Thymeleaf

教程：使用 *Thymeleaf*

文档版本： 20161119 - 2016年11月19日
项目版本： 3.0.2.RELEASE
项目网站： http://www.thymeleaf.org

1介绍Thymeleaf

1.1什么是Thymeleaf?

Thymeleaf是Web和独立环境，能够处理HTML，XML，JavaScript，CSS和甚至是纯文本的现代服务器端的Java模板引擎。

Thymeleaf的主要目标是提供创建模板的优雅和高维护的方法。为了实现这一目标，它建立在的概念自然模板注入它的逻辑到模板文件的方式，并不被用于作为设计原型影响的模板。这提高了设计的沟通和桥梁的设计和开发团队之间的差距。

尤其是- Thymeleaf也已经从Web标准考虑设计之初**HTML5** -允许您创建充分验证模板，如果这是一个需要你。

1.2什么样的模板可以Thymeleaf过程？

外的开箱，Thymeleaf让你处理6种模板，每个被称为模板方式：

- HTML
- XML
- 文本
- JAVASCRIPT
- CSS
- 生的

有两种标记模板模式（ **HTML** 和 **XML** ），三文本模板模式（ **TEXT** ， **JAVASCRIPT** 和 **CSS** ）和无操作模板模式（ **RAW** ）。

该**HTML**模板模式将允许任何HTML的输入，包括HTML5，HTML 4和XHTML。无验证或良好性检查会执行，模板代码/结构将得到遵守，以输出最大可能的程度。

该**XML**模板模式将允许XML输入。在这种情况下，代码有望得到很好的形成-没有未关闭的标签，不带引号的属性，等等-如果良好性侵犯被发现解析器会抛出异常。注意，没有确认（根据DTD或XML模式）将被执行。

所述**TEXT**模板模式将允许对非标记性质的模板使用特殊的语法。这样的模板例子可能是文本电子邮件或模板文件。需要注意的是HTML或XML模板也可以处理成 **TEXT** ，在这种情况下，他们不会被解析为标记，每个标记，DOCTYPE，评论等，将被视为单纯的文字。

该**JAVASCRIPT**模板模式将允许在Thymeleaf应用程序的JavaScript文件的处理。这意味着能够使用模型数据的JavaScript文件内以同样的方式就可以在HTML文件中进行，但JavaScript的特定的集成，如专门的逃逸或自然的脚本。所述 **JAVASCRIPT** 模板模式被认为是文本模式，因此，使用相同的特殊语法作为 **TEXT** 模板模式。

所述**CSS**模板模式将允许参与一个Thymeleaf应用CSS文件的处理。类似 **JAVASCRIPT** 模式中， **CSS** 模板模式也是一个文本模式和使用从该特殊处理的语法 **TEXT** 模板模式。

该**RAW**模板模式将根本不处理模板。它是指用于被处理插入不变资源（文件，URL响应等）插入模板。例如，以HTML格式的外部，不受控制的资源可以被包括到应用模板，安全地知道任何Thymeleaf代码，这些资源可能包括将不被执行。

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，它可以被称为一个模板引擎架构），它允许您定义和自定义的模板将被处理细节的精细程度的方式。

适用一些逻辑一种标记工件的对象（一个标签，一些文本，注释，或如果模板不标记仅仅占位符）被称为处理器，和一组这些处理器-加上一些额外的工件-是什么一个方言通常由。开箱，Thymeleaf的核心库提供了方言叫做标准方言，应该是足够大多数用户使用。

需要注意的是方言实际上没有处理器和完全由其他种类的工作，但是处理器绝对是最常见的情况。

本教程介绍了标准的方言。你会在下面的网页了解每个属性和语法功能通过这个方言定义的，即使未明确提及。

当然，用户可以创建自己的方言（甚至延长了标准的），如果他们想同时利用图书馆的高级功能来定义自己的处理逻辑。Thymeleaf也可以配置在一个时间使用几种方言。

官方thymeleaf-spring3和thymeleaf-spring4集成包都定义一个方言叫“SpringStandard方言”，其中大部分是一样的标准方言，但与小的修改，以更好地利用Spring框架的一些功能（例如通过使用Spring表达式语言或SpringEL代替OGNL）。所以，如果你是你是不是在浪费你的时间Spring MVC的用户，你在这里学到会使用在你的Spring应用程序，因为几乎一切。

大多数标准方言处理器是属性处理器。这使得浏览器甚至被处理，因为他们会简单地忽略附加属性之前正确显示HTML模板文件。例如，当一个使用JSP标签库可以包括代码通过浏览器不是直接显示的片段，如：

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

.....在Thymeleaf标准方言将使我们能够实现与相同的功能：

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

这不仅会被浏览器正确地显示，但是这也使我们可以（任选地）指定一个值属性（“詹姆斯胡萝卜”，在这种情况下），当原型在浏览器中被静态打开将被显示，并且将由从评估所得的值的被取代 **${user.name}** 的模板的处理过程中。

这有助于你的设计师和开发人员就非常相同的模板文件的工作，并减少改造静态原型到工作的模板文件所需的工作量。要做到这一点的能力是一个称为自然模板。

2好 Thymes虚拟杂货店

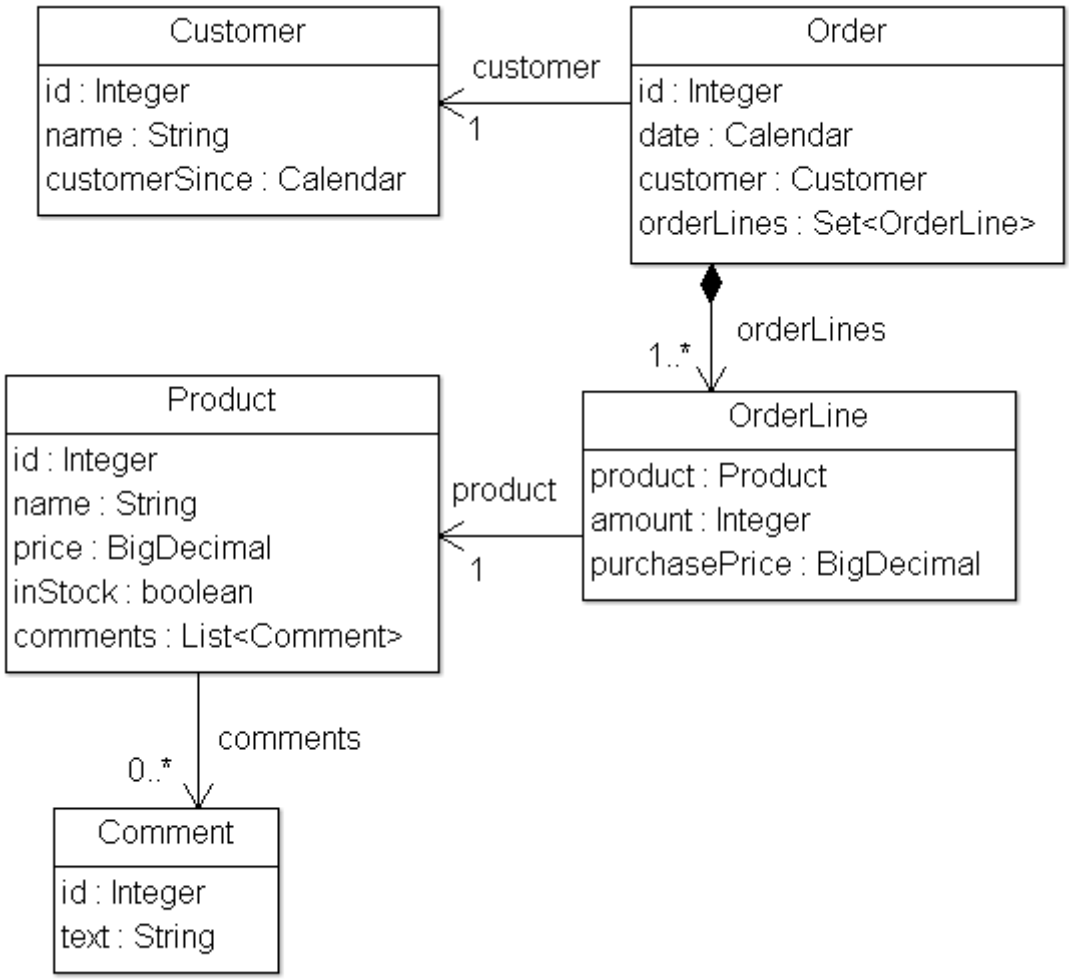
在这个所示的例子，本指南的后续章节的源代码，可以在找到好Thymes虚拟杂货店GitHub的信息库。

2.1网站杂货店

为了更好地解释涉及与Thymeleaf处理模板的概念，本教程将使用演示应用程序，你可以从项目的网站下载。

此应用程序是一个虚构的虚拟超市的网站，并为我们提供了很多场景，展示Thymeleaf的许多功能。

首先，我们需要一个简单的为我们的应用程序模型的实体： **Products** 其中销往 **Customers** 创建 **Orders** 。我们还将管理 **Comments** 有关的 **Products** ：



示例应用模型

我们的应用程序也将有一个非常简单的服务层，通过由 **Service** 含有类似方法的对象：

```
public class ProductService {

    ...

    public List<Product> findAll() {
        return ProductRepository.getInstance().findAll();
    }

    public Product findById(Integer id) {
        return ProductRepository.getInstance().findById(id);
```

```
        }

    }
```

在web层我们的应用程序将有一个过滤器，将委托执行，以根据请求的URL启用Thymeleaf的命令：

```
    private boolean process(HttpServletRequest request, HttpServletResponse response)
            throws ServletException {

        try {

            // This prevents triggering engine executions for resource URLs
            if (request.getRequestURI().startsWith("/css") ||
                    request.getRequestURI().startsWith("/images") ||
                    request.getRequestURI().startsWith("/favicon")) {
                return false;
            }


            /*
             * 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 = this.application.resolveControllerForRequest(request);
            if (controller == null) {
                return false;
            }

            /*
             * Obtain the TemplateEngine instance.
             */
            ITemplateEngine templateEngine = this.application.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) {
            try {
                response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            } catch (final IOException ignored) {
                // Just ignore this
            }
            throw new ServletException(e);
        }

    }
```

这是我们的 **IGTVGController** 界面：

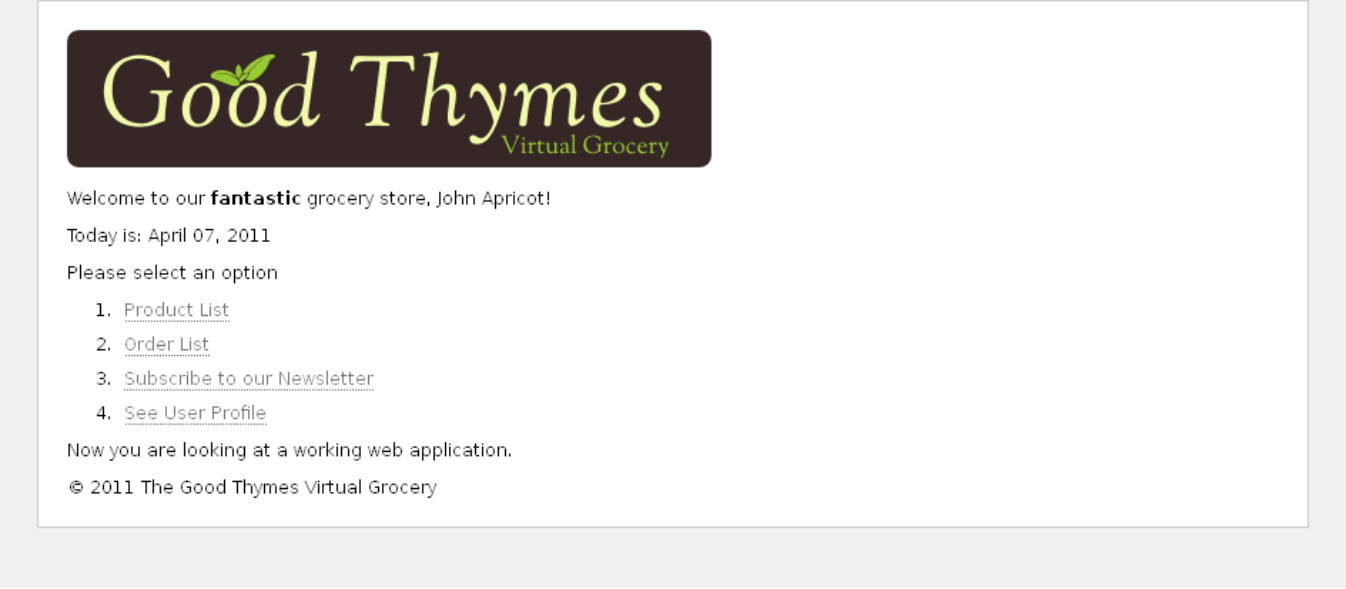
```
    public interface IGTVGController {
```

```
    public void process(
            HttpServletRequest request, HttpServletResponse response,
            ServletContext servletContext, ITemplateEngine templateEngine);

}
```

现在我们所要做的就是创建的实现 **IGTVGController** 界面，检索从使用的服务和处理模板，数据 **ITemplateEngine** 对象。

最后，它看起来就像这样：



示例应用程序主页

但是，首先让我们来看看该模板引擎是如何初始化。

2.2创建和配置模板引擎

该过程（.....）在我们的过滤方法包含这一行：

```
ITemplateEngine templateEngine = this.application.getTemplateEngine();
```

这意味着该*GTVGApplication*类是负责创建和配置在一个Thymeleaf应用的最重要的对象之一的：在 **TemplateEngine** 实例（实施的 **ITemplateEngine** 接口）。

我们的 **org.thymeleaf.TemplateEngine** 对象初始化是这样的：

```
public class GTVGApplication {


    ...
    private static TemplateEngine templateEngine;
    ...


    public GTVGApplication(final ServletContext servletContext) {

        super();

        ServletContextTemplateResolver templateResolver =
                new ServletContextTemplateResolver(servletContext);

        // HTML is the default mode, but we set it anyway for better understanding of code
        templateResolver.setTemplateMode(TemplateMode.HTML);
        // 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(Long.valueOf(3600000L));

        // Cache is set to true by default. Set to false if you want templates to
        // be automatically updated when modified.
        templateResolver.setCacheable(true);

        this.templateEngine = new TemplateEngine();
        this.templateEngine.setTemplateResolver(templateResolver);

        ...

    }

}
```

