**Main.js入口文件修改**

如果你是用vue.js官网提供的脚手架工具并沿用默认配置的话，

你执行npm run dev的时候会出来页面，

是因为你根目录下的package.json文件里script配置了

"dev": "node build/dev-server.js"，

也就是其实执行的是dev-server.js这个文件，里面有定义

var webpackConfig = require('./webpack.dev.conf');

因为我们这个脚手架工具里是用webpack来打包项目文件的，依赖的webpack.dev.conf文件里又定义了

var baseWebpackConfig = require('./webpack.base.conf');

**在这个依赖webpack.base.conf文件里面entry入口文件就配置了**

**app: './src/main.js'**

所以当你运行npm run dev的时候就从main.js这个入口文件开始执行了

**如何解决vue项目不能自动打开浏览器**





#### 3.目录结构及其对应作用

通过vue-cli搭建一个vue项目，会自动生成一系列文件，而这些文件具体是怎样的结构、文件对应起什么作用，可以看看下面的解释：

├── build/ # webpack 编译任务配置文件: 开发环境与生产环境

│ └── ...

├── config/

│ ├── index.js # 项目核心配置

│ └── ...

├ ── node\_module/ #项目中安装的依赖模块

── src/

│ ├── main.js # 程序入口文件

│ ├── App.vue # 程序入口vue组件

│ ├── components/ # 组件

│ │ └── ...

│ └── assets/ # 资源文件夹，一般放一些静态资源文件

│ └── ...

├── static/ # 纯静态资源 (直接拷贝到dist/static/里面)

├── test/

│ └── unit/ # 单元测试

│ │ ├── specs/ # 测试规范

│ │ ├── index.js # 测试入口文件

│ │ └── karma.conf.js # 测试运行配置文件

│ └── e2e/ # 端到端测试

│ │ ├── specs/ # 测试规范

│ │ ├── custom-assertions/ # 端到端测试自定义断言

│ │ ├── runner.js # 运行测试的脚本

│ │ └── nightwatch.conf.js # 运行测试的配置文件

├── .babelrc # babel 配置文件

├── .editorconfig # 编辑配置文件

├── .gitignore # 用来过滤一些版本控制的文件，比如node\_modules文件夹

├── index.html # index.html 入口模板文件

└── package.json # 项目文件，记载着一些命令和依赖还有简要的项目描述信息

└── README.md #介绍自己这个项目的，可参照github上star多的项目。

build/

因为有了模块，我们就可以更方便地使用别人的代码，想要什么功能，就加载什么模块。  
但是，这样做有一个前提，那就是大家必须以同样的方式编写模块，否则你有你的写法，我有我的写法，岂不是乱了套

CommonJS, AMD, CMD都是JS模块化的规范。

　　CommonJS是服务器端js模块化的规范，NodeJS是这种规范的实现。

　　AMD(异步模块定义)和CMD(通用模块定义)都是浏览器端js模块化的规范。RequireJS 遵循的是 AMD，SeaJS 遵循的是 CMD。

前端模块规范有三种：CommonJs,AMD和CMD。  
CommonJs用在服务器端，AMD和CMD用在浏览器环境  
AMD 是 RequireJS 在推广过程中对模块定义的规范化产出。  
CMD 是 SeaJS 在推广过程中对模块定义的规范化产出。  
AMD:提前执行（异步加载：依赖先执行）+延迟执行  
CMD:延迟执行（运行到需加载，根据顺序执行）

  


# Vue --（vue-cli脚手架）webpack配置文件详细分析

#### 1、脚手架版本

##### vue-cli 2.8.1

#### 2、配置文件 package.json

功能：在 package.json 中，可以找到开发和生产环境的入口

"scripts": {

"dev": "node build/dev-server.js", // 开发环境入口

"build": "node build/build.js" // 生产环境入口

}

* 1
* 2
* 3
* 4

#### 3、开发环境

* **build/dev-server.js** **：开发环境的入口文件**

功能：该文件中，使用express作为后端框架，结合一些关于webpack的中间件，搭建了一个开发环境。

// 配置文件

var config = require('../config')

