

Thymeleaf 是一款用于渲染 XML/XHTML/HTML5 内容的模板引擎。它与 JSP, Velocity, FreeMaker 等模板引擎类似, 也可以轻易地与 Spring MVC 等 Web 框架集成。与其它模板引擎相比, Thymeleaf 最大的特点是, 即使不启动 Web 应用, 也可以直接在浏览器中打开并正确显示模板页面。

## 1. Thymeleaf 简介

Thymeleaf 是新一代 Java 模板引擎, 与 Velocity、FreeMarker 等传统 Java 模板引擎不同, Thymeleaf 支持 HTML 原型, 其文件后缀为“.html”, 因此它可以直接被浏览器打开, 此时浏览器会忽略未定义的 Thymeleaf 标签属性, 展示 thymeleaf 模板的静态页面效果; 当通过 Web 应用程序访问时, Thymeleaf 会动态地替换掉静态内容, 使页面动态显示。

Thymeleaf 通过在 html 标签中, 增加额外属性来达到“模板+数据”的展示方式, 示例代码如下。

```
1 <!DOCTYPE html><html lang="en" xmlns:th="http://www.thymeleaf.org"><head>
2   <meta charset="UTF-8">
3   <title>Title</title></head><body><!--th:text 为 Thymeleaf 属性, 用于在展示文本-->
4   <h1 th:text="欢迎来到Thymeleaf">欢迎您访问静态页面 HTML</h1>
5   </body>
6   </html>
```

当直接使用浏览器打开时, 浏览器展示结果如下。

```
1 欢迎您访问静态页面HTML
```

当通过 Web 应用程序访问时, 浏览器展示结果如下。

```
1 欢迎来到Thymeleaf
```

## Thymeleaf 的特点

Thymeleaf 模板引擎具有以下特点:

- 动静结合: Thymeleaf 既可以直接使用浏览器打开, 查看页面的静态效果, 也可以通过 Web 应用程序进行访问, 查看动态页面效果。
- 开箱即用: Thymeleaf 提供了 Spring 标准方言以及一个与 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。
- 多方言支持: 它提供了 Thymeleaf 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; 必要时, 开发人员也可以扩展和创建自定义的方言。

- 与 SpringBoot 完美整合：SpringBoot 为 Thymeleaf 提供了默认配置，并且还为 Thymeleaf 设置了视图解析器，因此 Thymeleaf 可以与 Spring Boot 完美整合。

## 2. Thymeleaf 语法规则

在使用 Thymeleaf 之前，首先要在页面的 html 标签中声明名称空间，示例代码如下。

<http://www.thymeleaf.org>

在 html 标签中声明此名称空间，可避免编辑器出现 html 验证错误，但这一步并非必须进行的，即使我们不声明该命名空间，也不影响 Thymeleaf 的使用。

Thymeleaf 作为一种模板引擎，它拥有自己的语法规则。Thymeleaf 语法分为以下 2 类：

- 标准表达式语法
- th 属性

### 2.1 标准表达式语法

Thymeleaf 模板引擎支持多种表达式：

- 变量表达式：\${...}
- 选择变量表达式：\*{...}
- 链接表达式：@{...}
- 国际化表达式：#{...}
- 片段引用表达式：~{...}

#### 2.1.1 变量表达式

使用 \${} 包裹的表达式被称为变量表达式，该表达式具有以下功能：

- 获取对象的属性和方法
- 使用内置的基本对象
- 使用内置的工具对象

##### ① 获取对象的属性和方法

使用变量表达式可以获取对象的属性和方法，例如，获取 person 对象的 lastName 属性，表达式形式如下：



##### ② 使用内置的基本对象

使用变量表达式还可以使用内置基本对象，获取内置对象的属性，调用内置对象的方法。Thymeleaf 中常用的内置基本对象如下：

- #ctx：上下文对象；
- #vars：上下文变量；
- #locale：上下文的语言环境；

- #request: HttpServletRequest 对象（仅在 Web 应用中可用）；
- #response: HttpServletResponse 对象（仅在 Web 应用中可用）；
- #session: HttpSession 对象（仅在 Web 应用中可用）；
- #servletContext: ServletContext 对象（仅在 Web 应用中可用）。

