**Thymeleaf 模板引擎**

Thymeleaf 应该是目前最受欢迎的模板引擎技术了，Spring Boot 官方也推荐 Java web 开发中使用该技术来替代 JSP 技术，主要由于其“原型即页面”的理念与 Spring Boot 倡导的快速开发非常契合，同时 Thymeleaf 模板引擎技术也确实拥有其他技术所不具备的优点。

Thymeleaf 3 的特性：

* **完整的 HTML5 标记支持全新的解析器**
* **自带多种模板模式，也可扩展支持其他模板格式。**
* **在 web 和非 web 环境（离线）下都可以正常工作**
* **对 Spring web 开发的支持非常完善**
* **独立于 Servlet API**
* **其他特性**:
  + Thymeleaf 3.0 引入了一种新型表达式作为一般 *Thymeleaf 标准表达*系统的一部分：\_片段表达式；\_。
  + Thymeleaf 3.0 中 *Thymeleaf 标准表达式* 的另一个新特性是 NO-OP（无操作）令牌，由下划线符号（\_）表示。
  + Thymeleaf 3.0 允许在模板和模板模式下完全（和可选）将模板逻辑与模板本身*解耦*，从而实现 100％-Thymeleaf-free 无逻辑模板。
  + Thymeleaf 3.0 采用全新的方言系统。
  + Thymeleaf 3.0 完成了核心 API 的重构。

Thymeleaf 是高级语言的模板引擎，语法更简单，功能也更强大，接下来的内容将是 Thymeleaf 与 Spring Boot 的整合过程，以及 Thymeleaf 模板引擎的语法介绍。

**引入 Thymeleaf 依赖**

因为 Spring Boot 官方提供了 Thymeleaf 的场景启动器 spring-boot-starter-thymeleaf ，因此可以直接在 pom.xml 文件中添加该场景启动器，最终的 pom.xml 文件如下：

*<*dependency*>  
 <*groupId*>*org.springframework.boot*</*groupId*>  
 <*artifactId*>*spring-boot-starter-thymeleaf*</*artifactId*>  
</*dependency*>*

**创建模板文件**

在 resources/templates 目录新建模板文件 thymeleaf.html ，Thymeleaf 模板引擎的默认后缀名即为 html，新增文件后，首先在模板文件的 <html> 标签中导入 Thymeleaf 的名称空间：

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

导入该名称空间主要是为了 Thymeleaf 的语法提示和 Thymeleaf 标签的使用，之后我们在模板中增加如前文 JSP 中相同的显示内容，最终的模板文件如下：

<!DOCTYPE html>

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

<head>

<title>Thymeleaf demo</title>

</head>

<body>

<p>description字段值为：</p>

<p th:text="${description}">这里显示的是 description 字段内容</p>

</body>

</html>

**编辑 Controller 代码**

在 controller 包下新增 ThymeleafController.java 文件，将模板文件所需的 description 字段赋值并转发至模板文件，编码如下：

package com.lou.springboot.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestParam;

import javax.servlet.http.HttpServletRequest;

@Controller

public class ThymeleafController {

@GetMapping("/thymeleaf")

public String hello(HttpServletRequest request, @RequestParam(value = "description", required = false, defaultValue = "springboot-thymeleaf") String description) {

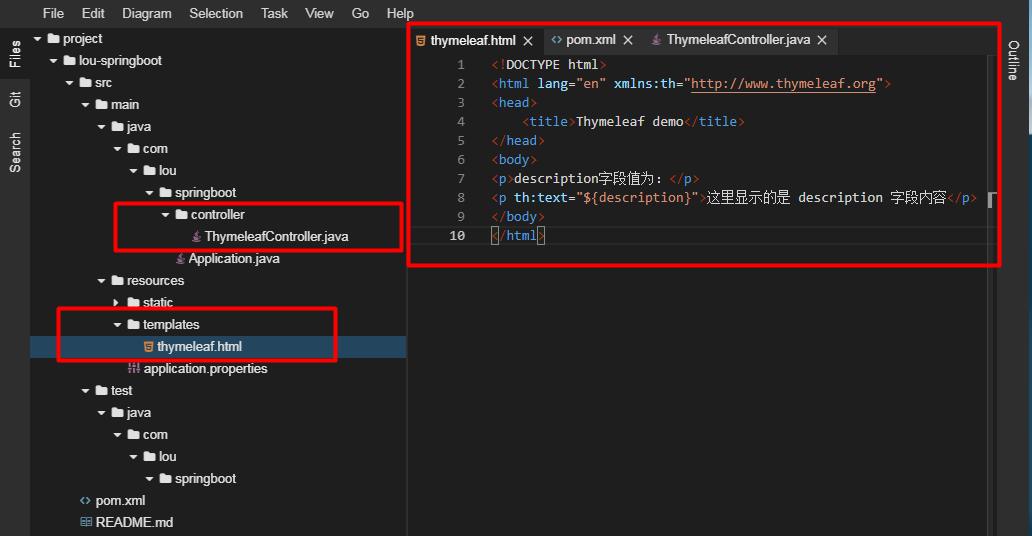
request.setAttribute("description", description);

