# Spring Boot 整合 Thymeleaf

虽然现在慢慢在流行前后端分离开发，但是据松哥所了解到的，还是有一些公司在做前后端不分的开发，而在前后端不分的开发中，我们就会需要后端页面模板（实际上，即使前后端分离，也会在一些场景下需要使用页面模板，例如邮件发送模板）。

早期的 Spring Boot 中还支持使用 Velocity 作为页面模板，现在的 Spring Boot 中已经不支持 Velocity 了，页面模板主要支持 Thymeleaf 和 Freemarker ，当然，作为 Java 最最基本的页面模板 Jsp ，Spring Boot 也是支持的，只是使用比较麻烦。

松哥打算用三篇文章分别向大家介绍一下这三种页面模板技术。

今天我们主要来看看 Thymeleaf 在 Spring Boot 中的整合！

## Thymeleaf 简介

Thymeleaf 是新一代 Java 模板引擎，它类似于 Velocity、FreeMarker 等传统 Java 模板引擎，但是与传统 Java 模板引擎不同的是，Thymeleaf 支持 HTML 原型。

它既可以让前端工程师在浏览器中直接打开查看样式，也可以让后端工程师结合真实数据查看显示效果，同时，SpringBoot 提供了 Thymeleaf 自动化配置解决方案，因此在 SpringBoot 中使用 Thymeleaf 非常方便。

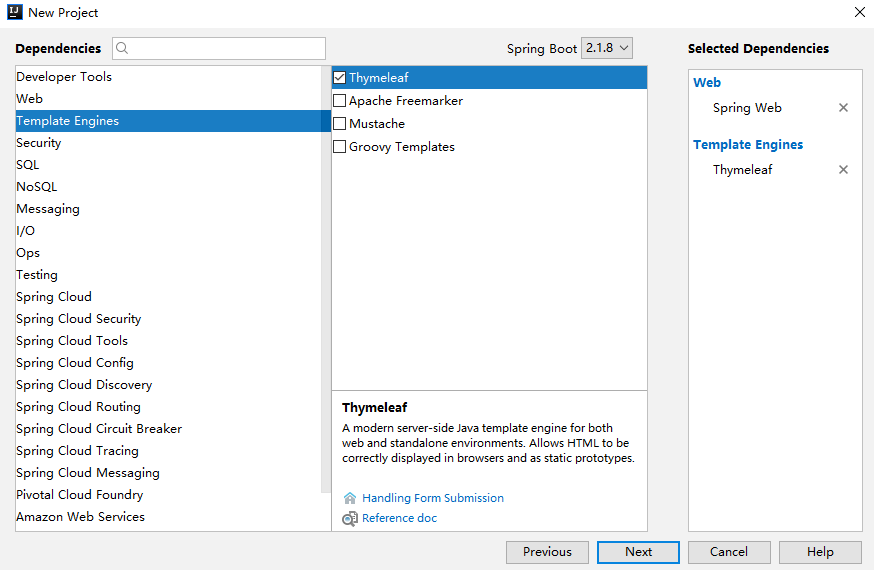
事实上， Thymeleaf 除了展示基本的 HTML ，进行页面渲染之外，也可以作为一个 HTML 片段进行渲染，例如我们在做邮件发送时，可以使用 Thymeleaf 作为邮件发送模板。

另外，由于 Thymeleaf 模板后缀为 .html，可以直接被浏览器打开，因此，预览时非常方便。

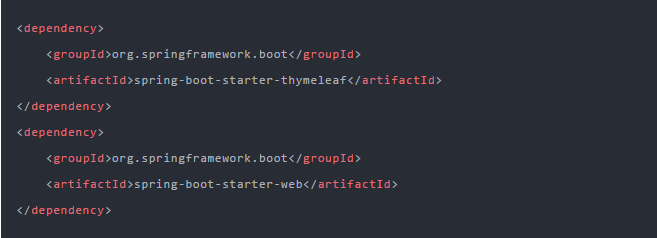
## 整合

* 创建项目

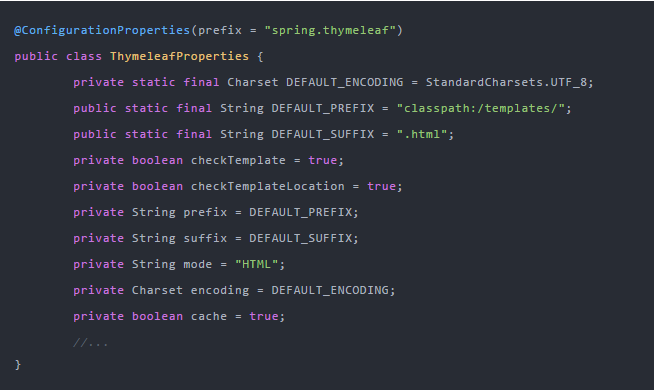
Spring Boot 中整合 Thymeleaf 非常容易，只需要创建项目时添加 Thymeleaf 即可：



