# Thymeleaf模板引擎：

## 1引子：

Spring Boot提供了多种前端模板引擎的默认配置支持，但嵌入式容器JSP有限制。

所以主要用的比较多还是Spring boot与模板引擎Thymeleaf、Freemaker的集成，当然也可以做JSP的集成，但是需要修改的配置比较多，在这里我们主要介绍模板引擎Thymeleaf的集成，其余模板引擎的集成大家可以后续进行了解。

<https://www.cnblogs.com/paddix/p/8905531.html> 该博客有介绍3种模板引擎。

## **2Thymeleaf** 概念：

**Thymeleaf** 是面向 Web 和独立环境的现代服务器端 Java 模板引擎，能够处理 HTML、XML、JavaScript、CSS 甚至纯文本。

简单说， Thymeleaf 是一个跟 Velocity、FreeMarker 类似的模板引擎，它可以完全替代 JSP 。相较与其他的模板引擎，它有如下三个极吸引人的特点：

1.Thymeleaf 在有网络和无网络的环境下皆可运行，即它可以让美工在浏览器查看页面的静态效果，也可以让程序员在服务器查看带数据的动态页面效果。这是由于它支持 html 原型，然后在 html 标签里增加额外的属性来达到模板+数据的展示方式。浏览器解释 html 时会忽略未定义的标签属性，所以 thymeleaf 的模板可以静态地运行；当有数据返回到页面时，Thymeleaf 标签会动态地替换掉静态内容，使页面动态显示。

2.Thymeleaf 开箱即用的特性。它提供标准和spring标准两种方言，可以直接套用模板实现JSTL、 OGNL表达式效果，避免每天套模板、该jstl、改标签的困扰。同时开发人员也可以扩展和创建自定义的方言。

3.Thymeleaf 提供spring标准方言和一个与 SpringMVC 完美集成的可选模块，可以快速的实现表单绑定、属性编辑器、国际化等功能。

4、Thymeleaf 的设计从一开始就遵从 Web 标准，特别是 HTML5，这样就能创建完全符合验证的模板。

## 3优点：

其实就是HTML，直接可以用浏览器打开。对比JSP，在浏览器中会直接显示的代码片断。

<input tyep=”text” name="userName" value="${user.name}" />

Thymeleaf 标准方言将允许我们实现与以上功能相同的功能：

<input type="text" name="userName" value="James Carrot" th:value="${user.name}" />

浏览器不仅可以正确显示这些信息，而且还可以（可选地）在浏览器中静态打开原型时显示的值（可选地）指定一个值属性（在这种情况下为“James Carrot”），将在模板处理期间由${user.name}的评估得到的值代替。

## 4创建一个例子：

### 11.4.1在pom.xml文件中添加模板依赖：

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

### 11.4.2配置thymeleaf：

其实完全可以使用，可以不用配置，但是Spring Boot官方文档建议在开发时将缓存关闭，否则每次修改都要重新启动。

spring:

thymeleaf:

