快速构建vue项目可以使用vue官方提供的vue-cli来实现，但是为了理解在构建过程中每个配置的作用和含义所以没有采用vue-cli。

# 一、构建一个完整的工程。

## 1、https://nodejs.org/en/下载安装node.js。

## 2、新建工程目录demo。

## 3、在demo下执行npm init来创建一个package.json，编辑package.json添加必备模块：

"devDependencies": {

"vue": "^2.1.10",

"vue-router": "^2.1.3" // vue 路由插件

},

"dependencies": {

"vue-loader": "^10.0.2", // vue 组件（\*.vue）的webpack模块加载器

"webpack": "^1.14.0", // 模块加载器兼打包工具

"webpack-dev-server": "^1.16.2" // 轻量的 node.js express 服务器，用于开发调试

}

备注：关于package.json的一点说明：

dependencies 里配置安装的包是程序正常运行需要的包。  
devDependencies 里配置安装的包是仅开发时需要的包，比如 一些单元测试的包之类的。

## 4、初始化工程：新建目录结构，目录结构如下所示：

demo

|-- package.json

|-- index.html // 启动页面

|-- webpack.config.js // webpack配置文件

|-- src

|-- components // vue UI 组件目录

|-- views // vue页面组件目录

|-- main.js // 入口文件

|-- router.js // vue-router配置

|-- app.vue // 工程首页组件

## 5、配置webpack

### （1）、webpack 默认读取 webpack.config.js，文件名不能随便改，其中 entry 是必须配置的。

module.exports = {

entry: './src/main.js',

output: {

path: './dist',

publicPath: '/dist/',

filename: 'build.js'

}

}

其中output的各配置项作用如下：

path: './dist' 打包后js、css、image等存放的目录；

publicPath: '/dist/' 可以不配置，如果不配置则取默认publicPath: '/'，在实际项目中，静态资源一般集中放在一个文件夹下，比如static目录，那么这里就应该改成publicPath: '/static/'，相应的 index.html中引用的 JS 也要改成src="/static/build.js"，publicPath 可以解释为最终发布的服务器上 build.js 所在的目录，其他静态资源也应当在这个目录下。

filename: 'build.js' 打包的js文件名，index.html 引用的 JS 要和这里保持一致。

### （2）、配置 webpack-dev-server，只需在package.json添加以下启动命令即可。

"scripts": {

"dev": "webpack-dev-server --inline --hot --open"

}

各命令参数作用如下：

--inline 一共两种模式，默认为iframe模式，inline和iframe模式最明显的区别就是访问路径的不同，iframe模式的访问路径是http://localhost:8080/webpack-dev-server/，实际上iframe模式页面嵌入的<iframe>的地址还是http://localhost:8080/，那岂不是可以直接访问，不禁想问为啥不直接访问呢？因为无论是哪种模式，都是为了做到修改代码后能自动刷新，其中iframe模式是在修改代码后，重新加载iframe，而--inline是刷新浏览器，本质上都是重新全部加载一遍

--hot iframe不需要配置，配置了反而不能正常刷新了，所以只能配合--inline使用，作用是开启热替换，修改代码后，浏览器只会重新加载修改的组件代码，不会全部重新加载

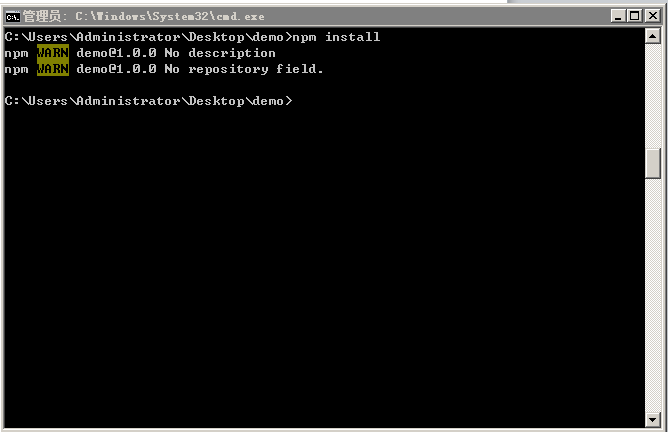
--open 自动打开浏览器。

## 6、执行npm install（npm i）安装。

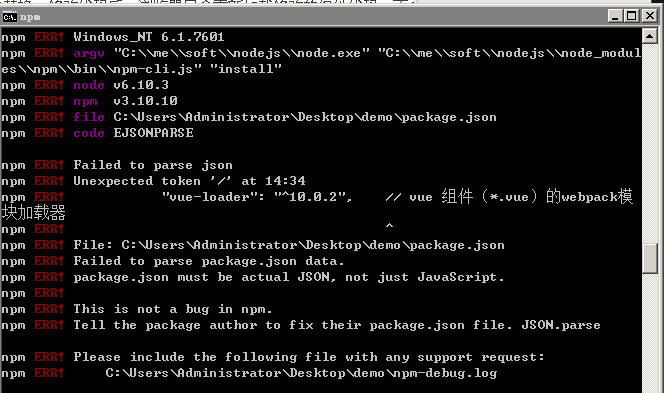
安装成功会在demo下生成一个文件夹node\_modules，里面会有package.json里面的devDependencies和dependencies配置的依赖库文件。

其中node的安装分为全局模式和本地模式：  
本地模式运行（$npm install），包会被安装到和你的应用程序代码的本地node\_modules目录下。  
在全局模式下（$npm install -g），Node包会被安装到Node的安装目录下的node\_modules下。  
获知使用$npm set global=true来设定安装模式，$npm get global可以查看当前使用的安装模式。