创建完成后，pom.xml 依赖如下：

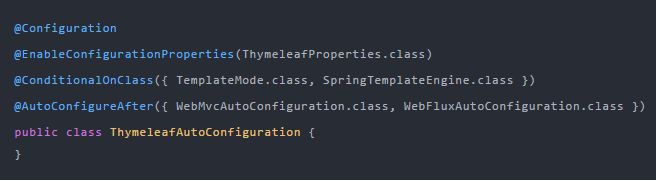


当然，Thymeleaf 不仅仅能在 Spring Boot 中使用，也可以使用在其他地方，只不过 Spring Boot 针对 Thymeleaf 提供了一整套的自动化配置方案，这一套配置类的属性在org.springframework.boot.autoconfigure.thymeleaf.ThymeleafProperties 中，部分源码如下：



1. 首先通过 @ConfigurationProperties 注解，将 application.properties 前缀为 spring.thymeleaf 的配置和这个类中的属性绑定。
2. 前三个 static 变量定义了默认的编码格式、视图解析器的前缀、后缀等。
3. 从前三行配置中，可以看出来，Thymeleaf 模板的默认位置在 resources/templates目录下，默认的后缀是 html 。
4. 这些配置，如果开发者不自己提供，则使用 默认的，如果自己提供，则在 application.properties 中以 spring.thymeleaf 开始相关的配置。

而我们刚刚提到的，Spring Boot 为 Thymeleaf 提供的自动化配置类，则是 org.springframework.boot.autoconfigure.thymeleaf.ThymeleafAutoConfiguration ，部分源码如下：



可以看到，在这个自动化配置类中，首先导入 ThymeleafProperties ，然后 @ConditionalOnClass 注解表示当当前系统中存在 TemplateMode 和 SpringTemplateEngine类时，当前的自动化配置类才会生效，即只要项目中引入了 Thymeleaf 相关的依赖，这个配置就会生效。

这些默认的配置我们几乎不需要做任何更改就可以直接使用了。如果开发者有特殊需求，则可以在 application.properties 中配置以 spring.thymeleaf 开头的属性即可。

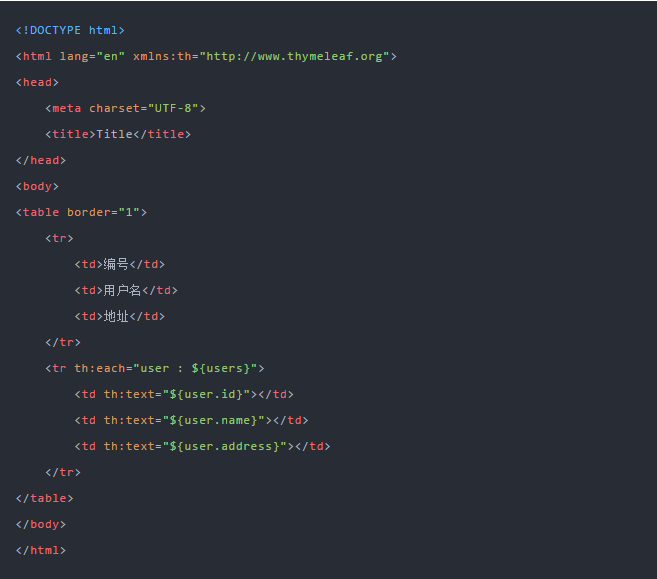
* 创建 Controller

接下来我们就可以创建 Controller 了，实际上引入 Thymeleaf 依赖之后，我们可以不做任何配置。新建的 IndexController 如下：



在 IndexController 中返回逻辑视图名+数据，逻辑视图名为 index ，意思我们需要在 resources/templates 目录下提供一个名为 index.html 的 Thymeleaf 模板文件。

