Thymeleaf是一款用于渲染XML/XHTML/HTML5内容的模板引擎，类似JSP，Velocity，FreeMaker等，它也可以轻易的与Spring MVC等Web框架进行集成作为Web应用的模板引擎。

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。**

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、改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。

1.引入提示

在html页面中引入thymeleaf命名空间，即，此时在html模板文件中动态的属性使用th:命名空间修饰

```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
```

这样才可以在其他标签里面使用th:这样的语法.这是下面语法的前提。

2.变量表达式(获取变量值)

```
<div th:text="'你是否读过，'+${session.book}+'!!'">
   同EL表达式有些相似的效果，如果有数据，被替换
   完成前后端分离效果(美工代码)
</div>
```

代码分析:

1.可以看出获取变量值用$符号,对于javaBean的话使用变量名.属性名方式获取,这点和EL表达式一样

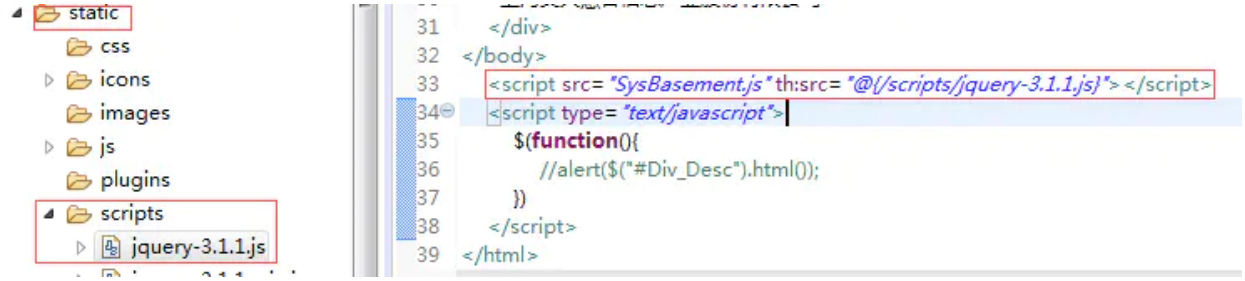
2.它通过标签中的th:text属性来填充该标签的一段内容，意思是$表达式只能写在th标签内部,不然不会生效,上面例子就是使用th:text标签的值替换div标签里面的值,至于div里面的原有的值只是为了给前端开发时做展示用的.这样的话很好的做到了前后端分离.意味着div标签中的内容会被表达式${session.book}的值所替代，无论模板中它的内容是什么，之所以在模板中“多此一举“地填充它的内容，完全是为了它能够作为原型在浏览器中直接显示出来。

3.访问spring-mvc中model的属性，语法格式为“${}”，如${user.id}可以获取model里的user对象的id属性

4.牛叉的循环<li th:each="book : ${books}" >

3.URL表达式(引入URL)

引用静态资源文件(CSS使用th:href，js使用th:src)



href链接URL(使用th:href)

```
<a th:href="@{http://blog.csdn.net/u012706811}">绝对路径</a><br />
<a th:href="@{/}">basePath</a><br />
<a th:href="@{/usethymeleaf(name=${name})}">相对路径，带一个参数</a><br />
<a th:href="@{/usethymeleaf(name=${name},alis=${name})}">相对路径，带多个参数</a><br />
<a th:href="@{/usethymeleaf?name={name}&alis={alis}(name=${name},alis=${name})}">相对路径，带多个参数</a><br />
<a th:href="@{/usethymeleaf/{name}(name=${name})}">相对路径，替换URL一个变量</a><br />
<a th:href="@{/usethymeleaf/{name}/{alis}(name=${name},alis=${name})}">相对路径，替换URL多个变量</a><br />
```

代码分析

1.最终解析的href为:

/seconddemo/ 项目路径

/seconddemo/usethymeleaf?name=Dear 相对路径，带一个参数

/seconddemo/usethymeleaf?name=Dear&alis=Dear 相对路径，带多个参数

/seconddemo/usethymeleaf?name=Dear&alis=Dear 相对路径，带多个参数

/seconddemo/usethymeleaf/Dear 相对路径，替换URL一个变量

/seconddemo/usethymeleaf/Dear/Dear 相对路径，替换URL多个变量

2.URL最后的(name=${name})表示将括号内的内容作为URL参数处理，该语法避免使用字符串拼接，大大提高了可读性

3.@{/usethymeleaf}是Context相关的相对路径，在渲染时会自动添加上当前Web应用的Context名字，假设context名字为seconddemo，那么结果应该是/seconddemo/usethymeleaf，即URL中以“/“开头的路径(比如/usethymeleaf将会加上服务器地址和域名和应用cotextpath，形成完整的URL。