例如，我们通过以下 2 种形式，都可以获取到 session 对象中的 map 属性：

### ③ 使用内置的工具对象

除了能使用内置的基本对象外，变量表达式还可以使用一些内置的工具对象。

- strings: 字符串工具对象，常用方法有：equals、equalsIgnoreCase、length、trim、toUpperCase、toLowerCase、indexOf、substring、replace、startsWith、endsWith、contains 和 containsIgnoreCase 等；
- numbers: 数字工具对象，常用的方法有：formatDecimal 等；
- bools: 布尔工具对象，常用的方法有：isTrue 和 isFalse 等；
- arrays: 数组工具对象，常用的方法有：toArray、length、isEmpty、contains 和 containsAll 等；
- lists/sets: List/Set 集合工具对象，常用的方法有：toList、size、isEmpty、contains、containsAll 和 sort 等；
- maps: Map 集合工具对象，常用的方法有：size、isEmpty、containsKey 和 containsValue 等；
- dates: 日期工具对象，常用的方法有：format、year、month、hour 和 createNow 等。

例如，我们可以使用内置工具对象 strings 的 equals 方法，来判断字符串与对象的某个属性是否相等，代码如下。

### 2.1.2 选择变量表达式

选择变量表达式与变量表达式功能基本一致，只是在变量表达式的基础上增加了与 th:object 的配合使用。当使用 th:object 存储一个对象后，我们可以在其后代中使用选择变量表达式（\*{...}）获取该对象中的属性，其中，“\*”即代表该对象。

```
<div th:object="${session.user}" > <p th:text="*{fisrtName}">      </p></div>
```

th:object 用于存储一个临时变量，该变量只在该标签及其后代中有效，在后面的内容“th 属性”中我详细介绍。

### 2.1.3 链接表达式

不管是静态资源的引用，还是 form 表单的请求，凡是链接都可以用链接表达式（@{...}）。

链接表达式的形式结构如下：

- 无参请求：@{/xxx}

- 有参请求: @{/xxx(k1=v1,k2=v2)}

例如使用链接表达式引入 css 样式表, 代码如下。

```
<link href="asserts/css/signin.css" th:href="@{/asserts/css/signin.css}" rel="stylesheet">
```

#### 2.1.4 国际化表达式

消息表达式一般用于国际化的场景。结构如下。

██████████

注意: 此处了解即可, 我们会在后面的章节中详细介绍。

#### 2.1.5 片段引用表达式

片段引用表达式用于在模板页面中引用其他的模板片段, 该表达式支持以下 2 中语法结构:

- 推荐: ~{templatename::fragmentname}
- 支持: ~{templatename::#id}

以上语法结构说明如下:

- **templatename**: 模版名, Thymeleaf 会根据模版名解析完整路径: /resources/templates/templatename.html, 要注意文件的路径。
- **fragmentname**: 片段名, Thymeleaf 通过 th:fragment 声明定义代码块, 即:  
th:fragment="fragmentname"
- **id**: HTML 的 id 选择器, 使用时要在前面加上 # 号, 不支持 class 选择器。

### 2.2 th 属性

Thymeleaf 还提供了大量的 th 属性, 这些属性可以直接在 HTML 标签中使用, 其中常用 th 属性及其示例如下表。

| 属性         | 描述   | 示例   |
|------------|--|--|
| th:id      | 替换 HTML 的 id 属性  | <code>&lt;input id="html-id" th:id="thymeleaf-id" /&gt;</code>   |
| th:text    | 文本替换, 转义特殊字符   | <code>&lt;h1 th:text="hello, bianchengbang"&gt;hello&lt;/h1&gt;</code>   |
| th:utext   | 文本替换, 不转义特殊字符  | <code>&lt;div th:utext="'&lt;h1&gt;欢迎来到编程帮! &lt;/h1&gt;'"&gt;欢迎你&lt;/div&gt;</code>  |
| th:object  | 在父标签选择对象, 子标签使用 <code>{...}</code> 选择表达式选取值。<br>没有选择对象, 那子标签使用选择表达式和 <code>{...}</code> 变量表达式是一样的效果。<br>同时即使选择了对象, 子标签仍然可以使用变量表达式。 | <code>&lt;div th:object="\${session.user}"&gt;<br/>  &lt;p th:text="*{firstName}"&gt;{firstName}&lt;/p&gt;<br/>&lt;/div&gt;</code>   |
| th:value   | 替换 value 属性  | <code>&lt;input th:value = "\${user.name}" /&gt;</code>  |
| th:with    | 局部变量赋值运算   | <code>&lt;div th:with="isEvens = \${prodStat.count}%2 == 0" th:text="\${isEvens}"&gt;&lt;/div&gt;</code>   |
| th:style   | 设置样式   | <code>&lt;div th:style="color:#F00; font-weight:bold"&gt;编程帮 www.biancheng.net&lt;/div&gt;</code>  |
| th:onclick | 点击事件   | <code>&lt;td th:onclick = "'getInfo()'"&gt;&lt;/td&gt;</code>  |
| th:each    | 遍历, 支持 Iterable、Map、数组等。   | <code>&lt;table&gt;<br/>  &lt;tr th:each="m:\${session.map}"&gt;<br/>    &lt;td th:text="{m.getKey()}"&gt;&lt;/td&gt;<br/>    &lt;td th:text="{m.getValue()}"&gt;&lt;/td&gt;<br/>  &lt;/tr&gt;<br/>&lt;/table&gt;</code> |
| th:if      | 根据条件判断是否需要展示此标签  | <code>&lt;a th:if = "\${userId == collect.userId}"&gt;</code>  |
| th:unless  | 和 th:if 判断相反, 满足条件时不显示   | <code>&lt;div th:unless="\${m.getKey()=='name'}"&gt;&lt;/div&gt;</code>  |

