vue-cli脚手架中webpack配置基础文件详解  
 一、前言  
 原文：https://segmentfault.com/a/1190000014804826  
  
 vue-cli是构建vue单页应用的脚手架，输入一串指定的命令行从而自动生成vue.js+wepack的项目模板。这其中webpack发挥了很大的作用，它使得我们的代码模块化，引入一些插件帮我们完善功能可以将文件打包压缩，图片转base64等。后期对项目的配置使得我们对于脚手架自动生成的代码的理解更为重要，接下来我将基于webpack3.6.0版本结合文档将文件各个击破，纯干料。  
   
 二、主体结构  
 ├─build  
 ├─config  
 ├─dist  
 ├─node\_modules  
 ├─src  
 │ ├─assets  
 │ ├─components  
 │ ├─router  
 │ ├─App.vue  
 │ ├─main.js  
 ├─static  
 ├─.babelrc  
 ├─.editorconfig  
 ├─.gitignore  
 ├─.postcssrc.js  
 ├─index.html  
 ├─package-lock.json  
 ├─package.json  
 └─README.md

1、 package.json  
 项目作为一个大家庭，每个文件都各司其职。package.json来制定名单，需要哪些npm包来参与到项目中来，npm install命令根据这个配置文件增减来管理本地的安装包。  
  
 {  
 //从name到private都是package的配置信息，也就是我们在脚手架搭建中输入的项目描述  
 "name": "shop",//项目名称：不能以.(点)或者\_（下划线）开头，不能包含大写字母，具有明确的的含义与现有项目名字不重复  
 "version": "1.0.0",//项目版本号：遵循“大版本.次要版本.小版本”  
 "description": "A Vue.js project",//项目描述  
 "author": "qietuniu",//作者名字  
 "private": true,//是否私有  
 //scripts中的子项即是我们在控制台运行的脚本的缩写  
 "scripts": {  
 //①webpack-dev-server:启动了http服务器，实现实时编译;  
 //inline模式会在webpack.config.js入口配置中新增webpack-dev-server/client?http://localhost:8080/的入口,使得我们访问路径为localhost:8080/index.html（相应的还有另外一种模式Iframe）;  
 //progress:显示打包的进度  
 "dev": "webpack-dev-server --inline --progress --config build/webpack.dev.conf.js",  
 "start": "npm run dev",//与npm run dev相同，直接运行开发环境  
 "build": "node build/build.js"//使用node运行build文件  
 },  
 //②dependencies(项目依赖库):在安装时使用--save则写入到dependencies  
 "dependencies": {  
 "vue": "^2.5.2",//vue.js  
 "vue-router": "^3.0.1"//vue的路由插件  
 },  
 //和devDependencies（开发依赖库）：在安装时使用--save-dev将写入到devDependencies  
 "devDependencies": {  
 "autoprefixer": "^7.1.2",//autoprefixer作为postcss插件用来解析CSS补充前缀，例如 display: flex会补充为display:-webkit-box;display: -webkit-flex;display: -ms-flexbox;display: flex。  
 //babel:以下几个babel开头的都是针对es6解析的插件。用最新标准编写的 JavaScript 代码向下编译成可以在今天随处可用的版本  
 "babel-core": "^6.22.1",//babel的核心，把 js 代码分析成 ast ，方便各个插件分析语法进行相应的处理。  
 "babel-helper-vue-jsx-merge-props": "^2.0.3",//预制babel-template函数，提供给vue,jsx等使用  
 "babel-loader": "^7.1.1",//使项目运行使用Babel和webpack来传输js文件，使用babel-core提供的api进行转译  
 "babel-plugin-syntax-jsx": "^6.18.0",//支持jsx  
 "babel-plugin-transform-runtime": "^6.22.0",//避免编译输出中的重复，直接编译到build环境中  
 "babel-plugin-transform-vue-jsx": "^3.5.0",//babel转译过程中使用到的插件，避免重复  
 "babel-preset-env": "^1.3.2",//转为es5，transform阶段使用到的插件之一  
 "babel-preset-stage-2": "^6.22.0",//ECMAScript第二阶段的规范  
 "chalk": "^2.0.1",//用来在命令行输出不同颜色文字  
 "copy-webpack-plugin": "^4.0.1",//拷贝资源和文件  
 "css-loader": "^0.28.0",//webpack先用css-loader加载器去解析后缀为css的文件，再使用style-loader生成一个内容为最终解析完的css代码的style标签，放到head标签里  
 "extract-text-webpack-plugin": "^3.0.0",//将一个以上的包里面的文本提取到单独文件中  
 "file-loader": "^1.1.4",//③打包压缩文件，与url-loader用法类似  
 "friendly-errors-webpack-plugin": "^1.6.1",//识别某些类别的WebPACK错误和清理，聚合和优先排序，以提供更好的开发经验  
 "html-webpack-plugin": "^2.30.1",//简化了HTML文件的创建，引入了外部资源，创建html的入口文件，可通过此项进行多页面的配置  
 "node-notifier": "^5.1.2",//支持使用node发送跨平台的本地通知  
 "optimize-css-assets-webpack-plugin": "^3.2.0",//压缩提取出的css，并解决ExtractTextPlugin分离出的js重复问题(多个文件引入同一css文件)  
 "ora": "^1.2.0",//加载（loading）的插件  
 "portfinder": "^1.0.13",//查看进程端口  
 "postcss-import": "^11.0.0",//可以消耗本地文件、节点模块或web\_modules  
 "postcss-loader": "^2.0.8",//用来兼容css的插件  
 "postcss-url": "^7.2.1",//URL上重新定位、内联或复制  
 "rimraf": "^2.6.0",//节点的UNIX命令RM—RF,强制删除文件或者目录的命令  
 "semver": "^5.3.0",//用来对特定的版本号做判断的  
 "shelljs": "^0.7.6",//使用它来消除shell脚本在UNIX上的依赖性，同时仍然保留其熟悉和强大的命令，即可执行Unix系统命令  
 "uglifyjs-webpack-plugin": "^1.1.1",//压缩js文件  
 "url-loader": "^0.5.8",//压缩文件，可将图片转化为base64  
 "vue-loader": "^13.3.0",//VUE单文件组件的WebPACK加载器  
 "vue-style-loader": "^3.0.1",//类似于样式加载程序，您可以在CSS加载器之后将其链接，以将CSS动态地注入到文档中作为样式标签  
 "vue-template-compiler": "^2.5.2",//这个包可以用来预编译VUE模板到渲染函数，以避免运行时编译开销和CSP限制  
 "webpack": "^3.6.0",//打包工具  
 "webpack-bundle-analyzer": "^2.9.0",//可视化webpack输出文件的大小  
 "webpack-dev-server": "^2.9.1",//提供一个提供实时重载的开发服务器  
 "webpack-merge": "^4.1.0"//它将数组和合并对象创建一个新对象。如果遇到函数，它将执行它们，通过算法运行结果，然后再次将返回的值封装在函数中  
 },  
 //engines是引擎，指定node和npm版本  
 "engines": {  
 "node": ">= 6.0.0",  
 "npm": ">= 3.0.0"  
 },  
 //限制了浏览器或者客户端需要什么版本才可运行  
 "browserslist": [  
 "> 1%",  
 "last 2 versions",  
 "not ie <= 8"  
 ]  
 }  
 注释：  
 ①、点这里→webpack运行时的配置文档传送门

