

第二讲 SpringBoot 开发 Web

一、Thymeleaf 模板

Thymeleaf 是用来开发 Web 和独立环境项目的现代服务器端 Java 模板引擎。

1. 特点

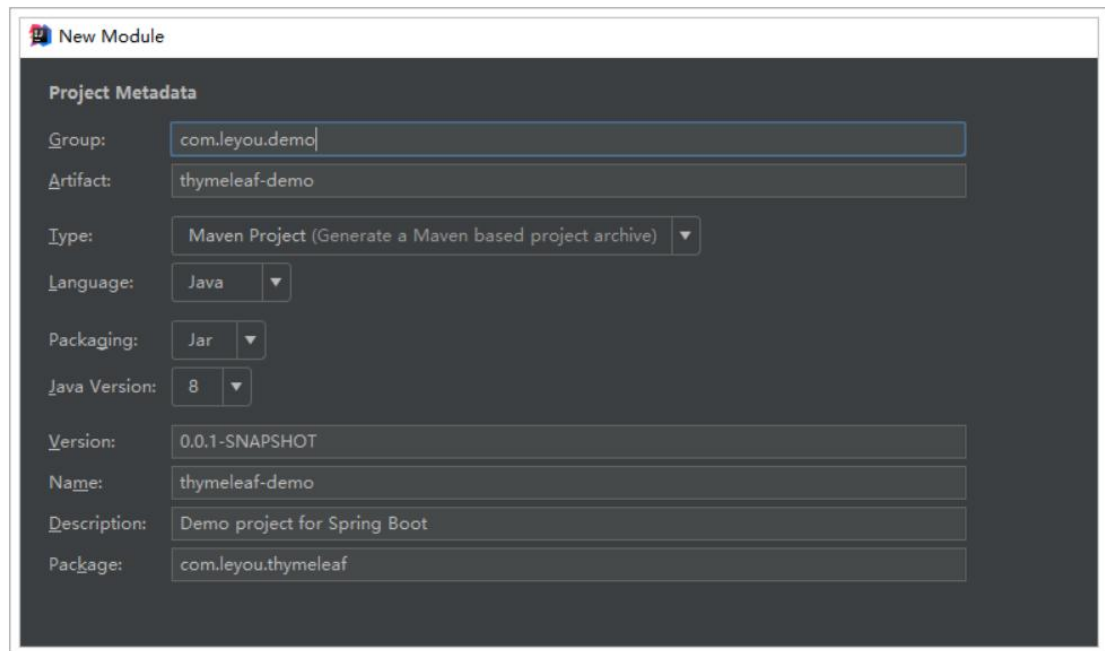
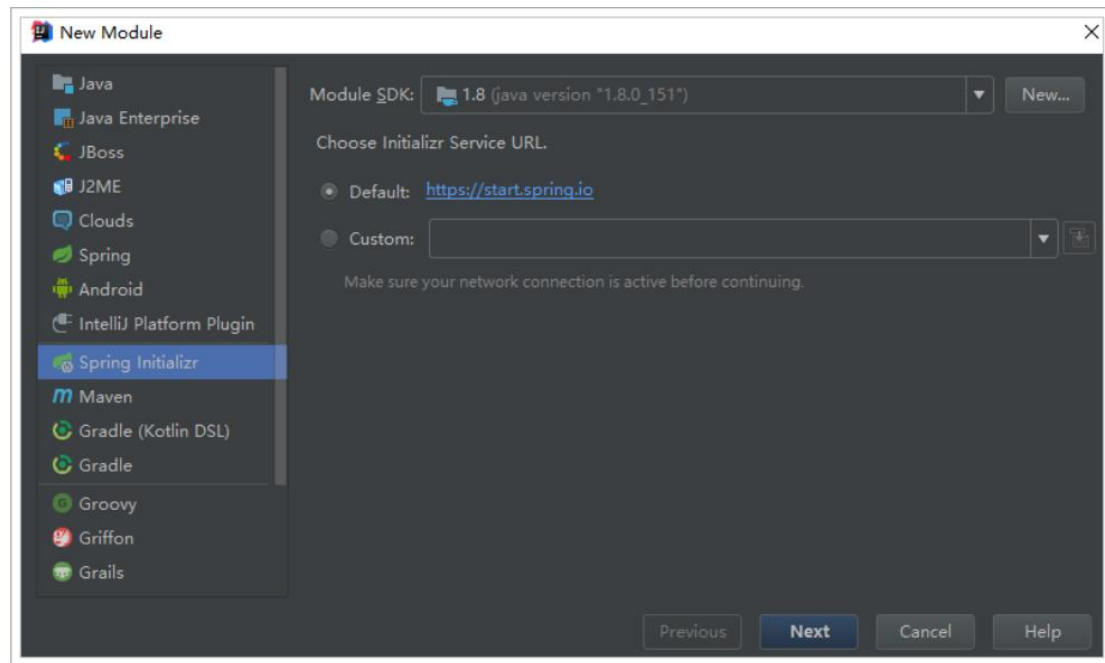
- 动静结合: Thymeleaf 在有网络和无网络的环境下皆可运行, 即它可以让美工在浏览器查看页面的静态效果, 也可以让程序员在服务器查看带数据的动态页面效果。这是由于它支持 html 原型, 然后在 html 标签里增加额外的属性来达到模板+数据的展示方式。浏览器解释 html 时会忽略未定义的标签属性, 所以 thymeleaf 的模板可以静态地运行; 当有数据返回到页面时, Thymeleaf 标签会动态地替换掉静态内容, 使页面动态显示。
- 开箱即用: 它提供标准和 spring 标准两种方言, 可以直接套用模板实现 JSTL、OGNL 表达式效果, 避免每天套模板、改 jstl、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。
- 多方言支持: Thymeleaf 提供 spring 标准方言和一个与 SpringMVC 完美集成的可选模块, 可以快速的实现表单绑定、属性编辑器、国际化等功能。
- 与 SpringBoot 完美整合, SpringBoot 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。代码几乎没有任何区别, 就是在模板语法上有区别。