有配置的方法很多 **TemplateEngine** 对象，但现在的这几行代码将足以让我们对所需的步骤。

模板解析器

让我们先从模板解析器：

```
ServletContextTemplateResolver templateResolver =
        new ServletContextTemplateResolver(servletContext);
```

模板解析器是实现从调用API Thymeleaf的接口的对象 **org.thymeleaf.templateresolver.ITemplateResolver** ：

```
public interface ITemplateResolver {

    ...

    /*
     * Templates are resolved by their name (or content) and also (optionally) their
     * owner template in case we are trying to resolve a fragment for another template.
     * Will return null if template cannot be handled by this template resolver.
     */
    public TemplateResolution resolveTemplate(
            final IEngineConfiguration configuration,
            final String ownerTemplate, final String template,
            final Map<String, Object> templateResolutionAttributes);
}
```

这些对象是负责确定我们的模板将如何被访问的，并且在该GTVG申请
中， **org.thymeleaf.templateresolver.ServletContextTemplateResolver** 我们将要检索我们的模板文件从资源装置*Servlet*上下文：一个应用范围 **javax.servlet.ServletContext** ，存在于每一个Java Web应用程序对象，并从Web应用程序根目录解析资源。

但是，这还不是全部，我们可以说一下模板的解析器，因为我们可以将它的一些配置参数。首先，模板模式：

```
templateResolver.setTemplateMode(TemplateMode.HTML);
```

HTML是默认模板模式 **ServletContextTemplateResolver** ，但它是无论如何建立它，这样我们的代码文件中明确到底是怎么回事很好的做法。

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

的前缀和后缀修改，我们将传递到发动机用于获得要使用的实际资源名称的模板名称。

使用这样的结构，的模板名称“产品/目录”将对应于：

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

任选的时间，一个解析模板可以住在缓存中的量被配置为在模板解析器通过的装置*cacheTTLMs*属性：

```
templateResolver.setCacheTTLMs(3600000L);
```

模板仍然可以从缓存中驱逐达到该TTL之前，如果达到最大缓存大小，这是当前缓存最早的条目。

缓存行为和尺寸可以由用户通过实现被定义 **ICacheManager** 接口或通过修改 **StandardCacheManager** 对象管理默认缓存。

还有很多学习的模板解析器，但现在让我们来看看在创建我们的模板引擎的对象。

模板引擎

模板引擎对象是的实现 **org.thymeleaf.ITemplateEngine** 接口。其中一个实现由Thymeleaf核心提供： **org.thymeleaf.TemplateEngine** ，我们在这里创建它的一个实例：

```
templateEngine = new TemplateEngine();
templateEngine.setTemplateResolver(templateResolver);
```

很简单，不是吗？我们需要的是创建一个实例，模板解析器设置它。

模板解析器是唯一必需的参数 **TemplateEngine** 需求，虽然有很多人稍后将覆盖（消息解析器，缓存大小等）。就目前而言，这都是我们需要的。

我们的模板引擎是现在已经准备好，我们可以开始创建使用Thymeleaf我们的网页。

3使用文本

3.1多语言欢迎

我们的首要任务将是为我们的食品网站创建一个主页。

本页面的第一个版本将是非常简单：只是一个标题和一个值得欢迎的消息。这是我们的 **/WEB-INF/templates/home.html** 文件：

```
<!DOCTYPE html>

<html 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>
```

你会注意到的第一件事是，这个文件是可以通过任何浏览器，因为它不包含任何非HTML标签（浏览器会忽略他们不明白所有的属性，如正确显示HTML5 **th:text** ）。

但你也可能会注意到这个模板是不是一个真正的有效的 HTML5文档，因为我们使用了这些不规范的属性 **th:*** 的形式不被HTML5规范允许的。事实上，我们甚至增加一个 **xmlns:th** 属性对我们 **<html>** 的标签，东西绝对非HTML5上下的：

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

.....有没有影响的模板处理所有，但可以作为一个咒语阻止我们的IDE从抱怨缺少一个命名空间定义的所有这些 **th:*** 属性。

因此，如果我们想使这个模板是什么**HTML5**的有效？易：切换到Thymeleaf的数据属性语法，使用 **data-** 属性名称和连字符（前缀 **-** ）分隔符代替分号（ **:** ）：

```
<!DOCTYPE html>

<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" data-th-href="@{/css/gtvg.css}" />
  </head>

  <body>

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

  </body>

</html>
```

自定义 **data-** 前缀属性由HTML5规范允许的，所以，上面这段代码中，我们的模板将是一个有效的*HTML5*文档。

> 这两个符号是完全等效的，可互换的，但对于代码样本的简单和紧凑的缘故，本教程将使用空间符号（ **th:*** ）。此外，该 **th:*** 表示法是更普遍的，并使其在每Thymeleaf模板模式（ **XML** ， **TEXT** ...），而 **data-** 符号只允许在 **HTML** 模式。

使用日：文本和文本外化

外化文本提取的模板码片段出模板文件的，以便它们可以保持在分开的文件（通常 **.properties** 是文件），并且它们可以与其他语言编写的等效文本容易地更换（这个过程被称为国际或仅仅*I18N*）。文字外在片段通常被称为*"信息"*。

消息总是有标识他们一个键，Thymeleaf允许您指定的文本应与与特定消息 **#{...}** 的语法：

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

我们可以在这里看到其实都是Thymeleaf标准方言两种不同的特点：

- 该 **th:text** 属性，评估其价值的表达，并将结果作为主机标记的主体，有效地取代了"欢迎来到我们的杂货店！"我们在代码中看到的文字。
- 的 **#{home.welcome}** 表达，在规定的标准表达式语法，指示，要由要使用的文本 **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，这是哪里的外向化文本？

在Thymeleaf外向化文本的位置是完全可配置的，这将取决于具体的 **org.thymeleaf.messageresolver.IMessageResolver** 实施被使用。通常情况下，根据一个实现 **.properties** 文件将被使用，但我们可以创建自己的实现方式，如果我们想要的，例如，从数据库中获取的信息。

但是，我们还没有指定我们的初始化过程中模板引擎的消息解析器，这意味着我们的应用程序正在使用标准信息解析器，实现者 **org.thymeleaf.messageresolver.StandardMessageResolver** 。

标准的消息解析器希望找到消息 **/WEB-INF/templates/home.html** 在同一文件夹属性文件，并使用相同的名称作为模板，如：

- **/WEB-INF/templates/home_en.properties** 英文文本。
- **/WEB-INF/templates/home_es.properties** 西班牙语言文本。
- **/WEB-INF/templates/home_pt_BR.properties** 对于葡萄牙语（巴西）语言文本。
- **/WEB-INF/templates/home.properties** 默认文本（如果该区域不匹配）。

让我们看看我们的 **home_es.properties** 文件：

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

这是我们需要作出Thymeleaf处理我们的模板。让我们来创建我们的Home控制器即可。

上下文

为了处理我们的模板，我们将创建一个 **HomeController** 类实现了 **IGTVGController** 我们以前看到的界面：

```
public class HomeController implements IGTVGController {

    public void process(
            final HttpServletRequest request, final HttpServletResponse response,
            final ServletContext servletContext, final ITemplateEngine templateEngine)
            throws Exception {

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

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

```
    }

}
```

我们看到的第一件事情是创建环境。一个Thymeleaf上下文是实现一个对象 **org.thymeleaf.context.IContext** 接口。上下文应该包含所有用于在变量映射模板引擎的执行所需的数据，并且还引用必须用于外部化消息的语言环境。

```
public interface IContext {

    public Locale getLocale();
    public boolean containsVariable(final String name);
    public Set<String> getVariableNames();
    public Object getVariable(final String name);

}
```

有这种接口的一个专门扩展 **org.thymeleaf.context.IWebContext** ，这意味着在基于ServletAPI的web应用（如用SpringMVC）的使用。

```
public interface IWebContext extends IContext {

    public HttpServletRequest getRequest();
    public HttpServletResponse getResponse();
    public HttpSession getSession();
    public ServletContext getServletContext();

}
```

该Thymeleaf核心库提供这些接口的实现：

- **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， **WebContext** 就是我们使用的一个。事实上，我们必须这样做，因为使用的 **ServletContextTemplateResolver** 要求，我们使用的上下文实现 **IWebContext** 。

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

只有四分之三的这四个构造函数的参数是必需的，因为如果没有指定将被用于在系统默认的locale（虽然你不应该让这种事发生在实际应用中）。

有一些专门的表达，我们将能够用来获得从所述请求参数和请求，会话和应用程序的属性 **WebContext** 中的模板。例如：

- **${x}** 将返回一个变量 **x** 存储到Thymeleaf上下文或作为请求属性。
- **${param.x}** 会返回一个请求参数称为 **x** （可能多值）。
- **${session.x}** 会返回一个会话属性叫 **x** 。
- **${application.x}** 会返回一个*servlet*上下文属性叫 **x** 。

执行模板引擎

随着我们的上下文对象准备好了，现在我们可以使用的范围内，并通过它的响应作家，这样的反应可以写入它告诉模板引擎来处理模板（通过其名称）：

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

让我们来看看这个用西班牙语语言环境的结果：

```
<!DOCTYPE html>

<html>

  <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

非转义的文本

我们主页的最简单的版本，现在似乎要准备好，但有些地方是我们有没有想过.....如果我们有这样的消息吗？

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

如果我们执行这个模板像以前一样，我们将获得：

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

这不正是我们所期望的，因为我们的 **<b>** 标签已经逃脱，因此它会显示在浏览器中。

这是默认行为 **th:text** 属性。如果我们希望Thymeleaf尊重我们的HTML标签和无法逃避他们，我们将不得不使用不同的属性： **th:utext** （对于“转义文本”）：

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

这将输出我们的信息，就像我们想要它：

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

使用和显示变量

现在，让我们添加一些更多的内容我们的主页。例如，我们可能想要显示低于我们欢迎邮件的日期，就像这样：

```
Welcome to our fantastic grocery store!

Today is: 12 july 2010
```

首先，我们必须修改我们的控制器，让我们添加日期作为上下文变量：

```
public void process(
            final HttpServletRequest request, final HttpServletResponse response,
            final ServletContext servletContext, final ITemplateEngine templateEngine)
```

```
            throws Exception {

        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());

    }
```

我们增加了一个 **String** 变量叫 **today** 我们的环境，现在我们可以在模板中显示它：

```
<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>
```

正如你可以看到，我们仍然使用 **th:text** 属性作业（这是正确的，因为我们要更换标签的机构），但语法是有点不同，这一次和，而不是一个 **#{...}** 表达式的值，我们用的是 **${...}** 一。这是一个可变表达，并且它包含在称为语言表达式*OGNL*（对象图形导航语言），将在上下文变量映射我们之前谈到被执行。

的 **${today}** 表达只是意味着“得到今天称为变量”，但这些表达式可以是更复杂的（例如 **${user.name}** 对于“获取被叫用户的变量，并且调用其 **getName()** 法”）。

有属性值相当多的可能性：信息，变量表达式...和相当多很多。下一章会告诉我们什么所有这些可能性。

4标准表达式语法

我们将在我们的杂货店虚拟商店发展的一个小破学习Thymeleaf标准方言的最重要的部分之一左右：在Thymeleaf标准表达式语法。

我们已经看到了两种类型的这种语法表达了有效的属性值：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>
```

但也有更多类型的表达式，以及更多有趣的细节，以了解我们已经知道的人。首先，让我们来看看标准的表达功能的简要说明：

- 简单的表达式：
  - 变量表达式： **${...}**
  - 选择变量表达式： ***{...}**
  - 消息表达式： **#{...}**
  - 链接URL表达式： **@{...}**
  - 片段表达式： **~{...}**
- 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**
  - null文本： **null**
  - 文字标记： **one** ， **sometext** ， **main** ，…
- 文本操作：
  - 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）： **-**
- 布尔操作：
  - 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** ）
- 有条件的经营者：
  - 如果 - 则： **(if) ? (then)**
  - 如果 - 则 - 否则： **(if) ? (then) : (else)**
  - 默认： **(value) ?: (defaultvalue)**
- 特殊记号：
  - 无操作： **_**

所有这些特征可以被组合并嵌套：

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

4.1信息

我们已经知道， **#{...}** 消息表达式允许我们链接此：

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

……这样的：

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

但是有我们还没有想到的一个方面：如果消息文本不完全静态的会发生什么？如果，例如，我们的应用程序就知道是谁访问的网站在任何时刻的用户，我们希望通过名字来迎接他们？

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

这意味着我们需要一个参数添加到我们的信息。像这样：

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

参数按规定的 **java.text.MessageFormat** 标准语法，这意味着在这些类的API文档指定可以格式化数字和日期。

为了给指定称为HTTP会话属性，我们的参数的值，并且 **user** ，我们将有：

```
<p th:utext="#{home.welcome(${session.user.name})}">
  Welcome to our grocery store, Sebastian Pepper!
</p>
```

一些参数可以指定用逗号分隔。事实上，该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 **${...}** 的表达式实际上包含在上下文变量在地图上执行OGNL（对象图形导航语言）表达式。

> 有关OGNL语法和功能的详细信息，你应该阅读OGNL语言指南
>
> 在Spring MVC启用OGNL将被替换应用**SpringEL**，但它的语法非常类似于OGNL（实际上，正是同为最常见的情况）的。

从OGNL的语法，我们知道，在表达：

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

……其实等价于：

```
ctx.getVariable("today");
```

但OGNL允许我们创建相当更强大的表现，这是如何：

```
<p th:utext="#{home.welcome(${session.user.name})}">
  Welcome to our grocery store, Sebastian Pepper!
</p>
```

.....通过执行获得用户名：

```
((User) ctx.getVariable("session").get("user")).getName();
```

但getter方法导航只是OGNL的特点之一。让我们来看看一些：