// 如果 Node 的环境无法判断当前是 dev / product 环境

// 使用 config.dev.env.NODE\_ENV 作为当前的环境

if (!process.env.NODE\_ENV) {

process.env.NODE\_ENV = JSON.parse(config.dev.env.NODE\_ENV)

}

// 可以强制打开浏览器并跳转到指定 url 的插件

// https://github.com/sindresorhus/opn

var opn = require('opn')

// node自带的文件路径工具

var path = require('path')

// express框架

var express = require('express')

var webpack = require('webpack')

// 测试环境，使用的配置与生产环境的配置一样

// 非测试环境，即为开发环境，因为此文件只有测试环境和开发环境使用

var proxyMiddleware = require('http-proxy-middleware')

var webpackConfig = process.env.NODE\_ENV === 'testing'

// 生产环境配置文件

? require('./webpack.prod.conf')

// 开发环境配置文件

: require('./webpack.dev.conf')

// 端口号为命令行输入的PORT参数或者配置文件中的默认值

var port = process.env.PORT || config.dev.port

// 配置文件中 是否自动打开浏览器

var autoOpenBrowser = !!config.dev.autoOpenBrowser

// 配置文件中 http代理配置

// https://github.com/chimurai/http-proxy-middleware

var proxyTable = config.dev.proxyTable

// 启动 express 服务

var app = express()

// 启动 webpack 编译

var compiler = webpack(webpackConfig)

// 可以将编译后的文件暂存到内存中的插件

// https://github.com/webpack/webpack-dev-middleware

var devMiddleware = require('webpack-dev-middleware')(compiler, {

// 公共路径，与webpack的publicPath一样

publicPath: webpackConfig.output.publicPath,

// 不打印

quiet: true

})

// Hot-reload 热重载插件

// https://github.com/glenjamin/webpack-hot-middleware

var hotMiddleware = require('webpack-hot-middleware')(compiler, {

log: () => {}

})

// 当tml-webpack-plugin template更改之后，强制刷新浏览器

compiler.plugin('compilation', function (compilation) {

compilation.plugin('html-webpack-plugin-after-emit', function (data, cb) {

hotMiddleware.publish({ action: 'reload' })

cb()

})

})

// 将 proxyTable 中的请求配置挂在到启动的 express 服务上

Object.keys(proxyTable).forEach(function (context) {

var options = proxyTable[context]

// 如果options的数据类型为string，则表示只设置了url，

// 所以需要将url设置为对象中的 target的值

if (typeof options === 'string') {

options = { target: options }

}

app.use(proxyMiddleware(options.filter || context, options))

})

// 使用 connect-history-api-fallback 匹配资源

// 如果不匹配就可以重定向到指定地址

// https://github.com/bripkens/connect-history-api-fallback

app.use(require('connect-history-api-fallback')())

// 将暂存到内存中的 webpack 编译后的文件挂在到 express 服务上

app.use(devMiddleware)

// 将 Hot-reload 挂在到 express 服务上

app.use(hotMiddleware)

// 拼接 static 文件夹的静态资源路径

var staticPath = path.posix.join(config.dev.assetsPublicPath, config.dev.assetsSubDirectory)

// 静态文件服务

app.use(staticPath, express.static('./static'))

var uri = 'http://localhost:' + port

// 编译成功后打印网址信息

devMiddleware.waitUntilValid(function () {

console.log('> Listening at ' + uri + '\n')

})

module.exports = app.listen(port, function (err) {

if (err) {

console.log(err)

return

}

// 如果配置了自动打开浏览器，且不是测试环境，则自动打开浏览器并跳到我们的开发地址

if (autoOpenBrowser && process.env.NODE\_ENV !== 'testing') {

opn(uri)

}

})

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* **webpack.dev.conf.js** **：开发环境的配置文件**

功能： dev-server.js 中使用了 webpack.dev.conf.js 文件，该文件是开发环境中webpack的配置入口。

// 工具函数集合

var utils = require('./utils')

var webpack = require('webpack')

// 配置文件

var config = require('../config')

// webpack 配置合并插件

var merge = require('webpack-merge')

// webpac基本配置

var baseWebpackConfig = require('./webpack.base.conf')

// 自动生成 html 并且注入到 .html 文件中的插件

// https://github.com/ampedandwired/html-webpack-plugin

var HtmlWebpackPlugin = require('html-webpack-plugin')

