

1.Thymeleaf简介

官方网站: <https://www.thymeleaf.org/index.html>

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。

Thymeleaf的主要目标是为您的开发工作流程带来优雅的*自然模板*- HTML。可以在直接浏览器中正确显示,并且可以作为静态原型,从而在开发团队中实现更强大的协作。

借助Spring Framework的模块,可以根据自己的喜好进行自由选择,可插拔功能组件, Thymeleaf是现代HTML5 JVM Web开发的理想选择 - 尽管它可以做的更多。

Spring官方支持的服务的渲染模板中,并不包含jsp。而是Thymeleaf和Freemarker等,而Thymeleaf与SpringMVC的视图技术,及SpringBoot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。

2.特点

特点:

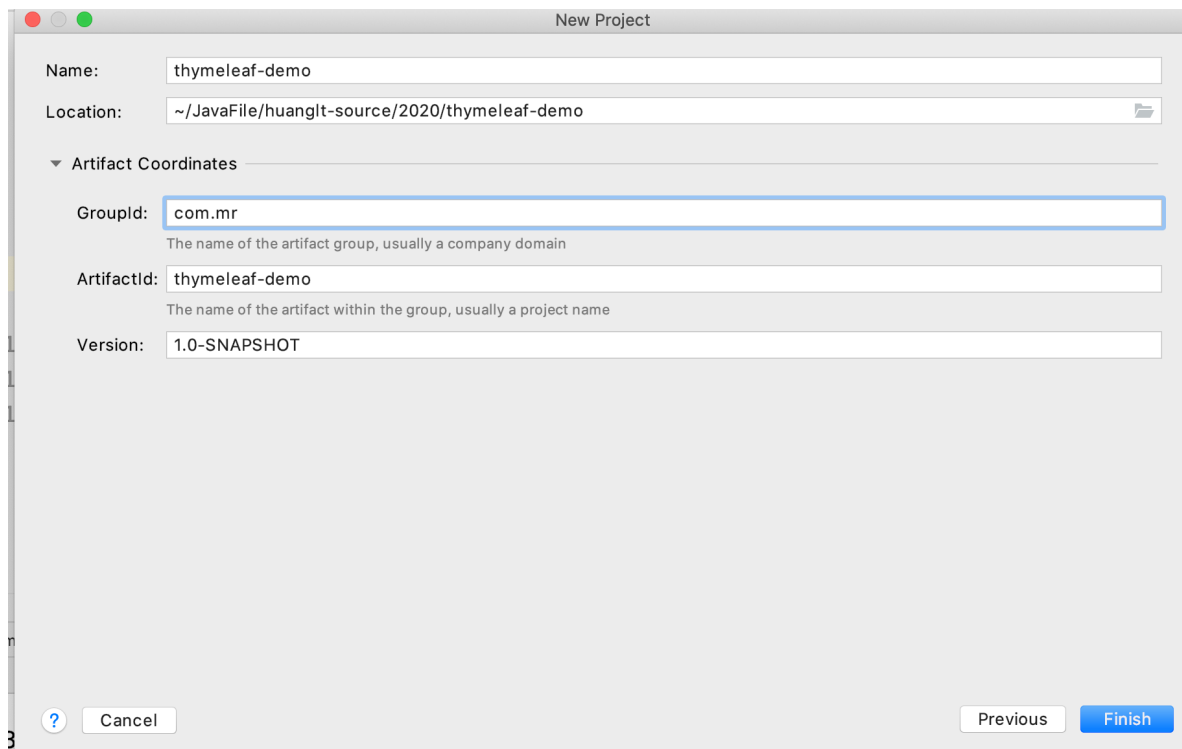
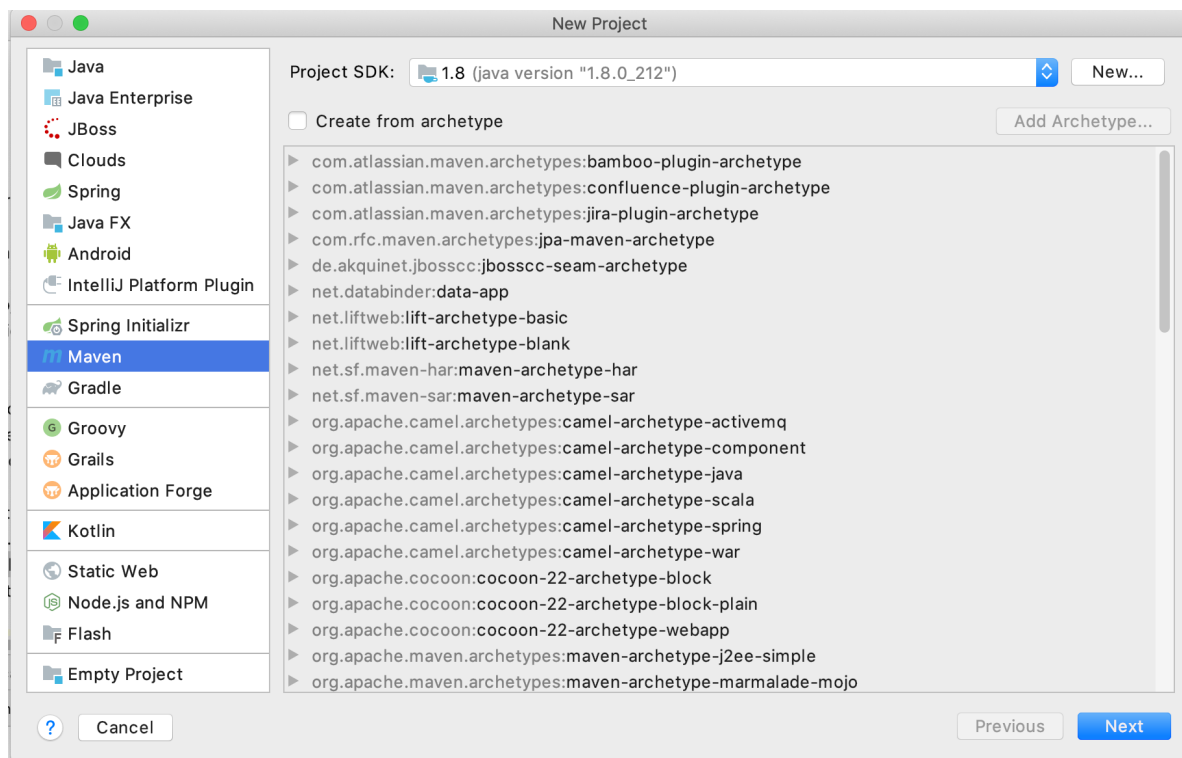
- 动静结合: Thymeleaf 在有网络和无网络的环境下皆可运行,即它可以让美工在浏览器查看页面的静态效果,也可以让程序员在服务器查看带数据的动态页面效果。这是由于它支持 html 原型,然后在 html 标签里增加额外的属性来达到模板+数据的展示方式。浏览器解释 html 时会忽略未定义的标签属性,所以 thymeleaf 的模板可以静态地运行;当有数据返回到页面时, Thymeleaf 标签会动态地替换掉静态内容,使页面动态显示。
- 开箱即用: 它提供标准和spring标准两种方言,可以直接套用模板实现JSTL、OGNL表达式效果,避免每天套模板、改jstl、改标签的困扰。同时开发人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。

