# 1 表达式

th:text = "???" 双引号中间的为表达式。例如，

<div>

学号: <span th:text="${stu.id}"> 20190001</span>

</div>

相当于

System.out.println( stu.getId() )

也可以采用嵌入写法 (Inline)，例如，

<div>

姓名: [[${stu.name}]]

</div>

其中, [[???]] 或 [(???)] 会被理解为表达式

表达式中，可以采用字符串拼接的写法。例如，

<div th:text="'性别:' + ${stu.sex} " > </div>

相当于：

System.out.println( '性别:' + stu.getSex() )

表达式中，可以使用字符串替换的写法。例如，

<div th:text="| 手机号: ${stu.cellphone} |" ></div>

其中，表达式两边用竖杠包围。

# 2 上下文对象

#ctx: the context object.

#vars: the context variables.

#locale: the context locale.

#request: (only in Web Contexts) the HttpServletRequest object.

#response: (only in Web Contexts) the HttpServletResponse object.

#session: (only in Web Contexts) the HttpSession object.

#servletContext: (only in Web Contexts) the ServletContext object.

如

<div th:text="${#session.user.name}">

或者

<div th:text="${session.user.name}">

也就是说，session 和 #session 都可以引用到 HttpSession 对象

# 3 简单运算

表达式里可以有算术运算，逻辑运算等表达式

例如，

th:text = " ${ num + 2} "

th:text = " ${sex ? '男' :'女'} "

# 4 默认值

用 ?: 当变量为null时取默认值

th:text = " ${name} ?: '未登录' "

# 5 内置工具

可以调用内置工具，对数据进行转换。

例如，对日期变量的转换，

th:text = "${#calendars.format(birthday, 'yyyy-MM-dd') }">

其中，

#calendars是一个内置全局对象，调用它的format方法进行转换

例如，对数字格式进行转换，

th:text = "${#numbers.formatDecimal( height \* 100 , 0,0) }"

其中，

#numbers是一个内置全局对象，提供一些数字转换的方法

其他内置对象：

#execInfo: information about the template being processed.

#messages: methods for obtaining externalized messages inside variables expressions, in the same way as they would be obtained using #{…} syntax.

#uris: methods for escaping parts of URLs/URIs

#conversions: methods for executing the configured conversion service (if any).

#dates: methods for java.util.Date objects: formatting, component extraction, etc.

#calendars: analogous to #dates, but for java.util.Calendar objects.

#numbers: methods for formatting numeric objects.

#strings: methods for String objects: contains, startsWith, prepending/appending, etc.

#objects: methods for objects in general.

#bools: methods for boolean evaluation.

#arrays: methods for arrays.

#lists: methods for lists.

#sets: methods for sets.

#maps: methods for maps.

#aggregates: methods for creating aggregates on arrays or collections.

#ids: methods for dealing with id attributes that might be repeated (for example, as a result of an iteration).

# 6 if 判断

可以用 th:if 进行条件判断。例如，

<div th:if=" ${user!=null} ">

你好，[[ ${user} ]] !

<a href='logout'> 退出登录 </a>

</div>

<div th:if=" ${user==null} ">

你好，<a href='login'> 请登录 </a>

</div>

当条件成立时，显示这段HTML。当条件不成立时，不显示其中的HTML。

Thymeleaf 中没有 if..else的逻辑，如果需要，可以用switch..case替代。

# 7 switch..case 判断

用 switch..catch 表示多选一的判断。例如，

<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>

也可以用于表示 if..else的逻辑，例如，

<div class='row' th:switch=" ${user!=null} ">

<div th:case="true">

你好，[[ ${user} ]] !

