Thymeleaf中文参考手册

```
文档版本:20170418 - 18 April 2017

项目版本:3.0.5.RELEASE

项目地址:[http://www.thymeleaf.org](http://www.thymeleaf.org/)
```

1.Thymeleaf简介

1.1 Thymeleaf是什么

Thymeleaf是面向Web和独立环境的现代服务器端Java模板引擎，能够处理HTML，XML，JavaScript，CSS甚至纯文本。

Thymeleaf旨在提供一个优雅的、高度可维护的创建模板的方式。 为了实现这一目标，Thymeleaf建立在自然模板的概念上，将其逻辑注入到模板文件中，不会影响模板设计原型。 这改善了设计的沟通，弥合了设计和开发团队之间的差距。

Thymeleaf从设计之初就遵循Web标准——特别是HTML5标准 ，如果需要，Thymeleaf允许您创建完全符合HTML5验证标准的模板。

1.2 Thymeleaf能处理哪些模版

开箱即用，Thymeleaf可让处理六种类型的模板，每种类型的模板称为模板模式：

- HTML
- XML
- TEXT
- JAVASCRIPT
- CSS
- RAW

这六种模版模式包含两种标记模板模式（HTML和XML），三种文本模板模式（TEXT，JAVASCRIPT和CSS）和一个无操作模板模式（RAW）。

HTML模板模式将允许任何类型的HTML输入，包括HTML5，HTML 4和XHTML。Thymeleaf在html5非验证模式和验证模式下都能正确执行，并且在输出结果中最大程度的遵循模板代码/结构。

XML模板模式将允许XML输入。在这种情况下，代码预期形式良好 - 没有未关闭的标签，没有引用属性等，如果出现非法XML输入，解析器将抛出异常。请注意，Thymeleaf不会执行XML验证（针对DTD或XML架构）。

TEXT模板模式将允许对非标记特性的模板使用特殊语法。例如：文本电子邮件或模板文档。请注意，HTML或XML模板也可以作为TEXT处理，在这种情况下，它们将不会被解析为标记，并且每个标签如：DOCTYPE，注释等都将被视为纯文本。

JAVASCRIPT模板模式将允许在Thymeleaf应用程序中处理JavaScript文件。这意味着可以在JavaScript文件中像与HTML文件中一样的方式使用模型数据，但可以使用特定于JavaScript的集成，例如专门的转义或自然脚本。 JAVASCRIPT模板模式被认为是文本模式，因此使用与TEXT模板模式相同的特殊语法。

CSS模板模式将允许处理涉及Thymeleaf应用程序的CSS文件。与JAVASCRIPT模式类似，CSS模板模式也是文本模式，并使用TEXT模板模式下的特殊处理语法。

RAW模板模式根本不会处理模板。它用于将未经修改的资源（文件，URL响应等）插入正在处理的模板中。例如，HTML格式的外部不受控制的资源可以包含在应用程序模板中，安全地知道这些资源可能包含的任何Thymeleaf代码将不会被执行。

1.3 Thymeleaf标准方言

Thymeleaf是一个扩展性很强的模板引擎（实际上它可以称为模板引擎框架），Thyme Leaf允许您自定义模板，并且很好的处理该模版的细节。

将一些逻辑应用于标记组件（标签，某些文本，注释或只有占位符）的一个对象被称为处理器，通常这些处理器的集合以及一些额外的组件就组成了Thymeleaf方言。 开箱即用，Thymeleaf的核心库提供了一种称为标准方言的方言，这对大多数用户来说应该是足够的。

> 请注意，方言实际上可能不包含处理器，并且完全由其他类型的组件构成，但处理器绝对是最常见的用例。

本教程涵盖Thyme Leaf的标准方言。 您将在后面章节中的每个属性和语法功能都由Thyme Leaf标准方言定义，即使没有明确提及。

当然，如果用户希望在使用标准方言库的高级功能的同时还想定义自己的处理逻辑，您也可以创建自己的方言（甚至扩展标准的方言）。您 也可以将Thymeleaf配置为一次使用几种方言。

> 官方的thymeleaf-spring3和thymeleaf-spring4的整合包里都定义了一种称为“spring标准方言”的方言，该方言与“Thyme Leaf标准方言”大致相同，但是对于Spring框架中的某些功能（例如，通过使用SpringEL表达式代替OGNL表达式）做了一些简单的调整。 所以如果你是一个Spring MVC用户，使用ThymeLeaf并不会浪费你的时间，因为你在这里学到的所有东西都将可以应用到你的Spring应用程序中。

ThymeLeaf标准方言中的大多数处理器都是是属性处理器。 这样，即使在模版未被处理之前，浏览器也可以正确地显示HTML模板文件，因为浏览器将简单地忽略其不识别的属性。 例如，像下面这段JSP模版的代码片段就不能在模版被解析之前通过浏览器直接显示了：

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

然而Thymeleaf标准方言将允许我们实现与上述代码相同的功能：

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

浏览器不仅可以正确显示这些信息，而且还可以（可选地）在浏览器中静态打开时显示一个默认的值（可选地），（在本列中为“James Carrot”）， 在模板处理期间由$ {user.name}的值代替value的真实值。

这有助于您的设计师和开发人员处理相同的模板文件，并减少将静态原型转换为工作模板文件所需的工作量。 具备这种能力的模版我们称为自然模板。

2.示例项目：Good Thymes Virtual Grocery

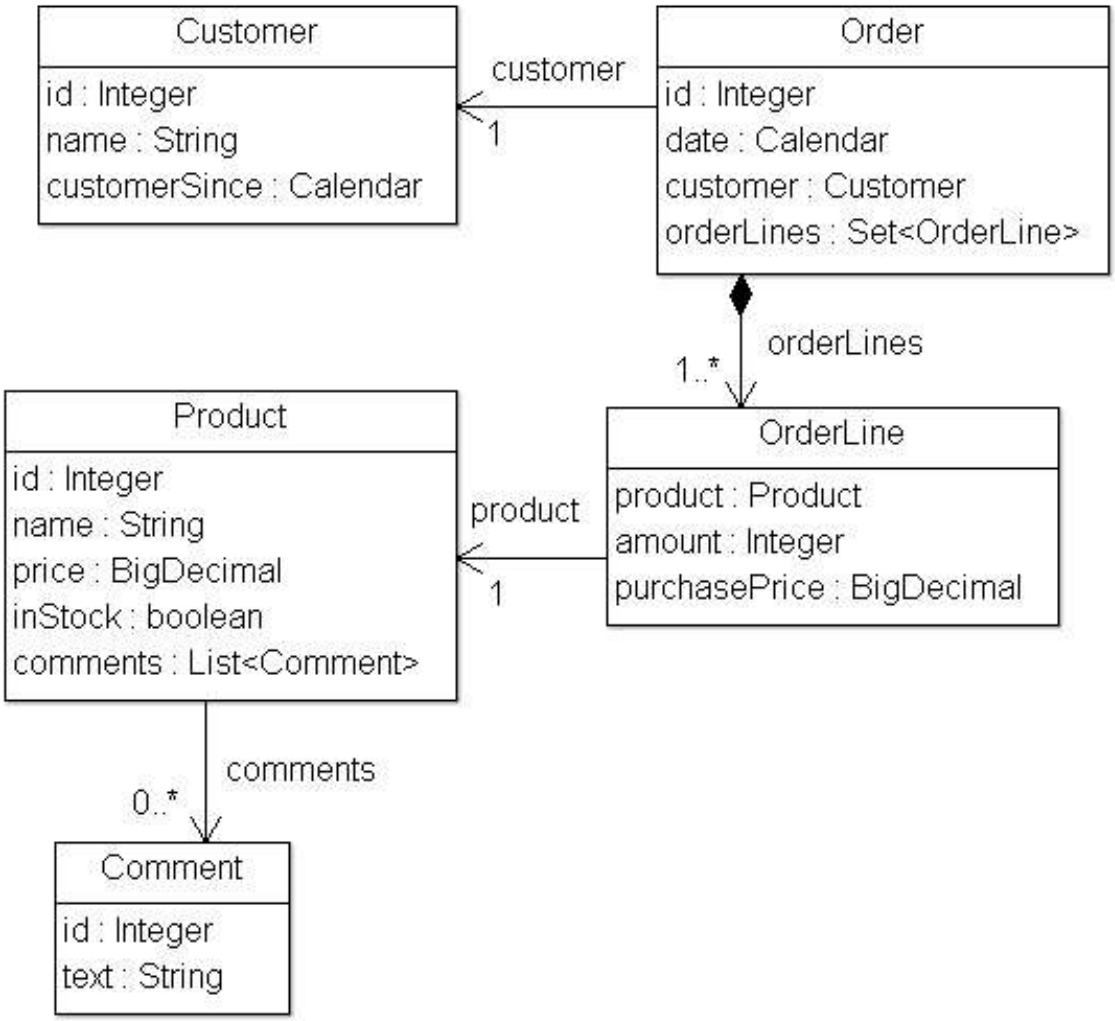
本参考手册以及本手册后续章节中的示例源代码可以在Good Thymes Virtual Grocery GitHub仓库库中找到。

2.1 一个杂货店的网站

为了更好地解释使用Thymeleaf模板所涉及的概念，本教程将通过一个示例程序来说明，该示列程序可以从站点下载。

这个示例程序是一个假想的虚拟杂货店的网站，并将为我们提供许多场景来展示Thymeleaf的许多功能。

在开始介绍示例之前，我们先简单的分析一下示例程序的实体模型：客户（Customers）通过创建订单（Orders）来购买的产品（Products）。系统还提供了对产品评论（Comments）的管理：



示例程序模型

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

```
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，将根据请求URL将执行委托给启用Thymeleaf的命令：

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

    try {

        // This prevents triggering engine executions for
resource URLs
        if (request.getRequestURI().startsWith("/css") ||
                request.getRequestURI().startsWith("/image
s") ||
                request.getRequestURI().startsWith("/favic
```

```
on")) {
            return false;
        }


        /*
         * Query controller/URL mapping and obtain the con
troller
         * that will process the request. If no controller
 is available,
         * return false and let other filters/servlets pro
cess the request.
         */
        IGTVGController controller = this.application.reso
lveControllerForRequest(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 templat
```

```
e,
          * writing the results to the response writer.
          */
         controller.process(
                 request, response, this.servletContext, te
mplateEngine);

         return true;

     } catch (Exception e) {
         try {
             response.sendError(HttpServletResponse.SC_INTE
RNAL_SERVER_ERROR);
         } catch (final IOException ignored) {
             // Just ignore this
         }
         throw new ServletException(e);
     }

}
```

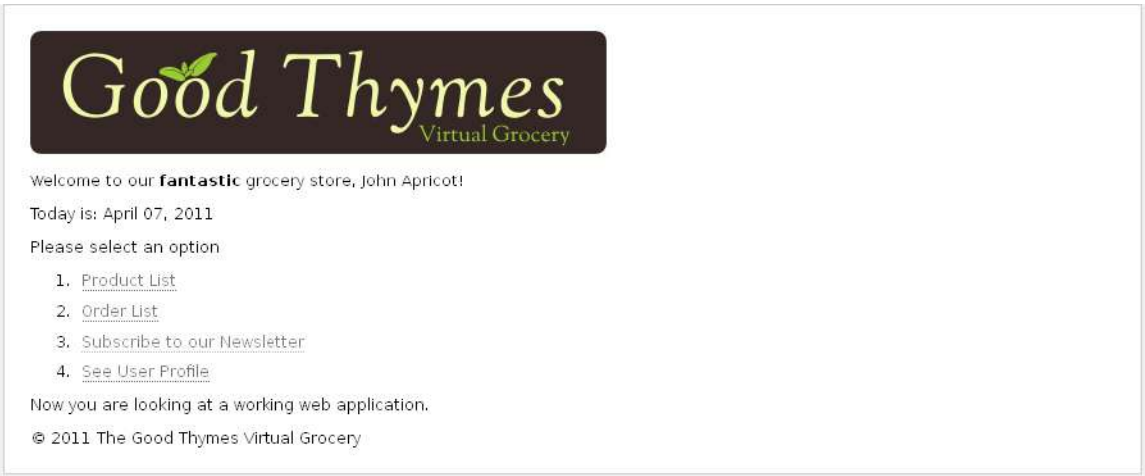
这是我们的IGTVGController接口：

```
public interface IGTVGController {

    public void process(
            HttpServletRequest request, HttpServletRespons
e response,
            ServletContext servletContext, ITemplateEngine
 templateEngine);

}
```

我们现在要做的就是创建IGTVGController接口的实现，并且从服务层中获取数据然后用ITemplateEngine对象处理模板并完成数据渲染。最后程序运行结果如下所示：



示例程序运行结果

首先我们先看一下模版引擎是如何初始化的。

2.2 创建和配置模版引擎

在过滤器的process(……）方法中包含下面一行代码：

```
ITemplateEngine templateEngine = this.application.getTempl
ateEngine();
```

这意味着GTVGApplication类负责创建和配置Thymeleaf应用程序中最重要的对象之一：TemplateEngine实例（ITemplateEngine接口的实现）。