3.环境准备

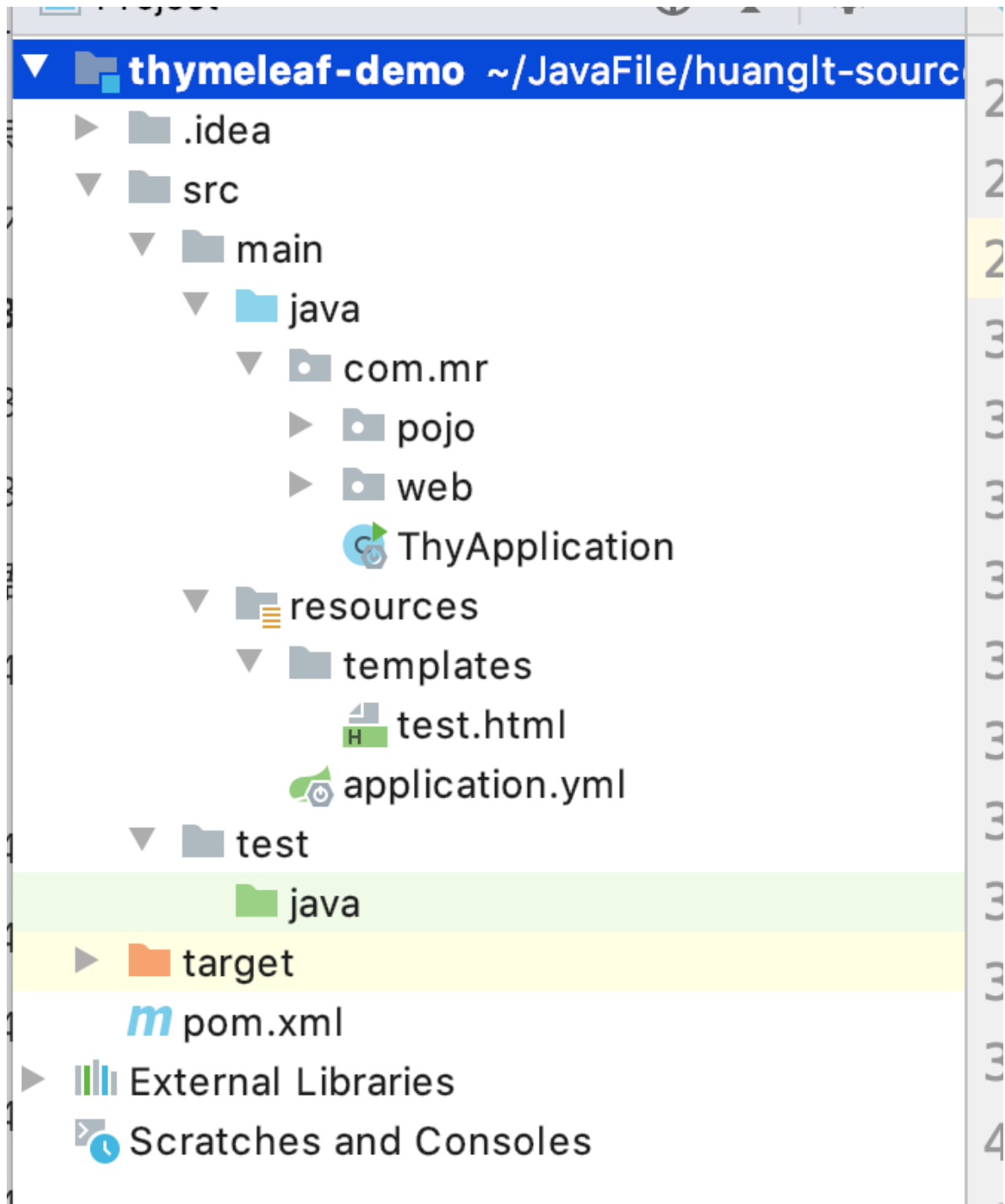
创建一个project

3.1.创建project

使用spring 脚手架创建:



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.mr</groupId>
  <artifactId>thymeleaf-demo</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <!--spring boot 依赖-->
  <parent>
```

```

    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.2.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<dependencies>
    <!--thymeleaf 依赖-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <!--web mvc 模块依赖-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

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

<build>
    <plugins>
        <!--maven 打包便衣依赖-->
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>

</project>

```

yaml配置

```

server: #端口
  port: 8083
spring: #清除页面缓存
  thymeleaf:
    prefix: file:src/main/resources/templates/
    cache: false

```

启动类

```

package com.mr;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ThyApplication {

    public static void main(String[] args) {
        SpringApplication.run(ThyApplication.class);
    }
}

```

3.2.默认配置

新建templates包结构在resources下

thymeleaf默认从templates读取静态页面

- 默认前缀: `classpath:/templates/`
- 默认后缀: `.html`

3.3.快速开始

我们准备一个controller，控制视图跳转：