```
/*
 * 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('-')}
```

表达基本对象

当在上下文变量评估OGNL表达式，某些对象提供给表达式更高的灵活性。这些对象（每个OGNL标准）开始的引用 **#** 符号：

- **#ctx** ：上下文对象。
- **#vars**：上下文变量。
- **#locale** ：上下文的语言环境。
- **#request** （仅在web上下文）的 **HttpServletRequest** 对象。
- **#response** （仅在web上下文）的 **HttpServletResponse** 对象。
- **#session** （仅在web上下文）的 **HttpSession** 对象。
- **#servletContext** （仅在web上下文）的 **ServletContext** 对象。

因此，我们可以这样做：

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

你可以阅读这些对象的完整参考附录A。

表达工具对象

除了这些基本对象，Thymeleaf会为我们提供了一套实用的对象，这将有助于我们在表达式执行常见任务。

- **#execInfo**：正在处理有关模板的信息。
- **#messages**：用于获得外部化消息内部变量的表达式，以同样的方式，因为他们将用#{...}语法来获得的方法。
- **#uris**：方法逃避的URL / URI的部分
- **#conversions**：用于执行所述配置的方法转换服务（如果有的话）。
- **#dates**：方法 **java.util.Date** 对象：格式化，成分提取等。
- **#calendars**：类似于 **#dates**，但对于 **java.util.Calendar** 对象。
- **#numbers**：方法格式化数值对象。
- **#strings**：所用方法 **String** 对象：包含，startsWith，预谋/附加等。
- **#objects**：对一般对象的方法。
- **#bools**：布尔评价方法。
- **#arrays**：数组的方法。
- **#lists**：对列表的方法。
- **#sets**：为集的方法。
- **#maps**：对地图的方法。
- **#aggregates**：用于创建对数组或集合聚集方法。
- **#ids**：方法与可能被重复的id属性处理（例如，作为迭代的结果）。

您可以检查每个在这些工具的对象提供哪些功能附录B。

在我们的主页重新格式化日期

现在我们知道这些实用物品，我们可以用它们来改变我们显示了我们的主页日期的方式。代替在我们这样做 **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());
```

.....我们能做的只是这一点：

```
WebContext ctx =
    new WebContext(request, response, servletContext, request.getLocale());
ctx.setVariable("today", Calendar.getInstance());

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

...然后在视图层本身执行日期格式：

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

*4.3*表达式的选择（星号语法）

不仅可变量表达式写为 **${...}**，也可作为 ***{...}**。

还有，虽然一个重要的区别：星号语法上求表达式选定的对象，而不是对整个背景。也就是说，只要是没有选择的对象，美元和星号语法做同样的。

什么是一个选择的对象？使用表达式的结果 **th:object** 属性。让我们用一个我们的用户配置文件（ **userprofile.html** ）页：

```
  <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>
```

这是完全等价于：

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

当然，美元和星号语法可以混：

```
<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>
```

当对象的选择是在地方，所选择的对象也将提供给元表达式作为 **#object** 表达变量：

```
<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>
```

至于说，如果没有对象选择已经被执行，美元和星号语法是等价的。

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

*4.4*链接的网址

由于其重要性，URL是Web应用程序模板一等公民，而*Thymeleaf*标准方言有他们特殊的语法，该 **@** 语法： **@{...}**

有不同类型的网址：

- 绝对网址： **http://www.thymeleaf.org**
- 相对URL，它可以是：
  - 页面相对： **user/login.html**
  - 上下文相关： **/itemdetails?id=3** （服务器将自动添加上下文名称）
  - 相对于服务器的： **~/billing/processInvoice** （允许在同一台服务器的另一个上下文（=应用程序）调用的URL。
  - 相关协议的URL： **//code.jquery.com/jquery-2.0.3.min.js**

这些表达式的实处理和它们转化成网址，就可以输出由的实现做 **org.thymeleaf.linkbuilder.ILinkBuilder** 被登记到该接口 **ITemplateEngine** 被使用的对象。

缺省情况下，一个单一的实施这一接口的注册的类的 **org.thymeleaf.linkbuilder.StandardLinkBuilder** ，这是足以为离线（非网络），并且还幅场景基于Servlet API。其他情况下（如非ServletAPI Web框架集成）可能需要链接Builder界面的具体实现。

让我们用这个新的语法。满足 **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>
```

有些东西这里要注意：

- **th:href** 是一个修改属性：一次处理，将计算出要使用的链接URL并将该值设定为 **href** 所述的属性 **<a>** 标记。
- 我们被允许使用的URL参数表达式（正如你可以看到 **orderId=${o.id}** ）。的URL参数编码所需的操作也将被自动执行。
- 如果需要几个参数，这些都将用逗号分隔： **@{/order/process(execId=${execId},execType='FAST')}**
- 可变模板也允许URL路径： **@{/order/{orderId}/details(orderId=${orderId})}**
- 首先是相对URL **/** （如： **/order/details** ）将应用程序上下文名称自动前缀。
- 如果Cookie未启用或本还不知道，一个 **";jsessionid=..."** 后缀可能被添加到相对URL，从而使会话被保留。这就是所谓的*URL*重写和Thymeleaf允许你在你自己改写的过滤器通过使用插件 **response.encodeURL(...)** 机制和Servlet API为每个URL。
- 该 **th:href** 属性可以让我们（可选）有一个静态的工作 **href** 在我们的模板属性，所以，当为原型的目的直接打开我们的模板链接仍然通过浏览器通航。

这一点与该消息的语法（壳体 **#{...}** ），网址碱也可以是评价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>
```

一种我们的主页菜单

现在我们知道了如何创建链接的URL，关于我们的主页增加一个小菜单，有的在网站其他页面的什么？

```
<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>
```

服务器的根目录相对 *URL*

另外一个语法可以使用，才能在同一台服务器链接到不同的上下文中创建服务器根目录相对（而不是上下文根目录相对）的URL。这些URL会像被指定 **@{~/path/to/something}**

*4.5*片段

片段表达式来表示标记片段和移动它们周围模板的简便方法。这使我们能够复制它们，将它们传递到其他模板作为参数，等等。

最常见的用途使用是片段的插入 **th:insert** 或 **th:replace** （更多关于这些在后面的章节）：

```
<div th:insert="~{commons :: main}">...</div>
```

但是，他们可以在任何地方使用，就像任何其他变量：

```
<div th:with="frag=~{footer :: #main/text()}">
  <p th:insert="${frag}">
</div>
```

在本教程后面有专门模板布局，包括片段表达的更深层次的解释一整节。

*4.6*字面

文字文本

文本文字是单引号之间的规定只是字符串。它们可以包括任何字符，但你应该使用逃脱在他们里面的任何单引号 **\'** 。

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

文字数

数字文字都只是：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"> ...
```

在这个例子中， **== false** 被写入的括号外，因此它是Thymeleaf该照顾它。如果它的括号内写的，这将是OGNL / SpringEL引擎的责任：

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

null文本

该 **null** 文字也可用于：

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

文字标记

数字，布尔和空文字其实都是特定情况下的文字记号。

这些令牌允许标准表达式简化的一点点。他们的工作完全一样的文本文字（ **'...'** ），但他们只允许字母（ **A-Z** 和 **a-z** ），数字（ **0-9** ），括号（ **[** 和 **]** ），点（ **.** ），连字符（ **-** ）和下划线（ **_** ）。因此，没有空格，没有逗号，等等。

从善意的角度？令牌不需要他们周围的任何报价。因此，我们可以这样做：

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

代替：

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

4.7追加文本

文本，无论他们是否是文字或评估变量或表达式消息的结果，可以很容易地使用附加的 **+** 运营商：

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

4.8字面换人

字面取代允许含有变量的值，而不需要附加文字与串的一个简单的格式 **'...' + '...'** 。

这些替代必须用竖线（包围 **|** ），如：

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

这相当于：

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

字面取代可以与其它类型的表达式进行组合：

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

> 只有变量表达式（ **${...}** ）被允许内 **|...|** 的文字替换。没有其他文字（ **'...'** ），布尔/数字令牌，条件表达式等都是。

4.9算术运算

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

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

请注意，这些操作员也可以OGNL变量表达式本身（在此情况下将通过OGNL代替Thymeleaf规范表达引擎执行）的内部施加：

```
<div th:with="isEven=${prodStat.count % 2 == 0}">
```

注意，文本别名一些运营商的存在： **div** （ **/** ） **mod** （ **%** ）。

4.10比较器和平等

在表达式值可与被比较 **>** ， **<** ， **>=** 和 **<=** 符号，并且 **==** 与 **!=** 运营商可以用来检查是否相等（或缺乏）。注意XML规定， **<** 和 **>** 符号不应在属性值被使用，因此它们应当被取代 **<** 和 **>** 。

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

一个更简单的选择可能是使用存在一些运营商的文本别名： **gt** （ **>** ）， （ **lt** ）， **<** （ ）， **ge** （），**>=** （）。另外（）**/**（）。 **le <= not ! eq == neq ne !=**

4.11条件表达式

条件表达式旨在根据评估的条件（它本身是另一种表达）的结果，以评估只有两个表达式中的一个。

让我们来看一个例子片段（引入另一个属性修饰符， **th:class** ）：

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

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

条件表达式也可以使用括号嵌套：

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

否则表达式也可以省略，在这种情况下返回null值，如果条件为假：

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

4.12默认表情（Elvis操作符）

一个默认的表情是一种特殊的条件值没有那么一部分。这相当于猫王运营商目前在如Groovy一些语言，让你指定两个表达式：如果不计算为NULL第一个被使用，但如果这样做，则使用第二个。

让我们来看看它在我们的用户的个人资料页行动：

```
<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，经营者 **?:** ，我们在这里使用它来指定一个名称的默认值（文字值，在这种情况下）只有当评估的结果 ***{age}** 为空。因此，这是等效于：

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

至于条件值，它们可以包含括号内嵌套表达式：

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

4.13无操作令牌

无操作标记由下划线符号表示（ **_** ）。

此令牌背后的想法是指定为表达式所期望的结果是什么都不做，即恰好做仿佛可加工属性（例如 **th:text** ）是不存在的。

在其他的可能性，这允许开发人员使用原型文本作为默认值。例如，而不是：

```
<span th:text="${user.name} ?: 'no user authenticated'">...</span>
```

.....我们可以直接用“无用户身份验证”为原型的文字，这导致代码是从设计的角度都更加简洁和通用的：

```
<span th:text="${user.name} ?: _">no user authenticated</span>
```

4.15数据转换/格式化

Thymeleaf定义了一个双括号的变量（语法 **${...}** ）和选择（ ***{...}** ）的表达式，它允许我们应用的数据转换通过配置的方式转换服务。

它基本上是这样的：

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

注意到有双括号？： **${{...}}** 。该指示Thymeleaf传递的结果 **user.lastAccessDate** 表达了转换服务，并要求它执行格式化操作（一次转化的 **String** 写入结果之前）。

假设 **user.lastAccessDate** 的类型是 **java.util.Calendar** ，如果一个转换服务（实施 **IStandardConversionService** ）已被登记，并包含一个有效的转换为 **Calendar -> String** ，它将被应用。

的默认实现 **IStandardConversionService** （在 **StandardConversionService** 类）只是执行 **.toString()** 上的任何对象转换为 **String** 。有关如何注册自定义的详细信息转换服务的实现，有看关于配置的更多部分。

> 官方thymeleaf-spring3和thymeleaf-spring4集成包透明地集成与Spring自己Thymeleaf的转换服务机制转换服务的基础设施，所以在Spring配置宣称，转换服务和格式化将作出自动提供给 **${{...}}** 和 ***{{...}}** 表达式。

4.14预处理

除了所有这些功能表达式处理，Thymeleaf具有的特征预处理表达式。

预处理是正常的，其允许最终将被执行的表达的修饰前所做的表达式的执行。

预处理的表情都酷似正常的，但出现一个双下划线（如包围 **__${expression}__** ）。

让我们想象一下，我们有一个国际化 **Messages_fr.properties** 包含OGNL表达式调用特定语言的静态方法，如条目：

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

...还有 **Messages_es.properties equivalent**：

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

我们可以创建标记，用于评估一个表达式或其他取决于区域的片段。为此，我们将首先选择表达式（由预处理），然后让Thymeleaf执行：

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

请注意，对于法语语言环境中的预处理步骤将创建下列等价的：

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

预处理字符串 **__** 可以在属性中使用转义 **__** 。

5设置属性值

本章将介绍中，我们可以设置（或修改）的属性值，在我们的标记方式。

5.1设置任何属性的值

说我们的网站发布新闻稿，我们希望我们的用户能够订阅它，所以我们创建了一个 **/WEB-INF/templates/subscribe.html** 与表单模板：

```
<form action="subscribe.html">
  <fieldset>
    <input type="text" name="email" />
    <input type="submit" value="Subscribe!" />
  </fieldset>
</form>
```

与Thymeleaf，这个模板开始了更喜欢比它为Web应用程序模板的静态原型。首先， **action** 在我们的形式属性静态链接到模板文件本身，使得存在对有用URL重写的地方。其次， **value** 在提交按钮属性使得它显示英文文本，但我们希望它被国际化。

然后输入 **th:attr** 属性，它改变它在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!" th:attr="value=#{subscribe.submit}"/>
  </fieldset>
</form>
```

这个概念很简单： **th:attr** 简单地取一个值赋给一个属性的表达式。在创建了相应的控制器和信息文件，处理这个文件将是结果：

```
<form action="/gtvg/subscribe">
  <fieldset>
    <input type="text" name="email" />
    <input type="submit" value="¡Suscribe!"/>
  </fieldset>
</form>
```

除了新的属性值，你也可以看到，applicacion上下文名称已经自动前缀的网址基地 **/gtvg/subscribe** ，在前面的章节解释。

但是，如果我们想要在同一时间设置多个属性？XML规则不允许你在标签两次设置属性，因此 **th:attr** 将采取逗号分隔的任务列表，如：

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

鉴于所需的信息文件，这将输出：

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

5.2设置值的特定属性

现在，你可能会想，是这样的：

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

...是一个非常难看的片片标记。指定属性的值内的分配可能会很实用，但它不是创建模板，如果你有做这一切的时候是最优雅的方式。

Thymeleaf同意和你在一起，这就是为什么 **th:attr** 在模板中很少使用。通常情况下，您将使用其他 **th:*** 属性，其任务是制定具体的标签属性（不就像任何属性 **th:attr** ）。

