

*教程：使用Thymeleaf*

**文件版本：** 20171104 - 2017年11月4日

**工程版本：** 2.1.6.RELEASE

**项目网站：** [http](http://www.thymeleaf.org/)**：**[//www.thymeleaf.org](http://www.thymeleaf.org/)

***1介绍Thymeleaf***

*1.1什么是百里香？*

Thymeleaf是一个Java库。它是一个XML / XHTML / HTML5模板引擎，能够将一组转换应用于模板文件，以便显示由应用程序生成的数据和/或文本。

它更适合在Web应用程序中为XHTML / HTML5提供服务，但它可以处理任何XML文件，无论是Web还是独立应用程序。

Thymeleaf的主要目标是提供创建模板的优雅和格式良好的方式。为了达到这个目的，它基于XML标签和属性来定义*DOM（文档对象模型）*上的预定义逻辑的执行，而不是将该逻辑作为代码明确地写入模板内。

其架构允许快速处理模板，依靠对分析文件的智能缓存，以便在执行过程中尽可能少地使用I / O操作。

最后但并非最不重要的是，Thymeleaf从一开始就设计了XML和Web标准，允许您创建完全验证的模板，如果这是您的需要。

*1.2 Thymeleaf可以处理什么样的模板？*

开箱即用，Thymeleaf允许您处理六种模板，每种模板都称为模板模式：

* XML
* 有效的XML
* XHTML
* 有效的XHTML
* HTML5
* 旧版HTML5

除了*Legacy HTML5*模式外，所有这些模式都是指格式良好的XML文件，它允许您使用独立（非关闭）标签，标签属性（不带值）或不在引号之间的特征处理HTML5文件。为了在这个特定的模式下处理文件，Thymeleaf将首先执行一个转换，将你的文件转换成格式完好的XML文件，这些文件仍然是完全有效的HTML5（实际上是推荐的创建HTML5代码的方式）[1](http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#fn1)。

另请注意，验证仅适用于XML和XHTML模板。

尽管如此，这些并不是Thymeleaf可以处理的唯一的模板类型，用户总是可以通过指定在这个模式下*解析*模板的方法和*写出*结果的方式来定义他/她自己的模式。这样，任何可以被建模为DOM树的东西（无论是否是XML）都可以被Thymeleaf有效地处理为模板。

*1.3方言：标准方言*

Thymeleaf是一个非常可扩展的模板引擎（事实上它应该被称为*模板引擎框架*），它允许您完全定义将在模板中处理的DOM节点以及它们将如何处理。

一个将一些逻辑应用到DOM节点的对象被称为*处理器*，一组这样的处理器（加上一些额外的工件）被称为方言，其中Thymeleaf的核心库提供了一个*标准方言*的开箱即用，这应该足以满足大部分用户的需求。

*标准方言是本教程涵盖的方言*。您将在后面的页面中了解到的每个属性和语法功能均由该方言定义，即使未明确提及。

当然，如果用户想要定义自己的处理逻辑，并利用该库的高级功能，就可以创建自己的方言（甚至是扩展标准方言）。模板引擎可以一次配置多个方言。

官方的thymeleaf-spring3和thymeleaf-spring4集成包都定义了一种称为“SpringStandard方言”的方言，大部分与标准方言相同，但用小的适应性来更好地使用Spring框架中的某些功能（例如，通过使用Spring表达式语言而不是Thymeleaf的标准OGNL）。所以，如果你是一个Spring MVC用户，你不会浪费你的时间，因为你在这里学到的几乎所有的东西都可以在你的Spring应用程序中使用。

Thymeleaf标准方言可以以任何模式处理模板，但特别适用于面向Web的模板模式（XHTML和HTML5）。除HTML5之外，它特别支持并验证了以下XHTML规范：*XHTML 1.0 Transitional*，*XHTML 1.0 Strict*，*XHTML 1.0 Frameset*和*XHTML 1.1*。

标准方言的大多数处理器是*属性处理器*。这允许浏览器在被处理之前正确地显示XHTML / HTML5模板文件，因为它们将简单地忽略附加属性。例如，虽然使用标记库的JSP可能包含不能直接由浏览器显示的代码片段，例如：

<form:inputText name="userName" value="${user.name}" />

... Thymeleaf标准方言将允许我们实现相同的功能：

<input type="text" name="userName" value="James Carrot" th:value="${user.name}" />

这不仅可以被浏览器正确显示，而且还允许我们（可选地）在其中指定一个值属性（在这种情况下为“James Carrot”），当在浏览器中静态打开原型时，将显示该属性将被替换**${user.name}**为模板的Thymeleaf处理过程中的评估结果。

如果需要的话，这将允许您的设计人员和开发人员使用完全相同的模板文件，并减少将静态原型转换为工作模板文件所需的工作量。这样做的能力通常被称为*自然模板*（*Nature Templating）*。

*1.4总体架构*

Thymeleaf的核心是一个DOM处理引擎。具体来说，它使用自己的高性能DOM实现（不是标准的DOM API）来构建模板的内存树表示，后者通过遍历其节点并在其上执行处理器来修改DOM当前*配置*以及传递给模板以进行表示的数据集（称为上下文）。

DOM模板表示的使用使得它非常适合于Web应用程序，因为Web文档经常被表示为对象树（事实上，DOM树是浏览器在内存中表示网页的方式）。此外，基于大多数Web应用程序仅使用几十个模板的想法，这些模板不是大文件，并且在应用程序运行时通常不会更改，Thymeleaf使用已分析的模板DOM树的内存缓存允许它在生产环境中快速运行，因为对于大多数模板处理操作来说，只需要很少的I / O（如果有的话）。

如果您想了解更多细节，请参阅本教程后面的整章内容，以及Thymeleaf优化内存和资源使用情况以加快运行速度。

尽管如此，还是有一个限制：与其他模板解析/处理方法相比，此架构还需要为每个模板执行大量的内存空间，这意味着您不应该使用该库来创建大数据XML文档（而不是网页文件）。作为一般的经验法则（并且总是取决于JVM的内存大小），如果要在单个模板执行中生成大小为几十兆字节的XML文件，则可能不应使用Thymeleaf。

我们认为这个限制的原因只适用于数据XML文件，而不适用于Web XHTML / HTML5，因为您永远不应该生成如此之大的Web文档以至于用户的浏览器设置为闪光和/或爆炸 - 请记住，这些浏览器还必须创建DOM树为您的网页！

*1.5在继续之前，您应该阅读...*

Thymeleaf特别适合在Web应用程序中工作。网络应用程序是基于一系列的标准，每个人都应该知道的很少，但很少有 - 即使他们已经与他们多年的工作。

随着HTML5的出现，当今Web标准的最新技术比以往更加混乱...... *我们*是否*从XHTML转向HTML？我们会放弃XML语法吗？为什么没有人在讨论XHTML 2.0呢？*

因此，在继续学习本教程之前，强烈建议您阅读Thymeleaf网站上的一篇名为*“从HTML到HTML（通过HTML）”的文章*，您可以在以下地址找到这篇文章：[http](http://www.thymeleaf.org/doc/articles/fromhtmltohtmlviahtml.html)：[//www.thymeleaf.org /doc/articles/fromhtmltohtmlviahtml.html](http://www.thymeleaf.org/doc/articles/fromhtmltohtmlviahtml.html)

***2好的虚拟杂货店***

本指南的本章和未来章节中所示示例的源代码可以在[Good Thymes Virtual Grocery GitHub存储库中找到](https://github.com/thymeleaf/thymeleafexamples-gtvg)。

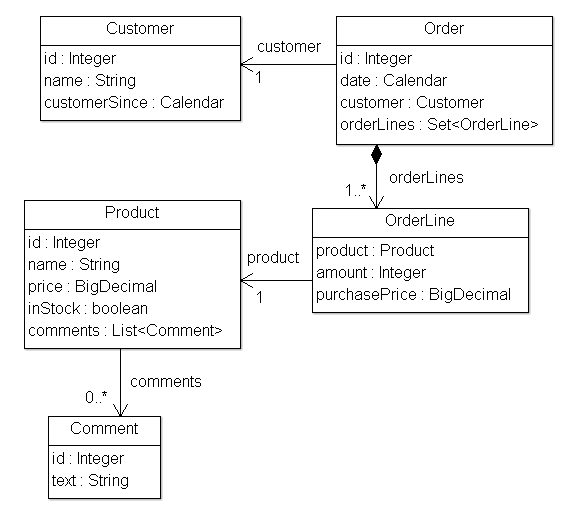
*2.1一个杂货店的网站*

为了更好地解释与Thymeleaf处理模板有关的概念，本教程将使用一个演示应用程序，您可以从项目网站下载。

这个应用程序代表了一个虚拟的虚拟杂货店的网站，并将为我们提供足够的场景来体现不同的百里香特色。

我们需要为我们的应用程序提供一套相当简单的模型实体：通过创建**Products**销售。我们也将管理这些：**CustomersOrdersCommentsProducts**

示例应用程序模型



我们的小应用程序也将有一个非常简单的服务层，由**Service**包含以下方法的对象组成：

public class ProductService {

...

public List<Product> findAll() {

return ProductRepository.getInstance().findAll();

}

public Product findById(Integer id) {

return ProductRepository.getInstance().findById(id);

}

}

最后，在web层，我们的应用程序将有一个过滤器，根据请求URL将执行委托给启用了Thymeleaf的命令：

private boolean process(HttpServletRequest request, HttpServletResponse response)

throws ServletException {

try {

/\*

\* Query controller/URL mapping and obtain the controller

\* that will process the request. If no controller is available,

\* return false and let other filters/servlets process the request.

\*/

IGTVGController controller = GTVGApplication.resolveControllerForRequest(request);

if (controller == null) {

return false;

}

/\*

\* Obtain the TemplateEngine instance.

\*/

TemplateEngine templateEngine = GTVGApplication.getTemplateEngine();

/\*

\* Write the response headers

\*/

response.setContentType("text/html;charset=UTF-8");

response.setHeader("Pragma", "no-cache");

response.setHeader("Cache-Control", "no-cache");

response.setDateHeader("Expires", 0);

/\*

\* Execute the controller and process view template,

\* writing the results to the response writer.

\*/

controller.process(

request, response, this.servletContext, templateEngine);

return true;

} catch (Exception e) {

throw new ServletException(e);

}

}

这是我们的**IGTVGController**界面：

public interface IGTVGController {

public void process(

HttpServletRequest request, HttpServletResponse response,

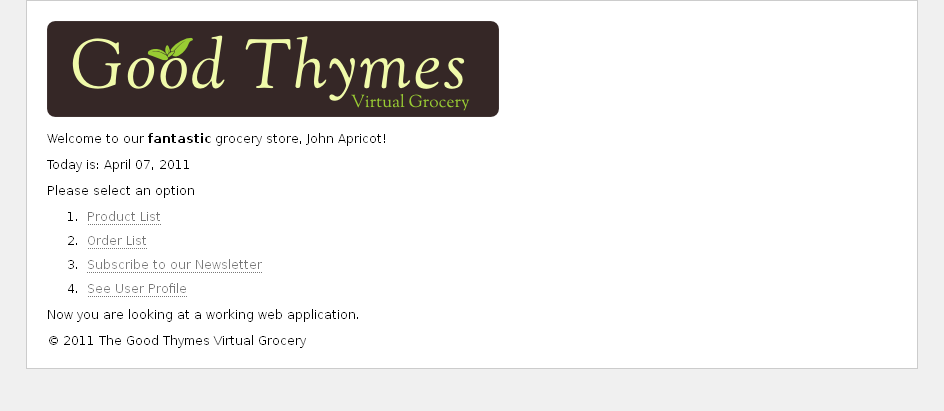
ServletContext servletContext, TemplateEngine templateEngine);

}

我们现在要做的就是创建**IGTVGController**接口的实现，从服务中检索数据并使用**TemplateEngine**对象处理模板。

最后，看起来像这样：

示例应用程序主页



但首先让我们看看这个模板引擎是如何初始化的。

*2.2创建和配置模板引擎*

我们的过滤器中的*过程（...）*方法包含这个句子：

TemplateEngine templateEngine = GTVGApplication.getTemplateEngine();

这意味着*GTVGApplication*类负责创建和配置启用Thymeleaf的应用程序中最重要的对象之一：**TemplateEngine**实例。

我们的**org.thymeleaf.TemplateEngine**对象是这样初始化的：

public class GTVGApplication {

...

private static TemplateEngine templateEngine;

...

static {

...

initializeTemplateEngine();

...

}

private static void initializeTemplateEngine() {

ServletContextTemplateResolver templateResolver =

new ServletContextTemplateResolver();

// XHTML is the default mode, but we set it anyway for better understanding of code

templateResolver.setTemplateMode("XHTML");

// This will convert "home" to "/WEB-INF/templates/home.html"

templateResolver.setPrefix("/WEB-INF/templates/");

templateResolver.setSuffix(".html");

// Template cache TTL=1h. If not set, entries would be cached until expelled by LRU

templateResolver.setCacheTTLMs(3600000L);

templateEngine = new TemplateEngine();

templateEngine.setTemplateResolver(templateResolver);

}

...

}

当然，配置**TemplateEngine**对象的方法有很多种，但是现在这几行代码就足够了解所需的步骤了。

***模板解析器***

让我们从模板解析器开始：

ServletContextTemplateResolver templateResolver = new ServletContextTemplateResolver();

模板解析器是实现来自Thymeleaf API的接口的对象，称为**org.thymeleaf.templateresolver.ITemplateResolver**：

public interface ITemplateResolver {

...

/\*

\* Templates are resolved by String name (templateProcessingParameters.getTemplateName())

\* Will return null if template cannot be handled by this template resolver.