```

@Controller//controller可以返回页面
public class TestController {

    @GetMapping("test")//定义url使用modelmap返回数据
    public String test(ModelMap map){
        map.put("name", "tomcat");
        return "test";
    }
}

```

新建一个html模板：

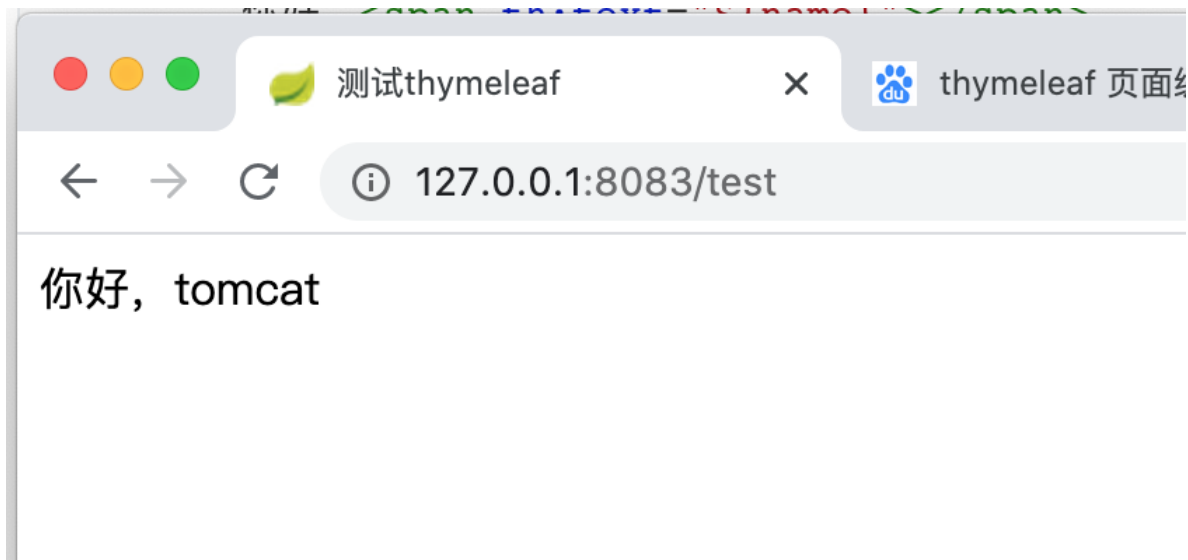
```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org" >
<head>
    <meta charset="UTF-8">
    <title>测试thymeleaf</title>
</head>
<body>
    你好, <span th:text="${name}"></span>
</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

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

4.1.变量

变量案例

我们先新建一个实体类：Student

```
public class Student {  
    String code;  
    String pass;  
    int age;  
    String likeColor;  
}
```

然后在模型中添加数据

```
/**  
 * 展示学生
```

```

    * @param map
    * @return
    */
    @GetMapping("stu")
    public String student(ModelMap map){
        Student student=new Student();
        student.setCode("007");
        student.setPass("9527");
        student.setAge(18);
        student.setLikeColor("<font color='red'>红色</font>");
        map.put("stu",student);
        return "student";
    }

```

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：

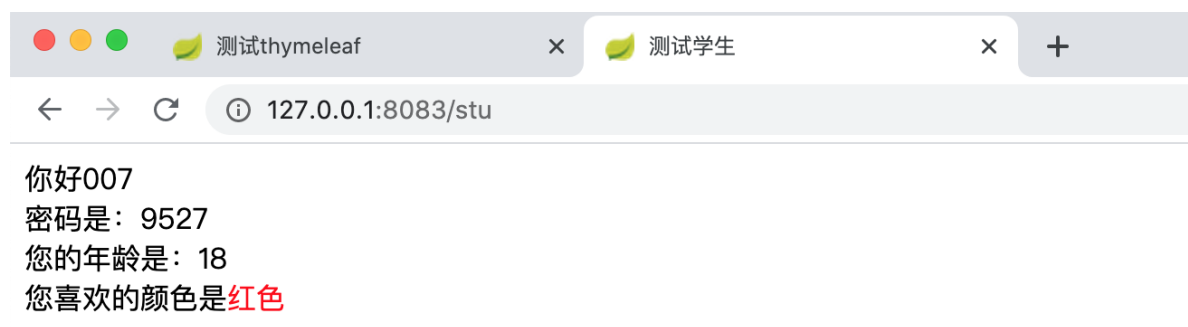
我们在页面获取stu数据：

```

<!-- th:指令依赖于h5-->
    你好<span th:text="${stu.code}"></span><br>
<!-- data-th-text 指令支持非 h5-->
    密码是：<span data-th-text="${stu.pass}"></span><br>
<!--对象['属性值'] 如果变量是动态的，可以用此属性-->
    您的年龄是：<span th:text="${stu['age']}"></span><br>
<!--对象的value中有html需要渲染的使用 utext 输出-->
    您喜欢的颜色是<span th:utext="${stu.likeColor}"></span>

