一、安装vue-cli，相信你既然会用到vue-cli，自然node环境是OK的，直接命令行下安装

npm install -g vue-cli

二、使用vue-cli创建vue项目

用法： vue init <template-name> <project-name>

template-name:

. webpack

. webpack-simple // 一个简单webpack+vue-loader的模板，不包含其他功能。

. browserify // 一个全面的Browserify+vueify 的模板，功能包括热加载，linting,单元检测。

. browserify-simple // 一个简单Browserify+vueify的模板，不包含其他功能。

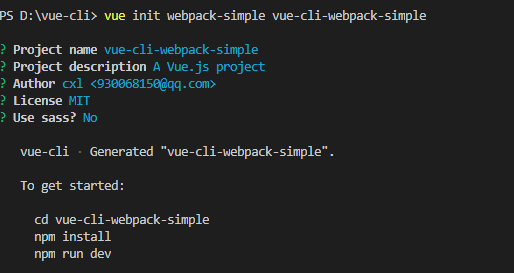
. pwa // 基于webpack模板的vue-cli的PWA模板

. simple // 一个最简单的单页应用模板

常用的就是webpack了，模板之间的不同，自己体验  
示例：

vue init webpack my-project

执行指令后，会让用户输入几个基本的选项，如图所示

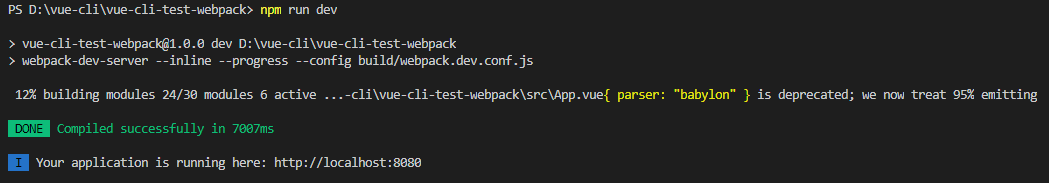


