# SpringBoot与web开发

## 创建过程

1. 创建SpringBoot应用，根据业务场景选中需要的模块，比如web,sql等。
2. SpringBoot已经将这些场景配置好了，只需要在配置文件中，指定少量配置就可以运行起来。
3. 自己编写业务代码。

关键核心在于弄清楚**自动配置原理：**

* **这个场景SpringBoot帮我们配置了什么，能不能修改，能修改哪些配置，能不能扩展？**
* **基本上都是\*\*\*AutoConfiguration：帮我们给容器中自动配置组件**
* **\*\*\*Properties：配置类来封装配置文件内容**

## SpringBoot对静态资源的映射规则

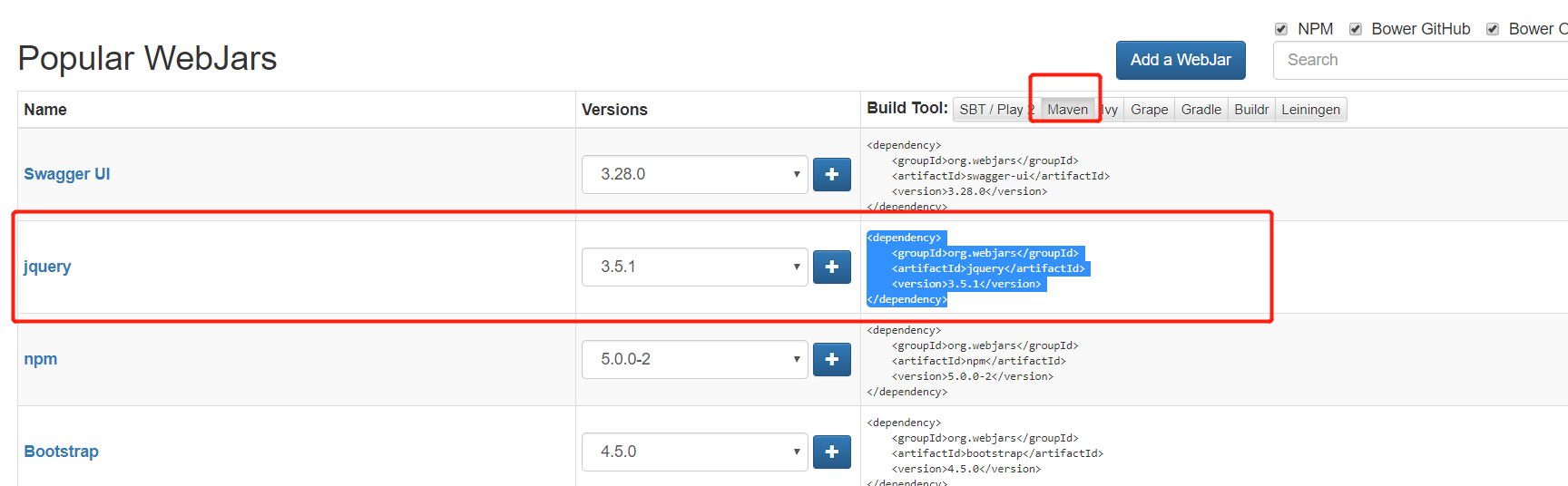
在WebMvcAutoConfiguration下的一个方法描述了映射规则：

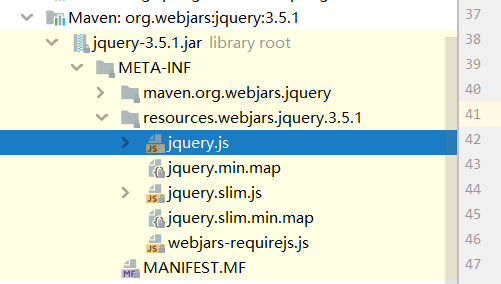
|  |
| --- |
| **public void** addResourceHandlers(ResourceHandlerRegistry registry) {  **if** (!**this**.resourceProperties.isAddMappings()) {  logger.debug(**"Default resource handling disabled"**);  } **else** {  Duration cachePeriod = **this**.resourceProperties.getCache().getPeriod();  CacheControl cacheControl = **this**.resourceProperties.getCache().getCachecontrol().toHttpCacheControl();  **if** (!registry.hasMappingForPattern(**"/webjars/\*\*"**)) {  **this**.customizeResourceHandlerRegistration(registry.addResourceHandler(**new** String[]{**"/webjars/\*\*"**}).addResourceLocations(**new** String[]{**"classpath:/META-INF/resources/webjars/"**}).setCachePeriod(**this**.getSeconds(cachePeriod)).setCacheControl(cacheControl));  }  String staticPathPattern = **this**.mvcProperties.getStaticPathPattern();  **if** (!registry.hasMappingForPattern(staticPathPattern)) {  **this**.customizeResourceHandlerRegistration(registry.addResourceHandler(**new** String[]{staticPathPattern}).addResourceLocations(WebMvcAutoConfiguration.getResourceLocations(**this**.resourceProperties.getStaticLocations())).setCachePeriod(**this**.getSeconds(cachePeriod)).setCacheControl(cacheControl));  }  } } |