2. 环境准备

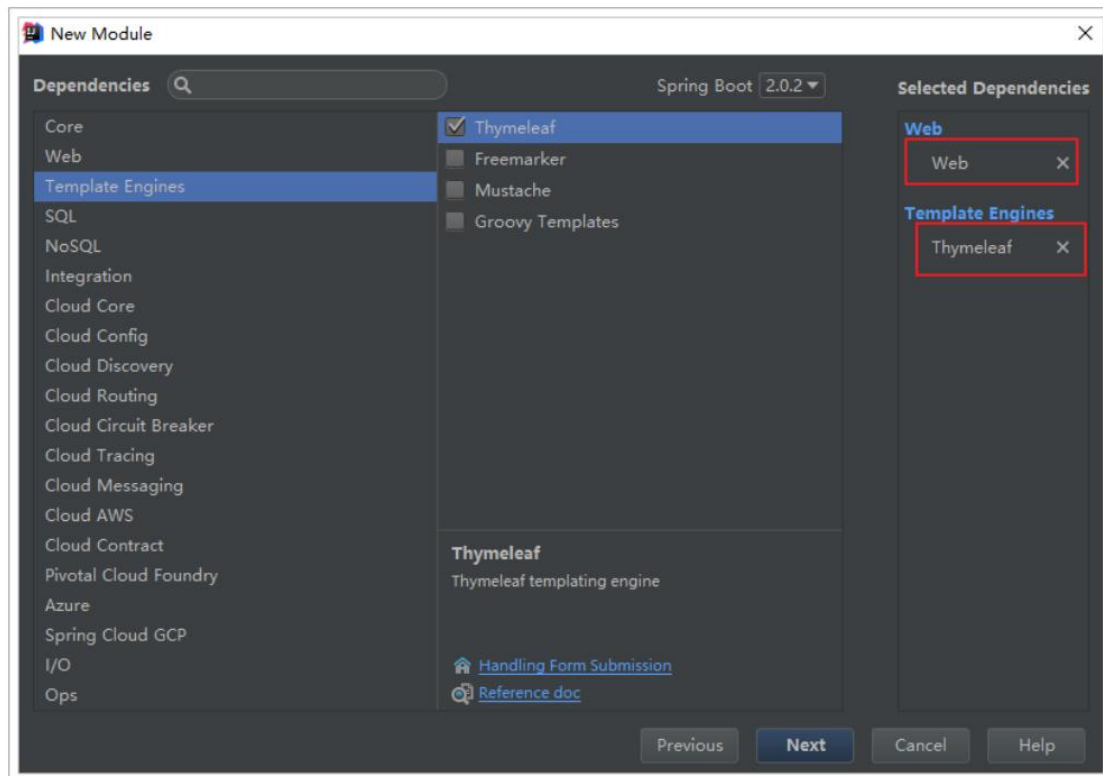
我们来创建一个 module, 为学习 Thymeleaf 做准备:

