第2-3课：模板引擎 Thymeleaf 基础使用

模板引擎

模板引擎是为了使用户界面与业务数据（内容）分离而产生的，它可以生成特定格式的文档，用于网站的模板引擎就会生成一个标准的 HTML 文档。

模板引擎的实现方式有很多，最简单的是“置换型”模板引擎，这类模板引擎只是将指定模板内容（字符串）中的特定标记（子字符串）替换，便生成了最终需要的业务数据（如网页）。

“置换型”模板引擎实现简单，但其效率低下，无法满足高负载的应用需求（比如有海量访问的网站），因此还出现了“解释型”模板引擎和“编译型”模板引擎等。

Thymeleaf 介绍

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，从而改善了设计的沟通，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 HTML 5 标准，如果需要，Thymeleaf 允许创建完全符合 HTML 5 验证标准的模板。

Spring Boot 体系内推荐使用 Thymeleaf 作为前端页面模板，并且 Spring Boot 2.0 中默认使用 Thymeleaf 3.0，性能提升幅度很大。

Thymeleaf 特点

简单说，Thymeleaf 是一个跟 Velocity、FreeMarker 类似的模板引擎，它可以完全替代 JSP。与其他的模板引擎相比较，它有如下三个极吸引人的特点。

- Thymeleaf 在有网络和无网络的环境下皆可运行，即它可以让美工在浏览器查看页面的静态效果，也可以让程序员在服务器查看带数据的动态页面效果。这是由于它支持 HTML 原型，然后在 HTML 标签里增加额外的属性来达到模板 + 数据的展示方式。浏览器解释 HTML 时会忽略未定义的标签属性，所以 Thymeleaf 的模板可以静态地运行；当有数据返回到页面时，Thymeleaf 标签会动态地替换掉静态内容，使页面动态显示。

- Thymeleaf 开箱即用的特性。它支持标准方言和 Spring 方言，可以直接套用模板实现 JSTL、 OGNL 表达式效果，避免每天套模板、改 JSTL、改标签的困扰。同时开发人员也可以扩展和创建自定义的方言。

- Thymeleaf 提供 Spring 标准方言和一个与 SpringMVC 完美集成的可选模块，可以快速地实现表单绑定、属性编辑器、国际化等功能。

对比

我们可以对比一下 Thymeleaf 和常用的模板引擎：Velocity、Freemaker，和其他模板一起不同的是，它使用了自然的模板技术。这意味着 Thymeleaf 的模板语法并不会破坏文档的结构，模板依旧是有效的 XML 文档。模板还可以用做工作原型，Thymeleaf 会在运行期替换掉静态值。Velocity 与 FreeMarker 则是连续的文本处理器。

下面的代码示例分别使用 Velocity、FreeMarker 与 Thymeleaf 打印出一条消息：

```
Velocity: <p>$message</p>
FreeMarker: <p>${message}</p>
Thymeleaf: <p th:text="${message}">Hello World!</p>
```

上面我们可以看出来 Thymeleaf 的作用域在 HTML 标签内，类似标签的一个属性来使用，这就是它的特点。

注意，由于 Thymeleaf 使用了 XML DOM 解析器，因此它并不适合于处理大规模的 XML 文件。

快速上手

来一个 Thymeleaf 的 Hello World 尝尝鲜。

相关配置

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

在 application.properties 中添加配置：

```
spring.thymeleaf.cache=false
```

其中，propertiesspring.thymeleaf.cache=false 是关闭 Thymeleaf 的缓存，不然在开发过程中修改页面不会立刻生效需要重启，生产可配置为 true。

一个简单的页面

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"></meta>
    <title>Hello</title>
</head>
<body>
<h1  th:text="${message}">Hello World</h1>
</body>
</html>
```

所有使用 Thymeleaf 的页面必须在 HTML 标签声明 Thymeleaf：

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

表明页面使用的是 Thymeleaf 语法。

Controller

```
@Controller
public class HelloController {
    @RequestMapping("/")
    public String index(ModelMap map) {
        map.addAttribute("message", "http://www.ityouknow.com");
        return "hello";
    }
}
```

这样就完成了，是不是很简单。启动项目后在浏览器中输入网址：http://localhost:8080/，会出现下面的结果：

```
http://www.ityouknow.com
```

说明页面的值，已经成功的被后端传入的内容所替换。

常用语法

我们新建 ExampleController 来封装不同的方法进行演示。

赋值、字符串拼接

赋值和拼接：

```
<p th:text="${userName}">neo</p>
<span th:text="'Welcome to our application, ' + ${userName} + '!'"></span>
```

字符串拼接还有另外一种简洁的写法：