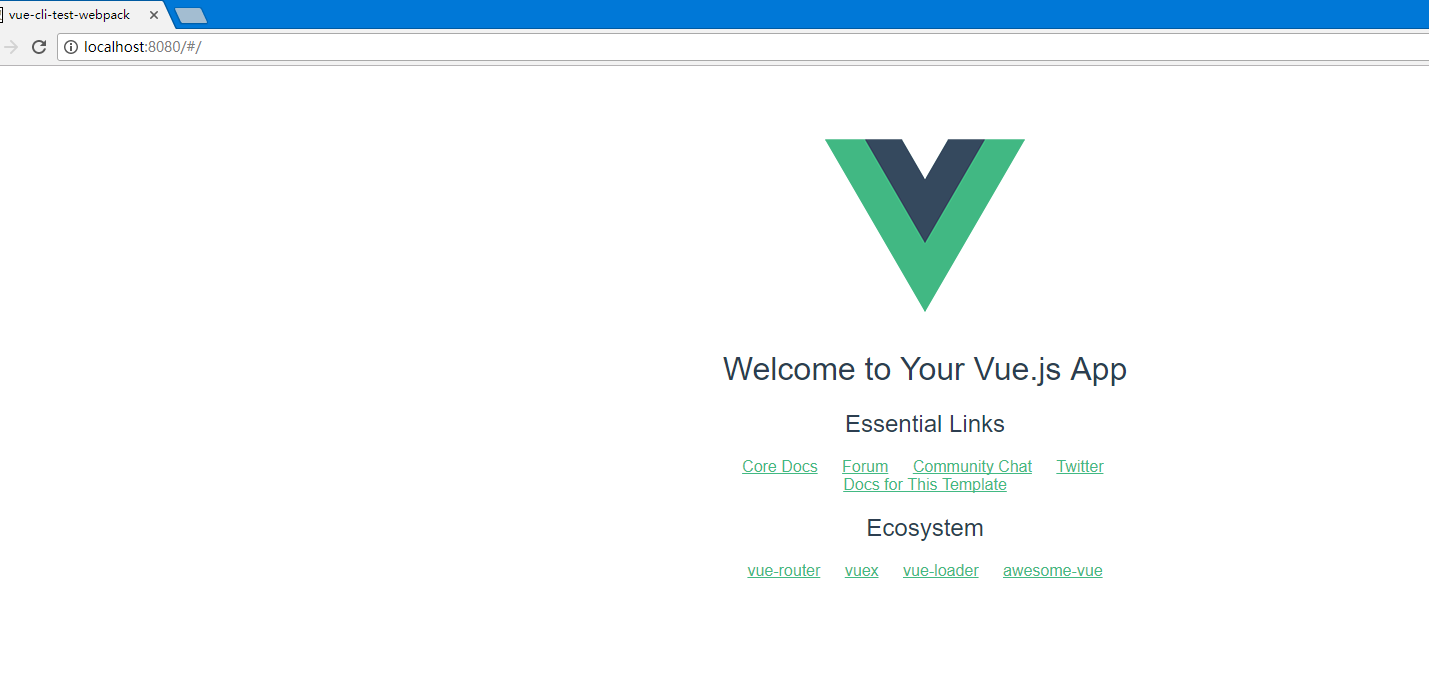
**需要注意的是项目的名称不能大写，不然会报错。**

* Project name :项目名称 ，如果不需要更改直接回车就可以了。注意：这里不能使用大写。
* Project description:项目描述，默认为A Vue.js project,直接回车，不用编写。
* Author：作者，如果你有配置git，他会读取.ssh文件中的user。
* Install vue-router? 是否安装vue的路由插件，Y代表安装，N无需安装，下面的命令也是一样的。
* Use ESLint to lint your code? 是否用ESLint来限制你的代码错误和风格
* setup unit tests with Karma + Mocha? 是否需要安装单元测试工具Karma+Mocha。
* Setup e2e tests with Nightwatch?是否安装e2e来进行用户行为模拟测试。
* Should we run npm install for you after the project has been created?(recommended)npm  
  询问你使用npm安装还是yarn安装包依赖，我这里选择的是npm，yarn更快更好，使用yarn之前确保你的电脑已经安装yarn。

