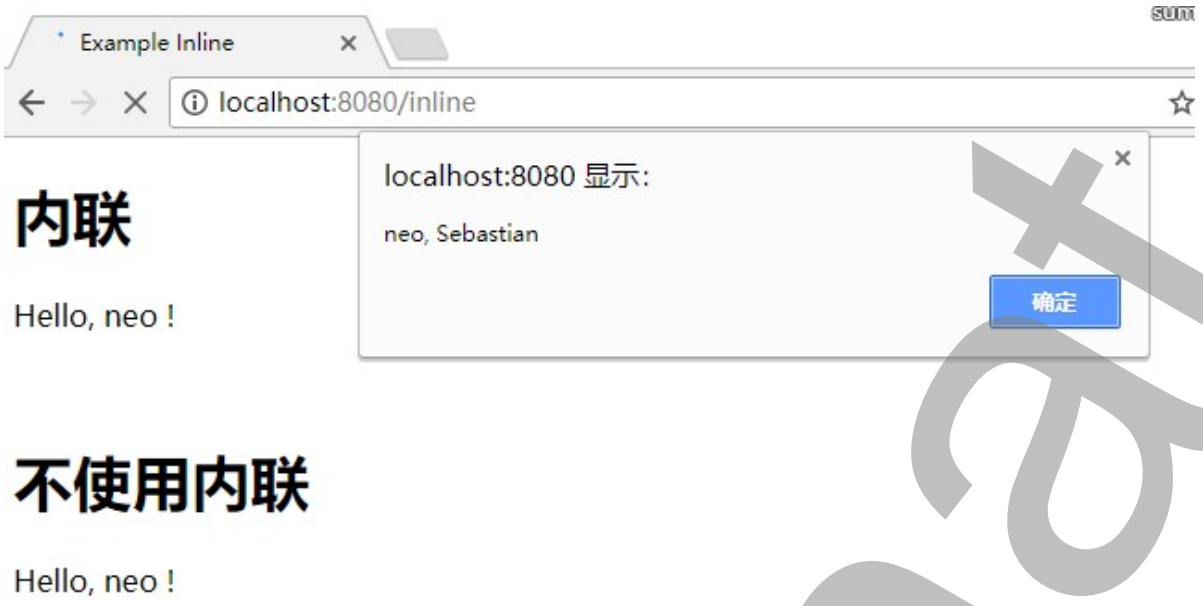
```
<script th:inline="javascript">
    var name = [[${userName}]] + ', Sebastian';
    alert(name);
</script>
```

这段脚本的含义是在访问页面的时候，根据后端传值拼接 name 值，并以 alert 的方式弹框展示。

后端传值：

```
@RequestMapping("/inline")
public String inline(ModelMap map) {
    map.addAttribute("userName", "neo");
    return "inline";
}
```

启动项目后在浏览器中输入该网址：http://localhost:8080/inline，则会出现下面的结果：



页面会先跳出一个 alert 提示框，然后再展示使用内联和不使用内联的页面内容。

基本对象

Thymeleaf 包含了一些基本对象，可以用于我们的视图中，这些基本对象使用 # 开头。

- `#ctx`：上下文对象
- `#vars`：上下文变量
- `#locale`：区域对象
- `#request`：（仅 Web 环境可用）HttpServletRequest 对象
- `#response`：（仅 Web 环境可用）HttpServletResponse 对象
- `#session`：（仅 Web 环境可用）HttpSession 对象
- `#servletContext`：（仅 Web 环境可用）ServletContext 对象

Thymeleaf 在 Web 环境中，有一系列的快捷方式用于访问请求参数、会话属性等应用属性，以其中几个常用的对象作为示例来演示。

- `#request`：直接访问与当前请求关联的 javax.servlet.http.HttpServletRequest 对象；
- `#session`：直接访问与当前请求关联的 javax.servlet.http.HttpSession 对象。

后台添加方法传值：

```
@RequestMapping("/object")
public String object(HttpServletRequest request) {
    request.setAttribute("request","i am request");
    request.getSession().setAttribute("session","i am session");
    return "object";
}
```

