

6.3 Spring Boot 基础

- 创建项目
- 创建基本网站
- Thymeleaf

目录

1

创建项目

2

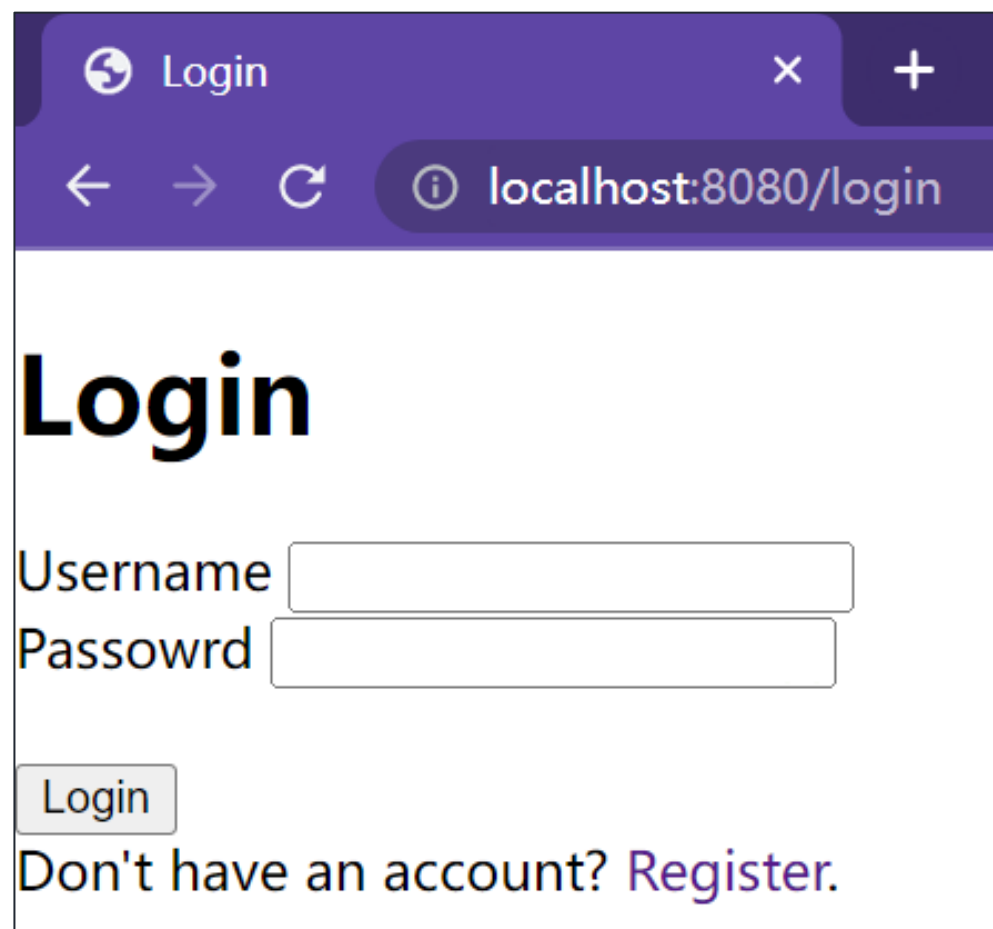
创建基本网站

3

Thymeleaf

目标：一个简单登录系统

- 本节课的目标是创建一个简单的登录网站，暂不与数据库交互。
- 首先，让我们一起来创建 Spring Boot 项目。



The screenshot shows a web browser window with a purple header. The address bar displays 'localhost:8080/login'. The page content includes a large 'Login' heading, two input fields for 'Username' and 'Passowrd', a 'Login' button, and a link to 'Register' for users without an account.

Login

Username

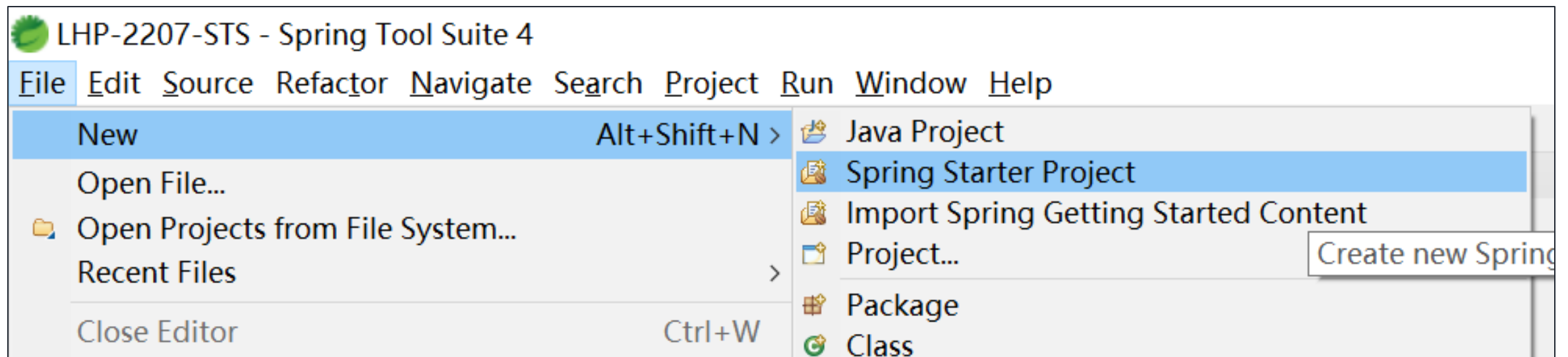
Passowrd

Login

Don't have an account? [Register.](#)

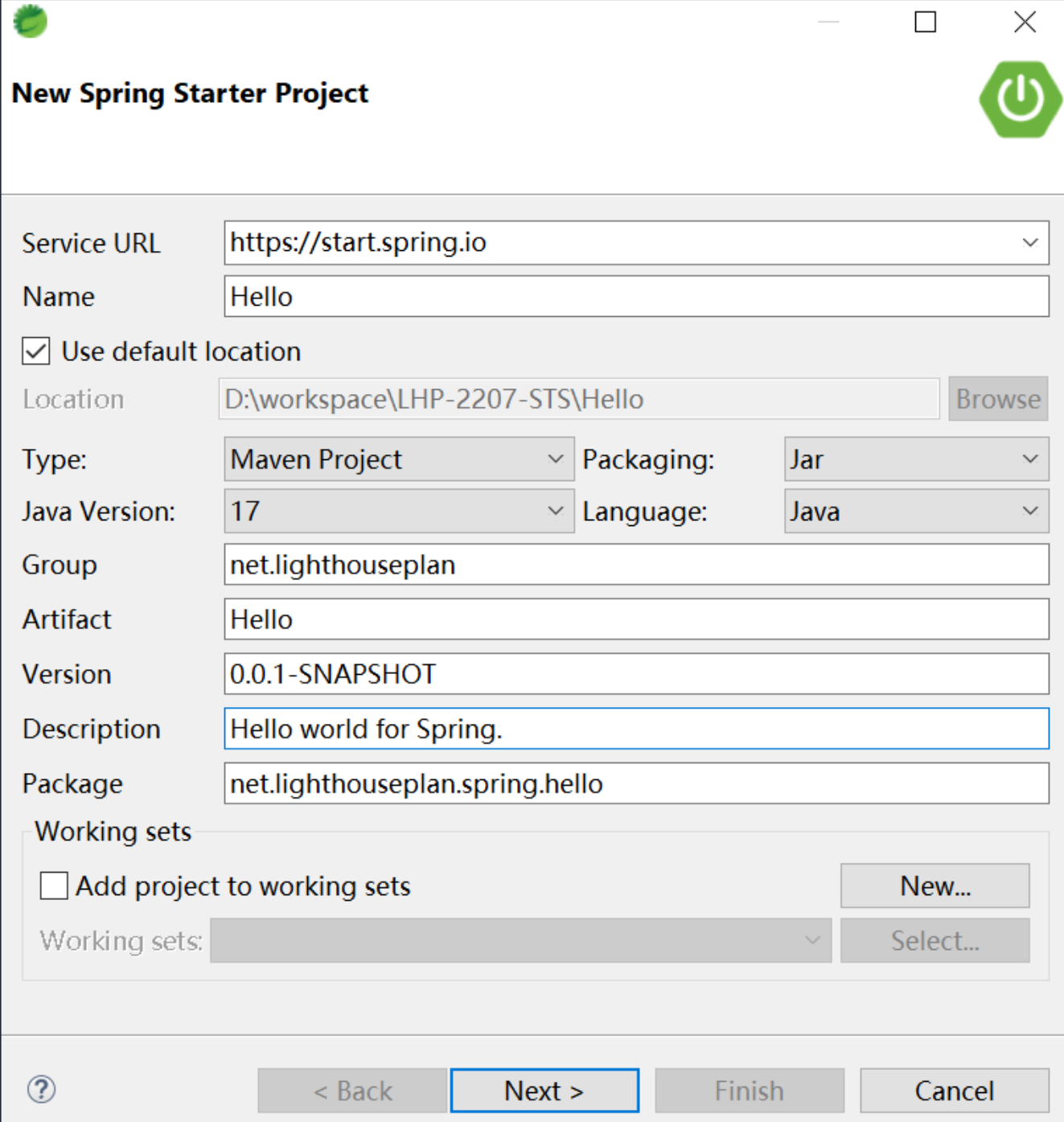
步骤 1：创建项目

- 点击目录区的 Create new Spring Starter Project。
- 如果没有该选项（比如已有现成项目），点击菜单栏的 File → New → Spring Starter Project。