根据提示，待模板加载完成之后,执行下面两条命令

cd my-project

npm run dev // dev代表下图框选的内容



## 三、文件目录结构

本文主要分析开发（dev）和构建（build）两个过程涉及到的文件，故下面文件结构仅列出相应的内容。

|-- build // 项目构建(webpack)相关代码

| |-- build.js // 生产环境构建代码

| |-- check-version.js // 检查node、npm等版本

| |-- utils.js // 构建工具相关

| |-- vue-loader.conf.js // webpack loader配置

| |-- webpack.base.conf.js // webpack基础配置

| |-- webpack.dev.conf.js // webpack开发环境配置,构建开发本地服务器

| |-- webpack.prod.conf.js // webpack生产环境配置

|-- config // 项目开发环境配置

| |-- dev.env.js // 开发环境变量

| |-- index.js // 项目一些配置变量

| |-- prod.env.js // 生产环境变量

| |-- test.env.js // 测试脚本的配置

|-- src // 源码目录

| |-- components // vue所有组件

| |-- router // vue的路由管理

| |-- App.vue // 页面入口文件

| |-- main.js // 程序入口文件，加载各种公共组件

|-- static // 静态文件，比如一些图片，json数据等

|-- test // 测试文件

| |-- e2e // e2e 测试

| |-- unit // 单元测试

|-- .babelrc // ES6语法编译配置

|-- .editorconfig // 定义代码格式

|-- .eslintignore // eslint检测代码忽略的文件（夹）

|-- .eslintrc.js // 定义eslint的plugins,extends,rules

|-- .gitignore // git上传需要忽略的文件格式

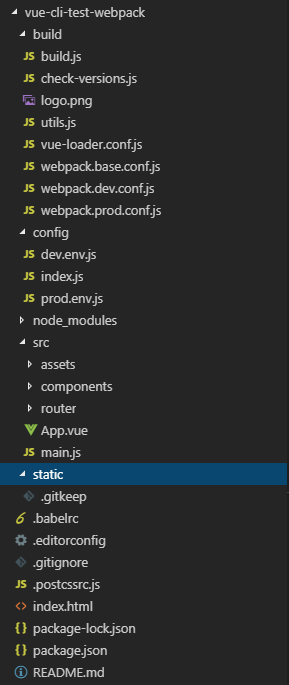
|-- .postcsssrc // postcss配置文件

|-- README.md // 项目说明，markdown文档

|-- index.html // 访问的页面

|-- package.json // 项目基本信息,包依赖信息等

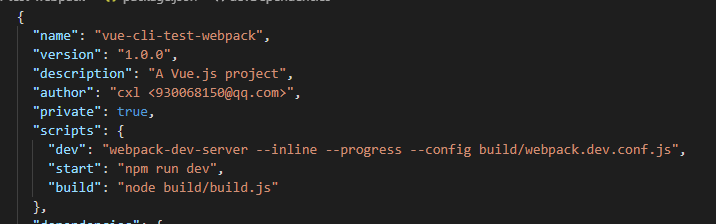
如图所示：



下边是具体文件的具体分析

### 1. package.json文件

package.json文件是项目的配置文件，定义了项目的基本信息以及项目的相关包依赖，npm运行命令等



**scripts** 里定义的是一些比较长的命令，用node去执行一段命令，比如

npm run dev

其实就是执行

webpack-dev-server --inline --progress --config build/webpack.dev.conf.js

这句话的意思是利用 webpack-dev-server 读取 webpack.dev.conf.js 信息并启动一个本地服务器。

### 2. dependencies VS devDependencies

简单的来说

dependencies 是运行时依赖（生产环境） npm install --save \*\*(package name)

devDependencies 是开发时的依赖（开发环境） npm install --save-dev \*\*(package name)

### 3. 基础配置文件 webpack.base.conf.js

基础的 webpack 配置文件主要根据模式定义了入口出口，以及处理 vue, babel等的各种模块，是最为基础的部分。其他模式的配置文件以此为基础通过 webpack-merge 合并。

'use strict'

const path = require('path')

const utils = require('./utils')

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

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

// 获取绝对路径

function resolve (dir) {

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

}

<!-- 定义一下代码检测的规则 -->

const createLintingRule = () => ({

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

loader: 'eslint-loader',

enforce: 'pre',

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

options: {

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

emitWarning: !config.dev.showEslintErrorsInOverlay

}

})

module.exports = {

// 基础上下文

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

// webpack的入口文件

entry: {

app: './src/main.js'

},

// webpack的输出文件

output: {

path: config.build.assetsRoot,

filename: '[name].js',

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

? config.build.assetsPublicPath

: config.dev.assetsPublicPath

},

/\*\*

\* 当webpack试图去加载模块的时候，它默认是查找以 .js 结尾的文件的，

\* 它并不知道 .vue 结尾的文件是什么鬼玩意儿，

\* 所以我们要在配置文件中告诉webpack，

\* 遇到 .vue 结尾的也要去加载，

\* 添加 resolve 配置项，如下：

\*/

resolve: {

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

alias: { // 创建别名

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

'@': resolve('src'), // 如 '@/components/HelloWorld'

}

},

// 不同类型模块的处理规则 就是用不同的loader处理不同的文件

module: {

rules: [

...(config.dev.useEslint ? [createLintingRule()] : []),

{// 对所有.vue文件使用vue-loader进行编译

test: /\.vue$/,

loader: 'vue-loader',

options: vueLoaderConfig

},

{// 对src和test文件夹下的.js文件使用babel-loader将es6+的代码转成es5

test: /\.js$/,

loader: 'babel-loader',

include: [resolve('src'), resolve('test'), resolve('node\_modules/webpack-dev-server/client')]