使用 request 和 session 分别传递了一个值，再来查看页面 object.html。

```
<body>
    <div >
        <h1>基本对象</h1>
        <p th:text="${#request.getAttribute('request')}">
        <br/>
        <p th:text="${session.session}"></p>
         Established locale country: <span th:text="${#locale.country}">CN</span>.
    </div>
</body>
```

启动项目后在浏览器中输入该网址：http://localhost:8080/object，则会出现下面的结果：

```
基本对象

i am request

i am session

Established locale country: CN.
```

第一个展示了 request 如何使用参数，第二行展示了 session 的使用，session 直接使用 `.` 即可获取到 session 中的值，最后展示了 locale 的用法。

内嵌变量

为了模板更加易用，Thymeleaf 还提供了一系列 Utility 对象（内置于 Context 中），可以通过 # 直接访问。

- dates：java.util.Date 的功能方法类
- calendars：类似 #dates，面向 java.util.Calendar
- numbers：格式化数字的功能方法类
- strings：字符串对象的功能类，contains、startWiths、prepending/appending 等
- objects：对 objects 的功能类操作
- bools：对布尔值求值的功能方法
- arrays：对数组的功能类方法
- lists：对 lists 的功能类方法
- sets：set 的实用方法
- maps：map 的实用方法
- …

下面用一段代码来举例说明一些常用的方法，页面是 utility.html。

1. dates

可以使用 dates 对日期格式化，创建当前时间等操作。

```
<!--格式化时间-->
<p th:text="${#dates.format(date, 'yyyy-MM-dd HH:mm:ss')}">neo</p>
<!--创建当前时间 精确到天-->
<p th:text="${#dates.createToday()}">neo</p>
<!--创建当前时间 精确到秒-->
<p th:text="${#dates.createNow()}">neo</p>
```

2. strings

strings 内置了一些对字符串经常使用的函数。

```
<!--判断是否为空-->
<p th:text="${#strings.isEmpty(userName)}">userName</p>
<!--判断 list 是否为空-->
<p th:text="${#strings.listIsEmpty(users)}">userName</p>
<!--输出字符串长度-->
<p th:text="${#strings.length(userName)}">userName</p>
<!--拼接字符串-->
<p th:text="${#strings.concat(userName,userName,userName)}"></p>
<!--创建自定长度的字符串-->
<p th:text="${#strings.randomAlphanumeric(count)}">userName</p>
```

后端传值：

```
@RequestMapping("/utility")
public String utility(ModelMap map) {
    map.addAttribute("userName", "neo");
    map.addAttribute("users", getUserList());
    map.addAttribute("count", 12);
    map.addAttribute("date", new Date());
    return "utility";
}
```

启动项目后在浏览器中输入该网址：http://localhost:8080/utility，则会出现下面的结果：

```
内嵌变量

2018-09-26 20:49:07

Wed Sep 26 00:00:00 CST 2018

Wed Sep 26 20:49:07 CST 2018

false

[false, false, false]

3

neoneoneo

QKERBVPRHFS9
```

接下来总结一下 Thymeleaf 表达式。

表达式

表达式共分为以下五类。

- 变量表达式：${...}
- 选择或星号表达式：*{...}

- 文字国际化表达式：#{...}
- URL 表达式：@{...}
- 片段表达式：~{...}

变量表达式

变量表达式即 OGNL 表达式或 Spring EL 表达式（在 Spring 术语中也叫 model attributes），类似 `${session.user.name}` 。

它们将以 HTML 标签的一个属性来表示：

```
<span th:text="${book.author.name}">
<li th:each="book : ${books}">
```

选择（星号）表达式

选择表达式很像变量表达式，不过它们用一个预先选择的对象来代替上下文变量容器（map）来执行，类似：`*{customer.name}` 。

被指定的 object 由 th:object 属性定义：

```
<div th:object="${book}">
  ...
  <span th:text="*{title}">...</span>
  ...
</div>
```

title 即为 book 的属性。

文字国际化表达式

文字国际化表达式允许我们从一个外部文件获取区域文字信息（.properties），用 Key 索引 Value，还可以提供一组参数（可选）。

```
#{main.title}
#{message.entrycreated(${entryId})}
```

