Thymeleaf参考指南

一、引用命名空间 <html xmlns:th="http://www.thymeleaf.org">
　　html也是标准的xml，所以命名空间跟xml中定义的一样，在正文就可以用
th:xxx来使用，当然命名空间的前缀可以换成t/p等等其他的，那么正文中后缀以此做前缀就可以了

二、表达式语法

#{...}：消息表达式，获取I18N变量信息

${...}：变量表达式，获取上下文变量信息
　　${x}将返回存储在Thymeleaf上下中的变量x或作为请求属性
　　${param.x}将返回一个名为x的请求参数（可能是多值的）
　　${session.x}将返回一个名为x的会话属性
　　${application.x}将返回一个名为x的servlet上下文属性
　　${user.name}将返回一个user变量的name属性

*{...} 选择变量表达式

@{...} 链接网址表达式

~{...} 片段表达式

文字：

- 字字：'one text', 'Another one!'
- 数字字值：0,34,3.0,12.3，...
- 布尔字：true，false
- 空字值：null
- 字Token：one，sometext，main，...

符号：

- 字符串连接：+
- 本替换：|The name is ${name}|
- 算术运算符 +，-，*，/，%
- 负号（元运算符）：-
- 布尔运算符 and、or，!，not
- 较和相等运算符：>，<，>=，<=（gt，lt，ge，le），==，!=（eq，ne）
- 条件运算符：
  f-then:\(if\) ? \(then\)
  If-then-else:\(if\) ? \(then\) : \(else\)
  Default:\(value\) ?: \(defaultvalue\)
- 特殊符号：
  哑操作符：_

所有这些功能可以组合和嵌套：
e.g　'User is of type ' + (${user.isAdmin()} ? 'Administrator': (${user.type} ?: 'Unknown'))

三、内置变量

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

e.g Established locale country: US.

四、工具表达式对象

类似于velocitty的宏

#execInfo：有关正在处理的模板的信息。
#messages：用于在变量表达式中获取外部化消息的方法，与使用#{...}语法获得的方式相同。
#uris：转义URL / URI部分的方法
#conversions：执配置的转换服务（如果有的话）的方法。
#dates：java.util.Date对象的方法：格式化，组件提取等
#calendars：类似于#dates，但对于java.util.Calendar对象。
#numbers：用于格式化数字对象的方法。
#strings：String对象的方法：contains，startsWith，prepending /appending等

#objects： 一般对象的方法。
#bools： 布尔评估的方法。
#arrays： 数组的方法。
#lists： 列表的方法。
#sets： 集合的方法。
#maps： 地图方法。
#aggregates： 在数组或集合上创建聚合的方法。
#ids： 处理可能重复的id属性的方法（例如， 作为迭代的结果）。

e.g Today is: 13 May 2011

五、选择表达式（星号语法）
变量表达式写为$ {...}， 也可以作为* {...}
区别： 星号语法计算所选对象不是整个上下的表达式。 也就是说， 只要没有选定的对象， $和*语法就会完全相同

e.g

```
<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>
```

等同于

```
<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>
```

混搭风格

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

如果没有执对象选择， 则美元和星号语法是等效的

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

六、运算符

算术运算符：

<div th:if="${prodStat.count} > 1"><span th:text="'Execution mode is ' + ((${execMode} == 'dev')? 'Development' : 'Production')">

比较表达式
<div th:if="${prodStat.count} > 1"><span th:text="'Execution mode is ' + ((${execMode} == 'dev')? 'Development' : 'Production')">
个更简单的替代案是使这些运算符的本别名： gt（>），lt（<），ge（>=），le（<=），（！）。 还有eq（==），neq /ne（！=）

条件表达式

```
<tr th:class="${row.even}? 'even' : 'odd'">
  ...
</tr>
<tr th:class="${row.even}? (${row.first}? 'first' : 'even') : 'odd'">
  ...
</tr>
<tr th:class="${row.even}? 'alt'">
  ...
</tr>
```

条件表达式（condition， then和else）的三个部分都有的表达式， 它们可以是变量（${...}， *{...}） ， 消息（#{...}），URL（@{...}）或字（'...'）

默认表达式

是种特殊类型的条件值， 没有then那个部分。 它相当指定两个表达式： 如果计算结果不为null， 则使第个表达式， 如果计算结果为null则使第个表达式

```
<div th:object="${session.user}">
  ...
  <p>Age: <span th:text="*{age}?: '(no age specified)'">27</span>.</p>
</div>
等价于
<p>Age: <span th:text="*{age != null}? *{age} : '(no age specified)'">27</span>.</p>
```

哑操作符号

由下划线符号（_）表示。

这个标记指定表达式不处理任何结果， 即相当于可执的属性（例如： 本） 不存在样。 除了其他可能性之外，这允许开发员使原型本作为默认值

e.g ...

