

狂神说SpringBoot10：Web开发静态资源处理

秦疆 狂神说 2020-03-17

狂神说SpringBoot系列连载课程，通俗易懂，基于SpringBoot2.2.5版本，欢迎各位狂粉转发关注学习。未经作者授权，禁止转载

Web开发探究

简介

好的，同学们，那么接下来呢，我们开始学习SpringBoot与Web开发，从这一章往后，就属于我们实战部分的内容了；

其实SpringBoot的东西用起来非常简单，因为SpringBoot最大的特点就是自动装配。

使用SpringBoot的步骤：

- 1、创建一个SpringBoot应用，选择我们需要的模块，SpringBoot就会默认将我们的需要的模块自动配置好
- 2、手动在配置文件中配置部分配置项目就可以运行起来了
- 3、专注编写业务代码，不需要考虑以前那样一大堆的配置了。

要熟练掌握开发，之前学习的自动配置的原理一定要搞明白！

比如SpringBoot到底帮我们配置了什么？我们能不能修改？我们能修改哪些配置？我们能不能扩展？

- 向容器中自动配置组件：***Autoconfiguration
- 自动配置类，封装配置文件的内容：***Properties

没事就找找类，看看自动装配原理！

我们之后来进行一个单体项目的小项目测试，让大家能够快速上手开发！

静态资源处理

静态资源映射规则

首先，我们搭建一个普通的SpringBoot项目，回顾一下HelloWorld程序！

写请求非常简单，那我们要引入我们前端资源，我们项目中有许多的静态资源，比如css，js等文件，这个SpringBoot怎么处理呢？

如果我们是一个web应用，我们的main下会有一个webapp，我们以前都是将所有的页面导在这里面的，对吧！但是我们现在的pom呢，打包方式是jar的方式，那么这种方式SpringBoot能不能来给我们写页面呢？当然是可以的，但是SpringBoot对于静态资源放置的位置，是有规定的！

我们先来聊聊这个静态资源映射规则：

SpringBoot中，SpringMVC的web配置都在 WebMvcAutoConfiguration 这个配置类里面；

我们可以去看看 WebMvcAutoConfigurationAdapter 中有很多配置方法；

有一个方法：addResourceHandlers 添加资源处理

```
@Override public void addResourceHandlers(ResourceHandlerRegistry registry) {
    if (!this.resourceProperties.isAddMappings()) { // 已禁用默认资源处理
        logger.debug("Default resource handling disabled"); return;
    } // 缓存控制 Duration cachePeriod = this.resourceProperties.getCache().getPeriod();
    CacheControl cacheControl = this.resourceProperties.getCache().getCachecontrol().toHttpCacheControl(); // webjars 配置 if (!registry.hasMappingForPattern("/webjars/**")) { customizeResourceHandlerRegistration(registry.addResourceHandler("/webjars/**")
        .addResourceLocations("classpath:/META-INF/resources/webjars/")
        .setCachePeriod(getSeconds(cachePeriod)).setCacheControl(cacheControl)); } // 静态资源配置 String staticPathPattern = this.mvcProperties.getStaticPathPattern();
    if (!registry.hasMappingForPattern(staticPathPattern)) { customizeResourceHandlerRegistration(registry.addResourceHandler(staticPathPattern)
        .addResourceLocations(getR
```

```
resourceLocations(this.resourceProperties.getStaticLocations()))  
                .setCachePeriod(getSeconds(cachePeriod)).s  
etCacheControl(cacheControl));    }}
```