例如，设置 **value** 属性，使用 **th:value** ：

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

这看起来好多了！让我们尝试做同样的 **action** 在属性 **form** 标签：

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

你还记得那些 **th:href** 我们把我们的 **home.html** 面前？他们正是这种相同的属性：

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

有不少这样的，他们每个人针对特定的HTML5特性的属性：

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:fragment
th:frame	th:frameborder	th:headers
th:height	th:high	th:href
th:hreflang	th:hspace	th:http-equiv
th:icon	th:id	th:inline
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:onabort	th:onafterprint
th:onbeforeprint	th:onbeforeunload	th:onblur
th:oncanplay	th:oncanplaythrough	th:onchange

th:onclick	th:oncontextmenu	th:ondblclick
th:ondrag	th:ondragend	th:ondragenter
th:ondragleave	th:ondragover	th:ondragstart
th:ondrop	th:ondurationchange	th:onemptied
th:onended	th:onerror	th:onfocus
th:onformchange	th:onforminput	th:onhashchange
th:oninput	th:oninvalid	th:onkeydown
th:onkeypress	th:onkeyup	th:onload
th:onloadeddata	th:onloadedmetadata	th:onloadstart
th:onmessage	th:onmousedown	th:onmousemove
th:onmouseout	th:onmouseover	th:onmouseup
th:onmousewheel	th:onoffline	th:ononline
th:onpause	th:onplay	th:onplaying
th:onpopstate	th:onprogress	th:onratechange
th:onreadystatechange	th:onredo	th:onreset
th:onresize	th:onscroll	th:onseeked
th:onseeking	th:onselect	th:onshow
th:onstalled	th:onstorage	th:onsubmit
th:onsuspend	th:ontimeupdate	th:onundo
th:onunload	th:onvolumechange	th:onwaiting
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*在时间设置多个值

有两个被称为比较特殊的属性 **th:alt-title** 和 **th:lang-xmllang** 它可用于同时设置两个属性为相同的值。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GTVG主页，这将使我们能够替代这一点：

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

...或此，这相当于：

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

...有了这个：

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

5.4追加和预先计算

Thymeleaf还提供 **th:attrappend** 和 **th:attrprepend** 属性，其中附加（后缀）或预先准备（前缀）的评估，以现有的属性值的结果。

例如，您可能希望存储添加一个CSS类的名称（未设置，只是增加），以在上下文变量的按钮中的一个，因为要使用的特定CSS类将取决于一些用户做了之前：

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

如果您处理此模板的 **cssStyle** 设置为变量 **"warning"** ，您将获得：

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

还有两个具体的追加属性的标准方言：在 **th:classappend** 和 **th:styleappend** 属性，它们用于添加CSS类或片段式的一个元素，而不会覆盖现有文件：

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

（不要担心 **th:each** 属性。这是一个迭代的属性，我们会谈论它。）

5.5固定值布尔属性

HTML有概念布尔属性，即没有任何价值和一个前场景意味着值是“真”的属性。在XHTML中，这些属性只取1值，这本身。

例如， **checked** ：

```
<input type="checkbox" name="option2" checked /> <!-- HTML -->
<input type="checkbox" name="option1" checked="checked" /> <!-- XHTML -->
```

标准方言包括一些属性，允许您通过评估的条件，因此，如果评估为true，则该属性将被设置为固定值来设置这些属性，而如果结果为假，则该属性将不会被设置：

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

下面的固定值布尔属性的标准方言存在：

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设置任何属性（默认属性处理器）的值

Thymeleaf提供了一个默认的属性处理器，它允许我们设置的值的属性，即使没有具体的 **th:*** 处理器已经为它在标准方言被定义。

因此，像：

```
<span th:whatever="${user.name}">...</span>
```

会导致：

```
<span whatever="John Apricot">...</span>
```

5.7支持HTML5友好的属性和元素名称

另外，也可以使用完全不同的语法来应用处理器到您的模板在一个更HTML5友好的方式。

```
<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}** 语法是编写自定义属性在HTML5中，而无需开发人员使用任何命名空间的名称，如标准的方式 **th:*** 。Thymeleaf使得这个语法自动提供给所有的方言（不仅是标准的）。

还有一种语法来指定自定义标签： **{prefix}-{name}** ，它遵循*W3C*自定义元素规范（较大的一部分*W3C Web*组件规格）。这可以用来，例如，对 **th:block** 元件（或也 **th-block** ），这将在后面的部分进行说明。

重要说明：此语法是除了命名空间 **th:*** 之一，它不会取代它。我们无意在所有弃用名称空间的语法的未来。

6迭代

到目前为止，我们已经建立了一个主页，用户的个人资料页，也为让用户订阅我们的通讯的网页...但我们的产品呢？为此，我们需要一种方法来遍历项目的集合，为了建设我们的产品页面。

6.1迭代基础知识

要显示产品在我们的 **/WEB-INF/templates/product/list.html** 页面，我们将使用一个表。我们的每一个产品都将显示在一行（一 **<tr>** 元），所以我们的模板中，我们需要创建一个模板行 -一个将体现我们要如何显示每个产品-然后指示Thymeleaf重复它，一旦为每个产品。

标准方言为我们提供了整整一个属性： **th:each** 。

使用日：每个

对于我们的产品列表页面，我们将需要检索的服务层的产品列表，并将其添加到模板上下文中的控制器的方法：

```
public void process(
        final HttpServletRequest request, final HttpServletResponse response,
        final ServletContext servletContext, final ITemplateEngine templateEngine)
        throws Exception {

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

    WebContext ctx = new WebContext(request, response, servletContext, request.getLocale());
    ctx.setVariable("prods", allProducts);

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

}
```

然后我们会用 **th:each** 我们的模板来遍历产品列表：

```
<!DOCTYPE html>

<html 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>
```

那 **prod : ${prods}** 您在上面看到的属性值是指“在评估结果的每个元素 **${prods}** ，重复模板的这个片段，使用一个变量称为督促当前元素”。让我们给一个名字的每一个我们所看到的事情：

- 我们将调用 **${prods}** 的迭代表达式或迭代变量。
- 我们将调用 **prod** 的迭代变量或者干脆*ITER*变量。

需要注意的是 **prod** ITER变量的作用域的 **<tr>** 元素，这意味着它可以像内标签 **<td>** 。

可迭代值

的 **java.util.List** 类不是可用于在Thymeleaf迭代onlyvalue。有一个比较完整的被认为是对象的迭代通过 **th:each** 属性：

- 任何执行对象 **java.util.Iterable**
- 任何对象实施 **java.util.Enumeration** 。
- 任何执行对象 **java.util.Iterator** ，其值将被使用，因为它们是由迭代器返回，而不需要在内存中缓存的所有值。
- 任何对象实施 **java.util.Map** 。当迭代地图，ITER变量将是一流的 **java.util.Map.Entry** 。
- 任何数组。
- 任何其他对象将被视为好像它是包含对象本身的单值列表。

*6.2*保持迭代状态

使用时 **th:each** ， Thymeleaf提供保持你的迭代的状态轨迹有用机制：状态变量。

状态变量在中定义的 **th:each** 属性，包含以下数据：

- 当前迭代指数，从0开始这是 **index** 属性。
- 当前迭代指数，从1开始这是 **count** 属性。
- 在迭代变量元素的总量。这是在 **size** 属性。
- 在国际热核实验堆变量对每个迭代。这是在 **current** 属性。
- 无论当前迭代是奇数还是偶数。这些是 **even/odd** 布尔属性。
- 是否当前迭代是第一个。这是 **first** 布尔属性。
- 是否当前迭代是最后一个。这是 **last** 布尔属性。

让我们看看如何与前面的例子中使用它：

```
<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>
```

状态变量（ **iterStat** 在这个例子中）在定义 **th:each** 属性的ITER变量本身，用逗号隔开后面写它的名字。就像ITER变量，状态变量也作用域的代码由标签抱着定义的片段 **th:each** 属性。

让我们来看看我们的处理模板的结果：

```
<!DOCTYPE html>

<html>

  <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>NAME</th>
        <th>PRICE</th>
        <th>IN STOCK</th>
      </tr>
      <tr class="odd">
        <td>Fresh Sweet Basil</td>
        <td>4.99</td>
        <td>yes</td>
      </tr>
      <tr>
        <td>Italian Tomato</td>
        <td>1.25</td>
        <td>no</td>
      </tr>
      <tr class="odd">
        <td>Yellow Bell Pepper</td>
        <td>2.50</td>
        <td>yes</td>
      </tr>
      <tr>
        <td>Old Cheddar</td>
        <td>18.75</td>
        <td>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完美工作，建立 **odd** 只奇数行CSS类。

如果没有明确设置一个状态变量，Thymeleaf将永远后面添加为您创建一个 **Stat** 到迭代变量的名称：

```
<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>
```

6.3通过数据的检索懒优化

有时候，我们可能要优化数据的集合（如从数据库）的检索，使这些藏品只能检索，如果他们真的要被使用。

> 实际上，这一点是可以应用到任何数据片，但考虑到在内存中的集合可能具有大小，检索该旨在进行迭代是此方案的最常见的情况集合。

为了支持这一点，Thymeleaf提供了一个机制，懒洋洋地加载上下文变量。实现了上下文变量 **ILazyContextVariable** 最有可能通过扩展其-界面 **LazyContextVariable** 默认实现-将在正在执行的时刻来解决。例如：

```
context.setVariable(
     "users",
     new LazyContextVariable<List<User>>() {
         @Override
         protected List<User> loadValue() {
             return databaseRepository.findAllUsers();
         }
     });
```

此变量可以没有其的知识被用于*lazyness*，在代码如：

```
<ul>
  <li th:each="u : ${users}" th:text="${u.name}">user name</li>
</ul>
```

但在同一时间，将永远不会被初始化（其 **loadValue()** 方法将永远不会被称为），如果 **condition** 计算结果为 **false** 在代码如：

```
<ul th:if="${condition}">
  <li th:each="u : ${users}" th:text="${u.name}">user name</li>
</ul>
```

7有条件的评价

7.1简单的条件句：“如果”和“除非”

有时你需要模板的片段，如果一定条件满足只出现在结果中。

例如，假设我们希望在我们的产品表，以显示与所存在的每个产品评论的数量的列和，如果有任何意见，在注释的细节页面的链接该产品。

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>
```

相当多的东西，看到这里，让我们专注于重要的一行：

```
<a href="comments.html"
   th:href="@{/product/comments(prodId=${prod.id})}"
   th:if="${not #lists.isEmpty(prod.comments)}">view</a>
```

这将创建一个链接到评论页面（使用URL **/product/comments** 用） **prodId** 设置参数 **id** 的产品，但前提是该产品有任何意见。

让我们来看看结果标记：

```
<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， **th:if** 属性将不仅评估布尔条件。它的功能走一点，除此之外，它会计算指定表达式 **true** 遵循以下规则：

- 如果值不为空：
  - 如果值是一个布尔值，是 **true** 。
  - 如果值是一个数字，是非零
  - 如果值是一个字符和非零
  - 如果值是一个字符串，而不是“假”，“关闭”或“否”
  - 如果值不是一个布尔值，数字，字符或字符串。
- （如果值为null，TH：如果将评估为false）。

此外， **th:if** 有一个逆向属性 **th:unless** ，我们是否能够在前面的例子，而不是使用中已经使用 **not** 了OGNL表达式中：

```
<a href="comments.html"
   th:href="@{/comments(prodId=${prod.id})}"
   th:unless="${#lists.isEmpty(prod.comments)}">view</a>
```

7.2 switch语句

也有有条件使用相当于要显示内容的方式开关中的Java结构： **th:switch** / **th:case** 属性集。

```
<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>
```

注意，只要一 **th:case** 属性作为评价 **true** ，每隔 **th:case** 在同一开关上下文属性作为评价 **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含模板片段

定义和引用片段

在我们的模板，我们会经常要包括从其他模板零件，部件，如页脚，标题，菜单...

为了做到这一点，Thymeleaf需要我们定义这些零件，“碎片”，以便纳入，可以使用进行 **th:fragment** 属性。

说，我们要一个标准的版权页脚添加到我们所有的杂货店页面，所以我们创建了一个 **/WEB-INF/templates/footer.html** 包含此代码文件：

```
<!DOCTYPE html>

<html 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名为片段 **copy** ，我们可以在我们的主页上使用的一身轻松包括 **th:insert** 或 **th:replace** 属性（也 **th:include** ，虽然因为Thymeleaf 3.0不再推荐使用）：

```
<body>

  ...

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

</body>
```

注意， **th:insert** 期望一个片段的表达（ **~{...}** ），这是导致在一个片段的表达。在虽然上面的例子，这是一个不复杂的片段表达时，（， **~{** ） **}** 包围完全是可选的，所以上面的代码将相当于：

```
<body>

  ...

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

</body>
```

片段规范语法

语法片段的表达是非常简单的。有三种不同的格式：

- **"~{templatename::selector}"** 包括来自名为模板应用指定的标记选择器产生的碎片 **templatename** 。请注意， **selector** 可仅仅是一个片段的名字，所以你可以指定为简单的东西 **~{templatename::fragmentname}** 像在 **~{footer :: copy}** 上面。

标记选择语法由底层AttoParser解析库中定义，并且是类似于XPath表达式或CSS选择。请参见附录C获取更多信息。

- **"~{templatename}"** 包括一个名为完整的模板 **templatename** 。

请注意，在使用模板名称 **th:insert / th:replace** 标签将必须由目前正在使用的模板引擎模板解析器解析。

- **~{::selector}"** 或 **"~{this::selector}"** 包括来自同一模板的片段。

都 **templatename** 与 **selector** 在上述实施例可以是全功能的表现形式（甚至条件句！），如：

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

再次注意周围如何 **~{...}** 信封是可选 **th:insert / th:replace** 。

片段可以包含任何 **th:*** 属性。一旦该片段包括在靶模板（一个与这些属性将被评估 **th:insert / th:replace** 属性），并且它们将能够参考在该目标模板中定义的任何上下文变量。

这种方法的片段一大优势是，你可以写在是完全显示的通过浏览器，具有完整的，甚至网页上的碎片有效的标记结构，同时仍可以让Thymeleaf包括他们到其他模板的能力。

引用片段不 *th:fragment*

由于标记选择器的能力，我们可以包括不使用任何片段 **th:fragment** 的属性。它甚至可以从完全没有Thymeleaf知识的不同的应用程序来标记代码：

```
...
<div id="copy-section">
  &copy; 2011 The Good Thymes Virtual Grocery
</div>
...
```

我们可以使用上述简单地通过其引用它的片段 **id** 属性，以类似的方式，以一个CSS选择：

```
<body>

  ...