步骤 2：配置项目信息

- 设置一些项目相关的配置信息为所需内容，点击 Next。



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

Package:

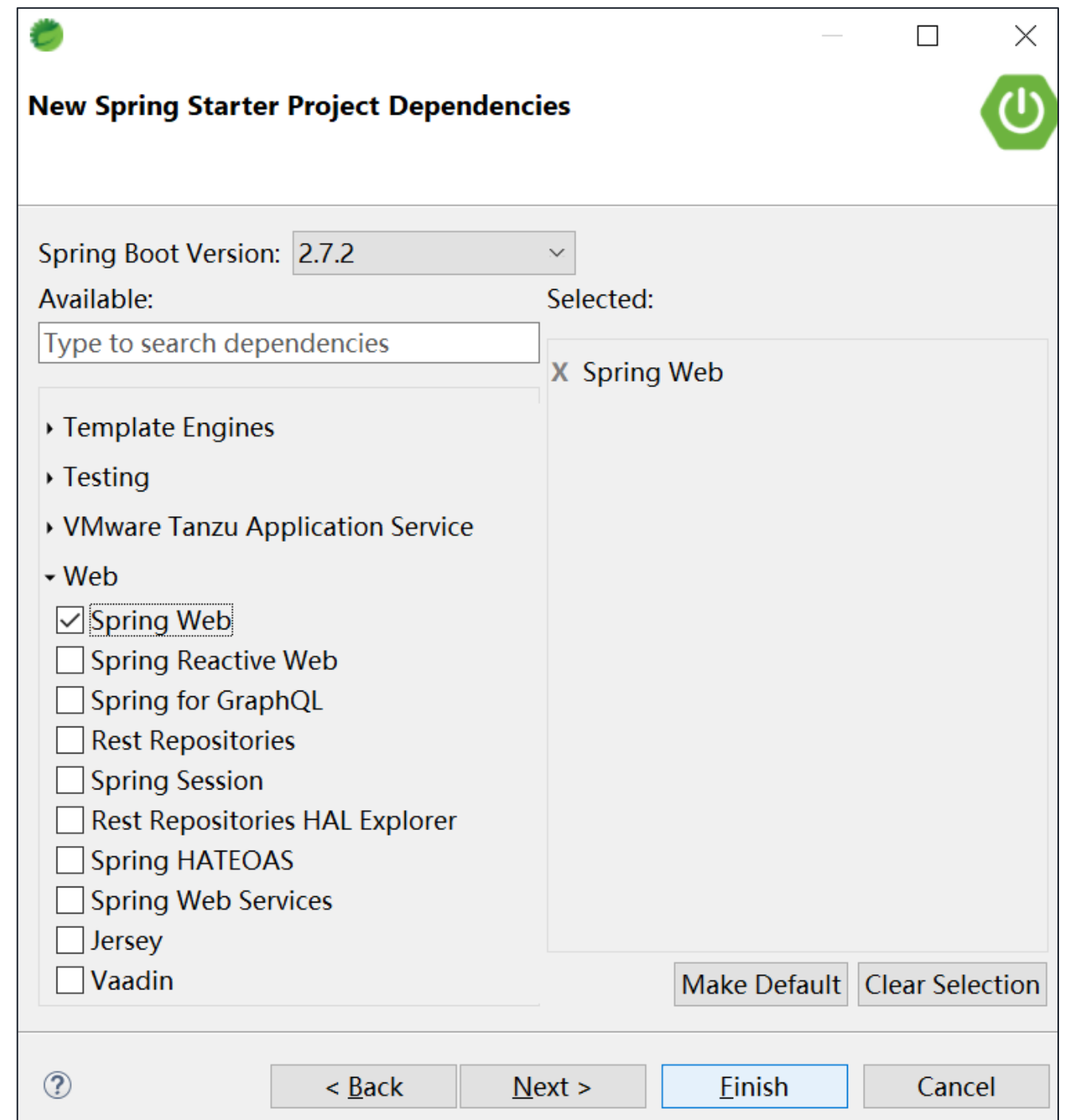
Working sets

☐ Add project to working sets

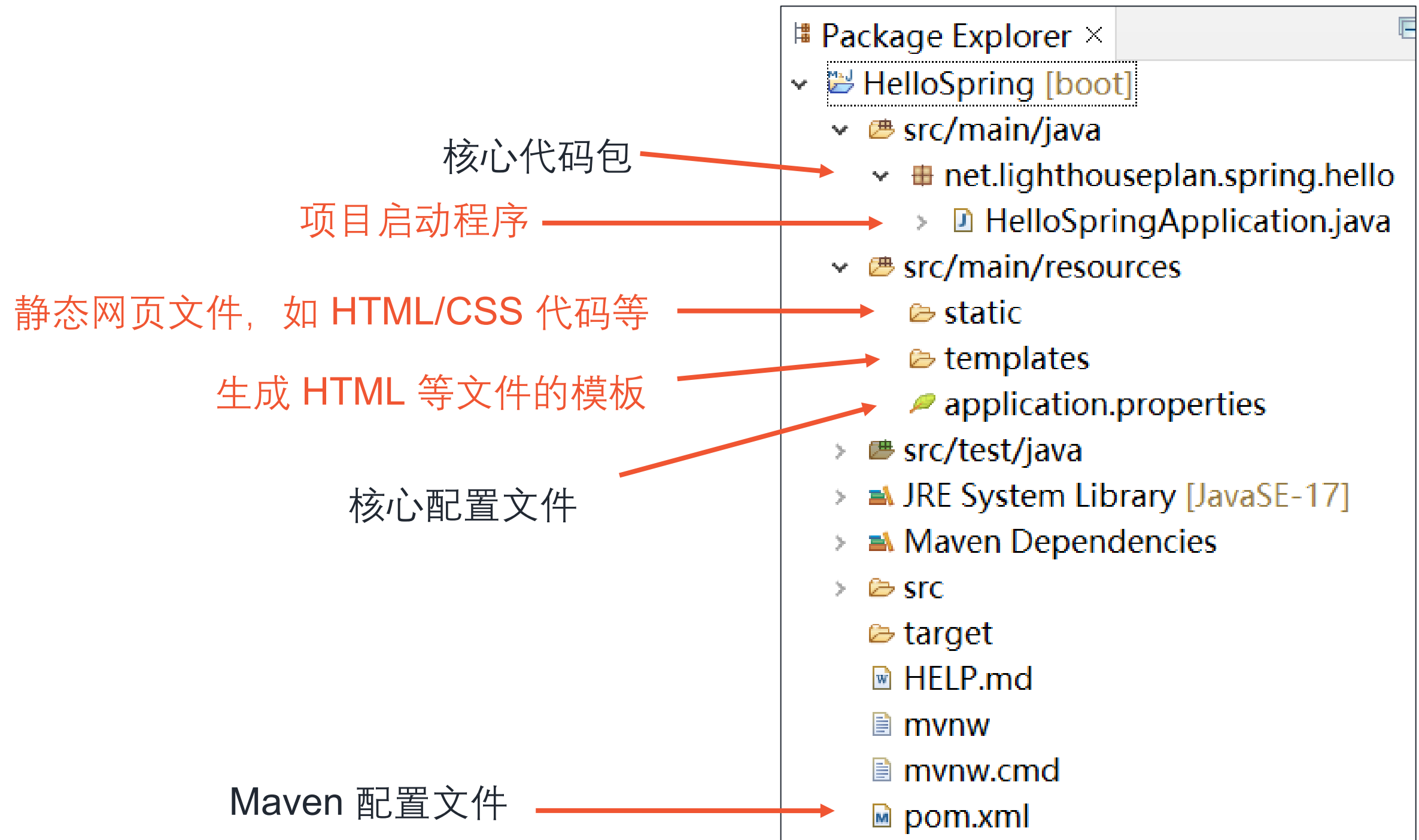
Working sets:

步骤 3：添加依赖

- 在这里，你可以提前搜索并添加所需依赖。你也可以之后再使用 Maven 添加依赖。
- 这里，添加 Web → Spring Web 依赖，点击 Finish。



项目结构



参考资料

- Spring Boot 官方 API:
 <https://docs.spring.io/spring-boot/docs/current/api/>。
- Spring Boot 日文参考手册:
 <https://spring.pleiades.io/projects/spring-boot>
- Spring Boot 中文参考手册:
 <https://www.springcloud.cc/spring-boot.html>

Q & A

Question and answer

目录

1

创建项目

2

创建基本网站

3

Thymeleaf

项目启动程序

- STS 创建的 Spring Starter 项目从一开始就带有一个 Java 源代码文件。
- 打开 **[你的项目名]Application.java** 文件。可以看见，该文件已有一个 main 方法，它将启动整个 Spring Boot 项目。我们暂时无需对其作任何改动：

```
1 @SpringBootApplication
2 public class HelloSpringApplication {
3
4     public static void main(String[] args) {
5         SpringApplication.run(HelloSpringApplication.class, args);
6     }
7
8 }
```