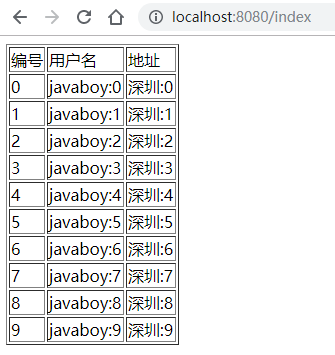
* 创建 Thymeleaf



在 Thymeleaf 中，通过 th:each 指令来遍历一个集合，数据的展示通过 th:text 指令来实现，

注意 index.html 最上面要引入 thymeleaf 名称空间。

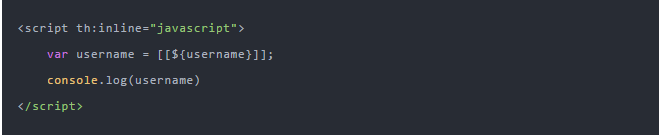
配置完成后，就可以启动项目了，访问 /index 接口，就能看到集合中的数据了：



另外，Thymeleaf 支持在 js 中直接获取 Model 中的变量。例如，在 IndexController 中有一个变量 username ：



在页面模板中，可以直接在 js 中获取到这个变量：



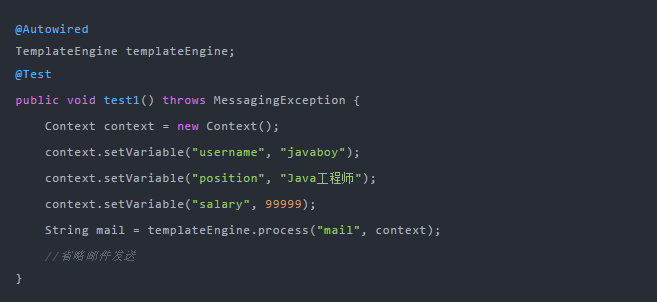
这个功能算是 Thymeleaf 的特色之一吧。

## 手动渲染

前面我们说的是返回一个 Thymeleaf 模板，我们也可以手动渲染 Thymeleaf 模板，这个一般在邮件发送时候有用，例如我在 resources/templates 目录下新建一个邮件模板，如下：



这一个 HTML 模板中，有几个变量，我们要将这个 HTML 模板渲染成一个 String 字符串，再把这个字符串通过邮件发送出去，那么如何手动渲染呢？



1. 渲染时，我们需要首先注入一个 TemplateEngine 对象，这个对象就是在 Thymeleaf 的自动化配置类中配置的（即当我们引入 Thymeleaf 的依赖之后，这个实例就有了）。
2. 然后构造一个 Context 对象用来存放变量。
3. 调用 process 方法进行渲染，该方法的返回值就是渲染后的 HTML 字符串，然后我们将这个字符串发送出去。

这是 Spring Boot 整合 Thymeleaf 的几个关键点，关于 Thymeleaf 这个页面模板本身更多的用法，大家可以参考 Thymeleaf 的文档：https://www.thymeleaf.org。

## 总结

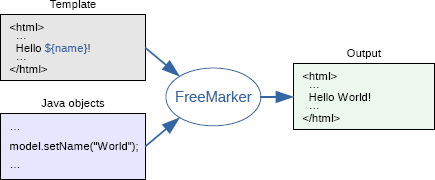
本文主要向大家简单介绍了 Spring Boot 和 Thymeleaf 整合时的几个问题，还是比较简单的，大家可以阅读 Thymeleaf 官方文档学习 Thymeleaf 的更多用法。

# Spring Boot 整合 Freemarker

## Freemarker 简介

这是一个相当老牌的开源的免费的模版引擎。通过 Freemarker 模版，我们可以将数据渲染成 HTML 网页、电子邮件、配置文件以及源代码等。Freemarker 不是面向最终用户的，而是一个 Java 类库，我们可以将之作为一个普通的组件嵌入到我们的产品中。

来看一张来自 Freemarker 官网的图片：



可以看到，Freemarker 可以将模版和数据渲染成 HTML 。

Freemarker 模版后缀为 .ftl(FreeMarker Template Language)。FTL 是一种简单的、专用的语言，它不是像 Java 那样成熟的编程语言。在模板中，你可以专注于如何展现数据， 而在模板之外可以专注于要展示什么数据。

