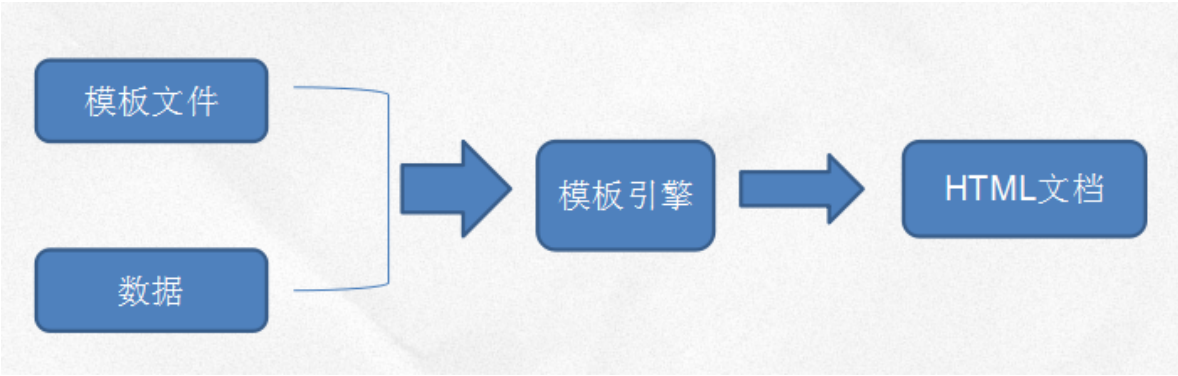
Thymeleaf

课程目标

- 了解模板引擎是什么，解决什么问题。
- 理解 Thymeleaf 是什么，有什么优势。
- 掌握使用 Thymeleaf 语法。

一、模板引擎（了解）

模板引擎（这里特指用于 Web 开发的模板引擎）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。从字面上理解模板引擎，最重要的就是模板二字，这个意思就是做好一个模板后套入对应位置的数据，最终以 HTML 的格式展示出来，这就是模板引擎的作用。



将模板设计好之后直接填充数据即可而不需要重新设计整个页面，从而提高页面、代码的复用性。若没有模板引擎技术我们想想之前是如何使用 Servlet 处理请求响应 HTML 格式内容回浏览器的?

在 Java 中模板引擎还有很多，模板引擎是动态网页发展进步的产物，在最初并且流传度最广的 JSP 它就是一个模板引擎。JSP 是官方标准的模板，但是由于 JSP 的缺点比较多也挺严重的，所以很多人弃用 JSP 选用第三方的模板引擎，市面上开源的第三方的模板引擎也比较多，有 Thymeleaf、FreeMaker、Velocity 等模板引擎受众较广。

二、Thymeleaf 简介（理解）

Thymeleaf 是适用于 Web 和独立环境的现代服务器端 Java 模板引擎。

Spring 官方支持模板引擎中，并不包含 JSP。而是 Thymeleaf 和 FreeMarker 等，而 Thymeleaf 与 Spring Boot 的自动化配置集成非常完美，几乎没有任何成本，你只用关注 Thymeleaf 的语法即可。

Thymeleaf 的特点有如下：

- 动静结合：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。
- 与 Spring Boot 完美整合，Spring Boot 提供了 Thymeleaf 的默认配置，并且为 Thymeleaf 设置了视图解析器，我们可以像以前操作 JSP 一样来操作 Thymeleaf。使用它时，代码几乎没有任何区别，只在模板语法上有区别。

对比 JSP，Thymeleaf 有如下优势：

- 很好地分离表现层和业务逻辑，明确分工。
  - JSP 功能很强大，它可以在前台编写业务逻辑代码，这也带来了一个弊端——页面内容凌乱，可读性差，这将会大大增加后期的维护难度。而 FreeMarker 很单纯，仅仅负责页面表现，而去掉了繁琐的逻辑代码。
  - Thymeleaf 的原理就是：模板 + 数据模型 = 输出，模板只负责数据在页面中的表现，不涉及任何的逻辑代码，而所有的逻辑都是由数据模型来处理的。用户最终看到的输出是模板和数据模型合并后创建的。
- 提高渲染视图效率。
  - 众所周知，JSP 在第一次执行的时候需要转换成 Servlet 类，之后的每次修改都要编译和转换。这样就造成了每次修改都需要等待编译的时间，开发效率低下。而 Thymeleaf 模板技术并不存在编译和转换的问题，所以就不会存在上述问题。相比而言，使用 Thymeleaf 可以提高一定的效率，而且 JSP 编译后要把字节码加载到 JVM 的内存中，如果页面多了容易导致内存溢出问题，但是 Thymeleaf 则不会一直占着内存，用完后被清理。