我们的org.thymeleaf.TemplateEngine对象初始化如下：

```
public class GTVGApplication {


    ...
    private static TemplateEngine templateEngine;
    ...


    public GTVGApplication(final ServletContext servletCon
text) {

        super();

        ServletContextTemplateResolver templateResolver =
                new ServletContextTemplateResolver(servlet
Context);

        // 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 woul
d be cached until expelled by LRU
        templateResolver.setCacheTTLMs(Long.valueOf(360000
0L));

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

        this.templateEngine = new TemplateEngine();
        this.templateEngine.setTemplateResolver(templateRe
solver);

        ...

    }

}
```

这里有很多方法可以配置TemplateEngine对象，但本示例程序中的这几行配置模版引擎代码将足够教给我们所需的步骤。

2.2.1 模版解析器

我们从模板解析器开始：

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

模板解析器对象是org.thymeleaf.templateresolver.ITemplateResolver接口的实现对象：

```
public interface ITemplateResolver {

    ...

    /*
     * Templates are resolved by their name (or content) a
nd also (optionally) their
     * owner template in case we are trying to resolve a f
ragment for another template.
     * Will return null if template cannot be handled by t
his template resolver.
     */
    public TemplateResolution resolveTemplate(
            final IEngineConfiguration configuration,
            final String ownerTemplate, final String templ
ate,
            final Map<String, Object> templateResolutionAt
tributes);
}
```

这些对象负责访问我们的模板，在GTVG应用程序中org.thymeleaf.templateresolver.ServletContextTemplateResolver意味着我们将从Servlet上下文中获取我们的模板文件资源：应用程序范围的每个

Java Web应用程序中都存在的javax.servlet.ServletContext对象，并从Web应用程序根目录中解析资源。

但是，这并不是我们可以对模板解析器所说的，因为我们可以设置模版解析器的一些配置参数。 一，模板模式：

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

HTML是ServletContextTemplateResolver的默认模板模式，但无论如何，建立它是很好的做法，以便我们的代码清楚地说明了发生了什么。

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

通过设置模版的前缀和后缀名称，以变获取真实的资源名称。

使用此配置，模板名称“product / list”将对应于：

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

可选地，可以通过cacheTTLMs属性配置已解析的模板实例的缓存生存的时间：

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

如果达到最大高速缓存大小，并且它是当前缓存的最早的条目，模板仍然可以在达到TTL之前被从缓存中排出。

> 缓存行为和大小可由用户通过实现ICacheManager接口或修改StandardCacheManager对象来管理默认缓存来定义。

关于模版解析器还有更多的知识需要介绍，但是现在我们来看看模板引擎对象的创建。

2.2.2 模版引擎

模板引擎对象是org.thymeleaf.ITemplateEngine接口的实现。 这些实现之一由Thymeleaf核心提供org.thymeleaf.TemplateEngine，我们在此创建一个实例：

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

相当简单，不是吗？ 我们所需要的就是创建一个实例并将Template Resolver设置为它的属性。

模板解析器是TemplateEngine需要的唯一必需参数，尽管还有很多其他参数将被覆盖（消息解析器，缓存大小等）。 现在，这就是我们所需要的。

我们的模板引擎现在已经准备就绪，我们可以开始使用Thymeleaf创建我们的页面。

3.1 多语言欢迎页

我们的第一个任务是为我们的杂货店创建一个主页。

该页面的第一个版本将非常简单：只需一个标题和一个欢迎信息。 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; ch
arset=UTF-8" />
    <link rel="stylesheet" type="text/css" media="all"
          href="../../css/gtvg.css" th:href="@{/css/gtvg.c
ss}" />
  </head>

  <body>

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

  </body>

</html>
```

您会注意到的第一件事是，该文件是HTML5，可以由任何浏览器正确显示，因为它不包括任何非HTML标签（浏览器忽略他们不明白的所有属性，如：th:text）。

但是您也可能会注意到，这个模板并不是一个真正有效的HTML5文档，因为HTML5规范不允许在th：*形式中使用这些非标准属性。 事实上，我们甚至在我们的<html>标签中添加了一个xmlns：th属性，这绝对是非HTML5标准：

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

...这些在模板处理中根本没有任何影响，但作为一种编程惯例，我们通常会阻止IDE提示th:*属性缺少命名空间定义的警告。

那么如何使这个HTML5模板有效呢？ Easy：切换到Thymeleaf的数据属性语法，使用data-属性名称和连字符（ - ）分隔符的数据前缀而不是分号（:）：

```
<!DOCTYPE html>

<html>

  <head>
    <title>Good Thymes Virtual Grocery</title>
    <meta http-equiv="Content-Type" content="text/html; ch
arset=UTF-8" />
    <link rel="stylesheet" type="text/css" media="all"
          href="../../css/gtvg.css" data-th-href="@{/css/g
tvg.css}" />
  </head>

  <body>

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

  </body>
```

```
</html>
```

HTML5规范要求用户自定义属性以data-前缀开头，因此，使用上面的代码，我们的模板将是一个有效的HTML5文档。

> 这两种形式是完全等同的并且是可互换的，但为了使我们的示例代码简单和紧凑，本教程将使用命名空间符号（th：*）这种形式。 此外，th：*符号在每个Thymeleaf模板模式（XML，TEXT ...）中更为通用，而data-这种形式只允许在HTML模式下使用。

3.1.1 使用th:text和外部化文本

外部化文本是从模板文件中提取模板代码的片段，以便它们可以保存在单独的文件（通常为.properties文件）中，并且可以轻松地替换为使用其他语言编写的等效文本（称为国际化或简单的i18n）。 文本的外部化片段通常称为“消息”。

消息总是具有标识它们的键，而Thymeleaf允许您使用 # {...}语法表达式指定对应的特定消息：

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

我们在这里看到的其实是Thymeleaf标准方言的两个不同特征：

- th：text属性，它计算其值表达式并将结果设置为标签的标签体，有效地替换了代码中我们看到的“欢迎使用我们的杂货店！”这段文本。
- #{home.welcome}标准表达式语法中指定的 # {home.welcome}表达式指示由th：text属性使用的文本应该是home.welcome键对应于我们正在处理模板的区域设置的消息。

现在，这个外部化文本在哪里?

Thymeleaf中外部化文本的位置是完全可配置的，它将取决于正在使用的具体的org.thymeleaf.messageresolver.IMessageResolver实现。通常，将使用基于.properties文件的实现，但是如果我们想要（例如）从数据库获取消息，我们可以创建自己的实现。

但是，我们在初始化期间尚未为模板引擎指定消息解析器，这意味着我们的应用程序正在使用由org.thymeleaf.messageresolver.StandardMessageResolver实现的标准消息解析器。

标准消息解析器期望在/WEB-INF/templates/home.html中找到与该模板相同的文件夹中的属性文件的消息，例如：

- /WEB-INF/templates/home_en.properties为英文文本。
- 西班牙语文本的/WEB-INF/templates/home_es.properties。
- 葡萄牙语（巴西）语言文本的/WEB-INF/templates/home_pt_BR.properties。
- /WEB-INF/templates/home.properties为默认文本（如果语言环境不匹配）。

我们来看看我们的home_es.properties文件：

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

这就是我们所需要的，让我们制作我们的模板。 然后让我们创建我们的Home controller控制器。

3.1.2 上下文

为了处理我们的模板，我们将创建一个实现IGTVGController接口的HomeController类：

```
public class HomeController implements IGTVGController {

    public void process(
            final HttpServletRequest request, final HttpSe
rvletResponse response,
            final ServletContext servletContext, final ITe
mplateEngine templateEngine)
            throws Exception {

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

        templateEngine.process("home", ctx, response.getWr
iter());

    }

}
```

我们首先看到的是创建一个上下文。 Thymeleaf上下文是实现org.thymeleaf.context.IContext接口的对象。 上下文应包含在变量映射中执行模板引擎所需的所有数据，并且还引用必须用于外部化消息的区域设置。

```
public interface IContext {

    public Locale getLocale();
```

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

}
```

这个接口有一个专门的扩展，org.thymeleaf.context.IWebContext，用于基于Servlet API的Web应用程序（如SpringMVC）中。

```
public interface IWebContext extends IContext {

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

}
```

Thymeleaf核心库提供了以下每个接口的实现：

- org.thymeleaf.context.Context实现IContext
- org.thymeleaf.context.WebContext实现IWebContext

而在控制器代码中可以看到，WebContext是我们使用的。实际上我们必须，因为使用一个ServletContextTemplateResolver要求我们使用实现IWebContext的上下文。

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

只需要这四个构造函数参数中的三个，因为如果没有指定，那么将使用系统的默认语言环境（尽管不应该在实际应用程序中发生）。

有一些专门的表达式，我们将能够用来从我们的模板中的WebContext获取请求参数和请求，会话和应用程序属性。 例如：

- $ {x}将返回存储在Thymeleaf上下文中的变量x或作为请求属性。
- $ {param.x}将返回一个名为x的请求参数（可能是多值的）。
- $ {session.x}将返回一个名为x的会话属性。
- $ {application.x}将返回一个名为x的servlet上下文属性。

3.1.3 执行模版引擎

随着我们的上下文对象准备就绪，现在我们可以告诉模板引擎使用上下文来处理模板（通过模版的名字），并传递一个响应写入器，以便可以将响应写入它：

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

让我们看看西班牙语言环境的结果：

```
<!DOCTYPE html>

<html>

  <head>
    <title>Good Thymes Virtual Grocery</title>
    <meta content="text/html; charset=UTF-8" http-equiv="C
ontent-Type"/>
    <link rel="stylesheet" type="text/css" media="all" hre
f="/gtvg/css/gtvg.css" />
  </head>

  <body>

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

  </body>

</html>
```

3.2.1 非转义文本

我们的主页的最简单的版本似乎已经准备好了，但是我们还没有想到什么呢？如果我们有这样的消息呢？

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

如果我们像以前一样执行此模板，我们将获得：

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

这不是我们预期的效果，因为我们的<b>标签已被转义，因此它将显示在浏览器中。

这是th：text属性的默认行为。 如果我们希望Thymeleaf原样输出我们的HTML标签，而不是转义他们，我们将不得不使用一个不同的属性：th：utext（对于“unescaped text”）：

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

这将输出我们的消息就像我们想要的：

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

3.2.2 使用和显示变量

现在我们再添加一些更多的内容到我们的主页。 例如，我们可能希望在我们的欢迎信息下方显示日期，如下所示：

```
Welcome to our fantastic grocery store!

Today is: 12 july 2010
```

首先，我们将必须修改我们的控制器，以便将该日期添加为上下文变量：

```
public void process(
            final HttpServletRequest request, final HttpSe
rvletResponse response,
            final ServletContext servletContext, final ITe
mplateEngine templateEngine)
            throws Exception {

    SimpleDateFormat dateFormat = new SimpleDateFormat("dd
 MMMM yyyy");
    Calendar cal = Calendar.getInstance();

    WebContext ctx =
            new WebContext(request, response, servletConte
xt, request.getLocale());
    ctx.setVariable("today", dateFormat.format(cal.getTime
()));

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

}
```

我们在我们的上下文中添加了一个名为today的String类型变量，现在我们可以在我们的模板中显示它：

```
<body>

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

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

</body>
```

正如你所看到的，我们仍然使用th：text属性（这是正确的，因为我们要替换标签的正文），但是这个时候语法有点不同，而不是＃{.. 。}表达式值，我们使用$ {...}表达式。 这是一个变量表达式，它包含一个名为OGNL（Object-Graph Navigation Language）的表达式，它将在之前讨论的上下文变量映射上执行。

$ {today}表达式只是意味着“获取当前调用的变量”，但是这些表达式可能更复杂（例如$ {user.name} for“get the variable called user，and call its getName（）method”）。属性值有很多可能性：消息，变量表达式...等等。 下一章将向大家介绍一下这些可能性。

4.标准表达式语法

我们将在我们的虚拟杂货商店项目的开发中休息一下，接下来我们来学习Thyme Leaf标准方言中最重要的部分之一：“Thyme Leaf标准表达式”语法。

我们已经看到了两种类型的属性值的表达方式：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</sp
an></p>
```

但是，Thyme Leaf还有更多类型的表达式和更多有趣的细节需要我们去学习。 首先，我们来看看标准表达式功能的快速总结：

- 简单表达式：

  ```
  变量表达式：$ {...}

  选择变量表达式：\* {...}

  消息表达式： #{...}

  链接网址表达式：@ {...}

  片段表达式：～{...}
  ```

- 文字

  文字文字： `'one text'` , `'Another one!'`

```
数字字面值：0,34,3.0,12.3，...

