Thymeleaf - Cheat sheet

Version : 3.0

Declare namespace

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

Simple **Expression** Syntax

Variable Expressions
Spring Expression langage – OGNL expression

```
<p th:text="${home.welcome}">text</p>
```

Selection Variable Expressions
Selection expressions. Will be executed on a previously selected object only.

```
<div th:object="${user}">
  <p>
    Name:
    <span th:text="*{firstName}">
    </span>
  </p>
</div>
```

Message Expressions
Message (i18n) expressions. Used to retrieve locale-specific messages from external sources

```
<p th:text="#{home.welcome}">msg</p>
```

Link URL Expressions
Link (URL) expressions. Used to build URLs

```
<a href="details.html"
   th:href="@{/images/test.png}">
     link
</a>
```

Literal **Expression** Syntax

Text literals : 'one text', 'Another one!'
Number literals : 0, 34, 3.0, 12.3,…
Boolean literals : true, false
Null literal : null
Literal tokens : one, sometext, main,…

Opérations Syntax

String concatenation : +
Literal substitutions : |name : ${name}|
Arithmetic operators : +, -, *, /, %
Binary operators : and, or, !, not
Comparators : >, <, >=, <=
Equality operators : ==, !=
If-then : (if) ? (then)
If-then-else : (if) ? (then) : (else)
Default : (value) ?: (defaultvalue)

Conditional Evaluation

Simple conditionnals

- *if*

```
<span
  th:if="${condition}">view
</span>
```

- *if not*

```
<span
  th:if="${condition}">view
</span>
```

Switch statements

```
<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>
```

Iteration

Using th:each

```
<div th:each="val : ${myList}">
</div>
```

- *Iterable values*
  - Iterable, Enumeration, Map with java.util.Map.Entry
  - Any Arrays
  - Anything else will be treated like a List with 1 object.

- *Keeping iteration status*

th:each attribute status var contains the following data :

- **index** : current iteration index (from 0)
- **count** : current iteration index (from 1)
- **size** : size of the iterated list
- **current** : iter var for each iteration
- **even** / **odd** properties
- **first** : current is the first ?
- **last** : current is the last ?

```
<div th:each="val,iterStat : ${myList}">
</div>
```

Using th:switch

```
<div th:switch="${user.role}">
  <p th:case="'admin'">I am admin</p>
</div>
```

Fragments & template layout

Create fragment

```
<div th:fragment="copy">
  Random text
</div>
```

Fragment expressions
There are three different formats:

"~{templatename::selector}" : Includes the fragment resulting from applying the specified Markup Selector on the template named templatename

"~{templatename}" : Includes the complete template named templatename

"::domselector" or "this::domselector": Includes the complete template named templatename

Insert / include fragment
th:insert *is the simplest: it will simply insert the specified fragment as the body of its host tag.*

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

th:replace *actually replaces its host tag with the specified fragment.*

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

Parametrizable fragment signatures
Fragments defined with th:fragment can Specify a set of parameters :

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

This require the use of one of these two syntaxes to call the fragment from th:include or th:replace :

```
<div th:include="::frag(${val1},${val2})">
…
</div>
```

Or give the param name, so order is not Important :

```
<div th:include="::frag(twovar=${val2},
     onevar=${val1})">
…
</div>
```

Removing template fragments
*To remove some part of a template when Rendering it, use **th:remove** :*

```
<div th:remove="param">
  Some static text
</div>
```

Param must be replaced by one of :

- **all:** Remove both the containing tag and all its children.
- **body:** Do not remove the containing tag, but remove all its children.
- **tag:** Remove the containing tag, but do not remove its children.
- **all-but-first :** Remove all children of the containing tag except the first one.
- **none :** Do nothing. This value is useful for dynamic evaluation

Using **texts**

Escaped Text
Will not accept html tags, will convert each character into html special

```
<p th:text="${home.welcome}">text</p>
```

Unescaped Text
Will accept html tags

```
<p th:utext="${home.welcomeHtml}">text</p>
```

Local **Variables**

Thymeleaf offers a way to declare local variables using the th:with attribute

```
<div th:with="firstPer=${persons[0]}">
  <p>
    The name of the first person is
    <span th:text="${firstPer.name}">
      Julius Caesar
    </span>.
  </p>
</div>
```

Or with multiple local variables :

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

Thymeleaf - Cheat sheet

Version : 3.0

Thymeleaf **tags**

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:frame
th:frameborder
th:headers
th:height
th:high
th:href
th:hreflang
th:hspace
th:http-equiv
th:icon
th:id
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: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

Attribute Precedence

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

Inlining

Expression inlining

- **[[...]]** **:** considered inlined escaped text expressions in Thymeleaf, can Contains anything that can be in a **th:text**.
- **[(...)]** **:** considered inlined unescaped text expressions in Thymeleaf, can Contains anything that can be in a **th:utext**.
- **th:inline="none"** **:** Disable the inlining feature for the content.

```
<p th:inline="none">
  A double array looks like this:
  [[1, 2, 3]
</p>
```

Javascript inlining

- **th:inline="javascript"** **:** enable the Javascript inlining feature

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

Javascript inlining is intelligent, it will Convert all these types to javascript :

Strings, Numbers, Booleans
Arrays, Collections, Maps
Beans (with getter and setter)

Comments

```
<!-- Classic comment -->

<!--/* This code will be removed at
   Thymeleaf parsing time!
*/-→

<!--/*/ Special comment blocks marked to
   Be comments for html static but normal
   markup for Thymeleaf
/*/-->
```

Blocks

```
<th:block th:each="var : ${myList}">
  ...
</th:block>
```

Spring integration

Maven dependency

```
<dependency>
  <groupId>org.thymeleaf</groupId>
  <artifactId>thymeleaf-spring4</artifactId>
  <version>2.1.4.RELEASE</version>
</dependency>
```

Bean configuration *(by default template resolver looks in resources/templates)*

```
@Bean
public SpringResourceTemplateResolver templateResolver(){...}

@Bean
public SpringTemplateEngine templateEngine(){...}
```

Eclipse Extension

The Thymeleaf plugin for Eclipse IDE adds content assist features that make working in Thymeleaf templates nicer and much more Comfortable.

Thymeleaf-extras-eclipse-plugin

Credits

Sources : *github.com/grzi/cheatSheets*

All the content is taken from the thymeleaf documentation :

www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html