


# What's new in Thymeleaf 1.1

## A new expression system: the *Standard Expressions*

The old *value expression* system had a series of limitations, especially regarding to operators to be applied between expressions. In Thymeleaf 1.1 this system is substituted in both the Standard and SpringStandard dialects by a new *Standard Expression* system, which builds on the previous one and keeps all of its syntax — so that all your templates still work alright — and adds the following new features:

- Numeric literals: `1`, `32`, `42.3`, `11.34`, etc.
- Unary operators: `!` (boolean negation), `-` (numeric minus sign)
- String operators: `+` (String concatenation)
- Numeric binary operators and comparators: `+`, `-`, `*`, `/`, `%`, `>`, `<`, `>=`,`<=`
- Boolean binary operators: **`and`**, **`or`**
- Equality checks: `==`, `!=`

For example, now you can concatenate Strings easily in your expressions, like:

```
<p th:text="#{title.username} + ': ' + ${user.name}">
```

## No more Value Processors

Thymeleaf 1.0 defined three types of *processors*: *attribute processors*, *tag processors* and *value processors*. The latter type, value processors, were defined as utility objects that executed common logic needed to process expressions, usually called from attribute or tag processors.

This resulted in code similar to this in order to execute an

expression from an attribute value:

```
final StandardValueProcessor valueProcessor =
    arguments.getConfiguration().getValueProcess

final Value value =
    StandardSyntax.parseValue(attributeValue, va

final String result =
    (String) valueProcessor.getValue(arguments,
```

Thymeleaf 1.1 completely removes the concept of *value processor* from its API, so that it allows each dialect to implement expression resolution in the way preferred by its developers.

For example, the *Standard* and *SpringStandard* dialects now implement the resolution of *Standard Expressions* using the **`StandardExpressionProcessor`** class, so that all the code above can now be written in a much more concise way:

```
final String result =
    (String) StandardExpressionProcessor.process
```

## New `th:substituteby` attribute in the standard dialects

Thymeleaf 1.1 adds a new attribute to both the Standard and SpringStandard dialects called **`th:substituteby`**, which works very similarly to the already existing **`th:include`** for including a fragment from an external template, but actually substitutes the host tag by the fragment's, instead of only substituting its contents — which is what **`th:include`** does.

In order to see the differences between these tags, the following fragment:

```
<footer th:fragment="copy">
    © 2011 The Good Thymes Virtual Grocery
</footer>
```

Referenced in your template with both a **`th:include`** and a **`th:substituteby`**:

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

The results will be slightly different, because **th:include** will only copy the fragment's contents whereas **th:substituteby** will actually replace the host tag by the whole fragment:

```
<body>
    ...
    <div>
        © 2011 The Good Thymes Virtual Grocery
    </div>
    <footer>
        © 2011 The Good Thymes Virtual Grocery
    </footer>
    ...
</body>
```

## New XHTML DTDs for the standard dialects

The new **th:substituteby** attribute requires a change in the existing Thymeleaf DTDs (those specified with a **DOCTYPE SYSTEM** clause), in order to include this new attribute and allow its validation.

New versions of the DTDs for the Standard and SpringStandard dialects have been created, so that the old **DOCTYPE** declarations:

```
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/
```

Can now be substituted by new versions containing the new attribute:

```
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/
```

## New date utility functions

Thymeleaf 1.1 adds new functions to the **`#dates`** and **`#calendars`** utility function objects, which can be used in variable expressions as usual:

```
#dates.hour(now)
#dates.minute(now)
#dates.second(now)
#dates.millisecond(now)
```

*Since 1.1.2, also:*

```
#dates.createNow()
#dates.createToday()
#dates.create(year,month,day)
#dates.create(year,month,day,hour,minute)
#dates.create(year,month,day,hour,minute,second)
#dates.create(year,month,day,hour,minute,second,
```

## Changes in condition evaluation

Condition evaluation —used in **`th:if`** or **`th:unless`** attributes, for example— changes a bit in Thymeleaf 1.1 so that the Strings **`"false"`**, **`"no"`** and **`"off"`** now evaluate as **false**, instead of *true* as they did before (Thymeleaf 1.0 evaluated as *true* any non-null String).

## Performance optimization

When a template is stored into the cache, Thymeleaf 1.1 now scans its whole DOM tree in order to determine which nodes —and also entire trees— are *not executable* (because no attribute or tag processor can be applied to them), and marks those DOM nodes so that they are skipped when actually processing the template.

This improves performance by allowing the engine to focus only on those nodes that trigger real execution logic once it retrieves a template from the cache.

## Fixed HTML entity processing

Thymeleaf 1.1 fixes a behaviour in version 1.0 regarding HTML entities (**` `**, **`&pound;`**, **`&aacute;`**, etc.) that made the engine raise an exception like this:

```
The entity "nbsp" was referenced, but not declared.
```

...in the following cases:

- When **`th:utext`** was used for writing a String containing one of such entities.
- In HTML5 templates, every time an HTML entity was used.

Thymeleaf 1.1 fixes this so that not only the presence of these entities does not cause an error, but also they are preserved as they are (this is, in *escaped* form) in output.

## Modified LEGACYHTML5 mode (since 1.1.2)

For performance and reliability reasons, the **`LEGACYHTML5`** mode now uses **nekoHTML 1.9.15** or newer as an auxiliary library instead of **htmlcleaner**, which was used in previous versions.

## Dart inlining mode (since 1.1.2)

Thymeleaf 1.1.2 adds a new "**`dart`**" inlining mode to its Standard Dialects (besides **`text`** and **`javascript`**) which allows you to inline your data in your Dart scripts in the same way as you can currently do with JavaScript..

## Minor modifications and bug fixing

Of course, many minor modifications and bug fixes have been applied too. In order to know more, have a look at the changelogs.

---