\*/

public TemplateResolution resolveTemplate(

TemplateProcessingParameters templateProcessingParameters);

}

这些对象负责确定我们的模板将如何被访问，在这个GTVG应用程序中，**org.thymeleaf.templateresolver.ServletContextTemplateResolver**我们使用的实现指定我们要从*Servlet上下文中*检索我们的模板文件作为资源：存在的应用程序范围的**javax.servlet.ServletContext**对象在每个Java Web应用程序中，并且将Web应用程序根视为资源路径的根来解析资源。

但是关于模板解析器，我们不能这么说，因为我们可以在其上设置一些配置参数。首先，模板模式是标准模板之一：

templateResolver.setTemplateMode("XHTML");

XHTML是默认的模板模式**ServletContextTemplateResolver**，但无论如何建立XHTML是一个很好的做法，这样我们的代码就可以清楚地记录下正在发生的事情。

templateResolver.setPrefix("/WEB-INF/templates/");

templateResolver.setSuffix(".html");

这些*前缀*和*后缀*的确如此：修改我们将传递给引擎的模板名称，以获取要使用的实际资源名称。

使用此配置，模板名称*“产品/列表”*将对应于：

servletContext.getResourceAsStream("/WEB-INF/templates/product/list.html")

可选地，经过解析的模板生活在缓存中的时间量将被视为有效的，可以在模板解析器中通过*cacheTTLMs*属性进行配置：

templateResolver.setCacheTTLMs(3600000L);

当然，如果达到最大高速缓存大小并且它是当前高速缓存的最旧的条目，则可以在达到TTL之前将模板从高速缓存中排除。

缓存行为和大小可以由用户通过实现**ICacheManager**接口来定义，或者仅仅修改**StandardCacheManager**对象集来默认管理缓存。

稍后我们将学习更多关于模板解析器的知识。现在让我们来看看我们的模板引擎对象的创建。

***模板引擎***

模板引擎对象是类*org.thymeleaf.TemplateEngine*，这些是在当前示例中创建引擎的行：

templateEngine = new TemplateEngine();

templateEngine.setTemplateResolver(templateResolver);

很简单，不是吗？我们所需要的只是创建一个实例并将模板解析器设置为它。

模板解析器是唯一需要的参数**TemplateEngine**，当然还有很多其他的东西（消息解析器，缓存大小等等）。现在，这就是我们所需要的。

我们的模板引擎已经准备就绪，我们可以使用Thymeleaf开始创建我们的页面。

***3使用文本***

*3.1多语言的欢迎*

我们的首要任务是为我们的杂货店网站创建一个主页。

我们将写这个页面的第一个版本将是非常简单的：只是一个标题和一个欢迎消息。这是我们的**/WEB-INF/templates/home.html**文件：

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

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

<head>

<title>Good Thymes Virtual Grocery</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<link rel="stylesheet" type="text/css" media="all"

href="../../css/gtvg.css" th:href="@{/css/gtvg.css}" />

</head>

<body>

<p th:text="#{home.welcome}">Welcome to our grocery store!</p>

</body>

</html>

首先你会注意到这个文件是XHTML，它可以被任何浏览器正确显示，因为它不包含任何非XHTML标签（浏览器忽略所有不理解的属性**th:text**）。此外，浏览器将以标准模式（不是以怪癖模式）显示它，因为它具有格式良好的**DOCTYPE**声明。

接下来，这也是*有效的* XHTML [2](http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#fn2)，因为我们已经指定了一个Thymeleaf DTD，它定义了一些属性，**th:text**以便您的模板可以被认为是有效的。甚至更多：一旦模板被处理（并且所有的**th:\***属性被删除），Thymeleaf将自动用**DOCTYPE**条款中的DTD声明替换标准DTD声明**XHTML 1.0 Strict**（我们将在后面的章节中保留这个DTD转换功能）。

一个thymeleaf命名空间也被声明为**th:\***属性：

<html xmlns="http://www.w3.org/1999/xhtml"

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

请注意，如果我们根本不关心模板的有效性或完整性，那么我们可以简单地指定一个标准**XHTML 1.0 Strict DOCTYPE**，并且不包含xmlns名称空间声明：

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html>

<head>

<title>Good Thymes Virtual Grocery</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<link rel="stylesheet" type="text/css" media="all"

href="../../css/gtvg.css" th:href="@{/css/gtvg.css}" />

</head>

<body>

<p th:text="#{home.welcome}">Welcome to our grocery store!</p>

</body>

</html>

...在XHTML模式下，Thymeleaf仍然可以很好地处理这个问题（尽管我们的IDE可能会使我们的生活变得相当悲惨，从而在各处发出警告）。

但足以验证。现在为模板的真正有趣的部分：让我们来看看什么**th:text**属性是什么。

***使用：文本和外部化文本***

对文本进行外部化是将模板代码片段从模板文件中提取出来，以便将它们保存在特定的单独文件（通常是**.properties**文件）中，并且可以用其他语言（称为国际化或简称*i18n*）编写的等效文本轻松替换它们。外部化的文本片段通常被称为“消息”。

消息总是有一个标识它们的键，而Thymeleaf允许你指定一个文本应该对应于一个特定的消息，**#{...}**语法如下：

<p th:text="#{home.welcome}">Welcome to our grocery store!</p>

我们在这里可以看到的实际上是“百里香标准方言”的两个不同特征：

* 该**th:text**属性评估其值表达式，并将此评估的结果设置为所在标签的主体，从而有效地替代代码中所见的“欢迎来到我们的杂货店！”文本。
* 的**#{home.welcome}**表达，在指定的*标准表达式语法*，指定要由所使用的文本**th:text**属性应与该消息**home.welcome**对应于哪个语言环境，我们正在处理与模板键。

现在，这个外部化文本在哪里？

Thymeleaf中的外化文本的位置是完全可配置的，并且将取决于**org.thymeleaf.messageresolver.IMessageResolver**所使用的特定实现。通常情况下，**.properties**将使用基于文件的实现，但是如果需要，我们可以创建自己的实现，例如从数据库中获取消息。

但是，我们在初始化过程中没有给我们的模板引擎指定一个消息解析器，这意味着我们的应用程序正在使用由类实现的*标准消息解析器***org.thymeleaf.messageresolver.StandardMessageResolver**。

这个标准的消息解析器希望**/WEB-INF/templates/home.html**在.properties文件中查找相同文件夹中的消息，并使用与模板相同的名称，如：

* **/WEB-INF/templates/home\_en.properties** 为英文文本。
* **/WEB-INF/templates/home\_es.properties** 为西班牙语文本。
* **/WEB-INF/templates/home\_pt\_BR.properties** 为葡萄牙语（巴西）的语言文字。
* **/WEB-INF/templates/home.properties** 默认文本（如果语言环境不匹配）。

让我们来看看我们的**home\_es.properties**文件：

home.welcome=¡Bienvenido a nuestra tienda de comestibles!

这就是我们需要使Thymeleaf流程成为我们的模板。那么我们来创建我们的家庭控制器。

***上下文***

为了处理我们的模板，我们将创建一个**HomeController**实现**IGTVGController**我们之前看到的接口的类：

public class HomeController implements IGTVGController {

public void process(

HttpServletRequest request, HttpServletResponse response,

ServletContext servletContext, TemplateEngine templateEngine) {

WebContext ctx =

new WebContext(request, response, servletContext, request.getLocale());

templateEngine.process("home", ctx, response.getWriter());

}

}

我们在这里可以看到的第一件事是创建一个上下文。一个Thymeleaf上下文是一个实现**org.thymeleaf.context.IContext**接口的对象。上下文应包含在变量映射中执行模板引擎所需的所有数据，并且还引用必须用于外部化消息的区域设置。

public interface IContext {

public VariablesMap<String,Object> getVariables();

public Locale getLocale();

...

}

这个接口有一个特殊的扩展**org.thymeleaf.context.IWebContext**：

public interface IWebContext extends IContext {

public HttpSerlvetRequest getHttpServletRequest();

public HttpSession getHttpSession();

public ServletContext getServletContext();

public VariablesMap<String,String[]> getRequestParameters();

public VariablesMap<String,Object> getRequestAttributes();

public VariablesMap<String,Object> getSessionAttributes();

public VariablesMap<String,Object> getApplicationAttributes();

}

Thymeleaf核心库提供了以下每个接口的实现：

* **org.thymeleaf.context.Context** 器物 **IContext**
* **org.thymeleaf.context.WebContext** 器物 **IWebContext**

正如你可以在控制器代码中看到的那样，**WebContext**我们将使用它。实际上我们不得不这样做，因为使用a **ServletContextTemplateResolver**需要我们使用上下文实现**IWebContext**。

WebContext ctx = new WebContext(request, servletContext, request.getLocale());

这三个构造函数参数中只有两个是必需的，因为如果没有指定系统的缺省语言环境（尽管在实际应用中绝对不要这样做）。

从接口定义中我们可以看出，**WebContext**将提供获取请求参数和请求，会话和应用程序属性的专门方法。但实际上**WebContext**会做的不仅仅是这一点：

* 将所有请求属性添加到上下文变量映射。
* 添加一个名为**param**包含所有请求参数的上下文变量。
* 添加一个名为**session**包含所有会话属性的上下文变量。
* 添加一个名为**application**包含所有ServletContext属性的上下文变量。

就在执行之前，一个特殊的变量被设置到所有的上下文对象（的实现**IContext**）中，包括两个，**Context**并**WebContext**称为执行信息（**execInfo**）。这个变量包含了两个可以在你的模板中使用的数据：

* 模板名称（**${execInfo.templateName}**），为引擎执行指定的名称，以及与正在执行的模板相对应的名称。
* 当前日期和时间（**${execInfo.now}**），**Calendar**对应于模板引擎开始执行此模板的对象。

***执行模板引擎***

使用我们的上下文对象，我们只需要执行指定模板名称和上下文的模板引擎，然后传递响应写入器，以便将响应写入其中：

templateEngine.process("home", ctx, response.getWriter());

让我们看看这个使用西班牙语区域的结果：

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Good Thymes Virtual Grocery</title>

<meta content="text/html; charset=UTF-8" http-equiv="Content-Type"/>

<link rel="stylesheet" type="text/css" media="all" href="/gtvg/css/gtvg.css" />

</head>

<body>

<p>¡Bienvenido a nuestra tienda de comestibles!</p>

</body>

</html>

*3.2更多关于文本和变量*

***未转义的文本***

我们的主页最简单的版本似乎已经准备好了，但是有一些我们还没有想过...如果我们有这样的信息呢？

home.welcome=Welcome to our <b>fantastic</b> grocery store!

如果我们像以前一样执行这个模板，我们将获得：

<p>Welcome to our &lt;b&gt;fantastic&lt;/b&gt; grocery store!</p>

这不完全是我们所期望的，因为我们的**<b>**标签已经被转义，因此它将被显示在浏览器中。

这是th：text属性的默认行为。如果我们想要Thymeleaf尊重我们的XHTML标签而不是逃避它们，我们将不得不使用不同的属性:( **th:utext**对于“未转义的文本”）：

<p th:utext="#{home.welcome}">Welcome to our grocery store!</p>

This will output our message just like we wanted it:

<p>Welcome to our <b>fantastic</b> grocery store!</p>

***使用和显示变量***

现在让我们添加更多的内容到我们的主页。例如，我们可能希望在我们的欢迎消息下面显示日期，如下所示：

Welcome to our fantastic grocery store!

Today is: 12 july 2010

首先，我们将不得不修改我们的控制器，以便将该日期添加为上下文变量：

public void process(

HttpServletRequest request, HttpServletResponse response,

ServletContext servletContext, TemplateEngine templateEngine) {

SimpleDateFormat dateFormat = new SimpleDateFormat("dd MMMM yyyy");

Calendar cal = Calendar.getInstance();

WebContext ctx =

new WebContext(request, response, servletContext, request.getLocale());

ctx.setVariable("today", dateFormat.format(cal.getTime()));

templateEngine.process("home", ctx, response.getWriter());

}

我们已经**String**在我们的上下文中添加了今天的变量，现在我们可以在我们的模板中显示它：

<body>

<p th:utext="#{home.welcome}">Welcome to our grocery store!</p>

<p>Today is: <span th:text="${today}">13 February 2011</span></p>

</body>

正如你所看到的，我们仍然在使用这个**th:text**作业的属性（这是正确的，因为我们想要替换标签的主体），但是这次的语法有些不同，而不是一个**#{...}**表达式值，我们使用**${...}**一。这是一个变量表达式值，它包含一个名为*OGNL（对象图导航语言）*的语言表达式，将在上下文变量映射上执行。

该**${today}**表达式只是表示“今天拿到称为变量”，但这些表述可能更加复杂（如**${user.name}**“获取被叫用户的变量，并调用它的**getName()**方法”）。

在属性值中有相当多的可能性：消息，变量表达式等等。下一章将向我们展示所有这些可能性。

***4标准表达式语法***

我们将在我们的杂货店虚拟商店的发展中稍作休息，以了解Thymeleaf Standard Dialect：Thymeleaf标准表达式语法中最重要的部分之一。

我们已经看到在这个语法中表达了两种有效的属性值：消息和变量表达式：

<p th:utext="#{home.welcome}">Welcome to our grocery store!</p>

<p>Today is: <span th:text="${today}">13 february 2011</span></p>

但还有更多我们还不知道的价值类型，以及我们已经知道的更多有趣的细节。首先，让我们看一下标准表达式功能的快速总结：

* 简单表达式：
  + 变量表达式： **${...}**
  + 选择变量表达式： **\*{...}**
  + 消息表达式： **#{...}**
  + 链接URL表达式： **@{...}**
* 字面
  + 文本文字：**'one text'**，**'Another one!'**，...
  + 号码文字：**0**，**34**，**3.0**，**12.3**，...
  + 布尔文字：**true**，**false**
  + 空文字： **null**
  + 文字标记：**one**，**sometext**，**main**，...
* 文字操作：
  + 字符串连接： **+**
  + 文字替换： **|The name is ${name}|**
* 算术运算：
  + 二元运算符：**+**，**-**，**\***，**/**，**%**
  + 减号（一元运算符）： **-**
* 布尔运算：
  + 二元运算符：**and**，**or**
  + 布尔否定（一元运算符）： **!**，**not**
* 比较和平等：
  + 比较：**>**，**<**，**>=**，**<=**（**gt**，**lt**，**ge**，**le**）
  + 平等运营商：**==**，**!=**（**eq**，**ne**）
* 有条件的操作符：
  + IF-THEN： **(if) ? (then)**
  + IF-THEN-ELSE： **(if) ? (then) : (else)**
  + 默认： **(value) ?: (defaultvalue)**

所有这些功能可以组合和嵌套：

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

*4.1消息*

我们已经知道，**#{...}**消息表达式允许我们链接这个：

<p th:utext="#{home.welcome}">Welcome to our grocery store!</p>

对此：

home.welcome=¡Bienvenido a nuestra tienda de comestibles!

但是还有一个方面我们还没有想到：如果消息文本不完全是静态的，会发生什么？例如，如果我们的应用程序知道谁是用户访问该网站的时间，我们想用名字迎接他/她？

<p>¡Bienvenido a nuestra tienda de comestibles, John Apricot!</p>

这意味着我们需要添加一个参数给我们的消息。像这样：

home.welcome=¡Bienvenido a nuestra tienda de comestibles, {0}!

参数根据**java.text.MessageFormat**标准语法指定，这意味着您可以将格式添加到该文档的API文档中指定的数字和日期。

为了给我们的参数指定一个值，并给定一个HTTP会话属性**user**，我们将会有：

<p th:utext="#{home.welcome(${session.user.name})}">

Welcome to our grocery store, Sebastian Pepper!

</p>

如果需要，可以指定几个参数，用逗号分隔。事实上，消息密钥本身可能来自一个变量：

<p th:utext="#{${welcomeMsgKey}(${session.user.name})}">

Welcome to our grocery store, Sebastian Pepper!

</p>

*4.2变量*

我们已经提到，**${...}**表达式实际上是在包含在上下文中的变量映射上执行的OGNL（对象图导航语言）表达式。

有关OGNL语法和功能的详细信息，请阅读OGNL语言指南：[http](http://commons.apache.org/ognl/) : [//commons.apache.org/ognl/](http://commons.apache.org/ognl/)

从OGNL的语法来看，我们知道这一点：

<p>Today is: <span th:text="${today}">13 february 2011</span>.</p>

实际上相当于这个：

ctx.getVariables().get("today");

但是OGNL允许我们创建更强大的表达式，这就是：

<p th:utext="#{home.welcome(${session.user.name})}">

Welcome to our grocery store, Sebastian Pepper!

</p>

...实际上通过执行以下操作获取用户名：

((User) ctx.getVariables().get("session").get("user")).getName();

但是getter方法导航只是OGNL的一个特性。让我们来看看更多：

/\*

\* Access to properties using the point (.). Equivalent to calling property getters.

\*/

${person.father.name}

/\*

\* Access to properties can also be made by using brackets ([]) and writing

\* the name of the property as a variable or between single quotes.

\*/

${person['father']['name']}

/\*

\* If the object is a map, both dot and bracket syntax will be equivalent to

\* executing a call on its get(...) method.