最基本的控制器

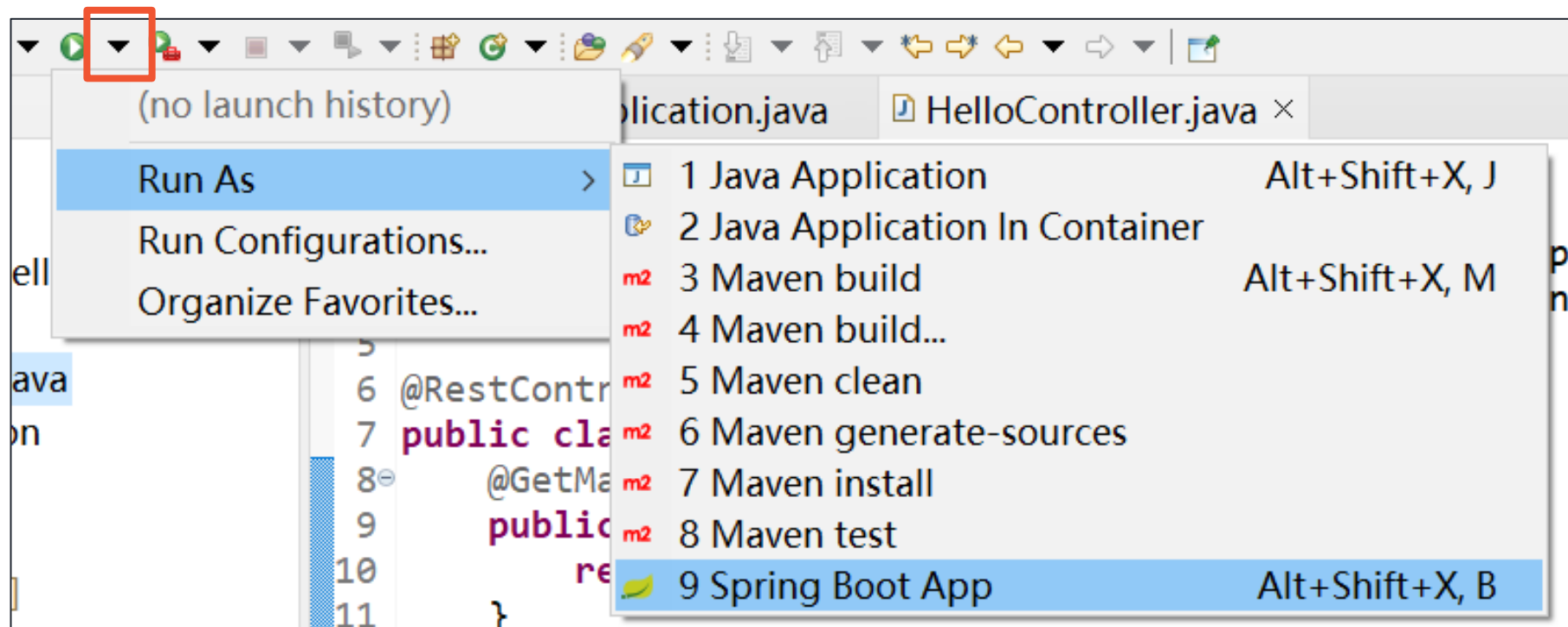
- 创建一个名为 `HelloController.java` 的 Java 代码文件，向其中添加以下内容：

```
1 @RestController
2 public class HelloController {
3     @GetMapping("/")
4     public String index() {
5         return "Hello, world!";
6     }
7 }
```

- 注意类和方法前的注解（`@RestController` 和 `@GetMapping`）。
- 如果注解处报错，让 STS 导入相应的包。

启动项目

- 右键 SpringHelloApplication.java（或项目名），Run As → Spring Boot App。

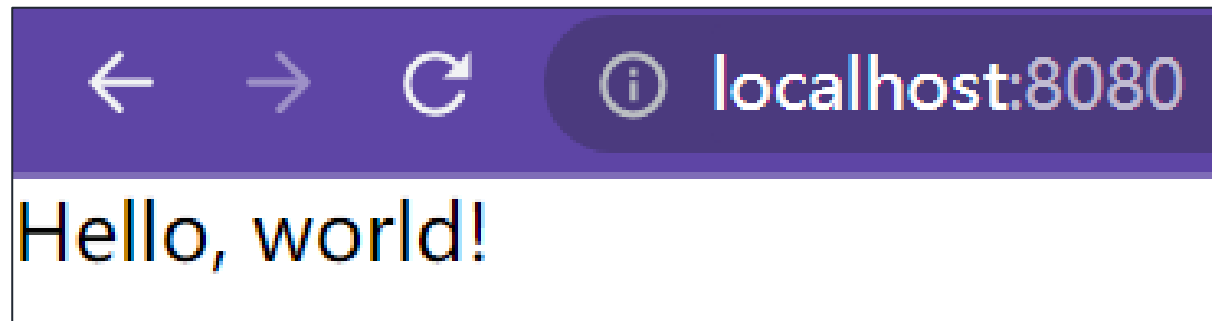


- 等待服务器启动完成。控制台中会出现以下输出：

```
: Started HelloSpringApplication in 1.525 seconds (JVM running for 2.092)
```

查看结果

- 使用浏览器打开以下链接：
<http://localhost:8080/>
- 你应该能看到以下结果：


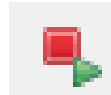


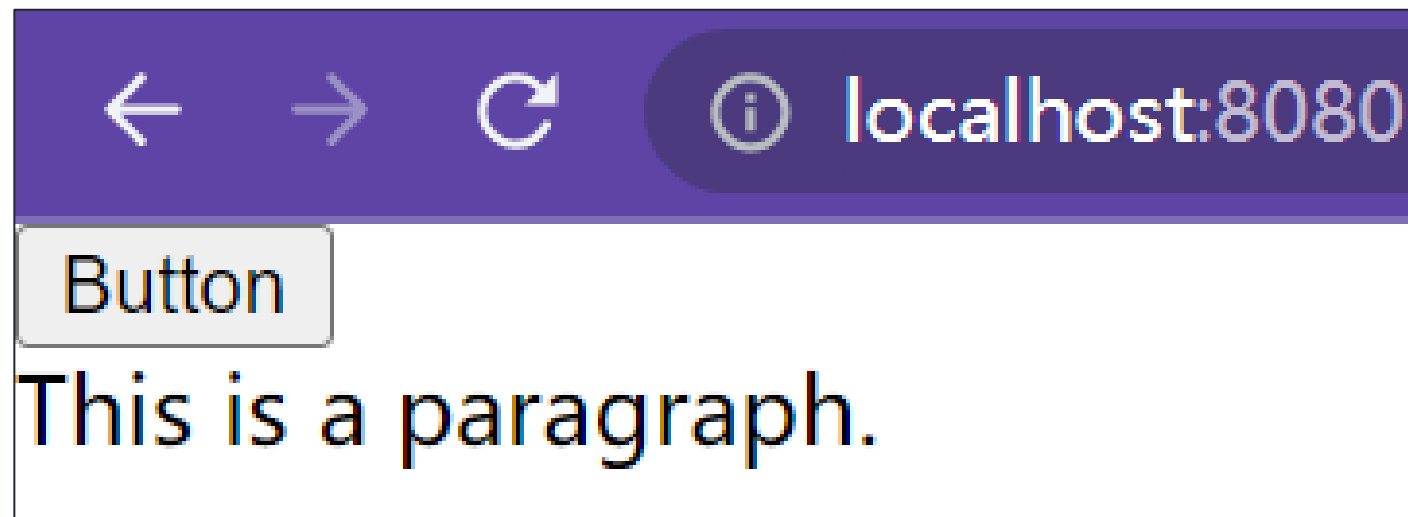
HelloController 代码解读: @RestController

- **@RestController** 注解告诉 Spring, 下面这个类是一个控制器, 描述了一些处理 **HTTP** 请求的方法。
- 当浏览器访问相应的网址 (URL), 它会向对应网址发送一个 **HTTP** 请求。控制器接收到请求后, 会返回一个字符串 (HTML 代码) 给浏览器用于显示。
- 我们定义的 `index()` 方法定义了控制器返回的内容。可以把返回值写成任意 **HTML** 代码, 比如:

```
1 @RestController
2 public class HelloController {
3     @GetMapping("/")
4     public String index() {
5         return "<button>Button</button><p>This is a paragraph.</p>";
6     }
7 }
```

重新启动项目

- 在启动修改后的项目之前，一定要先点击  关闭之前已经启动了的项目。
- 你也可以直接点击  一键关闭之前项目并启动。



HelloController 代码解读: @GetMapping()

- `@GetMapping("/")` 注解告诉 Spring 使用 `index()` 方法来回应用户访问 `http://localhost:8080/` 这个网址的 HTTP 请求动作。
- “Get” 表示该方法应对的请求使用 **GET** 方法。
- 括号里的 “/” 部分则表示用户要通过怎样的网址来访问。它标示出了 “localhost:8080” 之后的部分。

接次页 



- 比如，如果将 “/” 改为 “/hello”，则需要访问 <http://localhost:8080/hello> 才能看到刚才的页面。这样，我们可以编写多个方法响应不同网址的请求：

```
1 @GetMapping("/hello")
2 public String hello() {
3     return "Hello, world!";
4 }
5
6 @GetMapping("/bye")
7 public String bye() {
8     return "Bye, world!";
9 }
```

