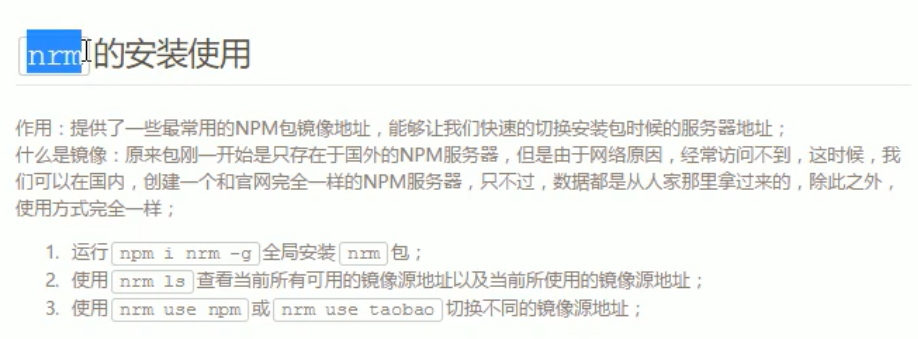
Vue.工程化

nrm工具的使用





nrm只是单纯的提供了几个常用的url地址

1 Webpack的学习

* 1. 网页常见的静态资源

注意:本内容的webpack版本是基于3.x的版本.新版本要跟进

JS : [.js .jsx .coffee .ts(TypeScript)]

CSS: [.less .sass .scss]

Images : .jpg .png .gif .bmp .svg

字体文件(Fonts):.svg .ttf .ect woff .woff2

模板文件 .ejs .jade .vue(这是在webpack中定义组件的方式,推荐这么用)

* + 1. 网页中引入静态资源多了以后会有什么问题

a 网页加载速度慢,因为我们要发起很多的二次请求;

b 要处理错综复杂的依赖关系

* + 1. 如何处理

a 合并,压缩,精灵图,图片的Base64编码字符串

b 可以使用require JS ,也可以使用webpack可以解决各个包之间的复杂依赖关系

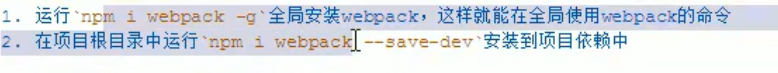
* 1. webpack 的介绍

是前端的一个项目的构建工具,是基于Node.js开封出来的一个前端打包工具

完美实现上述两种解决方案?