<a href='logout'> 退出登录 </a>

</div>

<div th:case="false">

你好，<a href='login'> 请登录</a>

</div>

</div>

# 8 迭代遍历

可以用 th:each 进行迭代遍历，适用于List,Map,Array等对象。

例如，

<table>

<tr>

<th style='width:100px'> 学号 </th>

<th style='width:120px'> 姓名 </th>

<th style='width:50px'> 性别 </th>

<th style='width:140px'> 手机 </th>

</tr>

<tr th:each=" row: ${studentList} ">

<td th:text=" ${row.id} "> 00000000 </td>

<td th:text=" ${row.name } "> 名称 </td>

<td th:text=" ${row.sex ? '男' :'女'} "> 女 </td>

<td th:text=" ${row.cellphone} "> 13800000000 </td>

</tr>

</table>

# 9 链接路径

分为绝对路径，和相对路径。

Absolute URLs: 绝对路径，例如 @{http://www.thymeleaf.org}

Relative URLs: 相对路径，包括以下4种：

Page-relative: 相对于当前页面，例如 @{user/login.html}

Context-relative: 相对于 ContextPath

例如 @{/js/jquery.js} ，处理后会自动加上ContextPath前缀

Server-relative: 相对于服务器地址

例如 @{ ~/billing/processInvoice }

处理后前面加上当前URL的HOST部分，

例如 ， 变成 <http://x.com/billing/processInvoice>

Protocol-relative URLs: 相对于协议的

例如 @{//code.jquery.com/jquery-2.0.3.min.js}

处理后前面加上当前URL的协议

变成 <http://code.jquery.com/jquery-2.0.3.min.js> （或https://）

常用的就是 Context-relative，自动添加上ContextPath前缀的方式

例如，

<img src="../image/123.jpg" th:src="@{/image/123.jpg}" />

其中，

th:src 表示替换本标签的 src属性

在URL中可以嵌入变量，示例如下。

样例：'http://localhost:8080/gtvg/order/details?orderId=3'

<a href="details.html"

th:href="@{/gtvg/order/details(orderId=${o.id})}">view</a>

样例： '/gtvg/order/details?orderId=3'

<a href="details.html"

th:href="@{/order/details(orderId=${o.id})}">view</a>

样例：'/gtvg/order/3/details'

<a href="details.html"

th:href="@{/order/{orderId}/details(orderId=${o.id})}">view</a>

# 10 属性替换

th:text 表示替换当前标签的正文

例如， <span th:text="${stu.name} > </span>

th:src 表示替换当前标签的 src属性

例如，<img src="../image/default\_icon.jpg " th:src="${row.url}" />

th:href 表示替换当前标签的 href 属性

例如，<a href="#" th:href="@{/s/info}" > 查看 </a>

显示，th:xxx 就是替换当前标签的xxx属性

等效写法：

<input type="submit"

value="Subscribe!"

th:attr="value=#{subscribe.submit}"/>

# 11 包含代码片段

在一个文件里，可以包含另一个代码里的片段。

(1) 添加一个模板 /template/common.html

里面定义一个片断

<div th:fragment="footer">

…………

</div>

(2) 在另一个模板里包含它

<div th:insert=" ~{common::footer}"></div>

其中， ~{common::footer} 表示从名称为common的模板中，取得footer片段。可以看出，这里的模板名是相对于模板根目录的。Thymeleaf 会自动找到 /template/common.html 这个模板文件。

包含时有三种写法：

th:insert 表示将目标插在当前 <div> 下

th:replace 表示将目标直接替换本元素

th:include 表示将目标的内容插过来

在定义th:fragment 时还可以指定参数，参考其官方文档。

例如，在定义 fragment的时候，定义a,b两个参数

<div th:fragment="footer(a,b)">

<div style='padding:20px; margin-top:40px; '>

Spring [[${a}]] , [[${b}]]

</div>

</div>

在包含片段的时候，传入参数的值，

<div th:insert=" ~{common::footer( ${stu.id} ,${stu.name)}"></div>

# 12 内嵌写法

在 <script> 和 <style> 里也可以添加表达式

<script th:inline="javascript">

...

var username = [[${session.user.name}]];

...

</script>

<style th:inline="css">

.[[${classname}]] {

text-align: [[${align}]];

}

</style>

# 13 模板缓存

在spring-mvc.xml中配置时，cacheable属性表示是否缓存

<bean id="templateResolver"

class="org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolver">

<property name="prefix" value="/template/"/>

<property name="suffix" value=".html"/>

<property name="templateMode" value="HTML5"/>

<property name="cacheable" value="false"/>

<property name="characterEncoding" value="UTF-8" />

</bean>

在开发的时候，cacheable设为 false。这样每次修改模板，刷新后就能生效。

在发布的时候，cacheable 设为 true。这样模板会被Thymeleaf放到缓存里，加快处理速度。