布尔文字：true，false

空字面值：null

文字Token：one，sometext，main，...
```

- 文本操作

  字符串连接：+

  文本替换：|The name is ${name}|

- 算术运算符

```
二进制运算符：+，-，\*，/，%
```

- 负号（一元运算符）：-
- 布尔运算符

```
二进制运算符：and 、or

布尔否定（一元运算符）：！，not
```

- 比较和相等运算符：

```
比较运算符：&gt;，&lt;，&gt; =，&lt;=（gt，lt，ge，le）

相等运算符：==，！=（eq，ne）
```

- 条件运算符：

```
 If-then:\(if\) ? \(then\)

If-then-else:\(if\) ? \(then\) : \(else\)

Default:\(value\) ?: \(defaultvalue\)
```

- 特殊符号:

```
哑操作符: \_
```

所有这些功能可以组合和嵌套:

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

4.1 消息

我们已经知道， #{...}消息表达式允许我们这样引用消息字符串：

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

上面表达式的消息字符串如下：

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

但是还有一个方面我们还没有想到：如果消息文本不是完全静态的，会发生什么？ 例如，如果我们的应用程序想知道是哪个用户访问该网站，我们是否可以通过用户名字来问候呢?

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

这意味着我们需要在我们的消息中添加一个参数。 像这样：

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

根据java.text.MessageFormat标准语法指定参数，并且可以根据这些类的API文档中指定的数字和日期格式来填充参数。

为了指定我们参数的值，并给出一个名为user的HTTP会话属性，我们将具有：

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

```
  </p>
```

可以指定几个参数，用逗号分隔。 实际上，消息键本身可以来自一个变量：

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

4.2 变量

我们已经提到$ {...}表达式实际上是在上下文中包含的变量的映射上执行的OGNL（Object-Graph Navigation Language）对象。

> 有关OGNL语法和功能的详细信息，请阅读OGNL语言指南。在Spring MVC启用的应用程序中，OGNL将被替换为SpringEL，但其语法与OGNL非常相似（实际上，在大多数常见情况下完全相同）。

从OGNL的语法，我们知道：

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

实际上等同于：

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

但OGNL允许我们创建更强大的表达式，这就是这样的：

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

通过执行如下代码获得用户名称

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

但是getter方法导航只是OGNL的一个功能。 我们再来看一下：

```
/*
 * 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 singl
e quotes.
 */
${person['father']['name']}

/*
 * If the object is a map, both dot and bracket syntax wil
l be equivalent to
 * executing a call on its get(...) method.
 */
${countriesByCode.ES}
${personsByName['Stephen Zucchini'].age}

/*
 * Indexed access to arrays or collections is also perform
ed with brackets,
 * writing the index without quotes.
 */
${personsArray[0].name}

/*
 * Methods can be called, even with arguments.
 */
${person.createCompleteName()}
${person.createCompleteNameWithSeparator('-')}
```

4.2.1 基本表达式对象

当对上下文变量计算OGNL表达式时，某些对象可用于表达式以获得更高的灵活性。 这些对象将被引用（按照OGNL标准），从 # 符号开始：

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

因此，我们可以这样做：

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

您可以在附录A中阅读这些对象的完整参考。

4.2.2 工具表达式对象

除了这些基本的对象之外，Thymeleaf将为我们提供一组工具对象，这些对象将帮助我们在表达式中执行常见任务。

- #execInfo：有关正在处理的模板的信息。
- #messages：用于在变量表达式中获取外部化消息的方法，与使用#{...}语法获得的方式相同。
- #uris：转义URL / URI部分的方法
- #conversions：执行配置的转换服务（如果有的话）的方法。
- #dates：java.util.Date对象的方法：格式化，组件提取等
- #calendars：类似于#dates，但对于java.util.Calendar对象。
- #numbers：用于格式化数字对象的方法。
- #strings：String对象的方法：contains，startsWith，prepending / appending等
- #objects：一般对象的方法。
- #bools：布尔评估的方法。
- #arrays：数组的方法。
- #lists：列表的方法。
- #sets：集合的方法。
- #maps：地图方法。
- #aggregates：在数组或集合上创建聚合的方法。
- #ids：处理可能重复的id属性的方法（例如，作为迭代的结果）。

您可以检查附录B中每个实用程序对象提供的功能。

4.2.3 重新格式化首页的日期

现在我们知道这些工具对象，我们可以使用它们来改变我们在主页上显示日期的方式。 而不是在我们的HomeController中这样做：

```
SimpleDateFormat dateFormat = new SimpleDateFormat("dd MMM
M yyyy");
Calendar cal = Calendar.getInstance();

WebContext ctx = new WebContext(request, servletContext, r
equest.getLocale());
ctx.setVariable("today", dateFormat.format(cal.getTime()))
;

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

我们可以这样做：

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

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

...然后在视图层执行日期格式化：

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

4.3选择表达式（星号语法）

我们不仅可以将变量表达式写为$ {...}，还可以作为* {...}。

这两种方式有一个重要的区别：星号语法计算所选对象而不是整个上下文的表达式。 也就是说，只要没有选定的对象，$和*语法就会完全相同。

什么是选定对象？ 使用th：object属性的表达式的结果。 我们在用户个人资料（userprofile.html）页面中使用一个：

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

这完全等同于：

```
<div>
  <p>Name: <span th:text="${session.user.firstName}">Sebas
tian</span>.</p>
  <p>Surname: <span th:text="${session.user.lastName}">Pep
per</span>.</p>
  <p>Nationality: <span th:text="${session.user.nationalit
y}">Saturn</span>.</p>
</div>
```

当然，美元和星号的语法可以混合使用：

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

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

当对象选择到位时，所选对象也将作为#object表达式变量可用于美元表达式：

```
<div th:object="${session.user}">
  <p>Name: <span th:text="${#object.firstName}">Sebastian<
/span>.</p>
  <p>Surname: <span th:text="${session.user.lastName}">Pep
per</span>.</p>
  <p>Nationality: <span th:text="*{nationality}">Saturn</s
pan>.</p>
</div>
```

如上所述，如果没有执行对象选择，则美元和星号语法是等效的。

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

4.4 URL链接

由于它们的重要性，URL是Web应用程序模板中的一等公民，而Thymeleaf标准方言具有特殊的语法，@语法：@ {...}

有不同类型的网址：

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

这些表达式的真实处理及其转换为将被输出的URL将通过注册到正在使用的ITemplateEngine对象中的org.thymeleaf.linkbuilder.ILinkBuilder接口的实现完成。

默认情况下，该接口的单个实现已注册到类org.thymeleaf.linkbuilder.StandardLinkBuilder，这对于脱机（非Web）以及基于Servlet API的Web场景都是足够的。其他情况（如与非Servlet API Web框架的集成）可能需要链接构建器接口的特定实现。

我们来使用这个新的语法。认识th:href属性：

```
<!-- Will produce 'http://localhost:8080/gtvg/order/detail
s?orderId=3' (plus rewriting) -->
<a href="details.html"
   th:href="@{http://localhost:8080/gtvg/order/details(ord
erId=${o.id})}">view</a>

<!-- Will produce '/gtvg/order/details?orderId=3' (plus re
```

```
writing) -->
<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>
```

注意：

- th:href是一个修饰符属性：一旦处理，它将计算要使用的链接URL，并将该值设置为<a>标签的href属性。
- 我们被允许使用表达式的URL参数（可以在orderId = $ {o.id}中看到）。所需的URL参数编码操作也将自动执行。
- 如果需要几个参数，这些参数将以逗号分隔：@ {/ order / process（execId = $ {execId}，execType ='FAST'）}
- URL路径中也允许使用变量模板：@ {/ order / {orderId} / details（orderId = $ {orderId}）}
- 以/开头的相对URL（例如：/ order / details）将自动以应用程序上下文名称为前缀。
- 如果cookie未启用或尚未知道，则可能会在相对URL中添加“; jsessionid = ...”后缀，以便会话被保留。这被称为URL重写，Thymeleaf允许您使用Servlet API中的每个URL的response.encodeURL（...）机制来插入自己的重写过滤器。
- th：href属性允许我们（可选地）在我们的模板中有一个工作的静态href属性，这样当我们直接打开原型设计时，我们的模板链接可以被浏览器导航。

与消息语法（＃{...}）的情况一样，URL基数也可以是计算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>
```

4.4.1 主页菜单

现在我们知道如何创建链接网址，如何在网站的其他一些页面中添加一个小菜单呢?

```
<p>Please select an option</p>
<ol>
  <li><a href="product/list.html" th:href="@{/product/list
}">Product List</a></li>
  <li><a href="order/list.html" th:href="@{/order/list}">O
rder List</a></li>
  <li><a href="subscribe.html" th:href="@{/subscribe}">Sub
scribe to our Newsletter</a></li>
  <li><a href="userprofile.html" th:href="@{/userprofile}"
>See User Profile</a></li>
</ol>
```

4.4.2 服务器相对URL

可以使用附加语法来创建服务器根目录（而不是上下文相对）URL，以链接到同一服务器中的不同上下文。 这些URL将被指定为

@ {～/path/to/something}

4.5 代码片段

代码片段表达式是表示标记片段的简单方法，并将其移动到模板周围。这允许我们复制它们，将它们传递给其他模板作为参数，等等。

最常见的用法是使用th:insert或th:replace进行片段插入（稍后部分更详细的介绍）：

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

它们可以在任何地方使用，就像任何其他变量一样：

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

在本教程的后面第8章，有一个专门用来说明模板布局的部分，这部分会对片段表达式进行更深入的剖析。

4.6 文本

4.6.1 纯文本

文本文字只是包含在单引号之间的字符串。 它们可以包含任何字符，但是您应该使用\'转义其中的任何单引号。

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

4.6.2 数字字面量

数字字面量就是这样：数字。

```
<p>The year is <span th:text="2013">1492</span>.</p>
<p>In two years, it will be <span th:text="2013 + 2">1494<
/span>.</p>
```

4.6.3 布尔字面量

布尔字面量包含true和false。 例如：

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

在这个例子中，== false被写在大括号之外，所以它是Thymeleaf来处理的。如果它是写在大括号内，那将是OGNL / SpringEL引擎的责任：

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

4.6.4 NULL 字面量

Thyme Leaf标准表达式语法中也可以使用null字面量：

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

4.6.5 文本符号

数字，布尔和空值实际上是文本符号的特殊情况。

这些符号允许在标准表达式中进行一点简化。 它们与文本文字（'...'）完全一样，但它们只允许使用字母（A-Z和a-z），数字（0-9），括号（[和]），点（。），连字符（ - ） 和下划线（_）。 所以没有空白，没有逗号等

这样做好吗？ 令牌不需要围绕它们的任何引号。 所以我们可以这样做：

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

来代替：

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

4.7 追加文本

无论是字符串文本常量，还是通过变量表达式或消息表达式计算的结果，都可以使用+运算符轻松地追加文本：

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

4.8 文本替换

文本替换允许容易地格式化包含变量值的字符串，而不需要使用“...”+“...”附加文字。

这些替换必须被垂直条（|）包围，如：

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

这等价于下面的代码：

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

文本替换可以与其他类型的表达式相结合使用：

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

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

4.9 算术运算符

Thyme Leaf标准表达式还支持一些算术运算：+， -，*，/和%。

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

请注意，这些运算符也可以在OGNL变量表达式本身中应用（在这种情况下将由OGNL解析执行，而不是Thymeleaf标准表达式引擎来解析执行）：

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

请注意，其中一些运算符存在文本别名：div（/），mod（%）。

4.10 比较和等值运算符

表达式中的值可以与>，<，> =和<=符号进行比较，并且可以使用==和！=运算符来检查是否相等（或缺少）。 请注意，XML类型的模版中不应在属性值中使用<和>符号，应该使用＆lt; 和＆gt;。

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

一个更简单的替代方案是使用这些运算符的文本别名：gt（>），lt（<），ge（> =），le（<=），非（！）。 还有eq（==），neq / ne（！=）。

4.11 条件表达式

条件表达式仅用于计算两个表达式中的一个，这取决于计算条件表达式（本身就是另一个表达式）的结果。

我们来看一个示例片段（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（condition，then和else）的三个部分都有自己的表达式，这意味着它们可以是变量（${...}，*{...}），消息（#{...}），URL（@{...}）或文字（'...'）。

条件表达式也可以使用括号嵌套：

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

Else表达式也可以省略，在这种情况下，如果条件为false，则返回null值：

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

4.12 默认表达式（Elvis operator）

默认表达式是一种特殊类型的条件值，没有then那个部分。 它相当于以一些语言（如Groovy）的Elvis操作符，允许您指定两个表达式：如果计算结果不为null，则使用第一个表达式，但如果计算结果为null则使用第二个表达式。

在我们的用户个人资料页面中：

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

您可以看到，运算符是?：，我们在这里使用它来指定age的默认值（在这种情况下是字面值），只有计算* {age}的结果为null时，才使用默认值（no age specified）。 这等价于：

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

与条件表达式一样，它们可以在括号之间包含嵌套表达式：

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

4.13 哑操作符号

哑操作符号由下划线符号（_）表示。

这个标记指定表达式不处理任何结果，即相当于可执行的属性（例如：文本）不存在一样。除了其他可能性之外，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>
```

我们可以直接使用*'no user authenticated'*作为原型设计文本，从而从设计的角度来看，这些代码更加简洁和多才多艺：

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

4.14 预处理

Thymeleaf除了表达式所处理的这些功能之外，还具有预处理表达式的功能。

预处理是在正常的表达式之前执行的表达式，允许修改最终将被执行的表达式。

预处理的表达式与正常表达式完全相同，但是由双下划线符号（如__ ${expression} __）显示。

我们假设我们有一个i18n Messages_fr.properties条目，其中包含一个OGNL表达式，该表达式调用语言特定的静态方法，如：

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

和一个对应的Messages_es.properties：

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

我们可以创建一个标记片段，根据语言环境来计算一个表达式或另一个表达式。 为此，我们首先选择表达式（通过预处理），然后让Thymeleaf执行它：

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

请注意，法语区域设置的预处理步骤将创建以下等效项：

```
<p th:text="${@myapp.translator.Translator@translateToFren
```

```
ch(textVar)}">Some text here...</p>
```

使用\ _ \ _可以在属性中转义预处理String __。

4.15 数据类型转换与格式化

Thymeleaf为变量（${...}）和选择（* {...}）表达式定义了一个双括号语法，允许我们通过配置的数据转换服务来进行数据类型转换。

它基本上用法如下的：

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

注意到那里的双括号了吗？。 这告诉Thymeleaf将user.lastAccessDate表达式的结果传递给注册转换服务，并要求在写入结果之前执行格式化操作（转换为String）。