```
<span th:text="|Welcome to our application, ${userName}!|"></span>
```

页面 string.html：

```
<!DOCTYPE html>
<html lang="en"  xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"></meta>
    <title>Example String </title>
</head>
<body>
    <div >
        <h1>text</h1>
        <p th:text="${userName}">neo</p>
        <span th:text="'Welcome to our application, ' + ${userName} + '!'"></span>
        <br/>
        <span th:text="|Welcome to our application, ${userName}!|"></span>
    </div>
</body>
</html>
```

后端传值：

```
@RequestMapping("/string")
public String string(ModelMap map) {
    map.addAttribute("userName", "ityouknow");
    return "string";
}
```

使用 ModelMap 以 KV 的方式存储传递到页面。

启动项目后在浏览器中输入网址：http://localhost:8080/string，会出现下面的结果：

```
text

ityouknow

Welcome to our application, ityouknow!
Welcome to our application, ityouknow!
```

条件判断 If/Unless

Thymeleaf 中使用 th:if 和 th:unless 属性进行条件判断，在下面的例子中， `<a>` 标签只有在 th:if 中条件成立时才显示：

```
<a th:if="${flag == 'yes'}"  th:href="@{http://favorites.ren/}"> home </a>
<a th:unless="${flag != 'no'}" th:href="@{http://www.ityouknow.com/}" >ityouknow</
a>
```

th:unless 与 th:if 恰好相反，只有表达式中的条件不成立，才会显示其内容。

页面 if.html：

```
<!DOCTYPE html>
<html lang="en"  xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"></meta>
    <title>Example If/Unless </title>
</head>
<body>
<div >
    <h1>If/Unless</h1>
    <a th:if="${flag == 'yes'}"  th:href="@{http://favorites.ren/}"> home </a>
    <br/>
    <a th:unless="${flag != 'no'}" th:href="@{http://www.ityouknow.com/}" >ityoukn
ow</a>
</div>
</body>
</html>
```

后端传值：

```
@RequestMapping("/if")
public String ifunless(ModelMap map) {
    map.addAttribute("flag", "yes");
    return "if";
}
```

使用 ModelMap 以 KV 的方式存储传递到页面。

启动项目后在浏览器中输入网址：http://localhost:8080/if，会出现下面的结果：

```
If/Unless

home
```

单击 home 链接会跳转到：http://favorites.ren/ 地址。

for 循环

for 循环在我们项目中使用的频率太高了，一般结合前端的表格来使用。

首先在后端定义一个用户列表：

```
private List<User> getUserList(){
    List<User> list=new ArrayList<User>();
    User user1=new User("大牛",12,"123456");
    User user2=new User("小牛",6,"123563");
    User user3=new User("纯洁的微笑",66,"666666");
    list.add(user1);
    list.add(user2);
    list.add(user3);
    return  list;
}
```

按照键 users，传递到前端：

```
@RequestMapping("/list")
public String list(ModelMap map) {
    map.addAttribute("users", getUserList());
    return "list";
}
```

页面 list.html 进行数据展示：

```
<!DOCTYPE html>
<html lang="en"  xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"></meta>
    <title>Example If/Unless </title>
</head>
<body>
<div >
    <h1>for 循环</h1>
    <table>
        <tr  th:each="user,iterStat : ${users}">
            <td th:text="${user.name}">neo</td>
            <td th:text="${user.age}">6</td>
            <td th:text="${user.pass}">213</td>
            <td th:text="${iterStat.index}">index</td>
        </tr>
    </table>
</div>
</body>
</html>
```

iterStat 称作状态变量，属性有：

- index，当前迭代对象的 index（从 0 开始计算）；
- count，当前迭代对象的 index（从 1 开始计算）；
- size，被迭代对象的大小；
- current，当前迭代变量；
- even/odd，布尔值，当前循环是否是偶数/奇数（从 0 开始计算）；
- first，布尔值，当前循环是否是第一个；
- last，布尔值，当前循环是否是最后一个。

在浏览器中输入网址：http://localhost:8080/list，页面展示效果如下：

```
for 循环
大牛  12  123456  0
小牛  6  123563  1
纯洁的微笑  66  666666  2
```

URL

URL 在 Web 应用模板中占据着十分重要的地位，需要特别注意的是 Thymeleaf 对于 URL 的处理是通过语法 `@{...}` 来处理的。如果需要 Thymeleaf 对 URL 进行渲染，那么务必使用 th:href、th:src 等属性，下面是一个例子：

```
<a th:href="@{http://www.ityouknow.com/{type}(type=${type})}">link1</a>
<a th:href="@{http://www.ityouknow.com/{pageId}/can-use-springcloud.html(pageId=${
pageId})}">view</a>
```

也可以使用 `@{...}` 设置背景：