① 创建 module

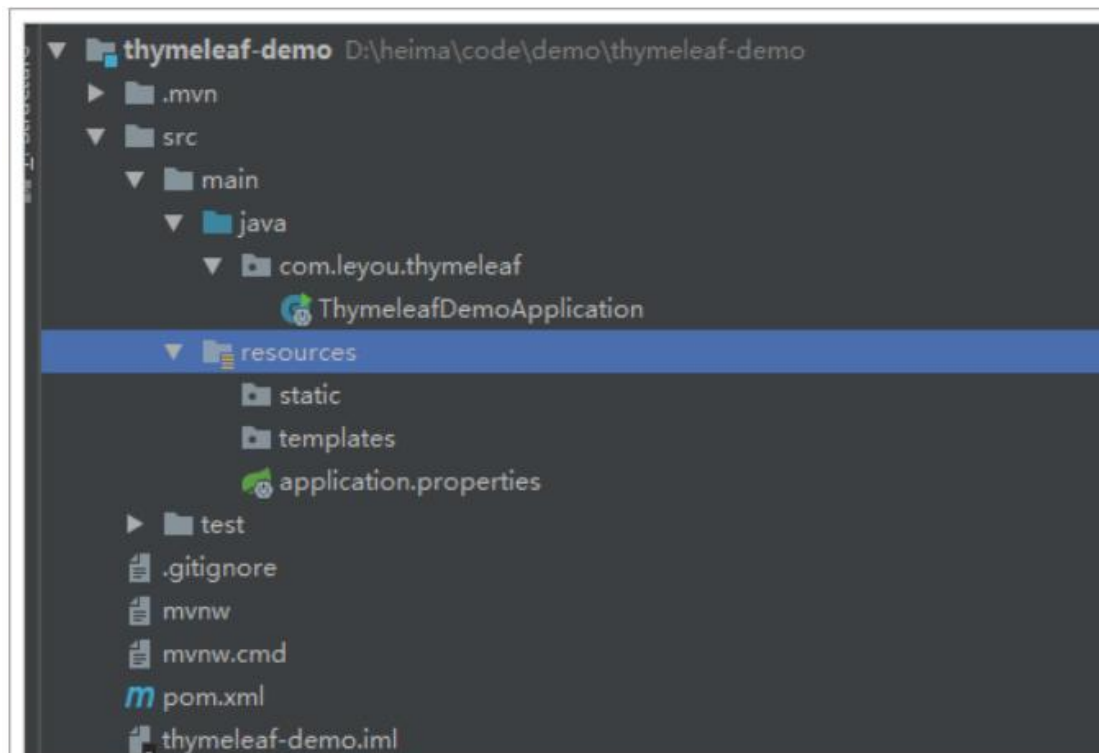
使用 spring 脚手架创建:



勾选 web 和 Thymeleaf 的依赖:



项目结构:



pom:

```
<?xml version="1.0" encoding="UTF-8"?>
<project                                xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.leyou.demo</groupId>
  <artifactId>thymeleaf-demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>thymeleaf-demo</name>
  <description>Demo project for Spring Boot</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.2.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

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

    <dependency>
```

```

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

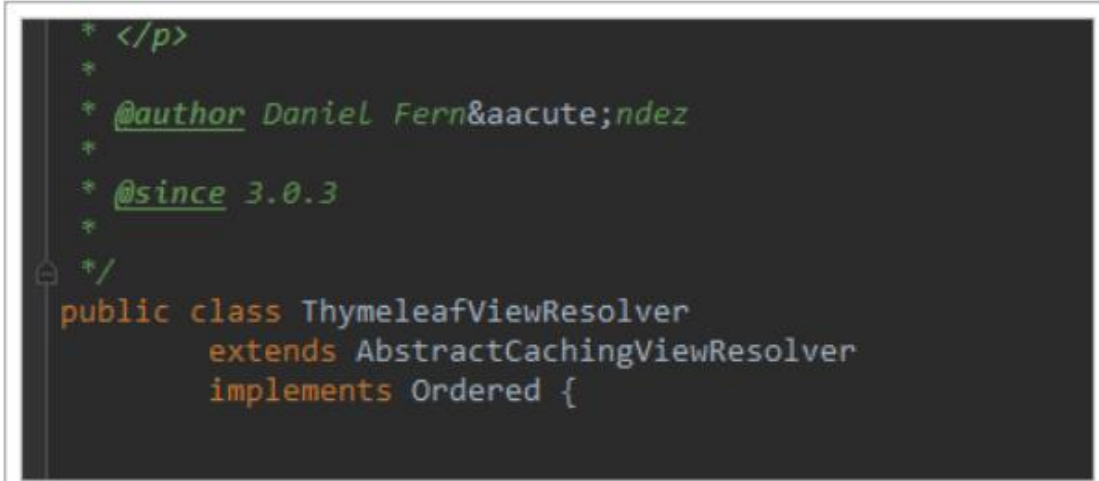
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>

</project>

```

② 默认配置

不需要做任何配置，启动器已经帮我们把 Thymeleaf 的视图器配置完成：



```

* </p>
*
* @author Daniel Fernandez
*
* @since 3.0.3
*
*/
public class ThymeleafViewResolver
    extends AbstractCachingViewResolver
    implements Ordered {

```

而且，还配置了模板文件（html）的位置，与 jsp 类似的前缀+ 视图名 + 后缀风格：

```

* @author Kazuki Shimizu
* @since 1.2.0
*/
@ConfigurationProperties(prefix = "spring.thymeleaf")
public class ThymeleafProperties {

    private static final Charset DEFAULT_ENCODING = StandardCharsets.UTF_8;

    public static final String DEFAULT_PREFIX = "classpath:/templates/";
    public static final String DEFAULT_SUFFIX = ".html";
}