假设user.lastAccessDate的类型为java.util.Calendar，如果转换服务（IStandardConversionService的实现）已经被注册，并且包含有效的Calendar - > String转换，它将被应用。

IStandardConversionService（StandardConversionService类）的默认实现只需对转换为String的任何对象执行.toString（）。 有关如何注册自定义转换服务实现的更多信息，请参阅本手册第15章关于Thyme Leaf的更多配置部分。

> 官方的thymeleaf-spring3和thymeleaf-spring4集成包将Thymeleaf的转换服务机制与Spring自己的转换服务模块进行了透明的集成，使得在Spring配置中声明的转换服务和格式化程序将自动应用到双大括号表达式。

5.设置属性值

本章将介绍如何在标记中设置（或修改）属性的值的。

5.1 设置任何属性的值

当我们通过网站发布资讯后希望用户能够订阅我们的资讯，所以我们需要创建一个订阅消息的模板（/WEB-INF/templates/subscribe.html）：

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

与Thymeleaf一样，这个模板相对于一个Web应用程序的模板来说更像是一个静态网页原型。 首先，我们表单中的action属性静态地链接到模板文件本身，这样就没有可用的URL重写的地方。 其次，提交按钮中的属性值以英文文本显示，但我们希望将其进行国际化。

输入th:attr属性，以设置标签的属性值：

```
<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>
```

这个实现原理很简单：th:attr只需要通过一个表达式将值赋给对应的属性。 创建相应的控制器和消息文件后，此表单的执行后输出结果如下：

```
<form action="/gtvg/subscribe">
```

```
  <fieldset>
    <input type="text" name="email" />
    <input type="submit" value="¡Suscríbe!"/>
  </fieldset>
</form>
```

除了新的属性值之外，您还可以看到，应用上下文名称已经自动作为前缀添加到基础URL中，最终的表单提交路径是 / gtvg/subscribe中，这在前面章节中已经有相关的说明。

但是如果我们想要一次设置多个属性值呢？ XML规则不允许您在标签中设置两次属性，因此th:attr将以逗号分隔的形式来设置多个属性值，如：

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

给定所需的消息文件，这将输出：

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

5.2 设置指定属性的值

现在，你可能会想到像：

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

...这是一个相当丑陋的标记。 在属性值内指定赋值可能非常实用，但如果您必须一直执行，则不是最优雅的创建模板的方式。

Thymeleaf同意你的意思，这就是为什么th:attr几乎不用的原因。 通常，您将使用其他th:*属性来设置特定的标记属性指（而不仅仅是像th:attr这样的任何属性）。

例如，要设置value属性，请使用th:value：

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

看起来好多了！ 我们尝试对表单标签中的action属性执行相同操作：

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

你还记得在我们的home.html之前的th:href吗？ 他们是同样的属性：

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

还有很多类似的属性，每个都针对特定的HTML5属性：

`th:abbr`	`th:accept`	`th:accept-charset`

th:accesskey	th:action	th:align
th:alt	th:archive	th:audio
th:autocomplete	th:axis	th:backg
th:bgcolor	th:border	th:cellpa
th:cellspacing	th:challenge	th:chars
th:cite	th:class	th:class
th:codebase	th:codetype	th:cols
th:colspan	th:compact	th:conte
th:contenteditable	th:contextmenu	th:data
th:datetime	th:dir	th:dragg
th:dropzone	th:enctype	th:for
th:form	th:formaction	th:forme
th:formmethod	th:formtarget	th:fragm
th:frame	th:frameborder	th:heade
th:height	th:high	th:href
th:hreflang	th:hspace	th:http-
th:icon	th:id	th:inlin
th:keytype	th:kind	th:label
th:lang	th:list	th:longd
th:low	th:manifest	th:margi
th:marginwidth	th:max	th:maxle
th:media	th:method	th:min
th:name	th:onabort	th:onaft
th:onbeforeprint	th:onbeforeunload	th:onblu
th:oncanplay	th:oncanplaythrough	th:oncha
th:onclick	th:oncontextmenu	th:ondbl

th:ondrag	th:ondragend	th:ondra
th:ondragleave	th:ondragover	th:ondra
th:ondrop	th:ondurationchange	th:onemp
th:onended	th:onerror	th:onfoc
th:onformchange	th:onforminput	th:onhas
th:oninput	th:oninvalid	th:onkey
th:onkeypress	th:onkeyup	th:onloa
th:onloadeddata	th:onloadedmetadata	th:onloa
th:onmessage	th:onmousedown	th:onmou
th:onmouseout	th:onmouseover	th:onmou
th:onmousewheel	th:onoffline	th:ononl
th:onpause	th:onplay	th:onpla
th:onpopstate	th:onprogress	th:onrat
th:onreadystatechange	th:onredo	th:onres
th:onresize	th:onscroll	th:onsee
th:onseeking	th:onselect	th:onsho
th:onstalled	th:onstorage	th:onsub
th:onsuspend	th:ontimeupdate	th:onund
th:onunload	th:onvolumechange	th:onwai
th:optimum	th:pattern	th:place
th:poster	th:preload	th:radio
th:rel	th:rev	th:rows
th:rowspan	th:rules	th:sandb
th:scheme	th:scope	th:scrol
th:size	th:sizes	th:span

th:spellcheck	th:src	th:srcla
th:standby	th:start	th:step
th:style	th:summary	th:tabin
th:target	th:title	th:type
th:usemap	th:value	th:value
th:vspace	th:width	th:wrap
th:xmlbase	th:xmllang	th:xmlsp

5.3 一次设置多个属性的值

还有两个相当特殊的属性叫做th:alt-title和th:lang-xmllang，可用于同时将两个属性设置为相同的值。 特别地：

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

对于我们的GTVG主页，这将允许我们用以下方式来代替：

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

上述代码等价于：

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

也可以这样简写为：

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

5.4 后缀和前缀

Thymeleaf还提供th:attrappend和th:attrprepend属性，它们为现有属性值设置前缀和后缀。

例如，您可能将要添加的CSS类的名称（未设置，刚添加）存储在上下文变量的其中一个按钮中，因为要使用的特定CSS类将取决于用户所做的某些操作 之前：

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

如果为cssStyle变量赋值为“warning”来处理此模板，您将获得：

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

Thymeleaf标准方言中还有两个特定的附加属性：th:classappend和th:styleappend属性，用于将CSS类或样式片段添加到元素中，而不覆盖现有属性：

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

（不要担心th:each属性，它是一个迭代属性，我们稍后会讨论）

5.5 固定值的布尔属性

HTML具有布尔属性的概念，这个布尔属性没有值，并且一旦这个布尔属性存在则意味着属性值为“true”。 但是在XHTML中，这些属性只取一个值，这个属性值就是它本身。

例如，checked属性：

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

Thymeleaf标准方言允许您通过计算条件表达式的结果来设置这些属性的值，如果条件表达式结果为true，则该属性将被设置为其固定值，如果评估为false，则不会设置该属性：

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

Thymeleaf标准方言还支持以下固定值布尔属性：

th:async	**th:autofocus**	**th:autoplay**
th:checked	th:controls	th:declare
th:default	th:defer	th:disabled
th:formnovalidate	th:hidden	th:ismap
th:loop	th:multiple	th:novalidate
th:nowrap	th:open	th:pubdate
th:readonly	th:required	th:reversed
th:scoped	th:seamless	th:selected

5.6 设置其他属性的值（默认属性处理器）

Thymeleaf提供了一个默认属性处理器，允许我们设置任何属性的值，即使在标准方言中没有为其定义特定的th:*处理器。

所以这样的东西：

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

输出结果如下：

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

5.7 HTML5友好属性和标签名的支持

ThymeLeaf还支持HTML5友好属性语法来处理模版。

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

data-{prefix}-{name}语法是在HTML5中编写自定义属性的标准方式，而不需要开发人员使用任何命名空间，如th:*。Thymeleaf使这种语法自动提供给所有的方言（不仅仅是标准的方法）。

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W3C Web Components规范的一部分）。 这可以用于例如第th:block元素（或第th-block），这将在后面的章节中进行解释。

重要提示：这个语法是命名空间(th:*)语法的一个补充，它不会替代命名空间语法的形式。 而且在将来，也根本不会放弃命名空间的语法。

6.循环迭代

到目前为止，我们已经创建了一个主页，一个用户个人资料页面，还有一个用户订阅资讯页面，但是我们的产品页面呢？ 为此，我们需要一种循环结构来迭代集合中的产品以此来构建我们的产品页面。

6.1 循环的基本语法

我们创建一个产品列表页面（/WEB-INF/templates/product/list.html），通过表格显示产品的信息。 每个产品将被显示在一行（一个<tr>元素）中，因此对于我们的模板，我们将需要创建一个模板行 - 这将会展示我们想要显示每个产品的方式，然后告诉Thymeleaf 对每个产品重复一次。

Thyme Leaf标准方言为我们提供了一个属性：th:each。

使用 th:each语法

对于我们的产品列表页面，我们将需要一个控制器方法，从Service层获取产品列表数据，并将数据添加到模板上下文中：

```
public void process(
        final HttpServletRequest request, final HttpServle
tResponse response,
        final ServletContext servletContext, final ITempla
teEngine templateEngine)
        throws Exception {

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

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

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

}
```

然后我们将使用th:each在我们的模板中遍历产品列表：

```
<!DOCTYPE html>

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

  <head>
    <title>Good Thymes Virtual Grocery</title>
    <meta http-equiv="Content-Type" content="text/html; ch
arset=UTF-8" />
    <link rel="stylesheet" type="text/css" media="all"
          href="../../../css/gtvg.css" th:href="@{/css/gtv
g.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>
```

在上面代码中，prod：$ {prods}属性值意味着“对于$ {prods}的结果中的每个元素，循环迭代当前模板片段，并使用名为”prod“的变量中作为当前迭代元素来填充模版数据。 让我们给迭代过程中的每个部分赋予一个名字：

- 我们称$ {prods}迭代表达式或被迭代变量。
- 我们称prod为迭代变量或简单的iter变量。

请注意，prod iter变量的作用域为<tr>元素，这意味着它可用于内部标记，如<td>。

被迭代变量的值类型

java.util.List类型不是可以在Thymeleaf中使用迭代的唯一值类型。下面这些类型的对象都是可以通过th:each进行迭代的：

- 任何实现java.util.Iterable接口的对象
- 任何实现java.util.Enumeration接口的对象。
- 任何实现java.util.Iterator接口的对象，其值将被迭代器返回，而不需要在内存中缓存所有值。
- 任何实现java.util.Map的接口对象。 迭代映射时，iter变量将是java.util.Map.Entry类。
- 任何数组。
- 任何其将被视为包含对象本身的单值列表。

6.2 保存迭代状态

当使用th:each时，Thymeleaf提供了一种用于跟踪迭代状态的机制：状态变量。

状态变量在每个th:each属性中定义，并包含以下数据：

- 当前迭代索引，从0开始。这是index属性。
- 当前的迭代索引，从1开始。这是count属性。
- 迭代变量中元素的总量。 这是size属性。
- 每次迭代的iter变量。 这是current属性。
- 当前的迭代是偶数还是奇数。 这些是even/odd布尔属性。
- 当前的迭代是否是第一个迭代。 这是first布尔属性。
- 当前的迭代是否是最后一个迭代。 这是last布尔属性。

我们通过前面的例子来看看迭代状态变量的用法：

```
<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
  </tr>
  <tr th:each="prod,iterStat : ${prods}" th:class="${iterS
tat.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>
```

迭代状态变量（本示例中的iterStat）在th:each属性中通过在iter变量本身之后直接写其名称来定义，用逗号分隔。 就像iter变量一样，状态变量的作用范围也是th:each属性的标签定义的代码片段中。

上面的模版代码执行结果如下：

```
<!DOCTYPE html>

<html>

  <head>
    <title>Good Thymes Virtual Grocery</title>
    <meta content="text/html; charset=UTF-8" http-equiv="C
ontent-Type"/>
    <link rel="stylesheet" type="text/css" media="all" hre
f="/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已经完美运行，仅将奇数CSS类建立到奇数行。

如果没有显式地设置状态变量，则Thymeleaf将始终为您创建一个默认的迭代变量，该状态迭代变量名称为：迭代变量+“Stat”:

```
<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 通过数据懒加载进行迭代优化

有时，我们可能希望延迟数据加载（例如从数据库），以便只有在真正使用这些数据集时才会加载数据。

> 实际上，任何数据都可以采用数据懒加载机制，但是考虑到迭代数据集占用内存中的大小，通常延迟加载迭代的集合是最常见的情况。

Thymeleaf提供了一种延迟加载上下文变量的机制。 实现ILazyContextVariable接口的上下文变量 - 很可能通过扩展其LazyContextVariable默认实现 - 将在执行时加载数据。 例如：