\*/

${countriesByCode.ES}

${personsByName['Stephen Zucchini'].age}

/\*

\* Indexed access to arrays or collections is also performed with brackets,

\* writing the index without quotes.

\*/

${personsArray[0].name}

/\*

\* Methods can be called, even with arguments.

\*/

${person.createCompleteName()}

${person.createCompleteNameWithSeparator('-')}

***表达式基本对象***

在上下文变量上评估OGNL表达式时，一些对象可用于表达式以获得更高的灵活性。这些对象将被引用（按照OGNL标准），从**#**符号开始：

* **#ctx**：上下文对象。
* **#vars:** 上下文变量。
* **#locale**：上下文语言环境。
* **#httpServletRequest**:(只在Web上下文中）**HttpServletRequest**对象。
* **#httpSession**:(只在Web上下文中）**HttpSession**对象。

所以我们可以做到这一点：

Established locale country: <span th:text="${#locale.country}">US</span>.

您可以在[附录A中](http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#appendix-a-expression-basic-objects)阅读这些对象的完整参考。

***表达式工具对象***

除了这些基本的对象之外，Thymeleaf还会为我们提供一套实用对象，帮助我们在表达式中执行常见任务。

* **#dates**：**java.util.Date**对象的实用方法：格式化，组件提取等
* **#calendars**：类似于**#dates**，但是对于**java.util.Calendar**物体。
* **#numbers**：格式化数字对象的实用方法。
* **#strings**：**String**对象的实用方法：contains，startsWith，prepending / appending等
* **#objects**：一般对象的实用方法。
* **#bools**：布尔评估的实用方法。
* **#arrays**：数组的实用方法。
* **#lists**：列表的实用方法。
* **#sets**：集合的实用方法。
* **#maps**：地图的实用方法。
* **#aggregates**：用于在数组或集合上创建聚合的实用方法。
* **#messages**：用于在变量表达式中获得外部消息的实用方法，与使用＃{...}语法获得的方式相同。
* **#ids**：用于处理可能重复的id属性的实用方法（例如，作为迭代的结果）。

您可以查看[附录B中](http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#appendix-b-expression-utility-objects)每个实用程序对象提供的功能。

***在我们的主页重新格式化日期***

现在我们知道这些实用程序对象，我们可以用它们来改变我们在主页中显示日期的方式。而不是在我们这样做**HomeController**：

SimpleDateFormat dateFormat = new SimpleDateFormat("dd MMMM yyyy");

Calendar cal = Calendar.getInstance();

WebContext ctx = new WebContext(request, servletContext, request.getLocale());

ctx.setVariable("today", dateFormat.format(cal.getTime()));

templateEngine.process("home", ctx, response.getWriter());

...我们可以做到这一点：

WebContext ctx = new WebContext(request, servletContext, request.getLocale());

ctx.setVariable("today", Calendar.getInstance());

templateEngine.process("home", ctx, response.getWriter());

...然后在视图图层本身中执行日期格式：

<p>

Today is: <span th:text="${#calendars.format(today,'dd MMMM yyyy')}">13 May 2011</span>

</p>

*4.3**选择表达式（星号语法）*

变量表达式不仅可以写成**${...}**表达式，还可以写入表达式中**\*{...}**。

不过，有一个重要区别：星号语法评估所选对象上的表达式，而不是整个上下文变量映射。这是：只要没有选定的对象，美元和星号语法就完全一样。

那对象选择是什么东西？一个**th:object**属性。让我们在我们的用户配置文件（**userprofile.html**）页面中使用它：

<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>

这完全等同于：

<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>

当然，美元和星号的语法可以混合使用：

<div th:object="${session.user}">

<p>Name: <span th:text="\*{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>

当一个对象选择到位时，选定的对象也可以用作**#object**表达式变量的美元表达式：

<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>

如上所述，如果没有执行对象选择，则美元和星号语法完全相同。

<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>

*4.4链接网址*

由于它们的重要性，URL是Web应用程序模板中的一等公民，而*Thymeleaf Standard Dialect*对它们有特殊的语法，**@**语法：**@{...}**

有不同类型的网址：

* 绝对网址，例如 **http://www.thymeleaf.org**
* 相对URL，可以是：
  + 与页面相关的 **user/login.html**
  + 与上下文相关，例如**/itemdetails?id=3**（服务器中的上下文名称将自动添加）
  + 与服务器相关，就像**~/billing/processInvoice**（允许在同一服务器中的另一个上下文（=应用程序）中调用URL一样）。
  + 与协议相关的网址，例如 **//code.jquery.com/jquery-2.0.3.min.js**

Thymeleaf可以在任何情况下处理绝对URL，但是对于相对的URL，它将要求您使用实现**IWebContext**接口的上下文对象，该对象包含来自HTTP请求的一些信息并需要创建相对链接。

我们来使用这个新的语法。符合**th:href**属性：

<!-- Will produce 'http://localhost:8080/gtvg/order/details?orderId=3' (plus rewriting) -->

<a href="details.html"

th:href="@{http://localhost:8080/gtvg/order/details(orderId=${o.id})}">view</a>

<!-- Will produce '/gtvg/order/details?orderId=3' (plus rewriting) -->

<a href="details.html" th:href="@{/order/details(orderId=${o.id})}">view</a>

<!-- Will produce '/gtvg/order/3/details' (plus rewriting) -->

<a href="details.html" th:href="@{/order/{orderId}/details(orderId=${o.id})}">view</a>

有些事情要注意这里：

* **th:href**是一个属性修饰符属性：处理完成后，它将计算要使用的链接URL，并将**<a>**标记的href属性设置为该URL。
* 我们被允许使用URL参数的表达式（如您所见**orderId=${o.id}**）。所需的URL编码操作也将自动执行。
* 如果需要几个参数，这些参数将用逗号分隔 **@{/order/process(execId=${execId},execType='FAST')}**
* 变量模板也被允许在URL路径中使用 **@{/order/{orderId}/details(orderId=${orderId})}**
* 以**/**（like **/order/details**）开头的相对URL 将自动添加应用程序上下文名称的前缀。
* 如果cookie未启用或尚未知道，则**";jsessionid=..."**可能会向相对URL添加后缀，以便保留会话。这被称为*URL重写*，而Thymeleaf允许你通过使用**response.encodeURL(...)**Servlet API中的机制为每个URL 插入自己的重写过滤器。
* 该**th:href**标签允许我们（可选）**href**在我们的模板中具有静态属性，以便我们的模板链接可以直接打开以供原型设计使用。

与消息语法（**#{...}**）的情况一样，URL基础也可以是评估另一个表达式的结果：

<a th:href="@{${url}(orderId=${o.id})}">view</a>

<a th:href="@{'/details/'+${user.login}(orderId=${o.id})}">view</a>

***我们主页的菜单***

现在我们知道如何创建链接网址，那么在我们的家中为网站中的其他页面添加一个小菜单呢？

<p>Please select an option</p>

<ol>

<li><a href="product/list.html" th:href="@{/product/list}">Product List</a></li>

<li><a href="order/list.html" th:href="@{/order/list}">Order List</a></li>

<li><a href="subscribe.html" th:href="@{/subscribe}">Subscribe to our Newsletter</a></li>

<li><a href="userprofile.html" th:href="@{/userprofile}">See User Profile</a></li>

</ol>

***服务器根目录相对URL***

可以使用额外的语法来创建服务器根相对（而不是上下文根相对）URL，以便链接到同一服务器中的不同上下文。这些URL将被指定为**@{~/path/to/something}**

*4.5文字*

***文字文字***

文本文字只是在单引号之间指定的字符串。他们可以包括任何角色，但是你应该逃避他们中的任何单引号**\'**。

<p>

Now you are looking at a <span th:text="'working web application'">template file</span>.

</p>

***数字文字***

数字文字看起来完全一样：数字。

<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>

***布尔文字***

布尔文字是**true**和**false**。例如：

<div th:if="${user.isAdmin()} == false"> ...

请注意，在上面的例子中，这**== false**是写在大括号外面，因此是Thymeleaf本身谁来照顾它。如果它写在花括号里面，那么OGNL / SpringEL引擎将负责这个工作：

<div th:if="${user.isAdmin() == false}"> ...

***null文字***

该**null**文本也可用于：

<div th:if="${variable.something} == null"> ...

***文字标记***

数字，布尔值和空文字实际上是*文字记号*的特殊情况。

这些令牌允许在标准表达式中进行一些简化。它们的工作方式与文本文字（**'...'**）相同，但只允许使用字母（**A-Z**和**a-z**），数字（**0-9**），括号（**[**和**]**），点（**.**），连字符（**-**）和下划线（**\_**）。所以没有空格，没有逗号等

好的部分？令牌不需要任何引号。所以我们可以做到这一点：

<div th:class="content">...</div>

代替：

<div th:class="'content'">...</div>

*4.6**附加文本*

文本，无论是文字还是评估变量或消息表达式的结果，都可以使用**+**运算符轻松附加：

th:text="'The name of the user is ' + ${user.name}"

*4.7**文字替换*

文字替换允许轻松地格式化包含变量值的字符串，而不需要附加文字**'...' + '...'**。

这些替换必须被垂直条（**|**）包围，如：

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

其实相当于：

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

文字替换可以与其他类型的表达式结合使用：

<span th:text="${onevar} + ' ' + |${twovar}, ${threevar}|">

**注意：**只有变量表达式（**${...}**）允许在**|...|**文字替换中。没有其他文字（**'...'**），布尔/数字标记，条件表达式等等。

*4.8**算术运算*

一些算术运算也可用：**+**，**-**，**\***，**/**和**%**。

th:with="isEven=(${prodStat.count} % 2 == 0)"

请注意，这些运算符也可以在OGNL变量表达式中使用（在这种情况下，将由OGNL而不是Thymeleaf标准表达式引擎执行）：

th:with="isEven=${prodStat.count % 2 == 0}"

请注意，其中一些运算符存在文本别名：**div**（**/**），**mod**（**%**）。

*4.9**比较和平等*

在表达式中的值可以与进行比较**>**，**<**，**>=**和**<=**符号，像往常一样，也是**==**和**!=**运营商可以被用于检查平等（或缺乏）。请注意，XML建立了**<**和**>**符号不应该用在属性值中，所以它们应该被替换为**&lt;**和**&gt;**。

th:if="${prodStat.count} &gt; 1"

th:text="'Execution mode is ' + ( (${execMode} == 'dev')? 'Development' : 'Production')"

请注意，这些运算符中有一些是文本别名：**gt**（**>**），**lt**（**<**），**ge**（**>=**），**le**（**<=**），**not**（**!**）。还有**eq**（**==**），**neq**/ **ne**（**!=**）。

*4.10**条件表达式*

*条件表达式*意味着根据条件评估的结果（这本身就是另一个表达式）只评估两个表达式中的一个。

让我们来看一个示例片段（这次引入另一个*属性修饰符***th:class**）：

<tr th:class="${row.even}? 'even' : 'odd'">

...

</tr>

条件表达式（**condition**，**then**和**else**）的所有三个部分本身都是表达式，这意味着它们可以是变量（**${...}**，**\*{...}**），消息（**#{...}**），URL（**@{...}**）或文字（**'...'**）。

条件表达式也可以使用圆括号嵌套：

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

...

</tr>

其他表达式也可以省略，在这种情况下，如果条件为false，则返回空值：

<tr th:class="${row.even}? 'alt'">

...

</tr>

*4.11**默认表达式（Elvis操作符）*

一个*默认的表情*是一种特殊的条件值的没有*那么*一部分。它相当于像Groovy这样的一些语言中存在的*Elvis运算符*，并允许指定两个表达式，第二个表达式仅在第一个返回null的情况下进行计算。

让我们看看它在我们的用户个人资料页面中的行动：

<div th:object="${session.user}">

...

<p>Age: <span th:text="\*{age}?: '(no age specified)'">27</span>.</p>

</div>

正如你所看到的那样，**?:**只有在求**\*{age}**值结果为null的情况下，运算符才是，我们在这里使用它来指定一个名字的默认值（在这种情况下是一个字面值）。因此这相当于：

<p>Age: <span th:text="\*{age != null}? \*{age} : '(no age specified)'">27</span>.</p>

与条件值一样，它们可以在括号之间包含嵌套表达式：

<p>

Name:

<span th:text="\*{firstName}?: (\*{admin}? 'Admin' : #{default.username})">Sebastian</span>

</p>

*4.12预处理*

除了表达处理的所有这些功能之外，Thymeleaf还为我们提供了*预处理*表达式的可能性。

那预处理的东西是什么？它是在正常表达式之前完成的表达式的执行，允许修改将最终执行的实际表达式。

预处理的表达式与普通表达式完全相同，但是出现一个双下划线符号（如**\_\_${expression}\_\_**）。

假设我们有一个**Messages\_fr.properties**包含OGNL表达式的i18n 条目，它调用特定于语言的静态方法，如：

article.text=@myapp.translator.Translator@translateToFrench({0})

...和a **Messages\_es.properties equivalent**：

article.text=@myapp.translator.Translator@translateToSpanish({0})

我们可以根据语言环境创建一个评估一个表达式或另一个表达式的标记片段。为此，我们将首先选择表达式（通过预处理），然后让Thymeleaf执行它：

<p th:text="${\_\_#{article.text('textVar')}\_\_}">Some text here...</p>

请注意，法语区域设置的预处理步骤将创建以下等效项目：

<p th:text="${@myapp.translator.Translator@translateToFrench(textVar)}">Some text here...</p>

预处理字符串**\_\_**可以使用属性转义**\\_\\_**。

***5设置属性值***

本章将解释如何设置（或修改）标记标记中的属性值，这可能是设置标记主体内容后需要的下一个最基本的功能。

*5.1设置任何属性的值*

假设我们的网站发布时事通讯，并且我们希望我们的用户能够订阅它，所以我们**/WEB-INF/templates/subscribe.html**用一个表单创建一个模板：

<form action="subscribe.html">

<fieldset>

<input type="text" name="email" />

<input type="submit" value="Subscribe me!" />

</fieldset>

</form>

它看起来相当不错，但事实是这个文件看起来更像一个静态的XHTML页面，而不是一个Web应用程序的模板。首先，我们表单中的action属性静态链接到模板文件本身，这样就没有有用的URL重写的地方。其次，提交按钮中的值属性使其显示英文文本，但我们希望它是国际化的。

然后输入**th:attr**属性，以及更改其设置标签的属性值的能力：

<form action="subscribe.html" th:attr="action=@{/subscribe}">

<fieldset>

<input type="text" name="email" />

<input type="submit" value="Subscribe me!" th:attr="value=#{subscribe.submit}"/>

</fieldset>

</form>

这个概念非常简单：**th:attr**只需要一个为属性赋值的表达式。创建相应的控制器和消息文件后，处理该文件的结果将如预期的那样：

<form action="/gtvg/subscribe">

<fieldset>

<input type="text" name="email" />

<input type="submit" value="¡Suscríbeme!"/>

</fieldset>

</form>

除了新的属性值之外，还可以看到应用程序上下文名称已自动添加到前一章中所述的URL基础中**/gtvg/subscribe**。

但是，如果我们想一次设置多个属性呢？XML规则不允许您在标签中设置两次属性，因此**th:attr**会使用逗号分隔的分配列表，如下所示：

<img src="../../images/gtvglogo.png"

th:attr="src=@{/images/gtvglogo.png},title=#{logo},alt=#{logo}" />

给定所需的消息文件，这将输出：

<img src="/gtgv/images/gtvglogo.png" title="Logo de Good Thymes" alt="Logo de Good Thymes" />