警告：demo的package.json文件没有描述信息，repository属性不规范。

****

**安装没成功：查看demo下生成了node\_modules文件夹，但是里面没有文件。**

****

没安装成功是因为在package.json文件里面写了注释，package.json文件不支持“//”添加注释。

## 7、安装成功，执行npm run dev。

浏览器会自动打开http://localhost:8080/，能访问（可以在index.html添加内容来确认是否启动成功）则说明上面的配置没问题。

## 8、安装其他依赖包。

执行**npm install <name>**,例如npm install jquery 就会默认安装jquery的最新版本，也可以通过在后面加版本号的方式安装指定版本，如npm install jquery @3.0.6**安装。**

## 9、小结npm常用指令。

### （1）、npm install <name> -g  将包安装到全局环境中。

但是代码中，直接通过require()的方式是没有办法调用全局安装的包的。全局的安装是供命令行使用的，就好像全局安装了vmarket后，就可以在命令行中直接运行vm命令

### （2）、npm install <name> --save  安装的同时，将信息写入package.json中。

项目路径中如果有package.json文件时，直接使用npm install方法就可以根据dependencies配置安装所有的依赖包，这样代码提交到github时，就不用提交node\_modules这个文件夹了。

### （3）、npm init会引导你创建一个package.json文件，包括名称、版本、作者这些信息等。

### （4）、npm remove <name>移除

### （5）、npm update <name>更新

### （6）、npm ls 列出当前安装的了所有包

### （7）、npm root 查看当前包的安装路径

### （8）、npm root -g  查看全局的包的安装路径

### （9）、npm help  帮助，如果要单独查看install命令的帮助，可以使用的npm help install

**（10）、npm install --save:**

* **会**把msbuild包安装到node\_modules目录中。
* **会**在package.json的**dependencies**属性下添加msbuild。
* 之后运行npm install命令时，**会**自动安装msbuild到node\_modules目录中。
* 之后运行npm install --production或者注明NODE\_ENV变量值为production时，**会**自动安装msbuild到node\_modules目录中。

**（11）、npm install --save-dev:**

* **会**把msbuild包安装到node\_modules目录中。
* **会**在package.json的**devDependencies**属性下添加msbuild。
* 之后运行npm install命令时，**会**自动安装msbuild到node\_modules目录中。
* 之后运行npm install --production或者注明NODE\_ENV变量值为production时，**不会**自动安装msbuild到node\_modules目录中。

**使用原则:**

**运行时需要用到的包使用--save，否则使用--save-dev。**

未完待续，在下一篇《vue-router开发单页面应用总结》中，准备记录总结使用vue-router开发一个单页面应用涉及的知识点和遇到的问题及解决办法。

本篇是在上一篇《vue2+vue-router2+webpack1构建一个完整的项目》完成的demo上继续开发单页面应用的总结，包括但不限于vue、vue-route、webpackr基础知识总结和开发过程中遇到的问题及解决办法。

# 二、vue-router单页面开发

## 1、新建页面

在 src/views 目录下新建一个 pageOne.vue，相当于新建一个“页面”，后面访问 /pageOne 的就是此页面的内容；

<template>

<div>

这是{{page}}页面

</div>

</template>

<script>

module.exports = {

data: function () {

return {

page: 'pageOne'

}

}

}

</script>

## 2、配置路由

把传入的配置提取到router.js中，方便后续配置，外部调用方式为new VueRouter(require('./router')) 。router.js内容如下：

module.exports = {

routes: [

{

path: '/pageOne',

component: require('./views/pageOne.vue')

}

]

}

## 3、首页

首页（index.html）只需引入打包后的 js 文件（src和webpack.config的output 配置一致），#app是整个网站的挂载点，简单点说其实整个网站就是一个 vue 的实例，#app就是实例el属性值。

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>vuedemo</title>

</head>

<body>

<div id="app"></div>

<script src="dist/build.js"></script>

</body>

</html>

## 4、配置入口js

main.js内容如下：

const Vue = require('vue');

const VueRouter = require('vue-router');

const App = require('./app.vue');

Vue.use(VueRouter);

const router = new VueRouter(require('./router'))

new Vue({

el: '#app',

router: router,

render: h => h(App)

})

## 5、链接路由

链接路由的不再使用<a v-link="{ path: '/ pageOne }"></a>，改为组件的方式。app.vue内容如下：

<template>

<div>

<div>

<router-link to="/pageOne"> pageOne </router-link>

</div>

<div>

<router-view></router-view>

</div>

</div>

</template>

## 6、配置loader

需要在webpack.config.js中添加vue对应的loader，才能将vue文件解析成可执行的代码。

module: {

loaders: [

{

test: /\.vue$/, loader: 'vue'

}

]

}

这是工程会报错：原因是一些依赖包没有安装，按照错误提示，安装依赖包：

在package.json的devDependencies里添加：

{

"vue-hot-reload-api": "^1.3.3",

"vue-html-loader": "^1.2.3",

"babel-core": "^6.13.2",

"babel-loader": "^6.2.4",

"babel-plugin-transform-runtime": "^6.12.0",

"babel-preset-es2015": "^6.13.2"

}

执行npm install

再次启动还是报错：是由于vue-template-compiler没有安装，执行npm install vue-template-compiler –save安装，重启成功。

## 7、支持CSS样式

### （1）、在.vue文件里使用<style></style>设置样式：

vue 文件由<template><style><script>三部分组成，各自的功能也不言而喻，

在上面的任何一个 vue 文件中添加<style>会报错：Module not found: Error: Cannot resolve module 'vue-style-loader' in，错误提示中的 loader 没有安装，vue-style-loader 依赖 css-loader，需要一并安装，不然又会报上面的错了。这里不需要在 webpack.config.js 中配置 loader，因为这两个其实是 vue-loader 所依赖的模块，只要安装了下面的两个loader就不会报错了。package.json 的dependencies里增加的配置如下：