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

</body>
```

之间的区别 *th:insert* 和 *th:replace*（和 *th:include*）

和之间有什么区别 **th:insert** 和 **th:replace** （而且 **th:include** ，因为3.0不推荐）？

- **th:insert** 是最简单的：它会简单地插入指定的片段作为其主机标记的主体。
- **th:replace** 实际上取代了与指定的片段其主机标签。
- **th:include** 类似 **th:insert** ，但而不是将它插入只插入片段内容这个片段。

因此，一个HTML片段是这样的：

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

.....包括在主三次 ** ** 的标签，就像这样：

```
<body>

  ...

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

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

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

</body>
```

.....将导致：

```
<body>

  ...

  <div>
    <footer>
      &copy; 2011 The Good Thymes Virtual Grocery
    </footer>
  </div>

  <footer>
    &copy; 2011 The Good Thymes Virtual Grocery
  </footer>

  <div>
    &copy; 2011 The Good Thymes Virtual Grocery
  </div>

</body>
```

8.2参数化片段签名

为了创建更函数状模板片段机构，与所定义的片段 **th:fragment** 可以指定一组参数：

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

这需要使用这两个语法之一的从调用片段 **th:insert** 或 **th:replace** ：

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

注意，为了不在最后的选择重要的是：

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

片段局部变量不片段参数

即使碎片没有这样的参数定义如下：

```
<div th:fragment="frag">
    ...
</div>
```

我们可以使用上面指定打电话给他们（和只有第二个），第二个语法：

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

这将相当于的组合 **th:replace** 和 **th:with**：

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

请注意本地变量的片段的本说明书-不管其是否具有一个参数签名或不-不会导致上下文中之前其执行排空。碎片将仍然能够访问每一个上下文变量处于调用模板中使用像他们目前是。

日：断言在模板断言

该 **th:assert** 属性可以指定一个逗号分隔的，应进行评估的表达式列表，并为每一个评价产生真正的，抛出一个异常如果不是。

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

这是在方便在片段签名验证参数：

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

*8.3*灵活的布局：超越了单纯的插入片段

由于片段的表达，我们可以指定不属于文字，数字，bean对象.....而是标记片段的片段参数。

这使我们的方式使得它们可以创建我们片段富集与标记从主叫模板来，产生了非常灵活的模板布局机制。

注意使用的 **title** 和 **links** 在下面的片段变量：

```
<head th:fragment="common_header(title,links)">

  <title th:replace="${title}">The awesome application</title>

  <!-- Common styles and scripts -->
  <link rel="stylesheet" type="text/css" media="all" th:href="@{/css/awesomeapp.css}">
  <link rel="shortcut icon" th:href="@{/images/favicon.ico}">
  <script type="text/javascript" th:src="@{/sh/scripts/codebase.js}"></script>

  <!--/* Per-page placeholder for additional links */-->
  <th:block th:replace="${links}" />

</head>
```

现在，我们可以把这个片段，如：

```
...
<head th:replace="base :: common_header(~{::title},~{::link})">

  <title>Awesome - Main</title>

  <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
  <link rel="stylesheet" th:href="@{/themes/smoothness/jquery-ui.css}">

</head>
...
```

……，结果会用实际 **<title>** 并 **<link>** 从我们的呼吁模板标签的值 **title** 和 **links** 变量，导致我们的片段插入时被定制：

```
...
<head>

  <title>Awesome - Main</title>

  <!-- Common styles and scripts -->
  <link rel="stylesheet" type="text/css" media="all" href="/awe/css/awesomeapp.css">
  <link rel="shortcut icon" href="/awe/images/favicon.ico">
  <script type="text/javascript" src="/awe/sh/scripts/codebase.js"></script>

  <link rel="stylesheet" href="/awe/css/bootstrap.min.css">
  <link rel="stylesheet" href="/awe/themes/smoothness/jquery-ui.css">

</head>
...
```

使用空片段

一个特殊的片段表达时，空的片段（ **~{}** ），可用于指定没有标记。使用上面的例子：

```
<head th:replace="base :: common_header(~{::title},~{})">

  <title>Awesome - Main</title>

</head>
...
```

注意如何片段的（第二个参数 **links** ）被设定为空的片段，因此，没有什么是用于写入 **<th:block th:replace="${links}" />** 块：

```
...
<head>

  <title>Awesome - Main</title>

  <!-- Common styles and scripts -->
  <link rel="stylesheet" type="text/css" media="all" href="/awe/css/awesomeapp.css">
  <link rel="shortcut icon" href="/awe/images/favicon.ico">
  <script type="text/javascript" src="/awe/sh/scripts/codebase.js"></script>

</head>
...
```

使用无操作令牌

无操作也可以作为参数传递给一个片段，如果我们只是想让我们的片段使用其当前的标记作为默认值。再次，使用 **common_header** 例如：

```
...
<head th:replace="base :: common_header(_,~{::link})">

  <title>Awesome - Main</title>

  <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
  <link rel="stylesheet" th:href="@{/themes/smoothness/jquery-ui.css}">

</head>
...
```

怎么看 **title** 参数（的第一个参数 **common_header** 片段）设置为无操作（ **_** ），这会导致无法在所有（执行的片段，这部分 **title** = 无操作）：

```
  <title th:replace="${title}">The awesome application</title>
```

这样的结果是：

```
...
<head>

  <title>The awesome application</title>

  <!-- Common styles and scripts -->
  <link rel="stylesheet" type="text/css" media="all" href="/awe/css/awesomeapp.css">
  <link rel="shortcut icon" href="/awe/images/favicon.ico">
  <script type="text/javascript" src="/awe/sh/scripts/codebase.js"></script>

  <link rel="stylesheet" href="/awe/css/bootstrap.min.css">
  <link rel="stylesheet" href="/awe/themes/smoothness/jquery-ui.css">

</head>
...
```

片段的高级条件插入

双方的可用性*emtpy*片段和空操作标记允许我们以一个非常简单而优雅的方式进行片段插入条件。

例如，我们可以为了做到这一点，以我们的插入 **common :: adminhead** 片段只有当用户是管理员，如果不插入任何内容（emtpy片段）：

```
...
<div th:insert="${user.isAdmin()} ? ~{common :: adminhead} : ~{}">...</div>
...
```

此外，我们可以使用无操作标记，以便插入片段只有在满足指定的条件，但没有留下修改的标记，如果条件不满足：

```
...
<div th:insert="${user.isAdmin()} ? ~{common :: adminhead} : _">
    Welcome [[${user.name}]], click <a th:href="@{/support}">here</a> for help-desk support.
</div>
...
```

此外，如果我们已经配置了模板解析器来检查它是否存在模板资源-其手段 **checkExistence** 标志-我们可以使用片段本身作为一个条件存在默认的操作：

```
...
<!-- The body of the <div> will be used if the "common :: salutation" fragment  -->
<!-- does not exist (or is empty).                                               -->
```

```
<div th:insert="~{common :: salutation} ?: _">
    Welcome [[${user.name}]], click <a th:href="@{/support}">here</a> for help-desk support.
</div>
...
```

8.4拆卸模板片段

回到示例应用程序，让我们重新审视我们的产品清单模板的最新版本：

```
<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>
```

这段代码就好了作为模板，而是作为一个静态页面（在直接而不Thymeleaf处理它浏览器中打开）不会是一个不错的原型。

为什么？因为，虽然完全由浏览器显示的，该表只有一个行，此行具有模拟数据。为原型，它根本就不会显得不够现实.....我们应该有一个以上的产品，我们需要更多的行。

因此，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>
```

好了，现在我们有三个，一个原型肯定更好。但是.....会发生什么，当我们与Thymeleaf处理它？：

```
<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！好，当然它们是：迭代仅施加到第一行，所以没有理由Thymeleaf应该除去其他两个。

我们需要一种方法模板处理过程中去除这两个行。让我们使用 **th:remove** 属性上的第二和第三个 **<tr>** 标签：

```
  <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>
```

处理完毕后，一切都将重新看看，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>
```

而这是什么 **all** 的属性值，是什么意思？ **th:remove** 可以表现在五个不同的方式，这取决于它的值：

- **all**： 同时删除包含标签及其所有子项。
- **body**： 不要删除含有标记，但删除其所有的孩子。
- **tag**： 删除包含标记，但不要删除它的孩子。
- **all-but-first**： 删除包含标记的所有儿童，除了第一个。
- **none**： 没做什么。该值是动态评价非常有用。

有什么可以说 **all-but-first** 的价值是有益的？这将让我们节省一些 **th:remove="all"** 原型时：

```
  <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** 属性可以采取任何*Thymeleaf*标准表达，因为它返回允许的字符串值中的一个，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>
```

还注意到， **th:remove** 认为 **null** 一个同义词 **none** ，所以下面的工作原理与上述相同的例子：

```
<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， **null** 将被返回，因此没有去除将被执行。

9局部变量

Thymeleaf要求局部变量了为模板的特异性片段定义，并且仅适用于片段内评估的变量。

我们已经看到一个例子是 **prod** 在我们的产品列表页面ITER变量：

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

这 **prod** 变量将只在的范围可 **<tr>** 标记。特别：

- 这将是可用于任何其他 **th:*** 在该标签用更少的执行属性的优先级高于 **th:each** （这意味着他们将之后执行 **th:each** ）。
- 这将是可用于任何的子元素 **<tr>** 标签，例如任何 **<td>** 的元素。

Thymeleaf提供你一个方法，而不迭代声明局部变量，使用 **th:with** 属性，其语法就是这样的属性值分配：

```
<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，并加入到变量映射从上下文来，因此，它可用于评估与上下文中声明的任何其它变量一起，但仅在含有的边界 ** ** 标记。

您可以使用通常的多重分配的语法在同一时间定义几个变量：

```
<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** 属性允许重新使用相同的属性定义的变量：

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

让我们用这个在我们的杂货店的主页！还记得我们输出格式化的日期写的代码？

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

那么，如果我们想要一个 **"dd MMMM yyyy"** 真正依赖本地化？例如，我们可能需要将以下信息添加到我们的 **home_en.properties** ：

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

...和等效之一我们 **home_es.properties** ：

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

现在，让我们使用 **th:with** ，以获得本地化的日期格式转换成一个变量，然后在我们使用 **th:text** 表达式：

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

这是干净和简单。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：优先级？我们还没有谈过这个问题呢！好了，不用担心，因为这是下一个章节是关于什么。

*10*属性优先

当你写一个以上的会发生什么事 **th:*** 在同一个标记属性？例如：

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

我们预期 **th:each** 属性之前执行 **th:text** ，使我们得到我们想要的结果，但鉴于对HTML / XML标准，不给任何样的意义在其中一个标签属性的排列顺序，一个优先机构不得不在自己的属性，以确保按预期这将工作建立。

因此，所有Thymeleaf属性定义一个数字优先级，它建立在它们在标签中执行的顺序。这个顺序是：

订购	特征	属性
1	碎片杂质	th:insert th:replace
2	片段迭代	th:each
3	有条件的评价	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	具体属性修改	th:value th:href th:src ...
7	文本（标签体修改）	th:text th:utext
8	片段规范	th:fragment
9	片段去除	th:remove

这个优先级机制意味着，如果该属性位置反转的上述迭代片段会给完全相同的结果（虽然是略少可读）：

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

11.评论和块

11.1。标准HTML / XML注释

标准HTML / XML注释 **<!-- ... -->** 可以在任何地方Thymeleaf的模板使用。这些意见中的任何内容不会被Thymeleaf进行处理，并会逐字复制到结果：

```
<!-- User info follows -->
<div th:text="${...}">
  ...
</div>
```

11.2。Thymeleaf分析器级注释块

解析器级注释块是将从模板简单地取出时Thymeleaf分析它的代码。他们是这样的：

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

Thymeleaf将除去一切之间 **<!--/*** 和 ***/-->** ，所以这些注释块也可以被用于显示当模板是静态开放代码，知道当Thymeleaf处理它，将被删除：

```
<!--/*-->
  <div>
     you can see me only before Thymeleaf processes me!
  </div>
<!--*/-->
```

这可能来非常方便的原型表有很多 **<tr>** 的，例如：

```
<table>
   <tr th:each="x : ${xs}">
     ...
   </tr>
   <!--/*-->
   <tr>
     ...
   </tr>
   <tr>
     ...
   </tr>
   <!--*/-->
</table>
```

11.3。Thymeleaf仅原型注释块

Thymeleaf允许标记是注释特殊注释块的定义时，模板是静态打开（如为原型），但在执行模板时考虑了Thymeleaf正常的标记。

```
<span>hello!</span>
<!--/*/
  <div th:text="${...}">
    ...
  </div>
```

```
/*/-->
<span>goodbye!</span>
```

Thymeleaf的解析系统会直接删除 **<!--/*/** 和 **/*/-->** 标记，而不是它的内容，这将留给因此注释。因此，在执行模板时，Thymeleaf会真正看到这一点：

```
<span>hello!</span>

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

<span>goodbye!</span>
```

与分析器级注释块，这个功能是方言无关。

11.4。合成***th:block***标签

Thymeleaf的唯一元素处理器（不是属性）包含在标准的方言是 **th:block** 。

th:block 仅仅是一种属性的容器，它允许开发人员模板指定任何属性他们想要的。Thymeleaf将执行这些属性，然后简单地使该块，而不是它的内容，消失。

因此它可能是有用的，例如，创建需要一个以上迭代表时 **<tr>** 的每个元素：

```
<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>
```

尤其是有用的，只有原型的注释块结合使用时：

```
<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>
```

请注意，此解决方案是如何使模板是有效的HTML（无需加禁止 ** ** 内部块 **<table>** ），仍然工程确定在浏览器中打开时静态作为原型！

12内联

12.1表达内联

虽然标准的方言使我们能够做几乎使用标签属性的一切，也有我们可能更喜欢写作的表达直接进入我们的HTML文本的情况。例如，我们可以比较喜欢写这个：

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

...而不是这样的：

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

之间的表达式 **[[...]]** 或者 **[(...)]** 被认为是内联表现在Thymeleaf，并在他们里面我们可以使用任何一种表现，这也将是有效 **th:text** 或 **th:utext** 属性。

需要注意的是，虽然 **[[...]]** 对应于 **th:text** （即结果将是*HTML*转义）， **[(...)]** 对应 **th:utext** 且不会执行任何HTML转义。因此，与可变诸如 **msg = 'This is great!'** 给定该片段，：

```
<p>The message is "[(${msg})]"</p>
```

其结果将有那些 **** 标签转义，所以：

```
<p>The message is "This is <b>great!</b>"</p>
```

而如果像逃脱：

```
<p>The message is "[[${msg}]]"</p>
```

其结果将是HTML转义：

```
<p>The message is "This is &lt;b&gt;great!&lt;/b&gt;"</p>
```

请注意，文本内联是活跃在默认情况下，在我们每一个标记标签的机构-而不是标签本身-，所以我们没有什么需要做的来启用它。

内联*VS*自然模板

如果您来自其他模板引擎在输出文本的方式是常态，您可能会问：我们为什么不从一开始就这样做呢？它比所有的代码少 **th:text** 的属性！

好了，要小心那里，因为虽然你可能会发现内联挺有意思的，你应该永远记住，内联表达式将被逐字在你的HTML文件，当您打开静态显示他们，所以你可能会无法使用它们作为设计原型了！

如何与一个浏览器会静态显示我们的代码片断不使用内联的区别...