1. 所有/webjars/\*\*，都去**classpath:/META-INF/resources/webjars/找资源。**

Webjars:以jar包的方式引入静态资源。Webjars网站：<https://www.webjars.org/>

例如引入jquery



如果访问jquery下的jquery.js资源，

则访问路径：

localhost:8080/webjars/jquery/3.5.1/jquery.js

1. “/\*\*”访问当前项目的任何资源

**"classpath:/META-INF/resources/"**,

**"classpath:/resources/"**,

**"classpath:/static/"**,

**"classpath:/public/",**

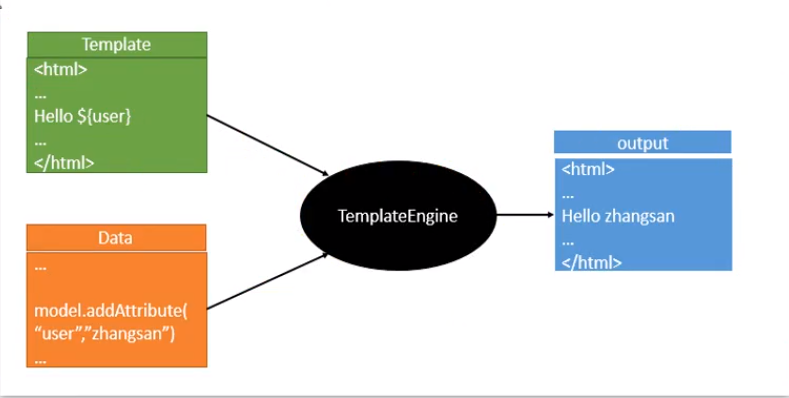
**"/"：当前项目的根路径**

1. 自定义静态文件夹下的路径

**spring.resources.static-locations**=**classpath:/hello/,classpath:/pcq/**

## 模板引擎

市面上有JSP、Velocity、Freemarker、Thymeleaf等。



页面以及数据交给模板引擎，组装好返回给浏览器。

### Thymeleaf的使用

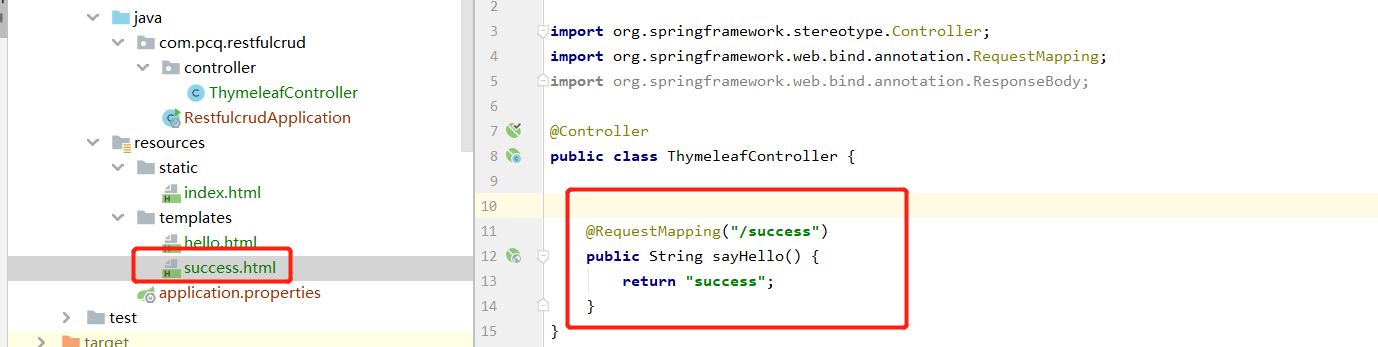
#### 引入thymeleaf

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

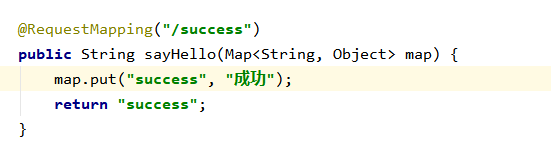
版本可以在properties的标签中定义。

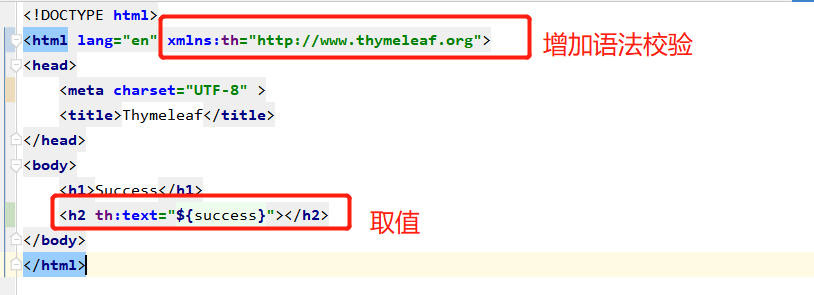
#### 使用方式

只要把html页面放在类路径templates下，thymeleaf就能自动渲染。



#### 将数据动态加载到页面中

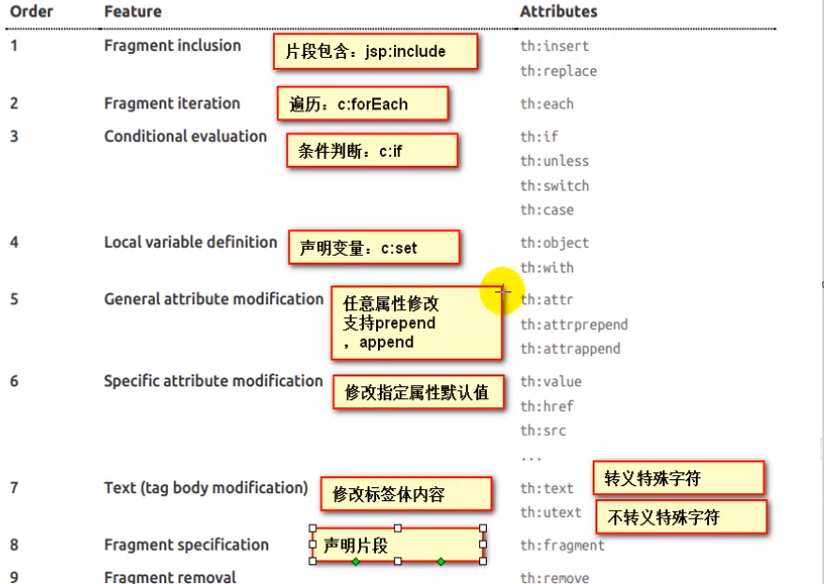




### 语法规则

#### th:text 改变当前元素中的文本内容

th:任意html属性，来替换原生属性的值。例如th:id=”${key}”、 th:class=”${key}”



#### 表达式

详细参考官方操作手册：<https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.pdf>

|  |
| --- |
| Simple expressions:  Variable Expressions: ${...} 获取变量值：OGNL   1. 获取对象属性、调用方法 2. 使用内置的基本对象 3. 内置的工具对象   Selection Variable Expressions: \*{...} 选择表达式，和${}一样  补充：配合th:object来使用。  <div th:object="${session.user}">  <p>Name: <span th:text="\*{firstName}">Sebastian</span>.</p>  <p>Surname: <span th:text="\*{lastName}">Pepper</span>.</p>  <p>Nationality: <span th:text="\*{nationality}">Saturn</span>.</p>  </div>  Message Expressions: #{...