读一下源代码：比如所有的 `/webjars/**`，都需要去 `classpath:/META-INF/resources/webjars/` 找对应的资源；

什么是webjars呢？

Webjars本质就是以jar包的方式引入我们的静态资源，我们以前要导入一个静态资源文件，直接导入即可。

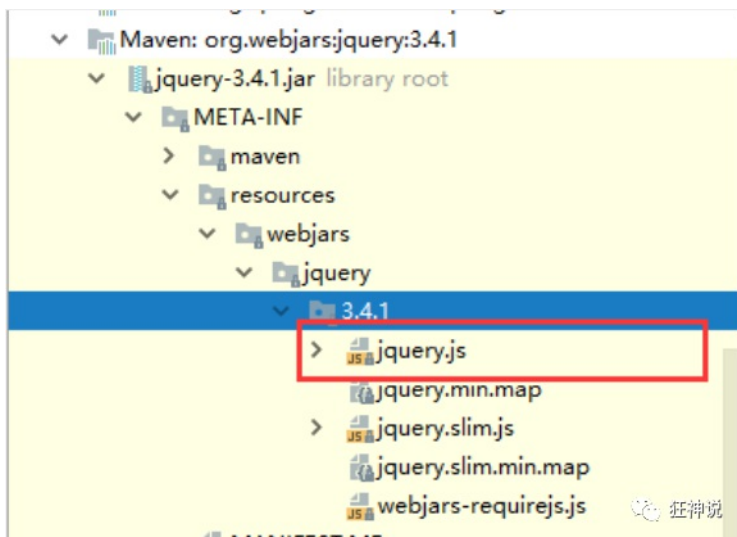
使用SpringBoot需要使用Webjars，我们可以去搜索一下：

网站：<https://www.webjars.org>

要使用jQuery，我们只要引入jQuery对应版本的pom依赖即可！

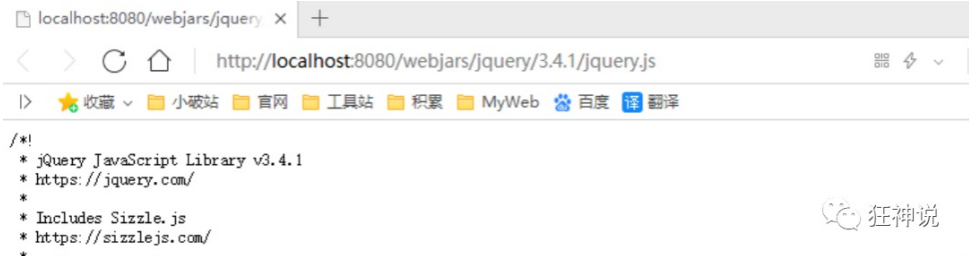
```
<dependency>    <groupId>org.webjars</groupId>    <artifactId>jquery</artif  
actId>    <version>3.4.1</version></dependency>
```

导入完毕，查看webjars目录结构，并访问jQuery.js文件！



访问：只要是静态资源，SpringBoot就会去对应的路径寻找资源，我们这里访问：

`http://localhost:8080/webjars/jquery/3.4.1/jquery.js`



第二种静态资源映射规则

那我们项目中要是使用自己的静态资源该怎么导入呢？我们看下一行代码：

我们去找`staticPathPattern`发现第二种映射规则：`/**`，访问当前的项目任意资源，它会去找`resourceProperties`这个类，我们可以点进去看一下分析：

```
// 进入方法public String[] getStaticLocations() {    return this.staticLocations;}// 找到对应的值private String[] staticLocations = CLASSPATH_RESOURCE_LOCATIONS;// 找到路径private static final String[] CLASSPATH_RESOURCE_LOCATIONS = {    "classpath:/META-INF/resources/",    "classpath:/resources/",    "classpath:/static/",    "classpath:/public/"    };
```

`ResourceProperties` 可以设置和我们静态资源有关的参数；这里面指向了它会去寻找资源的文件夹，即上面数组的内容。

所以得出结论，以下四个目录存放的静态资源可以被我们识别：

```
"classpath:/META-INF/resources/" "classpath:/resources/" "classpath:/static/" "classpath:/public/"
```

我们可以在`resources`根目录下新建对应的文件夹，都可以存放我们的静态文件：

比如我们访问 `http://localhost:8080/1.js` , 他就会去这些文件夹中寻找对应的静态资源文件;

自定义静态资源路径

我们也可以自己通过配置文件来指定一下, 哪些文件夹是需要我们放静态资源文件的, 在 `application.properties` 中配置:

```
spring.resources.static-locations=classpath:/coding/,classpath:/kuang/
```

一旦自己定义了静态文件夹的路径, 原来的自动配置就都会失效了!

首页处理

静态资源文件夹说完后, 我们继续向下看源码! 可以看到一个欢迎页的映射, 就是我们的首页!

```
@Beanpublic WelcomePageHandlerMapping welcomePageHandlerMapping(Application
Context applicationContext,
    FormattingConversionService mvcConversionService,
    ResourceUrlProvider mvcResource
UrlProvider) {    WelcomePageHandlerMapping welcomePageHandlerMapping = new
    WelcomePageHandlerMapping(        new TemplateAvailabilityProviders(applic
ationContext), applicationContext, getWelcomePage(), // getWelcomePage 获得
欢迎页        this.mvcProperties.getStaticPathPattern());    welcomePageHand
lerMapping.setInterceptors(getInterceptors(mvcConversionService, mvcResourc
eUrlProvider));    return welcomePageHandlerMapping;}
```

点进去继续看

```
private Optional<Resource> getWelcomePage() {    String[] locations = getRe
sourceLocations(this.resourceProperties.getStaticLocations());    // ::是ja
va8 中新引入的运算符    // Class::function的时候function是属于Class的，应该是静态方
法。    // this::function的function是属于这个对象的。    // 简而言之，就是一种语法糖而
已，是一种简写    return Arrays.stream(locations).map(this::getIndexHtml).filt
er(this::isReadable).findFirst();} // 欢迎页就是一个location下的的 index.html 而
private Resource getIndexHtml(String location) {    return this.resourceLoa
der.getResource(location + "index.html");}
```

欢迎页，静态资源文件夹下的所有 index.html 页面；被 /** 映射。

比如我访问 <http://localhost:8080/>，就会找静态资源文件夹下的 index.html

新建一个 index.html，在我们上面的3个目录中任意一个；然后访问测试 <http://localhost:8080/> 看结果！

关于网站图标说明：

Welcome Page

Spring Boot supports both static and templated welcome pages. It first looks for an `index.html` file in the configured static content locations. If one is not found, it then looks for an `index` template. If either is found, it is automatically used as the welcome page of the application.

Custom Favicon

As with other static resources, Spring Boot looks for a `favicon.ico` in the configured static content location. If such a file is present, it is automatically used as the favicon of the application.

与其他静态资源一样，Spring Boot在配置的静态内容位置中查找 `favicon.ico`。如果存在这样的文件，它将自动用作应用程序的favicon。

1、关闭SpringBoot默认图标

#关闭默认图标spring.mvc.favicon.enabled=false

2、自己放一个图标在静态资源目录下，我放在 public 目录下

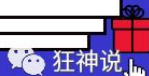
3、清除浏览器缓存！刷新网页，发现图标已经变成自己的了！





🔊 长按关注

据说关注小狂神的人都
走向人生巅峰了，还不
长按关注一下？



狂神说

喜欢就在看！



狂神说