https://webpack.js.org/configuration/dev-server/#devserver  
  
 ②、devDependencies和dependencies的区别： devDependencies里面的插件只用于开发环境，不用于生产环境，即辅助作用，打包的时候需要，打包完成就不需要了。而dependencies是需要发布到生产环境的，自始至终都在。比如wepack等只是在开发中使用的包就写入到devDependencies，而像vue这种项目全程依赖的包要写入到dependencies  
 点这里→更多安装包文档搜索页传送门  
 https://www.npmjs.com/

③、file-loader和url-loader的区别：以图片为例，file-loader可对图片进行压缩，但是还是通过文件路径进行引入，当http请求增多时会降低页面性能，而url-loader通过设定limit参数，小于limit字节的图片会被转成base64的文件，大于limit字节的将进行图片压缩的操作。总而言之，url-loader是file-loader的上层封装。  
  
 2、.postcssrc.js  
 .postcssrc.js文件其实是postcss-loader包的一个配置，在webpack的旧版本可以直接在webpack.config.js中配置，现版本中postcss的文档示例独立出.postcssrc.js，里面写进去需要使用到的插件  
  
 module.exports = {  
 "plugins": {  
 "postcss-import": {},//①  
 "postcss-url": {},//②  
 "autoprefixer": {}//③  
 }  
 }