{

"vue-style-loader":"^1.0.0",

"css-loader":"^0.25.0"

}

安装了这两个 loader 之后，再启动发现就不报错了，而且在<sytle>中添加的样式也生效了。

### （2）、使用外部样式：

引入外部 CSS 文件，需要在webpack.config中添加CSS文件的loader配置，上面已经安装过了相应的loader。

{

test:/\.css$/,

loader:'vue-style!css'

}

JS中引用CSS文件可以像引入JS模块一样调用require，比如在pageOne.vue中添加require('../css/pageOne.css');，那么pageOne.css的样式就会被添加到页面中。

在src下新建一个文件夹css，在该文件夹下新建一个文件pageOne.css，在pageOne.vue中引入：

<template>

<div id="aaa">

这是{{page}}页面

</div>

</template>

<script>

require('../css/pageOne.css');

module.exports = {

data: function () {

return {

page: 'pageOne'

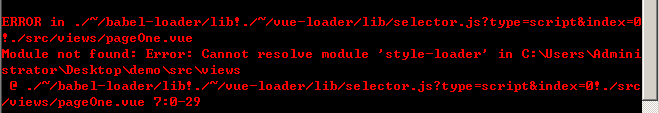
}

}

}

</script>

报错：



查询后再webpack.base.conf.js添加：

{

  test: /\.css$/,

  include: [

    /src/ //表示在src目录下的css需要编译

  ],

  loader: 'style-loader!css-loader'

},

重新启动依旧报错，后来发现是css依赖包没安装

Vue引入外部文件依赖：css-loader 、style-loader 、file-loader ，css-loader已经安装，安装缺少的包：

cnpm install style-loader --save-dev

cnpm install file-loader --save-dev

工程正常成功启动。

## 8、支持添加图片等静态资源

静态资源（图片、图标字体、音频、视频、svg文件等）对应的loader为url-loader，loader信息及配置如下：

Package.json:

{

"url-loader":"^0.5.7",

"file-loader":"^0.9.0" *// url-loader依赖file-loader*

}

webpack.config:

{

test: /\.(jpe?g|png|gif|svg|mp3)$/,

loader: "url"

}

配置完成执行npm install

在src下新建文件夹images，在images下放一张图片a.png，修改pageOne.css：

#aaa{

background-image: url(../images/a.png);

}

执行成功。

## 9、支持CSS预处理语言（less、sass、stylus）

### （1）、只在.vue文件里使用：

在package.json里添加这些预处理语言对应的loader，各个预处理语言的loader信息如下：

{

"less": "^2.3.1", *// less-loader依赖less*

"less-loader": "^2.2.3",

"node-sass": "^3.4.2", *// sass-loader依赖node-sass*

"sass-loader": "^4.0.2",

"stylus": "^0.54.5", *// stylus-loader依赖stylus*

"stylus-loader": "^2.4.0"

}

添加完成执行npm install

在pageOne.vue添加测试代码如下：

<template>

<div class="about">

<div class="test-less">测试less</div>

<div class="test-sass">测试sass</div>

<div class="test-stylus">测试stylus</div>

</div>

</template>

<style lang="less">

@color: #00f;

.test-less {

color: @color;

}

</style>

<style lang="sass">

$color: #0ff;

.test-sass {

color: $color;

}

</style>

<style lang="stylus">

color = #f00;

.test-stylus {

color: color;

}

</style>

执行成功。

### （2）、引用文件的方式来加载样式，须要在webpack.config.js里配置loader。

配置如下：

{

test: /\.less$/,

loader: "vue-style!css!less"

},

{

test: /\.scss/,

loader: "vue-style!css!sass"

},

{

test: /\.styl/,

loader: "vue-style!css!stylus"

}

将pageOne.vue中各种语言<style>节点中的样式放进一个对应文件中，然后在pageOne.vue中引用各个文件如下：

require('../css/pageOne.less');

require('../css/pageOne.scss');

require('../css/pageOne.styl');

## 10、创建另一个页面pageTwo

### （1）、在src/views下新建pageTwo.vue:

<template>

<div id="">

这是{{page}}页面

</div>

</template>

<script>

module.exports = {

data: function () {

return {

page: 'pageTwo'

}

}

}

</script>

<style type="text/css">

</style>

### （2）、配置pageTwo的路由，修改router.js:

module.exports = {

routes: [

{

path: '/pageOne',

component: require('./views/pageOne.vue')

},

{

path: '/pageTwo',

component: require('./views/pageTwo.vue')

}

]

}

### （3）、在app.vue里添加路由链接:

<template>

<div>

<div>

<router-link to="/pageOne">pageOne</router-link>

<router-link to="/pageTwo">pageTwo</router-link>

</div>

<div>

<router-view></router-view>

</div>

</div>

</template>

## 11、打包发布

### （1）、在package.json里添加如下命令：

"scripts": {

"dev": "webpack-dev-server --inline --hot --open",

"build":"webpack --progress --colors"

}

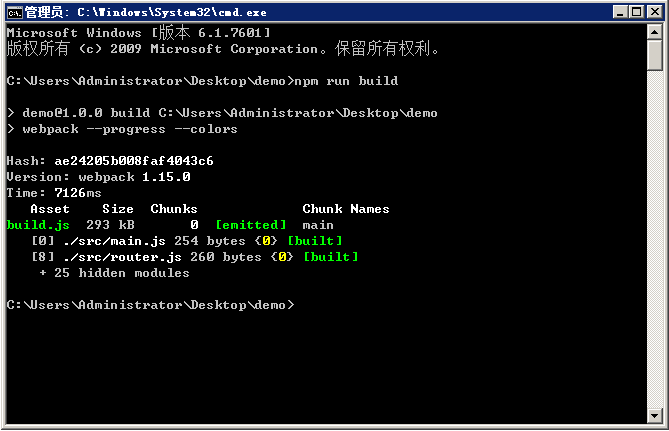
说明：

Progress：让编译的输出内容带有进度。

Colors：让编译的输出内容带有颜色。

### （2）、执行npm run build

执行完后会新增一个dist目录，里面只有一个build.js文件，新增的目录及文件是由webpack.config.js的output配置指定的。



该命令会把.vue、所有的css、图片资源等等全部放到build.js问价里面，这样有些混乱，所以为了把这些资源在打包的时候分开，要使用webpack插件。

## 12、webpack插件的使用。

### （1）、如何使用插件

插件的使用一般是在 webpack 的配置信息（webpack.config.js） plugins 选项中指定。比如：

**var** webpack = require('webpack');

module.exports = {

plugins: [

**new** webpack.BannerPlugin('Hello webpack')

]