```
<div th:style="'background:url(' + @{${img url}} + ');'">
```

几点说明：

- 上例中 URL 最后的 `(pageId=${pageId})` 表示将括号内的内容作为 URL 参数处理，该语法避免使用字符串拼接，大大提高了可读性；
- `@{...}` 表达式中可以通过 `{pageId}` 访问 Context 中的 pageId 变量；
- `@{/order}` 是 Context 相关的相对路径，在渲染时会自动添加上当前 Web 应用的 Context 名字，假设 context 名字为 app，那么结果应该是 `/app/order` 。

完整的页面内容 url.html：

```
<!DOCTYPE html>
<html lang="en"  xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"></meta>
    <title>Example If/Unless </title>
</head>
<body>
<div >
    <h1>URL</h1>
    <a  th:href="@{http://www.ityouknow.com/{type}(type=${type})}">link1</a>
    <br/>
    <a th:href="@{http://www.ityouknow.com/{pageId}/can-use-springcloud.html(pageI
d=${pageId})}">view</a>
    <br/>
    <div th:style="'background:url(' + @{${img}} + ');'">
        <br/><br/><br/>
    </div>
</div>
</body>
</html>
```

后端程序：

```
@RequestMapping("/url")
public String url(ModelMap map) {
    map.addAttribute("type", "link");
    map.addAttribute("pageId", "springcloud/2017/09/11/");
    map.addAttribute("img", "http://www.ityouknow.com/assets/images/neo.jpg");
    return "url";
}
```

在浏览器中输入网址：http://localhost:8080/url，页面展示效果：

三目运算

三目运算是我们常用的功能之一，普遍应用在各个项目中，下面来做一下演示。

三目运算及表单显示：

```
<input th:value="${name}"/>
<input th:value="${age gt 30 ? '中年':'年轻'}"/>
```

说明：在表单标签中显示内容使用：`th:value;${age gt 30 ? '中年':'年轻'}` 表示如果 age 大于 30 则显示中年，否则显示年轻。

- gt：great than（大于）
- ge：great equal（大于等于）
- eq：equal（等于）
- lt：less than（小于）
- le：less equal（小于等于）
- ne：not equal（不等于）

结合三目运算也可以将上面的 if else 改成这样：

```
<a th:if="${flag eq 'yes'}"  th:href="@{http://favorites.ren/}"> favorites </a>
```

完整的页面内容 eq.html：

```
<!DOCTYPE html>
<html lang="en"  xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"></meta>
    <title>Example If/Unless </title>
</head>
<body>
<div >
    <h1>EQ</h1>
    <input th:value="${name}"/>
    <br/>
    <input th:value="${age gt 30 ? '中年':'年轻'}"/>
    <br/>
    <a th:if="${flag eq 'yes'}"  th:href="@{http://favorites.ren/}"> favorites </a
>
</div>
</body>
</html>
```

后端程序：

```
@RequestMapping("/eq")
public String eq(ModelMap map) {
    map.addAttribute("name", "neo");
    map.addAttribute("age", 30);
    map.addAttribute("flag", "yes");
    return "eq";
}
```

在浏览器中输入网址：http://localhost:8080/eq，页面展示效果如下：

```
EQ

neo
年轻

favorites
```

单击 favorites 链接会跳转到：http://favorites.ren/ 地址。

switch 选择

switch\case 多用于多条件判断的场景下，以性别举例：

```
<!DOCTYPE html>
<html lang="en"  xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"></meta>
    <title>Example switch </title>
</head>
<body>
<div >
    <div th:switch="${sex}">
        <p th:case="'woman'">她是一个姑娘...</p>
        <p th:case="'man'">这是一个爷们!</p>
        <!-- *: case的默认的选项 -->
        <p th:case="*">未知性别的一个家伙。</p>
    </div>
</div>
</body>
</html>
```

后端程序：

```
@RequestMapping("/switch")
public String switchcase(ModelMap map) {
    map.addAttribute("sex", "woman");
    return "switch";
}
```

在浏览器中输入网址：http://localhost:8080/switch，页面展示效果如下：

```
她是一个姑娘...
```

可以在后台改 sex 的值来查看结果。

总结

本课介绍了模板引擎、Thymeleaf 的使用特点、应用场景，通过实例演练展示了 Thymeleaf 各种语法特性。通过学习可以了解到，Thymeleaf 是一个非常灵活和优秀的前端页面模板引擎，使用 Thymeleaf 可以非常灵活地展示页面内容。

> 点击这里下载源码。