```

默认前缀: classpath:/templates/

默认后缀: .html

所以如果我们返回视图: users, 会指向到 classpath:/templates/users.html

Thymeleaf 默认会开启页面缓存, 提高页面并发能力。但会导致我们修改页面不会立即被展现, 因此我们关闭缓存:

关闭 Thymeleaf 的缓存

spring.thymeleaf.cache=false

另外, 修改完毕页面, 需要使用快捷键: Ctrl + Shift + F9 来刷新工程。

3. 快速开始

① 准备一个 controller

- 控制视图跳转

```

@Controller
public class HelloController {

    @GetMapping("show1")
    public String show1(Model model){
        model.addAttribute("msg", "Hello, Thymeleaf!");
        return "hello";
    }
}

```

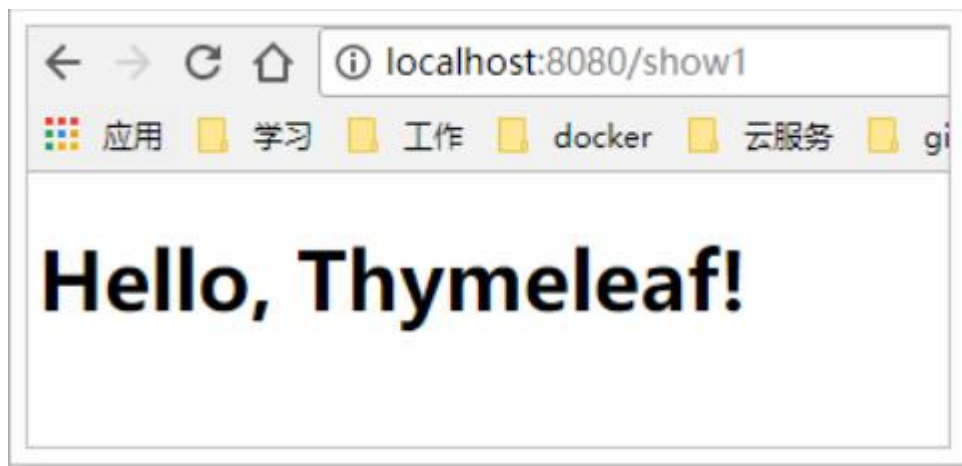
② 新建一个 html 模板

```
<!DOCTYPE html>
```

```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>hello</title>
</head>
<body>
  <h1 th:text="${msg}">大家好</h1>
</body>
</html>
```

注意，把 html 的名称空间，改成：`xmlns:th="http://www.thymeleaf.org"` 会有语法提示

③ 启动项目，访问页面：



4. 语法

Thymeleaf的主要作用是把 model 中的数据渲染到 html 中, 因此其语法主要是如何解析 model 中的数据。从以下方面来学习：

- 变量
- 方法
- 条件判断
- 循环
- 运算
 - ◆ 逻辑运算
 - ◆ 布尔运算
 - ◆ 比较运算

- ◆ 条件运算
- 其它

① 变量

变量案例

- 先新建一个实体类: User

```
public class User {  
    String name;  
    int age;  
    User friend;// 对象类型属性  
}
```

- 在模型中添加数据

```
@GetMapping("show2")  
public String show2(Model model){  
    User user = new User();  
    user.setAge(21);  
    user.setName("Jack Chen");  
    user.setFriend(new User("李小龙", 30));  
  
    model.addAttribute("user", user);  
    return "show2";  
}
```

语法说明:

Thymeleaf 通过\${} 来获取 model 中的变量, 注意这不是 el 表达式, 而是 ognl 表达式, 但是语法非常像。

示例:

我们在页面获取 user 数据:

```
<h1>  
    欢迎您: <span th:text="${user.name}">请登录</span>  
</h1>
```

- 效果:



感觉跟 el 表达式几乎是一样的。不过区别在于，我们的表达式写在一个名为：th:text 的标签属性中，这个叫做指令

② 动静结合

- 指令：

Thymeleaf 崇尚自然模板，意思就是模板是纯正的 html 代码，脱离模板引擎，在纯静态环境也可以直接运行。现在如果我们直接在 html 中编写 `${}` 这样的表达式，显然在静态环境下就会出错，这不符合 Thymeleaf 的理念。

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 `${user.name}` 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！现在，我们不经过 SpringMVC，而是直接用浏览器打开页面看看：



静态页面中，th 指令不被识别，但是浏览器也不会报错，把它当做一个普通属性处理。这样 span 的默认值请登录就会展现在页面

如果是在 Thymeleaf 环境下，th 指令就会被识别和解析，而 th:text 的含义就是替换所在标签中的文本内容，于是 user.name 的值就替代了 span 中默认的请登录