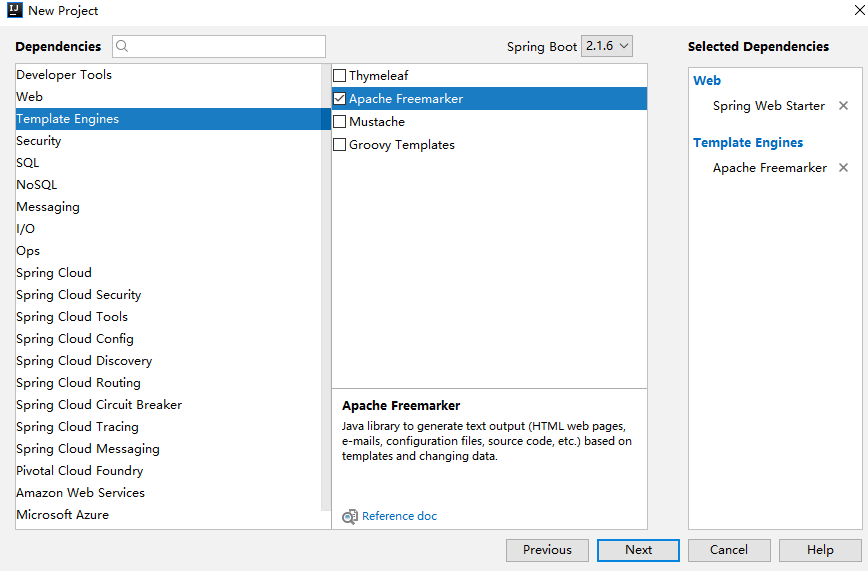
好了，这是一个简单的介绍，接下来我们来看看 Freemarker 和 Spring Boot 的一个整合操作。

## 实践

在 SSM 中整合 Freemarker ，所有的配置文件加起来，前前后后大约在 50 行左右，Spring Boot 中要几行配置呢？0 行！

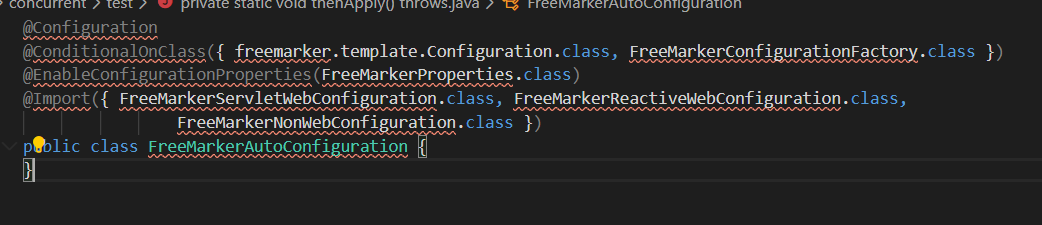
## 1.创建工程

首先创建一个 Spring Boot 工程，引入 Freemarker 依赖，如下图：





工程创建完成后，在 org.springframework.boot.autoconfigure.freemarker.FreeMarkerAutoConfiguration 类中，可以看到关于 Freemarker 的自动化配置：



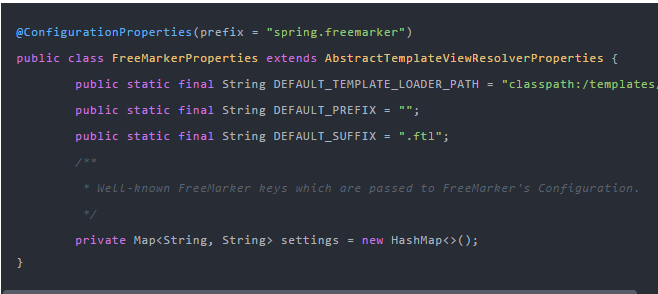
从这里可以看出，当 classpath 下存在 freemarker.template.Configuration 以及FreeMarkerConfigurationFactory 时，配置才会生效，也就是说当我们引入了 Freemarker 之后，配置就会生效。但是这里的自动化配置只做了模板位置检查，其他配置则是在导入的 FreeMarkerServletWebConfiguration 配置中完成的。那么我们再来看看 FreeMarkerServletWebConfiguration 类，部分源码如下：