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

这个变量可以在不知道其懒惰的情况下使用，代码如下：

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

但是同时，如果条件表达式值为false，被迭代变量将永远不会被初始化（它的loadValue（）方法永远不会被调用），例如：

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

7.条件判断

7.1 简单条件判断：if和unless

有时，如果满足某个条件，才将一个模板片段显示在结果中。

例如，假设我们想在产品表中显示一列，其中包含每个产品的评论数，如果有评论，则指向该产品的评论页面的链接。

为了做到这一点，我们将使用th:if属性：

```
<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
    <th>COMMENTS</th>
  </tr>
  <tr th:each="prod : ${prods}" th:class="${prodStat.odd}?
 'odd'">
    <td th:text="${prod.name}">Onions</td>
    <td th:text="${prod.price}">2.41</td>
    <td th:text="${prod.inStock}? #{true} : #{false}">yes<
/td>
    <td>
      <span th:text="${#lists.size(prod.comments)}">2</spa
n> comment/s
      <a href="comments.html"
         th:href="@{/product/comments(prodId=${prod.id})}"

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

在这里可以看到很多东西，所以让我们来关注重点：

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

只有产品有评论信息时才会创建一个评论页面链接（URL：/product/comments），prodId参数为产品的ID。

我们来看看结果的标记：

```
<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>
```

完美！这正是我们想要的。

请注意，th:if属性不仅只以布尔值作为判断条件。 它的功能有点超出这一点，它将按照这些规则判定指定的表达式值为true：

- 如果表达式的值不为null：
- 如果值为布尔值，则为true。
- 如果值是数字，并且不为零
- 如果值是一个字符且不为零
- 如果value是String，而不是“false”，“off”或“no”
- 如果值不是布尔值，数字，字符或字符串。

(如果表达式的值为null,th:if将判定此表达式为false)

此外，th:if还有一个反向属性，th:unless，我们可以在前面的例子中使用，而不是在OGNL表达式中使用：

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

7.2 switch语句

还有一种使用Java中等效的switch语句结构有条件地显示内容的方法：th:switch / th:case。

```
<div th:switch="${user.role}">
  <p th:case="'admin'">User is an administrator</p>
  <p th:case="#{roles.manager}">User is a manager</p>
</div>
```

请注意，只要第一个th:case的值为true，则同一个switch语句中的其他th:case属性将被视为false。

switch语句的default选项指定为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>
```

8.2 可参数化的片段签名

为了使模板片段具有多类似函数的功能，用th:fragment定义的片段可以指定一组参数：

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

可以通过以下两种语法中的一种来引用上述模版片段，下面的th:replace改成th:insert也是一样的：

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

请注意，在上述第二种语法中，可以改变参数顺序：

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

不带片段参数的片段局部变量

即使片段没有定义参数：

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

我们可以使用上面指定的第二个语法来调用它们（只有第二个语法才可以）：

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

上面的代码等价于th:replace和th:with的组合：

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

请注意，片段的局部变量规范 - 无论是否具有参数签名 - 都不会导致上下文在执行之前被清空。 片段仍然能够访问调用模板中正在使用的每个上下文变量。

用th:assert进行模版内部断言

th:assert属性可以指定逗号分隔的表达式列表，如果每个表达式的结果都为true，则正确执行，否则引发异常。

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

这有助于验证片段签名中的参数：

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

8 模版布局

8.1 包含模板片段

定义和引用模版片段

在我们的模板中，我们经常希望从其他模板中包含一些部分，如页脚，页眉，菜单等部分

为了做到这一点，Thymeleaf需要我们定义这些部分“片段”，以供其他模版包含，可以使用th:fragment属性来定义被包含的模版片段。

假设我们要向所有杂货页面添加标准版权页脚，因此我们创建一个包含以下代码的/WEB-INF/templates/footer.html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copy的片段，我们可以使用th:insert或th:replace属性（以及th:include，尽管Thymeleaf 3.0不再推荐使用它），容易地包含在我们的主页中：

```
<body>

  ...

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

</body>
```

请注意，th:insert期望一个片段表达式（～{...}）。 在上面的例子中，这是一个简单的片段表达式，（～{，}）包围是完全可选的，所以上面的代码将等价于：

```
<body>

  ...

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

</body>
```

片段规范语法

片段表达式的语法是非常简单的。 有三种不同的格式：

- “～{templatename :: selector}” 包含在名称为templatename的模板上应用指定的标签选择器匹配的片段。 请注意，选择器可以只是一个片段名称，因此您可以像～{footer :: copy}中的～{templatename :: fragmentname}指定一些简单的东西。

> 标记选择器语法由底层的AttoParser解析库定义，类似于XPath表达式或CSS选择器。 有关详细信息，请参阅附录C.

- “～{templatename}” 包含名为templatename的整个模板。

> 请注意，您在th:insert/th:replace标签中使用的模板名称必须由Template Engine当前正在使用的Template Resolver可解析。

- ～{:: selector}“或”～{this :: selector}“ 包含在同一模板中的匹配指定选择器的片段

上述示例中的模板名和选择器都可以是表达式（甚至是条件表达式！），如：

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

再次注意，在th:insert/th:replace中,～{...}是否可选。

片段可以包含任何th:*属性。 一旦将片段包含在目标模板（具有th:insert / th:replace属性的模板）中，这些属性将被计算属性表达式的值，并且它们将能够引用此目标模板中定义的任何上下文变量。

> 这种分片方法的一大优点是，您可以将页面中的片段写入单独的文件，并且该文件具有完整而有效的标记结构，还能在浏览器中完美的显示该片段页面，同时允许Thymeleaf包含在其他模板中。

不用th:fragment引用片段

由于标签选择器的强大功能，我们可以包含不使用th:fragment属性定义的片段。 甚至可以使用没有使用Thymeleaf模版的应用程序的标记代码：

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

```
</div>
...
```

我们可以通过ID属性来引用上面的片段，类似于CSS选择器：

```
<body>

  ...

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

</body>
```

th:insert和th:replace(th:include)之间的区别

th:insert和th:replace之间有什么区别?(th:include在3.0之后不推荐使用了)?

- th:insert是最简单的：它将简单地插入指定宿主标签的标签体中。
- th:replace实际上用指定的片段替换其宿主标签。
- th:include类似于th:insert，而不是插入片段，它只插入此片段的内容。

所以这样一个HTML片段：

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

...在宿主 标签中包含三次上述片段，如下所示：

```
<body>

  ...
```

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

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

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

</body>
```

结果如下：

```
<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>
```

8.3 灵活的布局：不仅仅是片段插入

由于片段表达式的强大功能，我们不仅可以为片段指定文本类型，数字类型，bean对象类型的参数，还可以指定标记片段作为参数。

这允许我们以一种方式创建我们的片段，使得它们可以调用模板的标记，从而产生非常灵活的模板布局机制。

请注意在下面的片段中使用标题和链接变量：

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

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

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

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

</head>
```

我们现在调用这个片段：

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

```
  <title>Awesome - Main</title>

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

</head>
...
```

...，结果将使用调用模板中的实际<title>和<link>标签作为标题和链接变量的值，导致我们的片段在插入期间被定制化：

```
...
<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/code
base.js"></script>

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

</head>
...
```

使用空片段

一个特殊的片段表达式，空的片段（～{}）可以用于指定没有标记。 使用前面的例子：

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

  <title>Awesome - Main</title>

</head>
...
```

注意片段（链接）的第二个参数如何设置为空片段，因此没有为<th:block th:replace =“$ {links}”/>块写入：

```
...
<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/code
base.js"></script>

</head>
...
```

使用哑操作符

如果我们只想让我们的片段将其当前标记用作默认值，那么no-op也可以用作片段的参数。 再次，使用common_header示例：

```
...
<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/jqu
ery-ui.css}">

</head>
...
```

看看title参数（common_header片段的第一个参数）如何设置为no-op（_），这导致片段的这一部分不被执行（title = no-operation）：

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

结果如下：

```
...
<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/code
base.js"></script>

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

</head>
...
```

高级条件插入片段

空片段和无操作片段允许我们以非常简单和优雅的方式执行片段的条件插入。

例如，我们可以这样做，以便只有当用户是管理员时插入我们的common :: adminhead片段，如果不是管理员插入空片段：

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

另外，只有满足指定的条件，我们才可以使用哑操作符来插入片段，但是如果不满足条件，则不需要修改则保留标记：

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

```
...
```

另外，如果我们配置了我们的模板解析器来检查模板资源的存在 - 通过它们的checkExistence标志，我们可以使用片段本身的存在作为默认操作的条件：

```
...
<!-- 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 删除模版片段

回到我们的示例应用程序，让我们回顾一下我们的产品列表模板的最后一个版本：

```
<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
    <th>COMMENTS</th>
  </tr>
  <tr th:each="prod : ${prods}" th:class="${prodStat.odd}?
 'odd'">
    <td th:text="${prod.name}">Onions</td>
    <td th:text="${prod.price}">2.41</td>
    <td th:text="${prod.inStock}? #{true} : #{false}">yes<
/td>
    <td>
      <span th:text="${#lists.size(prod.comments)}">2</spa
n> comment/s
      <a href="comments.html"
         th:href="@{/product/comments(prodId=${prod.id})}"

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

这个代码只是一个模板，但是作为一个静态页面（当没有Thymeleaf的浏览器直接打开它时），它不会成为一个很好的原型。

为什么？ 因为尽管浏览器完全可以显示，该表只有一行，而这行有模拟数据。 作为一个原型，它根本看起来不够现实，我们应该有多个产品，我们需要更多的行。

所以我们来补充一下：

```
<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</spa
n> 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>
```

好的，现在我们有三个产品，肯定是一个好的原型。 但是，当我们用Thymeleaf处理它会发生什么?

```
<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>
```

最后两行是模拟数据行！ 那么当然它们是：迭代只适用于第一行，所以没有什么理由为什么Thymeleaf应该删除另外两个。

我们需要一种在模板处理过程中删除这两行的方法。 我们使用第二和第三个<tr>标签中的th:remove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</spa
n> comment/s
      <a href="comments.html"
         th:href="@{/product/comments(prodId=${prod.id})}"

         th:unless="${#lists.isEmpty(prod.comments)}">view
</a>
    </td>
  </tr>
  <tr class="odd" th:remove="all">
    <td>Blue Lettuce</td>
```

```
      <td>9.55</td>
      <td>no</td>
      <td>
        <span>0</span> comment/s
      </td>
    </tr>
    <tr th:remove="all">
      <td>Mild Cinnamon</td>
      <td>1.99</td>
      <td>yes</td>
      <td>
        <span>3</span> comment/s
        <a href="comments.html">view</a>
      </td>
    </tr>
  </table>
```

一旦处理，一切看起来应该是：

```
<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>
```

这个属性中的all 值是什么意思？ th:remove可以有五种不同的删除方式，具体取决于它的值：

- all：删除包含标签及其所有子项。
- body：不要删除包含的标签，但删除所有的子项。
- tag：删除包含的标签，但不要删除其子项。
- all-but-first：删除除第一个包含标签之外的所有子代。
- none：什么都不做。 该值对于动态计算是有用的。

all-but-first有什么用呢？ 它可以实现保留删除原型设计：

```
<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="${prod.inStock}? #{true} : #{false}">ye
s</td>
      <td>
        <span th:text="${#lists.size(prod.comments)}">2</s
pan> comment/s
        <a href="comments.html"
           th:href="@{/product/comments(prodId=${prod.id})
}"
           th:unless="${#lists.isEmpty(prod.comments)}">vi
ew</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>
```

只要它返回一个允许的字符串值（all，tag，body，all-but-first或none），则th:remove属性可以使用任何Thymeleaf标准表达式。

这意味着删除可能是有条件的，如：

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

另请注意，th:remove将null视为none，因此以下工作与上述示例相同：

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

在这种情况下，如果$ {condition}为false，则返回null，因此不会执行删除。

9.局部变量

Thymeleaf中的局部变量是指定义在模版片段中的变量，并且该变量的作用域为所在的模版片段。

在我们的产品列表页面中的prod 迭代变量就是一个局部变量：

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

该prod变量仅在<tr>标签的范围内可用。 特别：

- 它可用于在该标签中执行的优先级低于th:each的任何其他th:*属性（这意味着它们将在th:each之后执行）。
- 它可用于<tr>标签的任何子元素，例如任何<td>元素。

Thymeleaf为您提供了一种声明局部变量的方式，它使用th:with属性，其语法与属性值分配类似：

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

当th:with被处理时，firstPer变量被创建为局部变量并被添加到来自上下文map中，以便它可以与上下文中声明的任何其他变量一起使用，但只能在包含在 标签内使用。

您可以使用通常的多重赋值语法同时定义多个变量：

```
<div th:with="firstPer=${persons[0]},secondPer=${persons[1
]}">
  <p>
    The name of the first person is <span th:text="${first
Per.name}">Julius Caesar</span>.
  </p>
  <p>
    But the name of the second person is
    <span th:text="${secondPer.name}">Marcus Antonius</spa
n>.
  </p>
</div>
```

th:with属性允许重用在同一属性中定义的变量：

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

在我们的杂货店的主页上使用一下如何定义局部变量吧！ 还记得我们为输出格式化日期写的代码?

```
<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“dd MMMM yyyy”根据地区而改变呢？ 例如，我们可能需要将以下消息添加到home_en.properties中：

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

和我们的home_es.properties等价的一个：

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

现在，我们使用th:with将本地化的日期格式变成一个变量，然后在我们的th:text表达式中使用它：

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

那干净而简单。 实际上，考虑到th:with具有比th:text更高的优先级，因此我们可以在span标签中解决这一切：

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

你可能会想：优先级？ 我们还没有说过！ 那么，别担心，因为这正是下一章的内容。

10.属性优先级

当在同一个标签中写入多于一个的th:*属性时会发生什么？ 例如：

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

我们期望th:each属性在th:text之前执行，以便得到我们想要的结果，但是鉴于HTML / XML标准对于某个属性的渲染顺序没有任何定义， 因此Thyme Leaf如果要达到这种效果，必须在属性本身中建立优先机制，以确保这些属性按预期工作。

因此，所有的Thymeleaf属性都定义一个数字优先级，以便确定在标签中执行它们的顺序。 这个优先级清单如下：

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

8	Fragment specification	`th:fragment`
9	Fragment removal	`th:remove`

这个优先机制意味着如果属性位置被反转，上述迭代片段将给出完全相同的结果（尽管它的可读性稍差）：

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

11.4 合成块标签-th:block

Thymeleaf标准方言中唯一的元素处理器（不是属性）是：th:block。

th:block是一个只允许模板开发人员指定他们想要的属性的属性容器。Thymeleaf将执行这些属性，然后简单地制作块。

所以它可能是有用的，例如，当为每个元素创建多个<tr>的迭代表时：