}

### （2）、压缩js

压缩js的插件为webpack.optimize.UglifyJsPlugin，修改webpack.config.js为：

var webpack = require('webpack');

module.exports = {

entry: './src/main.js',

output: {

path: './dist', //打包后js、css、image等存放的目录；

publicPath: '/dist/', //最终发布的服务器上 build.js 所在的目录，其他静态资源也应当在这个目录下。

filename: 'build.js' //打包的js文件名，index.html 引用的 JS 要和这里保持一致。

},

module: {

loaders: [

{

test: /\.vue$/, //配置loaders加载.vue文件

loader: 'vue'

},

{

test: /\.css$/, //配置loaders加载.css文件

include: [

/src/ //表示在src目录下的css需要编译

],

loader: 'style-loader!css-loader'

},

{

test: /\.(jpe?g|png|gif|svg|mp3)$/, //配置loaders加载静态资源

loader: 'url'

},

{

test: /\.less$/,

loader: "vue-style!css!less"

},

{

test: /\.scss/,

loader: "vue-style!css!sass"

},

{

test: /\.styl/,

loader: "vue-style!css!stylus"

}

]

},

plugins: [

new webpack.optimize.UglifyJsPlugin({ //js压缩插件

compress: {

warnings: false //warinings:false是在删除无用代码的时候，不显示警告

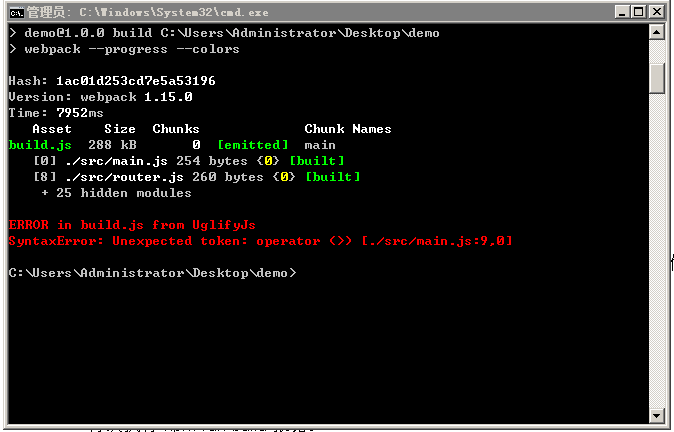
}

})

]

}

再次执行npm run build报错：



原因是：不识别箭头函数，Chrome支持es6所以运行的时候没有报错，单 import就报错了。

解决办法：添加babel-loader，将ES6语法进行转换：

a、在package.json添加如下模块并安装：

{

"babel-loader": "^6.2.8",

"babel-core": "^6.18.2",

"babel-preset-es2015":"6.18.0"

}

b、配置webpack.config.js：

{

test: /\.js$/,

exclude: /node\_modules/,

loader: 'babel'

}

C、在package.json里添加如下配置：

"babel":{

"presets":["es2015"],

"comments":false

},

或者在demo下新建.babelrc文件：

{

"presets":["es2015"],

"comments":false

}

再次执行 npm run build，js压缩成功105k。

### （3）、提取CSS

css提取插件为extract-text-webpack-plugin，这个不是webpack自带的，需要自己安装，在package.json中添加并安装：

{

"extract-text-webpack-plugin": "^1.0.1"

}

webpack.config.js配置如下，原来的style-loader就要去掉了，毕竟我们的目标是将CSS提取出来而不是放在head中。

var ExtractTextPlugin = require("extract-text-webpack-plugin");

module.exports = {

module: {

loaders: [{

test: /\.css$/,

loader: ExtractTextPlugin.extract("css")

},

{

test: /\.less$/,

loader: ExtractTextPlugin.extract("css!less")

},

{

test: /\.scss/,

loader: ExtractTextPlugin.extract("css!sass")

},

{

test: /\.styl/,

loader: ExtractTextPlugin.extract("css!stylus")

},

]

},

plugins: [

new ExtractTextPlugin("css/style.css")

]

}

执行npm run build 后再dist下生成css/style.css, 在index.html中添加引用后就生效了。

启动发现vue文件中<style>里的样式并未纳入到style.css中，而是继续嵌入式在head中，这是由于vue-loader默认用的是vue-style-loader，想要将vue文件中的样式也提取到文件中，需要在webpack.config添加vue的loaders的配置：

module.exports = {

module: {},

plugins: [],

vue: {

loaders: {

css: ExtractTextPlugin.extract("css"),

less: ExtractTextPlugin.extract("css!less"),

scss: ExtractTextPlugin.extract("css!sass"),

stylus: ExtractTextPlugin.extract("css!stylus")

}

}

}

### （4）、使用PostCSS压缩css。

extract-text-webpack-plugin插件只负责将css提取出来，使用PostCSS插件来压缩。

a、在package.json中添加postcss-loader并安装：

{

"postcss-loader":"^1.2.0"