```
Hello, Sebastian!
```

.....并用它...

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

...在设计实用性方面很清楚。

禁用内联

这种机制虽然可以被禁用，因为有可能实际上是在的场合，我们都希望输出 **[[...]]** 或 **[(...)]** 序列，而无需其内容被处理为一个表达式。为此，我们将使用 **th:inline="none"** ：

```
<p th:inline="none">A double array looks like this: [[1, 2, 3], [4, 5]]!</p>
```

这将导致：

```
<p>A double array looks like this: [[1, 2, 3], [4, 5]]!</p>
```

12.2文本内联

文本内联是非常相似的表达内联我们刚才看到的能力，但它实际上增加了更多的力量。它必须具有明确启用 **th:inline="text"** 。

文本内联不仅可以让我们使用相同的内联表达式我们刚才看到，但实际上处理标记机构，如果他们在处理模板 **TEXT** 模板方式，这使我们能够进行基于文本的模板逻辑（不仅输出表达式）。

我们将看到更多关于这在对下一章文本模板模式。

12.3 JavaScript的内联

JavaScript的内联允许一个更好的整合JavaScript代码 **<script>** 模板在被处理的块 **HTML** 模板模式。

与文本内联，这其实就相当于好像他们是在模板中处理脚本内容 **JAVASCRIPT** 模板方式，因此所有的力量文本模板模式（见下一章）将在眼前。然而，在本节中，我们将重点放在我们如何使用它加入我们Thymeleaf表达式的输出到我们的JavaScript块。

这种模式必须使用明确启用 **th:inline="javascript"** ：

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

这将导致：

```
<script th:inline="javascript">
    ...
    var username = "Sebastian \"Fruity\" Applejuice";
    ...
</script>
```

在上面的代码中要注意两个重要的事情：

首先，了JavaScript内联不仅将输出所需的文本，还用引号括起来和JavaScript，逃脱它的内容，这样就表达结果作为输出格式良好的**JavaScript**文字。

其次，这种情况的发生，因为我们输出 **${session.user.name}** 表达式作为逃脱使用双括号表达式，
即： **[[${session.user.name}]]** 。相反，如果我们使用转义，如：

```
<script th:inline="javascript">
    ...
```

```
    var username = [(${session.user.name})];
    ...
</script>
```

其结果将如下所示：

```
<script th:inline="javascript">
    ...
    var username = Sebastian "Fruity" Applejuice;
    ...
</script>
```

.....这是畸形的JavaScript代码。但输出转义的东西可能是我们所需要的，如果我们通过附加内嵌表达式来建设我们的脚本的一部分，所以它的好，手头有这个工具。

*JavaScript*的自然模板

上面提到的智能 JavaScript的内联机制的远不止仅仅应用特定的JavaScript转义和输出表达式的结果是有效的文字更进一步。

例如，我们可以包装在诸如JavaScript注释我们（逃脱）内联表达式：

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

而Thymeleaf会忽略一切，我们已经写了评论之后和分号之前（在这种情况下 **`'Gertrud Kiwifruit'`** ），所以执行这个看起来几乎完全当我们不使用包装的意见一样的结果：

```
<script th:inline="javascript">
    ...
    var username = "Sebastian \"Fruity\" Applejuice";
    ...
</script>
```

但在原始模板代码中的另一个仔细一看：

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

请注意这是如何有效的**JavaScript**代码。而当你在一个静态的方式打开模板文件（没有在服务器上执行它），它会完全执行。

所以，我们这里有什么是没有办法做到的**JavaScript**自然模板！

先进的内嵌评价和*JavaScript*系列化

一个重要的事情要注意关于JavaScript的内联的是，这个表达式求值很聪明，不仅限于字符串。Thymeleaf会在正确的JavaScript语法编写以下类型的对象：

- 字符串
- 数字
- 布尔值
- 阵列
- 集合

- 地图
- 豆类（带对象的*getter*和*setter*方法的方法）

例如，如果我们有以下代码：

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

这 **${session.user}** 表达式会一个 **User** 对象，Thymeleaf会正确地将其转换为JavaScript语法：

```
<script th:inline="javascript">
    ...
    var user = {"age":null,"firstName":"John","lastName":"Apricot",
                "name":"John Apricot","nationality":"Antarctica"};
    ...
</script>
```

该JavaScript序列做的方法是通过所述的实施方案的装置 **org.thymeleaf.standard.serializer.IStandardJavaScriptSerializer** 的接口，从而可以在所述的实例被配置 **StandardDialect** 在模板引擎使用感。

这样做的默认实现JS的序列化机制将寻找杰克逊库在classpath中，如果存在，将使用它。如果没有，它将应用覆盖的大多数情况的需要，并产生类似的结果（但是较少柔性）一个内置的序列化机制。

*12.4 CSS*内联

Thymeleaf还允许使用CSS中的内联的 **<style>** 标签，如：

```
<style th:inline="css">
  ...
</style>
```

例如，假设我们有两个变量设置为两个不同 **String** 的值：

```
classname = 'main elems'
align = 'center'
```

我们可以使用它们就像：

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

其结果将是：

```
<style th:inline="css">
    .main\ elems {
      text-align: center;
    }
</style>
```

注意，如何CSS内联也负有一定的智能，就像JavaScript的。具体而言，通过表情输出逃脱这样的表达式 **[[${classname}]]** 将被转义为**CSS**标识符。这就是为什么我们 **classname = 'main elems'** 已经变成 **main\ elems** 在上面的代码片段。

高级功能：*CSS*自然模板等

在等效的方式来为JavaScript之前什么解释，CSS内联也允许我们的 **<style>** 标记来作为工作静态和动态，即**CSS**天然模板通过在评论包装内联的表达式的装置。看到：

```
<style th:inline="css">
    .main\ elems {
      text-align: /*[[${align}]]*/ left;
    }
</style>
```

13文本模板模式

13.1文本语法

在Thymeleaf的三个模板模式被认为是文字： **TEXT** ， **JAVASCRIPT** 和 **CSS** 。这种区分它们从标记模板模式： **HTML** 和 **XML** 。

之间的主要区别文本模板模式和标记的人是在一个文本模板没有标记成在属性的形式插入逻辑，所以我们必须依靠其他的机制。

第一，最基本的这些机制是内联，我们已经详细介绍了前面的章节。内联语法是最简单的方式在文本模板模式表达式的输出结果，所以这是一个文本的电子邮件是非常有效的模板。

```
Dear [(${name})],

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

Sincerely,
  The Reporter.
```

即使没有标记，上面的例子是，可以在被执行一个完整和有效的Thymeleaf模板 **TEXT** 模板模式。

但为了有比单纯的更复杂的逻辑输出表达式，我们需要一个新的基于非标记语法：

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

其实这是浓缩的更详细的版本：

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

请注意，此新语法如何根据被声明为元素（即加工的标签） **[#element ...]** 来代替 **<element ...>** 。元素就像打开 **[#element ...]** 和关闭的像 **[/element]** ，和独立标签可以通过最小化与开放的元素声明 **/** 中几乎等同于XML标签的方式： **[#element ... /]** 。

标准方言只包含这些元素中的一个的处理器：在已知的 **th:block** ，但我们可以在我们的方言延伸此并创建以通常的方式的新元素。另外， **th:block** 元件（ **[#th:block ...] ... [/th:block]** ）被允许缩写为空字符串（ **[# ...] ... [/]** ），因此，上述块是实际上等同于：

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

并给予 **[# th:utext="${item}" /]** 相当于一个内联的转义表达，我们可以只使用它才能有更少的代码。因此，我们结束了我们上面看到的代码的第一个片段：

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

需要注意的是文本的语法要求全元素平衡（无未闭合的标签），并引用属性 -它更XML的款式比HTML样式。

让我们看一个更完整的例子 **TEXT** 模板，一个纯文本电子邮件模板：

```
Dear [(${customer.name})],

This is the list of our products:

[# th:each="prod : ${products}"]
   - [(${prod.name})]. Price: [(${prod.price})] EUR/kg
[/]

Thanks,
  The Thymeleaf Shop
```

执行后，这样的结果可能是这样的：

```
Dear Mary Ann Blueberry,

This is the list of our products:

   - Apricots. Price: 1.12 EUR/kg
   - Bananas. Price: 1.78 EUR/kg
   - Apples. Price: 0.85 EUR/kg
   - Watermelon. Price: 1.91 EUR/kg

Thanks,
  The Thymeleaf Shop
```

和在另一实例中 **JAVASCRIPT** 模板模式，一个 **greeter.js** 文件，我们处理为文本模板，其导致我们从我们的HTML页面调用。注意，这是不一个 **<script>** 在HTML模板块，而是一个 **.js** 文件被处理作为关于它自己的模板：

```
var greeter = function() {

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

    [# th:each="salut : ${salutations}"]
      alert([[${salut}]] + " " + username);
    [/]

};
```

执行后，这样的结果可能是这样的：

```
var greeter = function() {

    var username = "Bertrand \"Crunchy\" Pear";

      alert("Hello" + " " + username);
      alert("Ol\u00E1" + " " + username);
      alert("Hola" + " " + username);

};
```

转义元素属性

为了避免与可能在其他模式进行处理的模板部分相互作用（例如 **text** -mode内侧内嵌 **HTML** 模板），Thymeleaf 3.0允许在它的元素的属性的文本语法转换的。所以：

- 在属性 **TEXT** 模板模式将是*HTML*的转义。
- 在属性 **JAVASCRIPT** 模板模式将是*JavaScript*的转义。
- 在属性 **CSS** 模板方式将*CSS*转义。

因此，这将是一个完全确定 **TEXT** -模式模板（注意 **>** ）：

```
[# th:if="${120&lt;user.age}"]
   Congratulations!
[/]
```

当然，这的 **<** 会使一个没有意义的真正的文字模板，但它是一个好主意，如果我们处理了一个HTML模板 **th:inline="text"** 包含上面的代码块，我们希望确保我们的浏览器不采取 **<user.age** 针对的名字一个开放的标签，当静态打开该文件作为原型。

13.2扩展

一个这种语法的优点在于，它仅仅是作为作为可扩展标记之一。开发商仍然可以定义自己的方言与习俗元素和属性，应用前缀他们（可选），然后在文本模板模式下使用：

```
[#myorg:dosomething myorg:importantattr="211"]some text[/myorg:dosomething]
```

13.3文本仅原型的注释块：添加代码

在 **JAVASCRIPT** 和 **CSS** 模板模式（不适用于 **TEXT** ），允许包括一个特殊的注释语法之间的代码 **/*[+...+]*/** ，这样Thymeleaf将处理模板时自动取消注释这样的代码：

```
var x = 23;

/*[+

var msg  = "This is a working application";

+]*/

var f = function() {
    ...
```

将执行如下：

```
var x = 23;

var msg  = "This is a working application";

var f = function() {
...
```

您可以包括这些评论里表达，他们将进行评估：

```
var x = 23;

/*[+

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

+]*/

var f = function() {
...
```

13.4文本分析器级注释块：移除代码

以类似于只有原型注释块的一种方式，所有三个文本模板模式（ **TEXT** ， **JAVASCRIPT** 和 **CSS** ），使其能指示Thymeleaf特殊之间移除代码 **/*[- */** 和 **/* -]*/** 标记，例如：

```
var x = 23;

/*[- */

var msg  = "This is shown only when executed statically!";

/* -]*/

var f = function() {
...
```

或者这一点，在 **TEXT** 模式：

```
...
/*[- Note the user is obtained from the session, which must exist -]*/
Welcome [(${session.user.name})]!
...
```

*13.5*天然*JavaScript*和*CSS*模板

正如在前面的章节看到，JavaScript和CSS内联提供的可能性，包括内部的JavaScript内联表达式/ CSS的意见，如：

```
...
var username = /*[[${session.user.name}]]*/ "Sebastian Lychee";
...
```

.....这是有效的JavaScript，而一旦执行可能看起来像：

```
...
var username = "John Apricot";
...
```

同样的伎俩封闭的注释中内联表达式其实也可以用于整个文本模式语法：

```
/*[# th:if="${user.admin}"]*/
   alert('Welcome admin');
/*[/]*/
```

在代码警报上述将被显示时，模板打开静态 - 因为它是100％有效的JavaScript - ，以及当如果用户是管理员运行的模板。它相当于：

```
[# th:if="${user.admin}"]
   alert('Welcome admin');
[/]
```

.....这实际上是其最初版本模板解析过程中转换的代码。

但是请注意，在评论包围件不干净，他们住在（向右直到行 **;** 被发现）为内联的输出表达式做。这种行为仅保留给内联输出表达式。

所以Thymeleaf 3.0允许开发复杂的**JavaScript**脚本和**CSS**样式表中的天然模板 的形式，无论是作为一个有效的原型和作为工作模板。

14有些多页我们的杂货店

现在我们知道了很多关于使用Thymeleaf，我们可以添加一些新的页面，我们的网站，订单管理。

请注意，我们将集中于HTML代码，但你可以看看捆绑的源代码，如果你想看到相应的控制器。

14.1订单列表

让我们通过创建一个订单列表页面，请 **/WEB-INF/templates/order/list.html** ：