4.th:href属性修饰符:它将计算并替换使用href链接URL 值，并放入的href属性中。

5.th:href中可以直接使用静态地址

4.选择或星号表达式

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

*{customer.name}

```
<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>

//等价于
<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>
```

1.如果不考虑上下文的情况下，两者没有区别；星号语法评估在选定对象上表达，而不是整个上下文，什么是选定对象？就是父标签的值。上面的*{title}表达式可以理解为${book.title}。(父对象)

2.当然，美元符号和星号语法可以混合使用

小插曲：三和四的变量表达式、URL表达式所对应的属性都可以使用统一的方式：

th.attr=“属性名=属性值”的方式来设置，参考第“七.设置属性值”部分

5.文字国际化表达式

j简单看一下就可以，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:text="#{header.address.country}">
</table>
```

6. 表达式支持的语法

（1）字面量（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

（2）文本操作（Text operations）

字符串连接（String concatenation）: +

文本替换（Literal substitutions）: |The name is ${name}|

```
<div th:class="'content'">...</div>
<span th:text="|Welcome to our application, ${user.name}!|">
//Which is equivalent to:
<span th:text="'Welcome to our application, ' + ${user.name} + '!'">
<span th:text="${onevar} + ' ' + |${twovar}, ${threevar}|">
```

（3）算术运算（Arithmetic operations）

二元运算符（Binary operators）: + , - , * , / , %

减号（Minus sign (unary operator)）: -

（4）布尔操作（Boolean operations）

Binary operators: and , or

Boolean negation (unary operator): ! , not

（5）比较和等价(Comparisons and equality)

Comparators: > , < , >= , <= (gt , lt , ge , le)

Equality operators: == , != (eq , ne)

（6）条件运算符（Conditional operators）三元运算符

If-then: (if) ? (then)

If-then-else: (if) ? (then) : (else)

Default: (value) ?: (defaultvalue)

示例一:

```
<h2 th:text="${expression} ? 'Hello' : 'Something else'"></h2>
```

示例二:

```
<!-- IF CUSTOMER IS ANONYMOUS -->
<div th:if="${customer.anonymous}">
  <div>Welcome, Gues!</div>
</div>
<!-- ELSE -->
<div th:unless="${customer.anonymous}">
  <div th:text=" 'Hi,' + ${customer.name}">Hi, User</div>
</div>
```

（7）Special tokens:

No-Operation: _

（8）switch

Switch

Thymeleaf同样支持多路选择Switch结构：

```
<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>
```

默认属性default可以用*表示：

```
<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>
```

所有这些特征可以被组合并嵌套：

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

（9）循环

渲染列表数据是一种非常常见的场景，例如现在有n条记录需要渲染成一个表格或li列表标签该数据集合必须是可以遍历的，使用th:each标签

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

代码分析：

循环，在html的标签中，加入th:each=“value:${list}”形式的属性，如可以迭代prods的数据 又如带状态变量的循环：

```
<tr  th:each="collect,iterStat : ${collects}">
    <th scope="row" th:text="${collect.id}">1</th>
    <td >
       <img th:src="${collect.webLogo}"/>
    </td>
    <td th:text="${collect.url}">Mark</td>
    <td th:text="${collect.title}">Otto</td>
    <td th:text="${collect.description}">@mdo</td>
    <td th:text="${terStat.index}">index</td>
</tr>
```

iterStat称作状态变量，属性有：

- index:当前迭代对象的index（从0开始计算）
- count: 当前迭代对象的index(从1开始计算)
- size:被迭代对象的大小
- current:当前迭代变量
- even/odd:布尔值，当前循环是否是偶数/奇数（从0开始计算）
- first:布尔值，当前循环是否是第一个
- last:布尔值，当前循环是否是最后一个

若不指定状态变量，Thymeleaf会默认生成一个名为“变量名Stat”的状态变量：