Tips

方法名无关紧要，
可以任意选择。

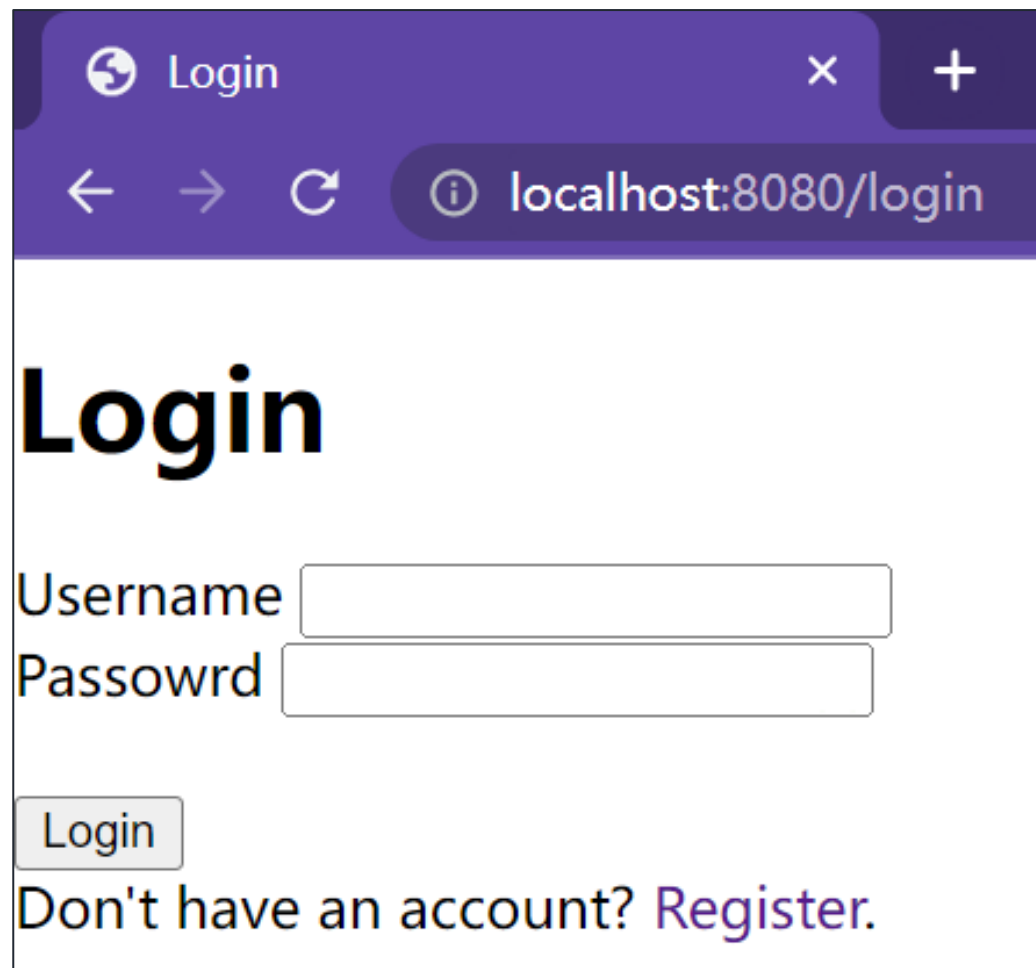
回顾：HTTP 请求方法

- **GET 方法**是默认的、最简单的 HTTP 请求方法。当我们通过在浏览器的地址栏中直接输入网址的方式去访问网页的时候，浏览器采用的就是 GET 方法向服务器获取响应。
- **POST 方法**可以用于向服务器提交大批量的、包含隐私信息的数据。
- 在 Spring 中，使用 **@GetMapping()** 和 **@PostMapping()** 注解分别对应 GET 和 POST 方法的请求。

接次页 

↶ 接前页

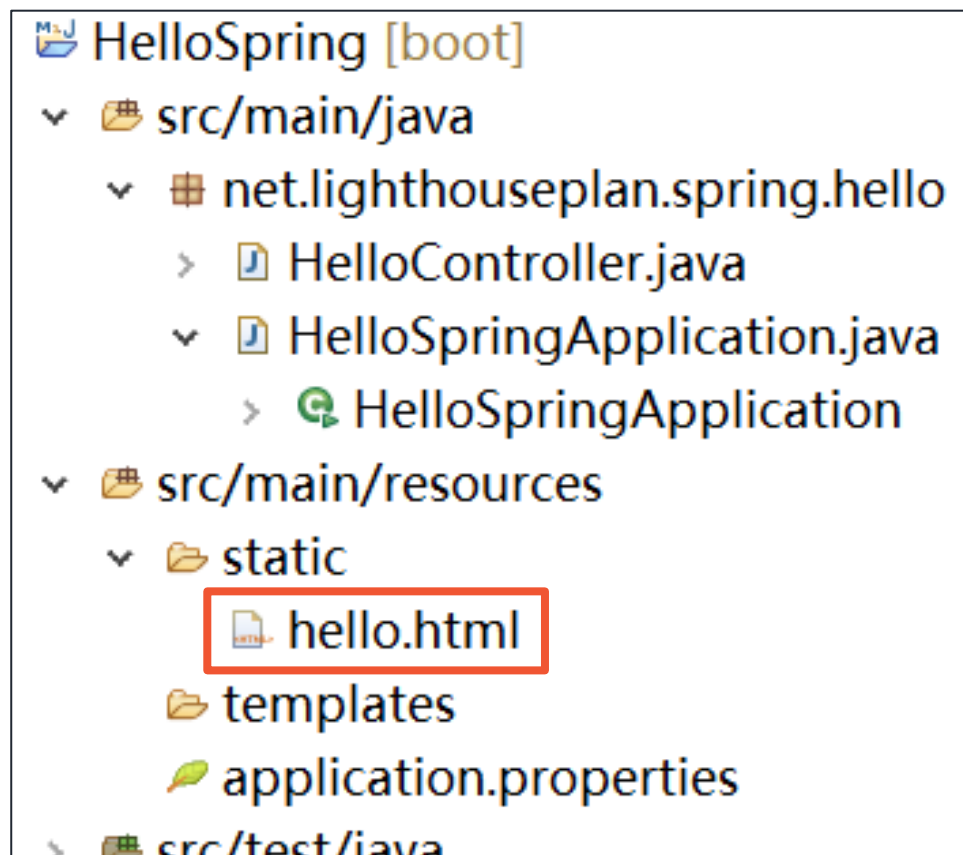
- 思考：登录页面点击 Login 按钮时发送的请求应该用 @GetMapping() 还是 @PostMapping() 进行对应？



The screenshot shows a web browser window with a purple header. The address bar displays 'localhost:8080/login'. The page content includes a large 'Login' heading, two input fields labeled 'Username' and 'Passowrd', a 'Login' button, and a link that says 'Don't have an account? Register.'.

创建 HTML 页面

- 目前为止我们只能向浏览器返回字符串。想要用这种方式返回一个完整的网页非常麻烦。我们希望能够直接返回一个写好的 **HTML 文件**。
- 首先，让我们在 **static** 目录中创建想被返回的 HTML 页面：



hello.html:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Hello Spring</title>
6 </head>
7
8 <body>
9   <h1>Hello, world!</h1>
10 </body>
11 </html>
```

在控制器中返回页面

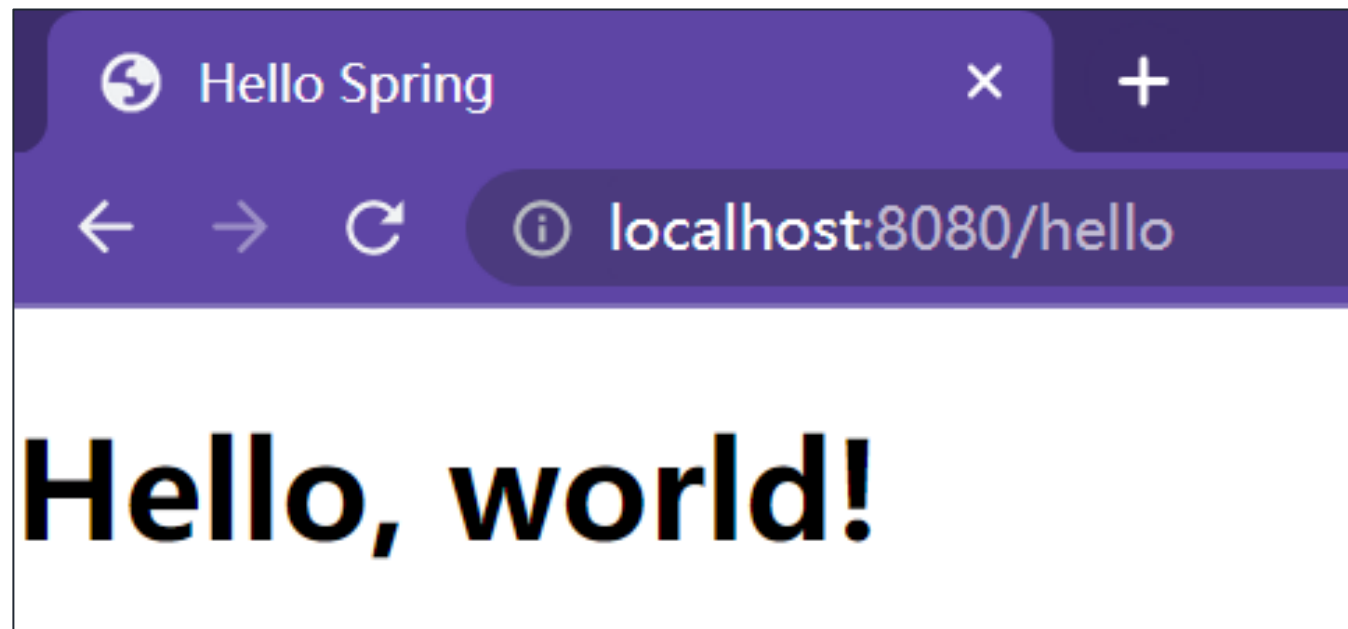
- 接下来，我们需要修改控制器中的代码：