*5.2将值设置为特定的属性*

到现在为止，你可能会想到这样的事情：

<input type="submit" value="Subscribe me!" th:attr="value=#{subscribe.submit}"/>

...是一个非常丑陋的标记。在一个属性值中指定一个赋值可能是非常实用的，但是如果你必须一直这样做的话，它不是创建模板的最优雅的方法。

百里香与你同意。这就是为什么事实上**th:attr**很少用于模板的原因。通常情况下，您将使用其他**th:\***任务设置特定标签属性的属性（而不仅仅是任何属性**th:attr**）。

Standard Dialect为我们提供了哪些属性来设置**value**我们的按钮的属性？那么，在一个相当明显的方式，它是**th:value**。我们来看一下：

<input type="submit" value="Subscribe me!" th:value="#{subscribe.submit}"/>

这看起来好多了！让我们试着**action**对**form**标签中的属性做同样的事情：

<form action="subscribe.html" th:action="@{/subscribe}">

你还记得**th:href**我们**home.html**之前放的那些吗？他们正是这种属性：

<li><a href="product/list.html" th:href="@{/product/list}">Product List</a></li>

像这样的属性有很多，每个属性都针对一个特定的XHTML或HTML5属性：

|  |  |  |
| --- | --- | --- |
| 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 |  |

*5.3一次设置多个值*

有两个叫比较特殊的属性**th:alt-title**和**th:lang-xmllang**可用于同时设置两个属性相同的值。特别：

* **th:alt-title**将设置**alt**和**title**。
* **th:lang-xmllang**将设置**lang**和**xml:lang**。

对于我们的GTVG主页，这将允许我们替换这个：

<img src="../../images/gtvglogo.png"

th:attr="src=@{/images/gtvglogo.png},title=#{logo},alt=#{logo}" />

...或者这个，相当于：

<img src="../../images/gtvglogo.png"

th:src="@{/images/gtvglogo.png}" th:title="#{logo}" th:alt="#{logo}" />

…这样：

<img src="../../images/gtvglogo.png"

th:src="@{/images/gtvglogo.png}" th:alt-title="#{logo}" />

*5.4附加和预先确定*

**th:attr**Thymeleaf 以等同的方式提供**th:attrappend**和**th:attrprepend**属性，它们将评估结果附加（后缀）或前缀（前缀）到现有的属性值中。

例如，您可能希望将CSS类的名称存储到上下文变量中的某个按钮中（未设置，仅添加），因为要使用的特定CSS类将取决于用户所做的某些操作之前。简单：

<input type="button" value="Do it!" class="btn" th:attrappend="class=${' ' + cssStyle}" />

如果您使用**cssStyle**设置为的变量处理此模板**"warning"**，您将获得：

<input type="button" value="Do it!" class="btn warning" />

标准方言中还有两个特定的*附加属性*：**th:classappend**和**th:styleappend**属性，它们用于在不覆盖现有元素的情况下向元素添加CSS类或*样式*片段：

<tr th:each="prod : ${prods}" class="row" th:classappend="${prodStat.odd}? 'odd'">

（不要担心这个**th:each**属性，它是一个*迭代属性*，我们稍后会讨论它。）

*5.5固定值布尔属性*

一些XHTML / HTML5属性是特殊的，不管它们是以特定的和固定的值存在于它们的元素中，还是根本不存在。

例如**checked**：

<input type="checkbox" name="option1" checked="checked" />

<input type="checkbox" name="option2" />

**"checked"**根据**checked**属性的XHTML标准，没有其他值超出允许范围（HTML5规则稍微宽松一点）。而同样的情况与**disabled**，**multiple**，**readonly**和**selected**。

标准方言包含的属性允许您通过评估一个条件来设置这些属性，以便如果评估为true，则该属性将被设置为其固定值，并且如果评估为false，则不会设置该属性：

<input type="checkbox" name="active" th:checked="${user.active}" />

标准方言中存在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 |

*5.6支持HTML5友好的属性和元素名称*

也可以使用完全不同的语法将处理器应用于模板，这对HTML5友好性更高。

<table>

<tr data-th-each="user : ${users}">

<td data-th-text="${user.login}">...</td>

<td data-th-text="${user.name}">...</td>

</tr>

</table>

该**data-{prefix}-{name}**语法是在HTML5中编写自定义属性的标准方法，无需开发人员使用任何名称空间名称**th:\***。Thymeleaf使得所有的方言都能自动使用这个语法（不仅是标准的）。

还有一个语法来指定自定义标签：**{prefix}-{name}**它遵循*W3C自定义元素规范*（更大的*W3C Web组件规范的一部分*）。例如，这可以用于**th:block**元素（或者也可以**th-block**），这将在后面的章节中解释。

**重要提示：**这个语法是对名称空间的补充**th:\***，它不会替代它。根本不打算在将来废弃名称空间的语法。

***6迭代***

到目前为止，我们已经创建了一个主页，一个用户档案页面，还有一个让用户订阅我们的通讯的页面...但是我们的产品呢？我们不应该建立一个产品清单，让游客知道我们卖什么？那么，显然是的。现在我们走了

*6.1迭代基础*

为了在我们的**/WEB-INF/templates/product/list.html**页面上列出我们的产品，我们需要一个表格。我们的每个产品都将显示在一行（一个**<tr>**元素），所以对于我们的模板，我们将需要创建一个*模板行* - 一个将举例说明我们希望如何显示每个产品 - 然后指示Thymeleaf *迭代它*一次为每个产品。

标准方言为我们提供了一个属性**th:each**。

***使用th：each***

对于我们的产品列表页面，我们需要一个控制器来从服务层检索产品列表并将其添加到模板上下文中：

public void process(

HttpServletRequest request, HttpServletResponse response,

ServletContext servletContext, TemplateEngine templateEngine) {

ProductService productService = new ProductService();

List<Product> allProducts = productService.findAll();

WebContext ctx = new WebContext(request, servletContext, request.getLocale());

ctx.setVariable("prods", allProducts);

templateEngine.process("product/list", ctx, response.getWriter());

}

然后我们将**th:each**在我们的模板中使用迭代产品列表：

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

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

<head>

<title>Good Thymes Virtual Grocery</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<link rel="stylesheet" type="text/css" media="all"

href="../../../css/gtvg.css" th:href="@{/css/gtvg.css}" />

</head>

<body>

<h1>Product list</h1>

<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>

<p>

<a href="../home.html" th:href="@{/}">Return to home</a>

</p>

</body>

</html>

**prod : ${prods}**您在上面看到的那个属性值的意思是“对于评估结果中的每个元素**${prods}**，重复这个模板片段将该元素设置成一个名为prod的变量”。让我们给每个我们看到的东西命名：

* 我们将调用**${prods}**的*迭代式*或*迭代变量*。
* 我们将调用**prod**的*迭代变量*或者干脆*ITER变量*。

请注意，**prod**iter变量只能在**<tr>**元素内部使用（包括内部标签**<td>**）。

***可重复的值***

不仅**java.util.List**可以在Thymeleaf中使用对象进行迭代。实际上，有一个相当完整的对象被认为可以被一个属性*迭代***th:each**：

* 任何对象的实现 **java.util.Iterable**
* 任何对象的实现**java.util.Map**。当迭代映射时，iter变量将是类的**java.util.Map.Entry**。
* 任何数组
* 任何其他对象将被视为包含对象本身的单值列表。

*6.2保持迭代状态*

当使用**th:each,**Thymeleaf提供了一个有用的机制来跟踪你的迭代*状态*：*状态变量*。

状态变量在一个**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*迭代索引*，从1开始。这是**count**属性。
* 迭代变量中的元素总数。这是**size**财产。
* 每个迭代的*iter变量*。这是**current**财产。
* 目前的迭代是偶数还是奇数。这些是**even/odd**布尔属性。
* 目前的迭代是否是第一个。这是**first**布尔属性。
* 目前的迭代是否是最后一个。这是**last**布尔属性。

让我们来看看在前面的例子中我们如何使用它：

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

</tr>

<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>

</table>

正如你所看到的，状态变量（**iterStat**在这个例子中）是**th:each**通过在iter变量本身之后写入名字来定义的，用逗号分隔。就像iter变量一样，状态变量只能在持有**th:each**属性的标签定义的代码片段中可用。

让我们来看看处理我们的模板的结果：

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Good Thymes Virtual Grocery</title>

<meta content="text/html; charset=UTF-8" http-equiv="Content-Type"/>

<link rel="stylesheet" type="text/css" media="all" href="/gtvg/css/gtvg.css" />

</head>

<body>

<h1>Product list</h1>

<table>

<tr>

<th colspan="1" rowspan="1">NAME</th>

<th colspan="1" rowspan="1">PRICE</th>

<th colspan="1" rowspan="1">IN STOCK</th>

</tr>

<tr>

<td colspan="1" rowspan="1">Fresh Sweet Basil</td>

<td colspan="1" rowspan="1">4.99</td>

<td colspan="1" rowspan="1">yes</td>

</tr>

<tr class="odd">

<td colspan="1" rowspan="1">Italian Tomato</td>

<td colspan="1" rowspan="1">1.25</td>

<td colspan="1" rowspan="1">no</td>

</tr>

<tr>

<td colspan="1" rowspan="1">Yellow Bell Pepper</td>

<td colspan="1" rowspan="1">2.50</td>

<td colspan="1" rowspan="1">yes</td>

</tr>

<tr class="odd">

<td colspan="1" rowspan="1">Old Cheddar</td>

<td colspan="1" rowspan="1">18.75</td>

<td colspan="1" rowspan="1">yes</td>

</tr>

</table>

<p>

<a href="/gtvg/" shape="rect">Return to home</a>

</p>

</body>

</html>

请注意，我们的迭代状态变量已经完美工作，只建立**odd**奇数行的CSS类（行计数从0开始）。

所有**<td>**标签中的colspan和rowspan属性以及**<a>**Thymeleaf自动添加的形状都是根据所选*XHTML 1.0严格*标准的DTD自动添加的，并将这些值确定为这些属性的默认值（请记住，我们的模板不会为他们设定一个价值）。不要担心它们，因为它们不会影响页面的显示。例如，如果我们使用HTML5（没有DTD），那么这些属性将永远不会被添加。

如果你没有明确地设置一个状态变量，Thymeleaf将总是为你添加一个**Stat**迭代变量的名字后缀：

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

</tr>

<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>

</table>

***7条件评估***

*7.1简单条件：“如果”和“除非”*

有时，如果满足某些条件，只需要模板的一个片段即可显示在结果中。

例如，想象一下，我们希望在产品表中显示每个产品存在的评论数量的列，并且如果有任何评论，则指向该产品的评论详细信息页面的链接。

为了做到这一点，我们将使用**th:if**属性：

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

<th>COMMENTS</th>

</tr>

<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>

</table>

在这里看到很多东西，所以让我们把重点放在重要的一行：

<a href="comments.html"

th:href="@{/product/comments(prodId=${prod.id})}"

th:if="${not #lists.isEmpty(prod.comments)}">view</a>

实际上，这个代码很少有解释：我们将创建一个链接到**/product/comments**带有**prodId**参数设置**id**的产品的评论页面（带有URL ），但是只有当产品有任何评论的时候。

让我们来看看结果标记（摆脱干净的视图的默认**rowspan**和**colspan**属性）：

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

<th>COMMENTS</th>

</tr>

<tr>

<td>Fresh Sweet Basil</td>

<td>4.99</td>

<td>yes</td>

<td>

<span>0</span> comment/s

</td>

</tr>

<tr class="odd">

<td>Italian Tomato</td>

<td>1.25</td>

<td>no</td>

<td>

<span>2</span> comment/s

<a href="/gtvg/product/comments?prodId=2">view</a>

</td>

</tr>

<tr>

<td>Yellow Bell Pepper</td>

<td>2.50</td>

<td>yes</td>

<td>

<span>0</span> comment/s

</td>

</tr>

<tr class="odd">

<td>Old Cheddar</td>

<td>18.75</td>

<td>yes</td>

<td>

<span>1</span> comment/s

<a href="/gtvg/product/comments?prodId=4">view</a>

</td>

</tr>

</table>

完善！这正是我们想要的。

请注意，该**th:if**属性不仅会评估*布尔*条件。它的功能稍微超出了这个范围，它将按照**true**以下规则评估指定的表达式：

* 如果值不为空：
  + 如果value是一个布尔值并且是**true**。
  + 如果值是一个数字并且是非零的
  + 如果值是一个字符，并且是非零的
  + 如果value是一个String而不是“false”，“off”或“no”
  + 如果值不是布尔值，数字，字符或字符串。
* （如果值为null，则th：如果将评估为false）。

此外，**th:if**还有一个负面的对应，**th:unless**我们可以在前面的例子中使用它，而不是**not**在OGNL表达式中使用：

<a href="comments.html"

th:href="@{/comments(prodId=${prod.id})}"

th:unless="${#lists.isEmpty(prod.comments)}">view</a>

*7.2交换语句*

还有一种方法可以在Java中使用相当于*开关*结构的有条件显示内容：**th:switch**/ **th:case**属性集。

它们的工作方式与您预期的完全相同

<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>

请注意，只要一个**th:case**属性被评估为**true**，**th:case**同一交换机上下文中的每一个其他属性都被评估为**false**。

默认选项被指定为**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>

***8模板布局***

*8.1包括模板片段*

***定义和引用片段***

我们经常希望在其他模板中包含我们的模板片段。常见的用途是页脚，标题，菜单...

为了做到这一点，Thymeleaf需要我们定义可用于包含的片段，我们可以通过使用该**th:fragment**属性来完成。

现在让我们说，我们要添加一个标准的版权页脚到我们所有的杂货页面，为此我们定义一个**/WEB-INF/templates/footer.html**包含这个代码的文件：

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

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

<body>

<div th:fragment="copy">

&copy; 2011 The Good Thymes Virtual Grocery

</div>

</body>

</html>

上面的代码定义了一个叫做的片段**copy**，我们可以很容易地在我们的主页中使用其中的一个**th:include**或者一个**th:replace**属性来包含：

<body>

...

<div th:include="footer :: copy"></div>

</body>

这两个包含属性的语法非常简单。有三种不同的格式：

* **"templatename::domselector"**或等同**templatename::[domselector]**包含在名为的模板上执行指定的DOM选择器所产生的片段**templatename**。
  + 请注意，**domselector**可以是一个单纯的片段名称，所以你可以指定一些**templatename::fragmentname**像**footer :: copy**上面那样简单的东西。

DOM选择器语法与XPath表达式和CSS选择器相似，请参阅[附录C](http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#appendix-c-dom-selector-syntax)了解有关此语法的更多信息。

* **"templatename"**包括名为的完整模板**templatename**。

请注意，您在**th:include**/ **th:replace**标记中使用的模板名称必须通过模板引擎当前使用的模板解析器来解析。

* **::domselector"**或者**"this::domselector"**包含来自相同模板的片段。

双方**templatename**并**domselector**在上面的例子可以是全功能的表达式（甚至条件语句！），如：

<div th:include="footer :: (${user.isAdmin}? #{footer.admin} : #{footer.normaluser})"></div>

碎片可以包括任何**th:\* attributes**。一旦片段被包含到目标模板（具有**th:include**/ **th:replace**属性的模板）中，这些属性将被评估，并且它们将能够引用在该目标模板中定义的任何上下文变量。

这种方法片段的一大优势是，你可以写你的碎片的代码，通过浏览器是完全显示的网页，有一个完整的，甚至验证XHTML结构，同时仍可以让Thymeleaf包括他们到其他模板的能力。

***没有引用片段 th:fragment***

另外，由于DOM选择器的强大功能，我们可以包含不使用任何**th:fragment**属性的片段。它甚至可以是来自不同应用程序的标记代码，根本不知道Thymeleaf：

...