cache: **false**

prefix: classpath:/templates/

suffix: .html

check-template-location: **true**

encoding: UTF-8

mode: HTML5

解释：

prefix：指定模板所在的目录

suffix：指定视图文件后缀名

　　check-tempate-location: 检查模板路径是否存在

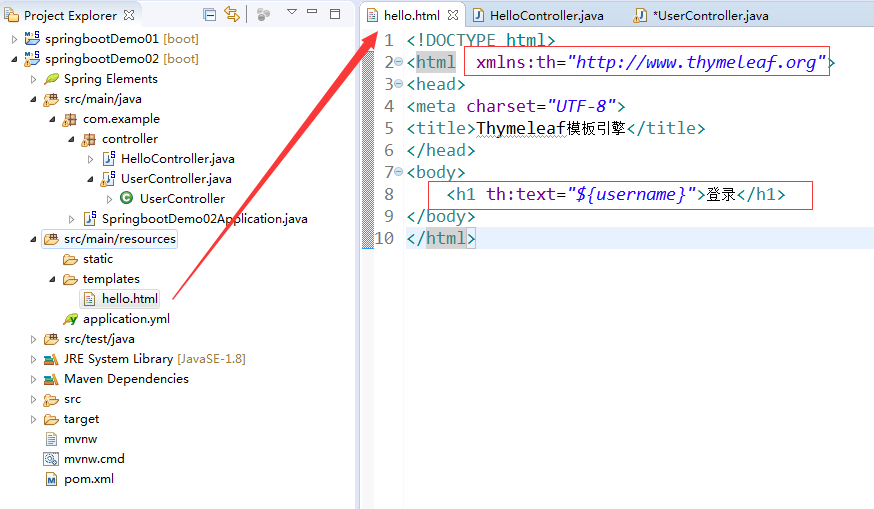
　　cache: 是否缓存，开发模式下设置为false，避免改了模板还要重启服务器，线上设置为true，可以提高性能。

### 11.4.3编写thymeleaf模板文件：

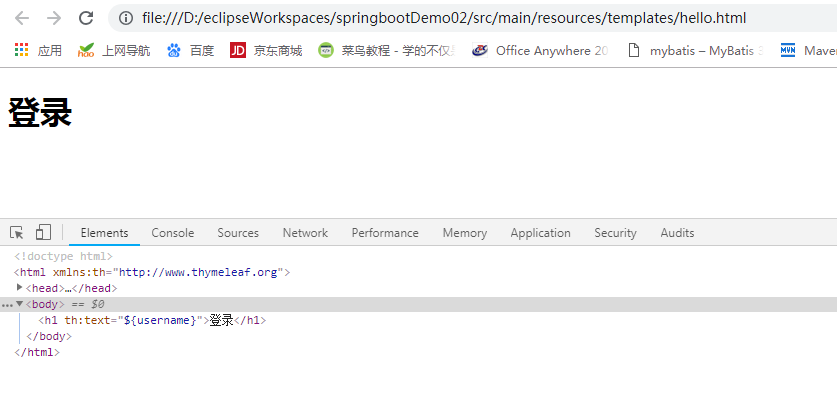
在classpath:resources下创建一个名为templates的文件夹，如果已经有，则无需创建。

创建一个hello.html的文件。

**在html节点中添加<html xmlns:th="http://www.thymeleaf.org">**



可以在浏览器中直接打开，看没有数据交互时候的效果：



如果是JSP，是必须将服务启动之后，才可以访问JSP页面。

### 11.4.5 编写控制器

新建模板controller类，使用@Controller注解，返回String或者ModelAndView值，该值指向templates目录下文件的路径。



运行测试：



### 11.4.6疑问

如果开发工具是IDE，html5可能会提示诸如“缺少th:\*属性命名空间定义”等字样的告警。

如果我们想让这些模板对于 HTML5 验证是有效的，需要做简单地修改，将属性语法改为data-前缀，(:)改为 (-) 即可：



这样，data-前缀就符合 HTML5 中的规范了。

注：在实际开发过程中，由于th:\*的辨识度往往必data-th-\*这种方式更高，所以很多人仍然是选择使用th:\*，这也是可以理解的。

## 5Thymeleaf 标准表达式语法：

Thymeleaf 提供了多种标准表达式包括：

* 简单表达式：
  + Variable expressions（变量表达式）${...}
  + Selection expressions（选择表达式）\*{...}
  + Message (i18n) expressions（消息表达式） #{...}
  + Link (URL) expressions（链接表达式）@{...}
  + Fragment expressions（分段表达式）~{...}
* 字面量:
  + 文本：'one text'、'Another one!'等；
  + 数值：0、34、3.0、12.3等；
  + 布尔：true、false
  + Null：null
  + Literal token(字面标记): one、sometext、 main等；
* 文本操作:
  + 字符串拼接：+
  + 文本替换：|The name is ${name}|
* 算术操作:
  + 二元运算符：+、-、 \*、/、%
  + 减号（单目运算符）：-
* 布尔操作：
  + 二元运算符：and、or
  + 布尔否定（一元运算符）：!、not
* 比较和等价：
  + 比较：>、<、>=、<=（gt、lt、ge、le）
  + 等价:==、!=（eq、ne）
* 条件运算符：
  + If-then：(if) ? (then)
  + If-then-else：(if) ? (then) : (else)
  + Default：(value) ?: (defaultvalue)
* 特殊标记：
  + No-Operation（无操作）：\_