```
<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>
```

当与原型注释块组合使用时尤其有用：

```
<table>
    <!--/*/ <th:block th:each="user : ${users}"> /*/-->
    <tr>
        <td th:text="${user.login}">...</td>
        <td th:text="${user.name}">...</td>
    </tr>
    <tr>
        <td colspan="2" th:text="${user.address}">...</td>
    </tr>
    <!--/*/ </th:block> /*/-->
</table>
```

注意这个标签解决了HTML标签的合法验证问题（不需要在<table>中添加禁止的 块），并且仍然可以在浏览器中作为原型静态打开！

11.1 标准HTML/XML注释

标准HTML/XML注释<!- ... - >可以在Thymeleaf模板中的任何地方使用。这些注释中的任何内容都不会被Thymeleaf处理，并将逐字复制到结果中：

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

11.2 ThymeLeaf解析器级注释

解析器级注释块是当Thymeleaf解析时将简单地从模板中删除的代码。 他们看起来像这样：

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

Thymeleaf将删除<！ - / *和* / - >之间的所有内容，因此当模板静态打开时，这些注释块也可用于显示代码，因为当Thymeleaf处理它时，它将被删除：

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

对于具有很多<tr>的设计原型表，这可能会非常方便，例如：

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

```
</table>
```

11.3 Thymeleaf专有注释

当模板被静态打开（即作为原型）时，Thymeleaf允许定义特殊注释块，并且在执行模板时，Thymeleaf被认为是正常的标记。

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

Thymeleaf的解析系统将简单地删除<！ - / * /和/ * / - >标记，但不删除包含其中的内容，因此标记之间的内容将被保留。 所以当执行模板时，Thymeleaf实际上会看到这样的：

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

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

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

与解析器级注释块一样，此功能与方言无关。

12.1 内联表达式

虽然通过Thymeleaf标准方言中的标签属性已经几乎满足了我们开发中的所有需求，但是有些情况下我们更喜欢将表达式直接写入我们的HTML文本。 例如，我们喜欢这样写代码：

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

而不喜欢这样写代码：

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

[[...]]或[(...)]中的表达式被认为是在Thymeleaf中内联的表达式，任何在th:text或th:utext属性中使用的表达式都可以出现在[[]]或[()]中。

请注意，虽然[[...]]等价于th:text（即结果将被HTML转义），[(...)]等价于th:utext，不会执行任何HTML转义。 所以有一个变量，如msg ='这是<b>很棒！</ b>'，给定这个片段：

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

结果将使这些<b>标签未转义，所以：

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

而如果需要转义就这样写：

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

结果将被HTML转义：

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

请注意，默认情况下，内联文本在我们的标签体中都是活动的，因此我们无法做到这一点。（这里有问题，是在不解）

内联与自然模板比较

如果您使用过以常规方式输出文本的其他模板引擎，您可能会问：为什么我们从一开始不直接使用内联表达式输出文本呢？ 它比th:text方式的代码少很多！

请注意，因为虽然你可能会发现内联非常有趣，但是您应该永远记住，当您静态打开HTML文件时，内嵌的表达式将逐字显示在HTML文件中，因此您可能无法将其用作设计原型了！

不使用内联表达式时浏览器静态显示我们的代码段如下：

```
Hello, Sebastian!
```

而使用内联表达式时浏览器静态显示我们的代码如下：

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

...这在设计有用性方面的差别就很清楚了。

禁用内联

内联机制可以被禁用，因为在实际应用中可能会出现我们想输出[[...]]或[（...）]序列而不将其内容作为表达式处理的情况。 为此，我们将使用th:inline =“none”来禁用内联：

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

结果如下：

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

12.2 内联文本

内联文本非常类似于我们刚刚看到的内联表达式功能，但实际上它的功能更加强大。 内联文本必须显式地使用th:inline =“text”。

内联文本不仅允许我们使用内嵌表达式，而且还可以处理标签体，就像它们是以TEXT模板模式处理的模板一样，这允许我们执行基于文本的模板逻辑（不仅仅是输出表达式）。

我们将在下一章中看到关于文本模板模式的更多信息。

12.3 内联JavaScript

JavaScript内联允许在HTML模板模式中更好地集成JavaScript <script> 块。

与文本内联一样，这实际上等同于处理脚本内容，就像它们是JAVASCRIPT模板模式中的模板一样，因此文本模板模式的所有功能（见下一章）都可以在内联脚本中使用。 然而，在本节中，我们将重点介绍如何使用它将我们的Thymeleaf表达式的输出添加到我们的JavaScript块中。

必须使用th:inline =“javascript”显式启用此模式：

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

结果如下：

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

以上代码中要两个重要的注意事项：

首先，JavaScript内联函数不仅会输出所需的文本，还可以使用引号和JavaScript来包含其中的内容，以便将表达式结果作为一个格式良好的JavaScript字符串输出。

其次，这是因为我们输出$ {session.user.name}表达式时进行了转义，即使用双括号表达式：[[$ {session.user.name}]]。 如果我们使不进行转义，如：

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

结果将时这样的：

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

...这是格式错误的JavaScript代码。 但是，如果我们通过附加内联表达式来构建脚本的一部分，那么输出一些未转义的字符串可能就是我们需要的，所以这个功能也很有用。

JavaScript自然模版

JavaScript的内联机制的不仅能将JavaScript特定的字符串转义输出，而且在处理表达式时更加智能化。

例如，我们可以在JavaScript注释中包含（转义）内联表达式，如：

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

```
iwifruit";
    ...
</script>
```

而Thymeleaf将忽略在注释之后和分号之前写的所有内容（本例中为“Gertrud Kiwifruit”），因此执行此操作的结果将与我们不使用包装注释时完全相同：

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

但是请仔细查看原始模板代码：

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

注意这是有效的JavaScript代码。 当您以静态方式打开模板文件（不在服务器上执行）时，它将完全执行。

所以我们称这种方法为JavaScript自然模板！

高级内联表达式和JavaScript序列化

关于JavaScript内联的一个重要的特性是，内联表达式的计算结果不限于字符串，它能自动的将以下对象序列化为javascript对象。 Thymeleaf支持以下几种序列化对象：

- Strings
- Numbers
- Booleans
- Arrays
- Collections
- Maps
- Beans (有*getter _and _setter*方法)

例如，我们有以下代码：

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

$ {session.user}表达式将计算为输出User对象，Thymeleaf将正确地将其转换为JavaScript对象：

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

JavaScript序列化的方式是通过org.thymeleaf.standard.serializer.IStandardJavaScriptSerializer接口的实现，可以在模板引擎的StandardDialect的实例中进行配置。

该JavaScript序列化机制的默认实现将在类路径中查找Jackson库，如果存在，将使用它。 如果没有，它将应用一个内置的序列化机制，内置的序列化机制涵盖大多数场景的需求，并和Jackson序列化机制产生类似的结果（但这种方法不太灵活）。

12.4 内联CSS

Thymeleaf还允许在CSS <style>标签中使用内联，如：

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

例如，假设我们有两个变量设置为两个不同的String值：

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

我们可以这样做：

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

输出结果如下：

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

请注意，CSS内联是否也具有像JavaScript内联一样的智能特性。 具体来说，通过转义表达式输出的表达式，如[[$ {classname}]]将被转义为CSS标识符。 这就是为什么我们的classname ='main elems'已经变成上面的代码片段中的main \ elems。

高级功能：CSS自然模板等

与内联JavaScript一样，CSS内联也允许我们的<style>标签可以静态和动态地工作，即通过在注释中包含内联表达式作为CSS自然模板。 看到：

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

13.文本模板模式

Thymeleaf模板中的三个模板被认为是文本：TEXT，JAVASCRIPT和CSS。 这将它们与标记模板模式区分开来：HTML和XML。

文本模板模式和标记模式之间的关键区别在于，在文本模板中，没有标签以属性的形式插入逻辑，因此我们必须依赖其他机制。

最基本的机制是内联，我们在前一章已经详细介绍了这些内联机制。 内联语法是以文本模板模式输出表达式结果的最简单方法，因此邮件形式模板来说，内联是最好的机制。

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

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

  Sincerely,
    The Reporter.
```

即使没有标签，上面的示例也是一个完整有效的Thymeleaf模板，可以在TEXT模板模式下执行。

但是为了包含比单纯的输出表达式更复杂的逻辑，我们需要一个新的基于非标记的语法：

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

其实上述代码是下面的简写形式：

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

请注意，这种新语法是基于被声明为[#element ...]而不是<element ...>的元素（即可处理标签）。 元素是像[#element ...]这样打开并像[/element]这样关闭，独立的标签可以通过使用/以几乎相当于XML标签的方法最小化开放元素：[#element ... /]。

Thymeleaf标准方言只包含以下这些元素之一的处理器：已知的th:block，尽管我们可以通过自定义方言来扩展它，并以常规的方式创建新的元素。此外，第th:block（[#th：block ...] ... [/ th：block]）被允许缩写为空字符串（[＃...] ... [/]），所以上面的块实际上等于：

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

并且给定[＃th：utext =“$ {item}”/]等效于一个内联的非转义表达式，我们可以使用它来减少代码。 因此，我们结束了上面看到的第一个代码片段：

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

请注意，文本语法需要完整的元素平衡（没有未封闭的标签）和引用的属性 - 它更像XML风格。

我们来看看一个更完整的TEXT模板示例，一个纯文本的电子邮件模板：

```
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，其结果是：

```
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
```

另一个例子在JAVASCRIPT模板模式，一个greeter.js文件，我们作为一个文本模板进行处理，我们从HTML页面调用哪个结果。 请注意，这不是HTML模板中的<script>块，而是.js文件作为模板自行处理：

```
var greeter = function() {

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

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

```
};
```

执行后，其结果是：

```
var greeter = function() {

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

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

};
```

转义的元素属性

为了避免与其他模式（例如，HTML模板内的文本模式内联）处理的模板部分的交互，Thymeleaf 3.0允许转义其文本语法中的元素中的属性。 所以：

- TEXT模板模式下的属性将被HTML转义。
- JAVASCRIPT模板模式中的属性将是JavaScript转义的。
- CSS模板模式中的属性将被CSS转义。

所以在TEXT模式模板（注意＆gt;）中这样做完全可以：

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

当然， 在真实的文本模板中没有意义，但是如果我们正在使用包含上述代码的th:inline =“text”块处理HTML模板是一个好主意，我们希望确保我们的浏览器将文件静态打开时，打开标签的名称为<user.age。

13.2 扩展性

这种语法的一个优点是它与标记一样可扩展。 开发人员仍然可以使用自定义元素和属性来定义自己的方言，为它们应用前缀（可选），然后在文本模板模式下使用它们：

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

13.3 文本原型注释块：添加代码

JAVASCRIPT和CSS模板模式（不适用于TEXT）允许在特殊注释语法/*[+...+]*/之间包含代码，以便Thymeleaf在处理模板时会自动取消对这些代码的注释：

```
var x = 23;

/*[+

var msg  = "This is a working application";

+]*/

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

将被执行为：

```
var x = 23;

var msg  = "This is a working application";

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

您可以在这些注释中包含表达式，并对它们进行计算：

```
var x = 23;

/*[+
```

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

+]*/

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

13.4 文本解析器级注释块：删除代码

Thymeleaf中，以类似于原型的注释块的方式，所有三种文本模板模式（TEXT，JAVASCRIPT和CSS）都可以使用删除特殊/ * [- * /和/ * -] * /标记，像这样：

```
var x = 23;

/*[- */

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

/* -]*/

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

或者，在TEXT模式下：

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

13.5 自然的JavaScript和CSS模板

如前一章所示，JavaScript和CSS内联提供了在JavaScript / CSS注释中包含内置表达式的可能性，如：

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

...这是有效的JavaScript，执行结果如下：

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

在注释中封闭内联表达式的同样技巧实际上可用于整个文本模式语法：

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

当模板静态打开时，上述代码会报警告 - 因为它是100%有效的JavaScript - 以及当用户是admin时运行模板。 相当于：

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

...这实际上是在模板解析期间转换初始版本的代码。

请注意，注释中的包含的元素不会像内联的输出表达式那样清除它们所在的行（向右到直到找到）。 该行为仅用于内联输出表达式。

所以Thymeleaf 3.0允许以自然模板的形式开发复杂的JavaScript脚本和CSS样式表，既可以作为原型，也可以作为工作模板。

14.杂货店示例项目其他页面

现在我们已经掌握了很多关于如何使用Thymeleaf的知识，我们可以在我们的网站上添加一些新的页面来进行订单管理。

请注意，我们将专注于HTML代码，但如果要查看相应的控制器，可以查看相关的源代码。

14.1 订单列表页面

首先我们创建一个订单列表页面，/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; ch
arset=UTF-8" />
    <link rel="stylesheet" type="text/css" media="all"
          href="../../../css/gtvg.css" th:href="@{/css/gtv
g.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="${#calendars.format(o.date,'dd/MMM/yy
yy')}">13 jan 2011</td>
        <td th:text="${o.customer.name}">Frederic Tomato</
```

```
td>
        <td th:text="${#aggregates.sum(o.orderLines.{purch
asePrice * amount})}">23.32</td>
        <td>
          <a href="details.html" th:href="@{/order/details
(orderId=${o.id})}">view</a>
        </td>
      </tr>
    </table>

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

  </body>

</html>
```