```

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

但是要注意，`th` 指令是依赖于h5的

如果浏览器不支持h5 那么可以使用 `data-th-text` 指令

如果值中包含html代码需要渲染的 则使用 `th:utext` 来输出。

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，

ognl提供了类似js的语法方式：

例如: `${stu.age}` 可以写作 `${stu['age']}`

4.2.自定义变量

场景

看下面的案例:

```
<!-- th:指令依赖于h5-->
    你好<span th:text="${stu.code}"></span><br>
<!-- data-th-text 指令支持非 h5-->
    密码是: <span data-th-text="${stu.pass}"></span><br>
<!--对象['属性值'] 如果变量是动态的, 可以用此属性-->
    您的年龄是: <span th:text="${stu['age']}"></span><br>
<!--对象的value中有html需要渲染的使用 utext 输出-->
    您喜欢的颜色是<span th:utext="${stu.likeColor}"></span>
```

我们获取用户的所有信息, 分别展示。

当数据量比较多的时候, 频繁的写 `stu.` 就会非常麻烦。

因此, Thymeleaf提供了自定义变量来解决:

示例:

```
<div th:object="${stu}">
    <span th:text="*{code}"></span><br>
    <span th:text="*{pass}"></span><br>
    <span th:text="*{age}"></span><br>
    <span th:text="*{likeColor}"></span><br>
</div>
```

- 首先在 `div` 上用 `th:object="${stu}"` 获取`stu`的值, 并且保存
- 然后, 在 `div` 内部的任意元素上, 可以通过 `*{属性名}` 的方式, 来获取`stu`中的属性, 这样就省去了大量的 `stu.` 前缀了

4.3.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:

```
<!-- 可以调用字符串截取, 分割等函数-->
<span th:text="${stu.code.substring(2)}"></span><br>
<span th:text="${stu.code.split(' ')[1]}"></span><br>
```

- 可以调用字符串截取, 分割等函数

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
<code>#ctx</code>	获取Thymeleaf自己的Context对象
<code>#request</code>	如果是web程序，可以获取HttpServletRequest对象
<code>#response</code>	如果是web程序，可以获取HttpServletResponse对象
<code>#session</code>	如果是web程序，可以获取HttpSession对象
<code>#servletContext</code>	如果是web程序，可以获取ServletContext对象

- Thymeleaf提供的全局对象：

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

4.4 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：

```
<span th:text="'666'"></span>
```

`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

```
数字 :<span th:text="1024*2"></span>
```

- 布尔字面值 结合if使用

布尔类型的字面值是true或false:

```
<span th:if="${stu.age} >=18">
    成年
</span>
```

这里引用了一个 `th:if` 指令, 跟vue中的 `v-if` 类似

4.5 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="'你好' + ${stu.code} + '真的6666'"></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="'|你好${stu.code}真的6666|'"></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。

字符串: 666

数字 :2048

成年

你好007真的6666

你好 007 真的6666

4.6 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="${stu.age} *2"></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/neq (!=)`.

- 条件运算
 - 三元运算

```
<span th:text="${stu.age} >=18 ? '成年' : '未成年' "></span>
```

三元运算符的三个部分：condition ? 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。

成年

- 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="${stu.code} ?: ''"></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。

4.7 循环

```
@GetMapping("list")
public String list(ModelMap map){
    Student s1=new Student("001","111",18,"red");
    Student s2=new Student("002","222",19,"red");
    Student s3=new Student("003","333",16,"blue");
    Student s4=new Student("004","444",28,"blue");
    Student s5=new Student("005","555",68,"blue");

    //转为List
    map.put("stuList", Arrays.asList(s1,s2,s3,s4,s5));
    return "list";
}
```

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：stuList在Context中。

```

<ul>
  <li> >编码--密码--年龄</li>
  <li th:each="stu : ${stuList}">
    <span th:text="${stu.code}"></span>--
    <span th:text="${stu.pass}"></span>--
    <span th:text="${stu.age}"></span>
  </li>
</ul>