三、Thymeleaf 入门（了解）

3.1、添加依赖

```
<dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf-spring5</artifactId>
    <version>3.0.12.RELEASE</version>
</dependency>
```

3.2、配置视图解析器

```
<bean id="templateResolver"
class="org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolver">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".html" />
    <property name="templateMode" value="HTML" />
```

```
    <property name="cacheable" value="false" />
</bean>

<bean id="templateEngine" class="org.thymeleaf.spring5.SpringTemplateEngine">
    <property name="templateResolver" ref="templateResolver" />
</bean>

<bean class="org.thymeleaf.spring5.view.ThymeleafViewResolver">
    <property name="templateEngine" ref="templateEngine" />
    <property name="characterEncoding" value="UTF-8" />
</bean>
```

3.3、编写控制器

```
@RequestMapping("/test")
public String test(Model model) {
    model.addAttribute("msg", "Hello Thymeleaf");
    return "t";
}
```

3.4、编写模板文件

在 WEB-INF/views 目录下新建名称为 t.html 的模板文件。

```
<p th:text="${msg}">默认值</p>
```

3.5、动静结合

Thymeleaf 中所有的表达式都需要写在"指令"中，指令是 HTML5 中的自定义属性，在 Thymeleaf 中所有指令都是以 `th:` 开头。因为表达式 `${msg}` 是写在自定义属性中，因此在静态环境下，表达式的内容会被当做是普通字符串，浏览器会自动忽略这些指令，这样就不会报错了。

指令的设计，正是 Thymeleaf 的高明之处，也是它优于其它模板引擎的原因。动静结合的设计，使得无论是前端开发人员还是后端开发人员可以完美契合。

四、Thymeleaf 使用（掌握）

1、Thymeleaf 内置对象

1.1、一些环境相关对象

对象	作用
`ctx`	获取 Thymeleaf 自己的 Context 对象
`requset`	若是 Web 程序，可以获取 HttpServletRequest 对象
`response`	若是 Web 程序，可以获取 HttpServletReponse 对象
`session`	若是 Web 程序，可以获取 HttpSession 对象
`servletContext`	若是 Web 程序，可以获取 HttpServletContext 对象

1.2、Thymeleaf 提供的全局对象

对象	作用
`#dates`	处理 java.util.Date 的工具对象
`#calendars`	处理 java.util.Calendar 的工具对象
`#numbers`	用来对数字格式化的方法
`#strings`	用来处理字符串的方法
`#bools`	用来判断布尔值的方法
`#arrays`	用来护理数组的方法
`#lists`	用来处理 List 集合的方法
`#sets`	用来处理 set 集合的方法
`#maps`	用来处理 map 集合的方法

```
<p>
    今天是: <span th:text="${#dates.format(today,'yyyy-MM-dd')}">2021-06-
14</span>
</p>
```

2、字面值

有的时候，我们需要在指令中填写基本类型如：字符串、数值、布尔等，并不希望被 Thymeleaf 解析为变量，这个时候称为字面值。

2.1、字符串字面值

使用一对'引用的内容就是字符串字面值了：

```
<p>
  你正在观看 <span th:text="'thymeleaf'">template</span> 的字符串常量值。
</p>
```

`th:text` 中的 Thymeleaf 并不会被认为是变量，而是一个字符串。

我们经常会用到普通字符串与表达式拼接的情况：

```
<span th:text="''欢迎您：' + ${name} + '!'"></span>
```

字符串字面值需要用 `''`，拼接起来非常麻烦，Thymeleaf 对此进行了简化，使用一对 `|` 即可：

```
<span th:text="|欢迎您：${name}|"></span>
```

2.2、数字字面值

数字不需要任何特殊语法，写的什么就是什么，而且可以直接进行算术运算。

```
<p>今年是 <span th:text="2021">1970</span></p>
<p>十年后将会是 <span th:text="2021 + 10">1980</span></p>
```