}

b、在package.json中添加PostCSS依赖插件cssnano并安装：

{

"cssnano":"^3.8.1"

}

c、在webpack.config.js中注册PostCSS所需插件cssnano，vue要单独指定:

module.exports = {

module: {

loaders: [

{

test: /\.css$/,

loader: ExtractTextPlugin.extract("css!postcss")

},

{

test: /\.less$/,

loader: ExtractTextPlugin.extract("css!postcss!less")

},

{

test: /\.scss/,

loader: ExtractTextPlugin.extract("css!postcss!sass")

},

{

test: /\.styl/,

loader: ExtractTextPlugin.extract("css!postcss!stylus")

}

]

},

vue: {

loaders: {

css: ExtractTextPlugin.extract("css!postcss"),

less: ExtractTextPlugin.extract("css!postcss!less"),

scss: ExtractTextPlugin.extract("css!postcss!sass"),

stylus: ExtractTextPlugin.extract("css!postcss!stylus")

},

postcss: [require("cssnano")]

},

postcss: [require("cssnano")]

}

配置完成执行build，利用cssnano完成了压缩，其实cssnano不止是完成了压缩，还优化了CSS代码，比如丢弃重复的样式规则、压缩选择器、合并规则等等。PostCSS还有很多插件可以帮助提升效率，比如autoprefixer可以自动添加浏览器前缀等。

### （5）、提取图片

提取图片只需要在webpack.config.js增加url-loader的参数即可。

{

test: /\.(jpe?g|png|gif|svg)$/,

loader: "url",

query:{

name:'images/[name].[ext]',

limit:10000 // 单位：字节

}

}

name:'images/[name].[ext]' 将符合test正则的图片都存在images目录下，[name].[ext]是文件名模板，更多占位符请参考file-loader文档。

limit:10000 小于10kb的图片才会被转化成DataUrl，设为0并不意味着所有的图片都不被转换，如果想所有图片都不被转换，建议设为1。

## 13、静态资源添加版本号

对于前后端分离的站点来讲，前端所有资源都是静态的，因此防止浏览器缓存就非常有必要。防止缓存的方法一般有两种，一种是在文件名中添加文件内容的hash值（build.xxxx.js），另外一种方法是给每个http请求加版本号参数（build.js?xxxx）。下面分别说明下两种方法如何配置。

### （1）、文件名模板占位符。

JS、CSS 及 Image 的配置如下：

// hash可配置选项 [<hashType>:hash:<digestType>:<length>] 即 [哈希算法类型:hash:哈希摘要类型:长度]

// JS

filename: 'build.[hash:8].js' // hash

filename: 'build.js?[hash:8]' // query

// CSS

new ExtractTextPlugin("css/style.[hash:8].css") //hash

new ExtractTextPlugin("css/style.css?[hash:8]") //query

// Image

name: 'images/[name][hash:8].[ext]' // hash

name: 'images/[name].[ext]?[hash:8]' // query

打包后的CSS和JS文件名改变了，那意味着index.html中对应的引用地址也要跟着改变，如果每次构建完都要手动去改index.html中引用地址也很不方便，希望最好能在打包完成后自动修改对应的引用地址，就像图片一样，CSS或页面中引用的图片自动改成了最新的文件名。下面将阐述如何通过生成首页来解决这个问题。

### （2）、自动生成首页。

自动生成首页需要安装webpack插件html-webpack-plugin，模块信息如下：

{

"html-webpack-plugin":"^2.24.1"

}

调用插件可以不传配置项，那么index.html会生成到output.path所指定的目录下，生成的页面会引用构建好的CSS和JS文件，但是不会包含<div id="app">。通过配置生成首页的模板（template），可以在模板中添加无法通过配置来完成的内容，还可以设置首页标题（title）、生成的目录及文件名（filename）、favicon图标（favicon）等等，所有配置参见项目github地址。

// 使用默认配置

new HtmlWebpackPlugin()

// 自定义配置

new HtmlWebpackPlugin({

title:'首页标题',

filename:'../index.html',

template:'index.tpl.html',

favicon:'src/images/favicon.ico'

...

})

自动生成的首页只能用于发布，webpack-dev-server并不会访问生成后的首页，而且在我们开发的过程中也不需要配置诸如JS及CSS的压缩、添加版本号等，因为这些配置无疑会带来server启动变慢、占用资源等问题，解决这些问题最直接的办法就是将开发和生产得配置文件分开。

## 14、其他插件

除了上述插件外，还有一些打包时需要用到的插件，列举如下。

使用 webpack 的 DefinePlugin 来指定生产环境，以便在压缩时可以让 UglifyJS 自动删除代码块内的警告语句。

new webpack.DefinePlugin({

'process.env': {

NODE\_ENV: '"production"'

}

})

OccurrenceOrderPlugin 可以根据模块调用次数，给模块分配 ids，常被调用的 ids 分配更短的 id，使得 ids 可预测，降低文件大小。既然能减小打包文件体积，当然要用上。

new webpack.optimize.OccurrenceOrderPlugin()

## 15、分离配置文件

现在所有配置都放在一个配置文件上，不区分开发和生产模式，导致了开发过程中启动 webpack-dev-server 慢、许多配置占用资源等问题，如何解决？最有效的办法就是分离配置文件，将开发和生产的配置文件分开。

使用 vue-cli 生成的 webpack模板的 vue 工程就将开发与生产的配置文件分开，开发配置文件中不会包含 JS 压缩、CSS 提取等配置。

能够分离配置文件基于以下两点：

1. webpack及webpack-dev-server命令行支持指定配置文件webpack --config webpack.my.config.js
2. webpack及webpack-dev-server支持NodeJS API调用。