这里没有什么应该让我们惊讶，除了这一点的OGNL表达式语法：

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

对于订单中的每个订单行（OrderLine对象），将其purchasePrice和amount属性（通过调用相应的getPurchasePrice（）和getAmount（）方法）相乘并将结果返回到数字列表中，然后由 # aggregates.sum（...）函数以获得订单总价。

你必须爱上OGNL的力量。

14.2 订单详情页面

在订单详细信息页面，我们将大量使用星号语法：

```
<!DOCTYPE html>

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

  <head>
    <title>Good Thymes Virtual Grocery</title>
    <meta http-equiv="Content-Type" content="text/html; ch
arset=UTF-8" />
    <link rel="stylesheet" type="text/css" media="all"
          href="../../../css/gtvg.css" th:href="@{/css/gtv
g.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 yy
yy')}">13 jan 2011</span>
      </p>
    </div>

    <h2>Customer</h2>

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

```
ato</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">2
3.32</td>
      </tr>
    </table>

    <div>
      <b>TOTAL:</b>
      <span th:text="*{#aggregates.sum(orderLines.{purchas
ePrice * amount})}">35.23</span>
    </div>

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

```
  </body>

</html>
```

这里没有太多的新意，除了这个嵌套的对象选择：

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

  ...

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

  ...
</body>
```

...这使得*{name}等价于：

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

15 更多配置

15.1 模版解析器

在我们的杂货店项目中，我们配置了一个称为ServletContextTemplateResolver的类作为模版解析器实现类，该类实现类ITemplateResolver接口，它允许我们从Servlet上下文中获取板文件资源。

除了通过实现ITemplateResolver接口来自定义模板解析器之外，Thymeleaf还提供四个开箱即用的解析器实现类：

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

```
return Thread.currentThread().getContextClassLoader().getR
esourceAsStream(template);
```

- org.thymeleaf.templateresolver.FileTemplateResolver将文件系统中的模板解析为文件，如：

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

- org.thymeleaf.templateresolver.UrlTemplateResolver，它将模板解析为URL（甚至非本地的），如：

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

org.thymeleaf.templateresolver.StringTemplateResolver，它直接将模版解析为字符串（或模板名称，在这种情况下，这显然不仅仅是一个名称）：

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

ITemplateResolver的上述四个实现类都允许使用相同的配置参数集，其中包括：

- 前缀和后缀（已经看到）：

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

Thymeleaf模板别名允许模版名不直接映射到文件名称。 如果后缀/前缀和别名都存在，则别名优先级更高：

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

- 读取模板时应用的编码：

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

- 设置模版模式：

```
// Default is HTML
templateResolver.setTemplateMode("XML");
```

- 设置模板的默认缓存模式以及定义特定模板是否缓存：

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

- 设置模版解析器的缓存失效时间TTL。 如果未设置，当超过缓存最大存储空间时，则根据LRU算法清理缓存。

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

> Thymeleaf + Spring集成包提供了一个SpringResourceTemplateResolver实现，它使用Spring基础架构来访问和读取应用程序中的资源，这是在Spring的应用程序中推荐的模版解析器。

模版解析器链

此外，模板引擎可以指定多个模板解析器，在这种情况下，可以在它们之间建立模板解析的顺序，以便如果第一个解析器无法解析模板，则会使用第二个模板解析器，依此类推：

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

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

```
);

templateEngine.addTemplateResolver(classLoaderTemplateReso
lver);
templateEngine.addTemplateResolver(servletContextTemplateR
esolver);
```

建议当应用几个模板解析器时，建议为每个模板解析器指定需要解析的模版路径匹配模式，以便Thymeleaf可以快速丢弃那些不是本模板解析器所处理的模版资源，从而提高性能。

```
ClassLoaderTemplateResolver classLoaderTemplateResolver =
new ClassLoaderTemplateResolver();
classLoaderTemplateResolver.setOrder(Integer.valueOf(1));
// This classloader will not be even asked for any templat
es not matching these patterns
classLoaderTemplateResolver.getResolvablePatternSpec().add
Pattern("/layout/*.html");
classLoaderTemplateResolver.getResolvablePatternSpec().add
Pattern("/menu/*.html");

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

如果未指定可解析的模版路径匹配模式，我们将依赖于正在使用的每个ITemplateResolver的具体实现。 请注意，并不是所有的解析器都可以在解析之前确定模板的存在，因此如果将模板视为总是可解析的，但最终又无法解析时，将会导致解析链被破坏（不允许其他解析器检查相同的模板）。

Thymeleaf核心中的所有ITemplateResolver实现都提供一种机制，使我们能够器在解析之前检查资源是否存在。 这个机制通过checkExistence标志位来实现，其用法如下：

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

此checkExistence标志强制解析器在解析阶段期间判断资源是否存在（如果返回false，则调用链中的后续解析器）。 尽管这种做法听起来不错，但在大多数情况下，这意味着对资源本身的双重访问（一次用于检查存在，另一次用于读取它），因此在某些情况下将导致性能问题，例如。解析一个基于远程URL的模板资源时就存在一个潜在的性能问题，这种情况下会通过使用模板缓存（在这种情况下，模板首次被访问时才能被解析）得到很大的缓解。

15.2 消息解析器

我们没有为我们的杂货店应用程序显式指定Message Resolver实现，这意味着正在使用org.thymeleaf.messageresolver.StandardMessageResolver作为默认的消息解析器。

StandardMessageResolver是IMessageResolver接口的标准实现，但我们自定义消息解析器。

> 默认情况下，Thymeleaf + Spring集成包提供了使用标准Spring方式检索外部化消息的IMessageResolver实现，通过使用Spring应用程序上下文中声明的MessageSource bean来实现。

标准消息解析器

那么StandardMessageResolver如何寻找在特定模板上所需要的消息呢?

如果模板名称是home，并且位于/WEB-INF/templates/home.html中，并且所请求的区域设置为gl_ES，则此解析器将按以下顺序在以下文件中查找：

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

有关完整的消息解析机制的更多详细信息，请参阅JavaDoc文档中的StandardMessageResolver类。

配置消息解析器

如果我们要向模板引擎添加消息解析器（或更多），该怎么办？ 简单：

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

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

为什么我们要配置多个消息解析器？ 这与配置多个模板解析器类似：消息解析器被排序，如果第一个无法解析特定消息，则会询问第二个消息，然后是第三个，依此类推。

15.3 转换服务

Thymeleaf通过双重大括号括号语法（$\{\{...\}\}）执行数据转换和格式化操作的功能是由Thymeleaf标准方言来实现的，该功能并不是Thymeleaf模板引擎来实现的。

因此，配置它的方式是将自定义IStandardConversionService接口的实现类对象直接设置为模板引擎中的StandardDialect实例。配置方法如下：

```
IStandardConversionService customConversionService = ...

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

templateEngine.setDialect(dialect);
```


```
IStandardConversionService customConversionService = ...

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

templateEngine.setDialect(dialect);
```


> 请注意，thymeleaf-spring3和thymeleaf-spring4包含SpringStandardDialect，并且该方言已经预先配置了将Spring自身的转换服务基础架构集成到Thymeleaf中的IStandardConversionService实现。

15.4 日志

Thymeleaf对日志记录非常重视，并且始终通过其日志接口提供最大量的有用信息。

Thymeleaf使用slf4j作为日志门面，实际上它可以作为我们应用程序中使用的日志记录实现的桥梁（例如log4j）。

Thymeleaf类将支持TRACE，DEBUG和INFO级别的信息，具体取决于我们需要的详细程度，除了通用日志之外，Thymeleaf还将使用与TemplateEngine类相关联的三个特殊日志类，我们可以单独配置用于不同的目的：

- org.thymeleaf.TemplateEngine.CONFIG将在初始化期间输出库的详细配置。
- org.thymeleaf.TemplateEngine.TIMER将输出有关处理每个模板所用时间的信息（对基准测试有用）！
- org.thymeleaf.TemplateEngine.cache是一组记录器的前缀，用于输出有关缓存的特定信息。 尽管缓存记录器的名称可由用户配置，因此可能会更改，默认情况下它们是：

```
                                    org.thymeleaf.TemplateEn
gine.cache.TEMPLATE\_CACHE

                                    org.thymeleaf.TemplateEn
gine.cache.EXPRESSION\_CACHE
```

使用log4j的Thymeleaf的日志记录基础结构的示例配置可以是：

```
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_C
```

```
ACHE=TRACE
```

16.模版缓存

Thymeleaf的工作过程主要由一组解析器（用于解析标签和文本）和一系列处理器来完成，其中解析器将模板解析为事件序列（标签开始，文本，标签结束，注释等）；处理器将处理解析过程中的各种行为，并且修改模板解析事件序列，以便通过将原始模板与我们的数据组合来创建我们期望的结果。

它还包括 - 默认情况下，存储解析模板的缓存;在处理它们之前从读取和解析模板文件导致的事件序列。这在Web应用程序中工作时特别有用，并且基于以下概念：

- 输入/输出几乎总是任何应用程序中最慢的部分。相比之下，内存中的处理速度非常快。
- 克隆现有的内存中事件序列总是比读取模板文件快一点，解析它并为其创建一个新的事件序列。
- Web应用程序通常只有几十个模板。
- 模板文件是小到中等大小的，并且它们在应用程序运行时不被修改。

这一切都导致了在Web应用程序中缓存最常用的模板是可行的，而不浪费大量的内存，同时也可以节省大量的输入/输出操作时间，事实上，一些文件从来没有改变，不需要重新解析。

我们如何控制这个缓存？首先，我们之前已经学到了，我们可以在模板解析器中启用或禁用它，甚至只针对特定的模板来启用缓存：

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

此外，我们可以通过自己创建CacheManager对象来修改模版缓存配置，该对象可能是默认的StandardCacheManager实现的一个实例：

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

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

可以从模板缓存中手动删除不需要缓存的条目：

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

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

17 模版解耦逻辑

17.1 解耦逻辑：概念

到目前为止，我们已经按照常规的方式在我们的杂货店项目中使用了Thymeleaf模版，而且将我们的业务逻辑以属性的形式插入到我们的模板中。

但是，Thymeleaf还允许我们将模板标签与其要实现的逻辑完全分离，从而允许在HTML和XML模板模式中创建完全无逻辑的标记模板。

模版解耦逻辑的主要思想是将模板逻辑在一个单独的逻辑文件中定义（更准确的说是逻辑资源，因为它不一定是一个文件）。 默认情况下，该逻辑资源将是与模板文件放在同一文件夹中，并且名称相同，但扩展名为.th.xml：

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

所以home.html文件可以是完全无逻辑的。 它可能看起来像这样：

```
<!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>
```

这里没有任何thymeleaf代码。 因此即使是不懂Thymeleaf的设计师也可以创建，编辑和读懂这个模板文件。这个模版文件还可以是一些外部系统提供的HTML片段，即使这些外部系统文件并不是Thymeleaf模版文件。

现在，我们先将home.html模板转换成Thymeleaf模板，方法是创建我们的逻辑资源home.th.xml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>
```

在上述代码中，我们可以看到一个<thlogic>标签,这个标签还包含了很多<attr>标签。 而这些<attr>标签则通过其sel属性来选择原始模板的节点执行属性注入，其中包含Thymeleaf标记选择器（实际上是AttoParser标记选择器）。

还要注意，<attr>标签可以嵌套，以便追加它们的选择器。 例如，上面代码中的sel =“/ tr [0]”将被处理为sel =“ # usersTable / tr [0]”。 而username<td>的选择器将被处理为sel =“ # usersTable / tr [0] // td.username”。

所以一旦合并原始模版和逻辑资源，上面看到的两个文件将是这样的：

```
<!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，并且比创建两个单独的文件确实不那么冗长。 但是模板解耦逻辑的优点是，我们可以使我们的模板完全独立于Thymeleaf，因此从设计的角度来看系统的可维护性更高。

当然，这仍然需要设计师或开发人员之间的一些契约。 用户<table>将需要id =“usersTable”的事实，但是在许多情况下，纯HTML模板将是设计和开发团队之间更好的通信工件。

17.2 配置解耦模版

启用解耦模板

默认情况下，每个模板都没有启用解耦逻辑。 因此，如果需要启用模版解耦，则可以通过配置的模板解析器（ITemplateResolver的实现）来实现。

除了StringTemplateResolver（不允许解耦逻辑）之外，ITemplateResolver的所有其他开箱即用的实现都提供一个名为useDecoupledLogic的标志，该标志将标记由该解析器解析的所有模板可能使其全部或部分逻辑保存在单独的资源文件中：