} 获取国际化内容  Link URL Expressions: @{...} 定义URL连接  <a href="details.html" th:href="@{/order/details(orderId=${o.id})}">view</a>  Fragment Expressions: ~{...} 片段表达式  Literals:字面量  Text literals: 'one text' , 'Another one!' ,…  Number literals: 0 , 34 , 3.0 , 12.3 ,…  Boolean literals: true , false  Null literal: null  Literal tokens: one , sometext , main ,…  Text operations: 文本操作  String concatenation: +  Literal substitutions: |The name is ${name}|  Arithmetic operations: 数学运算  Binary operators: + , - , \* , / , %  Minus sign (unary operator): -  Boolean operations: 布尔运算  Binary operators: and , or  Boolean negation (unary operator): ! , not  Comparisons and equality: 比较运算  Comparators: > , < , >= , <= ( gt , lt , ge , le )  Equality operators: == , != ( eq , ne )  Conditional operators: 条件运算  If-then: (if) ? (then)  If-then-else: (if) ? (then) : (else)  Default: (value) ?: (defaultvalue)  Special tokens:  No-Operation: \_ |

示例：

|  |
| --- |
| @RequestMapping(**"/success"**) **public** String sayHello(Map<String, Object> map) {  map.put(**"success"**, **"成功"**);  map.put(**"list"**, Arrays.*asList*(**"张三"**, **"李四"**, **"王五"**));  **return "success"**; }  <!DOCTYPE **html**> <**html lang="en" xmlns:th="http://www.thymeleaf.org"**> <**head**>  <**meta charset="UTF-8"** >  <**title**>Thymeleaf</**title**> </**head**> <**body**>  <**h1**>Success</**h1**>  <**h2 th:text="${success}"**></**h2**>  <**h3 th:text="${user}" th:each="user:${list}"**></**h3**> <**hr**/>  <**h3**>  <**span th:each="user:${list}"**>[[${user}]]</**span**>  </**h3**> </**body**> </**html**> |