可以在模板文件中找到这样的表达式代码：

```
<table>
  ...
  <th th:text="#{header.address.city}">...</th>
  <th th:text="#{header.address.country}">...</th>
  ...
</table>
```

URL 表达式

URL 表达式指的是把一个有用的上下文或回话信息添加到 URL，这个过程经常被叫做 URL 重写，比如 `@{/order/list}` 。

- URL 还可以设置参数：`@{/order/details(id=${orderId})}`
- 相对路径：`@{../documents/report}`

让我们看这些表达式：

```
<form th:action="@{/createOrder}">
<a href="main.html" th:href="@{/main}">
```

片段表达式

片段表达式是 3.x 版本新增的内容。片段表达式是一种标记的片段，并将其移动到模板中的方法。片段表达式的优势是，片段可以被复制或者作为参数传递给其他模板等。

最常见的用法是使用 th:insert 或 th:replace: 插入片段：

```
<div th:insert="~{commons :: main}">...</div>
```

也可以在页面的其他位置去使用：

```
<div th:with="frag=~{footer :: #main/text()}">
  <p th:insert="${frag}">
</div>
```

片段表达式可以有参数。

变量表达式和星号表达有什么区别

如果不考虑上下文的情况下，两者没有区别；星号语法是在选定对象上表达，而不是整个上下文。什么是选定对象？就是父标签的值，如下：

```
<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>
```

这是完全等价于：

```
<div th:object="${session.user}">
  <p>Name: <span th:text="${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>
```

当然，两种语法可以混合使用：

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

表达式支持的语法

字面（Literals）

- 文本文字（Text literals）：`'one text', 'Another one!',…`
- 数字文本（Number literals）：`0, 34, 3.0, 12.3,…`
- 布尔文本（Boolean literals）：`true, false`
- 空（Null literal）：`null`
- 文字标记（Literal tokens）：`one, sometext, main,…`

文本操作（Text operations）

- 字符串连接（String concatenation）：`+`
- 文本替换（Literal substitutions）：`|The name is ${name}|`

算术运算（Arithmetic operations）

- 二元运算符（Binary operators）：`+, -, *, /, %`
- 减号（单目运算符）Minus sign（unary operator）：`-`

布尔操作（Boolean operations）

- 二元运算符（Binary operators）：`and, or`
- 布尔否定（一元运算符）Boolean negation (unary operator)：`!, not`

比较和等价（Comparisons and equality）

- 比较（Comparators）：`>, <, >=, <= (gt, lt, ge, le)`
- 等值运算符（Equality operators）：`==, != (eq, ne)`

条件运算符（Conditional operators）

- If-then：`(if) ? (then)`
- If-then-else：`(if) ? (then) : (else)`
- Default: (value) ?：`(defaultvalue)`

所有这些特征可以被组合并嵌套：

```
'User is of type ' + (${user.isAdmin()} ? 'Administrator' : (${user.type} ?: 'Unknown'))
```

常用 th 标签

页面常用的 HTML 标签几乎都有 Thymeleaf 对应的 th 标签。

关键字	功能介绍	案例
th:id	替换 id	`<input th:id="'xxx' + ${collect.id}"/>`
th:text	文本替换	`<p th:text="${collect.description}">description</p>`
	支持 html 的	

th:utext	文本替换	`<p th:utext="${htmlcontent}">conten</p>`
th:object	替换对象	`<div th:object="${session.user}">`
th:value	属性赋值	`<input th:value="${user.name}" />`
th:with	变量赋值运算	`<div th:with="isEven=${prodStat.count}%2==0"></div>`
th:style	设置样式	`th:style="'display:' + @{(${sitrue} ? 'none' : 'inline-block')} + ''"`
th:onclick	点击事件	`th:onclick="'getCollect()'"`
th:each	属性赋值	`tr th:each="user,userStat:${users}">`
th:if	判断条件	`<a th:if="${userId == collect.userId}" >`
th:unless	和 th:if 判断相反	`<a th:href="@{/login}" th:unless=${session.user != null}>Login</a>`
th:href	链接地址	`<a th:href="@{/login}" th:unless=${session.user != null}>Login</a> />`
th:switch	多路选择 配合 th:case 使用	`<div th:switch="${user.role}">`
th:case	th:switch 的一个分支	`<p th:case="'admin'">User is an administrator</p>`
th:fragment	布局标签，定义一个代码片段，方便其他地方引用	`<div th:fragment="alert">`
th:include	布局标签，替换内容到引入	`<head th:include="layout :: htmlhead" th:with="title='xx'"></head> />`