// webpack错误信息提示插件

// https://github.com/geowarin/friendly-errors-webpack-plugin

var FriendlyErrorsPlugin = require('friendly-errors-webpack-plugin')

// 将 Hol-reload 热重载的客户端代码添加到 webpack.base.conf 的 对应 entry 中，一起打包

Object.keys(baseWebpackConfig.entry).forEach(function(name) {

baseWebpackConfig.entry[name] = ['./build/dev-client'].concat(baseWebpackConfig.entry[name])

})

module.exports = merge(baseWebpackConfig, {

module: {

// styleLoaders

rules: utils.styleLoaders({ sourceMap: config.dev.cssSourceMap })

},

// 最新的配置为 cheap-module-eval-source-map，虽然 cheap-module-eval-source-map更快，但它的定位不准确

// 所以，换成 eval-source-map

devtool: '#eval-source-map',

plugins: [

// definePlugin 接收字符串插入到代码当中, 所以你需要的话可以写上 JS 的字符串

// 此处，插入适当的环境

// https://webpack.js.org/plugins/define-plugin/

new webpack.DefinePlugin({

'process.env': config.dev.env

}),

// HotModule 插件在页面进行变更的时候只会重绘对应的页面模块，不会重绘整个 html 文件

// https://github.com/glenjamin/webpack-hot-middleware#installation--usage

new webpack.HotModuleReplacementPlugin(),

new webpack.NoEmitOnErrorsPlugin(),

// 将 index.html 作为入口，注入 html 代码后生成 index.html文件

// https://github.com/ampedandwired/html-webpack-plugin

new HtmlWebpackPlugin({

filename: 'index.html',

template: 'index.html',

inject: true

}),

// webpack错误信息提示插件

new FriendlyErrorsPlugin()

]

})

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* **webpack.base.conf.js** **：公共webpack配置文件**

功能： 在 webpack.dev.conf.js 中出现 webpack.base.conf.js，这个文件是开发环境和生产环境，甚至测试环境，这些环境的公共webpack配置。可以说，这个文件相当重要。

// node自带的文件路径工具

var path = require('path')

// 工具函数集合

var utils = require('./utils')

// 配置文件

var config = require('../config')

// 工具函数集合

var vueLoaderConfig = require('./vue-loader.conf')

/\*\*

\* 获得绝对路径

\* @method resolve

\* @param {String} dir 相对于本文件的路径

\* @return {String} 绝对路径

\*/

function resolve(dir) {

return path.join(\_\_dirname, '..', dir)

}

module.exports = {

entry: {

app: './src/main.js'

},

output: {

// 编译输出的静态资源根路径

path: config.build.assetsRoot,

// 编译输出的文件名

filename: '[name].js',

// 正式发布环境下编译输出的上线路径的根路径

publicPath: process.env.NODE\_ENV === 'production' ?

config.build.assetsPublicPath : config.dev.assetsPublicPath

},

resolve: {

// 自动补全的扩展名

extensions: ['.js', '.vue', '.json'],

// 路径别名

alias: {

// 例如 import Vue from 'vue'，会自动到 'vue/dist/vue.common.js'中寻找

'vue$': 'vue/dist/vue.esm.js',

'@': resolve('src'),

}

},

module: {

rules: [{

// 审查 js 和 vue 文件

// https://github.com/MoOx/eslint-loader

test: /\.(js|vue)$/,

loader: 'eslint-loader',

// 表示预先处理

enforce: "pre",

include: [resolve('src'), resolve('test')],

options: {

formatter: require('eslint-friendly-formatter')

}

},

{

// 处理 vue文件

// https://github.com/vuejs/vue-loader

test: /\.vue$/,

loader: 'vue-loader',

options: vueLoaderConfig

},

{

// 编译 js

// https://github.com/babel/babel-loader

test: /\.js$/,

loader: 'babel-loader',

include: [resolve('src'), resolve('test')]

},

{

// 处理图片文件

// https://github.com/webpack-contrib/url-loader

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

loader: 'url-loader',

query: {

limit: 10000,

name: utils.assetsPath('img/[name].[hash:7].[ext]')