```
<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>
```

利用状态变量判断：

```
<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="${prod.inStock}? #{true} : #{false}">yes</td>
    <td>
        <span th:text="${#lists.size(prod.comments)}">2</span> comment/s
        <a href="comments.html" th:href="@{/product/comments(prodId=${prod.id})}" th:if="${not
#lists.isEmpty(prod.comments)}">view</a>
    </td>
</tr>
```

7.设置属性值

1. th:attr
   任何属性值，语法格式：th:attr="属性名=属性值,[属性名=属性值]"
   属性值如果是使用表达式的话：通常有URL表达式@{}和变量表达式${}
   但此标签语法不太优雅
   示例：

th:attr="action=@{/subscribe}" //当然也可以直接使用th:action
th:attr="src=@{/images/gtvglogo.png},title=#{logo},alt=#{logo}" //可直接使用th:src
th:attr="value=#{subscribe.submit}"//可直接使用th:value
<input type="checkbox" name="active" th:attr="checked=${user.active}"/>
设置多个属性在同一时间,有两个特殊的属性可以这样设置：
   th:alt-title 和 th:lang-xmllang
   th:src="@{/images/gtvglogo.png}" th:alt-title="#{logo}"

2.前置和后置添加属性值
   th:attrappend 和 th:attrprepend
   主要对class和style两个属性
   class="btn" th:attrappend="class=${' ' + cssStyle}"
   转换后：class="btn warning"

3.还有两个特定的添加属性
   th:classappend 和 th:styleappend
   与上面的attrappend功能一样
   class="row" th:classappend="${prodStat.odd}? 'odd'"
   转换后：奇数行class="row odd"，偶数行class="row"

8.内嵌变量Utilities

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功能类方法
sets
maps

代码示例：

```
${#dates.format(dateVar, 'dd/MMM/yyyy HH:mm')}
${#dates.arrayFormat(datesArray, 'dd/MMM/yyyy HH:mm')}
${#dates.listFormat(datesList, 'dd/MMM/yyyy HH:mm')}
${#dates.setFormat(datesSet, 'dd/MMM/yyyy HH:mm')}
${#dates.createNow()}
${#dates.createToday()}
${#strings.isEmpty(name)}
${#strings.arrayIsEmpty(nameArr)}
${#strings.listIsEmpty(nameList)}
${#strings.setIsEmpty(nameSet)}
${#strings.startsWith(name,'Don')}
// also array*, list* and set*
${#strings.endsWith(name,endingFragment)}
// also array*, list* and set*
${#strings.length(str)}
${#strings.equals(str)}
${#strings.equalsIgnoreCase(str)}
${#strings.concat(str)}
${#strings.concatReplaceNulls(str)}
${#strings.randomAlphanumeric(count)}//产生随机字符串
```

9.thymeleaf布局

定义代码片段

```
<footer th:fragment="copy">
&copy; 2016
</footer>
```

在页面任何地方引入：

```
<body>
  <div th:include="footer :: copy"></div>
  <div th:replace="footer :: copy"></div>
 </body>
```

th:include 和 th:replace区别，include只是加载，replace是替换

返回的HTML如下：

```
<body>
   <div> &copy; 2016 </div>
  <footer>&copy; 2016 </footer>
</body>
```

下面是一个常用的后台页面布局，将整个页面分为头部，尾部、菜单栏、隐藏栏，点击菜单只改变content区域的页面

```
<body class="layout-fixed">
  <div th:fragment="navbar"  class="wrapper"  role="navigation">
    <div th:replace="fragments/header :: header">Header</div>
    <div th:replace="fragments/left :: left">left</div>
    <div th:replace="fragments/sidebar :: sidebar">sidebar</div>
    <div layout:fragment="content" id="content" ></div>
    <div th:replace="fragments/footer :: footer">footer</div>
  </div>
</body>
```

任何页面想使用这样的布局值只需要替换中见的 content模块即可

```
<html xmlns:th="http://www.thymeleaf.org" layout:decorator="layout">
  <body>
     <section layout:fragment="content">
    ...
```

也可以在引用模版的时候传参

```
<head th:include="layout :: htmlhead" th:with="title='Hello'"></head>
```

layout 是文件地址，如果有文件夹可以这样写 fileName/layout:htmlhead

htmlhead 是指定义的代码片段 如 `th:fragment="copy"`

10.附录

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>`
th:utext	支持html的文本替换	`<p th:utext="${htmlcontent}">content</p>`
th:object	替换对象	`<div th:object="${session.user}">`
th:value	属性赋值	`<input th:value = "${user.name}" />`
th:with	变量赋值运算	`<div th:with="isEvens = ${prodStat.count}%2 == 0"></div>`

th:style	设置样式	`<div th:style="'display:' + @{(${sitrue} ? 'none' : 'inline-block')} + ''"></div>`
th:onclick	点击事件	`<td th:onclick = "'getCollect()'"></td>`
th:each	属性赋值	`<tr th:each = "user,userStat:${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:fragment	th:switch的一个分支	`<p th:case = "'admin'">User is an administrator</p>`
th:includ	布局标签，替换内容到引入的文件	`<head th:include="layout :: htmlhead" th:with="title='xx'"></head>`

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:selectd	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孩子。2.body:不包含标记删除,但删除其所有的孩子。3.tag:包含标记的删除,但不删除它的孩子。4.all-but-first:删除所有包含标签的孩子,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。