<https://segmentfault.com/a/1190000007021883?utm_source=tuicool&utm_medium=referral>

# [springboot+gradle+vue+webpack 组合使用](https://segmentfault.com/a/1190000007021883)

最近使用springboot, vue, webpack开发了一款SPA应用，这里主要记录开发+调试+打包的过程及做法。  
使用的技术  
[gradle](https://gradle.org/)  
[springboot](http://projects.spring.io/spring-boot/)  
[vue.js](https://github.com/vuejs/vue)  
[webpack](http://webpack.github.io/)

下面是开发时的项目结构目录，主要分成后端项目与前端项目，后端主要用于编写服务器逻辑，前端则是编写页面和存放静态资源。

-- springboot-vue-webpack-examples

|

|-- server

| |

| |-- src // 源码

| |-- build // 构建输出目录

|

|-- ui

| |

| |-- build-scripts // vue 构建脚本

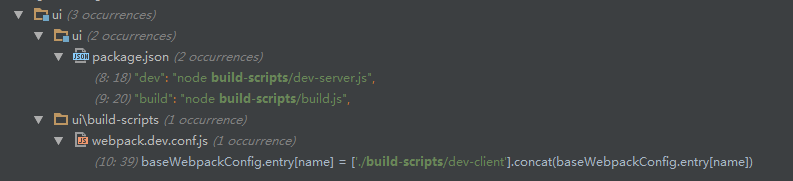
| |-- config // vue 配置文件

| |-- src // 源码

| |-- static // 静态资源

| |-- build // 构建输出目录

ui项目通过vue init webpack ui命令创建使用的是[vue提供的模板](https://github.com/vuejs-templates/webpack)，之所以使用该模板因为里面已经配置好了调试与生产打包方式，我们不需要做什么更改就可以直接使用。

默认在vue的模板中构建脚本是放置在build目录中，这里将构建脚本移置build-scripts目录中，主要是因为gradle的默认构建输出目录是build为了减少配置而修改了vue构建脚本的目录，我感觉这是最简单方便的，因为比如我们在使用代码版本控制器的时候build目录会被忽略而不提交至服务器，我不想去修改太多的配置。将vue构建脚本移动后需要修改几个点，如下图  


ui项目的gradle脚本配置，最终我们打包发布都是直接通过gradle命令，所以需要将node构建集成到gradle中。

project(':ui') {

apply plugin: 'com.moowork.node'

task **cnpmInstall**(type: NpmTask) {

**group** = 'node'

args = ['install', '--registry=http://registry.cnpmjs.org']

}

task **buildUI**(type: NpmTask, dependsOn: cnpmInstall) {

**group** = 'node'

args = ['run', 'build']

}

jar.dependsOn buildUI

task **runDev**(type: NpmTask, dependsOn: cnpmInstall) {

**group** = 'node'

args = ['run', 'dev']

}

}

cnpmInstall 该命令主要用于依赖安装，之所以需要这个命令主要是因为我们的网络环境不太好，将镜像设置为国内的下载依赖比较稳定。  
buildUI 调用package.json中的命令构建ui。  
runDev 可以通过gradlew :ui:runDev命令启动ui。

ui/config/index.js 配置修改

// see http://vuejs-templates.github.io/webpack **for** documentation.

var path = require('path')

var assetsRoot = path.resolve(\_\_dirname, '../build/resources/main/ui')

module.exports = {

build: {

env: require('./prod.env'),

index: path.resolve(assetsRoot, 'index.html'),

assetsRoot: assetsRoot,

assetsSubDirectory: 'static',

assetsPublicPath: '/',

productionSourceMap: true,

// Gzip off **by** default as many popular static hosts such as

// Surge **or** Netlify already gzip all static assets **for** you.

// Before setting to `true`, make sure to:

// npm install --save-dev compression-webpack-plugin

productionGzip: false,

productionGzipExtensions: ['js', 'css']

},

dev: {

env: require('./dev.env'),

port: 3000,

assetsSubDirectory: 'static',

assetsPublicPath: '/',

proxyTable: {

'/api/\*\*': 'http://localhost:8080'

},

// CSS Sourcemaps off **by** default because relative paths are "buggy"

// with **this** option, according to the CSS-Loader README

// (https://github.com/webpack/css-loader#sourcemaps)

// In our experience, they generally work as expected,

// just be aware **of** **this** issue **when** enabling **this** option.

cssSourceMap: false

}

}

assetsRoot 将webpack构建的资源输出至build/resources/main/ui目录，gradle打包会将ui加入至classpath中，spring查找静态资源有ui目录区分比较方便。  
proxyTable 增加代理配置将/api/\*\* URL下的所有请求转发至服务后端即springboot启动的服务，这样做的目的是为了方便debug，在vue webpack中已经配置好了hot-dev，在开发中我们修改了前端js或者vue不需要**重新构建**或**重启**应用前端就能响应。所以在开发中我们启动服务后的访问入口是  
http://localhost:3000  
其实3000是express dev-server的端口，这里也体现了上面为什么要配置proxyTable，当我们从dev-server作为访问入口时与后端服务并不是同一个，存在**跨域**问题，但是通过代理可以避免这个问题，同时这并不会对生产环境造成影响，当我们发布项目之后可以直接通过springboot服务作为访问入口，因为在生产环境中我们不需要hot-reload功能。

服务端WEB配置

@Configuration

@RestController

**public** **class** **WebConfiguration** **extends** **WebMvcConfigurerAdapter** {

@Value("classpath:/ui/index.html")

**private** Resource indexHtml;

@Bean

@Order(Ordered.HIGHEST\_PRECEDENCE)

**public** CharacterEncodingFilter **characterEncodingFilter**() {

CharacterEncodingFilter filter = **new** CharacterEncodingFilter("UTF-8", **true**);

**return** filter;

}

@Override

**public** **void** **addResourceHandlers**(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/\*\*").addResourceLocations("classpath:/ui/");

}

@Bean

**public** ServletRegistrationBean **apiV1ServletBean**(WebApplicationContext wac) {

DispatcherServlet servlet = **new** DispatcherServlet(wac);

ServletRegistrationBean bean = **new** ServletRegistrationBean(servlet, "/api/v1/\*");

bean.setName("api-v1");

**return** bean;

}

@RequestMapping("/")

**public** Object **index**() {

**return** ResponseEntity.ok().body(indexHtml);

}

}

addResourceHandlers 增加静态资源访问路径。  
apiV1ServletBean 之所以增加一个servlet配置，是为了与静态资源区分，后端服务都是通过restful接口交互而静态资源是通过/根目录的方式访问。  
index 根目录返回index.html。

至此基础配置就差不多是这样打包发布就可以直接通过gradlew build。发布时我们无需修改任何配置及代码，与开发环境是一致。而在开发环境中我们也保留良好的开发，及调试环境。  
**注：运行时不需要nodejs环境**

示例代码：[springboot-vue-webpack-examples](https://github.com/kevin70/springboot-vue-webpack-examples)

<https://github.com/kevin70/springboot-vue-webpack-examples>