return "thymeleaf";

}

}

最终的代码目录结构如下：



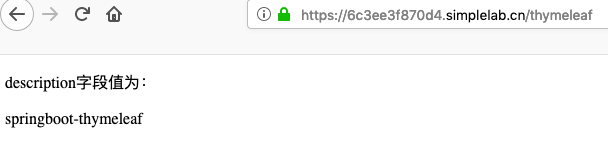
**启动并访问**

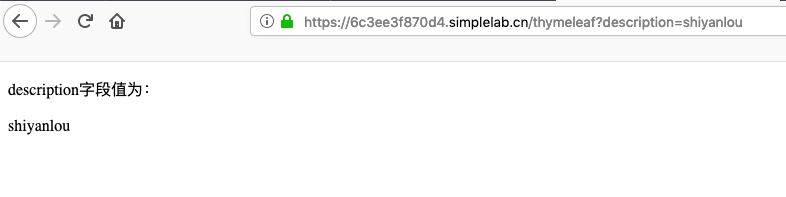
在编码完成后，我们启动项目，由于默认是在 project 目录下，因此想要启动咱们的项目，首先需要切换到 lou-springboot 目录下，之后可以通过 Maven 插件的方式启动 Spring Boot 项目，命令为 mvn spring-boot:run，之后就可以等待项目启动。

在项目启动成功后，可以点击页面上方的 Web 服务直接在显示查看网站效果，之后会在浏览器中弹出 https://\*\*\*\*\*\*\*\*.simplelab.cn 页面，我们访问 /thymeleaf，可以看到，原来静态 html 文件 <p> 标签中的内容已经替换为 "springboot-thymeleaf" 字符串，而不再是默认内容，过程如下：