注释：  
 ①、点这里→postcss-import文档传送门

https://www.npmjs.com/package/postcss-import  
 ②、点这里→postcss-url文档传送门

https://www.npmjs.com/package/postcss-url  
 ③、点这里→autoprefixer文档传送门

https://www.npmjs.com/package/autoprefixer  
  
 3、 .babelrc  
 该文件是es6解析的一个配置  
  
 {  
 //制定转码的规则  
 "presets": [  
 //env是使用babel-preset-env插件将js进行转码成es5，并且设置不转码的AMD,COMMONJS的模块文件，制定浏览器的兼容  
 ["env", {  
 "modules": false,  
 "targets": {  
 "browsers": ["> 1%", "last 2 versions", "not ie <= 8"]  
 }  
 }],  
 "stage-2"  
 ],  
  
 "plugins": ["transform-vue-jsx", "transform-runtime"]//①  
 }  
 注释：  
 ①、点这里→transform-vue-jsx文档传送门  
 点这里→transform-runtime文档传送门  
  
 4、src内文件  
 我们开发的代码都存放在src目录下，根据需要我们通常会再建一些文件夹。比如pages的文件夹，用来存放页面让components文件夹专门做好组件的工作；api文件夹，来封装请求的参数和方法；store文件夹，使用vuex来作为vue的状态管理工具，我也常叫它作前端的数据库等。  
  
 ①、assets文件：脚手架自动会放入一个图片在里面作为初始页面的logo。平常我们使用的时候会在里面建立js，css，img，fonts等文件夹，作为静态资源调用  
  
 ②、components文件夹：用来存放组件，合理地使用组件可以高效地实现复用等功能，从而更好地开发项目。一般情况下比如创建头部组件的时候，我们会新建一个header的文件夹，然后再新建一个header.vue的文件  
  
 ③、router文件夹：该文件夹下有一个叫index.js文件，用于实现页面的路由跳转，具体使用请点击→vue-router传送门  
  
 ④、App.vue：作为我们的主组件，可通过使用<**router-view**/>开放入口让其他的页面组件得以显示。  
  
 ⑤、main.js：作为我们的入口文件，主要作用是初始化vue实例并使用需要的插件，小型项目省略router时可放在该处  
  
 注释：具体vue的用法可查看vue官方中文文档传送门  
  
 5、其他文件  
 ①、.editorconfig：编辑器的配置文件  
  
 ②、.gitignore：忽略git提交的一个文件，配置之后提交时将不会加载忽略的文件  
  
 ③、index.html：页面入口，经过编译之后的代码将插入到这来。  
  
 ④、package.lock.json：锁定安装时的包的版本号，并且需要上传到git，以保证其他人在npm install时大家的依赖能保证一致  
  
 ⑤、README.md：可此填写项目介绍  
  
 ⑥、node\_modules：根据package.json安装时候生成的的依赖（安装包）  
  
 三、config文件夹  
 ├─config  
 │ ├─dev.env.js  
 │ ├─index.js  
 │ ├─prod.env.js  
 1、config/dev.env.js  
 config内的文件其实是服务于build的，大部分是定义一个变量export出去。  
  
 'use strict'//采用严格模式  
 const merge = require('webpack-merge')//①  
 const prodEnv = require('./prod.env')  
 //webpack-merge提供了一个合并函数，它将数组和合并对象创建一个新对象。  
 //如果遇到函数，它将执行它们，通过算法运行结果，然后再次将返回的值封装在函数中.这边将dev和prod进行合并  
 module.exports = merge(prodEnv, {  
 NODE\_ENV: '"development"'  
 })  
 注释：①、点这里→webpack-merge文档传送门  
  
 2、config/prod.env.js  
 当开发是调取dev.env.js的开发环境配置，发布时调用prod.env.js的生产环境配置  
  
 'use strict'  
 module.exports = {  
 NODE\_ENV: '"production"'  
 }  
 3、config/index.js  
 'use strict'  
 const path = require('path')  
  
 module.exports = {  
 dev: {  
 // 开发环境下面的配置  
 assetsSubDirectory: 'static',//子目录，一般存放css,js,image等文件  
 assetsPublicPath: '/',//根目录  
 proxyTable: {},//可利用该属性解决跨域的问题  
 host: 'localhost', // 地址  
 port: 8080, //端口号设置，端口号占用出现问题可在此处修改  
 autoOpenBrowser: false,//是否在编译（输入命令行npm run dev）后打开http://localhost:8080/页面，以前配置为true，近些版本改为false，个人偏向习惯自动打开页面  
 errorOverlay: true,//浏览器错误提示  
 notifyOnErrors: true,//跨平台错误提示  
 poll: false, //使用文件系统(file system)获取文件改动的通知devServer.watchOptions  
 devtool: 'cheap-module-eval-source-map',//增加调试，该属性为原始源代码（仅限行）不可在生产环境中使用  
 cacheBusting: true,//使缓存失效  
 cssSourceMap: true//代码压缩后进行调bug定位将非常困难，于是引入sourcemap记录压缩前后的位置信息记录，当产生错误时直接定位到未压缩前的位置，将大大的方便我们调试  
 },  
  