## SpringMVC自动配置原理

### 1.自动配置原理

如何修改SpringBoot的默认配置

模式：

1. **SpringBoot在自动配置很多组件时，先看容器中有没有用户自己配置的@Bean,@Component。如果有使用用户的，否则自动配置**。有些组件可以有多个，例如ViewResolver，用户配置的和SpringBoot的组合起来。

### 2.扩展SpringMVC

编写一个配置类，使用@Configuration注解：

|  |
| --- |
| @Configuration **public class** MyMvcConfig **extends** WebMvcConfigurationSupport {    @Override  **public void** addViewControllers(ViewControllerRegistry registry) {    registry.addViewController(**"/test"**).setViewName(**"success"**);  } } |

既保留了所有的自动配置，也能扩展自己的配置。

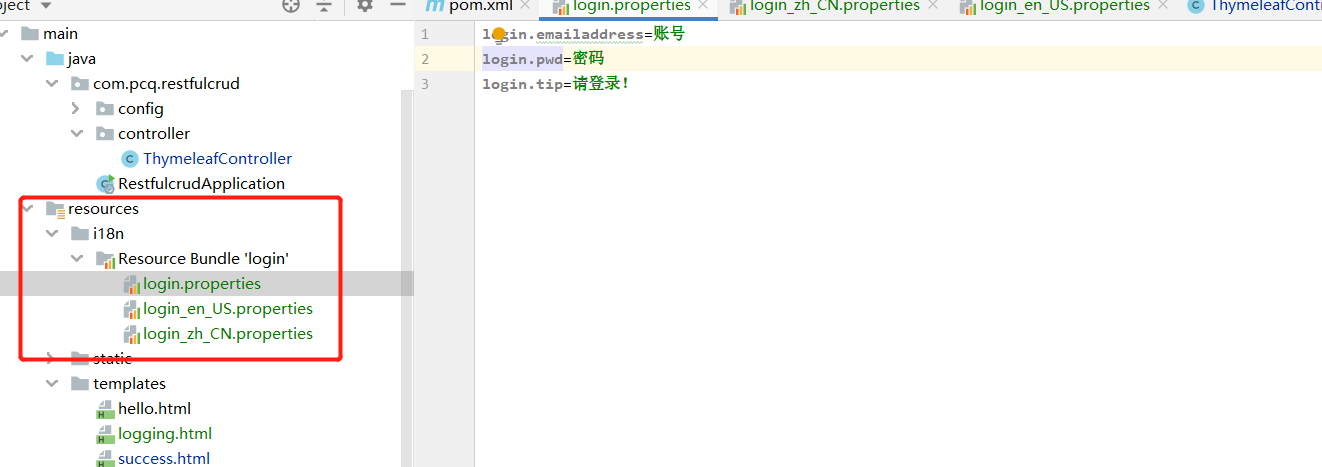
如果想要全面接管SpringBoot SpringMvc的自动配置，则添加注解：@EnableWebMvc。

### 国际化

1. 编写国际化配置文件
2. 使用ResourceBundleMessageSource管理国际化资源文件
3. 在页面使用fmt:message取出国际化内容

步骤：

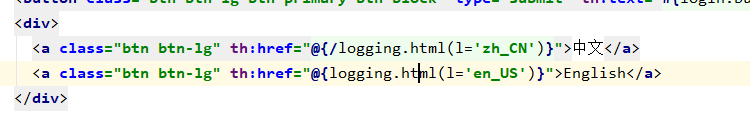
* 编写国际化配置文件 ，抽取页面需要的国际化信息



* SpringBoot 自动配置好了管理国际化资源文件的组件
* 在页面获取国际化的值，页面会根据浏览器的设置来确定何种语言



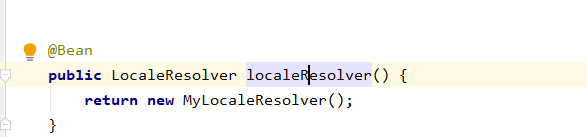
也可以通过点击按钮的方式来切换中英文



先写一个LocaleResolver解析器

|  |
| --- |
| **public class** MyLocaleResolver **implements** LocaleResolver {  @Override  **public** Locale resolveLocale(HttpServletRequest request) {  String l = request.getParameter(**"l"**);  Locale locale = Locale.*getDefault*();*//开始是默认的* **if**(!StringUtils.*isEmpty*(l)) {  String[] params = l.split(**"\_"**);  locale = **new** Locale(params[0], params[1]);  }  **return** locale;  }   @Override  **public void** setLocale(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Locale locale) {   } } |

之后注册到SprigBoot容器中，作为组件使用



### 登陆以及拦截器使用