我们来简单看下这段源码：

1. @ConditionalOnWebApplication 表示当前配置在 web 环境下才会生效。
2. ConditionalOnClass 表示当前配置在存在 Servlet 和 FreeMarkerConfigurer 时才会生效。
3. @AutoConfigureAfter 表示当前自动化配置在 WebMvcAutoConfiguration 之后完成。
4. 代码中，主要提供了 FreeMarkerConfigurer 和 FreeMarkerViewResolver。
5. FreeMarkerConfigurer 是 Freemarker 的一些基本配置，例如 templateLoaderPath、defaultEncoding 等
6. FreeMarkerViewResolver 则是视图解析器的基本配置，包含了viewClass、suffix、allowRequestOverride、allowSessionOverride 等属性。

另外还有一点，在这个类的构造方法中，注入了 FreeMarkerProperties：



FreeMarkerProperties 中则配置了 Freemarker 的基本信息，例如模板位置在 classpath:/templates/ ，再例如模板后缀为 .ftl，那么这些配置我们以后都可以在 application.properties 中进行修改。

如果我们在 SSM 的 XML 文件中自己配置 Freemarker ，也不过就是配置这些东西。现在，这些配置由  FreeMarkerServletWebConfiguration  帮我们完成了。

## 2.创建类

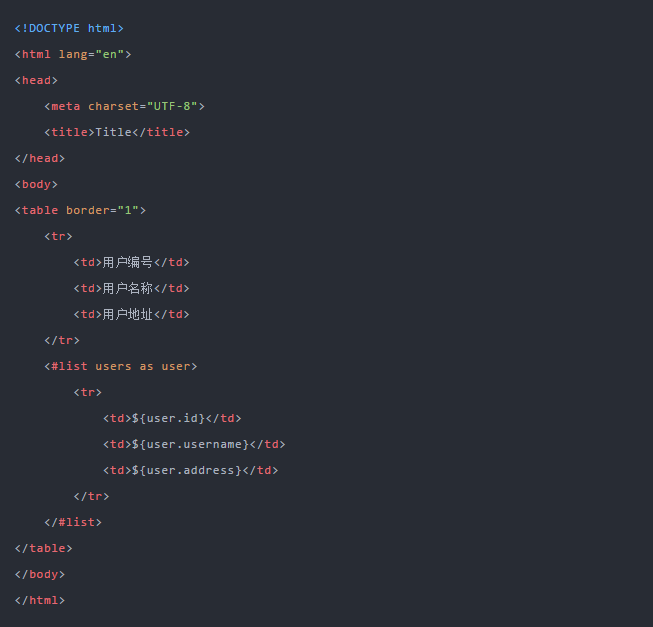
首先我们来创建一个 User 类，如下：



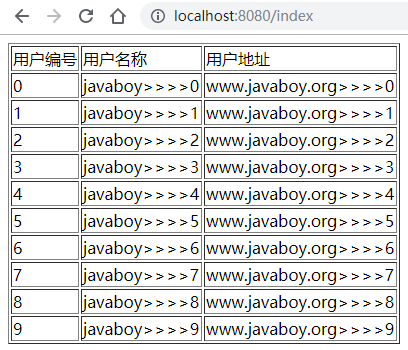
再来创建 UserController：



最后在 freemarker 中渲染数据：

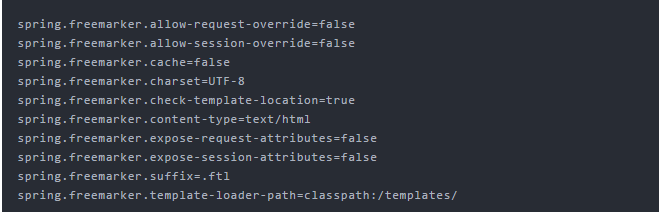


运行效果如下：



## 其他配置

如果我们要修改模版文件位置等，可以在 application.properties 中进行配置：



配置文件按照顺序依次解释如下：

1. HttpServletRequest的属性是否可以覆盖controller中model的同名项
2. HttpSession的属性是否可以覆盖controller中model的同名项
3. 是否开启缓存
4. 模板文件编码
5. 是否检查模板位置
6. Content-Type的值
7. 是否将HttpServletRequest中的属性添加到Model中
8. 是否将HttpSession中的属性添加到Model中
9. 模板文件后缀
10. 模板文件位置

好了，整合完成之后，Freemarker 的更多用法，就和在 SSM 中使用 Freemarker 一样了，这里我就不再赘述。

## 结语

本文和大家简单聊一聊 Spring Boot 整合 Freemarker，算是对 Spring Boot2 教程的一个补充（后面还会有一些补充），有问题欢迎留言讨论。