 build: {  
 // 生产环境下面的配置  
 index: path.resolve(\_\_dirname, '../dist/index.html'),//index编译后生成的位置和名字，根据需要改变后缀，比如index.php  
 assetsRoot: path.resolve(\_\_dirname, '../dist'),//编译后存放生成环境代码的位置  
 assetsSubDirectory: 'static',//js,css,images存放文件夹名  
 assetsPublicPath: '/',//发布的根目录，通常本地打包dist后打开文件会报错，此处修改为./。如果是上线的文件，可根据文件存放位置进行更改路径  
 productionSourceMap: true,  
 devtool: '#source-map',//①  
 //unit的gzip命令用来压缩文件，gzip模式下需要压缩的文件的扩展名有js和css  
 productionGzip: false,  
 productionGzipExtensions: ['js', 'css'],  
 bundleAnalyzerReport: process.env.npm\_config\_report  
 }  
 }  
 注释：①点击→devtool文档传送门  
  
 四、build文件夹  
 ├─build  
 │ ├─build.js  
 │ ├─check-versions.js  
 │ ├─utils.js  
 │ ├─vue-loader.conf.js  
 │ ├─webpack.base.conf.js  
 │ ├─webpack.dev.conf.js  
 │ ├─webpack.prod.conf.js  
 1、build/build.js  
 该文件作用，即构建生产版本。package.json中的scripts的build就是node build/build.js，输入命令行npm run build对该文件进行编译生成生产环境的代码。  
  
 'use strict'  
 require('./check-versions')()//check-versions：调用检查版本的文件。加（）代表直接调用该函数  
 process.env.NODE\_ENV = 'production'//设置当前是生产环境  
 //下面定义常量引入插件  
 const ora = require('ora')//①加载动画  
 const rm = require('rimraf')//②删除文件  
 const path = require('path')  
 const chalk = require('chalk')//③对文案输出的一个彩色设置  
 const webpack = require('webpack')  
 const config = require('../config')//默认读取下面的index.js文件  
 const webpackConfig = require('./webpack.prod.conf')  
 //调用start的方法实现加载动画，优化用户体验  
 const spinner = ora('building for production...')  
 spinner.start()  
 //先删除dist文件再生成新文件，因为有时候会使用hash来命名，删除整个文件可避免冗余  
 rm(path.join(config.build.assetsRoot, config.build.assetsSubDirectory), err => {  
 if (err) throw err  
 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()) {  
 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'  
 ))  
 })  
 })  
 注释：  
 ①、点这里→ora文档传送门  
 ②、点这里→chalk文档传送门  
 ③、点这里→rimraf文档传送门  
  
 2、build/check-version.js  
 该文件用于检测node和npm的版本，实现版本依赖  
  
 'use strict'  
 const chalk = require('chalk')  
 const semver = require('semver')//①对版本进行检查  
 const packageConfig = require('../package.json')  
 const shell = require('shelljs')  
  
 function exec (cmd) {  
 //返回通过child\_process模块的新建子进程，执行 Unix 系统命令后转成没有空格的字符串  
 return require('child\_process').execSync(cmd).toString().trim()  
 }  
  
 const versionRequirements = [  
 {  
 name: 'node',  
 currentVersion: semver.clean(process.version),//使用semver格式化版本  
 versionRequirement: packageConfig.engines.node//获取package.json中设置的node版本  
 }  
 ]  
  
 if (shell.which('npm')) {  
 versionRequirements.push({  
 name: 'npm',  
 currentVersion: exec('npm --version'),// 自动调用npm --version命令，并且把参数返回给exec函数，从而获取纯净的版本号  
 versionRequirement: packageConfig.engines.npm  
 })  
 }  
  
 module.exports = function () {  
 const warnings = []  
 for (let i = 0; i < **versionRequirements.length**; i++) {  
 const mod = versionRequirements[i]  
  
 if (!semver.satisfies(mod.currentVersion, mod.versionRequirement)) {  
 //上面这个判断就是如果版本号不符合package.json文件中指定的版本号，就执行下面错误提示的代码  
 warnings.push(mod.name + ': ' +  
 chalk.red(mod.currentVersion) + ' should be ' +  
 chalk.green(mod.versionRequirement)  
 )  
 }  
 }  
  
 if (warnings.length) {  
 console.log('')  
 console.log(chalk.yellow('To use this template, you must update following to modules:'))  
 console.log()  
  
 for (let i = 0; i < **warnings.length**; i++) {  
 const warning = warnings[i]  
 console.log(' ' + warning)  
 }  
  
 console.log()  
 process.exit(1)  
 }  
 }  
 注释：  
 ①、点这里→chalk文档传送门  
 点这里→semver文档传送门  
  