}

},

{

// 处理字体文件

test: /\.(woff2?|eot|ttf|otf)(\?.\*)?$/,

loader: 'url-loader',

query: {

limit: 10000,

name: utils.assetsPath('fonts/[name].[hash:7].[ext]')

}

}

]

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* **config/index.js** **：环境配置文件**

功能： 该文件在很多文件中都用到，是主要的配置文件，包含静态文件的路径、是否开启sourceMap等。分为两个部分： dev（开发环境的配置）和 build（生产环境的配置）。

// 详情见文档：https://vuejs-templates.github.io/webpack/env.html

var path = require('path')

module.exports = {

// production 生产环境

build: {

// 构建环境

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

// 构建输出的index.html文件

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

// 构建输出的静态资源路径

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

// 构建输出的二级目录

assetsSubDirectory: 'static',

// 构建发布的根目录，可配置为资源服务器域名或 CDN 域名

assetsPublicPath: '/',

// 是否开启 cssSourceMap

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

// 默认关闭 gzip，因为很多流行的静态资源主机，例如 Surge、Netlify，已经为所有静态资源开启gzip

productionGzip: false,

// 需要使用 gzip 压缩的文件扩展名

productionGzipExtensions: ['js', 'css'],

// Run the build command with an extra argument to

// View the bundle analyzer report after build finishes:

// `npm run build --report`

// Set to `true` or `false` to always turn it on or off

// 运行“build”命令行时，加上一个参数，可以在构建完成后参看包分析报告

// true为开启，false为关闭

bundleAnalyzerReport: process.env.npm\_config\_report

},

// dev 开发环境

dev: {

// 构建环境

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

// 端口号

port: 3333,

// 是否自动打开浏览器

autoOpenBrowser: true,

assetsSubDirectory: 'static',

// 编译发布的根目录，可配置为资源服务器域名或 CDN 域名

assetsPublicPath: '/',

// proxyTable 代理的接口（可跨域）

// 使用方法：https://vuejs-templates.github.io/webpack/proxy.html

proxyTable: {},

// 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.

// 默认情况下，关闭 CSS Sourcemaps，因为使用相对路径会报错。

// CSS-Loader README：https://github.com/webpack/css-loader#sourcemaps

cssSourceMap: false

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* **utils.js** **：工具文件**

功能： utils.js 也是一个被使用频率很高的文件，这个文件包含了三个工具函数：

* 生成静态资源的路径
* 生成 ExtractTextPlugin对象或loader字符串
* 生成 style-loader的配置

// node自带的文件路径工具

var path = require('path')

// 配置文件

var config = require('../config')

// 提取css的插件

// https://github.com/webpack-contrib/extract-text-webpack-plugin

var ExtractTextPlugin = require('extract-text-webpack-plugin')

/\*\*

\* 生成静态资源的路径

\* @method assertsPath

\* @param {String} \_path 相对于静态资源文件夹的文件路径

\* @return {String} 静态资源完整路径

\*/

exports.assetsPath = function (\_path) {

var assetsSubDirectory = process.env.NODE\_ENV === 'production'

? config.build.assetsSubDirectory

: config.dev.assetsSubDirectory

// path.posix.join与path.join一样，不过总是以 posix 兼容的方式交互

return path.posix.join(assetsSubDirectory, \_path)

}

/\*\*

\* 生成处理css的loaders配置

\* @method cssLoaders

\* @param {Object} options 生成配置

\* option = {

\* // 是否开启 sourceMap

\* sourceMap: true,

\* // 是否提取css

\* extract: true

\* }

\* @return {Object} 处理css的loaders配置对象

\*/

exports.cssLoaders = function (options) {

options = options || {}

var cssLoader = {

loader: 'css-loader',

options: {

minimize: process.env.NODE\_ENV === 'production',

sourceMap: options.sourceMap

}

}

/\*\*

\* 生成 ExtractTextPlugin对象或loader字符串

\* @method generateLoaders

\* @param {Array} loaders loader名称数组

\* @return {String|Object} ExtractTextPlugin对象或loader字符串

\*/

function generateLoaders (loader, loaderOptions) {

var loaders = [cssLoader]

if (loader) {

loaders.push({

// 例如，sass?indentedSyntax

// 在?号前加上“-loader”

loader: loader + '-loader',

options: Object.assign({}, loaderOptions, {

sourceMap: options.sourceMap

})