<div id="copy-section">

&copy; 2011 The Good Thymes Virtual Grocery

</div>

...

我们可以使用上面的片段简单地引用它的**id**属性，类似于CSS选择器：

<body>

...

<div th:include="footer :: #copy-section"></div>

</body>

***th:include和之间的区别th:replace***

和之间有什么区别**th:include**和**th:replace**？而**th:include**将片段的内容包含到其宿主标记中，**th:replace**将实际上用片段的替换宿主标记。所以这样的HTML5片段：

<footer th:fragment="copy">

&copy; 2011 The Good Thymes Virtual Grocery

</footer>

...在主机**<div>**标签中包含两次，如下所示：

<body>

...

<div th:include="footer :: copy"></div>

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

</body>

...将导致：

<body>

...

<div>

&copy; 2011 The Good Thymes Virtual Grocery

</div>

<footer>

&copy; 2011 The Good Thymes Virtual Grocery

</footer>

</body>

该**th:substituteby**属性也可以用作别名**th:replace**，但推荐使用后者。请注意，**th:substituteby**在将来的版本中可能会被弃用。

*8.2可参数化的片段签名*

为了创建一个更*类似于*模板片段的机制，定义的片段**th:fragment**可以指定一组参数：

<div th:fragment="frag (onevar,twovar)">

<p th:text="${onevar} + ' - ' + ${twovar}">...</p>

</div>

这需要使用这两种语法之一的呼叫从片段**th:include**，**th:replace**：

<div th:include="::frag (${value1},${value2})">...</div>

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

请注意，订单在最后一个选项中并不重要：

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

***片段局部变量没有片段签名***

即使片段没有签名，如下所示：

<div th:fragment="frag">

...

</div>

我们可以使用上面指定的第二种语法来调用它们（只有第二个）：

<div th:include="::frag (onevar=${value1},twovar=${value2})">

这将是，实际上，相当于一个组合**th:include**和**th:with**：

<div th:include="::frag" th:with="onevar=${value1},twovar=${value2}">

**请注意**，一个片段的局部变量的规范 - 无论它是否具有签名 - 都不会导致上下文先前被清空。片段仍然能够像当前那样访问在调用模板中使用的每个上下文变量。

***th：断言模板内断言***

该**th:assert**属性可以指定一个逗号分隔的表达式列表，这些表达式应该被评估并且对于每个评估都是真实的，如果不是则引发异常。

<div th:assert="${onevar},(${twovar} != 43)">...</div>

这对于验证片段签名中的参数非常方便：

<header th:fragment="contentheader(title)" th:assert="${!#strings.isEmpty(title)}">...</header>

*8.3删除模板碎片*

我们来重温产品列表模板的最后一个版本：

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

<th>COMMENTS</th>

</tr>

<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>

</table>

这个代码作为一个模板是很好的，但是作为一个静态页面（当没有Thymeleaf处理的时候直接用浏览器打开的话），它不会是一个很好的原型。

为什么？因为尽管浏览器可以完美显示，但该表只有一行，并且该行具有模拟数据。作为一个原型，它看起来不够现实...我们应该有多个产品，*我们需要更多的行*。

所以让我们添加一些：

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

<th>COMMENTS</th>

</tr>

<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>

</table>

好吧，现在我们有三个，对于一个原型肯定更好。但是...当我们用Thymeleaf处理时会发生什么事情？

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

<th>COMMENTS</th>

</tr>

<tr>

<td>Fresh Sweet Basil</td>

<td>4.99</td>

<td>yes</td>

<td>

<span>0</span> comment/s

</td>

</tr>

<tr class="odd">

<td>Italian Tomato</td>

<td>1.25</td>

<td>no</td>

<td>

<span>2</span> comment/s

<a href="/gtvg/product/comments?prodId=2">view</a>

</td>

</tr>

<tr>

<td>Yellow Bell Pepper</td>

<td>2.50</td>

<td>yes</td>

<td>

<span>0</span> comment/s

</td>

</tr>

<tr class="odd">

<td>Old Cheddar</td>

<td>18.75</td>

<td>yes</td>

<td>

<span>1</span> comment/s

<a href="/gtvg/product/comments?prodId=4">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>

</table>

最后两行是模拟行！当然，他们是：迭代只适用于第一行，所以没有理由为什么Thymeleaf应该删除其他两个。

我们需要一种在模板处理过程中删除这两行的方法。我们使用**th:remove**第二个和第三个**<tr>**标签上的属性：

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

<th>COMMENTS</th>

</tr>

<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" th:remove="all">

<td>Blue Lettuce</td>

<td>9.55</td>

<td>no</td>

<td>

<span>0</span> comment/s

</td>

</tr>

<tr th:remove="all">

<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>

</table>

一旦处理完毕，所有内容都会再次显示，因为它应该：

<table>

<tr>

<th>NAME</th>

<th>PRICE</th>

<th>IN STOCK</th>

<th>COMMENTS</th>

</tr>

<tr>

<td>Fresh Sweet Basil</td>

<td>4.99</td>

<td>yes</td>

<td>

<span>0</span> comment/s

</td>

</tr>

<tr class="odd">

<td>Italian Tomato</td>

<td>1.25</td>

<td>no</td>

<td>

<span>2</span> comment/s

<a href="/gtvg/product/comments?prodId=2">view</a>

</td>

</tr>

<tr>

<td>Yellow Bell Pepper</td>

<td>2.50</td>

<td>yes</td>

<td>

<span>0</span> comment/s

</td>

</tr>

<tr class="odd">

<td>Old Cheddar</td>

<td>18.75</td>

<td>yes</td>

<td>

<span>1</span> comment/s

<a href="/gtvg/product/comments?prodId=4">view</a>

</td>

</tr>

</table>

那么**all**这个属性的价值又是什么呢？那么，实际上**th:remove**可以通过五种不同的方式，取决于它的价值：

* **all**：删除包含标签及其所有子项。
* **body**：不要删除包含标签，而是删除其所有子项。
* **tag**：删除包含标签，但不要删除其子项。
* **all-but-first**：删除第一个标签以外的所有标签。
* **none**： 没做什么。该值对于动态评估非常有用。

这个**all-but-first**价值有什么用处？**th:remove="all"**原型设计时，我们可以节省一些：

<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>

该**th:remove**属性可采取任何*Thymeleaf标准表示*，因为它返回允许字符串值中的一个，只要（**all**，**tag**，**body**，**all-but-first**或**none**）。

这意味着清除可能是有条件的，如：

<a href="/something" th:remove="${condition}? tag : none">Link text not to be removed</a>

另外请注意，**th:remove**考虑**null**一个同义词**none**，以便下面的工作完全如上例所示：

<a href="/something" th:remove="${condition}? tag">Link text not to be removed</a>

在这种情况下，如果**${condition}**是假的，**null**将被退回，因此不会执行移除。

***9个局部变量***

Thymeleaf调用*局部变量*为模板的特定片段定义的那些变量，并且只能在该片段内进行评估。

我们已经看到的例子是**prod**我们的产品列表页面中的iter变量：

<tr th:each="prod : ${prods}">

...

</tr>

该**prod**变量只能在**<tr>**标签的绑定中使用。特别：

* 它将可用于**th:\***在该标签中执行的任何其他属性，*优先级*低于**th:each**（这意味着它们将在之后执行**th:each**）。
* 它将可用于**<tr>**标签的任何子元素，例如**<td>**元素。

Thymeleaf为您提供了一种无迭代声明局部变量的方法。它是**th:with**属性，它的语法就像属性值赋值：

<div th:with="firstPer=${persons[0]}">

<p>

The name of the first person is <span th:text="${firstPer.name}">Julius Caesar</span>.

</p>

</div>

当**th:with**被处理时，该**firstPer**变量被创建为一个局部变量，并被添加到来自上下文的变量映射中，以便它可以像在上下文中从头开始声明的任何其他变量一样可用，但是仅在包含**<div>**标签。

您可以使用通常的多重赋值语法同时定义多个变量：

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

<p>

The name of the first person is <span th:text="${firstPer.name}">Julius Caesar</span>.

</p>

<p>

But the name of the second person is

<span th:text="${secondPer.name}">Marcus Antonius</span>.

</p>

</div>

该**th:with**属性允许重用在相同属性中定义的变量：

<div th:with="company=${user.company + ' Co.'},account=${accounts[company]}">...</div>

让我们在我们的杂货店主页使用这个！请记住我们为输出格式化日期而编写的代码吗？

<p>

Today is:

<span th:text="${#calendars.format(today,'dd MMMM yyyy')}">13 february 2011</span>

</p>

那么，如果我们想要**"dd MMMM yyyy"**真正依赖于语言环境呢？例如，我们可能想要添加以下消息给我们**home\_en.properties**：

date.format=MMMM dd'','' yyyy

...和我们的一个等价的**home\_es.properties**：

date.format=dd ''de'' MMMM'','' yyyy

现在，让我们使用**th:with**获取本地化的日期格式到一个变量，然后在我们的**th:text**表达式中使用它：

<p th:with="df=#{date.format}">

Today is: <span th:text="${#calendars.format(today,df)}">13 February 2011</span>

</p>

这是干净和容易的。事实上，鉴于这一事实**th:with**具有较高的**precedence**比**th:text**，我们可以解决这一切的**span**标签：

<p>

Today is:

<span th:with="df=#{date.format}"

th:text="${#calendars.format(today,df)}">13 February 2011</span>

</p>

你可能会想：优先？我们还没有谈到这个呢！那么不要担心，因为这正是下一章的内容。

***10属性优先***

**th:\***在同一个标​​签中写入多个属性会发生什么？例如：

<ul>

<li th:each="item : ${items}" th:text="${item.description}">Item description here...</li>

</ul>

当然，我们希望这个**th:each**属性在执行之前**th:text**能够得到我们想要的结果，但是考虑到DOM（文档对象模型）标准没有给出任何类型的属性的含义标签被写入，必须在属性本身中建立一个*优先*机制，以确保这将按预期工作。

所以，所有的Thymeleaf属性都定义了一个数字优先级，它确定了它们在标签中的执行顺序。这个命令是：

| **订购** | **特征** | **属性** |
| --- | --- | --- |
| 1 | 片段包含 | th:include th:replace |
| 2 | 片段迭代 | th:each |
| 3 | 有条件的评估 | th:if th:unless th:switch th:case |
| 4 | 局部变量定义 | th:object th:with |
| 五 | 一般属性修改 | th:attr th:attrprepend th:attrappend |
| 6 | 特定的属性修改 | th:value th:href th:src ... |
| 7 | 文本（标签主体修改） | th:text th:utext |
| 8 | 片段规格 | th:fragment |
| 9 | 片段去除 | th:remove |

这种优先机制意味着如果属性位置反转，上面的迭代片段将给出完全相同的结果（尽管它可能会稍微不可读）：

<ul>

<li th:text="${item.description}" th:each="item : ${items}">Item description here...</li>

</ul>

***11.评论和块***

*11.1。标准的HTML / XML评论*

标准的HTML / XML注释**<!-- ... -->**可以在thymeleaf模板的任何地方使用。这些注释中的任何内容都不会被Thymeleaf或浏览器处理，只会被逐字复制到结果中：

<!-- User info follows -->

<div th:text="${...}">

...

</div>

*11.2。Thymeleaf解析器级注释块*

解析器级注释块是在thymeleaf解析时从模板中简单删除的代码。他们看起来像这样：

<!--/\* This code will be removed at thymeleaf parsing time! \*/-->

Thymeleaf将消除绝对之间的一切**<!--/\***和**\*/-->**，所以这些注释块也可以用于显示当模板是静态开放代码，知道当thymeleaf处理它，它都将被删除：

<!--/\*-->

<div>

you can see me only before thymeleaf processes me!

</div>

<!--\*/-->

这样做可能对于有很多原型的表格来说非常方便**<tr>**，例如：

<table>

<tr th:each="x : ${xs}">

...

</tr>

<!--/\*-->

<tr>

...

</tr>

<tr>

...

</tr>

<!--\*/-->

</table>

*11.3。Thymeleaf原型评论块*

当模板静态打开（即作为原型）时，Thymeleaf允许将特殊注释块的定义标记为注释，但是当Thymeleaf在执行模板时将其视为正常的标记。

<span>hello!</span>

<!--/\*/

<div th:text="${...}">

...

</div>

/\*/-->

<span>goodbye!</span>

Thymeleaf的解析系统将简单地删除**<!--/\*/**和**/\*/-->**标记，而不是它的内容，因此将不留下评论。所以在执行模板时，Thymeleaf会看到这个：

<span>hello!</span>

<div th:text="${...}">

...

</div>

<span>goodbye!</span>

就像解析器级别的注释块一样，注意这个特性是与方言无关的。

*11.4。合成****th:block****标签*

标准方言中包括的百里香唯一的元素处理器（不是属性）是**th:block**。

**th:block**是一个单纯的属性容器，允许模板开发人员指定他们想要的任何属性。Thymeleaf将执行这些属性，然后简单地使块消失无踪。

因此，例如，在创建**<tr>**每个元素需要多个表的迭代表时，这可能是有用的：

<table>

<th:block th:each="user : ${users}">

<tr>

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

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

</tr>

<tr>

<td colspan="2" th:text="${user.address}">...</td>

</tr>

</th:block>

</table>

与原型评论块结合使用时特别有用：

<table>

<!--/\*/ <th:block th:each="user : ${users}"> /\*/-->

<tr>

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

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

</tr>

<tr>

<td colspan="2" th:text="${user.address}">...</td>

</tr>

<!--/\*/ </th:block> /\*/-->

</table>

注意这个解决方案如何使模板成为有效的HTML（不需要在**<div>**里面添加禁止块**<table>**），并且在浏览器中以静态方式打开时仍然可以正常工作！

***12内联***

*12.1文本内联*

虽然标准方言允许我们通过使用标签属性来完成我们可能需要的几乎所有事情，但是在某些情况下，我们可能更喜欢将表达式直接写入到我们的HTML文本中。例如，我们可以更喜欢这样写：

<p>Hello, [[${session.user.name}]]!</p>

而不是这个：

<p>Hello, <span th:text="${session.user.name}">Sebastian</span>!</p>

表达式之间**[[...]]**的表达式被认为是Thymeleaf中的表达式内联，在这些表达式中，您可以使用任何类型的表达式，这些表达式在**th:text**属性中也是有效的。

为了让内联工作，我们必须使用激活**th:inline**属性，它有三个可能的值或模式（**text**，**javascript**和**none**）。让我们试试**text**：

<p th:inline="text">Hello, [[${session.user.name}]]!</p>

持有标签的标签**th:inline**不一定是包含内联表达式的标签，任何父标签都可以：

<body th:inline="text">

...

<p>Hello, [[${session.user.name}]]!</p>

...