 3、build/utils.js  
 utils是工具的意思，是一个用来处理css的文件。  
  
 'use strict'  
 const path = require('path')  
 const config = require('../config')  
 const ExtractTextPlugin = require('extract-text-webpack-plugin')  
 const packageConfig = require('../package.json')  
 //导出文件的位置，根据环境判断开发环境和生产环境，为config文件中index.js文件中定义的build.assetsSubDirectory或dev.assetsSubDirectory  
 exports.assetsPath = function (\_path) {  
 const assetsSubDirectory = process.env.NODE\_ENV === 'production'  
 ? config.build.assetsSubDirectory  
 : config.dev.assetsSubDirectory  
 //Node.js path 模块提供了一些用于处理文件路径的小工具①  
 return path.posix.join(assetsSubDirectory, \_path)  
 }  
  
 exports.cssLoaders = function (options) {  
 options = options || {}  
 //使用了css-loader和postcssLoader，通过options.usePostCSS属性来判断是否使用postcssLoader中压缩等方法  
 const cssLoader = {  
 loader: 'css-loader',  
 options: {  
 sourceMap: options.sourceMap  
 }  
 }  
  
 const postcssLoader = {  
 loader: 'postcss-loader',  
 options: {  
 sourceMap: options.sourceMap  
 }  
 }  
 function generateLoaders (loader, loaderOptions) {  
 const loaders = options.usePostCSS ? [cssLoader, postcssLoader] : [cssLoader]  
 if (loader) {  
 loaders.push({  
 loader: loader + '-loader',  
 //Object.assign是es6语法的浅复制，后两者合并后复制完成赋值  
 options: Object.assign({}, loaderOptions, {  
 sourceMap: options.sourceMap  
 })  
 })  
 }  
  
 if (options.extract) {  
 //ExtractTextPlugin可提取出文本，代表首先使用上面处理的loaders，当未能正确引入时使用vue-style-loader  
 return ExtractTextPlugin.extract({  
 use: loaders,  
 fallback: 'vue-style-loader'  
 })  
 } else {  
 //返回vue-style-loader连接loaders的最终值  
 return ['vue-style-loader'].concat(loaders)  
 }  
 }  
 return {  
 css: generateLoaders(),//需要css-loader 和 vue-style-loader  
 postcss: generateLoaders(),//需要css-loader和postcssLoader 和 vue-style-loader  
 less: generateLoaders('less'),//需要less-loader 和 vue-style-loader  
 sass: generateLoaders('sass', { indentedSyntax: true }),//需要sass-loader 和 vue-style-loader  
 scss: generateLoaders('sass'),//需要sass-loader 和 vue-style-loader  
 stylus: generateLoaders('stylus'),//需要stylus-loader 和 vue-style-loader  
 styl: generateLoaders('stylus')//需要stylus-loader 和 vue-style-loader  
 }  
 }  
 exports.styleLoaders = function (options) {  
 const output = []  
 const loaders = exports.cssLoaders(options)  
 //将各种css,less,sass等综合在一起得出结果输出output  
 for (const extension in loaders) {  
 const loader = loaders[extension]  
 output.push({  
 test: new RegExp('\\.' + extension + '$'),  
 use: loader  
 })  
 }  
  
 return output  
 }  
  
 exports.createNotifierCallback = () => {  
 //发送跨平台通知系统  
 const notifier = require('node-notifier')  
  
 return (severity, errors) => {  
 if (severity !== 'error') return  
 //当报错时输出错误信息的标题，错误信息详情，副标题以及图标  
 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')  
 })  
 }  
 }  
 注释：  
 ①、path.posix：提供对路径方法的POSIX（可移植性操作系统接口）特定实现的访问，即可跨平台，区别于win32。  
 path.join：用于连接路径，会正确使用当前系统的路径分隔符，Unix系统是"/"，Windows系统是""  
 点击→path用法传送门  
  
 4、vue-loader.conf.js  
 该文件的主要作用就是处理.vue文件，解析这个文件中的每个语言块（template、script、style),转换成js可用的js模块。  
  