```

- `${stuList}` 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：

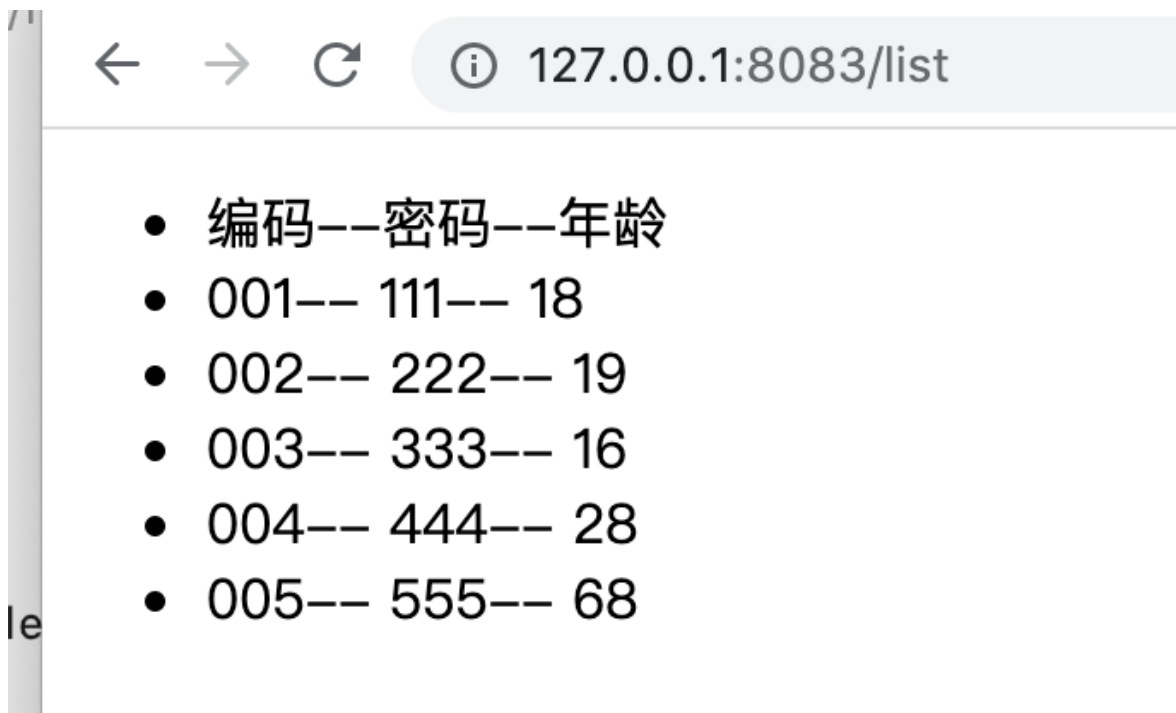
```

<ul>
  <li> >下标--编码--密码--年龄</li>
  <li th:each="stu, stat : ${stuList}">
    <span th:text="${stat.index}"></span>--
    <span th:text="${stu.code}"></span>--
    <span th:text="${stu.pass}"></span>--
    <span th:text="${stu.age}"></span>
  </li>
</ul>

```

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



4.8 逻辑判断

if和else

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

```
<span th:if="${stu.age>60}">可以退休</span>
<span th:unless="${stu.age>60}">不能退休</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

- 编码--密码--年龄
- 001-- 111-- 不能退休
- 002-- 222-- 不能退休
- 003-- 333-- 不能退休
- 004-- 444-- 不能退休
- 005-- 555-- 可以退休

4.9 分支控制switch

这里要使用两个指令：`th:switch` 和 `th:case`