等价于

no user authenticated

七、文本及字面量输出

```
<p th:text="#{home.welcome}">快捷通后台管理系统欢迎你!</p>
<p>Today is: <span th:text="${today}">13 february 2011</span>.</p>
```

传入变量：

```
<p th:utext="#{home.welcome(${session.user.name})}">快捷通后台管理系统欢迎你!</p>
```

#在I18N中：home.welcome=快捷通后台管理系统欢迎你!{0}

字面量：

```
<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>
<div th:if="${user.isAdmin()} == false">
<div th:if="${variable.something} == null">
<div th:class="content">...</div>  代替 <div th:class="'content'">...</div>
<span th:text="'The name of the user is ' + ${user.name}" >
```

文本替换，使用 |...| ：

```
<span th:text="|Welcome to our application, ${user.name}!|">
```

等价于

```
<span th:text="'Welcome to our application, ' + ${user.name} + '!'">
<span th:text="${onevar} + ' ' + |${twovar}, ${threevar}|">
```

只有变量/消息表达式（${...}， *{...}， #{...}）才允许包含在|...|中来实现本替换。 其他情况则不允许， 如本（'...'） ， 布尔/数字令牌，条件表达式等

格式化输出

使用双括号通过配置的数据转换服务来进数据类型转换

```
<td th:text="${{user.lastAccessDate}}">...</td>
```

非转义输出

```
<p th:utext="#{home.welcome}">快捷通后台管理系统欢迎你!</p>
```

#非转义文本输出 html标签将原样输出

使用Thymeleaf的数据属性语法输出

```
<p data-th-text="#{home.welcome}">快捷通后台管理系统欢迎你!</p>
```

#Thymeleaf的数据属性语法， 使data-属性名称和连字符（-）分隔符的数据前缀不是分号（:）：

八、属性配置

使用th:attr标签， 以设置标签的属性值

```
e.g <form action="subscribe.html" th:attr="action=@{/subscribe}">
<input type="submit" value="Subscribe!" th:attr="value=#{subscribe.submit}"/>
```

th:attr 逗号分隔的形式来设置多个属性值

通常使用通常， 使 th:*标签来设置特定的属性，来代替th:attr 标签，即
<input type="submit" value="Subscribe!" th:value="#{subscribe.submit}"/>
<form action="subscribe.html" th:action="@{/subscribe}">
还有很多类似的属性， 每个都针对特定的HTML5属性：参见附录

还有两个相当特殊的属性叫做th:alt-title和th:lang-xmllang， 可于同时将两个属性设置为相同的值
th:alt-title将设置alt和title
th:lang-xmllang将设置lang和xml:lang
前缀和后缀属性
th:attrappend和th:attrprepend属性，分别为现有属性值设置前缀和后缀
e.g <input type="button" value="Do it!" class="btn" th:attrappend="class=${' ' + cssStyle}" /> 等价于 <input type="button" value="Do it!" class="btn warning" />
特定的附加属性
th:classappend和th:styleappend属性， 于将CSS类或样式段添加到元素中， 不覆盖现有属性
e.g <tr th:each="prod : ${prods}" class="row" th:classappend="${prodStat.odd}? 'odd'">
bool属性
thymeleaf标准允许您通过计算条件表达式的结果来设置这些属性的值， 如果条件表达式结果为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
e.g <input type="checkbox" name="active" th:checked="${user.active}" />
Thymeleaf标准还持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

默认属性，即属性不存在
e.g ... 输出 ...
友好属性和标签名的持
e.g

```
<table>
  <tr data-th-each="user : ${users}">
     <td data-th-text="${user.login}">...</td>
     <td data-th-text="${user.name}">...</td>
   </tr>
</table>
```

data-{prefix}-{name}语法是在HTML5中编写定义属性的标准式， 不需要开发员使任何命名空间， 如th:*。 Thymeleaf使这种语法动提供给所有的（不仅仅是标准的法）。
还有种语法来指定定义标签： {prefix} - {name}， 它遵循W3C定义元素规范（较的W3C Web Components规范的部分）。这可以于例如第th:block元素（或第th-block）