```
1 @Controller
2 public class HelloController {
3     @GetMapping("/hello")
4     public String hello() {
5         return "hello.html";
6     }
7 }
```

- 其中：
 - **@Controller** 注解表示这是一个返回 HTML 页面的控制器。
 - hello() 方法返回的是接受到请求时响应的 **HTML 文件的路径**。

查看结果

- 重启项目，访问 <http://localhost:8080/hello>，可以看到对应页面被加载出来：

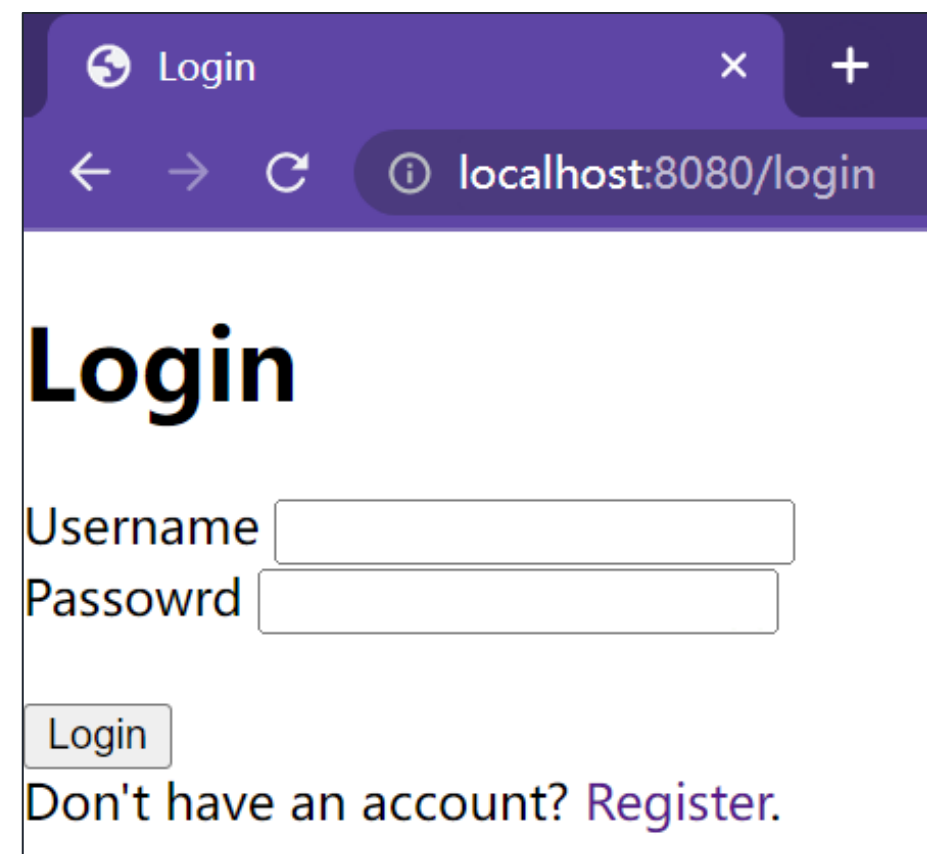


Q & A

Question and answer

练习：创建登录界面

- 目标：创建一个登录页面。
- 创建一个如下的登录页面，并保存为 login.html 文件。
- 编写控制器中的方法，使访问 `http://localhost:8080/login` 时会显示该页面。
- 先自己动手试一试！



Login

← → ↻ ⓘ localhost:8080/login

Login

Username

Passowrd

Login

Don't have an account? [Register.](#)

发送登录表单

- 目前的登录界面没有实际的功能。我们希望点击 Login 按钮后，服务器可以根据我们输入的数据进行响应。
- 为此，首先我们需要在 HTML 代码中定义表单发送的 HTTP 请求的路径和方法：

```
<form action="/login" method="post">
```

- 回顾一下，在 `<form>` 标签中，**action** 属性定义了请求发送的目标网址，而 **method** 属性定义了其方法。
- 在这个例子中，点击 Login 按钮会向 `http://localhost:8080/login` 发送一个 POST 请求。

登录表单中的参数

- 为了使登录表单中用户填入的数据被传送至服务器，我们需要设置 `<input>` 标签的 **name** 属性：

```
1 <div>
2   <label for="username">Username</label>
3   <input id="username" name="username" type="text"></input>
4 </div>
5
6 <div>
7   <label for="pwd">Passowrd</label>
8   <input id="pwd" name="password" type="password"></input>
9 </div>
```

处理表单请求的控制器方法

- 接下来，在控制器中，我们需要编写以下方法响应这个请求：

```
@PostMapping("/login")  
public String login(@RequestParam String username, @RequestParam String password) {
```

- 这里，**@PostMapping("/login")** 注解说明该方法处理发送到 `http://localhost:8080/login` 的 POST 请求。（和 HTML 发送的请求对应。）
- **@RequestParam** 注解说明这个请求会有两个参数：username 和 password。（和 HTML 发送的参数对应。）

Note



Java 中的参数名对应的是 **name** 属性而不是 **id** 属性！

编写方法内容

- 我们希望用户点击登录按钮后访问 hello 页面。由于我们已经有了一个返回 hello 页面的网址 `http://localhost:8080/hello`，只要让浏览器访问这个网址就可以了。
- 这被称为网址的**重定向**`[リダイレクト]`。在 Spring 中，我们只要让方法返回如下字符串：