```
<!--switch 选择-->
<span th:switch="${stu.code}">
  <span th:case="'001'">1号员工，骨灰级</span>
  <span th:case="'002'">2号员工，元老级</span>
  <span th:case="'003'">3号员工，老员工</span>
  <span th:case="'004'">4号员工，新员工</span>
  <span th:case="*">临时工</span>
</span>
```

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。

效果

- 编码--密码--退休状态---资历
- 001-- 111-- 不能退休--- 1号员工，骨灰级
- 002-- 222-- 不能退休--- 2号员工，元老级
- 003-- 333-- 不能退休--- 3号员工，老员工
- 004-- 444-- 不能退休--- 4号员工，新员工
- 005-- 555-- 可以退休 --- 临时工

4.10.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">
    //预处理js值
    const stuList=/*[[${stuList}]]*/;
    const stu=/*[[${stuList[0]}]]*/;
    const age=/*[[${stuList[0].age}]]*/;

    console.log("集合: "+stuList);
    console.log("对象: "+stu);
    console.log("属性: "+age);

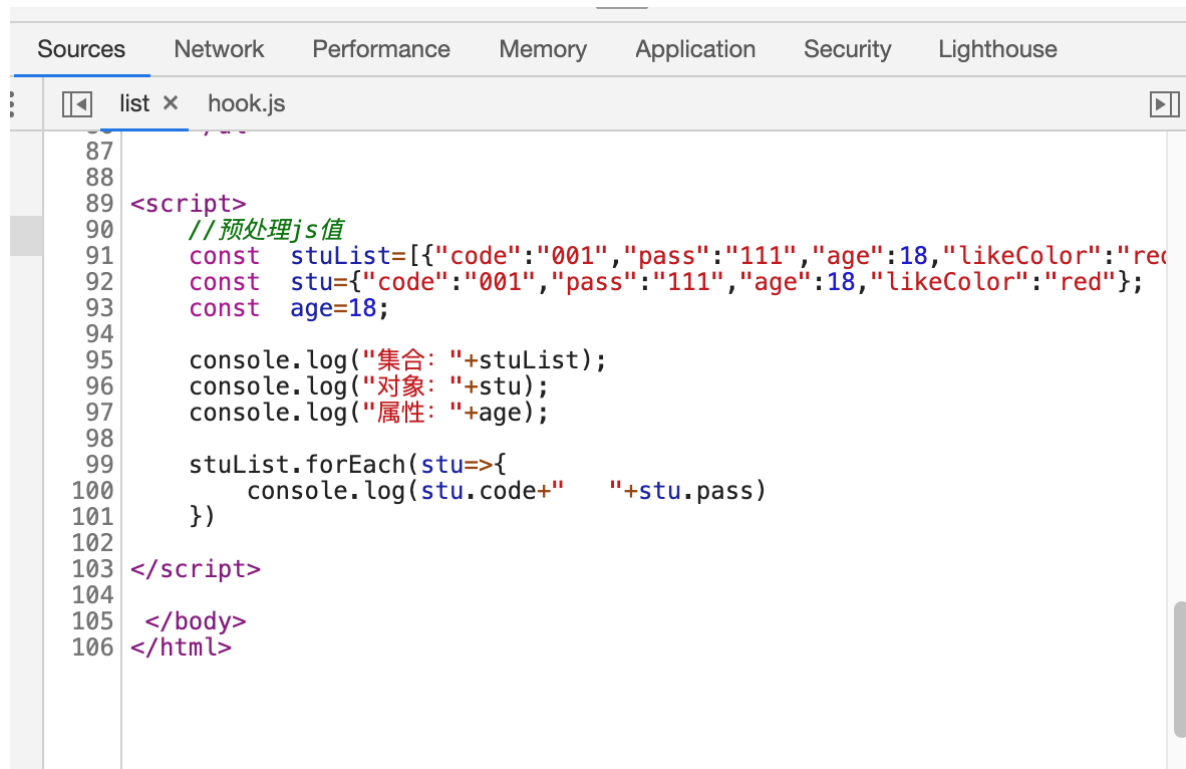
    stuList.forEach(stu=>{
        console.log(stu.code+" "+stu.pass)
    })
</script>
```

- 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 obj = /*[[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：



```
87
88
89 <script>
90     //预处理js值
91     const stuList=[{"code":"001","pass":"111","age":18,"likeColor":"red"}];
92     const stu={"code":"001","pass":"111","age":18,"likeColor":"red"};
93     const age=18;
94
95     console.log("集合: "+stuList);
96     console.log("对象: "+stu);
97     console.log("属性: "+age);
98
99     stuList.forEach(stu=>{
100         console.log(stu.code+" "+stu.pass)
101     })
102
103 </script>
104
105 </body>
106 </html>
```

我们的stuList对象被直接处理为json格式了，非常方便。

控制台：

Elements

Console

Sources

Network

Performance

Memory

Application

Security

Lighthouse

top

Filter

Default levels

集合: [object Object],[object Object],[object Object],[object Object],[object Object]

对象: [object Object]

属性: 18

001111

002222

003333

004444

005555

> |