九、链接属性

@语法： @ {...} 表示链接

有不同类型的网址：
•绝对网址： http://www.thymeleaf.org
•相对URL， 可以是：
•页面相对： user/login.html
•上下文相关： /itemdetails?id = 3（服务器中的上下文名称将自动添加）
•服务器相对： ~/billing / processInvoice（允许在同一服务器中的其他上下文（=应用程序）中调用URL。
•协议相关URL： //code.jquery.com/jquery-2.0.3.min.js

e.g
view
#输出：http://localhost:8080/gtvg/order/details?orderId=3
view
#输出：/gtvg/order/details?orderId=3
view
#输出：/gtvg/order/3/details

说明：
th:href是一个修饰符属性：一旦处理， 它将计算要使用的链接URL，并将该值设置为<a>标签的href属性。

可以使用表达式的URL参数（可以在orderId = $ {o.id}中看到）。 所需的URL参数编码操作也将自动执行。
如果需要几个参数， 这些参数将以逗号分隔： @ {/order/process（execId = ${execId}, execType ='FAST'）}
URL路径中也允许使用变量模板： @{/order/{orderId}/details（orderId = $ {orderId}）}
以/开头的相对URL（例如： /order/details）将自动以应用程序上下文名称为前缀。
如果cookie未启用或尚未知道， 则可能会在相对URL中添加";jsessionid = ..."”后缀， 以便会话被保留。 这被称为URL重写，
可以使用Servlet API中的每个URL的response.encodeURL（... ）机制来插入自己的重写过滤器。
th:href属性允许我们（可选地）
在我们的模板中有一个工作的静态href属性，这样当我们直接打开原型设计时，我们的模板链接可以被浏览器识别。

与消息语法（#{...}）的情况样， URL基数也可以是计算另个表达式的结果：
e.g <a th:href="@{${url}(orderId=${o.id})}">view
<a th:href="@{'/details/'+${user.login}(orderId=${o.id})}">view

十、循环迭代 th:each

```
<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>
```

下面这些类型的对象都是可以通过th:each进行迭代的。
•任何实现java.util.Iterable接口的对象。
•任何实现java.util.Enumeration接口的对象。
•任何实现java.util.Iterator接口的对象， 其值将被迭代器返回， 而不需要在内存中缓存所有值。
•任何实现java.util.Map的接口对象。 迭代映射时， iter变量将是java.util.Map.Entry类。
•任何数组。
•任何其将被视为包含对象本身的单值列表。

thymeleaf提供了一种用于跟踪迭代状态的机制： 状态变量。状态变量在每个th:each属性中定义， 并包含以下数据：
•当前迭代索引， 从0开始。 这是index属性。
•当前的迭代索引， 从1开始。 这是count属性。
•迭代变量中元素的总量。 这是size属性。
•每次迭代的iter变量。 这是current属性。
•当前的迭代是偶数还是奇数。 这些是even/odd布尔属性。
•当前的迭代是否是第一个迭代。 这是first布尔属性。
•当前的迭代是否是最后一个迭代。 这是last布尔属性。

十一、条件判断
e.g view
只有产品有评论信息时才会创建个评论链接

th:if 属性不仅只以布尔值作为判断条件。 它的功能有点超出这点， 它将按照这些规则判定指定的表达式值为true
•如果表达式的值不为null在代表true，为null则代表false
•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， 且不是“false”， “off”或“no”
•如果值不是布尔值， 数字， 字符或字符串。

反向属性， th:unless
e.g view
只有产品有评论信息时才会创建个评论链接

switch条件th:switch / th:case

```
<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>
```

语法解读：跟java的switch case default一样

十二、模板布局
我们经常希望从其他模板中包含这些部分，如页脚，页眉，菜单等部分为了做到这一点， 我们可以定义这些部分“片段”， 以供其他模版包含，使用th:fragment属性来定义被包含的模版片段
e.g footer.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
  <div th:fragment="copy"> © 2011 The Good Thymes Virtual Grocery
   </div>
</body>
</html>
```

上的代码定义了个名为copy的段， 我们可以使th:insert或th:replace属性（以及th:include， 尽管Thymeleaf 3.0不再推荐使它），容易地包含在我们的主中

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

请注意， th:insert期望个段表达式（~{...}）。 在上的例中， 这是个简单的段表达式，（~{，}）包围是完全可选的， 所以上的代码将等价于