|             |  |   |
|-------------|--|---|
| th:unless   | 和 th:if 判断相反, 满足条件时不显示   | <code>&lt;div th:unless="\${m.getKey()=='name'}"&gt;&lt;/div&gt;</code>   |
| th:switch   | 与 Java 的 switch case 语句类似<br>通常与 th:case 配合使用, 根据不同的条件展示不同的内容                        | <code>&lt;div th:switch="\${name}"&gt;<br/>  &lt;span th:case="a"&gt;编程帮&lt;/span&gt;<br/>  &lt;span th:case="b"&gt;www.biancheng.net&lt;/span&gt;<br/>&lt;/div&gt;</code>  |
| th:fragment | 模板布局, 类似 JSP 的 tag, 用来定义一段被引用或包含的模板片段  | <code>&lt;footer th:fragment="footer"&gt;插入的内容&lt;/footer&gt;</code>  |
| th:insert   | 布局标签;<br>将使用 th:fragment 属性指定的模板片段 (包含标签) 插入到当前标签中。                                  | <code>&lt;div th:insert="commons/bar::footer"&gt;&lt;/div&gt;</code>  |
| th:replace  | 布局标签;<br>使用 th:fragment 属性指定的模板片段 (包含标签) 替换当前整个标签。                                   | <code>&lt;div th:replace="commons/bar::footer"&gt;&lt;/div&gt;</code>   |
| th:selected | select 选择框选中   | <code>&lt;select&gt;<br/>  &lt;option&gt;——&lt;/option&gt;<br/>  &lt;option th:selected="\${name=='a'}"&gt;<br/>    编程帮<br/>  &lt;/option&gt;<br/>  &lt;option th:selected="\${name=='b'}"&gt;<br/>    www.biancheng.net<br/>  &lt;/option&gt;<br/>&lt;/select&gt;</code> |
| th:src      | 替换 HTML 中的 src 属性  | <code>&lt;img th:src="@{/asserts/img/bootstrap-solid.svg}" src="asserts/img/bootstrap-solid.svg" /&gt;</code>   |
| th:inline   | 内联属性;<br>该属性有 text、none、javascript 三种取值,<br>在 <script> 标签中使用时, js 代码中可以获取到后台传递页面的对象。 | <code>&lt;script type="text/javascript" th:inline="javascript"&gt;<br/>  var name = /*[[{name}]]*/ 'bianchengbang';<br/>  alert(name)<br/>&lt;/script&gt;</code>  |
| th:action   | 替换表单提交地址   | <code>&lt;form th:action="@{/user/login}" th:method="post"&gt;&lt;/form&gt;</code>  |

## 3. Thymeleaf 公共页面抽取

在 Web 项目中，通常会存在一些公共页面片段（重复代码），例如头部导航栏、侧边菜单栏和公共的 js css 等。我们一般会把这些公共页面片段抽取出来，存放在一个独立的页面中，然后再由其他页面根据需要进行引用，这样可以消除代码重复，使页面更加简洁。

### 3.1 抽取公共页面

Thymeleaf 作为一种优雅且高度可维护的模板引擎，同样支持公共页面的抽取和引用。我们可以将公共页面片段抽取出来，存放在一个独立的页面中，并使用 Thymeleaf 提供的 `th:fragment` 属性为这些抽取出来的公共页面片段命名。

#### 示例 1

将公共页面片段抽取出来，存放在 `commons.html` 中，代码如下。

```
<div th:fragment="fragment-name" id="fragment-id"> <span> </span></div>
```