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

混合使用逻辑耦合和逻辑解耦

解耦模板逻辑不是强制要求的。 启用时，这意味着引擎将查找包含解耦逻辑的资源，解析并将其与原始模板合并（如果存在）。 如果解耦逻辑资源不存在，则不会抛出任何异常。

另外，在同一个模板中，我们可以混合使用逻辑耦合和逻辑解耦，例如通过在原始模板文件中添加一些Thymeleaf属性，而将其他逻辑留给另外的解耦逻辑资源文件。 最常见的情况是使用新的（在v3.0）th:ref属性。

17.3 th:ref属性

th:ref只是一个标记属性。 从解析器处理的角度来看，它什么也不做，当模板被处理时就会消失，但是它的用处在于它作为一个标记引用，即可以通过标记选择器的名称来解析，就像标签名称或片段一样（TH：片段）。

所以如果我们有一个选择器：

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

这将匹配：

- 任何<whatever>标签。
- 任何带有th:fragment =“whatever”属性的标签。
- 任何带有th:ref =“whatever”属性的标签。

th:ref比纯HTML id属性的优点是什么呢？ 因为我们可能不想在标签中添加如此多的id和类属性作为逻辑锚，这可能会导致我们的输出污染。

在同样的意义上，th:ref的缺点是什么？ 那么很明显，我们会在我们的模板中添加一些Thymeleaf逻辑（“逻辑”）。

请注意，th:ref属性不仅适用于解耦逻辑模板文件：它在其他类型的场景中也是一样的，如片段表达式（~{...}）。

17.4 模板解耦逻辑对性能的影响

模板解耦逻辑对性能的影响极小。 当解析的模板被标记为使用解耦逻辑并且不被缓存时，模板逻辑资源将先被解析，并处理成内存中的一系列指令：基本上是要注入到每个标记选择器的属性列表。

但是，这是唯一需要的附加步骤，因为在此之后，真正的模板将被解析，并且在解析时，这些属性将被解析器本身自动注入，这得益于AttoParser中节点选择的高级功能。 所以解析的节点将从解析器中获取，就好像他们注入的属性写在原始模板文件中一样。

这个最大的优点是当模板被配置为缓存时，它将被缓存已经包含注入的属性。 因此，对可缓存模板使用解耦模板的开销，一旦被缓存，将绝对为零。

17.5 模版解耦逻辑的分辨率

Thymeleaf解决与每个模板相对应的解耦逻辑资源的方式可由用户配置。它由扩展点确定，即org.thymeleaf.templateparser.markup.decoupled.IDecoupledTemplateLogicResolver，为其提供了默认实现：StandardDecoupledTemplateLogicResolver。

这个标准实现是做什么的?

- 首先，它为模板资源的基本名称（通过其ITemplateResource # getBaseName()方法获取）应用前缀和后缀。 可以配置前缀和后缀，默认情况下，前缀将为空，后缀将为.th.xml。
- 其次，它要求模板资源通过其ITemplateResource # relative（String relativeLocation）方法来解析具有计算名称的相对资源。

要使用的IDecoupledTemplateLogicResolver的具体实现可以轻松地在TemplateEngine上进行配置：

```
final StandardDecoupledTemplateLogicResolver decoupledreso
lver =
        new StandardDecoupledTemplateLogicResolver();
decoupledResolver.setPrefix("../viewlogic/");
...
templateEngine.setDecoupledTemplateLogicResolver(decoupled
Resolver);
```

18.附录A：基本对象表达式

一些对象和变量映射始终可以被调用。 我们来看看他们：

18.1基本对象

- #ctx：上下文对象。 根据我们的环境（独立环境或Web环境），实现org.thymeleaf.context.IContext或org.thymeleaf.context.IWebContext。

注意#vars和#root是同一对象的不同名称，但建议使用#ctx。

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

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

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

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

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

```
${#locale}
```

18.2 request和session属性的web命名空间

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

> 注意这些不是上下文对象，而是添加到上下文中的Map集合作为变量，所以我们在没有 # 的情况下访问它们。 在某种程度上，它们作为命名空间。

- param：用于获取请求参数。 $ {param.foo}是一个带有foo请求参数值的String []，所以$ {param.foo [0]}通常用于获取第一个值。

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

${param.foo}              // Retrieves a String[] with the
 values of request parameter 'foo'
${param.size()}
${param.isEmpty()}
${param.containsKey('foo')}
...
```

- session：用于获取session属性。

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

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

${session.foo}                 // Retrieves the session at
ttribute 'foo'
${session.size()}
${session.isEmpty()}
${session.containsKey('foo')}
...
```

- application：用于获取应用程序或servlet上下文属性。

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

${application.foo}              // Retrieves the ServletCo
ntext atttribute 'foo'
${application.size()}
${application.isEmpty()}
${application.containsKey('foo')}
...
```

> 请注意，无需为访问请求属性（而不是请求参数）指定命名空间，因为所有请求属性将自动添加到上下文中作为上下文根中的变量：

```
${myRequestAttribute}
```

18.3 web上下文对象

在Web环境中，还可以直接访问以下对象（请注意，这些对象不是Map集合/命名空间）：

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

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

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

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

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

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

19.附录B：工具类对象表达式

Execution Info

- #execInfo：提供有关在Thymeleaf标准表达式内正在处理的模板的信息。

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

/*
 * Return the name and mode of the 'leaf' template. This m
eans the template
 * from where the events being processed were parsed. So i
f 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 re
turn "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 m
```

```
eans the template
 * that the template engine was originally asked to proces
s. 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", thi
s will still
 * return "A" and A's template mode.
 */
${#execInfo.processedTemplateName}
${#execInfo.processedTemplateMode}

/*
 * Return the stacks (actually, List<String> or List<Templ
ateMode>) of
 * templates being processed. The first element will be th
e
 * 'processedTemplate' (the root one), the last one will b
e 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}
```

Messages

- #messages：用于在变量表达式中获取外部化消息的工具方法，与使用#{...}语法获得的方式相同。

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

/*
 * Obtain externalized messages. Can receive a single key,
 a key plus arguments,
 * or an array/list/set of keys (in which case it will ret
urn an array/list/set of
 * externalized messages).
 * If a message is not found, a default message (like '??m
sgKey??') 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[] {pa
ram1, param2, param3, param4})}
${#messages.arrayMsgOrNull(messageKeyArray)}
${#messages.listMsgOrNull(messageKeyList)}
${#messages.setMsgOrNull(messageKeySet)}
```

URIs/URLs

- #uris：用于在Thymeleaf标准表达式中执行URI / URL操作（尤其是转义/取消转义）的工具对象。

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

Conversions

- #conversions：允许在模板任意位置执行转换服务的实用程序对象：

```
/*
```

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

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

Dates

- #dates：java.util.Date对象的实用程序方法：

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

/*
 * 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 arrayMonthN
ame(...), listMonthName(...), etc.
${#dates.monthNameShort(date)}          // also arrayMonthN
ameShort(...), listMonthNameShort(...), etc.
${#dates.year(date)}                    // also arrayYear(.
```

```
..), listYear(...), etc.
${#dates.dayOfWeek(date)}                     // also arrayDayOfW
eek(...), listDayOfWeek(...), etc.
${#dates.dayOfWeekName(date)}                 // also arrayDayOfW
eekName(...), listDayOfWeekName(...), etc.
${#dates.dayOfWeekNameShort(date)}            // also arrayDayOfW
eekNameShort(...), 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 arrayMillis
econd(...), listMillisecond(...), etc.

/*
 * Create date (java.util.Date) objects from its component
s
 */
${#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,millisec
ond)}

/*
 * Create a date (java.util.Date) object for the current d
ate and time
 */
${#dates.createNow()}

${#dates.createNowForTimeZone()}

/*
 * Create a date (java.util.Date) object for the current d
```

```
ate (time set to 00:00)
 */
${#dates.createToday()}

${#dates.createTodayForTimeZone()}
```

Calendars

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

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

/*
 * 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 arrayMonthN
ame(...), listMonthName(...), etc.
${#calendars.monthNameShort(date)}     // also arrayMonthN
ameShort(...), listMonthNameShort(...), etc.
${#calendars.year(date)}               // also arrayYear(.
..), listYear(...), etc.
${#calendars.dayOfWeek(date)}          // also arrayDayOfW
eek(...), listDayOfWeek(...), etc.
${#calendars.dayOfWeekName(date)}      // also arrayDayOfW
eekName(...), listDayOfWeekName(...), etc.
${#calendars.dayOfWeekNameShort(date)} // also arrayDayOfW
eekNameShort(...), 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 arrayMillis
econd(...), listMillisecond(...), etc.

/*
 * Create calendar (java.util.Calendar) objects from its c
omponents
 */
${#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,mill
isecond)}

${#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 c
urrent date and time
 */
${#calendars.createNow()}

${#calendars.createNowForTimeZone()}

/*
 * Create a calendar (java.util.Calendar) object for the c
urrent date (time set to 00:00)
 */
```

```
${#calendars.createToday()}

${#calendars.createTodayForTimeZone()}
```

Numbers

- #numbers：数字对象的实用程序方法：

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


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

Strings

- **#strings String工具类**

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

/*
 * 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 fragmen
t
 * 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-c
ase
 */
${#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)}
```

Objects

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

/*
 * 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)}
```

Booleans

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

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

/*
 * Evaluate a condition in the same way that it would be e
valuated 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)}
```

Arrays

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

/*
 * Converts to array, trying to infer array component clas
s.
 * Note that if resulting array is empty, or if the elemen
ts
 * 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)}
```

Lists

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

/*
 * 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)}
```

Sets

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

Maps

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

聚合函数

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

/*
 * Compute sum. Returns null if array or collection is emp
ty
 */
${#aggregates.sum(array)}
```

```
${#aggregates.sum(collection)}

/*
 * Compute average. Returns null if array or collection is
 empty
 */
${#aggregates.avg(array)}
${#aggregates.avg(collection)}
```

IDs

- #ids：处理可能重复的id属性的实用方法（例如，作为迭代的结果）。

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

/*
 * Normally used in th:id attributes, for appending a coun
ter to the id attribute value
 * so that it remains unique even when involved in an iter
ation 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 t
he 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')}
```

20.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（AttoParser）中借用。

该选择器的语法与XPath，CSS和jQuery中的选择器具有很大的相似性，这使得它们可以方便大多数用户使用。 您可以在AttoParser文档中查看完整的语法参考。

例如，以下选择器将在标记内的每个位置中选择每个类名包含content的 标签（请注意，这不是简洁的，继续读下去就知道为什么了）：

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

基本语法如下：

- /x表示名为x的当前节点的直接子节点。
- //x表示任何深度的名为x的当前节点的子节点。
- x [@ z =“v”]表示名称为x的元素，属性z的值为“v”。
- x [@ z1 =“v1”和@ z2 =“v2”]分别表示名称为x，属性z1和z2的值分别为“v1”和“v2”的元素。
- x [i]表示元素x位于其同辈元素之间的第i个元素。
- x [@ z =“v”] [i]表示元素名称为x，属性z的值为“v”的元素，并且位于其同属于该条件的兄弟之中的数字i中。

但是也可以使用更简洁的语法：

- x完全等同于//x（在任何深度级别搜索名称或引用x的元素，引用是th:ref或th:fragment属性）。
- 只要包含参数的规范，选择器也不允许使用元素名称/引用。 所以[@ class ='oneclass']是一个有效的选择器，用于查找具有值

为“oneclass”的类属性的任何元素（标签）。

高级属性选择功能：

- 除了=，其他比较运算符也是有效的：！=（不等于），^ =（开始）和$ =（以结尾）。 例如：x [@class ^ ='section']表示名称为并且类属性以section开头的元素。
- 属性可以从@（XPath-style）和without（jQuery-style）开始指定。所以x [z ='v']等价于x [@ z ='v']。
- 多属性修饰符可以与（XPath样式）和链接多个修饰符（jQuery样式）一起使用。 所以x [@ z1 ='v1'和@ z2 ='v2']实际上等同于x [@ z1 ='v1'] [@ z2 ='v2']（以及x [z1 ='v1'] [z2 ='v2']）。

jQuery式的选择器：

- x.oneclass相当于x [class ='oneclass']。
- .oneclass相当于[class ='oneclass']。
- x # oneid相当于x [id ='oneid']。
- #oneid等同于[id ='oneid']。
- x%oneref表示具有th:ref =“oneref”或th:fragment =“oneref”属性的<x>标签。
- %oneref表示具有th:ref =“oneref”或th:fragment =“oneref”属性的任何标签。 注意，这实际上等同于简单的oneref，因为可以使用引用而不是元素名称。
- 直接选择器和属性选择器可以混合使用：a.external [@href ^ ='https']。

所以上面的标签表达式：

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

可以这样写：

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

测试一个不同的例子，如下：

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

将寻找一个th:fragment =“myfrag”的Fragment（或th:ref引用）。 但是，如果存在名称为myfrag的标签，那么它们也不会在HTML中查找。 注意与...的区别：

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

实际上它将会查找class =“mydefrag”的任何元素，而不需要关心th:fragment（或th:ref引用）。

多值类匹配

标记选择器将类属性理解为多值，因此即使该元素具有多个类值，也允许在该属性上应用选择器。

例如，div.two将匹配<div class =“one two three”/>