```
<body>
   ...
  <div th:insert="footer :: copy"></div>
</body>
```

段表达式的语法有三种不同的格式
“~{templatename :: selector}” 包含在名称为templatename的模板上应用指定的标签选择器匹配的片段。
选择器可以只是一个片段名称， 因此可以像~{footer :: copy}中的~{templatename ::fragmentname)指定一些简单的东西。
“~{templatename}” 包含名为templatename的整个模板。
"~{:: selector}”或”~{this :: selector}“ 包含在同模板中的匹配指定选择器的段

上述示例中的模板名和选择器都可以是表达式（甚是条件表达式！ ）
e.g
段可以包含任何th:*属性。 旦将段包含在标模板（具有th:insert /th:replace属性的模板） 中， 这些属性将被计算属性表达式的值，并且它们将能够引此标模板中定义的任何上下变量
通过标签选择器，甚至可以包含不使th:fragment属性定义的段
e.g © 2011 The Good Thymes Virtual Grocery
通过ID属性来应用上面的片段

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

th:insert和th:replace(th:include)之间的区别
th:insert是最简单的： 它将简单地插指定宿主标签的标签体中。
th:replace实际上指定的段替换其宿主标签。
th:include类似于th:insert， 不是插段， 它只插此段的内容
模板段具有多类似函数的功能， th:fragment定义的段可以指定组参数
e.g

```
<div th:fragment="frag (onevar,twovar)">
    <p th:text="${onevar} + ' - ' + ${twovar}">...</p>
</div>
```

等价于下面中的一种，th:replace改成th:insert也是样的

```
<div th:replace="::frag (${value1},${value2})">...</div>
<div th:replace="::frag (onevar=${value1},twovar=${value2})">...</div>
```

段的局部变量规范 - 论是否具有参数签名 - 都不会导致上下在执之前被清空。 段仍然能够访问调模板中正在使的每个上下变量
th:assert进模版内部断
th:assert属性可以指定逗号分隔的表达式列表， 如果每个表达式的结果都为true， 则正确执， 否则引发异常
e.g <div th:assert="${onevar},(${twovar} != 43)">...
th:remove 删除片段，th:remove可以有五种不同的删除式，具体取决于它的值：

•all： 删除包含标签及其所有项。
•body： 不要删除包含的标签， 但删除所有的项。
•tag： 删除包含的标签， 但不要删除其项。
•all-but-first： 删除除第个包含标签之外的所有代，可以实现保留删除原型设计。
•none： 什么都不做。 该值对于动态计算是有的。
e.g

```
<table>
<thead>
   <tr>
      <th>NAME</th>
      <th>PRICE</th>
      <th>IN STOCK</th>
      <th>COMMENTS</th>
   </tr>
</thead>
<tbody th:remove="all-but-first">
   <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=${prod.id})}"th:unless="${#lists.isEmpty(prod
.comments)}">view</a>
      </td>
</tr>
<tr class="odd">
 <td>Blue Lettuce</td>
 <td>9.55</td>
 <td>no</td>
 <td><span>0</span> comment/s</td>
</tr>
<tr>
 <td>Mild Cinnamon</td>
 <td>1.99</td>
 <td>yes</td>
 <td><span>3</span> comment/s<a href="comments.html">view</a></td>
</tr>
</tbody>
</table>
```

删除可能是有条件的
e.g Link text not to be removed
th:remove将null视为none， 因此以下作与上述示例相同
  Link text not to be removed
如果$ {condition}为false， 则返回null， 因此不会执删除

十三、局部变量

局部变量是指定义在模版段中的变量， 并且该变量的作域为所在的模版段
Thymeleaf提供了种声明局部变量的式， 它使th:with属性， 其语法与属性值分配类似 ，还可以同时定义多个变量，还允许重在同属性中定义的变量
  e.g <div th:with="firstPer=${persons[0]},secondPer=${persons[1]}">
      <div th:with="company=${user.company + ' Co.'},account=${accounts[company]}">...

十四、属性优先级

同个标签中写多于个的th:*属性，按照属性的优先级计算后再展示

Order	Feature	Attributes

1	Fragment inclusion	th:insert th:replace
2	Fragment iteration	th:each
3	Conditional evaluation	th:if th:unless th:switch th:case
4	Local variable definition	th:object th:with
5	General attribute modification	th:attr th:attrprepend th:attrappend
6	Specific attribute modification	th:value th:href th:src ...
7	Text (tag body modification)	th:text th:utext
8	Fragment specification	th:fragment
9	Fragment removal	th:remove

十五、内联

[[...]]或[(...)]中的表达式被认为是在Thymeleaf中内联的表达式， 任何在th:text或th:utext属性中使的表达式都可以出现在[[]]或[()]中。
请注意， 虽然[[...]]等价于th:text（即结果将被HTML转义） ， [(...)]等价于th:utext， 不会执任何HTML转义。
e.g Hello, [[${session.user.name}]]!
等价于
Hello, Sebastian!
注意：内联表达式将逐字显示在HTML件中， 因此可能法将其作设计原型
内联机制可以被禁， 因为在实际应中可能会出现我们想输出[[...]]或[（...）]序列不将其内容作为表达式处理的情况。 为此， 我们将使th:inline =“none”来禁内联
e.g A double array looks like this: [[1, 2, 3], [4, 5]]!
内联JavaScript
e.g