	的文件	
th:replace	布局标签，替换整个标签到引入的文件	`<div th:replace="fragments/header :: title"></div>`
th:selected	selected 选择框选中	`th:selected="(${xxx.id} == ${configObj.dd})"`
th:src	图片类地址引入	`<img class="img-responsive" alt="App Logo" th:src="@{/img/logo.png}" />`
th:inline	定义 js 脚本可以使用变量	`<script type="text/javascript" th:inline="javascript">`
th:action	表单提交的地址	`<form action="subscribe.html" th:action="@{/subscribe}">`
th:remove	删除某个属性	`<tr th:remove="all">` 1.all：删除包含标签和所有的子节点； 2.body：不包含标记删除，但删除其所有的子节点； 3.tag：包含标记的删除，但不删除它的子节点； 4.all-but-first：删除所有包含标签的子节点，除了第一个。 5.none：什么也不做。这个值是有用的动态评估
th:attr	设置标签属性，多个属性可以用逗号分隔	比如 th:attr="src=@{/image/aa.jpg},title=#{logo}"，此标签不太优雅，一般用的比较少

还有非常多的标签，这里只列出最常用的几个，由于一个标签内可以包含多个 th:x 属性，其生效的优先级顺序为：

```
include,each,if/unless/switch/case,with,attr/attrprepend/attrappend,value/href,src ,etc,text/utext,fragment,remove。
```

Thymeleaf 配置

我们可以通过 application.properties 文件灵活的配置 Thymeleaf 的各项特性，以下为 Thymeleaf 的配置和默认参数：

```
# THYMELEAF (ThymeleafAutoConfiguration)
#开启模板缓存（默认值：true）
spring.thymeleaf.cache=true
#检查模板是否存在，然后再呈现
spring.thymeleaf.check-template=true
#检查模板位置是否正确（默认值:true）
spring.thymeleaf.check-template-location=true
#Content-Type的值（默认值：text/html）
spring.thymeleaf.content-type=text/html
#开启MVC Thymeleaf视图解析（默认值：true）
spring.thymeleaf.enabled=true
#模板编码
spring.thymeleaf.encoding=UTF-8
#要被排除在解析之外的视图名称列表，用逗号分隔
spring.thymeleaf.excluded-view-names=
#要运用于模板之上的模板模式。另见StandardTemplate-ModeHandlers(默认值：HTML5)
spring.thymeleaf.mode=HTML5
#在构建URL时添加到视图名称前的前缀（默认值：classpath:/templates/）
spring.thymeleaf.prefix=classpath:/templates/
#在构建URL时添加到视图名称后的后缀（默认值：.html）
spring.thymeleaf.suffix=.html
#Thymeleaf 模板解析器在解析器链中的顺序，默认情况下，它排第一位，顺序从1开始，只有在定义了额外的 TemplateResolv
er Bean 时才需要设置这个属性。
spring.thymeleaf.template-resolver-order=
#可解析的视图名称列表，用逗号分隔
spring.thymeleaf.view-names=
```

在实际项目中可以根据实际使用情况来修改。

总结

Thymeleaf 的使用方式非常灵活，可以结合 JS 来获取后端传递的值，Thymeleaf 本身也内嵌了很多对象和函数方便我们在页面来直接调用。Thymeleaf 通过不同的表达式来灵活的控制页面结构和内容，配合着 Spring Boot 的使用，Thymeleaf 可以通过多项参考来控制其特性。

> 点击这里下载源码。