### 3.2 引用公共页面

在 Thymeleaf 中，我们可以使用以下 3 个属性，将公共页面片段引入到当前页面中。

- `th:insert`：将代码块片段整个插入到使用了 `th:insert` 属性的 HTML 标签中；
- `th:replace`：将代码块片段整个替换使用了 `th:replace` 属性的 HTML 标签中；
- `th:include`：将代码块片段包含的内容插入到使用了 `th:include` 属性的 HTML 标签中。

使用上 3 个属性引入页面片段，都可以通过以下 2 种方式实现。

- `~{templatename::selector}`：模板名::选择器
- `~{templatename::fragmentname}`：模板名::片段名

通常情况下，`~{}` 可以省略，其行内写法为 `[[~{...}]]` 或 `[(~{...})]`，其中 `[[~{...}]]` 会转义特殊字符，`[(~{...})]` 则不会转义特殊字符。

#### 示例 2

1. 在页面 `fragment.html` 中引入 `commons.html` 中声明的页面片段，可以通过以下方式实现。

```
1 <!--th:insert 片段名引入-->
2 <div th:insert="commons::fragment-name"></div>
3 <!--th:insert id 选择器引入-->
4 <div th:insert="commons::#fragment-id"></div>
5 -----
6 <!--th:replace 片段名引入-->
7 <div th:replace="commons::fragment-name"></div>
8 <!--th:replace id 选择器引入-->
```

```

9 <div th:replace="commons::#fragment-id"></div>
10 -----
11 <!--th:include 片段名引入-->
12 <div th:include="commons::fragment-name"></div>
13 <!--th:include id 选择器引入-->
14 <div th:include="commons::#fragment-id"></div>

```

2. 启动 Spring Boot，使用浏览器访问 fragment.html，查看源码，结果如下。

```

1 <!--th:insert 片段名引入-->
2 <div>
3     <div id="fragment-id">
4         <span>公共页面片段</span>
5     </div>
6 </div>
7 <!--th:insert id 选择器引入-->
8 <div>
9     <div id="fragment-id">
10         <span>公共页面片段</span>
11     </div>
12 </div>
13 -----
14 <!--th:replace 片段名引入-->
15 <div id="fragment-id">
16     <span>公共页面片段</span>
17 </div>
18 <!--th:replace id 选择器引入-->
19 <div id="fragment-id">
20     <span>公共页面片段</span>
21 </div>
22 -----
23 <!--th:include 片段名引入-->
24 <div>
25     <span>公共页面片段</span>
26 </div>
27 <!--th:include id 选择器引入-->
28 <div>
29     <span>公共页面片段</span>

```

### 3.3 传递参数

Thymeleaf 在抽取和引入公共页面片段时，还可以进行参数传递，大致步骤如下：

1. 传入参数；
2. 使用参数。

#### 3.3.1 传入参数

引用公共页面片段时，我们可以通过以下 2 种方式，将参数传入到被引用的页面片段中：

- 模板名::选择器名或片段名(参数1=参数值1,参数2=参数值2)
- 模板名::选择器名或片段名(参数值1,参数值2)

注：若传入参数较少时，一般采用第二种方式，直接将参数值传入页面片段中；若参数较多时，建议使用第一种方式，明确指定参数名和参数值，。

示例代码如下：

```

1 <!--th:insert 片段名引入-->
2 <div th:insert="commons::fragment-name(var1='insert-name',var2='insert-name2')"></div>
3 <!--th:insert id 选择器引入-->
4 <div th:insert="commons::#fragment-id(var1='insert-id',var2='insert-id2')"></div>
5 -----
6 <!--th:replace 片段名引入-->
7 <div th:replace="commons::fragment-name(var1='replace-name',var2='replace-name2')">
  </div>
8 <!--th:replace id 选择器引入-->
9 <div th:replace="commons::#fragment-id(var1='replace-id',var2='replace-id2')"></div>
10 -----
11 <!--th:include 片段名引入-->
12 <div th:include="commons::fragment-name(var1='include-name',var2='include-name2')">
  </div>
13 <!--th:include id 选择器引入-->
14 <div th:include="commons::#fragment-id(var1='include-id',var2='include-id2')"></div>/

```

#### 3.3.2 使用参数

在声明页面片段时，我们可以在片段中声明并使用这些参数，例如：



```

1 <!--使用 var1 和 var2 声明传入的参数，并在该片段中直接使用这些参数 -->
2 <div th:fragment="fragment-name(var1,var2)" id="fragment-id">
3     <p th:text="'参数1:' + ${var1} + '-----参数2:' + ${var2}">...</p>
4 </div>/

```

启动 Spring Boot，使用浏览器访问 fragment.html，结果如下图