#### Variable expressions（变量表达式）

变量表达式可以是OGNL表达式或者是 Spring EL，如果集成了Spring的话，可以在上下文变量（context variables ）中执行。

有关OGNL语法和功能的详细信息，请阅读[OGNL语言指南](http://commons.apache.org/proper/commons-ognl/) 在 Spring MVC 启用的应用程序中，OGNL将被替换为****SpringEL****，但其语法与OGNL非常相似（实际上，在大多数常见情况下完全相同）。

在Spring术语中，变量表达式也称为模型属性（model attributes）。 他们看起来像这样：

${session.user.name}

他们作为属性值或作为属性的一部分：

<span th:text="${book.author.name}">

上面的表达式在在OGNL和SpringEL中等价于：

((Book)context.getVariable("book")).getAuthor().getName()

这些变量表达式不仅涉及输出，还包括更复杂的处理，如条件判断、迭代等：

<li th:each="book : ${books}">

这里${books}从上下文中选择名为books的变量，并将其评估为可在th:each循环中使用的迭代器（iterable）。

更多 OGNL 的功能有：

/\* \* 使用点(.)来访问属性，等价于调用属性的 getter \*/

${person.father.name}

/\* \* 访问属性也可以使用([])块 \*/

${person['father']['name']}

/\* \* 如果对象是一个map，则点和块语法等价于调用其get(...)方法 \*/

${countriesByCode.ES}

${personsByName['Stephen Zucchini'].age}

/\* \* 在块语法中，也可以通过索引来访问数组或者集合 \*/

${personsArray[0].name}

/\* \* 可以调用方法，同时也支持参数 \*/

${person.createCompleteName()}

${person.createCompleteNameWithSeparator('-')}

#### 表达式基本对象（Expression Basic Objects）：

当对上下文变量评估 OGNL 表达式时，某些对象可用于表达式以获得更高的灵活性。 这些对象将被引用（按照OGNL标准），从＃符号开始：

* #ctx：上下文对象。
* #vars：上下文变量。
* #locale：上下文区域设置。
* #request:HttpServletRequest 对象(仅在 Web 上下文中）。
* #response：HttpServletResponse 对象(仅在 Web 上下文中）。
* #session:HttpSession对象(仅在 Web 上下文中）。
* #servletContext:ServletContext对象(仅在 Web 上下文中）。

所以我们可以这样做：

Established locale country: <span th:text="${#session.country}">US</span>.

#### 表达式工具对象（Expression Utility Objects）：

除了上面这些基本的对象之外，Thymeleaf 将为我们提供一组工具对象，这些对象将帮助我们在表达式中执行常见任务：

* #execInfo: 模版执行的信息
* #messages: 在变量内获取外部消息的方法 表达式，与使用＃{...}语法获得的方式相同。.
* #uris: 用于转义 URL/URI 部分的方法
* #conversions: 执行已配置的 conversion service
* #dates: java.util.Date对象的方法，比如格式化，组件提取等
* #calendars:类似于＃dates，但是对应于java.util.Calendar对象
* #numbers: 格式化数字对象的方法。
* #strings: String对象的方法，包括 contains、startsWith、prepending/appending等     等等
* #objects: 对象通常的方法
* #bools: 布尔判断的方法
* #arrays: array 方法
* #lists: list 方法
* #sets: set 方法
* #maps: map 方法
* #aggregates:在数组或集合上创建聚合的方法
* #ids: 用于处理可能重复的id属性的方法（例如，作为迭代的结果）。