 'use strict'  
 const utils = require('./utils')  
 const config = require('../config')  
 const isProduction = process.env.NODE\_ENV === 'production'  
 const sourceMapEnabled = isProduction  
 ? config.build.productionSourceMap  
 : config.dev.cssSourceMap  
 //处理项目中的css文件，生产环境和测试环境默认是打开sourceMap，而extract中的提取样式到单独文件只有在生产环境中才需要  
 module.exports = {  
 loaders: utils.cssLoaders({  
 sourceMap: sourceMapEnabled,  
 extract: isProduction  
 }),  
 cssSourceMap: sourceMapEnabled,  
 cacheBusting: config.dev.cacheBusting,  
 // 在模版编译过程中，编译器可以将某些属性，如 src 路径，转换为require调用，以便目标资源可以由 webpack 处理.  
 transformToRequire: {  
 video: ['src', 'poster'],  
 source: 'src',  
 img: 'src',  
 image: 'xlink:href'  
 }  
 }  
 5、webpack.base.conf.js  
 webpack.base.conf.js是开发和生产共同使用提出来的基础配置文件，主要实现配制入口，配置输出环境，配置模块resolve和插件等  
  
 'use strict'  
 const path = require('path')  
 const utils = require('./utils')  
 const config = require('../config')  
 const vueLoaderConfig = require('./vue-loader.conf')  
  
 function resolve (dir) {  
 //拼接出绝对路径  
 return path.join(\_\_dirname, '..', dir)  
 }  
 module.exports = {  
 //path.join将路径片段进行拼接，而path.resolve将以/开始的路径片段作为根目录，在此之前的路径将会被丢弃  
 //path.join('/a', '/b') // 'a/b',path.resolve('/a', '/b') // '/b'  
 context: path.resolve(\_\_dirname, '../'),  
 //配置入口，默认为单页面所以只有app一个入口  
 entry: {  
 app: './src/main.js'  
 },  
 //配置出口，默认是/dist作为目标文件夹的路径  
 output: {  
 path: config.build.assetsRoot,//路径  
 filename: '[name].js',//文件名  
 publicPath: process.env.NODE\_ENV === 'production'  
 ? config.build.assetsPublicPath  
 : config.dev.assetsPublicPath//公共存放路径  
 },  
 resolve: {  
 //自动的扩展后缀，比如一个js文件，则引用时书写可不要写.js  
 extensions: ['.js', '.vue', '.json'],  
 //创建路径的别名，比如增加'components': resolve('src/components')等  
 alias: {  
 'vue$': 'vue/dist/vue.esm.js',  
 '@': resolve('src'),  
 }  
 },  
 //使用插件配置相应文件的处理方法  
 module: {  
 rules: [  
 //使用vue-loader将vue文件转化成js的模块①  
 {  
 test: /\.vue$/,  
 loader: 'vue-loader',  
 options: vueLoaderConfig  
 },  
 //js文件需要通过babel-loader进行编译成es5文件以及压缩等操作②  
 {  
 test: /\.js$/,  
 loader: 'babel-loader',  
 include: [resolve('src'), resolve('test'), resolve('node\_modules/webpack-dev-server/client')]  
 },  
 //图片、音像、字体都使用url-loader进行处理，超过10000会编译成base64③  
 {  
 test: /\.(png|jpe?g|gif|svg)(\?.\*)?$/,  
 loader: 'url-loader',  
 options: {  
 limit: 10000,  
 name: utils.assetsPath('img/[name].[hash:7].[ext]')  
 }  
 },  
 {  
 test: /\.(mp4|webm|ogg|mp3|wav|flac|aac)(\?.\*)?$/,  
 loader: 'url-loader',  
 options: {  
 limit: 10000,  
 name: utils.assetsPath('media/[name].[hash:7].[ext]')  
 }  
 },  
 {  
 test: /\.(woff2?|eot|ttf|otf)(\?.\*)?$/,  
 loader: 'url-loader',  
 options: {  
 limit: 10000,  
 name: utils.assetsPath('fonts/[name].[hash:7].[ext]')  
 }  
 }  
 ]  
 },  
 //以下选项是Node.js全局变量或模块，这里主要是防止webpack注入一些Node.js的东西到vue中  
 node:  
 setImmediate: false,  
 dgram: 'empty',  
 fs: 'empty',  
 net: 'empty',  
 tls: 'empty',  
 child\_process: 'empty'  
 }  
 }  
 注释：  
 ①、点击→vue-loader文档传送门  
 ②、点击→babel-loader文档传送门  
  