```
1 @PostMapping("/login")
2 public String login(@RequestParam String username, @RequestParam String password) {
3     return "redirect:/hello";
4 }
```

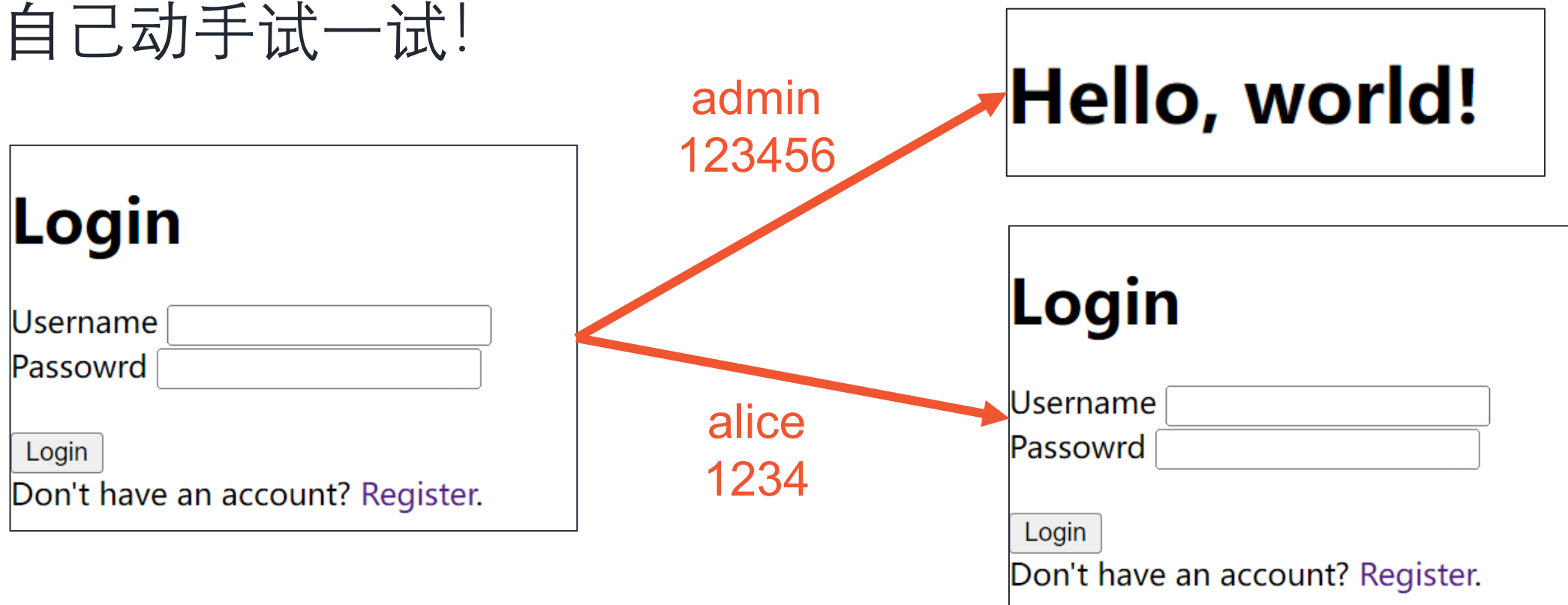
- 这样，用户在点击 Login 按钮后就会被自动导向至 `http://localhost:8080/hello`，最终打开 hello 页面。

Q & A

Question and answer

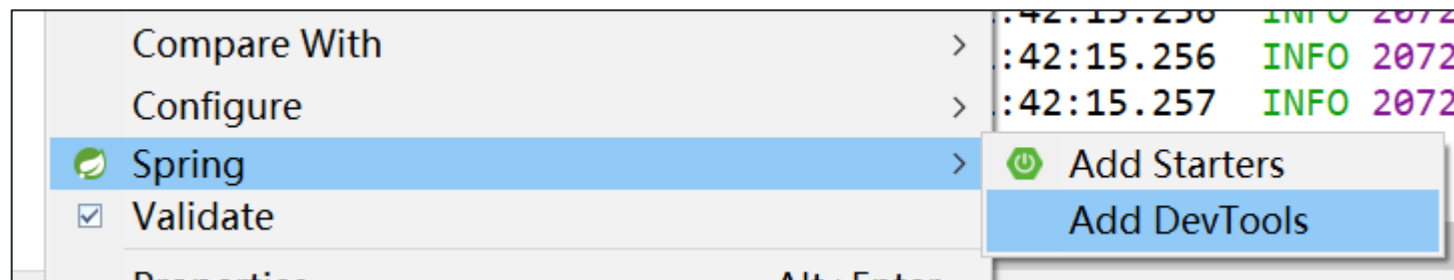
练习：验证输入

- 最后，我们希望服务器能验证用户的输入。
- 如果用户名为 admin 并且密码为 123456，则跳转到 hello 页面；如果用户名或密码错误，则跳转到 login 页面（让用户重新输入）。
- 先自己动手试一试！



DevTools

- 注意到我们每次修改代码后都需要手动重启服务器，可以使用 **DevTools** 插件简化这一流程。
- 右键项目 → Spring → Add DevTools:



- 安装该插件后，每次你保存代码，Spring 都会自动重启服务器。有选择地使用，这可能会导致你的电脑卡顿。

目录

1

创建项目

2

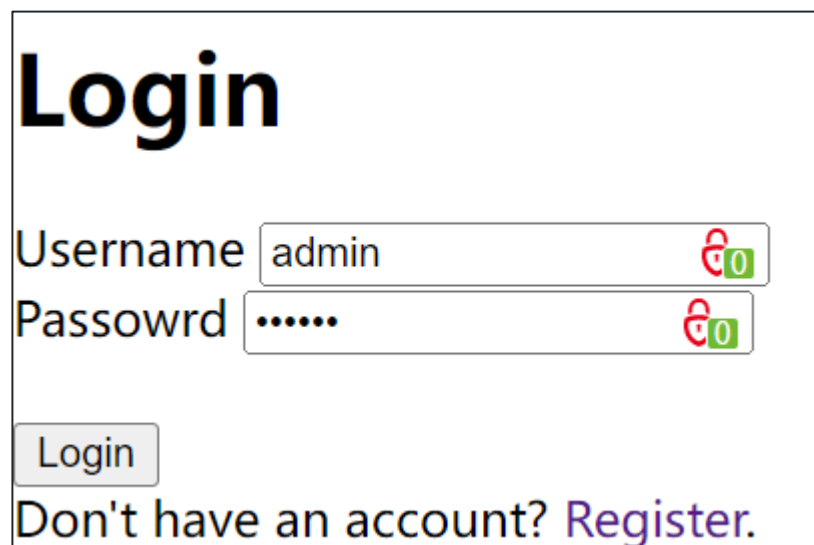
创建基本网站

3

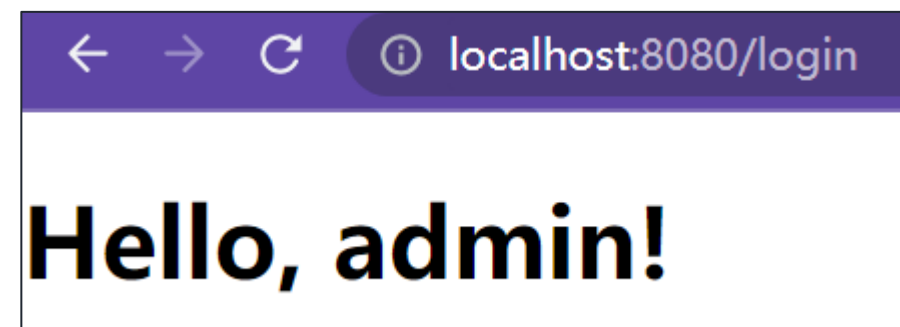
Thymeleaf

动态网页

- 目前为止，我们只能向浏览器返回不变的静态网页。我们希望服务器能够根据数据（比如用户输入的表单数据、数据库中的数据等）**动态地生成网页**。比如，在 hello 页面中，显示“Hello, [用户名]!”：



A login form titled "Login". It contains two input fields: "Username" with the value "admin" and "Passowrd" (misspelled) with masked characters ".....". Both fields have a red padlock icon and a green "0" in a box to their right. Below the fields is a "Login" button. At the bottom, it says "Don't have an account? [Register.](#)"



- 这就是之前 (👉 § 4.1.1) 提到的**动态网页**[\[動的ページ\]](#)的概念。

模板引擎

- **模板引擎**[テンプレートエンジン]可以被用于生成这样的动态网页。
- 使用模板引擎时，我们可以使用一种特殊的模板语言（大多是 HTML 语言的变种）来编写网页。它可以接受控制器方法给它的一些数据，并使用这些数据渲染[レンダリング]出最终的 HTML 页面。
- 当然，（就像我们在学习 Servlet 时做的一样，）我们也能直接使用 Java 代码生成页面。然而，使用模板引擎，我们可以减少 Java 代码中对 HTML 代码的描述，把 HTML 代码和 Java 代码分离开来，提高代码的可读性，从而提高开发效率。

Thymeleaf

- 传统 Java 服务器开发中最常用的模板引擎包括 *JSP* 等。但 Spring Boot 建议我们使用 **Thymeleaf** 这一轻量开源引擎。
- JSP 最大的问题在于它使用一些**特殊的 HTML 标签**来显示服务器传来的数据（比如 “<%@ variable%>”）。这导致我们无法在不启动服务器时使用浏览器预览这些网页的显示效果。
- 与之相对，Thymeleaf 使用的模板语言完全基于原版 HTML 标签。为了显示数据，显示数据靠的是在现有的 HTML 标签上添加一些**特殊的属性**。这就保证了即使你没有启动服务器，这些网页也能正常被预览。



Thymeleaf

添加 Thymeleaf 依赖

- 要使用 Thymeleaf, 我们需要在创建项目时选择使用 Thymeleaf 依赖:

New Spring Starter Project Dependencies

Spring Boot Version: 2.7.2

Frequently Used:

☒ Spring Web ☒ Thymeleaf

Available:

Type to search dependencies

- Spring Cloud Tools
- ▾ Template Engines
 - ☒ Thymeleaf
 - ☐ Apache Freemarker
 - ☐ Mustache
 - ☐ Groovy Templates
- Testing
- VMware Tanzu Application Service
- ▾ Web
 - ☒ Spring Web
 - ☐ Spring Reactive Web

Selected:

- X Thymeleaf
- X Spring Web

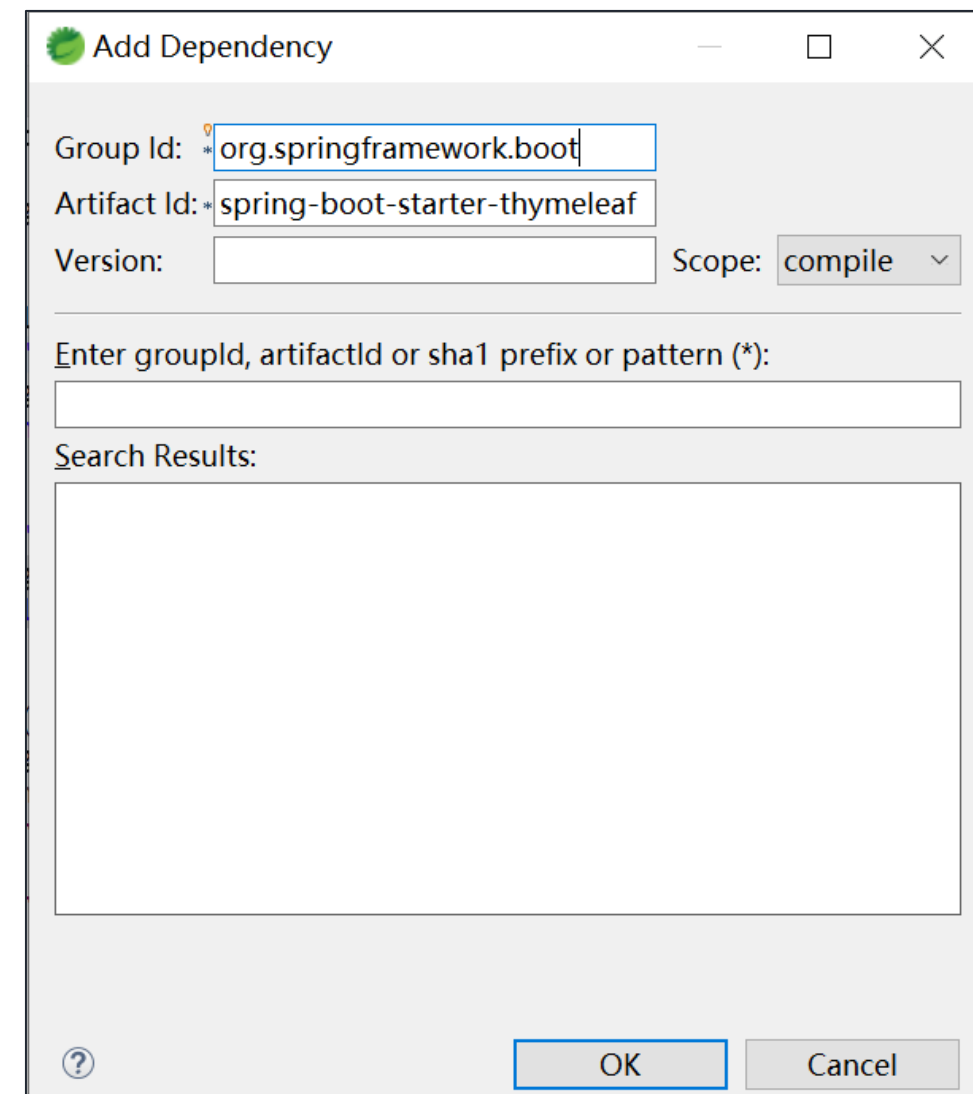
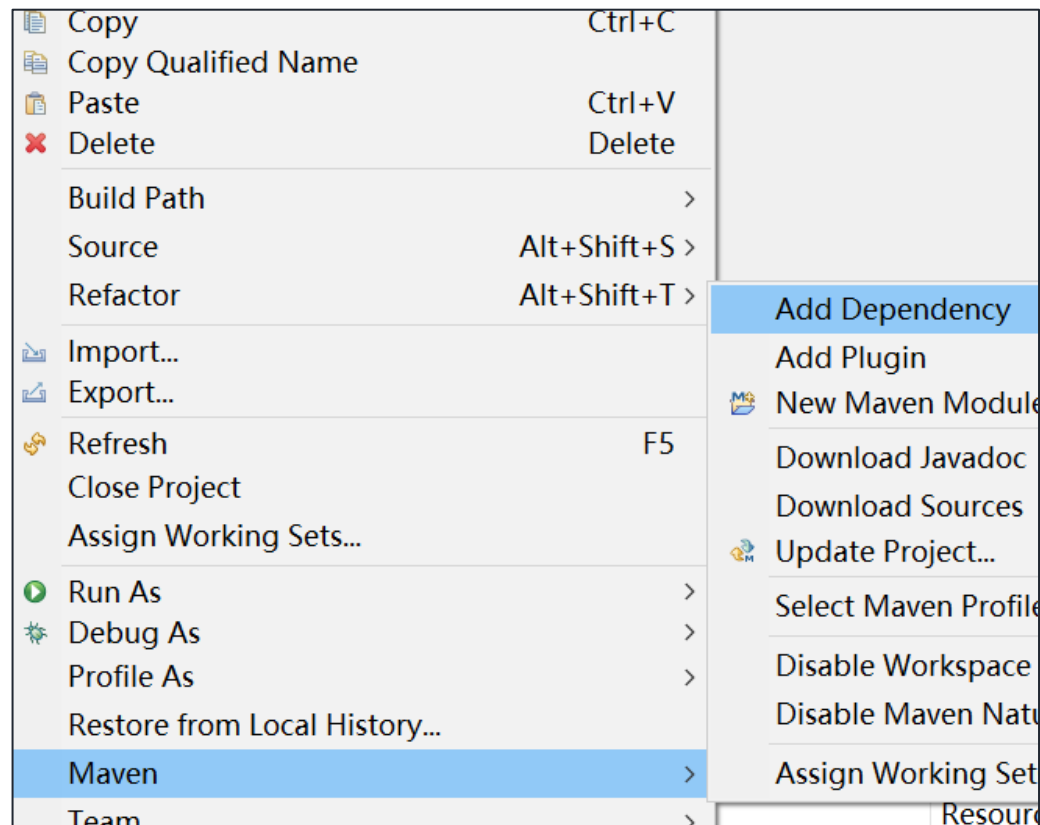
Make Default Clear Selection

? < Back Next > Finish Cancel

接次页 ➞



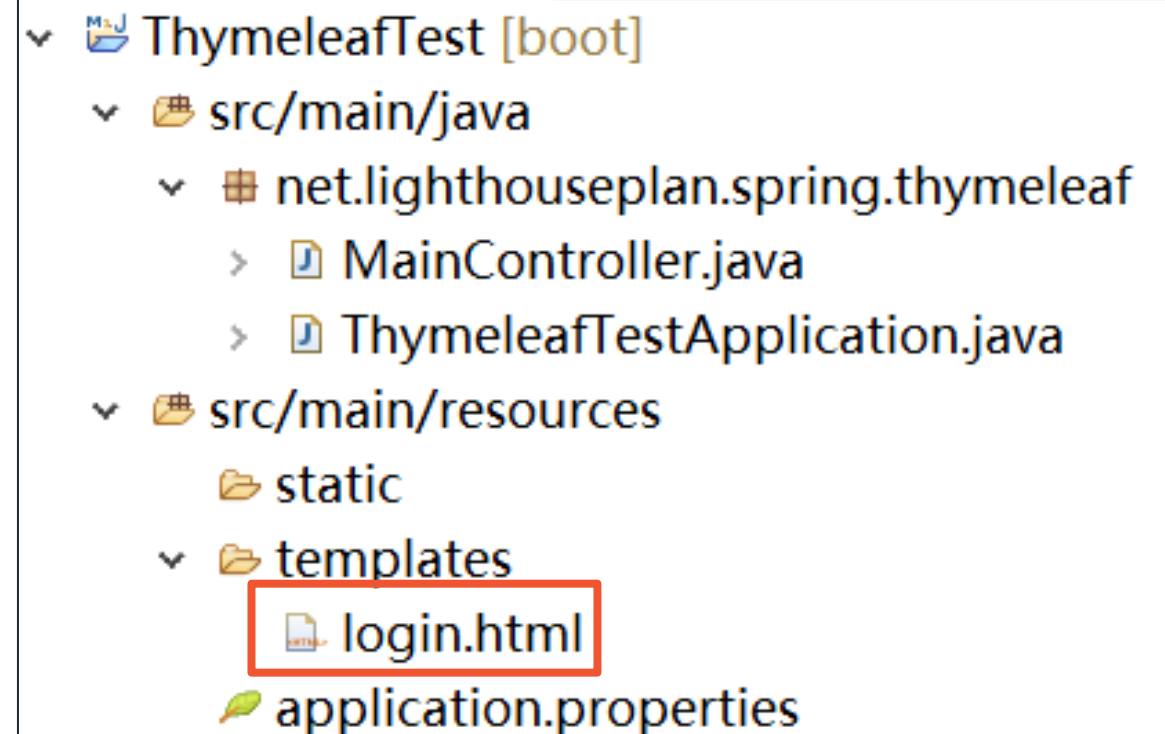
- 如果要在创建项目后添加依赖，右键项目 → Maven → Add Dependency，并输入 Thymeleaf 的 groupId 和 artifactId:



添加控制器和模板网页

- 首先，让我们添加一个静态的登录页面。控制器的书写方式和之前一样，但是要把 HTML 文件放在 **templates** 文件夹下：