访问：<http://localhost:8088/thyemeleaf>

http://localhost:8088/thymeleaf?description=shiyanlou

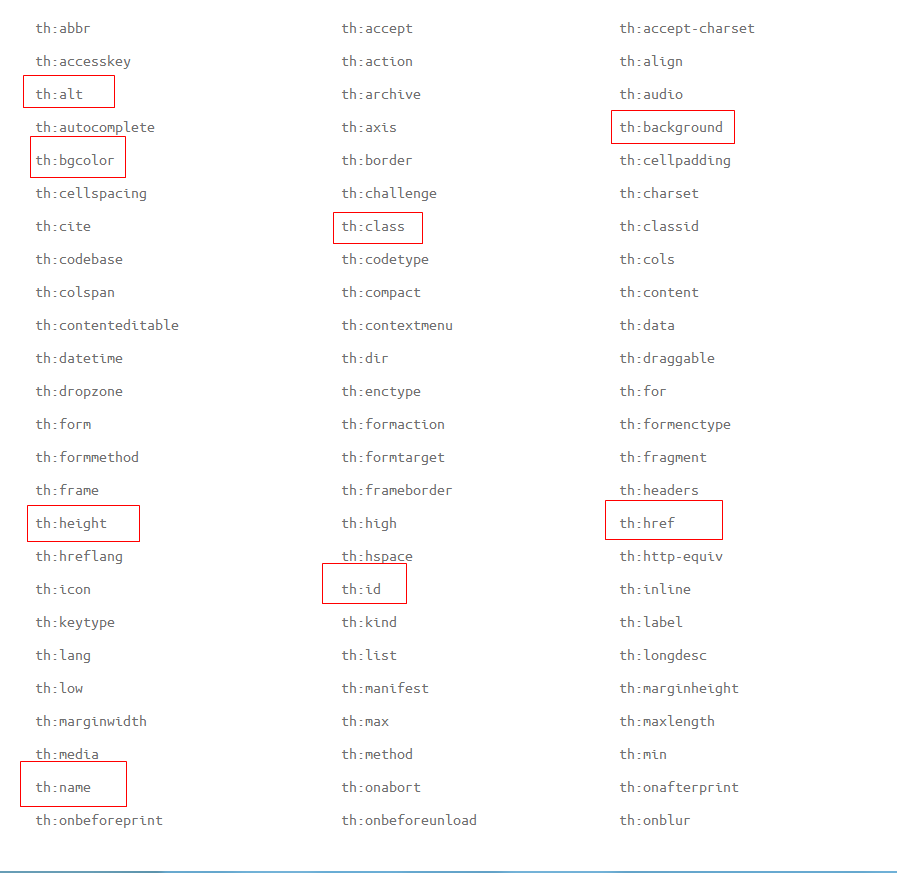




**Thymeleaf 设置属性值**

th:\* 属性：

th:text 对应的是 HTML5 中的 text 属性，除 th:text 属性外，Thymeleaf 也提供了其他的标签属性来替换 HTML5 原生属性的值，属性节选如下：



* th:background 对应 HTML5 中的背景属性
* th:class 对应 HTML5 中的 class 属性
* th:href 对应 HTML5 中的链接地址属性
* th:id 和 th:name 分别对应 HTML5 中的 id 和 name 属性...
* th:block 比较特殊，它是 Thymeleaf 提供的唯一的一个 Thymeleaf 块级元素，其特殊性在于 Thymeleaf 模板引擎在处理<th:block>的时候会删掉它本身，而保留其内容。

**修改属性值实践**

项目完整结构如下：

lou-springboot

└── src

├── main

│ ├── java

│ │ └── com

│ │ └── lou

│ │ └── springboot

│ │ ├── Application.java

│ │ └── controller

│ │ └── ThymeleafController.java

│ └── resources

│ ├── application.properties

│ ├── static

│ └── templates

│ ├── attributes.html

│ ├── simple.html

│ ├── test.html

│ └── thymeleaf.html

└── test

└── java

└── com

└── lou

└── springboot

attributes.html 代码如下：

<!DOCTYPE html>

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

<head>

<title>Thymeleaf setting-value-to-specific-attributes</title>

<meta charset="UTF-8" />

</head>

<!-- background 标签-->

<body th:background="${th\_background}" background="#D0D0D0">

<!-- text 标签-->

<h1 th:text="${title}">html标签演示</h1>

<div>

<h5>id、name、value标签:</h5>

<!-- id、name、value标签-->

<input

id="input1"

name="input1"

value="1"

th:id="${th\_id}"

th:name="${th\_name}"

th:value="${th\_value}"

/>

</div>

<br />

<div class="div1" th:class="${th\_class}">

<h5>class、href标签:</h5>

<!-- class、href标签-->

<a th:href="${th\_href}" href="##/">链接地址</a>

</div>

</body>

</html>

其中包含 background 、id 、name 、class 等 html 标签，设置默认值，并在每个标签体中添加对应的 th 标签读取动态数据，直接选择该文件，右键选择 open with -> Preview 或 Mini Browser 可查看页面效果如下：



需要注意的是，当前是直接打开该 html 文件并没有通过 web 服务器，此时页面能够直接正常访问且页面中的内容和元素的属性值都是默认值，之后，在 controller 包下新增对应的 Controller 方法并将请求转发至该模板页面，代码如下：

// ThymeleafController.java

package com.lou.springboot.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.web.bind.annotation.GetMapping;

@Controller

public class ThymeleafController {

@GetMapping("/attributes")

public String attributes(ModelMap map) {

// 更改 h1 内容

map.put("title", "Thymeleaf 标签演示");

// 更改 id、name、value

map.put("th\_id", "thymeleaf-input");

map.put("th\_name", "thymeleaf-input");

map.put("th\_value", "13");

// 更改 class、href

map.put("th\_class", "thymeleaf-class");

map.put("th\_href", "http://13blog.site");

return "attributes";

}

}

在编码完成后，我们启动项目，切换到 lou-springboot 目录下通过 Maven 插件的方式启动 Spring Boot 项目，命令为 mvn spring-boot:run ，之后就可以等待项目启动。然后访问<http://localhost:8088/att>；