```
<!DOCTYPE html>

<html 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>
```

这里没有什么是应该让我们感到惊讶，除了OGNL的魔力这一点点：

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

什么，做，对于每个订单（ **OrderLine** 在订单对象），乘以其 **purchasePrice** 和 **amount** 性质（通过调用相应 **getPurchasePrice()** 和 **getAmount()** 方法）并将结果返回到数字的列表，后来由聚集 **#aggregates.sum(...)** 功能，以获得该命令总价钱。

你一定要爱OGNL的力量。

14.2订单详细信息

现在的订单详细信息页面，在此我们将作出大量使用星号语法：

```
<!DOCTYPE html>

<html 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>
```

没有多少真正的新这里，除了这嵌套对象的选择：

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

  ...

  <div th:object="*{customer}">
    <p><b>Name:</b> <span th:text="*{name}">Frederic Tomato</span></p>
    ...
  </div>

  ...
</body>
```

...这使得这 **`*{name}`** 等同于：

```
<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

对于我们的好Thymes虚拟杂货，我们选择了一个 **ITemplateResolver** 被称为实施 **ServletContextTemplateResolver** ，使我们获得模板作为从servlet上下文资源。

除了让我们通过实施创建自己的模板解析的能力 **ITemplateResolver**，Thymeleaf包括其它三种实现开箱即用：

- **org.thymeleaf.templateresolver.ClassLoaderTemplateResolver** 从而解决了模板，类加载器的资源，如：

```
return Thread.currentThread().getContextClassLoader().getResourceAsStream(template);
```

- **org.thymeleaf.templateresolver.FileTemplateResolver** ，这样就解决模板作为从文件系统，如文件：

```
return new FileInputStream(new File(template));
```

- **org.thymeleaf.templateresolver.UrlTemplateResolver** 从而解决了模板，网址（甚至非本地的），如：

```
return (new URL(template)).openStream();
```

- **org.thymeleaf.templateresolver.StringTemplateResolver** ，这直接解析模板作为 **String** 被指定为 **template** （或模板名称，而在这种情况下，显然是比仅仅名得多）：

```
return new StringReader(templateName);
```

所有的预先捆绑实现 **ITemplateResolver** 允许同一组配置参数，其中包括：

- 前缀和后缀（如已经看到）：

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

- 模板别名允许使用模板名称不直接对应于文件名。如果这两个后缀/前缀和别名存在，别名将前缀/后缀之前应用：

```
templateResolver.addTemplateAlias("adminHome","profiles/admin/home");
templateResolver.setTemplateAliases(aliasesMap);
```

- 阅读模板时要应用的编码：

```
templateResolver.setEncoding("UTF-8");
```

- 默认模板模式和图案用于限定特定的模板其它模式：

```
// Default is TemplateMode.XHTML
templateResolver.setTemplateMode("HTML5");
templateResolver.getXhtmlTemplateModePatternSpec().addPattern("*.xhtml");
```

- 默认模式，模板缓存和模式的具体定义模板是否缓存与否：

```
// 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发源于此模板解析器。如果没有设置，删除从缓存中的项目的唯一方法是LRU（缓存最大尺寸超过和条目是最老的）。

```
// Default is no TTL (only LRU would remove entries)
templateResolver.setCacheTTLMs(60000L);
```

所述Thymeleaf + Spring集成封装提供 **SpringResourceTemplateResolver** 它使用的所有弹簧基础设施访问和应用中读取资源实现，并且这是在启用Spring的应用的建议的执行情况。

链接模板解析器

此外，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的分辨率来建立一个订单这样，如果第一个是不能够解决模板，第二个被要求，等：

```
ClassLoaderTemplateResolver classLoaderTemplateResolver = new ClassLoaderTemplateResolver();
classLoaderTemplateResolver.setOrder(Integer.valueOf(1));

ServletContextTemplateResolver servletContextTemplateResolver =
        new ServletContextTemplateResolver(servletContext);
servletContextTemplateResolver.setOrder(Integer.valueOf(2));

templateEngine.addTemplateResolver(classLoaderTemplateResolver);
templateEngine.addTemplateResolver(servletContextTemplateResolver);
```

当施加若干模板解析器，建议为每个模板解析器指定模式，以便Thymeleaf可以快速抛弃那些并不意味着解析模板的模板解析器，提高性能。这样做并不是必需的，但一个建议：

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

如果这些解析模式没有规定的，我们将依托各的具体功能 **ITemplateResolver** 我们使用的实现。注意，并非所有的实现可能能够确定解决前一模板的存在，并因此总是可以考虑一个模板作为解析和打破分辨率链（未允许其他解析器检查同一模板），但是随后不能阅读真正的资源。

所有 **ITemplateResolver** 附带核心Thymeleaf实现包括一种机制，使我们能够使解析器真正检查如果资源考虑之前存在解析。它是 **checkExistence** 标志，它的工作原理是：

```
ClassLoaderTemplateResolver classLoaderTemplateResolver = new ClassLoaderTemplateResolver();
classLoaderTemplateResolver.setOrder(Integer.valueOf(1));
classLoaderTempalteResolver.setCheckExistence(true);
```

这 **checkExistence** 标志强制解析执行实际检查过程中消退期资源存在（如果让存在确认返回false链以下解析器调用）。虽然这可能在任何情况下音质好，在大多数情况下，这将意味着该资源本身（一次用于检查存在，另一个时间读它）双重访问，并且可以是在某些情况下的性能问题，例如远程基于URL模板资源-可能无论如何通过使用模板缓存的获得主要是减轻潜在性能问题（在这种情况下，模板将只解决他们被访问的第一次）。

15.2解析器留言

我们没有明确指定消息解析器实现我们的杂货店应用程序，因为它是以前解释的那样，这意味着正在使用的执行是一个 **org.thymeleaf.messageresolver.StandardMessageResolver** 对象。

StandardMessageResolver 是标准实现的 **IMessageResolver** 接口，但是如果我们想要的，适合我们的应用程序的特定需求，我们可以创造我们自己的。

> 该Thymeleaf + Spring集成包在默认情况下提供一个 **IMessageResolver** 它使用检索外向的消息，通过使用标准的Spring方式实现 **MessageSource** 在Spring应用程序上下文中声明豆。

标准信息解析器

那么，如何 **StandardMessageResolver** 寻找在一个特定的模板所要求的消息？

如果模板名称是 **home** ，它位于 **/WEB-INF/templates/home.html** ，并且请求的语言环境是 **gl_ES** 那么这个解析器将寻找消息在下列文件中，顺序如下：

- **/WEB-INF/templates/home_gl_ES.properties**
- **/WEB-INF/templates/home_gl.properties**
- **/WEB-INF/templates/home.properties**

请参阅的Javadoc文档 **StandardMessageResolver** 类的完整信息解决机制如何运作的更多细节。

配置消息解析器

如果我们想添加一个消息解析器（或更多）的模板引擎？简单：

```
// For setting only one
templateEngine.setMessageResolver(messageResolver);

// For setting more than one
templateEngine.addMessageResolver(messageResolver);
```

我们为什么要拥有一个以上的消息解析器？出于同样的原因为模板解析器：消息解析器是有序的，如果第一个无法解析特定的消息，第二个将被要求，则第三等

15.3转换服务

的转换服务，使我们的手段来执行数据转换和格式化操作双括号语法（ **${{...}}** ）实际上是标准方言的特征，而不是Thymeleaf模板引擎本身。

因此，配置的方式是通过设置在我们的自定义实现 **IStandardConversionService** 界面直接进入实例 **StandardDialect** 正在被配置到模板引擎。喜欢：

```
IStandardConversionService customConversionService = ...

StandardDialect dialect = new StandardDialect();
dialect.setConversionService(customConversionService);

templateEngine.setDialect(dialect);
```

需要注意的是thymeleaf-spring3和thymeleaf-spring4包包含的 **SpringStandardDialect** ，而这个话已经自带预配置的实现 **IStandardConversionService** ，集成Spring自己的转换服务的基础设施，Thymeleaf。

*15.4*日志

Thymeleaf支付相当多的关注日志记录，并总是试图通过它的日志接口，提供有用的信息的最大数量。

使用的日志库是 **slf4j,** 这实际上充当取其记录执行我们可能希望在我们的应用程序中使用（例如，桥接器 **log4j** ）。

Thymeleaf班会记录 **TRACE** ， **DEBUG** 并 **INFO** -level信息，这取决于我们希望的详细程度，而除了一般的记录它会使用与TemplateEngine类，我们可以为不同的目的分别配置相关的三个特殊的记录器：

- **org.thymeleaf.TemplateEngine.CONFIG** 初始化期间将输出详述的库的结构。
- **org.thymeleaf.TemplateEngine.TIMER** 将要处理每个模板所花费的时间量输出信息（基准有用！）
- **org.thymeleaf.TemplateEngine.cache** 是一组记录仪的有关缓存的输出特定信息的前缀。虽然缓存记录程序的名称是由用户配置的，因此可以改变，默认情况下它们是：
    - **org.thymeleaf.TemplateEngine.cache.TEMPLATE_CACHE**
    - **org.thymeleaf.TemplateEngine.cache.EXPRESSION_CACHE**

为Thymeleaf的日志基础设施的示例配置，使用 **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工作多亏了一套分析器 - 用于标记和文字 - 即分析模板导入事件序列（开放标签，文本，近距标签，注释等）和一个系列的处理器 - 每种类型的行为之一，需要被应用于 - 用于修改，以便通过原始模板与我们的数据相结合，以创建我们所期望的结果的模板解析事件序列。

它还包括 - 默认情况下 - 的缓存中存储分析模板; 读和处理它们之前解析模板文件产生的事件的顺序。在Web应用程序时，这是非常有用的，并建立在以下概念：

- 输入/输出几乎总是任何应用程序的最慢的部分。在内存中处理比较起来非常快。
- 克隆现有内存事件序列总是比读一个模板文件，解析它并为它创造一个新的事件序列快得多。
- Web应用程序通常只有几十模板。
- 模板文件是小到中等大小，以及应用程序运行时，他们不会被修改。

这一切都导致了想法，在Web应用程序缓存中最常用的模板，而无需浪费大量的内存是可行的，而且，这将节省大量的时间，将输入/输出操作将花费一小部分文件，事实上，永远不会改变。

以及我们如何利用这个缓存的控制权？首先，我们才了解到，我们可以启用或模板解析器禁用它，甚至作用只在特定的模板：

```
// Default is true
templateResolver.setCacheable(false);
templateResolver.getCacheablePatternSpec().addPattern("/users/*");
```

此外，我们可以通过建立我们自己修改其配置缓存管理对象，它可能是默认的一个实例 **StandardCacheManager** 实现：

```
// Default is 50
StandardCacheManager cacheManager = new StandardCacheManager();
cacheManager.setTemplateCacheMaxSize(100);
...
templateEngine.setCacheManager(cacheManager);
```

请参阅的javadoc的API **org.thymeleaf.cache.StandardCacheManager** 有关配置缓存的详细信息。

条目可以从模板缓存手动删除：

```
// Clear the cache completely
templateEngine.clearTemplateCache();

// Clear a specific template from the cache
templateEngine.clearTemplateCacheFor("/users/userList");
```

17解耦模板逻辑

17.1解耦逻辑：概念

到目前为止，我们已经工作了我们的杂货店与做模板通常的方式，用逻辑被插入到我们的属性的表单模板。

但Thymeleaf也让我们彻底脱钩，从它的逻辑模板标记，允许创建完全逻辑较少标记模板 在 **HTML** 和 **XML** 模板模式。

的主要思想是，模板逻辑将在一个单独的被定义的逻辑文件（更确切地一个逻辑资源，因为它并不需要是一个文件）。默认情况下，逻辑资源将是一个额外的文件，住在同一个地方（例如文件夹）模板文件，名称相同但 **.th.xml** 扩展：

```
/templates
+->/home.html
+->/home.th.xml
```

因此， **home.html** 文件可以完全逻辑少。它可能是这样的：

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

绝对没有Thymeleaf代码那里。这是没有Thymeleaf或模板知识设计者可能已经创建，编辑和/或理解的模板文件。或者通过一些外部系统没有Thymeleaf提供HTML片段挂钩的。

现在，让我们将这种 **home.html** 通过创建额外我们的模板到一个模板Thymeleaf **home.th.xml** 像这样的文件：

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

在这里，我们可以看到很多的 **<attr>** 标签里面的一个 **thlogic** 块。这些 **<attr>** 标签进行属性注入由他们来选择原始模板节点 **sel** 属性，其中包含Thymeleaf 标记选择（实际上*AttoParser*标记选择器）。

另请注意， **<attr>** 标签可以嵌套，使他们的选择是追加。那 **sel="/tr[0]"** 上面，例如，将被处理为 **sel="#usersTable/tr[0]"** 。并为用户名选择 **<td>** 将被处理为 **sel="#usersTable/tr[0]//td.username"** 。

所以一旦合并，上面看到这两个文件将是相同的：

```
<!DOCTYPE html>
<html>
```

```
    <body>
      <table id="usersTable" th:remove="all-but-first">
        <tr th:each="user : ${users}">
          <td class="username" th:text="${user.name}">Jeremy Grapefruit</td>
          <td class="usertype" th:text="#{|user.type.${user.type}|}">Normal User</td>
        </tr>
        <tr>
          <td class="username">Alice Watermelon</td>
          <td class="usertype">Administrator</td>
        </tr>
      </table>
    </body>
  </html>
```

这看起来比较熟悉，而且的确是少冗长不是创建两个单独的文件。但优势解耦模板是我们可以给我们的模板完全独立Thymeleaf，因此从设计的角度更好的可维护性。

当然，有些合同仍需要设计师或开发者之间-比如一个事实，即用户 **<table>** 将需要一个 **id="usersTable"** -，但在许多情况下的纯HTML模板将设计和开发团队之间更好的沟通神器。

*17.2*配置模板分离

启用解耦模板

去耦逻辑将不被预期为默认每个模板。相反，配置的模板解析器（的实现 **ITemplateResolver** ），就需要特别纪念他们解决作为模板，使用分离的逻辑。

除了 **StringTemplateResolver** （不允许解耦逻辑），所有其它外的所述盒实现 **ITemplateResolver** 将提供一个称为标记 **useDecoupledLogic** ，将标记由该分解为可能具有单独的资源的逻辑生活的全部或部分解决了所有的模板：