```
1 @Controller
2 public class MainController {
3     @GetMapping("/login")
4     public String getLoginPage() {
5         return "login.html";
6     }
7 }
```



向 Thymeleaf 页面传递信息

- 接下来，我们为用户登录时的 POST 请求编写控制器。在登录成功，访问页面时，我们想把用户的一些信息传递给 Thymeleaf 模板。
- 在编写控制器时，有 2 种方式实现这一效果：
 - 使用 Model 对象。
 - 使用 ModelAndView 对象。
- 这两种方式是完全等价的。我们自己编写时选择任意一种即可。

使用 Model 对象

- 第一种方式是使用 **Model** 对象。
- 为原本的方法添加一个新的，Model 类型的参数，并使用其 **addAttribute()** 方法添加一些数据：

```
1 @PostMapping("/login")
2 public String login(@RequestParam String username,
3                     @RequestParam String password, Model model) {
4     model.addAttribute("name", username);
5     return "hello.html";
6 }
```


使用 ModelAndView 对象

- 第二种方式是使用 **ModelAndView** 对象。
- 为方法添加一个 ModelAndView 类型的参数，通过其 **addObject()** 方法添加一些数据，并通过其 **setViewName()** 设置其 HTML 模板文件。最终，返回该 ModelAndView 对象：

```
1 @PostMapping("/login")
2 public ModelAndView login(@RequestParam String username,
3     @RequestParam String password, ModelAndView mav) {
4     mav.addObject("name", username);
5     mav.setViewName("hello.html");
6     return mav;
7 }
```

创建模板页面

- 在 templates 文件夹下创建和之前相同的 hello.html 文件。
- 我们该如何获取控制器传来的数据呢？在 Thymeleaf 中，为标签添加一个特殊的属性 **th:text**，可以使标签的内容变为指定数据的值：