由于 th 标签的存在，页面通过 Thymeleaf 渲染后，与静态页面相比较，内容和元素属性已经动态的切换了。

**Thymeleaf 语法规则**

Thymeleaf 官方对于标准表达式特性的总结：

* **表达式语法**
  + 变量表达式： ${...}
  + 选择变量表达式： \*{...}
  + 信息表达式： #{...}
  + 链接 URL 表达式： @{...}
  + 分段表达式： ~{...}
* **字面量**
  + 字符串： 'one text', 'Another one!' ...
  + 数字： 0, 34, 3.0, 12.3 ...
  + 布尔值： true, false
  + Null 值： null
  + 字面量标记：one, sometext, main ...
* **文本运算**
  + 字符串拼接： +
  + 字面量置换: |The name is ${name}|
* **算术运算**
  + 二元运算符： +, -, \*, /, %
  + 负号（一元运算符）： (unary operator): -
* **布尔运算**
  + 二元运算符： and, or
  + 布尔非（一元运算符）： !, not
* **比较运算**
  + 比较： >, <, >=, <= (gt, lt, ge, le)
  + 相等运算符： ==, != (eq, ne)

比较运算符也可以使用转义字符，比如大于号，可以使用 Thymeleaf 语法 gt 也可以使用转义字符&gt;

* **条件运算符**
  + If-then: (if) ? (then)
  + If-then-else: (if) ? (then) : (else)
  + Default: (value) ?: (defaultvalue)
* **特殊语法**
  + 无操作： \_

**简单语法**

创建 simple.html 模板页面，该案例主要介绍字面量及简单的运算操作，包括字符串、数字、布尔值等常用的字面量及常用的运算和拼接操作，代码如下：

<!DOCTYPE html>

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

<head>

<title>Thymeleaf simple syntax</title>

<meta charset="UTF-8" />

</head>

<body>

<h1>Thymeleaf简单语法</h1>

<div>

<h5>基本类型操作(字符串):</h5>

<p>

一个简单的字符串：<span th:text="'thymeleaf text'">default text</span>.

</p>

<p>

字符串连接：<span th:text="'thymeleaf text concat,'+${thymeleafText}"

>default text</span

>.

</p>

<p>

字符串连接：<span th:text="|thymeleaf text concat,${thymeleafText}|"

>default text</span

>.

</p>

</div>

<div>

<h5>基本类型操作(数字):</h5>

<p>一个简单的神奇的数字：<span th:text="2019">1000</span>.</p>

<p>算术运算： 2019+1=<span th:text="${number1}+1">0</span>.</p>

<p>算术运算： 14-1=<span th:text="14-1">0</span>.</p>

<p>算术运算： 673 \* 3=<span th:text="673\*${number2}">0</span>.</p>

<p>算术运算： 39 ÷ 3=<span th:text="39/3">0</span>.</p>

</div>

<div>

<h5>基本类型操作(布尔值):</h5>

<p>

一个简单的数字比较：2019 > 2018=<span th:text="2019&gt;2018"> </span>.

</p>

<p>

字符串比较：thymeleafText == 'shiyanlou'，结果为<span

th:text="${thymeleafText} == 'shiyanlou'"

>0</span

>.

</p>

<p>数字比较：13 == 39/3 结果为： <span th:text="13 == 39/3">0</span>.</p>

</div>

</body>

</html>

其中也有部分变量为后台设置的值，与字面量结合进行实践演示，新增 Controller 方法：

@GetMapping("/simple")

public String simple(ModelMap map) {

map.put("thymeleafText", "shiyanlou");

map.put("number1", 2018);

map.put("number2", 3);

return "simple";

}

之后可重启并查看效果，以上代码的结果如下：



左侧为静态 html 结果，右侧为 Thymeleaf 模板引擎渲染的结果，可以看到字面量的展示及运算结果。

**表达式语法**

表达式包括：变量表达式 ${...}、选择变量表达式 \*{...}、信息表达式 #{...}、链接 URL 表达式：@{...}、分段表达式： ~{...}，这些表达式一般只能写在 Thymeleaf 模板文件的 th 标签内，不然不会生效，表达式语法的主要作用就是获取变量值、获取绑定对象的变量值、国际化变量取值、URL 拼接与生成、 Thymeleaf 模板布局。

**变量表达式**

变量表达式即 OGNL 表达式或 Spring EL 表达式,作用是获取模板中与后端返回数据所绑定对象的值，写法为 ${...}，这是最常见的一个表达式，在取值赋值、逻辑判断、循环语句中都可以使用该表达式，示例如下：

<!-- 读取参数 -->

<p>算术运算： 2018+1=<span th:text="${number1}+1">0</span>.</p>

<!-- 读取参数并运算 -->

<div th:class="${path}=='links'?'nav-link active':'nav-link'"></div>

<!-- 读取对象中的属性 -->

<p>

读取blog对象中title字段：<span th:text="${blog.blogTitle}">default text</span

>.