},

{// 对图片资源文件使用url-loader

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

loader: 'url-loader',

options: {

// 小于10K的图片转成base64编码的dataURL字符串写到代码中

limit: 10000,

// 其他的图片转移到静态资源文件夹

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

}

},

{// 对多媒体资源文件使用url-loader

test: /\.(mp4|webm|ogg|mp3|wav|flac|aac)(\?.\*)?$/,

loader: 'url-loader',

options: {

// 小于10K的资源转成base64编码的dataURL字符串写到代码中

limit: 10000,

// 其他的资源转移到静态资源文件夹

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

}

},

{// 对字体资源文件使用url-loader

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

loader: 'url-loader',

options: {

limit: 10000,

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

}

}

]

},

node: {

// prevent webpack from injecting useless setImmediate polyfill because Vue

// source contains it (although only uses it if it's native).

setImmediate: false,

// prevent webpack from injecting mocks to Node native modules

// that does not make sense for the client

dgram: 'empty',

fs: 'empty',

net: 'empty',

tls: 'empty',

child\_process: 'empty'

}

}

### 4. 开发环境配置文件 webpack.dev.conf.js

'use strict'

const utils = require('./utils')

const webpack = require('webpack')

const config = require('../config') // 基本配置的参数

const merge = require('webpack-merge') // webpack-merge是一个可以合并数组和对象的插件

const path = require('path')

const baseWebpackConfig = require('./webpack.base.conf') // webpack基本配置文件（开发和生产环境公用部分）

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

// html-webpack-plugin用于将webpack编译打包后的产品文件注入到html模板中

// 即在index.html里面加上<link>和<script>标签引用webpack打包后的文件

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

// friendly-errors-webpack-plugin用于更友好地输出webpack的警告、错误等信息

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

const portfinder = require('portfinder') // 自动检索下一个可用端口

const HOST = process.env.HOST

const PORT = process.env.PORT && Number(process.env.PORT) ) // 读取系统环境变量的port

// 合并baseWebpackConfig配置

const devWebpackConfig = merge(baseWebpackConfig, {

module: {

// 对一些独立的css文件以及它的预处理文件做一个编译

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

},

// cheap-module-eval-source-map is faster for development

devtool: config.dev.devtool,

// these devServer options should be customized in /config/index.js

devServer: { // webpack-dev-server服务器配置

clientLogLevel: 'warning', // console 控制台显示的消息，可能的值有 none, error, warning 或者 info

historyApiFallback: {

rewrites: [

{ from: /.\*/, to: path.posix.join(config.dev.assetsPublicPath, 'index.html') },

],

},

hot: true, // 开启热模块加载

contentBase: false, // since we use CopyWebpackPlugin.

compress: true,

host: HOST || config.dev.host, // process.env 优先

port: PORT || config.dev.port, // process.env 优先

open: config.dev.autoOpenBrowser,

overlay: config.dev.errorOverlay

? { warnings: false, errors: true }

: false,

publicPath: config.dev.assetsPublicPath,

proxy: config.dev.proxyTable, // 代理设置

quiet: true, // necessary for FriendlyErrorsPlugin

watchOptions: { // 启用 Watch 模式。这意味着在初始构建之后，webpack 将继续监听任何已解析文件的更改

poll: config.dev.poll, // 通过传递 true 开启 polling，或者指定毫秒为单位进行轮询。默认为false

}

},

plugins: [

new webpack.DefinePlugin({

'process.env': require('../config/dev.env')

}),

/\*模块热替换它允许在运行时更新各种模块，而无需进行完全刷新\*/

new webpack.HotModuleReplacementPlugin(),

new webpack.NamedModulesPlugin(), // HMR shows correct file names in console on update.

new webpack.NoEmitOnErrorsPlugin(),// 跳过编译时出错的代码并记录下来，主要作用是使编译后运行时的包不出错

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

new HtmlWebpackPlugin({

// 指定编译后生成的html文件名

filename: 'index.html',

// 需要处理的模板

template: 'index.html',

// 打包过程中输出的js、css的路径添加到html文件中

// css文件插入到head中

// js文件插入到body中，可能的选项有 true, 'head', 'body', false

inject: true

}),

// copy custom static assets

new CopyWebpackPlugin([

{

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

to: config.dev.assetsSubDirectory,

ignore: ['.\*']