下面是一个格式化日期的例子：

<p>

Today is: <span th:text="${#calendars.format(today,'dd MMMM yyyy')}">13 May 2011</span>

</p>

#### Literals（字面量）：

**文本**

文本文字只是在单引号之间指定的字符串。 他们可以包含任何字符，但您应该避免其中的任何单引号使用\'。

<p>

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

</p>

**数字**

数字文字就是数字。

<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>

**布尔**

布尔文字为“true”和“false”。 例如：:

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

在这个例子中，== false写在大括号之外，Thymeleaf 会做处理。如果是写在大括号内，那就是由 OGNL/SpringEL 引擎负责处理：

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

**null**

null 字面量使用如下：

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

**字面量标记**

数字、布尔和 null 字面实际上是\_字面量标记(literal tokens)\_的特殊情况。

这些标记允许在标准表达式中进行一点简化。 他们工作与文字文字（'...'）完全相同，但只允许使用字母（A-Z）和a-z'），数字（0-9），括号（[和]），点（.），连字符（-） 和下划线（\_`）。 所以没有空白，没有逗号等

标记不需任何引号。 所以我们可以这样做：

<div th:class="content">...</div>

用来代替:

<div th:class="'content'">...</div>

#### 算术运算：

支持算术运算：+, -, \*, / 和 %。

<div th:with="isEven=(${prodStat.count} % 2 == 0)">

请注意，这些运算符也可以在OGNL变量表达式本身中应用（在这种情况下将由OGNL执行，而不是Thymeleaf标准表达式引擎）：

<div th:with="isEven=${prodStat.count % 2 == 0}">

请注意，其中一些运算符存在文本别名：div (/)、 mod (%)。

#### 比较与相等：

表达式中的值可以与>, <, >= 和 <= 号进行比较，并且可以使用== 和 != 运算符来检查是否相等。 请注意， < 和 >符号不应该在XML属性值中使用，因此它们应被替换为&lt; 和&gt;。

<div th:if="${prodStat.count} &gt; 1">

<span th:text="'Execution mode is ' + ( (${execMode} == 'dev')? 'Development' : 'Production')">

一个更简单的替代方案可能是使用一些这些运算符存在的文本别名：gt (>), lt (<), ge (>=), le (<=), not (!). eq(==), neq/ne (!=)

#### Selection expressions（选择表达式）

选择表达式与变量表达式很像，区别在于它们是在当前选择的对象而不是整个上下文变量映射上执行。 他们看起来像这样：

\*{customer.name}

它们所作用的对象由th:object属性指定：

<div th:object="${book}">

...

<span th:text="\*{title}">...</span>

...

</div>

这等价于：

{

// th:object="${book}"

final Book selection = (Book) context.getVariable("book");

// th:text="\*{title}"

output(selection.getTitle());

}

#### Message (i18n) expressions（消息表达式）

消息表达式（通常称为文本外化、国际化或i18n）允许我们从外部源（.properties文件）检索特定于语言环境的消息，通过键引用它们（可选）应用一组参数。

在Spring应用程序中，这将自动与Spring的MessageSource机制集成。

#{main.title}

#{message.entrycreated(${entryId})}

在模版中的应用如下：

<table>

...

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

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

...

