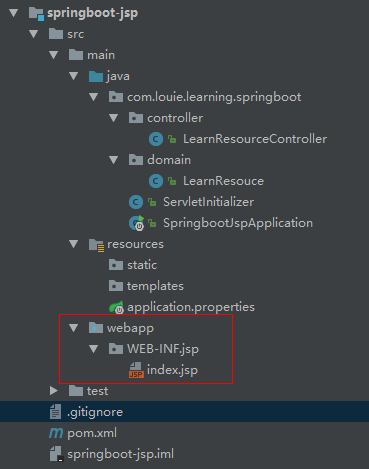
**整体框架结构**

先来看看整体的框架结构，跟前面介绍Thymeleaf的时候差不多，只是多了webapp这个用来存放jsp的目录，静态资源还是放在resources的static下面。  


**引入依赖**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | <!--WEB支持--> <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId> </dependency>  <!--jsp页面使用jstl标签--> <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId> </dependency>  <!--用于编译jsp--> <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  <scope>provided</scope> </dependency> |

使用内嵌的tomcat容器来运行的话只要这3个就好了。这里介绍下maven中scope依赖范围的概念，因为后续涉及到这个会有问题。

依赖范围就是用来控制依赖和三种classpath(编译classpath，测试classpath、运行classpath)的关系，Maven有如下几种依赖范围：

* **compile:**编译依赖范围。如果没有指定，就会默认使用该依赖范围。使用此依赖范围的Maven依赖，对于编译、测试、运行三种classpath都有效。典型的例子是spring-code,在编译、测试和运行的时候都需要使用该依赖。
* **test:** 测试依赖范围。使用次依赖范围的Maven依赖，只对于测试classpath有效，在编译主代码或者运行项目的使用时将无法使用此依赖。典型的例子是Jnuit,它只有在编译测试代码及运行测试的时候才需要。
* **provided:**已提供依赖范围。使用此依赖范围的Maven依赖，对于编译和测试classpath有效，但在运行时候无效。典型的例子是servlet-api,编译和测试项目的时候需要该依赖，但在运行项目的时候，由于容器以及提供，就不需要Maven重复地引入一遍。

**application.properties配置**

要支持jsp，需要在application.properties中配置返回文件的路径以及类型

spring.mvc.view.prefix=/WEB-INF/jsp/  
spring.mvc.view.suffix=.jsp

这里指定了返回文件类型为jsp,路径是在/WEB-INF/jsp/下面。

**控制类**

上面步骤有了，这里就开始写控制类，直接上简单的代码，跟正常的springMVC没啥区别：

@Controller  
public class LearnResourceController {  
  
 @RequestMapping("/")  
 public String index(Model model) {  
 List<LearnResouce> learnList = new ArrayList<LearnResouce>();  
 LearnResouce bean = new LearnResouce("官方参考文档", "Spring Boot Reference Guide", "http://docs.spring.io/spring-boot/docs/1.5.1.RELEASE/reference/htmlsingle/#getting-started-first-application");  
 learnList.add(bean);  
 bean = new LearnResouce("官方SpriongBoot例子", "官方SpriongBoot例子", "https://github.com/spring-projects/spring-boot/tree/master/spring-boot-samples");  
 learnList.add(bean);  
 bean = new LearnResouce("龙国学院", "Spring Boot 教程系列学习", "http://www.roncoo.com/article/detail/125488");  
 learnList.add(bean);  
 bean = new LearnResouce("嘟嘟MD独立博客", "Spring Boot干货系列 ", "http://tengj.top/");  
 learnList.add(bean);  
 bean = new LearnResouce("后端编程嘟", "Spring Boot教程和视频 ", "http://www.toutiao.com/m1559096720023553/");  
 learnList.add(bean);  
 bean = new LearnResouce("程序猿DD", "Spring Boot系列", "http://www.roncoo.com/article/detail/125488");  
 learnList.add(bean);  
 bean = new LearnResouce("纯洁的微笑", "Sping Boot系列文章", "http://www.ityouknow.com/spring-boot");  
 learnList.add(bean);  
 bean = new LearnResouce("CSDN——小当博客专栏", "Sping Boot学习", "http://blog.csdn.net/column/details/spring-boot.html");  
 learnList.add(bean);  
 bean = new LearnResouce("梁桂钊的博客", "Spring Boot 揭秘与实战", "http://blog.csdn.net/column/details/spring-boot.html");  
 learnList.add(bean);  
 bean = new LearnResouce("林祥纤博客系列", "从零开始学Spring Boot ", "http://412887952-qq-com.iteye.com/category/356333");  
 learnList.add(bean);  
 model.addAttribute("learnList", learnList);  
 return "index";  
 }  
}

**jsp页面编写**

<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<%@ **taglib** prefix="spring" uri="http://www.springframework.org/tags"%>  
<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>  
<!DOCTYPE html>  
<html >  
<head>  
 <title>learn Resources</title>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
</head>  
<body>  
  