}

])

]

})

module.exports = new Promise((resolve, reject) => {

portfinder.basePort = process.env.PORT || config.dev.port // 获取当前设定的端口

portfinder.getPort((err, port) => {

if (err) {

reject(err)

} else {

// publish the new Port, necessary for e2e tests 发布新的端口，对于e2e测试

process.env.PORT = port

// add port to devServer config

devWebpackConfig.devServer.port = port

// Add FriendlyErrorsPlugin

devWebpackConfig.plugins.push(new FriendlyErrorsPlugin({

compilationSuccessInfo: {

messages: [`Your application is running here: http://${devWebpackConfig.devServer.host}:${port}`],

},

onErrors: config.dev.notifyOnErrors

? utils.createNotifierCallback()

: undefined

}))

resolve(devWebpackConfig)

}

})

})

### 5. 生产模式配置文件 webpack.prod.conf.js

'use strict'

const path = require('path')

const utils = require('./utils')

const webpack = require('webpack')

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

const merge = require('webpack-merge')

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

// copy-webpack-plugin，用于将static中的静态文件复制到产品文件夹dist

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

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

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

// optimize-css-assets-webpack-plugin，用于优化和最小化css资源

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

// uglifyJs 混淆js插件

const UglifyJsPlugin = require('uglifyjs-webpack-plugin')

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

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

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

const webpackConfig = merge(baseWebpackConfig, {

module: {

// 样式文件的处理规则，对css/sass/scss等不同内容使用相应的styleLoaders

// 由utils配置出各种类型的预处理语言所需要使用的loader，例如sass需要使用sass-loader

rules: utils.styleLoaders({

sourceMap: config.build.productionSourceMap,

extract: true,

usePostCSS: true

})

},

devtool: config.build.productionSourceMap ? config.build.devtool : false,

// webpack输出路径和命名规则

output: {

path: config.build.assetsRoot,

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

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

},

plugins: [

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

new webpack.DefinePlugin({

'process.env': env

}),

// 丑化压缩JS代码

new UglifyJsPlugin({

uglifyOptions: {

compress: {

warnings: false

}

},

sourceMap: config.build.productionSourceMap,

parallel: true

}),

// extract css into its own file

// 将css提取到单独的文件

new ExtractTextPlugin({

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

// Setting the following option to `false` will not extract CSS from codesplit chunks.

// Their CSS will instead be inserted dynamically with style-loader when the codesplit chunk has been loaded by webpack.

// It's currently set to `true` because we are seeing that sourcemaps are included in the codesplit bundle as well when it's `false`,

// increasing file size: https://github.com/vuejs-templates/webpack/issues/1110

allChunks: true,

}),

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

// duplicated CSS from different components can be deduped.

// 优化、最小化css代码，如果只简单使用extract-text-plugin可能会造成css重复

// 具体原因可以看npm上面optimize-css-assets-webpack-plugin的介绍

new OptimizeCSSPlugin({

cssProcessorOptions: config.build.productionSourceMap

? { safe: true, map: { inline: false } }

: { safe: true }

}),

// generate dist index.html with correct asset hash for caching.