</body>

所以你现在可能会问：*为什么我们从一开始就这样做呢？这是比所有这些* **th:text** *属性更少的代码！*那么要小心，因为虽然你可能会发现内联很有趣，但是你应该永远记住内联表达式在静态打开它们的时候会在你的HTML文件中逐字显示出来，所以你可能不会再把它们用作原型了！

浏览器如何在不使用内联的情况下静态显示我们的代码片段

Hello, Sebastian!

...并使用它...

Hello, [[${session.user.name}]]!

...很清楚

*12.2脚本内联（JavaScript和Dart）*

Thymeleaf为其内联功能提供了一系列“脚本”模式，以便您可以将数据集成到使用某些脚本语言创建的脚本中。

当前的脚本模式是**javascript**（**th:inline="javascript"**）和**dart**（**th:inline="dart"**）。

我们可以使用脚本内联的第一件事是将表达式的值写入我们的脚本中，如：

<script th:inline="javascript">

/\*<![CDATA[\*/

...

var username = /\*[[${session.user.name}]]\*/ 'Sebastian';

...

/\*]]>\*/

</script>

在**/\*[[...]]\*/**语法，指示Thymeleaf评估包含的表达式。但是这里有更多的含义：

* 作为javascript评论（**/\*...\*/**），我们的表达式将被忽略，当浏览器静态显示页面。
* 内联表达式（**'Sebastian'**）之后的代码将在静态显示页面时执行。
* Thymeleaf将执行表达式并插入结果，但它也将删除内联表达式本身（静态显示时执行的部分）后面的所有代码。

所以，执行这个的结果是：

<script th:inline="javascript">

/\*<![CDATA[\*/

...

var username = 'John Apricot';

...

/\*]]>\*/

</script>

你也可以在没有注释的情况下使用相同的效果，但这会使你的脚本在静态加载时失败：

<script th:inline="javascript">

/\*<![CDATA[\*/

...

var username = [[${session.user.name}]];

...

/\*]]>\*/

</script>

请注意，这个评估是聪明的，不限于字符串。Thymeleaf将正确地使用Javascript / Dart语法编写以下类型的对象：

* 字符串
* 数字
* 布尔
* 数组
* 集合
* 地图
* 豆（与*getter*和*setter*方法的对象）

例如，如果我们有以下代码：

<script th:inline="javascript">

/\*<![CDATA[\*/

...

var user = /\*[[${session.user}]]\*/ null;

...

/\*]]>\*/

</script>

该**${session.user}**表达式将评估为一个**User**对象，而Thymeleaf将正确地将其转换为Javascript语法：

<script th:inline="javascript">

/\*<![CDATA[\*/

...

var user = {'age':null,'firstName':'John','lastName':'Apricot',

'name':'John Apricot','nationality':'Antarctica'};

...

/\*]]>\*/

</script>

***添加代码***

使用javascript内联的一个附加功能是能够在特殊注释语法之间包含代码，**/\*[+...+]\*/**以便Thymeleaf在处理模板时自动取消注释该代码：

var x = 23;

/\*[+

var msg = 'This is a working application';

+]\*/

var f = function() {

...

将被执行为：

var x = 23;

var msg = 'This is a working application';

var f = function() {

...

您可以在这些注释中包含表达式，并对它们进行评估：

var x = 23;

/\*[+

var msg = 'Hello, ' + [[${session.user.name}]];

+]\*/

var f = function() {

...

***删除代码***

也可以让Thymeleaf在特殊**/\*[- \*/**和**/\* -]\*/**注释之间移除代码，如下所示：

var x = 23;

/\*[- \*/

var msg = 'This is a non-working template';

/\* -]\*/

var f = function() {

...

***13验证和文档类型***

*13.1验证模板*

正如前面提到的，Thymeleaf为我们提供外的开箱即处理它们之前验证我们的模板两个标准的模板模式：**VALIDXML**和**VALIDXHTML.**这些模式要求我们的模板，不仅*格式良好的XML*（这是他们应该永远是），但在实际上按照规定有效**DTD**。

问题是，如果我们使用**VALIDXHTML**包含如下**DOCTYPE**子句的模板：

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

......我们将因为获得验证错误**th:\***标签不根据存在**DTD.**这是完全正常的，因为W3C显然没有理由包括在他们的标准，但Thymeleaf的功能，我们该如何解决呢？通过改变**DTD.**

Thymeleaf包含一系列**DTD**反映XHTML标准原始文件的文件，但添加**th:\***了标准方言的所有可用属性。这就是为什么我们一直在我们的模板中使用它：

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

该**SYSTEM**标识符指示Thymeleaf解析器解析特殊的启用Thymeleaf的**XHTML 1.0 Strict DTD**文件并将其用于验证我们的模板。不用担心**http**，因为这只是一个标识符，**DTD**文件将从Thymeleaf的jar文件本地读取。

请注意，因为这个DOCTYPE声明是一个完全有效的声明，所以如果我们打开一个浏览器将我们的模板静态显示为一个原型，它将在*标准模式下*呈现。

在这里，您可以获得**DTD**支持所有支持的XHTML风格的Thymeleaf-enabled 声明的完整集合：

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-transitional-thymeleaf-4.dtd">

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-frameset-thymeleaf-4.dtd">

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml11-thymeleaf-4.dtd">

还要注意的是，为了使IDE快乐起来，并且即使不在验证模式下工作，也需要**th**在**html**标记中声明命名空间：

<html xmlns="http://www.w3.org/1999/xhtml"

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

*13.2 Doctype翻译*

我们的模板有**DOCTYPE**这样的好处：

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

但是，我们的Web应用程序将XHTML文档发送**DOCTYPE**到客户端浏览器并不好，因为：

* 他们不是**PUBLIC**（他们是**SYSTEM DOCTYPE**），因此我们的网站将无法验证与W3C验证。
* 他们是不需要的，因为一旦处理，所有的**th:\***标签将消失。

这就是为什么Thymeleaf包含一个*DOCTYPE翻译*机制，它会自动将您的特定**DOCTYPE**于thymeleaf的XHTML 转换为标准**DOCTYPE**。

例如，如果您的模板是*XHTML 1.0 Strict，*并且看起来像这样：

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

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

...

</html>

在使用Thymeleaf过程模板之后，生成的XHTML将如下所示：

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

...

</html>

您不必为这些转换发生任何事情：Thymeleaf将自动处理它们。

***14我们杂货的更多页面***

现在我们了解了很多关于使用Thymeleaf的知识，我们可以在我们的网站上添加一些新的页面进行订单管理。

请注意，我们将专注于XHTML代码，但是如果您想查看相应的控制器，则可以查看捆绑的源代码。

*14.1订单列表*

我们首先创建一个订单列表页面**/WEB-INF/templates/order/list.html**：

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

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

<head>

<title>Good Thymes Virtual Grocery</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<link rel="stylesheet" type="text/css" media="all"

href="../../../css/gtvg.css" th:href="@{/css/gtvg.css}" />

</head>

<body>

<h1>Order list</h1>

<table>

<tr>

<th>DATE</th>

<th>CUSTOMER</th>

<th>TOTAL</th>

<th></th>

</tr>

<tr th:each="o : ${orders}" th:class="${oStat.odd}? 'odd'">

<td th:text="${#calendars.format(o.date,'dd/MMM/yyyy')}">13 jan 2011</td>

<td th:text="${o.customer.name}">Frederic Tomato</td>

<td th:text="${#aggregates.sum(o.orderLines.{purchasePrice \* amount})}">23.32</td>

<td>

<a href="details.html" th:href="@{/order/details(orderId=${o.id})}">view</a>

</td>

</tr>

</table>

<p>

<a href="../home.html" th:href="@{/}">Return to home</a>

</p>

</body>

</html>

除了这个OGNL魔法的一点点之外，这里什么都不应该给我们带来惊喜：

<td th:text="${#aggregates.sum(o.orderLines.{purchasePrice \* amount})}">23.32</td>

什么，做，对于每个订单行（**OrderLine**在该命令对象），乘其**purchasePrice**和**amount**性质（通过调用相应的**getPurchasePrice()**和**getAmount()**方法）并将结果返回到数字的列表，后来由聚集**#aggregates.sum(...)**函数，以获得订单总额价钱。

你一定要喜欢OGNL的力量。

*14.2订单详情*

现在订购详细信息页面，我们将大量使用星号语法：

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

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

<head>

<title>Good Thymes Virtual Grocery</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<link rel="stylesheet" type="text/css" media="all"

href="../../../css/gtvg.css" th:href="@{/css/gtvg.css}" />

</head>

<body th:object="${order}">

<h1>Order details</h1>

<div>

<p><b>Code:</b> <span th:text="\*{id}">99</span></p>

<p>

<b>Date:</b>

<span th:text="\*{#calendars.format(date,'dd MMM yyyy')}">13 jan 2011</span>

</p>

</div>

<h2>Customer</h2>

<div th:object="\*{customer}">

<p><b>Name:</b> <span th:text="\*{name}">Frederic Tomato</span></p>

<p>

<b>Since:</b>

<span th:text="\*{#calendars.format(customerSince,'dd MMM yyyy')}">1 jan 2011</span>

</p>

</div>

<h2>Products</h2>

<table>

<tr>

<th>PRODUCT</th>

<th>AMOUNT</th>

<th>PURCHASE PRICE</th>

</tr>

<tr th:each="ol,row : \*{orderLines}" th:class="${row.odd}? 'odd'">

<td th:text="${ol.product.name}">Strawberries</td>

<td th:text="${ol.amount}" class="number">3</td>

<td th:text="${ol.purchasePrice}" class="number">23.32</td>

</tr>

</table>

<div>

<b>TOTAL:</b>

<span th:text="\*{#aggregates.sum(orderLines.{purchasePrice \* amount})}">35.23</span>

</div>

<p>

<a href="list.html" th:href="@{/order/list}">Return to order list</a>

</p>

</body>

</html>

除了这个嵌套的对象选择之外，这里并没有太多新东西：

<body th:object="${order}">

...

<div th:object="\*{customer}">

<p><b>Name:</b> <span th:text="\*{name}">Frederic Tomato</span></p>

...

</div>

...

</body>

...这**\*{name}**实际上相当于：

<p><b>Name:</b> <span th:text="${order.customer.name}">Frederic Tomato</span></p>

***15更多关于配置***

*15.1模板解析器*

对于我们的Good Thymes虚拟食品杂货店，我们选择了一个**ITemplateResolver**实现**ServletContextTemplateResolver**，它允许我们从Servlet上下文获取模板作为资源。

除了让您能够通过实施**ITemplateResolver,**Thymeleaf 来创建您自己的模板解析程序外，还包括三个其他开箱即用的实现：

* **org.thymeleaf.templateresolver.ClassLoaderTemplateResolver**，它将模板解析为类加载器资源，如：

return Thread.currentThread().getContextClassLoader().getResourceAsStream(templateName);

* **org.thymeleaf.templateresolver.FileTemplateResolver**，它将模板解析为来自文件系统的文件，如：

return new FileInputStream(new File(templateName));

* **org.thymeleaf.templateresolver.UrlTemplateResolver**，它将模板解析为URL（甚至是非本地模板），例如：

return (new URL(templateName)).openStream();

所有预捆绑的实现**ITemplateResolver**允许相同的一组配置参数，其中包括：

* 前缀和后缀（如已经看到）：
* templateResolver.setPrefix("/WEB-INF/templates/");

templateResolver.setSuffix(".html");

* 允许使用不直接对应于文件名的模板名称的模板别名。如果同时存在后缀/前缀和别名，则会在前缀/后缀之前应用别名：
* templateResolver.addTemplateAlias("adminHome","profiles/admin/home");

templateResolver.setTemplateAliases(aliasesMap);

* 阅读模板时应用的编码：

templateResolver.setEncoding("UTF-8");

* 默认模板模式，以及用于为特定模板定义其他模式的模式：
* // Default is TemplateMode.XHTML
* templateResolver.setTemplateMode("HTML5");

templateResolver.getXhtmlTemplateModePatternSpec().addPattern("\*.xhtml");

* 模板缓存的默认模式以及用于定义特定模板是否可缓存的模式：
* // Default is true
* templateResolver.setCacheable(false);

templateResolver.getCacheablePatternSpec().addPattern("/users/\*");

* TTL，以毫秒为单位，解析模板缓存条目源于此模板解析器。如果未设置，则从缓存中删除条目的唯一方法是LRU（缓存最大大小超出，条目最早）。
* // Default is no TTL (only LRU would remove entries)

templateResolver.setCacheTTLMs(60000L);

而且，一个模板引擎可以指定几个模板解析器，在这种情况下，可以在它们之间为模板解析建立一个顺序，这样，如果第一个模板不能解析模板，则第二个被问到，依此类推：

ClassLoaderTemplateResolver classLoaderTemplateResolver = new ClassLoaderTemplateResolver();

classLoaderTemplateResolver.setOrder(Integer.valueOf(1));

ServletContextTemplateResolver servletContextTemplateResolver = new ServletContextTemplateResolver();

servletContextTemplateResolver.setOrder(Integer.valueOf(2));

templateEngine.addTemplateResolver(classLoaderTemplateResolver);

templateEngine.addTemplateResolver(servletContextTemplateResolver);

应用多个模板解析器时，建议为每个模板解析器指定模式，以便Thymeleaf可以快速丢弃那些无意解析模板的模板解析器，从而提高性能。这样做并不是要求，而是一个优化：

ClassLoaderTemplateResolver classLoaderTemplateResolver = new ClassLoaderTemplateResolver();

classLoaderTemplateResolver.setOrder(Integer.valueOf(1));

// This classloader will not be even asked for any templates not matching these patterns

classLoaderTemplateResolver.getResolvablePatternSpec().addPattern("/layout/\*.html");

classLoaderTemplateResolver.getResolvablePatternSpec().addPattern("/menu/\*.html");

ServletContextTemplateResolver servletContextTemplateResolver = new ServletContextTemplateResolver();

servletContextTemplateResolver.setOrder(Integer.valueOf(2));

*15.2消息解析器*

我们没有明确指定我们的Grocery应用程序的消息解析器实现，正如之前所解释的，这意味着所使用的实现是一个**org.thymeleaf.messageresolver.StandardMessageResolver**对象。

这个**StandardMessageResolver,**以已经解释过的方式查找与模板同名的消息文件实际上是Thymeleaf核心开箱即用的唯一消息解析器实现，当然，您也可以通过实现**org.thymeleaf.messageresolver.IMessageResolver**接口来创建自己的消息解析器。

Thymeleaf + Spring集成包提供了一个**IMessageResolver**实现，它使用标准的Spring方法通过使用**MessageSource**对象来检索外部消息。

如果你想添加一个消息解析器（或更多）到模板引擎？简单：

// For setting only one

templateEngine.setMessageResolver(messageResolver);

// For setting more than one

templateEngine.addMessageResolver(messageResolver);

为什么你想要有多个消息解析器？出于与模板解析器相同的原因：消息解析器是有序的，如果第一个解析器不能解析特定的消息，则会询问第二个，然后是第三个，等等。

*15.3记录*

Thymeleaf对日志记录非常重视，并且总是试图通过日志记录界面提供最大量的有用信息。

所使用的日志库**slf4j,**实际上是您可能希望在应用程序中使用的任何日志实现的桥梁（例如**log4j**）。

Thymeleaf班会记录**TRACE**，**DEBUG**并**INFO**-level信息，这取决于你的愿望的详细程度，并且除了一般的记录它会使用与TemplateEngine类，你可以为不同的目的而单独配置相关的三个特殊记录器：