</table>

请注意，如果希望消息键由上下文变量的值确定，或者要将变量指定为参数，则可以在消息表达式中使用变量表达式：

#{${config.adminWelcomeKey}(${session.user.name})}

#### Link (URL) expressions（链接表达式）

链接表达式旨在构建URL并向其添加有用的上下文和会话信息（通常称为URL重写的过程）。

因此，对于部署在Web服务器的/myapp上下文中的Web应用程序，可以使用以下表达式：

<a th:href="@{/order/list}">...</a>

可以转成：

<a href="/myapp/order/list">...</a>

cookie没有启用下，如果我们需要保持会话，可以这样：

<a href="/myapp/order/list;jsessionid=23fa31abd41ea093">...</a>

URL 可以携带参数：

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

结果如下：

<!-- Note ampersands (&) should be HTML-escaped in tag attributes... -->

<a href="/myapp/order/details?id=23&amp;type=online">...</a>

链接表达式可以是相对的，在这种情况下，应用程序上下文将不会作为URL的前缀：

<a th:href="@{../documents/report}">...</a>

也 可以是服务器相对（同样，没有应用程序上下文前缀）：

<a th:href="@{~/contents/main}">...</a>

和协议相对（就像绝对URL，但浏览器将使用在显示的页面中使用的相同的HTTP或HTTPS协议）：

<a th:href="@{//static.mycompany.com/res/initial}">...</a>

当然，Link表达式可以是绝对的：

<a th:href="@{http://www.mycompany.com/main}">...</a>

### 模板：

#### 创建模板：

创建模板的两种方式：



直接使用id属性。

#### 引用模板：

"~{templatename::selector}" 名为templatename的模板上的指定标记选择器。 selector可以只是一个片段名。

"~{templatename}" ： 包含完整的模版 templatename

~{::selector}" or "~{this::selector}" 相同模版中的代码片段

引用模板的三种方式：

th：insert是最简单的：它将简单地插入指定的片段作为正文 的主标签。

th：replace用指定实际片段来替换其主标签。

th：include类似于th：insert，但不是插入片段它只插入此片段的内容。

通过模板名称引用：

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

通过id引用：

<div th:insert="~{footer :: #copy-section}"></div>

### 设置值：

#### 设置值到指定的属性：

现在，你可能会想到像：

<input type="submit" value="Subscribe!" th:attr="value=#{subscribe.submit}"/>

上面可以指定一个属性的值，但看上去有点丑陋，且并不是最优雅的方式。 通常，你将使用其他任务的th：\*属性 设置特定的标签属性（而不仅仅是像“th：attr”这样的任意属性）。

例如，要设置value属性，使用th：value：

<input type="submit" value="Subscribe!" th:value="#{subscribe.submit}"/>

要设置 action 属性，使用th:action：

<form action="subscribe.html" th:action="@{/subscribe}">

Thymeleaf 提供了很多属性，每个都针对特定的HTML5属性：

|  |  |  |
| --- | --- | --- |
| th:abbr | th:accept | th:accept-charset |
| th:accesskey | th:action | th:align |
| th:alt | th:archive | th:audio |
| th:autocomplete | th:axis | th:background |
| th:bgcolor | th:border | th:cellpadding |
| th:cellspacing | th:challenge | th:charset |
| th:cite | th:class | th:classid |
| th:codebase | th:codetype | th:cols |
| th:colspan | th:compact | th:content |
| th:contenteditable | th:contextmenu | th:data |
| th:datetime | th:dir | th:draggable |
| th:dropzone | th:enctype | th:for |
| th:form | th:formaction | th:formenctype |
| th:formmethod | th:formtarget | th:fragment |
| th:frame | th:frameborder | th:headers |
| th:height | th:high | th:href |
| th:hreflang | th:hspace | th:http-equiv |
| th:icon | th:id | th:inline |
| th:keytype | th:kind | th:label |
| th:lang | th:list | th:longdesc |
| th:low | th:manifest | th:marginheight |
| th:marginwidth | th:max | th:maxlength |
| th:media | th:method | th:min |
| th:name | th:onabort | th:onafterprint |
| th:onbeforeprint | th:onbeforeunload | th:onblur |
| th:oncanplay | th:oncanplaythrough | th:onchange |
| th:onclick | th:oncontextmenu | th:ondblclick |
| th:ondrag | th:ondragend | th:ondragenter |
| th:ondragleave | th:ondragover | th:ondragstart |
| th:ondrop | th:ondurationchange | th:onemptied |
| th:onended | th:onerror | th:onfocus |
| th:onformchange | th:onforminput | th:onhashchange |
| th:oninput | th:oninvalid | th:onkeydown |
| th:onkeypress | th:onkeyup | th:onload |
| th:onloadeddata | th:onloadedmetadata | th:onloadstart |
| th:onmessage | th:onmousedown | th:onmousemove |
| th:onmouseout | th:onmouseover | th:onmouseup |
| th:onmousewheel | th:onoffline | th:ononline |
| th:onpause | th:onplay | th:onplaying |
| th:onpopstate | th:onprogress | th:onratechange |
| th:onreadystatechange | th:onredo | th:onreset |
| th:onresize | th:onscroll | th:onseeked |
| th:onseeking | th:onselect | th:onshow |
| th:onstalled | th:onstorage | th:onsubmit |
| th:onsuspend | th:ontimeupdate | th:onundo |
| th:onunload | th:onvolumechange | th:onwaiting |
| th:optimum | th:pattern | th:placeholder |
| th:poster | th:preload | th:radiogroup |
| th:rel | th:rev | th:rows |
| th:rowspan | th:rules | th:sandbox |
| th:scheme | th:scope | th:scrolling |
| th:size | th:sizes | th:span |
| th:spellcheck | th:src | th:srclang |
| th:standby | th:start | th:step |
| th:style | th:summary | th:tabindex |
| th:target | th:title | th:type |
| th:usemap | th:value | th:valuetype |
| th:vspace | th:width | th:wrap |
| th:xmlbase | th:xmllang | th:xmlspace |