// you can customize output by editing /index.html

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

// 将产品文件的引用注入到index.html

new HtmlWebpackPlugin({

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

? 'index.html'

: config.build.index,

template: 'index.html',

inject: true,

minify: {

// 删除index.html中的注释

removeComments: true,

// 删除index.html中的空格

collapseWhitespace: true,

// 删除各种html标签属性值的双引号

removeAttributeQuotes: true

// more options:

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

},

// necessary to consistently work with multiple chunks via CommonsChunkPlugin

// 注入依赖的时候按照依赖先后顺序进行注入，比如，需要先注入vendor.js，再注入app.js

chunksSortMode: 'dependency'

}),

// keep module.id stable when vendor modules does not change

new webpack.HashedModuleIdsPlugin(),

// enable scope hoisting

new webpack.optimize.ModuleConcatenationPlugin(),

// split vendor js into its own file

// 将所有从node\_modules中引入的js提取到vendor.js，即抽取库文件

new webpack.optimize.CommonsChunkPlugin({

name: 'vendor',

minChunks (module) {

// any required modules inside node\_modules are extracted to vendor

return (

module.resource &&

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

module.resource.indexOf(

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

) === 0

)

}

}),

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

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

// 从vendor中提取出manifest，原因如上

new webpack.optimize.CommonsChunkPlugin({

name: 'manifest',

minChunks: Infinity

}),

// This instance extracts shared chunks from code splitted chunks and bundles them

// in a separate chunk, similar to the vendor chunk

// see: https://webpack.js.org/plugins/commons-chunk-plugin/#extra-async-commons-chunk

new webpack.optimize.CommonsChunkPlugin({

name: 'app',

async: 'vendor-async',

children: true,

minChunks: 3

}),

// copy custom static assets

// 将static文件夹里面的静态资源复制到dist/static

new CopyWebpackPlugin([

{

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

to: config.build.assetsSubDirectory,

ignore: ['.\*']

}

])

]

})

// 如果开启了产品gzip压缩，则利用插件将构建后的产品文件进行压缩

if (config.build.productionGzip) {

// 一个用于压缩的webpack插件

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

webpackConfig.plugins.push(

new CompressionWebpackPlugin({

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

// 压缩算法

algorithm: 'gzip',

test: new RegExp(

'\\.(' +

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

')$'

),

threshold: 10240,

minRatio: 0.8

})

)

}

// 如果启动了report，则通过插件给出webpack构建打包后的产品文件分析报告

if (config.build.bundleAnalyzerReport) {

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

webpackConfig.plugins.push(new BundleAnalyzerPlugin())

}

module.exports = webpackConfig

### 6. build.js 编译入口

'use strict'

require('./check-versions')()

process.env.NODE\_ENV = 'production'

// ora，一个可以在终端显示spinner的插件

const ora = require('ora')

// rm，用于删除文件或文件夹的插件

const rm = require('rimraf')

const path = require('path')

// chalk，用于在控制台输出带颜色字体的插件

const chalk = require('chalk')

const webpack = require('webpack')

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

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

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

spinner.start() // 开启loading动画

// 首先将整个dist文件夹以及里面的内容删除，以免遗留旧的没用的文件

// 删除完成后才开始webpack构建打包

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

if (err) throw err

// 执行webpack构建打包，完成之后在终端输出构建完成的相关信息或者输出报错信息并退出程序

webpack(webpackConfig, (err, stats) => {

spinner.stop()

if (err) throw err

process.stdout.write(stats.toString({

colors: true,

modules: false,

children: false, // If you are using ts-loader, setting this to true will make TypeScript errors show up during build.

chunks: false,

chunkModules: false

}) + '\n\n')

if (stats.hasErrors()) {

console.log(chalk.red(' Build failed with errors.\n'))

process.exit(1)

}

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'

))

})

})

### 7. 实用代码段 utils.js

'use strict'

const path = require('path')

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

// extract-text-webpack-plugin可以提取bundle中的特定文本，将提取后的文本单独存放到另外的文件

// 这里用来提取css样式

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

const packageConfig = require('../package.json')

// 资源文件的存放路径

exports.assetsPath = function (\_path) {

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

? config.build.assetsSubDirectory

: config.dev.assetsSubDirectory

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

}

// 生成css、sass、scss等各种用来编写样式的语言所对应的loader配置

exports.cssLoaders = function (options) {

options = options || {}

// css-loader配置

const cssLoader = {

loader: 'css-loader',

options: {

// 是否使用source-map

sourceMap: options.sourceMap

}

}

const postcssLoader = {

loader: 'postcss-loader',

options: {

sourceMap: options.sourceMap

}

}

// generate loader string to be used with extract text plugin

// 生成各种loader配置，并且配置了extract-text-pulgin

function generateLoaders (loader, loaderOptions) {

const loaders = options.usePostCSS ? [cssLoader, postcssLoader] : [cssLoader]