<div style="text-align: center;margin:0 auto;width: 1000px; ">  
 <h1>学习资源大奉送，爱我就关注嘟嘟公众号：嘟爷java超神学堂（javaLearn）</h1>  
 <table width="100%" border="1" cellspacing="1" cellpadding="0">  
 <tr>  
 <td>作者</td>  
 <td>教程名称</td>  
 <td>地址</td>  
 </tr>  
 <**c:forEach** var="learn" items="**${**learnList**}**">  
 <tr class="text-info">  
 <td>**${**learn.author**}**</td>  
 <td>**${**learn.title**}**</td>  
 <td><a href="**${**learn.url**}**" class="btn btn-search btn-green" target="\_blank"><span>点我</span></a>  
 </td>  
 </tr>  
 </**c:forEach**>  
 </table>  
</div>  
</body>  
</html>

**启动类**

启动类不变还是最简单的

@SpringBootApplication  
public class SpringbootJspApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(SpringbootJspApplication.class, args);  
 }  
}

**内嵌Tomcat容器运行项目**

基本配置好了就可以启动项目，通过http://localhost:8080/访问，我使用的SpringBoot是１.５.6版本，jdk1.8,以前介绍过，运行项目有三种方式，这里我都做过了一次测试，发现在maven中jasper依赖有加provided和注释掉该依赖范围运行的效果不大一样，具体对比如下：

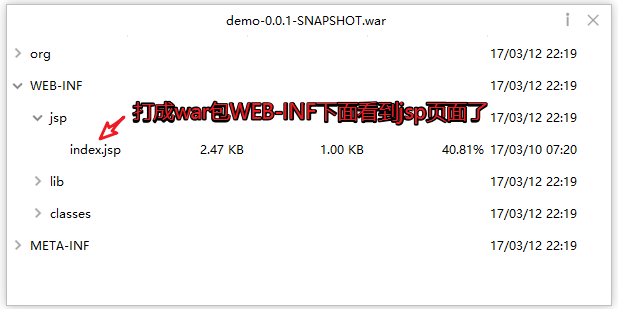
**有添加provided的情况：**

* 右键运行启动类，访问页面报404错误
* 使用spring-boot:run运行正常
* 打包成jar，通过 java -jar demo-0.0.1-SNAPSHOT.jar 运行报错
* 打包成war，通过 java -jar demo-0.0.1-SNAPSHOT.war 运行正常

**把provided 注释掉的情况**

* 右键运行启动类，访问页面正常
* spring-boot:run运行 访问页面正常
* 打包成jar，通过 java -jar demo-0.0.1-SNAPSHOT.jar 运行报错
* 打包成war，通过 java -jar demo-0.0.1-SNAPSHOT.war 运行正常

我测试了好几次都是这样，就是有加provided的时候，右键运行启动类访问页面的时候，提示404错误。  
其他3种情况都一样， jar运行也报404，spring-boot:run以及war运行都可以。

为什么jar包运行不行呢，我们打开打包的jar和war分别看看区别，如下2图所示：  
[](http://7xqch5.com1.z0.glb.clouddn.com/sringboot5-2.png)  
[](http://7xqch5.com1.z0.glb.clouddn.com/springboot5-33.png)  
从上面可以看出来，jar包运行的时候会404错误，因为默认jsp不会被拷贝进来，而war包里面有包含了jsp，所以没问题。

**内嵌Tomcat属性配置**

关于Tomcat的偶有属性都在org.springframework.boot.autoconfigure.web.ServerProperties配置类中做了定义，我们只需在application.properties配置属性做配置即可。通用的Servlet容器配置都已”server”左右前缀，而Tomcat特有配置都以”server.tomcat”作为前缀。下面举一些常用的例子。

**配置Servlet容器**：

|  |  |
| --- | --- |
| 1 | #配置程序端口，默认为8080  server.port= 8080  #用户绘画session过期时间，以秒为单位  server.session.timeout=  # 配置默认访问路径，默认为/  server.context-path= |

**配置Tomcat：**

|  |  |
| --- | --- |
| 1 | # 配置Tomcat编码,默认为UTF-8  server.tomcat.uri-encoding=UTF-8  # 配置最大线程数  server.tomcat.max-threads=1000 |

更为详细的Servlet容器配置以及Tomcat配置，可以前往博主之前文章查看：[Spring Boot干货系列：常用属性汇总](http://tengj.top/2017/02/28/springbootconfig/)

**外部的Tomcat服务器部署war包**

Spring Boot项目需要部署在外部容器中的时候，Spring Boot导出的war包如果直接在Tomcat的部署会报错，不信你可以试试看。  
需要做到下面两点修改才可以：

* 继承SpringBootServletInitializer  
  外部容器部署的话，就不能依赖于Application的main函数了，而是要以类似于web.xml文件配置的方式来启动Spring应用上下文，此时我们需要在启动类中继承SpringBootServletInitializer并实现configure方法：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | public class Application extends SpringBootServletInitializer {  @Override  protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {  return application.sources(Application.class);  } } |