</p>

<!-- 循环遍历 -->

<li th:each="blog : ${blogs}"></li>

变量表达式也可以使用内置的基本对象：

* ctx : the context object.
* vars : the context variables.
* locale : the context locale.
* request : web 环境下的 HttpServletRequest 对象.
* response :web 环境下的 HttpServletResponse 对象 .
* session : web 环境下的 HttpSession 对象.
* servletContext : web 环境下的 ServletContext 对象.

示例如下：

<p>

读取内置对象中 request 中的内容：<span

th:text="${#request.getAttribute('requestObject')}"

>default text</span

>.

</p>

<p>

读取内置对象中 session 中的内容：<span

th:text="${#session.getAttribute('sessionObject')}"

>default text</span

>.

</p>

同时，Thymeleaf 还提供了一系列 Utility 工具对象（内置于 Context 中），可以通过 # 直接访问，工具类如下：

* dates ： *java.util.Date 的功能方法类。*
* calendars : *类似 #dates，面向 java.util.Calendar*
* numbers : *格式化数字的工具方法类*
* strings : *字符串对象的工具方法类，contains,startWiths,prepending/appending 等等。*
* bools：*对布尔值求值的工具方法。*
* arrays：*对数组的工具方法。*
* lists：*对 java.util.List 的工具方法*
* sets：*对 java.util.Set 的工具方法*
* maps：*对 java.util.Map 的工具方法*

你可以将这些方法视为工具类，通过这些方法可以使得 Thymeleaf 在操作变量时更加方便。

#### 选择(星号)表达式

选择表达式与变量表达式类似，不过它会用一个预先选择的对象来代替上下文变量容器(map)来执行，语法如下： \*{blog.blogId}，被指定的对象由 th:object 标签属性进行定义，前文中读取 blog 对象的 title 字段可以替换为：

<p th:object="${blog}">

读取blog对象中title字段：<span th:text="\*{blogTitle}">text</span>.

</p>

如果不考虑上下文的情况下，两者没有区别，使用 ${...}读取的内容也完全可以替换为使用\*{...}进行读取，唯一的区别是使用\*{...}前可以预先在父标签中通过 th:object 定义一个对象并进行操作。

<p>

读取blog对象中title字段：<span th:text="\*{blog.blogTitle}">default text</span>

</p>

<p>读取text字段：<span th:text="\*{text}">default text</span>.</p>

#### URL 表达式

th:href 对应的是 html 中的 href 标签，它将计算并替换 href 标签中的链接 URL 地址，th:href 中可以直接设置为静态地址，也可以使用表达式语法读取到的变量值进行动态拼接 URL 地址。

比如一个详情页 URL 地址：http://localhost:8080/blog/1，当使用 URL 表达式时，可以写成这样：

<a th:href="@{'http://localhost:8080/blog/1'}">详情页</a>

也可以根据 id 值进行替换，写法为：

<a th:href="@{'/blog/'+${blog.blogId}}">详情页</a>

或者也可以写成这样：

<a th:href="@{/blog/{blogId}(blogId=${blog.blogId})">详情页</a>

以上三种表达式写法生成 URL 的最终结果都是相同的，开发者可以自己使用字符串拼接的方法组装 URL (第二种写法)，也可以使用 URL 表达式提供的语法进行 URL 组装（第三种写法）。如果有多个参数可以自行拼装字符串，或者使用逗号进行分隔，写法如下：

<a

th:href="@{/blog/{blogId}(blogId=${blog.blogId},title=${blog.blogTitle},tag='java')}"

>详情页</a

>

最终生成的 URL 为 http://localhost:8080/blog/1?title=lou-springboot&tag=java 另外，URL 中以 "/" 开头的路径(比如 /blog/1 )，默认生成的 URL 会加上项目的当前地址形成完整的 URL 。