// 例如generateLoaders('less')，这里就会push一个less-loader

// less-loader先将less编译成css，然后再由css-loader去处理css

// 其他sass、scss等语言也是一样的过程

if (loader) {

loaders.push({

loader: loader + '-loader',

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

sourceMap: options.sourceMap

})

})

}

// Extract CSS when that option is specified

// (which is the case during production build)

if (options.extract) {

// 配置extract-text-plugin提取样式

return ExtractTextPlugin.extract({

use: loaders,

fallback: 'vue-style-loader'

})

} else {

// 无需提取样式则简单使用vue-style-loader配合各种样式loader去处理<style>里面的样式

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

}

}

// https://vue-loader.vuejs.org/en/configurations/extract-css.html

// 得到各种不同处理样式的语言所对应的loader

return {

css: generateLoaders(),

postcss: generateLoaders(),

less: generateLoaders('less'),

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

scss: generateLoaders('sass'),

stylus: generateLoaders('stylus'),

styl: generateLoaders('stylus')

}

}

// Generate loaders for standalone style files (outside of .vue)

// 生成处理单独的.css、.sass、.scss等样式文件的规则

exports.styleLoaders = function (options) {

const output = []

const loaders = exports.cssLoaders(options)

for (const extension in loaders) {

const loader = loaders[extension]

output.push({

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

use: loader

})

}

return output

}

exports.createNotifierCallback = () => {

const notifier = require('node-notifier')

return (severity, errors) => {

if (severity !== 'error') return

const error = errors[0]

const filename = error.file && error.file.split('!').pop()

notifier.notify({

title: packageConfig.name,

message: severity + ': ' + error.name,

subtitle: filename || '',

icon: path.join(\_\_dirname, 'logo.png')

})

}

}

### 8. babel配置文件.babelrc

{ //设定转码规则

"presets": [

["env", {

"modules": false,

//对BABEL\_ENV或者NODE\_ENV指定的不同的环境变量，进行不同的编译操作

"targets": {

"browsers": ["> 1%", "last 2 versions", "not ie <= 8"]

}

}],

"stage-2"

],

//转码用的插件

"plugins": ["transform-vue-jsx", "transform-runtime"]

}

### 9 .编码规范.editorconfig （自定义）

root = true

[\*] // 对所有文件应用下面的规则

charset = utf-8 // 编码规则用utf-8

indent\_style = space // 缩进用空格

indent\_size = 2 // 缩进数量为2个空格

end\_of\_line = lf // 换行符格式

insert\_final\_newline = true // 是否在文件的最后插入一个空行

trim\_trailing\_whitespace = true // 是否删除行尾的空格

### 10 .src/app.vue文件解读

<template>

<div id="app">

<img src="./assets/logo.png">

<router-view></router-view>

</div>

</template>

<script>

export default {

name: 'app'

}

</script>

<style>

#app {

font-family: 'Avenir', Helvetica, Arial, sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

text-align: center;

color: #2c3e50;

margin-top: 60px;

}

</style>

<template></template> 标签包裹的内容：这是模板的HTMLDom结构

<script></script> 标签包括的js内容：你可以在这里写一些页面的js的逻辑代码。

<style></style> 标签包裹的css内容：页面需要的CSS样式。

### 11. src/router/index.js 路由文件

import Vue from 'vue'

import Router from 'vue-router'

import Hello from '@/components/Hello'

Vue.use(Router)

export default new Router({

routes: [//配置路由

{

path: '/', //访问路径

name: 'Hello', //路由名称

component: Hello //路由需要的组件（驼峰式命名）

}

]

网上找到一张挺有意思的图：

Vue-cli项目图