 6、webpack.dev.conf.js  
 'use strict'  
 const utils = require('./utils')  
 const webpack = require('webpack')  
 const config = require('../config')  
 //通过webpack-merge实现webpack.dev.conf.js对wepack.base.config.js的继承  
 const merge = require('webpack-merge')  
 const path = require('path')  
 const baseWebpackConfig = require('./webpack.base.conf')  
 const CopyWebpackPlugin = require('copy-webpack-plugin')  
 const HtmlWebpackPlugin = require('html-webpack-plugin')  
 //美化webpack的错误信息和日志的插件①  
 const FriendlyErrorsPlugin = require('friendly-errors-webpack-plugin')  
 const portfinder = require('portfinder')// 查看空闲端口位置，默认情况下搜索8000这个端口②  
 const HOST = process.env.HOST//③processs为node的一个全局对象获取当前程序的环境变量，即host  
 const PORT = process.env.PORT && Number(process.env.PORT)  
  
 const devWebpackConfig = merge(baseWebpackConfig, {  
 module: {  
 //规则是工具utils中处理出来的styleLoaders，生成了css，less,postcss等规则  
 rules: utils.styleLoaders({ sourceMap: config.dev.cssSourceMap, usePostCSS: true })  
 },  
  
 devtool: config.dev.devtool, //增强调试，上文有提及  
 //此处的配置都是在config的index.js中设定好了  
 devServer: {//④  
 clientLogLevel: 'warning',//控制台显示的选项有none, error, warning 或者 info  
 //当使用 HTML5 History API 时，任意的 404 响应都可能需要被替代为 index.html  
 historyApiFallback: {  
 rewrites: [  
 { from: /.\*/, to: path.posix.join(config.dev.assetsPublicPath, 'index.html') },  
 ],  
 },  
 hot: true,//热加载  
 contentBase: false,  
 compress: true,//压缩  
 host: HOST || config.dev.host,  
 port: PORT || config.dev.port,  
 open: config.dev.autoOpenBrowser,//调试时自动打开浏览器  
 overlay: config.dev.errorOverlay  
 ? { warnings: false, errors: true }  
 : false,// warning 和 error 都要显示  
 publicPath: config.dev.assetsPublicPath,  
 proxy: config.dev.proxyTable,//接口代理  
 quiet: true, //控制台是否禁止打印警告和错误,若用FriendlyErrorsPlugin 此处为 true  
 watchOptions: {  
 poll: config.dev.poll,//// 文件系统检测改动  
 }  
 },  
 plugins: [  
 new webpack.DefinePlugin({  
 'process.env': require('../config/dev.env')  
 }),  
 new webpack.HotModuleReplacementPlugin(),//⑤模块热替换插件，修改模块时不需要刷新页面  
 new webpack.NamedModulesPlugin(), // 显示文件的正确名字  
 new webpack.NoEmitOnErrorsPlugin(),//当webpack编译错误的时候，来中端打包进程，防止错误代码打包到文件中  
 // https://github.com/ampedandwired/html-webpack-plugin  
 // 该插件可自动生成一个 html5 文件或使用模板文件将编译好的代码注入进去⑥  
 new HtmlWebpackPlugin({  
 filename: 'index.html',  
 template: 'index.html',  
 inject: true  
 }),  
 new CopyWebpackPlugin([//复制插件  
 {  
 from: path.resolve(\_\_dirname, '../static'),  
 to: config.dev.assetsSubDirectory,  
 ignore: ['.\*']//忽略.\*的文件  
 }  
 ])  
 ]  
 })  
  
 module.exports = new Promise((resolve, reject) => {  
 portfinder.basePort = process.env.PORT || config.dev.port  
 //查找端口号  
 portfinder.getPort((err, port) => {  
 if (err) {  
 reject(err)  
 } else {  
 //端口被占用时就重新设置evn和devServer的端口  
 process.env.PORT = port  
 devWebpackConfig.devServer.port = port  
 //友好地输出信息  
 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)  
 }  
 })  
 })  
 注释：  
 ①、点击→friendly-errors-webpack-plugin文档传送门  
 ②、点击→process文档传送门  
 ③、点击→babel-loader文档传送门  
 ④、点击→devtool文档传送门  
 ⑤、点击→webpack的HotModuleReplacementPlugin文档传送门  
 ⑥、点击→html-webpack-plugin文档传送门  
  