### （1）、方法一。

分别建两个配置文件，一个用于开发，一个用于生产，开发配置文件只包含基本的配置，生产配置文件包含上文提到过的所有配置，最后两个配置文件分别对应如下两个命令：

"dev": "webpack-dev-server --inline --hot --config webpack.dev.config.js",

"prod": "webpack --progress --colors --config webpack.prod.config.js"

在config webpack.dev.config.js中删除多余配置，只留下工程正常运行所需配置。

至此，webpack配置文件如下：

Package.json:

{

"name": "demo",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"dev": "webpack-dev-server --inline --hot --open --config webpack.dev.config.js",

"prod": "webpack --progress --colors --config webpack.prod.config.js",

"build":"webpack --progress --colors"

},

"author": "",

"license": "ISC",

"babel":{

"presets":["es2015"],

"comments":false

},

"devDependencies": {

"babel-core": "^6.13.2",

"babel-loader": "^6.2.4",

"babel-plugin-transform-runtime": "^6.12.0",

"babel-preset-es2015": "^6.13.2",

"file-loader": "^0.11.2",

"style-loader": "^0.18.2",

"vue-hot-reload-api": "^1.3.3",

"vue-html-loader": "^1.2.3",

"vue-loader": "^10.0.2",

"webpack": "^1.14.0",

"webpack-dev-server": "^1.16.2",

"url-loader":"^0.5.7",

"babel-loader": "^6.2.8",

"babel-core": "^6.18.2",

"babel-preset-es2015":"6.18.0",

"extract-text-webpack-plugin": "^1.0.1",

"postcss-loader":"^1.2.0",

"cssnano":"^3.8.1",

"html-webpack-plugin":"^2.24.1"

},

"dependencies": {

"jquery": "^3.2.1",

"vue": "^2.1.10",

"vue-router": "^2.1.3",

"vue-template-compiler": "^2.3.4",

"vue-style-loader": "^1.0.0",

"css-loader": "^0.25.0",

"less": "^2.3.1",

"less-loader": "^2.2.3",

"node-sass": "^3.4.2",

"sass-loader": "^4.0.2",

"stylus": "^0.54.5",

"stylus-loader": "^2.4.0"

}

}

webpack.dev.config.js:

module.exports = {

entry: './src/main.js',

output: {

path: './dist', //打包后js、css、image等存放的目录；

publicPath: '/dist/', //最终发布的服务器上 build.js 所在的目录，其他静态资源也应当在这个目录下。

filename: 'build.js' //打包的js文件名，index.html 引用的 JS 要和这里保持一致。

},

module: {

loaders: [

{

test: /\.vue$/, //配置loaders加载.vue文件

loader: 'vue'

},

{

test: /\.css$/, //配置loaders加载.css文件

include: [

/src/ //表示在src目录下的css需要编译

],

loader:'vue-style!css'

},

{

test: /\.(jpe?g|png|gif|svg|mp3)$/, //配置loaders加载静态资源

loader: 'url'

},

{

test: /\.less$/,

loader: "vue-style!css!less"

},

{

test: /\.scss/,

loader: "vue-style!css!sass"

},

{

test: /\.styl/,

loader: "vue-style!css!stylus"

},

{

test: /\.js$/,

exclude: /node\_modules/,

loader: 'babel'

}

]

},

}

webpack.prod.config.js:

var webpack = require('webpack');

var ExtractTextPlugin = require("extract-text-webpack-plugin");

module.exports = {

entry: './src/main.js',

output: {

path: './dist', //打包后js、css、image等存放的目录；

publicPath: '/dist/', //最终发布的服务器上 build.js 所在的目录，其他静态资源也应当在这个目录下。

filename: 'build.[hash:8].js' //打包的js文件名，index.html 引用的 JS 要和这里保持一致。

},

module: {

loaders: [

{

test: /\.vue$/, //配置loaders加载.vue文件

loader: 'vue'

},

{

test: /\.css$/, //配置loaders加载.css文件

include: [

/src/ //表示在src目录下的css需要编译

],

loader: ExtractTextPlugin.extract("css!postcss")

},

{

test: /\.(jpe?g|png|gif|svg|mp3)$/, //配置loaders加载静态资源

loader: 'url',

query:{

name: 'images/[name][hash:8].[ext]' ,

limit:10000 // 单位：字节

}

},

{

test: /\.less$/,

loader: ExtractTextPlugin.extract("css!postcss!less")

},

{

test: /\.scss/,

loader: ExtractTextPlugin.extract("css!postcss!sass")

},

{

test: /\.styl/,

loader: ExtractTextPlugin.extract("css!postcss!stylus")

},

{

test: /\.js$/,

exclude: /node\_modules/,

loader: 'babel'

}

]

},

plugins: [

new webpack.optimize.UglifyJsPlugin({ //js压缩插件

compress: {

warnings: false //warinings:false是在删除无用代码的时候，不显示警告

}

}),

new ExtractTextPlugin("css/style.[hash:8].css")

],

vue: {

loaders: {

css: ExtractTextPlugin.extract("css!postcss"),

less: ExtractTextPlugin.extract("css!postcss!less"),

scss: ExtractTextPlugin.extract("css!postcss!sass"),

stylus: ExtractTextPlugin.extract("css!postcss!stylus")

},

postcss: [require("cssnano")]

},

postcss: [require("cssnano")]

}

执行：npm rn dev读取webpack.prod.config.js配置文件，不压缩文件。

打包发布时执行npm run prod 读取webpack.prod.config.js配置文件，完成js、css和图片等的打包压缩。

### （2）、方法二。

按照第二点的思路，也就是 vue-cli 的做法，我们需要写代码来启动服务器和构建项目，开发与生产对应的不仅仅是两个配置文件，还包括两个调用各自API的JS文件，或者将配置文件的内容放在调用API里面，这样开发和生产就分别对应一个文件，推荐使用前面的方式。最终启动的命令应该是这样的：