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

混合连接和脱离逻辑

解耦模板的逻辑，当启用时，是不是必需的。当启用时，这意味着该发动机将寻找含有去耦逻辑，解析和如果存在与原始模板合并它的资源。如果分离逻辑资源不存在任何错误将被抛出。

另外，在同一个模板中，我们可以混合两者耦合和解耦加入一些Thymeleaf在原始模板文件属性，但在离开其他的单独的去耦逻辑文件逻辑，例如。造成这种情况的最常见的情况是使用新的（在V3.0） **th:ref** 属性。

*17.3*个：*ref*属性

th:ref 仅一个标记属性。它不执行任何从处理的角度来看，只是当模板被处理消失，但它的用途在于它作为一个事实标记参考，也就是说，它可以通过名称从解析标记选择器就像一个标记名称或一个片段（ **th:fragment** ）。

因此，如果我们有一个像选择：

```
<attr sel="whatever" .../>
```

这将匹配：

- 任何 **<whatever>** 标记。
- 用任何标签 **th:fragment="whatever"** 属性。

- 用任何标签 **th:ref="whatever"** 属性。

什么是优势 **th:ref** 对，例如，使用纯的HTML **id** 属性？仅仅是一个事实，即我们可能不希望添加这么多 **id** 和 **class** 属性，我们的标记作为逻辑锚，这最终可能会污染我们的产量。

而在同样的意义，什么是缺点 **th:ref** ？ 嗯，很明显，我们会加入一个有点Thymeleaf逻辑（“逻辑”），以我们的模板。

请注意，这适用性 **th:ref** 属性并不只适用于去耦逻辑模板文件：它在其他类型的场景一样，喜欢在片段的表达式（ **~{...}** ）。

的解耦模板*17.4*对性能的影响

的影响是非常小的。当已解析模板标记使用解耦的逻辑和它没有被缓存，模板逻辑资源将被首先解决，解析并加工成的内存中的指令的secuence：基本上是一个属性列表被注入到每个标记选择器。

但是，这是唯一的额外步骤必要的，因为，在此之后，真正的模板会被解析，虽然它解析这些属性将被注入在即时解析器本身，这要归功于先进的功能，在AttoParser节点选择。所以，解析的节点将走出解析器，如果他们有他们的原文模板文件注入属性。

这样做的最大好处？当模板被配置为缓存，这将被缓存已经含有所注入的属性。因此，使用的开销解耦模板可缓存的模板，一旦被缓存，将是绝对为零。

*17.5*号决议解耦逻辑

顺便Thymeleaf解析对应于各模板解耦的逻辑资源是由用户配置的。它是由一个扩展点，所确定的 **org.thymeleaf.templateparser.markup.decoupled.IDecoupledTemplateLogicResolver** 一个，为此，默认实现提供： **StandardDecoupledTemplateLogicResolver** 。

这是什么标准执行呢？

- 首先，它应用 **prefix** 和 **suffix** 对基本名称的模板资源（通过其手段获得的 **ITemplateResource#getBaseName()** 方法）。前缀和后缀，可以配置，默认情况下，前缀将是空和后缀会 **.th.xml** 。
- 其次，它要求的模板资源来解决相对资源其方式与计算的名称 **ITemplateResource#relative(String relativeLocation)** 的方法。

的具体实施 **IDecoupledTemplateLogicResolver** 中使用可以在可配置 **TemplateEngine** 容易：

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

18附录A：表达基本对象

某些对象和可变的地图将随时为被调用。让我们来看看他们：

基本对象

- **#ctx**：上下文对象。的实现 **org.thymeleaf.context.IContext** 还是 **org.thymeleaf.context.IWebContext** 取决于我们的环境（单机或网络）。

注 **#vars** 和 **#root** 对于相同的对象synomyns，但使用 **#ctx** 建议。

```
/*
 * ======================================================================
 * See javadoc API for class org.thymeleaf.context.IContext
 * ======================================================================
 */

${#ctx.locale}
${#ctx.variableNames}

/*
 * ======================================================================
 * See javadoc API for class org.thymeleaf.context.IWebContext
 * ======================================================================
 */

${#ctx.request}
${#ctx.response}
${#ctx.session}
${#ctx.servletContext}
```

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

```
${#locale}
```

Web环境的名称空间的请求/会话属性，等等。

当在网络环境中使用Thymeleaf，我们可以使用一系列的快捷方式的访问请求参数，会话属性和应用属性：

> 注意，这些上下文对象，但添加到上下文作为变量的地图，所以我们访问它们却不需要 **#** 。在某种程度上，他们作为命名空间。

- 参数：用于检索请求参数。 **${param.foo}** 是一个 **String[]** 与值 **foo** 请求参数，所以 **${param.foo[0]}** 通常将用于获取第一个值。

```
/*
 * ======================================================================
 * 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')}
...
```

- 会议：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：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')}
...
```

注意有没有必要指定一个命名空间访问请求的属性 （而不是请求参数），因为所有申请属性会自动添加到上下文作为上下文根变量：

```
${myRequestAttribute}
```

*Web*上下文对象

内部网络环境也有直接访问以下对象（注意，这些都是对象，而不是图/命名空间）：

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

```
${#request.getAttribute('foo')}
${#request.getParameter('foo')}
${#request.getContextPath()}
${#request.getRequestName()}
...
```

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

```
${#session.getAttribute('foo')}
${#session.id}
${#session.lastAccessedTime}
...
```

- **#servletContext**：直接访问 **javax.servlet.ServletContext** 与当前请求关联的对象。

```
${#servletContext.getAttribute('foo')}
${#servletContext.contextPath}
...
```

19附录B：表达式实用对象

执行信息

- **#execInfo**：提供有关模板有用的信息表达对象Thymeleaf标准表达式中正在处理中。

```
/*
 * ======================================================================
 * See javadoc API for class org.thymeleaf.expression.ExecutionInfo
 * ======================================================================
 */

/*
 * Return the name and mode of the 'leaf' template. This means the template
 * from where the events being processed were parsed. So if this piece of
 * code is not in the root template "A" but on a fragment being inserted
 * into "A" from another template called "B", this will return "B" as a
 * name, and B's mode as template mode.
 */
${#execInfo.templateName}
${#execInfo.templateMode}

/*
 * Return the name and mode of the 'root' template. This means the template
 * that the template engine was originally asked to process. So if this
 * piece of code is not in the root template "A" but on a fragment being
 * inserted into "A" from another template called "B", this will still
 * return "A" and A's template mode.
 */
${#execInfo.processedTemplateName}
${#execInfo.processedTemplateMode}

/*
 * Return the stacks (actually, List<String> or List<TemplateMode>) of
 * templates being processed. The first element will be the
 * 'processedTemplate' (the root one), the last one will be the 'leaf'
 * template, and in the middle all the fragments inserted in nested
 * manner to reach the leaf from the root will appear.
 */
${#execInfo.templateNames}
${#execInfo.templateModes}

/*
 * Return the stack of templates being processed similarly (and in the
 * same order) to 'templateNames' and 'templateModes', but returning
 * a List<TemplateData> with the full template metadata.
 */
${#execInfo.templateStack}
```

消息

- **#messages**：用于获得外部化消息内部变量的表达式，以同样的方式，因为他们将使用得到实用方法 **#{...}** 的语法。

```
/*
 * ======================================================================
 * 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)}
```

的***URI /***网址

- **#uris**：Thymeleaf标准表达式中执行URI / URL操作工具的对象（ESP逃避/非转义）。

```
/*
 * ============================================================================
 * See javadoc API for class org.thymeleaf.expression.Uris
 * ============================================================================
 */

/*
 * Escape/Unescape as a URI/URL path
 */
${#uris.escapePath(uri)}
${#uris.escapePath(uri, encoding)}
${#uris.unescapePath(uri)}
${#uris.unescapePath(uri, encoding)}

/*
 * Escape/Unescape as a URI/URL path segment (between '/' symbols)
 */
${#uris.escapePathSegment(uri)}
${#uris.escapePathSegment(uri, encoding)}
${#uris.unescapePathSegment(uri)}
${#uris.unescapePathSegment(uri, encoding)}

/*
 * Escape/Unescape as a Fragment Identifier (#frag)
 */
${#uris.escapeFragmentId(uri)}
${#uris.escapeFragmentId(uri, encoding)}
${#uris.unescapeFragmentId(uri)}
${#uris.unescapeFragmentId(uri, encoding)}

/*
 * Escape/Unescape as a Query Parameter (?var=value)
 */
${#uris.escapeQueryParam(uri)}
${#uris.escapeQueryParam(uri, encoding)}
```

```
${#uris.unescapeQueryParam(uri)}
${#uris.unescapeQueryParam(uri, encoding)}
```

转换

- **#conversions**：实用工具，允许你在执行转换服务在模板的任何一点：

```
/*
 * =====================================================================
 * See javadoc API for class org.thymeleaf.expression.Conversions
 * =====================================================================
 */

/*
 * Execute the desired conversion of the 'object' value into the
 * specified class.
 */
${#conversions.convert(object, 'java.util.TimeZone')}
${#conversions.convert(object, targetClass)}
```

日期

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

```
/*
 * =====================================================================
 * 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**：对多个对象的实用方法：

```
/*
 * ======================================================================
 * 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')}


    /*
     * ======================
     * Formatting currencies
     * ======================
     */

    ${#numbers.formatCurrency(num)}
    ${#numbers.arrayFormatCurrency(numArray)}
    ${#numbers.listFormatCurrency(numList)}
    ${#numbers.setFormatCurrency(numSet)}


    /*
     * ======================
```

```
 * Formatting percentages
 * ======================
 */

${#numbers.formatPercent(num)}
${#numbers.arrayFormatPercent(numArray)}
${#numbers.listFormatPercent(numList)}
${#numbers.setFormatPercent(numSet)}

/*
 * Set minimum integer digits and (exact) decimal digits.
 */
${#numbers.formatPercent(num, 3, 2)}
${#numbers.arrayFormatPercent(numArray, 3, 2)}
${#numbers.listFormatPercent(numList, 3, 2)}
${#numbers.setFormatPercent(numSet, 3, 2)}


/*
 * ===============
 * 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**：实用方法 **String** 对象：

```
/*
 * ========================================================================
 * 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**：对于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)}
```

布尔值

- **#bools**：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)}
```

阵列

- **#arrays**：实用方法数组

```
/*
 * ========================================================================
 * 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

- **#lists**：实用方法列表

```
/*
 * ======================================================================
 * 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**：实用方法的集合

```
/*
 * ======================================================================
 * 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**：实用方法的地图

```
/*
 * =============================================================================
 * 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**：实用方法对数组或集合创建聚集

```
/*
 * =============================================================================
 * 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)}
```

标识

- **#ids**：用于与处理工具方法 **id** 可能被重复属性（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')}
```

19附录C：标记选择器语法

Thymeleaf的标记选择器直接从Thymeleaf的解析库借：AttoParser。

这个选择的语法与在XPath的，CSS和jQuery，这使得它们易于使用的大多数用户选择的大的相似性。你可以看看在完整的语法参考AttoParser文档。

例如，下面的选择将选择每一个 ** ** 与类 **content** （注意，这不是因为简洁，因为它可以，看了就知道为什么），在标记内的每个位置：

```
<div th:insert="mytemplate :: //div[@class='content']">...</div>
```

基本语法包括：

- **/x** 意味着名字x为当前节点的直接孩子。
- **//x** 意味着名字x为当前节点的孩子，在任何深度。
- **x[@z="v"]** 意味着名字x和叫z将值“V”的属性的元素。
- **x[@z1="v1" and @z2="v2"]** 装置名称为x的元素和属性Z1和Z2与值“V1”和“V2”，分别。
- **x[i]** 意味着名字元素x定位在我的兄弟姐妹之间的号码。
- **x[@z="v"][i]** 装置与名称x的元素，属性z将值“v”和它的兄弟姐妹也符合此条件中定位在编号i。

但更简洁的语法也可用于：

- **x** 是完全等同于 **//x** （搜索与名称或引用的元素 **x** 在任何深度的水平，基准是一个 **th:ref** 或一个 **th:fragment** 属性）。
- 选择器还允许无元素名称/参考，只要它们包括的参数的规范。所以 **[@class='oneclass']** 是查找任何元素（标签）与值类属性的有效选择 **"oneclass"** 。

高级属性选择功能：

- 除了 **=** （等于），其他比较运算符也有效： **!=** （不等于）， **^=** （开头）和 **$=** （结尾）。例如： **x[@class^='section']** 意味着名称的元素 **x** 和属性的值 **class** 的开头 **section** 。
- 属性可以同时指定开头 **@** （XPath样式）和无（jQuery的风格）。所以 **x[z='v']** 相当于 **x[@z='v']** 。
- 多属性修改器可以在有被连接 **and** （XPath样式），并且还通过链接多个修饰符（jQuery的风格）。所以 **x[@z1='v1' and @z2='v2']** 实际上等同于 **x[@z1='v1'][@z2='v2']** （并且也 **x[z1='v1'][z2='v2']** ）。

直接的*jQuery*般的选择：

- **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** 装置 **<x>** 具有标签 **th:ref="oneref"** 或 **th:fragment="oneref"** 属性。
- **%oneref** 意味着有任何标记 **th:ref="oneref"** 或 **th:fragment="oneref"** 属性。注意，这实际上相当于仅仅 **oneref** 由于引用可以被用来代替元素名称。
- 直接选择器和属性选择器可以混合 **a.external[@href^='https']** 。

所以上面的标记选择器表达式：

```
<div th:insert="mytemplate :: //div[@class='content']">...</div>
```

可以写成：

```
<div th:insert="mytemplate :: div.content">...</div>
```

检查不同的例子，这样的：

```
<div th:replace="mytemplate :: myfrag">...</div>
```

将寻找一个 **th:fragment="myfrag"** 片段签名（或 **th:ref** 引用）。而且也将寻找标签和名称 **myfrag** ，如果他们存在（他们不这样做，在HTML）。注意的区别：

```
<div th:replace="mytemplate :: .myfrag">...</div>
```

.....这实际上会寻找与任何元素 **class="myfrag"** ，而不关心 **th:fragment** 签名（或 **th:ref** 引用）。

多值类匹配

标记选择器了解class属性进行多值，因此可以选择在此属性的应用程序，即使元素拥有多个类值。

例如， **div.two** 将匹配 ** **