#### 附加和添加前缀：

th:attrappend 和 th:attrprepend 用于附加和添加前缀属性。例如

<input type="button" value="Do it!" class="btn" th:attrappend="class=${' ' + cssStyle}" />

执行模版， cssStyle 变量设置为 "warning"时，输出如下：

<input type="button" value="Do it!" class="btn warning" />

同时，有 th:classappend 和 th:styleappend 用于设置CSS 的 class 和 style。例如：

<tr th:each="prod : ${prods}" class="row" th:classappend="${prodStat.odd}? 'odd'">

#### 固定值布尔属性：

HTML具有布尔属性的概念，没有值的属性意味着该值为“true”。 在XHTML中，这些属性只取一个值，即它本身。

例如，checked:

<input type="checkbox" name="option2" checked /> <!-- HTML -->

<input type="checkbox" name="option1" checked="checked" /> <!-- XHTML -->

标准方言包括允许您通过评估条件来设置这些属性，如果评估为true，则该属性将被设置为其固定值，如果评估为false，则不会设置该属性：

<input type="checkbox" name="active" th:checked="${user.active}" />

标准方言中存在以下固定值布尔属性：

|  |  |  |
| --- | --- | --- |
| th:async | th:autofocus | th:autoplay |
| th:checked | th:controls | th:declare |
| th:default | th:defer | th:disabled |
| th:formnovalidate | th:hidden | th:ismap |
| th:loop | th:multiple | th:novalidate |
| th:nowrap | th:open | th:pubdate |
| th:readonly | th:required | th:reversed |
| th:scoped | th:seamless | th:selected |

### 迭代器：

#### 基本的迭代：

th:each将循环 array 或 list 中的元素并重复打印一组标签，语法相当于 Java foreach 表达式：

<li th:each="book : ${books}" th:text="${book.title}">En las Orillas del Sar</li>

可以使用th：each属性进行遍历的对象包括：

任何实现java.util.Iterable的对象

任何实现java.util.Enumeration的对象

任何实现java.util.Iterator的对象，其值将被迭代器返回，而不需要在内存中缓存所有的值

任何实现java.util.Map的对象。 迭代映射时，迭代变量 将是java.util.Map.Entry类

任何数组

任何其他对象将被视为包含对象本身的单值列表

#### 状态变量：

Thymeleaf 提供 状态变量（status variable） 来跟踪迭代器的状态。

th:each属性中，定义了如下状态变量：

* index 属性是当前 迭代器索引（iteration index），从0开始
* count 属性是当前 迭代器索引（iteration index），从1开始
* size 属性是迭代器元素的总数
* current 是当前 迭代变量（iter variable）
* even/odd 判断当前迭代器是否是 even 或 odd
* first 判断当前迭代器是否是第一个
* last 判断当前迭代器是否是最后