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。

- 向下兼容

但是要注意，如果浏览器不支持 Html5 怎么办？

如果不支持这种 `th:` 的命名空间写法，那么可以把 `th:text` 换成 `data-th-text`，Thymeleaf 也可以兼容。

- escape

另外，`th:text` 指令出于安全考虑，会把表达式读取到的值进行处理，防止 html 的注入。

例如，`<p>你好</p>` 将会被格式化输出为 `<p>你好</p>`。

如果想要不进行格式化输出，而是要输出原始内容，则使用 `th:utext` 来代替。

③ ognl 表达式的语法糖

刚才获取变量值，我们使用的是经典的对象.属性名方式。但有些情况下，我们的属性名可能本身也是变量，怎么办？

ognl 提供了类似 js 的语法方式：

例如：`${user.name}` 可以写作 `${user['name']}`

④ 自定义变量

场景

- 看下面的案例：

```
<h2>
  <p>Name: <span th:text="${user.name}">Jack</span>.</p>
  <p>Age: <span th:text="${user.age}">21</span>.</p>
  <p>friend: <span th:text="${user.friend.name}">Rose</span>.</p>
</h2>
```

我们获取用户的所有信息，分别展示。

当数据量比较多时，频繁的写 `user.` 就会非常麻烦。

因此，Thymeleaf 提供了自定义变量来解决：

- 示例：

```
<h2 th:object="${user}">
  <p>Name: <span th:text="*{name}">Jack</span>.</p>
  <p>Age: <span th:text="*{age}">21</span>.</p>
  <p>friend: <span th:text="*{friend.name}">Rose</span>.</p>
</h2>
```

首先在 `h2` 上用 `th:object="${user}"` 获取 `user` 的值，并且保存

然后，在 `h2` 内部的任意元素上，可以通过 `*{属性名}` 的方式，来获取 `user` 中的属性，这样就省去了大量的 `user.` 前缀了

⑤ 方法

- ognl 表达式中的方法调用

ognl 表达式本身就支持方法调用，例如：

```
<h2 th:object="${user}">
  <p>FirstName: <span th:text="*{name.split(' ')[0]}">Jack</span>.</p>
```

```
<p>LastName: <span th:text="*{name.split(' ')[1]}">Li</span>.</p>
</h2>
```

这里调用了 name（是一个字符串）的 split 方法。

● Thymeleaf 内置对象

Thymeleaf 中提供了一些内置对象，并且在这些对象中提供了一些方法，方便我们来调用。获取这些对象，需要使用#对象名来引用。

➤ 一些环境相关对象

对象	作用
#ctx	获取 Thymeleaf 自己的 Context 对象
#request	如果是 web 程序，可以获得 HttpServletRequest 对象
#response	如果是 web 程序，可以获得 HttpServletResponse 对象
#session	如果是 web 程序，可以获得 HttpSession 对象
#servletContext	如果是 web 程序，可以获得 ServletContext 对象

➤ 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	用来处理数组的方法
#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

● 举例

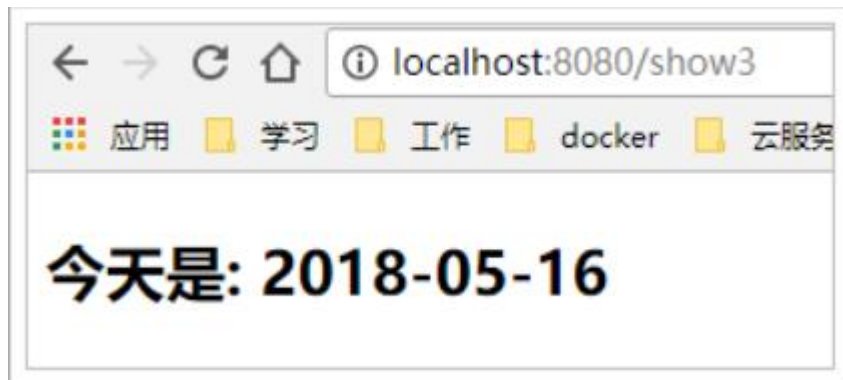
我们在环境变量中添加日期类型对象

```
@GetMapping("show3")
public String show3(Model model){
    model.addAttribute("today", new Date());
    return "show3";
}
```

在页面中处理

```
<p>
    今天是: <span th:text="${#dates.format(today,'yyyy-MM-dd')}">2018-04-25</span>
</p>
```

- 效果:



⑥ 字面值

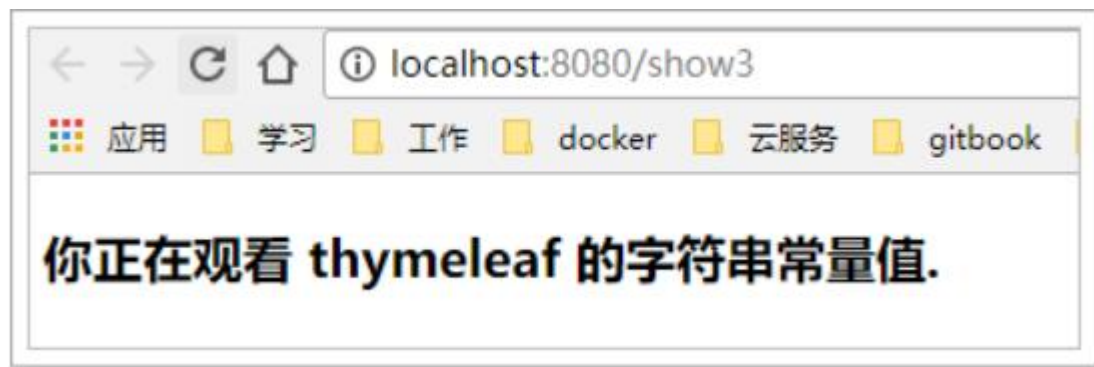
有的时候, 我们需要在指令中填写基本类型如: 字符串、数值、布尔等, 并不希望被 Thymeleaf 解析为变量, 这个时候称为字面值。

- 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>
```

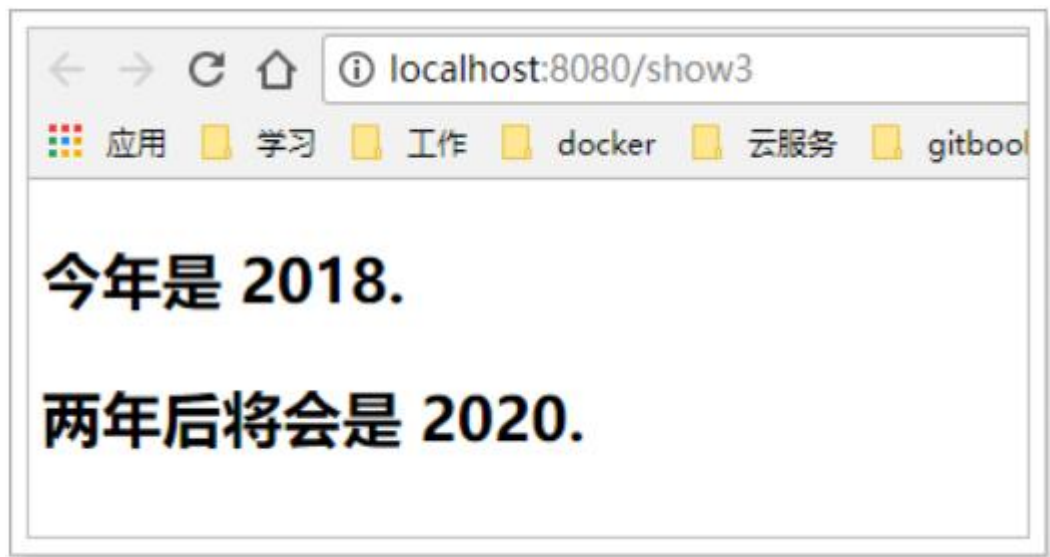
th:text 中的 thymeleaf 并不会被认为是变量, 而是一个字符串



- 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="2018">1900</span>.</p>
<p>两年后将会是 <span th:text="2018 + 2">1902</span>.</p>
```



- 布尔字面值

布尔类型的字面值是 true 或 false:

```
<div th:if="true">
    你填的是 true
</div>
```

这里引用了一个 th:if 指令，跟 vue 中的 v-if 类似

⑦ 拼接

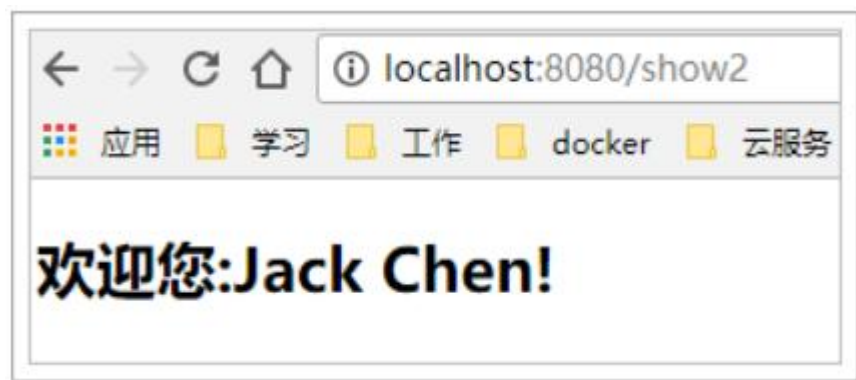
我们经常会用到普通字符串与表达式拼接的情况:

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

与上面是完全等效的，这样就省去了字符串字面值的书写。



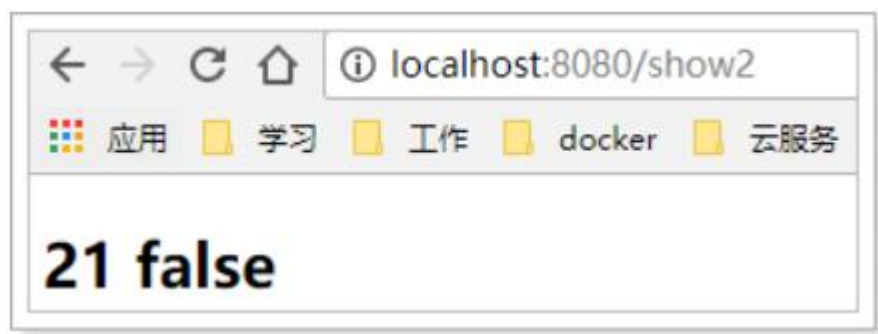
⑧ 运算

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。

- 算术运算

支持的算术运算符: `+` `-` `*` `/` `%`

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



- 比较运算

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

注意 `==` and `!=` 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 (!)`. Also `eq (==)`, `neq/ne (!=)`.

- 条件运算

- 三元运算

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

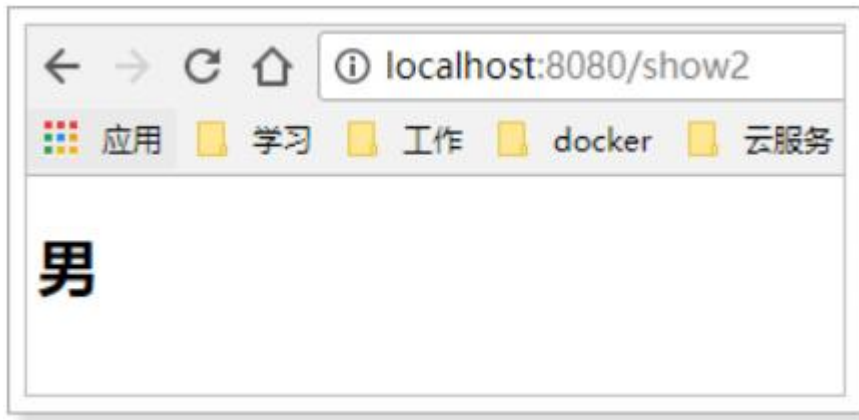
三元运算符的三个部分: `conditon ? then : else`

`condition`: 条件

`then`: 条件成立的结果

`else`: 不成立的结果

其中的每一个部分都可以是 Thymeleaf 中的任意表达式。



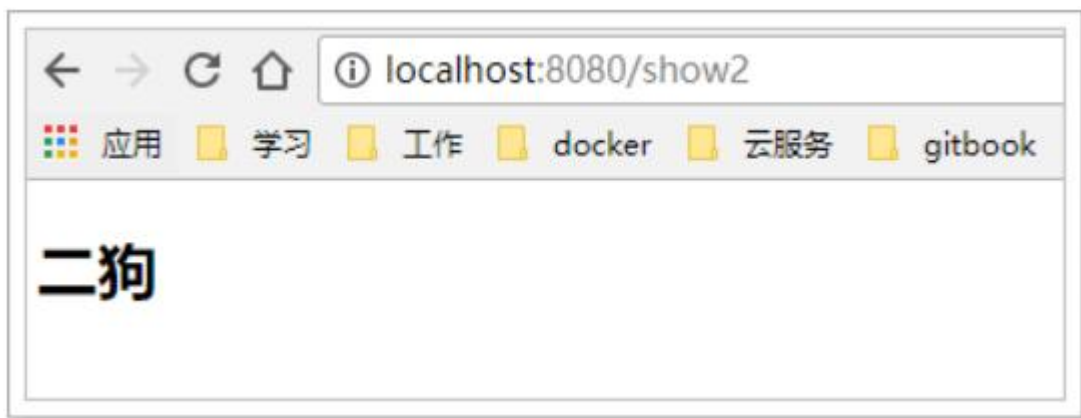
➤ 默认值

有的时候，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：

```
<span th:text="${user.name} ?: '二狗'"></span>
```

当前面的表达式值为 `null` 时，就会使用后面的默认值。

注意：`?:`之间没有空格。



⑨ 循环

循环也是非常频繁使用的需求，我们使用 `th:each` 指令来完成：

假如有用户的集合： `users` 在 `Context` 中。

```
<tr th:each="user : ${users}">
  <td th:text="${user.name}">Onions</td>
  <td th:text="${user.age}">2.41</td>
</tr>
```