```
<h1 th:text="${name}"></h1>
```

- 这个 h1 标签中的值便会变为控制器传来的 “name” 的值。“**\${}**” 中的内容将作为一个 **OGNL** 表达式被解读。简单来说，就是类似 Java 代码的表达式。


接次页 



- 修改 hello.html 中的标签，查看运行结果：

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Hello Spring</title>
6 </head>
7
8 <body>
9   <h1>Hello, <span th:text="${name}"></span>!</h1>
10 </body>
11 </html>
```

Thymeleaf 的功能

- 除了修改标签内容这样基本的功能以外，Thymeleaf 还提供了其它丰富的模板功能，包括：
 - 设置 HTML 属性的值；
 - 使用基本的运算符、Java 方法；
 - 显示列表、对象的数据；
 - 条件判断；
 - 自动补全 URL 等。
- 更多方法可以查看官方文档：
 <https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html>

设置 HTML 属性的值

- 除了设置 text 属性外，你还可以设置其它 **HTML 属性**，只需在原本的属性名前加上“**th:**”的前缀即可。比如，以下代码使用 type 数据设置标签的 class 属性：

```
<h1 th:class="${type}">Hello</h1>
```

内联文本

- 除了使用 `th:text` 属性外，你还可以直接在“**[[]]**”符号里书写内联文本：

```
<h1>Hello, [[${name}]]!</h1>
```

- 以上代码的 `[[${name}]]` 部分会被渲染为 `name` 的值。这种写法在你只想修改大段文本中的某些部分时很有用。

运算符的使用

- 一些 Java 中的基本运算符都可以直接在 Thymeleaf 中使用，比如算术运算符：

```
<p th:text="${num1 * num2 / 3}"></p>
```

- 或三元运算符：

```
<p th:text="${num1 > num2 ? 0 : 1}"></p>
```

- 但需注意几个不同点：
 - 字符串使用单引号表达：

```
<p th:text="${'num1 = ' + num1 + '.'}"></p>
```

- 逻辑运算符使用 **and**、**or** 而不是 **&&**、**||**。

条件判断

- 标签的 **th:if** 属性将评价表达式的布尔值，只有当表达式为真时才显示该标签。
- 比如，以下标签只有在传入的数据 num 大于 60 时才显示：

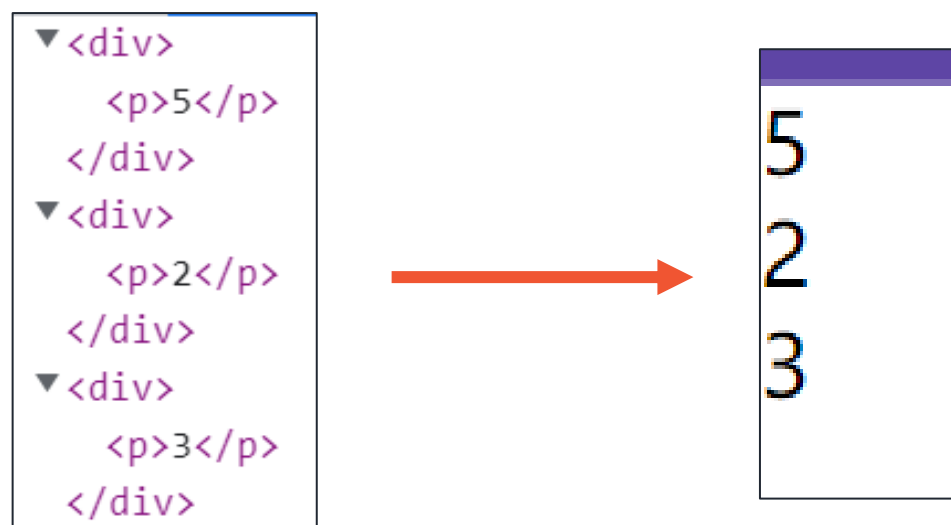
```
<p th:if="{num > 60}">num > 60!</p>
```


遍历列表

- 对于可以被遍历的数据，比如数组、列表、集合等，可以使用 **th:each** 属性进行遍历：

```
1 <div th:each="num : ${nums}">
2   <p th:text="${num}"></p>
3 </div>
```

- 这里，nums 是传入的列表数据。Thymeleaf 会自动为列表中的 **每一个数据 num** 生成一个 <div>。<div> 中的其它标签可以使用 num 的值。比如，nums 是列表 [5, 2, 3] 时，显示的结果为：



对象

- 如果传入的参数是一些对象，你可以像 Java 一样使用 “.” 运算符获得其属性或使用其方法：

```
<p th:text="${student.name}"></p>  
<p th:text="${student.sayHello( )}"></p>
```

对象表达式

- 然而，这在我们需要大量使用同一个对象的属性时很重复啰嗦：

```
1 <div>
2   <p th:text="'Name: ' + ${studentInformation.name}"></p>
3   <p th:text="'Age: ' + ${studentInformation.age}"></p>
4   <p th:text="'Score: ' + ${studentInformation.score}"></p>
5 </div>
```

- 我们可以使用 **th:object** 属性和对象表达式 “*{}” 来简洁地表达：

```
1 <div th:object="${studentInformation}">
2   <p th:text="'Name: ' + *{name}"></p>
3   <p th:text="'Age: ' + *{age}"></p>
4   <p th:text="'Score: ' + *{score}"></p>
5 </div>
```

URL 计算

- 由于使用模板生成，服务器上文件的相对路径（URL），比如 `<link>` 或 `<a>` 的 `href` 属性、`<script>` 的 `src` 属性等，计算起来十分麻烦。
- 使用 “@{}” 符号括起一个相对 URL，Thymeleaf 会根据 HTML 文件的位置计算出服务器上的正确 URL。
- 比如，以下代码将导入 `static/css/main.css` 文件：

```
<link rel="stylesheet" th:href="@{/css/main.css}">
```

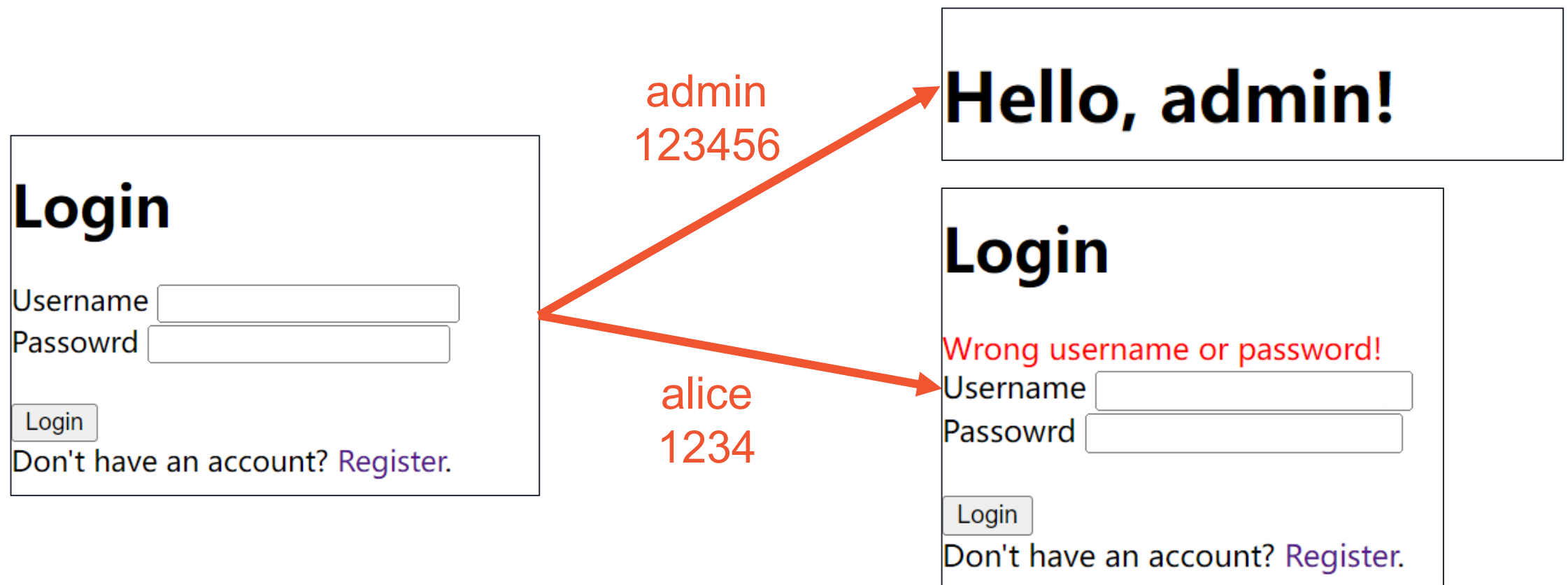
- 这样，我们的其它静态文件（CSS、JavaScript、图片等）就可以整理在 `static` 文件夹下了。

Q & A

Question and answer

练习：提示错误

- 我们希望当用户输入错误的用户名密码时，不仅回到 login 页面，还显示额外的错误提示信息。
- 不创建新的页面，如何使用 Thymeleaf 的功能实现这一效果？
- 先自己动手试一试！



总结

Sum Up

1. 创建 Spring Boot 项目以及添加 Maven 依赖的方法。
2. 在 Spring 项目中，控制器的书写方法：
@Controller、@RequestMapping、@RequestParam
等注解的含义。
3. 在 Spring 项目中，生成动态网页的方法：
Thymeleaf 模板语言的基本语法。

THANK YOU!