})

}

// extract为true时，提取css

// 生产环境中，默认为true

if (options.extract) {

return ExtractTextPlugin.extract({

use: loaders,

fallback: 'vue-style-loader'

})

} else {

return ['vue-style-loader'].concat(loaders)

}

}

// http://vuejs.github.io/vue-loader/en/configurations/extract-css.html

return {

css: generateLoaders(),

postcss: generateLoaders(),

less: generateLoaders('less'),

sass: generateLoaders('sass', { indentedSyntax: true }),

scss: generateLoaders('sass'),

stylus: generateLoaders('stylus'),

styl: generateLoaders('stylus')

}

}

/\*\*

\* 生成 style-loader的配置

\* style-loader文档：https://github.com/webpack/style-loader

\* @method styleLoaders

\* @param {Object} options 生成配置

\* option = {

\* // 是否开启 sourceMap

\* sourceMap: true,

\* // 是否提取css

\* extract: true

\* }

\* @return {Array} style-loader的配置

\*/

exports.styleLoaders = function (options) {

var output = []

var loaders = exports.cssLoaders(options)

for (var extension in loaders) {

var loader = loaders[extension]

output.push({

test: new RegExp('\\.' + extension + '$'),

use: loader

})

}

return output

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112

#### 4、生产环境

* **build/build.js** **：生产环境的入口文件**

功能：该文件，为构建打包文件，会将源码进行构建（编译、压缩等）后打包。

// 设置当前环境为生产环境

process.env.NODE\_ENV = 'production'

// loading 插件

// https://github.com/sindresorhus/ora

var ora = require('ora')

// 可以在 node 中执行`rm -rf`的工具

// https://github.com/isaacs/rimraf

var rm = require('rimraf')

// node自带的文件路径工具

var path = require('path')

// 在终端输出带颜色的文字

// https://github.com/chalk/chalk

var chalk = require('chalk')

var webpack = require('webpack')

// 配置文件

var config = require('../config')

var webpackConfig = require('./webpack.prod.conf')

// 在终端显示loading效果，并输出提示

var spinner = ora('building for production...')

spinner.start()

// 删除这个文件夹 （递归删除）

rm(path.join(config.build.assetsRoot, config.build.assetsSubDirectory), err => {

if (err) throw err

// 构建

webpack(webpackConfig, function (err, stats) {

// 构建成功

// 停止 loading动画

spinner.stop()

if (err) throw err

process.stdout.write(stats.toString({

colors: true,

modules: false,

children: false,

chunks: false,

chunkModules: false

}) + '\n\n')

// 打印提示

console.log(chalk.cyan(' Build complete.\n'))

console.log(chalk.yellow(

' Tip: built files are meant to be served over an HTTP server.\n' +

' Opening index.html over file:// won\'t work.\n'

))

})

})

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* **webpack.prod.conf.js** **：生产环境的配置文件**

功能： build.js 中使用了 webpack.prod.conf.js 文件，该文件是生产环境中webpack的配置入口。同时，它也依赖于前面提到的 webpack.base.conf.js 、utils.js 和 config/index.js。

// node自带的文件路径工具

var path = require('path')

// 工具函数集合

var utils = require('./utils')

var webpack = require('webpack')

// 配置文件

var config = require('../config')

// webpack 配置合并插件

var merge = require('webpack-merge')

// webpack 基本配置

var baseWebpackConfig = require('./webpack.base.conf')

// webpack 复制文件和文件夹的插件

// https://github.com/kevlened/copy-webpack-plugin

var CopyWebpackPlugin = require('copy-webpack-plugin')

// 自动生成 html 并且注入到 .html 文件中的插件

// https://github.com/ampedandwired/html-webpack-plugin

var HtmlWebpackPlugin = require('html-webpack-plugin')

// 提取css的插件

// https://github.com/webpack-contrib/extract-text-webpack-plugin

var ExtractTextPlugin = require('extract-text-webpack-plugin')

// webpack 优化压缩和优化 css 的插件

// https://github.com/NMFR/optimize-css-assets-webpack-plugin

var OptimizeCSSPlugin = require('optimize-css-assets-webpack-plugin')