* **org.thymeleaf.TemplateEngine.CONFIG** 将在初始化期间输出库的详细配置。
* **org.thymeleaf.TemplateEngine.TIMER** 将输出有关处理每个模板所花费的时间的信息（对基准测试有用！）
* **org.thymeleaf.TemplateEngine.cache**是输出有关缓存的特定信息的一组记录器的前缀。虽然缓存记录器的名称可以由用户配置，因此可以更改，默认情况下它们是：
  + **org.thymeleaf.TemplateEngine.cache.TEMPLATE\_CACHE**
  + **org.thymeleaf.TemplateEngine.cache.FRAGMENT\_CACHE**
  + **org.thymeleaf.TemplateEngine.cache.MESSAGE\_CACHE**
  + **org.thymeleaf.TemplateEngine.cache.EXPRESSION\_CACHE**

Thymeleaf的日志记录基础设施的配置示例**log4j**可能是：

log4j.logger.org.thymeleaf=DEBUG

log4j.logger.org.thymeleaf.TemplateEngine.CONFIG=TRACE

log4j.logger.org.thymeleaf.TemplateEngine.TIMER=TRACE

log4j.logger.org.thymeleaf.TemplateEngine.cache.TEMPLATE\_CACHE=TRACE

***16模板缓存***

Thymeleaf的工作得益于一个DOM处理引擎和一系列处理器（一种用于需要应用逻辑的每种类型的节点），用于修改文档的DOM树，以便通过将此树与您的数据结合来创建您期望的结果。

它还包括 - 默认 - 存储解析的模板的缓存，这是在处理它们之前通过读取和解析模板文件得到的DOM树。这在Web应用程序中工作时尤其有用，并建立在以下概念上：

* 输入/输出几乎总是任何应用程序中最慢的部分。与内存相比，内存过程非常快速。
* 克隆现有的内存中的DOM树总比读模板文件，解析它并为它创建一个新的DOM对象树要快得多。
* Web应用程序通常只有几十个模板。
* 模板文件是中小型的，在应用程序运行时不会被修改。

这一切都导致了这样的想法：在Web应用程序中缓存最常用的模板是可行的，而不会浪费大量的内存，并且还会节省大量的时间，这将花费在一小部分文件上的输入/输出操作上那实际上从来没有改变。

我们怎样才能控制这个缓存呢？首先，我们已经了解到，我们可以在模板解析器中启用或禁用它，即使只对特定的模板起作用：

// Default is true

templateResolver.setCacheable(false);

templateResolver.getCacheablePatternSpec().addPattern("/users/\*");

另外，我们可以通过建立自己的*Cache Manager*对象来修改它的配置，这个对象可以是默认**StandardCacheManager**实现的一个实例：

// Default is 50

StandardCacheManager cacheManager = new StandardCacheManager();

cacheManager.setTemplateCacheMaxSize(100);

...

templateEngine.setCacheManager(cacheManager);

**org.thymeleaf.cache.StandardCacheManager**有关配置缓存的更多信息，请参阅javadoc API 。

可以从模板缓存中手动删除条目：

// Clear the cache completely

templateEngine.clearTemplateCache();

// Clear a specific template from the cache

templateEngine.clearTemplateCacheFor("/users/userList");

***附录A：表达基本对象***

一些对象和变量映射总是可以在变量表达式中调用（由OGNL或SpringEL执行）。我们来看看他们：

***基础对象***

* **#ctx**：上下文对象。这将是一个实施**org.thymeleaf.context.IContext**，**org.thymeleaf.context.IWebContext**取决于我们的环境（独立或网络）。如果我们使用*Spring集成模块*，它将是一个实例**org.thymeleaf.spring[3|4].context.SpringWebContext**。

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.context.IContext

\* ======================================================================

\*/

${#ctx.locale}

${#ctx.variables}

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.context.IWebContext

\* ======================================================================

\*/

${#ctx.applicationAttributes}

${#ctx.httpServletRequest}

${#ctx.httpServletResponse}

${#ctx.httpSession}

${#ctx.requestAttributes}

${#ctx.requestParameters}

${#ctx.servletContext}

${#ctx.sessionAttributes}

* **#locale**：直接访问**java.util.Locale**与当前请求关联。

${#locale}

* **#vars**：**org.thymeleaf.context.VariablesMap**Context中所有变量的实例（通常包含在**#ctx.variables**本地变量中的变量）。

对这个对象评估不合格的表达式。其实**${something}**完全等同于（但比它更漂亮）**${#vars.something}**。

**#root** 是同一个对象的synomyn。

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.context.VariablesMap

\* ======================================================================

\*/

${#vars.get('foo')}

${#vars.containsKey('foo')}

${#vars.size()}

...

***请求/会话属性的Web上下文命名空间等***

在Web环境中使用Thymeleaf时，我们可以使用一系列快捷键来访问请求参数，会话属性和应用程序属性：

请注意，这些不是*上下文对象*，而是作为变量添加到上下文的映射，因此我们无需访问它们**#**。因此，它们以某种方式作为*名称空间*。

* **param**：用于检索请求参数。**${param.foo}**是一个**String[]**与**foo**请求参数的值，所以**${param.foo[0]}**通常会用于获取第一个值。

/\*

\* ============================================================================

\* See javadoc API for class org.thymeleaf.context.WebRequestParamsVariablesMap

\* ============================================================================

\*/

${param.foo} // Retrieves a String[] with the values of request parameter 'foo'

${param.size()}

${param.isEmpty()}

${param.containsKey('foo')}

...

* **会话**：用于检索会话属性。

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.context.WebSessionVariablesMap

\* ======================================================================

\*/

${session.foo} // Retrieves the session atttribute 'foo'

${session.size()}

${session.isEmpty()}

${session.containsKey('foo')}

...

* **应用程序**：用于检索应用程序/ servlet上下文属性。

/\*

\* =============================================================================

\* See javadoc API for class org.thymeleaf.context.WebServletContextVariablesMap

\* =============================================================================

\*/

${application.foo} // Retrieves the ServletContext atttribute 'foo'

${application.size()}

${application.isEmpty()}

${application.containsKey('foo')}

...

请注意，**不需要指定用于访问请求属性的名称空间**（与*请求参数*相对），因为所有请求属性都会自动作为上下文根中的变量添加到上下文中：

${myRequestAttribute}

***Web上下文对象***

在Web环境中，还可以直接访问以下对象（请注意这些对象，而不是地图/名称空间）：

* **#httpServletRequest**：直接访问**javax.servlet.http.HttpServletRequest**与当前请求关联的对象。

${#httpServletRequest.getAttribute('foo')}

${#httpServletRequest.getParameter('foo')}

${#httpServletRequest.getContextPath()}

${#httpServletRequest.getRequestName()}

...

* **#httpSession**：直接访问**javax.servlet.http.HttpSession**与当前请求关联的对象。

${#httpSession.getAttribute('foo')}

${#httpSession.id}

${#httpSession.lastAccessedTime}

...

***Spring上下文对象***

如果你使用Spring的Thymeleaf，你也可以访问这些对象：

* **#themes**：提供了与Spring **spring:theme**JSP标签相同的功能。

${#themes.code('foo')}

***春天的豆子***

Thymeleaf还允许以Spring EL定义的标准方式访问在Spring应用程序上下文中注册的bean，该标准方法使用语法**@beanName**，例如：

<div th:text="${@authService.getUserName()}">...</div>

***附录B：表达式实用程序对象***

***日期***

* **#dates**：**java.util.Date**对象的实用方法：

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Dates

\* ======================================================================

\*/

/\*

\* Format date with the standard locale format

\* Also works with arrays, lists or sets

\*/

${#dates.format(date)}

${#dates.arrayFormat(datesArray)}

${#dates.listFormat(datesList)}

${#dates.setFormat(datesSet)}

/\*

\* Format date with the ISO8601 format

\* Also works with arrays, lists or sets

\*/

${#dates.formatISO(date)}

${#dates.arrayFormatISO(datesArray)}

${#dates.listFormatISO(datesList)}

${#dates.setFormatISO(datesSet)}

/\*

\* Format date with the specified pattern

\* Also works with arrays, lists or sets

\*/

${#dates.format(date, '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')}

/\*

\* Obtain date properties

\* Also works with arrays, lists or sets

\*/

${#dates.day(date)} // also arrayDay(...), listDay(...), etc.

${#dates.month(date)} // also arrayMonth(...), listMonth(...), etc.

${#dates.monthName(date)} // also arrayMonthName(...), listMonthName(...), etc.

${#dates.monthNameShort(date)} // also arrayMonthNameShort(...), listMonthNameShort(...), etc.

${#dates.year(date)} // also arrayYear(...), listYear(...), etc.

${#dates.dayOfWeek(date)} // also arrayDayOfWeek(...), listDayOfWeek(...), etc.

${#dates.dayOfWeekName(date)} // also arrayDayOfWeekName(...), listDayOfWeekName(...), etc.

${#dates.dayOfWeekNameShort(date)} // also arrayDayOfWeekNameShort(...), listDayOfWeekNameShort(...), etc.

${#dates.hour(date)} // also arrayHour(...), listHour(...), etc.

${#dates.minute(date)} // also arrayMinute(...), listMinute(...), etc.

${#dates.second(date)} // also arraySecond(...), listSecond(...), etc.

${#dates.millisecond(date)} // also arrayMillisecond(...), listMillisecond(...), etc.

/\*

\* Create date (java.util.Date) objects from its components

\*/

${#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,millisecond)}

/\*

\* Create a date (java.util.Date) object for the current date and time

\*/

${#dates.createNow()}

${#dates.createNowForTimeZone()}

/\*

\* Create a date (java.util.Date) object for the current date (time set to 00:00)

\*/

${#dates.createToday()}

${#dates.createTodayForTimeZone()}

***日历***

* **#calendars**：类似于**#dates**，但对于**java.util.Calendar**对象：

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Calendars

\* ======================================================================

\*/

/\*

\* Format calendar with the standard locale format

\* Also works with arrays, lists or sets

\*/

${#calendars.format(cal)}

${#calendars.arrayFormat(calArray)}

${#calendars.listFormat(calList)}

${#calendars.setFormat(calSet)}

/\*

\* Format calendar with the ISO8601 format

\* Also works with arrays, lists or sets

\*/

${#calendars.formatISO(cal)}

${#calendars.arrayFormatISO(calArray)}

${#calendars.listFormatISO(calList)}

${#calendars.setFormatISO(calSet)}

/\*

\* Format calendar with the specified pattern

\* Also works with arrays, lists or sets

\*/

${#calendars.format(cal, 'dd/MMM/yyyy HH:mm')}

${#calendars.arrayFormat(calArray, 'dd/MMM/yyyy HH:mm')}

${#calendars.listFormat(calList, 'dd/MMM/yyyy HH:mm')}

${#calendars.setFormat(calSet, 'dd/MMM/yyyy HH:mm')}

/\*

\* Obtain calendar properties

\* Also works with arrays, lists or sets

\*/

${#calendars.day(date)} // also arrayDay(...), listDay(...), etc.

${#calendars.month(date)} // also arrayMonth(...), listMonth(...), etc.

${#calendars.monthName(date)} // also arrayMonthName(...), listMonthName(...), etc.

${#calendars.monthNameShort(date)} // also arrayMonthNameShort(...), listMonthNameShort(...), etc.

${#calendars.year(date)} // also arrayYear(...), listYear(...), etc.

${#calendars.dayOfWeek(date)} // also arrayDayOfWeek(...), listDayOfWeek(...), etc.

${#calendars.dayOfWeekName(date)} // also arrayDayOfWeekName(...), listDayOfWeekName(...), etc.

${#calendars.dayOfWeekNameShort(date)} // also arrayDayOfWeekNameShort(...), listDayOfWeekNameShort(...), etc.

${#calendars.hour(date)} // also arrayHour(...), listHour(...), etc.

${#calendars.minute(date)} // also arrayMinute(...), listMinute(...), etc.

${#calendars.second(date)} // also arraySecond(...), listSecond(...), etc.

${#calendars.millisecond(date)} // also arrayMillisecond(...), listMillisecond(...), etc.

/\*

\* Create calendar (java.util.Calendar) objects from its components

\*/

${#calendars.create(year,month,day)}

${#calendars.create(year,month,day,hour,minute)}

${#calendars.create(year,month,day,hour,minute,second)}

${#calendars.create(year,month,day,hour,minute,second,millisecond)}

${#calendars.createForTimeZone(year,month,day,timeZone)}

${#calendars.createForTimeZone(year,month,day,hour,minute,timeZone)}

${#calendars.createForTimeZone(year,month,day,hour,minute,second,timeZone)}

${#calendars.createForTimeZone(year,month,day,hour,minute,second,millisecond,timeZone)}

/\*

\* Create a calendar (java.util.Calendar) object for the current date and time

\*/

${#calendars.createNow()}

${#calendars.createNowForTimeZone()}

/\*

\* Create a calendar (java.util.Calendar) object for the current date (time set to 00:00)

\*/

${#calendars.createToday()}

${#calendars.createTodayForTimeZone()}

***数字***

* **#numbers**：数字对象的实用方法：

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Numbers

\* ======================================================================

\*/

/\*

\* ==========================

\* Formatting integer numbers

\* ==========================

\*/

/\*

\* Set minimum integer digits.

\* Also works with arrays, lists or sets

\*/

${#numbers.formatInteger(num,3)}

${#numbers.arrayFormatInteger(numArray,3)}

${#numbers.listFormatInteger(numList,3)}

${#numbers.setFormatInteger(numSet,3)}

/\*

\* Set minimum integer digits and thousands separator:

\* 'POINT', 'COMMA', 'WHITESPACE', 'NONE' or 'DEFAULT' (by locale).

\* Also works with arrays, lists or sets

\*/

${#numbers.formatInteger(num,3,'POINT')}

${#numbers.arrayFormatInteger(numArray,3,'POINT')}

${#numbers.listFormatInteger(numList,3,'POINT')}

${#numbers.setFormatInteger(numSet,3,'POINT')}

/\*

\* ==========================

\* Formatting decimal numbers

\* ==========================

\*/

/\*

\* Set minimum integer digits and (exact) decimal digits.

\* Also works with arrays, lists or sets

\*/

${#numbers.formatDecimal(num,3,2)}

${#numbers.arrayFormatDecimal(numArray,3,2)}

${#numbers.listFormatDecimal(numList,3,2)}

${#numbers.setFormatDecimal(numSet,3,2)}

/\*

\* Set minimum integer digits and (exact) decimal digits, and also decimal separator.

\* Also works with arrays, lists or sets

\*/

${#numbers.formatDecimal(num,3,2,'COMMA')}

${#numbers.arrayFormatDecimal(numArray,3,2,'COMMA')}

${#numbers.listFormatDecimal(numList,3,2,'COMMA')}

${#numbers.setFormatDecimal(numSet,3,2,'COMMA')}

/\*

\* Set minimum integer digits and (exact) decimal digits, and also thousands and

\* decimal separator.