**复杂语法**

test.html 模板文件如下：

<!DOCTYPE html>

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

<head>

<meta charset="UTF-8" />

<title th:text="${title}">语法测试</title>

</head>

<body>

<h3>#strings 工具类测试</h3>

<div th:if="${not #strings.isEmpty(testString)}">

<p>testString初始值 : <span th:text="${testString}" /></p>

<p>

toUpperCase : <span th:text="${#strings.toUpperCase(testString)}" />

</p>

<p>

toLowerCase : <span th:text="${#strings.toLowerCase(testString)}" />

</p>

<p>

equalsIgnoreCase :

<span th:text="${#strings.equalsIgnoreCase(testString, '13')}" />

</p>

<p>indexOf : <span th:text="${#strings.indexOf(testString, 'r')}" /></p>

<p>

substring : <span th:text="${#strings.substring(testString, 5, 9)}" />

</p>

<p>

startsWith :

<span th:text="${#strings.startsWith(testString, 'Spring')}" />

</p>

<p>

contains : <span th:text="${#strings.contains(testString, 'Boot')}" />

</p>

</div>

<h3>#bools 工具类测试</h3>

<!-- 如果 bool 的值为false的话，该div将不会显示-->

<div th:if="${#bools.isTrue(bool)}">

<p th:text="${bool}"></p>

</div>

<h3>#arrays 工具类测试</h3>

<div th:if="${not #arrays.isEmpty(testArray)}">

<p>length : <span th:text="${#arrays.length(testArray)}" /></p>

<p>contains : <span th:text="${#arrays.contains(testArray, 5)}" /></p>

<p>

containsAll :

<span th:text="${#arrays.containsAll(testArray, testArray)}" />

</p>

<p>循环读取 : <span th:each="i:${testArray}" th:text="${i+' '}" /></p>

</div>

<h3>#lists 工具类测试</h3>

<div th:unless="${#lists.isEmpty(testList)}">

<p>size : <span th:text="${#lists.size(testList)}" /></p>

<p>contains : <span th:text="${#lists.contains(testList, 0)}" /></p>

<p>sort : <span th:text="${#lists.sort(testList)}" /></p>

<p>循环读取 : <span th:each="i:${testList}" th:text="${i+' '}" /></p>

</div>

<h3>#maps 工具类测试</h3>

<div th:if="${not #maps.isEmpty(testMap)}">

<p>size : <span th:text="${#maps.size(testMap)}" /></p>

<p>

containsKey :

<span th:text="${#maps.containsKey(testMap, 'platform')}" />

</p>

<p>

containsValue : <span th:text="${#maps.containsValue(testMap, '13')}" />

</p>

<p>

读取map中键为title的值 :

<span

th:if="${#maps.containsKey(testMap,'title')}"

th:text="${testMap.get('title')}"

/>

</p>

</div>

<h3>#dates 工具类测试</h3>

<div>

<p>year : <span th:text="${#dates.year(testDate)}" /></p>

<p>month : <span th:text="${#dates.month(testDate)}" /></p>

<p>day : <span th:text="${#dates.day(testDate)}" /></p>

<p>hour : <span th:text="${#dates.hour(testDate)}" /></p>

<p>minute : <span th:text="${#dates.minute(testDate)}" /></p>

<p>second : <span th:text="${#dates.second(testDate)}" /></p>

<p>格式化: <span th:text="${#dates.format(testDate)}" /></p>

<p>

yyyy-MM-dd HH:mm:ss 格式化:

<span th:text="${#dates.format(testDate, 'yyyy-MM-dd HH:mm:ss')}" />

</p>

</div>

</body>

</html>

增加 Controller 方法如下：

@GetMapping("/test")

public String test(ModelMap map) {

map.put("title", "Thymeleaf 语法测试");

map.put("testString", "玩转 Spring Boot");

map.put("bool", true);

map.put("testArray", new Integer[]{2018,2019,2020,2021});

map.put("testList", Arrays.asList("Spring", "Spring Boot", "Thymeleaf", "MyBatis", "Java"));

Map testMap = new HashMap();

testMap.put("platform", "shiyanlou");

testMap.put("title", "玩转 Spring Boot");

testMap.put("author", "十三");