// 统一都放在build文件夹下

"dev": "node build/dev.js",

"prod": "node build/prod.js"

增加 build 文件夹及配置文件。

vue2pratice

|-- package.json

|-- build

|-- prod.js // 调用webpack NodeJS API构建

|-- dev.js // 调用webpack-dev-server NodeJS API启动服务器

|-- webpack.dev.config.js // 开发配置

|-- webpack.prod.config.js // 生产配置

#### a、开发环境配置

webpack-dev-server调用方式在webpack文档中都有详细的示例，结合webpack及webpack-dev-server各自提供的API，dev.js内容如下：

var webpack = require('webpack');

var webpackDevServer = require('webpack-dev-server');

var config = require("./webpack.dev.config");

var compiler = webpack(config);

var server = new webpackDevServer(compiler, {

publicPath: config.output.publicPath

});

server.listen(port, "0.0.0.0");

// 打包完毕后启动浏览器

server.middleware.waitUntilValid(function () {

console.log(`> Listening at ${url}`);

require('child\_process').exec(`start ${url}`);

})

不要试图将webpack.dev.config里的配置项entry及output的路径改为相对于build文件夹的路径，运行时还是相对根目录去查找的，因此可以保持不变，或者如果不放心，可以全部改为绝对路径，这样绝对不会出错。绝对路径示例如下：

var path = require('path');

module.exports = {

entry: path.resolve(\_\_dirname,'../src/main.js'),

output: {

path: path.resolve(\_\_dirname,'../dist'),

publicPath: '/dist/',

filename: 'build.js'

},

module: { }

};

热替换

采用了上述方式后，你会发现热替换无效了，即使配置了hot:true也没效果，这个问题其实在官网有详细说明，如何生效呢？需要三步：

把"webpack-dev-server/client?http://localhost:8080/", "webpack/hot/dev-server"加入到webpack.dev.config.js的entry；

把new webpack.HotModuleReplacementPlugin()加入到webpack配置文件的plugins项；

把hot:true加入到webpack-dev-server的配置项里面。

各步骤对应改动如下：

// 第一步：由于涉及端口，在dev.js中修改比较合适

var config = require("./webpack.dev.config");

config.entry.app = [`webpack-dev-server/client?${url}`, "webpack/hot/dev-server", config.entry.app];

// 第二步：在webpack.dev.config.js增加插件

plugins: [

new webpack.HotModuleReplacementPlugin()

]

// 第三步：配置hot:true

new webpackDevServer(compiler, {

stats: { colors: true },

hot:true,

publicPath: config.output.publicPath

});

注意：在dev.js中修改entry需要结合自己在webpack.dev.config.js中配置的entry类型来修改，entry可以配置成字符串、数组、对象，不管entry是什么类型，只要将"webpack-dev-server/client?http://localhost:8080/", "webpack/hot/dev-server"和各个入口JS放到一个数组即可。

#### b、生产环境配置

webpack打包的调用方式同样也在webpack文档中有详细的示例，prod.js内容如下：

var webpack = require("webpack");

var config = require('./webpack.prod.config');

webpack(config, function (err, stats) {

// 打印编译结果信息

process.stdout.write(stats.toString());

});

如果你采用hash防缓存的话，一旦文件改动过，那么就会生成新的文件名，如果之前打包的文件没删的话，就会越来越多，既然现在采用调用API的方式来构建，这意味这你可以调用NodeJS API来做很多事，比如在打包之前先把生成目录清空，打包完之后直接FTP上传到服务器等等。完整示例如下：

var webpack = require("webpack");

var config = require('./webpack.prod.config');

var fs = require('fs');

var path = require('path');

// 打包之前先清空dist目录

clearFolder(path.resolve(\_\_dirname, '../dist/'));

// 开始打包

webpack(config, function (err, stats) {

process.stdout.write(stats.toString());

// 打包完成后ftp上传到xx目录

upload()

});

// 清空文件夹

function clearFolder(path) {

var files = [];

if (fs.existsSync(path)) {

files = fs.readdirSync(path);

files.forEach(function (file, index) {

var curPath = path + "/" + file;

if (fs.statSync(curPath).isDirectory()) {

clearFolder(curPath);

} else {

fs.unlinkSync(curPath);

console.log(curPath);

}

});

fs.rmdirSync(path);

}

}

// ftp上传

function upload() {

// vinyl-fs、vinyl-ftp未在package.json列出，这里只是给出示意

var vfs = require('vinyl-fs');

var vftp = require('vinyl-ftp');

var conn = new vftp(/\* ... \*/);

vfs.src([path.resolve(\_\_dirname, '../dist/')], {buffer: false})

.pipe(conn.dest('/'));

}

## 16、懒加载

什么是懒加载？就是需要某个资源的时候才去请求，而不是“勤快的”一开始就把所有的资源给准备好。为什么需要懒加载？在前面打包的时候我们发现，最终整个网站的JS代码都被打包到一个JS中了，这无疑会造成JS文件太大，首次加载慢的问题，而且有些资源并不需要立即加载。vue 懒加载的原理是啥？vue 懒加载是由 vue 异步组件和 webpack 的代码分割技术来完成的，当路由被访问的时候才加载对应组件。具体原理及配置方式请参考[vue-router 文档](https://router.vuejs.org/zh-cn/advanced/lazy-loading.html)。按文档中的配置方式配置如下：

routes: [

{

path: '/pageOne',

component: resolve => require.ensure([], () => resolve(require('./views/pageOne.vue')),’ pageOne’)

}

]

路由到 pageOne 的时候，加载了 build.js 以及pageOne.vue 中引用的图片，这就是懒加载的效果，路由被访问了，路由对应的组件的资源才会请求。上面我们配置了 chunk 的名称，但发现请求的名字并不是 pageOne.js，这是因为 webpack 中还没有配置好，需要在 output 中增加 chunkFileName 的配置：