2.3、布尔字面值

布尔类型的字面值是 true 或 false。

3、运算

需要注意：`${}` 内部的是通过 OGNL 表达式引擎解析的，外部的才是通过 Thymeleaf 的引擎解析，因此运算符尽量放在 `${}` 外进行。

3.1、算术运算

支持的算术运算符：`+`、`-`、`*`、`/`、`%`。

```
<span th:text="${employee.age}"></span>
<span th:text="${employee.age}%2 == 0"></span>
```

3.2、比较运算

支持的比较运算：`>`，`<`，`>=`，`<=`，`==`，`!=`，`and`，`or`，但是 `>`，`<` 不能直接使用，因为 XML 会解析为标签，要使用别名。

注意 `==` 和 `!=` 不仅可以比较数值，类似于 equals 的功能。

可以使用的别名：

- gt（>）
- lt（<）
- ge（>=）
- le（<=）
- not（!）
- eq（==）
- neq/ne（!=）

3.3、条件运算

```
<span th:text="${employee.gender} ? '男':'女'"></span>
```

有的时候需要设置默认值，我们取一个值可能为空，这个时候需要做非空判断，可以使用表达式 `?:` 默认值简写（注意：`?:` 之间没有空格）：

```
<span th:text="${employee.name} ?: 'lony'"></span>
```

当前面的表达式值为 null 时，就会使用后面的默认值。

4、判断

使用 `th:if` 或者 `th:unless`，两者的意思恰好相反。

```
<span th:if="${age} > 18">年龄大于 18</span>
```

上面如果表达式的值为 true，则标签会渲染到页面，否则不进行渲染。以下情况被认定为 true：

- 表达式值为 true
- 表达式值为非 0 数值
- 表达式值为非 0 字符
- 表达式值为字符串，但不是 false，no，off
- 表达式不是布尔、字符串、数字、字符中的任何一种
- 其它情况包括 null 都被认定为 false

分支控制 switch，这里要使用两个指令：`th:switch` 和 `th:case`。

```
<div th:switch="${employee.role}">
    <p th:case="'admin'">员工是管理员</p>
    <p th:case="'manager'">员工是经理</p>
    <p th:case="*">员工是其它角色</p>
</div>
```

需要注意的是，一旦有一个 `th:case` 成立，其它的则不再判断。与 Java 中的 switch 是一样的。另外 `th:case="*"` 表示默认，放最后。

5、循环

```
<tr th:each="employee,stat:${employees}">
    <td th:text="${stat.count}">1</td>
    <td th:text="${employee.id}">1</td>
    <td th:text="${employee.name}">范伟</td>
    <td th:text="${employee.password}">666</td>
</tr>
```

其中 stat 对象包含以下属性：

- index，从 0 开始的角标
- count，元素的个数，从 1 开始
- size，总元素个数
- current，当前遍历到的元素
- even/odd，返回是否为奇偶，boolean 值
- first/last，返回是否为第一或最后，boolean 值

6、JS 模板

模板引擎不仅可以渲染 HTML，也可以对 JS 中的进行预处理。而且为了在纯静态环境下可以运行，其 Thymeleaf 代码可以被注释起来。如下操作：

- 首先在 script 标签中通过 th:inline="javascript" 来声明这是要特殊处理的 JS 脚本。
- 其次使用如下语法结构：

```
var 变量名 = /*[[Thymeleaf表达式]]*/ 静态环境下的默认值;
```

```
<script th:inline="javascript">
    var Employeename = /*[[${username}]]*/ 'lony';
    var age = /*[[${age}]]*/ 20;
    console.log(username);
    console.log(age)
</script>
```

7、fragment 方式

在 Thymeleaf 模板中有个指令实现 fragment 方式：

- `th:insert`：把整个 fragment 插入到当前节点内部。
- `th:replace`：用 fragment 替换到当前节点。
- `th:include`：把 fragment 的内容（不包括 fragment 的节点）插入到当前节点。

被引入模板文件 footer.html，内容如下

```
<span th:fragment="copytxt">
    Xxx 公司版权所有
</span>
```

引入上面模板的文件 home.html，内容如下：

```
<div th:replace="footer :: copytxt"></div>
<div th:insert="footer :: copytxt"></div>
<div th:include="footer :: copytxt"></div>
```

上面两个模板文件在同一目录下。

练习

- 在之前 SSM 项目基础上，把 JSP 改成 Thyemealf。