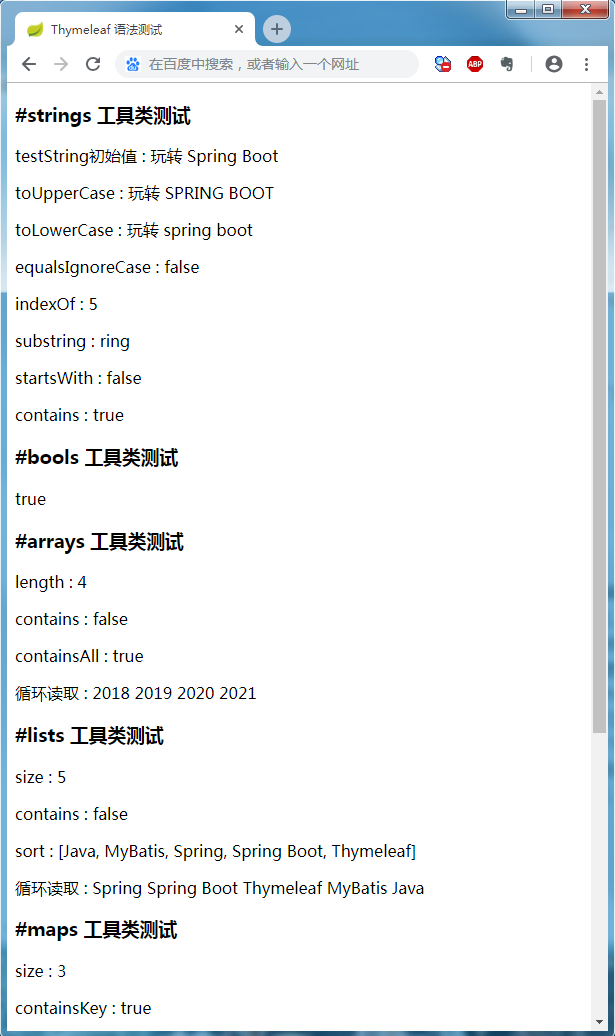
map.put("testMap", testMap);

map.put("testDate", new Date());

return "test";

}

之后重启项目并访问 /test 即可，结果如下



**Thymeleaf 模板引擎使用注意事项**

必须引入名称空间

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

即使不引入以上名称空间，静态资源访问以及模板动态访问都不会报错，因此有些开发者可能会忽略这个事情。但是建议在开发过程中最好引入该名称空间，因为引入之后会有 Thymeleaf 代码的语法提示，能够提升开发效率，也减少人为造成的低级错误。

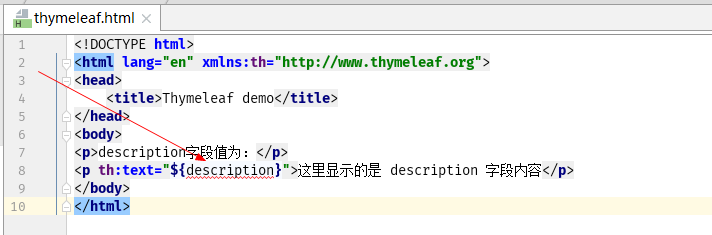
禁用模板缓存

Thymeleaf 的默认缓存设置是通过配置文件的 **spring.thymeleaf.cache** 配置属性决定的，通过如上 Thymeleaf 模板的配置属性类 ThymeleafProperties 可以发现该属性默认为 true，因此 Thymeleaf 默认是使用模板缓存的，该设置有助于改善应用程序的性能，因为模板只需编译一次即可，但是在开发过程中不能实时看到页面变更的效果，除非重启应用程序，因此建议将该属性设置为 false，在配置文件中修改如下：

spring.thymeleaf.cache=false

IDEA 中通过 Thymeleaf 语法读取变量时爆红色波浪线问题

如下图所示，在刚开始使用 Thymeleaf 开发时可能会碰到这种问题，在模板文件中通过 Thymeleaf 语法读取变量时，该变量名称下会出现红色波浪线，也就是错误的标志。

如果不熟的朋友可能会人为自己的模板文件有问题，但其实并不是那么严重，只是由于 IDEA 中默认开启了表达式参数验证，即使在后端的 model 数据中添加了该变量，但是对于前端文件是无法感知的，因此会报这个错误，可以在 IDEA 中默认将验证关闭或者将提示级别修改掉也可以。