// webpack.dev.config.js

output: {

chunkFilename:'[name].js'

}

// webpack.prod.config.js

output: {

chunkFilename:'[name].[hash:8].js'

}

打包时会遇到一个问题，就是这些异步加载的模块中引用的 CSS 并不会被打包进 style.css 中，而是放在异步模块的 JS 文件中，比如上面 about.vue 的 CSS 就放在 about.js 中，如果你想把异步模块的CSS都提起到 style.css 中，只需修改插件 ExtractTextPlugin 的配置如下：

new ExtractTextPlugin("css/style.[hash:8].css", {allChunks: true})

## 17、多入口打包配置

懒加载可以减小打包 JS 的体积，另一种方式就是配置多入口，多入口的好处至少有以下两点：

1. 多个 JS 文件可以充分利用浏览器并发请求，提高加载速度；
2. 将公共库代码提取出到一个 JS 文件中，提高304命中率，同时也分离了业务代码和公共代码。

目前我们的项目只配置了一个入口 JS，即 entry:'build.js'，多入口仍然是在 entry 上做配置，如何配置在 [webpack entry](http://webpack.github.io/docs/configuration.html#entry) 中有详细说明。配置如下：

entry: {

app: path.resolve(\_\_dirname, '../src/main.js'),

vendors: ['vue', 'vue-router']

},

output: {

filename: '[name].js', // 生产配置为： [name].[hash:8].js

...

}

* app：业务代码，原单入口的 JS 文件
* vendors：公共库代码，将公共库列在数组中即可，也可以是相对路劲
* output.filename：之前写死为 build.js，现在配置了多入口，那么打包应该输出多个 JS 文件，文件名就需要用模板来指定，最后默认会按 chunk 的名字输出，即 app.js 和 vendors.js。

对于公用模块的提取并不是像入口文件一样，webpack 可以根据入口文件逐层作静态分析，完成公共模块的提取还需要依赖 webpack 自带的一个插件webpack.optimize.CommonsChunkPlugin，插件只需传入上述 entry 的公共模块的块（chunk）名。更多 CommonsChunkPlugin 的配置请参考 [webpack plugins](http://webpack.github.io/docs/list-of-plugins.html#commonschunkplugin)。

// 开发和生产配置都一样

plugins: [

new webpack.optimize.CommonsChunkPlugin({

name: 'vendors'

})

]

完成了上述配置后，开发环境的 index.html 还需要手动将各个入口 JS 文件添加进去，注意： vendors.js 必须先引入。

<!-- index.html -->

<script type="text/javascript" src="/dist/vendors.js"></script>

<script type="text/javascript" src="/dist/app.js"></script>

在「热替换」章节提过，entry在上面改为对象后，在dev.js中修改entry为：

var config = require("./webpack.dev.config");

config.entry.app=["webpack-dev-server/client?http://localhost:8080/", "webpack/hot/dev-server", config.entry.app];

## 18、使用element-ui

### （1）、安装依赖。

npm install style-loader –S

npm install css-loader –S

npm install file-loader –S

### （2）、配置webpack.dev.config.js。

{

test: /\.(eot|svg|ttf|woff|woff2)(\?\S\*)?$/,

loader: 'file-loader'

},

{

test: /\.(png|jpe?g|gif|svg)(\?\S\*)?$/,

loader: 'file-loader',

query: {

name: '[name].[ext]?[hash]'

}

}

此时，完整的webpack.dev.config.js文件如下：

var path = require('path')

var webpack = require('webpack');

module.exports = {

entry: './src/main.js',

output: {

path: './dist', //打包后js、css、image等存放的目录；

publicPath: '/dist/', //最终发布的服务器上 build.js 所在的目录，其他静态资源也应当在这个目录下。

filename: 'build.js', //打包的js文件名，index.html 引用的 JS 要和这里保持一致。

chunkFilename:'[name].js'

},

module: {

loaders: [

{

test: /\.vue$/, //配置loaders加载.vue文件

loader: 'vue'

},

{

test: /\.css$/, //配置loaders加载.css文件

loader:'vue-style!css'

},

{

test: /\.(jpe?g|png|gif|svg|mp3)$/, //配置loaders加载静态资源

loader: 'url'

},

{

test: /\.less$/,

loader: "vue-style!css!less"

},

{

test: /\.scss/,

loader: "vue-style!css!sass"

},

{

test: /\.styl/,

loader: "vue-style!css!stylus"

},

{

test: /\.js$/,

exclude: /node\_modules/,

loader: 'babel'

},

{

test: /\.(eot|svg|ttf|woff|woff2)(\?\S\*)?$/,

loader: 'file-loader'

},

{

test: /\.(png|jpe?g|gif|svg)(\?\S\*)?$/,

loader: 'file-loader',

query: {

name: '[name].[ext]?[hash]'

}

}

]

},

plugins: [

new webpack.ProvidePlugin({

"$":"jquery",

"jQuery":"jquery",

"window.jQuery":"jquery"

})

]

}

### （3）、在mian.js中引用。

import ElementUI from 'element-ui'

import 'element-ui/lib/theme-default/index.css'

Vue.use(ElementUI)

## 19、使用axios请求资源。

### （1）、安装axios。

npm install axios –S

### （2）、在 main.js 中引入 axios。

import axios from 'axios'

这时候如果在其它的组件中，是无法使用 axios 命令的。所以要将 axios 改写为 Vue 的原型属性：

Vue.prototype.axios = axios

### （3）、在组件的 methods 中使用axios 命令。

methods: {

submitForm () {

this.axios ({

method: 'post',

url: '/user',

data: {

name: 'wise',

info: 'wrong'

}

})

}