`${users}` 是要遍历的集合，可以是以下类型：

- Iterable, 实现了 Iterable 接口的类
- Enumeration, 枚举
- Iterator, 迭代器
- Map, 遍历得到的是 Map.Entry
- Array, 数组及其它一切符合数组结果的对象

在迭代的同时, 我们也可以获取迭代的状态对象:

```
<tr th:each="user,stat : ${users}">
  <td th:text="${user.name}">Onions</td>
  <td th:text="${user.age}">2.41</td>
</tr>
```

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

⑩ 逻辑判断

Thymeleaf 中使用 th:if 或者 th:unless, 两者的意思恰好相反。

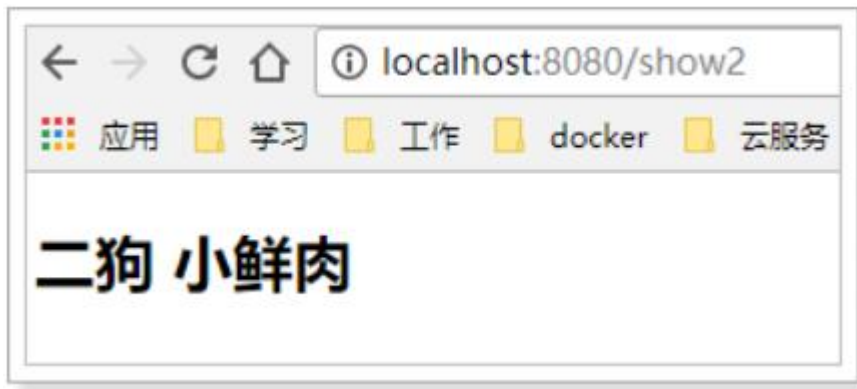
```
<span th:if="${user.age} < 24">小鲜肉</span>
```

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



⑪ 分支控制 switch

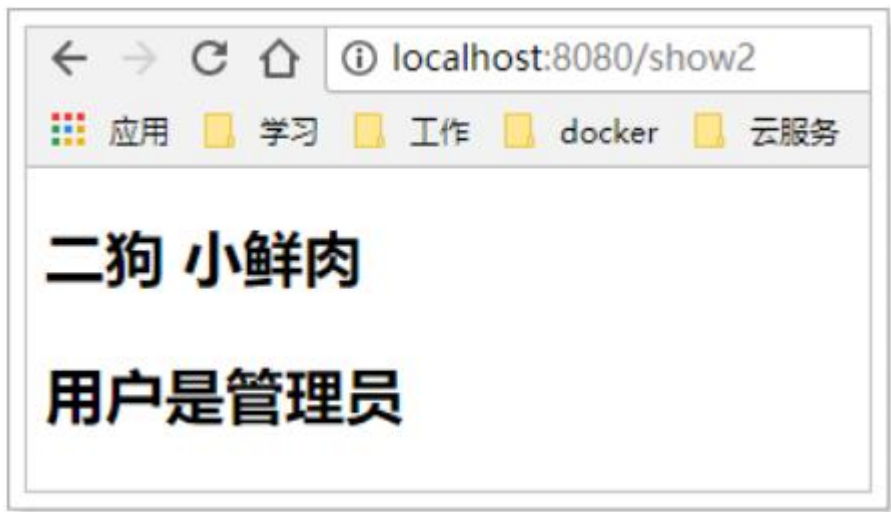
这里要使用两个指令: `th:switch` 和 `th:case`

```
<div th:switch="{user.role}">
  <p th:case="admin">用户是管理员</p>
  <p th:case="manager">用户是经理</p>
  <p th:case="*">用户是别的玩意</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。

```
@GetMapping("show2")
public String show2(Model model) {
    User user = new User();
    user.setAge(21);
    user.setSex(true);
    user.setRole("admin");
}
```

页面:



⑫ 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：

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

    const user = /*[[${user}]]*/ {};

    const age = /*[[${user.age}]]*/ 20;

    console.log(user);

    console.log(age)

</script>
```

在 script 标签中通过 th:inline="javascript" 来声明这是要特殊处理的 js 脚本

- 语法结构：

const user = /*[[Thymeleaf 表达式]]*/ "静态环境下的默认值";

因为 Thymeleaf 被注释起来，因此即便是静态环境下，js 代码也不会报错，而是采用表达式后面跟着的默认值。

看看页面的源码：

```
<html lang="en">
  <head>...</head>
  <body>
    <h2>
      <span>二狗</span>
      <span>小鲜肉</span>
    </h2>
    <div == $0
      <p>用户是管理员</p>
    </div>
  </body>
  <script>
    const user = {"name":null,"age":21,"friend":
    {"name":"\u674E\u5C0F\u9F99","age":30,"friend":null,"sex":false,"role":null},"sex":true,"role":"admin"};
    const age = 21;
    console.log(user);
    console.log(age)
  </script>
</html>
```

我们的 User 对象被直接处理为 json 格式了，非常方便。

控制台：