// 如果当前环境为测试环境，则使用测试环境

// 否则，使用生产环境

var env = process.env.NODE\_ENV === 'testing'

? require('../config/test.env')

: config.build.env

var webpackConfig = merge(baseWebpackConfig, {

module: {

// styleLoaders

rules: utils.styleLoaders({

sourceMap: config.build.productionSourceMap,

extract: true

})

},

// 是否开启 sourceMap

devtool: config.build.productionSourceMap ? '#source-map' : false,

output: {

// 编译输出的静态资源根路径

path: config.build.assetsRoot,

// 编译输出的文件名

filename: utils.assetsPath('js/[name].[chunkhash].js'),

// 没有指定输出名的文件输出的文件名

chunkFilename: utils.assetsPath('js/[id].[chunkhash].js')

},

plugins: [

// definePlugin 接收字符串插入到代码当中, 所以你需要的话可以写上 JS 的字符串

// 此处，插入适当的环境

// http://vuejs.github.io/vue-loader/en/workflow/production.html

new webpack.DefinePlugin({

'process.env': env

}),

// 压缩 js

new webpack.optimize.UglifyJsPlugin({

compress: {

warnings: false

},

sourceMap: true

}),

// 提取 css

new ExtractTextPlugin({

filename: utils.assetsPath('css/[name].[contenthash].css')

}),

// 压缩提取出来的 css

// 可以删除来自不同组件的冗余代码

// Compress extracted CSS. We are using this plugin so that possible

// duplicated CSS from different components can be deduped.

new OptimizeCSSPlugin(),

// 将 index.html 作为入口，注入 html 代码后生成 index.html文件

// https://github.com/ampedandwired/html-webpack-plugin

new HtmlWebpackPlugin({

filename: process.env.NODE\_ENV === 'testing'

? 'index.html'

: config.build.index,

template: 'index.html',

inject: true,

minify: {

removeComments: true,

collapseWhitespace: true,

removeAttributeQuotes: true

// 更多选项 https://github.com/kangax/html-minifier#options-quick-reference

},

// 必须通过 CommonsChunkPlugin一致地处理多个 chunks

chunksSortMode: 'dependency'

}),

// 分割公共 js 到独立的文件

// https://webpack.js.org/guides/code-splitting-libraries/#commonschunkplugin

new webpack.optimize.CommonsChunkPlugin({

name: 'vendor',

minChunks: function (module, count) {

// node\_modules中的任何所需模块都提取到vendor

return (

module.resource &&

/\.js$/.test(module.resource) &&

module.resource.indexOf(

path.join(\_\_dirname, '../node\_modules')

) === 0

)

}

}),

// 将webpack runtime 和模块清单 提取到独立的文件，以防止当 app包更新时导致公共 jsd hash也更新

// extract webpack runtime and module manifest to its own file in order to

// prevent vendor hash from being updated whenever app bundle is updated

new webpack.optimize.CommonsChunkPlugin({

name: 'manifest',

chunks: ['vendor']

}),

// 复制静态资源

// https://github.com/kevlened/copy-webpack-plugin

new CopyWebpackPlugin([

{

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

to: config.build.assetsSubDirectory,

ignore: ['.\*']

}

])

]

})

// 开启 gzip 的情况时，给 webpack plugins添加 compression-webpack-plugin 插件

if (config.build.productionGzip) {

// webpack 压缩插件

// https://github.com/webpack-contrib/compression-webpack-plugin

var CompressionWebpackPlugin = require('compression-webpack-plugin')

// 向webpackconfig.plugins中加入下方的插件

webpackConfig.plugins.push(

new CompressionWebpackPlugin({

asset: '[path].gz[query]',

algorithm: 'gzip',

test: new RegExp(

'\\.(' +

config.build.productionGzipExtensions.join('|') +

')$'

),

threshold: 10240,

minRatio: 0.8

})

)

}

// 开启包分析的情况时， 给 webpack plugins添加 webpack-bundle-analyzer 插件

if (config.build.bundleAnalyzerReport) {

// https://github.com/th0r/webpack-bundle-analyzer

var BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin

webpackConfig.plugins.push(new BundleAnalyzerPlugin())

}

module.exports = webpackConfig