 7、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')  
 const CopyWebpackPlugin = require('copy-webpack-plugin')  
 const HtmlWebpackPlugin = require('html-webpack-plugin')  
 const ExtractTextPlugin = require('extract-text-webpack-plugin')  
 const OptimizeCSSPlugin = require('optimize-css-assets-webpack-plugin')  
 const UglifyJsPlugin = require('uglifyjs-webpack-plugin')  
  
 const env = require('../config/prod.env')  
  
 const webpackConfig = merge(baseWebpackConfig, {  
 module: {  
 //调用utils.styleLoaders的方法  
 rules: utils.styleLoaders({  
 sourceMap: config.build.productionSourceMap,//开启调试的模式。默认为true  
 extract: true,  
 usePostCSS: true  
 })  
 },  
 devtool: config.build.productionSourceMap ? config.build.devtool : false,  
 output: {  
 path: config.build.assetsRoot,  
 filename: utils.assetsPath('js/[name].[chunkhash].js'),  
 chunkFilename: utils.assetsPath('js/[id].[chunkhash].js')  
 },  
 plugins: [  
 new webpack.DefinePlugin({  
 'process.env': env  
 }),  
 new UglifyJsPlugin({  
 uglifyOptions: {  
 compress: {//压缩  
 warnings: false//警告：true保留警告，false不保留  
 }  
 },  
 sourceMap: config.build.productionSourceMap,  
 parallel: true  
 }),  
 new ExtractTextPlugin({//抽取文本。比如打包之后的index页面有style插入，就是这个插件抽取出来的，减少请求  
 filename: utils.assetsPath('css/[name].[contenthash].css'),  
 allChunks: true,  
 }),  
  
 new OptimizeCSSPlugin({//优化css的插件  
 cssProcessorOptions: config.build.productionSourceMap  
 ? { safe: true, map: { inline: false } }  
 : { safe: true }  
 }),  
  
 new HtmlWebpackPlugin({//html打包  
 filename: config.build.index,  
 template: 'index.html',  
 inject: true,  
 minify: {//压缩  
 removeComments: true,//删除注释  
 collapseWhitespace: true,//删除空格  
 removeAttributeQuotes: true//删除属性的引号  
 },  
  
 chunksSortMode: 'dependency'//模块排序，按照我们需要的顺序排序  
 }),  
  
 new webpack.HashedModuleIdsPlugin(),  
 new webpack.optimize.ModuleConcatenationPlugin(),  
 new webpack.optimize.CommonsChunkPlugin({//抽取公共的模块  
 name: 'vendor',  
 minChunks (module) {  
 return (  
 module.resource &&  
 /\.js$/.test(module.resource) &&  
 module.resource.indexOf(  
 path.join(\_\_dirname, '../node\_modules')  
 ) === 0  
 )  
 }  
 }),  
 new webpack.optimize.CommonsChunkPlugin({  
 name: 'manifest',  
 minChunks: Infinity  
 }),  
 new webpack.optimize.CommonsChunkPlugin({  
 name: 'app',  
 async: 'vendor-async',  
 children: true,  
 minChunks: 3  
 }),  
 new CopyWebpackPlugin([//复制，比如打包完之后需要把打包的文件复制到dist里面  
 {  
 from: path.resolve(\_\_dirname, '../static'),  
 to: config.build.assetsSubDirectory,  
 ignore: ['.\*']  
 }  
 ])  
 ]  
 })  
  
 if (config.build.productionGzip) {  
 const CompressionWebpackPlugin = require('compression-webpack-plugin')  
  
 webpackConfig.plugins.push(  
 new CompressionWebpackPlugin({  
 asset: '[path].gz[query]',  
 algorithm: 'gzip',  
 test: new RegExp(  
 '\\.(' +  
 config.build.productionGzipExtensions.join('|') +  
 ')$'  
 ),  
 threshold: 10240,  
 minRatio: 0.8  
 })  
 )  
 }  
  
 if (config.build.bundleAnalyzerReport) {  
 const BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin  
 webpackConfig.plugins.push(new BundleAnalyzerPlugin())  
 }  
  
 module.exports = webpackConfig  
 注释：webpack.prod.conf.js详细内容  
  
 五、结语  
 第一篇博文总在想要写点什么，就根据自己的经验加查了下文档写了这么一篇工具类的文章，由于有些插件有些用法会重复，所以按照文章先后，写过用法或者给过链接的插件，在后面的文章就省略了，有时间的建议从头开始，如果单独看某章节的话遇到不懂的语法或插件可全文查找，也可以点击更多安装包传送门进行查找阅读。本文将vue本身自带的英文注释删除了，但英文注释非常有用可以仔细阅读，希望对大家学习vue和webpack都有所帮助。