\* Also works with arrays, lists or sets

\*/

${#numbers.formatDecimal(num,3,'POINT',2,'COMMA')}

${#numbers.arrayFormatDecimal(numArray,3,'POINT',2,'COMMA')}

${#numbers.listFormatDecimal(numList,3,'POINT',2,'COMMA')}

${#numbers.setFormatDecimal(numSet,3,'POINT',2,'COMMA')}

/\*

\* ==========================

\* Utility methods

\* ==========================

\*/

/\*

\* Create a sequence (array) of integer numbers going

\* from x to y

\*/

${#numbers.sequence(from,to)}

${#numbers.sequence(from,to,step)}

***字符串***

* **#strings**：**String**对象的实用方法：

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Strings

\* ======================================================================

\*/

/\*

\* Null-safe toString()

\*/

${#strings.toString(obj)} // also array\*, list\* and set\*

/\*

\* Check whether a String is empty (or null). Performs a trim() operation before check

\* Also works with arrays, lists or sets

\*/

${#strings.isEmpty(name)}

${#strings.arrayIsEmpty(nameArr)}

${#strings.listIsEmpty(nameList)}

${#strings.setIsEmpty(nameSet)}

/\*

\* Perform an 'isEmpty()' check on a string and return it if false, defaulting to

\* another specified string if true.

\* Also works with arrays, lists or sets

\*/

${#strings.defaultString(text,default)}

${#strings.arrayDefaultString(textArr,default)}

${#strings.listDefaultString(textList,default)}

${#strings.setDefaultString(textSet,default)}

/\*

\* Check whether a fragment is contained in a String

\* Also works with arrays, lists or sets

\*/

${#strings.contains(name,'ez')} // also array\*, list\* and set\*

${#strings.containsIgnoreCase(name,'ez')} // also array\*, list\* and set\*

/\*

\* Check whether a String starts or ends with a fragment

\* Also works with arrays, lists or sets

\*/

${#strings.startsWith(name,'Don')} // also array\*, list\* and set\*

${#strings.endsWith(name,endingFragment)} // also array\*, list\* and set\*

/\*

\* Substring-related operations

\* Also works with arrays, lists or sets

\*/

${#strings.indexOf(name,frag)} // also array\*, list\* and set\*

${#strings.substring(name,3,5)} // also array\*, list\* and set\*

${#strings.substringAfter(name,prefix)} // also array\*, list\* and set\*

${#strings.substringBefore(name,suffix)} // also array\*, list\* and set\*

${#strings.replace(name,'las','ler')} // also array\*, list\* and set\*

/\*

\* Append and prepend

\* Also works with arrays, lists or sets

\*/

${#strings.prepend(str,prefix)} // also array\*, list\* and set\*

${#strings.append(str,suffix)} // also array\*, list\* and set\*

/\*

\* Change case

\* Also works with arrays, lists or sets

\*/

${#strings.toUpperCase(name)} // also array\*, list\* and set\*

${#strings.toLowerCase(name)} // also array\*, list\* and set\*

/\*

\* Split and join

\*/

${#strings.arrayJoin(namesArray,',')}

${#strings.listJoin(namesList,',')}

${#strings.setJoin(namesSet,',')}

${#strings.arraySplit(namesStr,',')} // returns String[]

${#strings.listSplit(namesStr,',')} // returns List<String>

${#strings.setSplit(namesStr,',')} // returns Set<String>

/\*

\* Trim

\* Also works with arrays, lists or sets

\*/

${#strings.trim(str)} // also array\*, list\* and set\*

/\*

\* Compute length

\* Also works with arrays, lists or sets

\*/

${#strings.length(str)} // also array\*, list\* and set\*

/\*

\* Abbreviate text making it have a maximum size of n. If text is bigger, it

\* will be clipped and finished in "..."

\* Also works with arrays, lists or sets

\*/

${#strings.abbreviate(str,10)} // also array\*, list\* and set\*

/\*

\* Convert the first character to upper-case (and vice-versa)

\*/

${#strings.capitalize(str)} // also array\*, list\* and set\*

${#strings.unCapitalize(str)} // also array\*, list\* and set\*

/\*

\* Convert the first character of every word to upper-case

\*/

${#strings.capitalizeWords(str)} // also array\*, list\* and set\*

${#strings.capitalizeWords(str,delimiters)} // also array\*, list\* and set\*

/\*

\* Escape the string

\*/

${#strings.escapeXml(str)} // also array\*, list\* and set\*

${#strings.escapeJava(str)} // also array\*, list\* and set\*

${#strings.escapeJavaScript(str)} // also array\*, list\* and set\*

${#strings.unescapeJava(str)} // also array\*, list\* and set\*

${#strings.unescapeJavaScript(str)} // also array\*, list\* and set\*

/\*

\* Null-safe comparison and concatenation

\*/

${#strings.equals(first, second)}

${#strings.equalsIgnoreCase(first, second)}

${#strings.concat(values...)}

${#strings.concatReplaceNulls(nullValue, values...)}

/\*

\* Random

\*/

${#strings.randomAlphanumeric(count)}

***对象***

* **#objects**：一般对象的实用方法

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Objects

\* ======================================================================

\*/

/\*

\* Return obj if it is not null, and default otherwise

\* Also works with arrays, lists or sets

\*/

${#objects.nullSafe(obj,default)}

${#objects.arrayNullSafe(objArray,default)}

${#objects.listNullSafe(objList,default)}

${#objects.setNullSafe(objSet,default)}

***布尔***

* **#bools**：用于布尔评估的实用方法

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Bools

\* ======================================================================

\*/

/\*

\* Evaluate a condition in the same way that it would be evaluated in a th:if tag

\* (see conditional evaluation chapter afterwards).

\* Also works with arrays, lists or sets

\*/

${#bools.isTrue(obj)}

${#bools.arrayIsTrue(objArray)}

${#bools.listIsTrue(objList)}

${#bools.setIsTrue(objSet)}

/\*

\* Evaluate with negation

\* Also works with arrays, lists or sets

\*/

${#bools.isFalse(cond)}

${#bools.arrayIsFalse(condArray)}

${#bools.listIsFalse(condList)}

${#bools.setIsFalse(condSet)}

/\*

\* Evaluate and apply AND operator

\* Receive an array, a list or a set as parameter

\*/

${#bools.arrayAnd(condArray)}

${#bools.listAnd(condList)}

${#bools.setAnd(condSet)}

/\*

\* Evaluate and apply OR operator

\* Receive an array, a list or a set as parameter

\*/

${#bools.arrayOr(condArray)}

${#bools.listOr(condList)}

${#bools.setOr(condSet)}

***数组***

* **#arrays**：数组的实用方法

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Arrays

\* ======================================================================

\*/

/\*

\* Converts to array, trying to infer array component class.

\* Note that if resulting array is empty, or if the elements

\* of the target object are not all of the same class,

\* this method will return Object[].

\*/

${#arrays.toArray(object)}

/\*

\* Convert to arrays of the specified component class.

\*/

${#arrays.toStringArray(object)}

${#arrays.toIntegerArray(object)}

${#arrays.toLongArray(object)}

${#arrays.toDoubleArray(object)}

${#arrays.toFloatArray(object)}

${#arrays.toBooleanArray(object)}

/\*

\* Compute length

\*/

${#arrays.length(array)}

/\*

\* Check whether array is empty

\*/

${#arrays.isEmpty(array)}

/\*

\* Check if element or elements are contained in array

\*/

${#arrays.contains(array, element)}

${#arrays.containsAll(array, elements)}

***清单***

* **＃**列表：实用程序列表的方法

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Lists

\* ======================================================================

\*/

/\*

\* Converts to list

\*/

${#lists.toList(object)}

/\*

\* Compute size

\*/

${#lists.size(list)}

/\*

\* Check whether list is empty

\*/

${#lists.isEmpty(list)}

/\*

\* Check if element or elements are contained in list

\*/

${#lists.contains(list, element)}

${#lists.containsAll(list, elements)}

/\*

\* Sort a copy of the given list. The members of the list must implement

\* comparable or you must define a comparator.

\*/

${#lists.sort(list)}

${#lists.sort(list, comparator)}

***集***

* **#sets**：集合的实用方法

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Sets

\* ======================================================================

\*/

/\*

\* Converts to set

\*/

${#sets.toSet(object)}

/\*

\* Compute size

\*/

${#sets.size(set)}

/\*

\* Check whether set is empty

\*/

${#sets.isEmpty(set)}

/\*

\* Check if element or elements are contained in set

\*/

${#sets.contains(set, element)}

${#sets.containsAll(set, elements)}

***地图***

* **#maps**：地图的实用方法

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Maps

\* ======================================================================

\*/

/\*

\* Compute size

\*/

${#maps.size(map)}

/\*

\* Check whether map is empty

\*/

${#maps.isEmpty(map)}

/\*

\* Check if key/s or value/s are contained in maps

\*/

${#maps.containsKey(map, key)}

${#maps.containsAllKeys(map, keys)}

${#maps.containsValue(map, value)}

${#maps.containsAllValues(map, value)}

***骨料***

* **#aggregates**：用于在数组或集合上创建聚合的实用方法

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Aggregates

\* ======================================================================

\*/

/\*

\* Compute sum. Returns null if array or collection is empty

\*/

${#aggregates.sum(array)}

${#aggregates.sum(collection)}

/\*

\* Compute average. Returns null if array or collection is empty

\*/

${#aggregates.avg(array)}

${#aggregates.avg(collection)}

***消息***

* **#messages**：用于在变量表达式中获得外部消息的实用方法，与使用**#{...}**语法所获得的方式相同。

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Messages

\* ======================================================================

\*/

/\*

\* Obtain externalized messages. Can receive a single key, a key plus arguments,

\* or an array/list/set of keys (in which case it will return an array/list/set of

\* externalized messages).

\* If a message is not found, a default message (like '??msgKey??') is returned.

\*/

${#messages.msg('msgKey')}

${#messages.msg('msgKey', param1)}

${#messages.msg('msgKey', param1, param2)}

${#messages.msg('msgKey', param1, param2, param3)}

${#messages.msgWithParams('msgKey', new Object[] {param1, param2, param3, param4})}

${#messages.arrayMsg(messageKeyArray)}

${#messages.listMsg(messageKeyList)}

${#messages.setMsg(messageKeySet)}

/\*

\* Obtain externalized messages or null. Null is returned instead of a default

\* message if a message for the specified key is not found.

\*/

${#messages.msgOrNull('msgKey')}

${#messages.msgOrNull('msgKey', param1)}

${#messages.msgOrNull('msgKey', param1, param2)}

${#messages.msgOrNull('msgKey', param1, param2, param3)}

${#messages.msgOrNullWithParams('msgKey', new Object[] {param1, param2, param3, param4})}

${#messages.arrayMsgOrNull(messageKeyArray)}

${#messages.listMsgOrNull(messageKeyList)}

${#messages.setMsgOrNull(messageKeySet)}

***标识***

* **#ids**：用于处理**id**可能重复的属性的实用方法（例如，作为迭代的结果）。

/\*

\* ======================================================================

\* See javadoc API for class org.thymeleaf.expression.Ids

\* ======================================================================

\*/

/\*

\* Normally used in th:id attributes, for appending a counter to the id attribute value

\* so that it remains unique even when involved in an iteration process.

\*/

${#ids.seq('someId')}

/\*

\* Normally used in th:for attributes in <label> tags, so that these labels can refer to Ids

\* generated by means if the #ids.seq(...) function.

\*

\* Depending on whether the <label> goes before or after the element with the #ids.seq(...)

\* function, the "next" (label goes before "seq") or the "prev" function (label goes after

\* "seq") function should be called.

\*/

${#ids.next('someId')}

${#ids.prev('someId')}

***附录C：DOM选择器语法***

DOM选择器从XPATH，CSS和jQuery中借用语法功能，以提供一种功能强大且易于使用的方式来指定模板片段。

例如，下面的选择器将在标记中的每个位置中选择每个**<div>**类**content**：

<div th:include="mytemplate :: [//div[@class='content']]">...</div>

受XPath启发的基本语法包括：

* **/x** 意味着当前节点的名字x的直接子节点。
* **//x** 意味着任何深度的名字为x的当前节点的子节点。
* **x[@z="v"]** 意味着名称为x的元素和名为z并具有值“v”的属性。
* **x[@z1="v1" and @z2="v2"]** 意味着分别具有名称x和属性z1和z2的值分别为值“v1”和“v2”的元素。
* **x[i]** 意思是名字x在兄弟姐妹中位于数字i的元素。
* **x[@z="v"][i]** 意味着名称为x，属性为z的值为“v”的元素，并且在其同样符合此条件的兄弟中位于数字i中。

但是也可以使用更简洁的语法：

* **x**完全等同于**//x**（**x**在任何深度级别搜索名称或引用的元素）。
* 只要包含参数说明，选择器也可以不带元素名称/引用。所以**[@class='oneclass']**是一个有效的选择器，它查找具有值“oneclass”的类属性的任何元素（标签）。

高级属性选择功能：

* 除了**=**（等于），其他比较运算符也是有效的：（**!=**不等于），**^=**（以...开头）和**$=**（以...结尾）。例如：**x[@class^='section']**意味着具有名称**x**和属性值的元素**class**开头的元素**section**。
* 属性既可以以**@**（XPath风格）开始，也可以不以（jQuery风格）指定。所以**x[z='v']**相当于**x[@z='v']**。
* 多属性修饰符可以通过**and**（XPath样式）连接，也可以通过链接多个修饰符（jQuery样式）来连接。所以**x[@z1='v1' and @z2='v2']**实际上相当于**x[@z1='v1'][@z2='v2']**（也是**x[z1='v1'][z2='v2']**）。

直接*类似jQuery的*选择器：

* **x.oneclass**相当于**x[class='oneclass']**。
* **.oneclass**相当于**[class='oneclass']**。
* **x#oneid**相当于**x[id='oneid']**。
* **#oneid**相当于**[id='oneid']**。
* **x%oneref**意味着节点 - 不只是元素 - 名称x 根据指定的实现匹配参考*oneref***DOMSelector.INodeReferenceChecker**。
* **%oneref**意味着节点 - 不仅仅是元素 - 具有根据指定实现与任何名称匹配的引用*oneref***DOMSelector.INodeReferenceChecker**。注意这实际上相当于只是**oneref**因为可以使用引用而不是元素名称。
* 直接选择器和属性选择器可以混合：**a.external[@href^='https']**。

上面的DOM选择器表达式：

<div th:include="mytemplate :: [//div[@class='content']]">...</div>

可以写成：

<div th:include="mytemplate :: [div.content]">...</div>

***多值类匹配***

DOM选择器将类属性理解为**多值**，因此即使元素具有多个类值，也允许在该属性上应用选择器。

例如，**div[class='two']**将匹配**<div class="one two three" />**

***可选的括号***

片段包含属性的语法将每个片段选择转换为DOM选择，因此**[...]**不需要括号（尽管允许）。

所以下面没有括号，就等于上面括号里的选择器：

<div th:include="mytemplate :: div.content">...</div>

所以，总结一下：

<div th:replace="mytemplate :: myfrag">...</div>

将寻找一个**th:fragment="myfrag"**片段签名。但是，**myfrag**如果它们存在的话，也会查找带有名称的标签（它们不包含在HTML中）。注意不同之处：

<div th:replace="mytemplate :: .myfrag">...</div>

这将实际上寻找任何元素**class="myfrag"**，而不关心**th:fragment**签名。

1. 鉴于XHTML5只是XML / HTML5内容类型的应用程序，我们也可以说Thymeleaf支持XHTML5。[↩](http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#fnref1)
2. 请注意，虽然此模板是有效的XHTML，但我们之前选择了模板模式“XHTML”而不是“VALIDXHTML”。现在，我们只需要验证就可以了 - 但同时我们不希望我们的IDE抱怨太多。