```
<script th:inline="javascript">
    ...
    var username = [[${session.user.name}]];
    ...
</script>
```

内联CSS
e.g

```
<style th:inline="css">
    .[[${classname}]] {
        text-align: [[${align}]];
    }
</style>
```

十六、 文本模板

Thymeleaf模板中的三个模板被认为是本： TEXT， JAVASCRIPT和CSS。 这将它们与标记模板模式区分开来： HTML和XML。
本模板模式和标记模式之间的关键区别在于， 在本模板中， 没有标签以属性的形式插逻辑， 因此我们必须依赖其他机制。
最基本的机制是内联， 我们在前章已经详细介绍了这些内联机制。 内联语法是以本模板模式输出表达式结果的最简单法，
因此邮件、短信形式模板来说， 内联是最好的机制
e.g

```
Dear [(${name})],
Please find attached the results of the report you requested
with name "[(${report.name})]".
.....
```

非标记的语法

```
[# th:each="item : ${items}"]
  - [(${item})]
[/]
```

上述代码是下面的简写方式

```
[#th:block th:each="item : ${items}"]
    - [#th:block th:utext="${item}" /]
[/th:block]
```

十七、模板解析器
spring结合配置（待补充）

消息解析器
org.thymeleaf.messageresolver.StandardMessageResolver作为默认的消息解析器，是IMessageResolver接的标准实现
向模板引擎添加消息解析器： templateEngine.addMessageResolver(messageResolver);
转换服务：
Thymeleaf通过双重括号括号语法（$\{\{...\}\}）执数据转换和格式化操作的功能是由Thymeleaf标准来实现的， 该功能并不是Thymeleaf模板引擎来实现的。
因此， 配置它的式是将定义IStandardConversionService接的实现类对象直接设置为模板引擎中的StandardDialect实例。 配置法如下：

```
IStandardConversionService customConversionService = ...
StandardDialect dialect = new StandardDialect();
dialect.setConversionService(customConversionService);
templateEngine.setDialect(dialect);
IStandardConversionService customConversionService = ...
StandardDialect dialect = new StandardDialect();
dialect.setConversionService(customConversionService);
templateEngine.setDialect(dialect);
//thymeleaf-spring3和thymeleaf-spring4包含SpringStandardDialect，
并且该已经预先配置了将Spring身的转换服务基础架构集成到Thymeleaf中的IStandardConversionService实现
```

十八、模板解耦

Thymeleaf允许将模板标签与其要实现的逻辑完全分离， 从允许在HTML和XML模板模式中创建完全逻辑的标记模板。
模版解耦逻辑的主要思想是将模板逻辑在个单独的逻辑件中定义（更准确的说是逻辑资源， 因为它不定是个件）。 默认情况下，
该逻辑资源将是与模板件放在同件夹中， 并且名称相同， 但扩展名为.th.xml：

e.g 目录结构
/templates
    +->/home.html
    +->/home.th.xml

home.html

```
<!DOCTYPE html>
<html>
<body>
<table id="usersTable">
  <tr>
    <td class="username">Jeremy Grapefruit</td>
     <td class="usertype">Normal User</td>
  </tr>
  <tr>
   <td class="username">Alice Watermelon</td>
   <td class="usertype">Administrator</td>
  </tr>
</table>
</body>
</html>
```

home.th.xml