这个类的作用与在web.xml中配置负责初始化Spring应用上下文的监听器作用类似，只不过在这里不需要编写额外的XML文件了。

也可以不用启动类集成，另外写一个类也可以。

public class ServletInitializer extends SpringBootServletInitializer {  
 @Override  
 protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {  
 return application.sources(SpringbootJspApplication.class);  
 }  
}

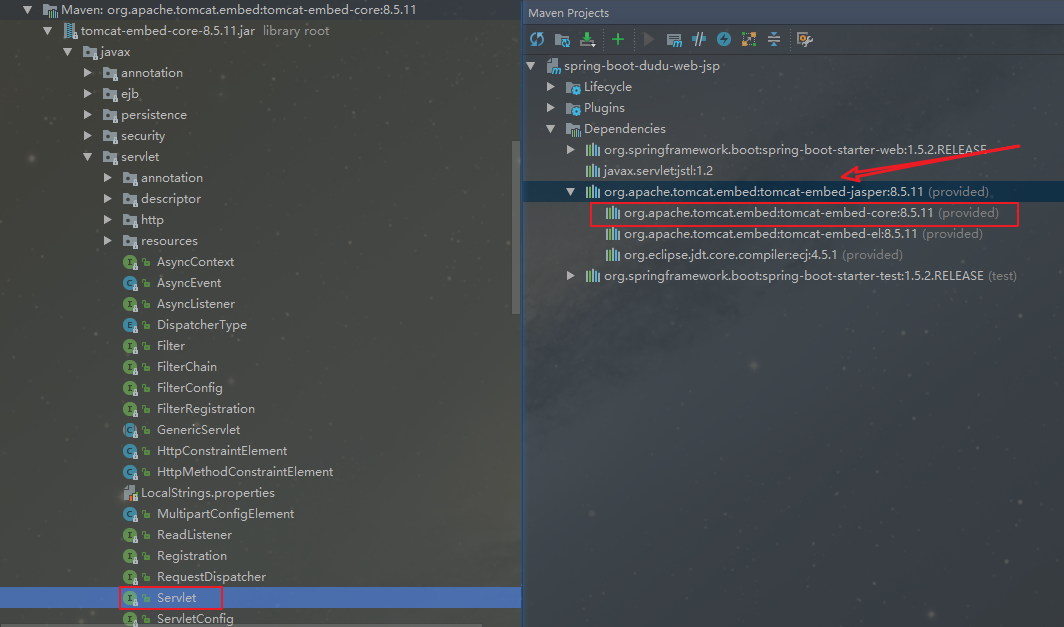
* pom.xml修改tomcat相关的配置  
  如果要将最终的打包形式改为war的话，还需要对pom.xml文件进行修改，因为spring-boot-starter-web中包含内嵌的tomcat容器，所以直接部署在外部容器会冲突报错。这里有两种方法可以解决，如下  
  **方法一**：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 | <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <exclusions>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  </exclusion>  </exclusions> </dependency> |

在这里需要移除对嵌入式Tomcat的依赖，这样打出的war包中，在lib目录下才不会包含Tomcat相关的jar包，否则将会出现启动错误。  
还有一个很关键的关键点，就是tomcat-embed-jasper中scope必须是provided。

|  |  |
| --- | --- |
| 1 2 3 4 5 | <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  <scope>provided</scope> </dependency> |

因为SpringBootServletInitializer需要依赖 javax.servlet，而tomcat-embed-jasper下面的tomcat-embed-core中就有这个javax.servlet，如果没用provided，最终打好的war里面会有servlet-api这个jar，这样就会跟tomcat本身的冲突了。这个关键点同样适应于下面说的第二种方法。

[](http://7xqch5.com1.z0.glb.clouddn.com/springboot5-4.png)

**方法二**：  
直接添加如下配置即可：

|  |  |
| --- | --- |
| 1 2 3 4 5 | <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  <scope>provided</scope> </dependency> |

provided的作用上面已经介绍的很透彻了，这里就不啰嗦了，这种方式的好处是，打包的war包同时适合java -jar命令启动以及部署到外部容器中。

如果你不喜欢默认的打包名称，你可以通过节点里添加内容。

|  |  |
| --- | --- |
| 1 2 3 | <build>  <finalName>springbootjsp</finalName> </bulid> |

最后启动tomcat输入http://localhost:8080/springbootjsp/查看效果，还是美美哒  


**关于使用jar部署**

上面已经测试过了，正常情况下包含jsp的页面是无法用jar的运行的，因为jsp默认是在webapp目录下，可是打包成jar是没有webapp这个目录结构的。

虽然网上有介绍说通过pom.xml配置，把WEB-INF目录复制到META-INF/resources下面。但是博主试了一整天还是访问不了，最后放弃了。各位如何有兴趣可以继续尝试，毕竟war也可以通过java -jar命令来启动的不是么。

spring-boot-maven-plugin 1.5.3版本的打包控件打出来的jar包是无法访问jsp的，直接404，即使你的jsp文件打包进去了。

spring-boot-maven-plugin 1.4.2版本就是可以的，我不知道这两个版本打包控件有什么区别，有知道的大神可以讲讲。