看下面的例子：

<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>

状态变量（在本示例中为“iterStat”）在th：each中定义了。

我们来看看模板的处理后的结果：

<!DOCTYPE html>

<html>

<head>

<title>Good Thymes Virtual Grocery</title>

<meta content="text/html; charset=UTF-8" http-equiv="Content-Type"/>

<link rel="stylesheet" type="text/css" media="all" href="/gtvg/css/gtvg.css" />

</head>

<body>

<h1>Product list</h1>

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

</tr>

<tr class="odd">

<td>Fresh Sweet Basil</td>

<td>4.99</td>

<td>yes</td>

</tr>

<tr>

<td>Italian Tomato</td>

<td>1.25</td>

<td>no</td>

</tr>

<tr class="odd">

<td>Yellow Bell Pepper</td>

<td>2.50</td>

<td>yes</td>

</tr>

<tr>

<td>Old Cheddar</td>

<td>18.75</td>

<td>yes</td>

</tr>

</table>

<p>

<a href="/gtvg/" shape="rect">Return to home</a>

</p>

</body>

</html>

请注意，我们的迭代状态变量已经运行良好，建立只有奇数行具有 “odd” CSS 类。

如果您没有明确设置状态变量，则 Thymeleaf 将始终创建一个状态变量，可以通过后缀“Stat”获取到迭代变量的名称：

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

</tr>

<tr th:each="prod : ${prods}" th:class="${prodStat.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>

### 条件语句：

#### "if" 和 "unless"：

th:if 属性用法如下：

<a href="comments.html"

th:href="@{/product/comments(prodId=${prod.id})}"

th:if="${not #lists.isEmpty(prod.comments)}">view</a>

请注意，th:if 属性不仅是将评估布尔条件。 它的功能有点超出这一点，它将按照这些规则评估指定的表达式：

* 如果值不为 null：
  + 如果值为布尔值，则为true。
  + 如果值是数字，并且不为零
  + 如果值是一个字符且不为零
  + 如果value是String，而不是“false”，“off”或“no”
  + 如果值不是布尔值，数字，字符或字符串。
* 如果值为null，则th：if 将为 false。

另外，th:if有一个相反的属性th:unless，前面的例子改为：

<a href="comments.html"

th:href="@{/comments(prodId=${prod.id})}"

th:unless="${#lists.isEmpty(prod.comments)}">view</a>

#### switch 语句：

switch 语句使用th:switch / th:case 属性集合来实现：

<div th:switch="${user.role}">

<p th:case="'admin'">User is an administrator</p>

<p th:case="#{roles.manager}">User is a manager</p>

</div>

请注意，只要一个th：case属性被评估为'true'，每个其他同一个 switch 语句中的th：case属性将被评估为false。

th:case="\*"来设置默认选项：

<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>

### 局部变量：

在迭代器中，我们可以使用局部变量prod：

<tr th:each="prod : ${prods}">

...

</tr>

Thymeleaf 为您提供了一种在不使用迭代的情况下声明局部变量的方法th：with属性，其语法与属性值类似：

<div th:with="firstPer=${persons[0]}">

<p>

The name of the first person is <span th:text="${firstPer.name}">Julius Caesar</span>.

</p>

</div>

当th：with被处理时，firstPer变量创建为局部变量，并添加到来自上下文的变量 map 中，以便它是可用于评估以及在上下文中声明的任何其他变量，但只能在包含<div>标签的范围内。

可以同时定义多个变量，赋值语法为：

<div th:with="firstPer=${persons[0]},secondPer=${persons[1]}">

<p>

The name of the first person is <span th:text="${firstPer.name}">Julius Caesar</span>.

</p>

<p>

But the name of the second person is

<span th:text="${secondPer.name}">Marcus Antonius</span>.

</p>

</div>

th：with属性允许重用在同一属性中定义的变量：

<div th:with="company=${user.company + ' Co.'},account=${accounts[company]}">...</div>