```
<?xml version="1.0"?>
<thlogic>
     <attr sel="#usersTable" th:remove="all-but-first">
     <attr sel="/tr[0]" th:each="user : ${users}">
     <attr sel="td.username" th:text="${user.name}" />
     <attr sel="td.usertype" th:text="#{|user.type.${user.type}|}" />
    </attr>
    </attr>
</thlogic>
```

个<thlogic>标签可以包含很多<attr>标签。 这些<attr>标签则通过其sel属性来选择原始模板的节点执属性注， 其中包含Thymeleaf标记选择器（实际上是AttoParser标记选择器）。
<attr>标签可以嵌套， 以便追加它们的选择器。 例如， 上代码中的sel =“/ tr [0]”将被处理为sel =“＃usersTable / tr [0]”。 username<td>的选择器将被处理为sel =“＃usersTable / tr [0] //td.username”。
配置解耦模版
默认情况下， 每个模板都没有启解耦逻辑。 因此， 如果需要启模版解耦， 则可以通过配置的模板解析器（ITemplateResolver的实现）来实现。
除了StringTemplateResolver（不允许解耦逻辑） 之外，ITemplateResolver的所有其他开箱即的实现都提供个名为useDecoupledLogic的标志，该标志将标记由该解析器解析的所有模板可能使其全部或部分逻辑保存在单独的资源件中：

```
final ServletContextTemplateResolver templateResolver =new
ServletContextTemplateResolver(servletContext);
...
templateResolver.setUseDecoupledLogic(true);
```

解耦模板逻辑不是强制要求的。 启时， 这意味着引擎将查找包含解耦逻辑的资源， 解析并将其与原始模板合并（如果存在）。
如果解耦逻辑资源不存在， 则不会抛出任何异常。
另外， 在同个模板中， 我们可以混合使逻辑耦合和逻辑解耦， 例如通过在原始模板件中添加些Thymeleaf属性，
将其他逻辑留给另外的解耦逻辑资源件。 最常的情况是使新的（在v3.0） th:ref属性。
th:ref只是个标记属性。 从解析器处理的度来看， 它什么也不做， 当模板被处理时就会消失， 但是它的处在于它作为个标记引，
即可以通过标记选择器的名称来解析， 就像标签名称或段样（TH： 段）。
e.g
<attr sel="whatever" .../>
将匹配：
  .任何<whatever>标签。
  .任何带有th:fragment =“whatever”属性的标签。
  .任何带有th:ref =“whatever”属性的标签
th:ref属性不仅适于解耦逻辑模板件： 它在其他类型的场景中也是样的， 如段表达式（∽{...}）
hymeleaf解决与每个模板相对应的解耦逻辑资源的式可由户配置。它由扩展点确定， 即
org.thymeleaf.templateparser.markup.decoupled.IDecoupledTemplateLogicResolver， 为其提供了默认实现：StandardDecoupledTemplateLogicResolver。
先， 它为模板资源的基本名称（通过其ITemplateResource＃getBaseName()法获取） 应前缀和后缀。 可以配置前缀和后缀，默认情况下，
前缀将为空， 后缀将为.th.xml。
其次， 它要求模板资源通过其ITemplateResource＃relative（StringrelativeLocation） 法来解析具有计算名称的相对资源。
使的IDecoupledTemplateLogicResolver的具体实现在TemplateEngine上进配置

```
final StandardDecoupledTemplateLogicResolver decoupledresolver =new
StandardDecoupledTemplateLogicResolver();
decoupledResolver.setPrefix("../viewlogic/");
...
templateEngine.setDecoupledTemplateLogicResolver(decoupledResolver);
```

十九、与springboot结合使用

1.jar依赖

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

2.属性配置如下

```
#thymelea模板配置
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.thymeleaf.mode=HTML5
spring.thymeleaf.encoding=UTF-8
spring.thymeleaf.content-type=text/html
spring.thymeleaf.cache=false
#开发环境不使用缓存
spring.resources.chain.strategy.content.enabled=true
spring.resources.chain.strategy.content.paths=/**
```

二十、 thymeleaf 设置不校验html标签

默认配置下，thymeleaf对.html的内容要求很严格，比如，如果少封闭符号/，就会报错而转到错误页。也比如你在使用Vue.js这样的库，然后有<div v-cloak> 这样的html代码，也会被thymeleaf认为不符合要求而抛出错误。

通过设置thymeleaf模板可以解决这个问题，下面是具体的配置:

```
spring.thymeleaf.cache=false
spring.thymeleaf.mode=LEGACYHTML5
```

LEGACYHTML5需要搭配一个额外的库NekoHTML才可用 项目中使用的构建工具是Maven添加如下的依赖即可完成:

```
<dependency>
    <groupId>net.sourceforge.nekohtml</groupId>
    <artifactId>nekohtml</artifactId>
    <version>1.9.22</version>
</dependency>
```