1. 使用Gulp,是基于task方案
2. 使用webpack,是基于整个项目构建的

1.2.1安装的两种方式



案例: 隔行变色

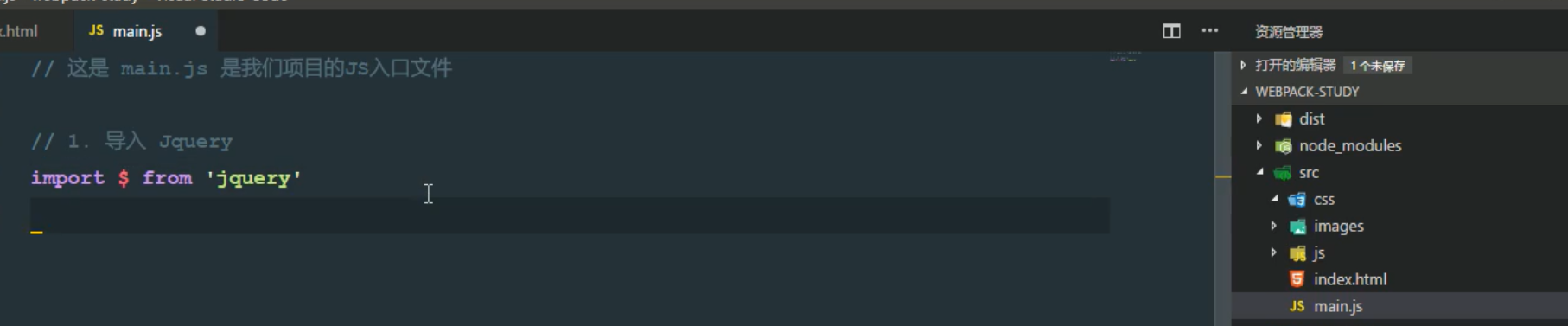
安装jquery要先init

npm init -y 初始

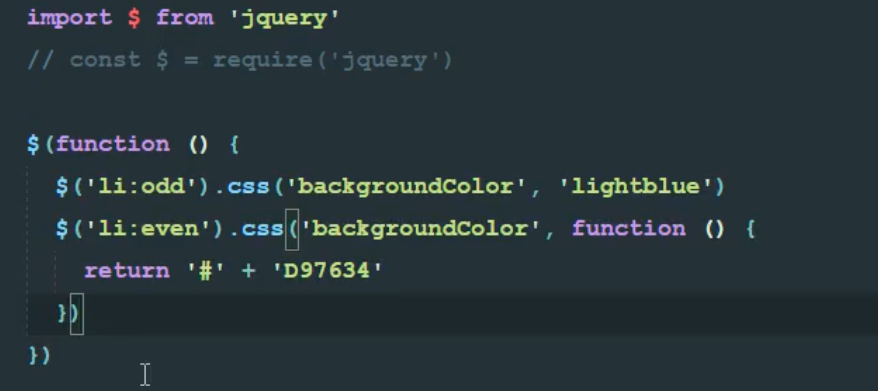
npm i jquery

不推荐在index.html中引入任何文件

而是在main,js 中用 import from 的方式导入



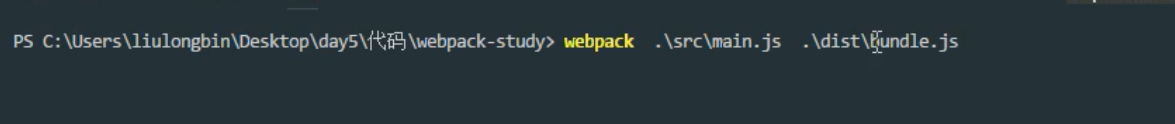
import $ from “xxx” 是ES6 导入模块的方式

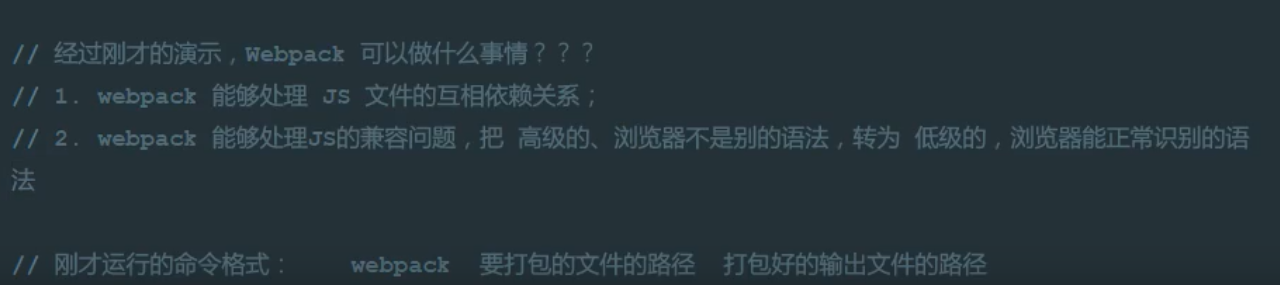


浏览器报错

这时有webpack 来解析

webpack :我要处理这个文件 bundle.js

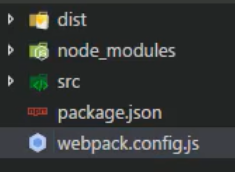




1. webpack的基本配置

2.1 新建webpack.config.js

在当前目录中新建webpack.config.js 来配置出入口文件



* + 1. 在文件中指定出入口文件

这个配置文件就是一个JS文件,通过Node中的模块操作,向外暴露了一个配置对象

module.exports= { 需要手动指定出口入口文件

entry : 入口 表示要使用webpack打包那个软件

output:出口 输出文件相关配置

}



当我们在控制台直接输入webpack命令执行的时候,webpack做了以下几步:

1. 首先,webpack发现,我们并没有通过命令的形式,给他指定的出口与入口
2. webpack就会去项目的根目录中,查找一个’webpack.config.js’的配置文件
3. 找到这个配置文件后,就解析执行这个配置文件,当解析完这个配置文件后,就会找到这个配置文件中,导出的配置对象
4. 当webpack拿到配置对象后,就拿到了配置对象中,指定的入口和出口,然后进行打包构建



以上失败

2.2 webpack-dev-server

这个工具,来实现自动打包编译的功能

1 运行npm i webpack-dev-server -D 把这个工具安装到项目的本地开发环境依赖

1. 安装完毕后,这个工具的用法和webpack命令的用法完全一样
2. 由于,我们是在项目中,本地安装的webpack-dev-server ,所以无法把它当做脚本命令,在powershell终端中直接运行;(只有那些安装到全局 -g的工具,才能在终端中正常执行)

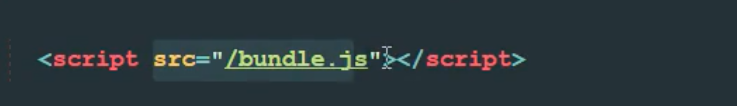
在webpack的package.json 文件中的script 中添加 “dev” : “webpack-dev-server” npm -run

删除node\_modules 在重新安装 node\_modules

1. webpack-dev-sever 这个工具,如果想要正常执行,要求在本地项目中必须安装webpack



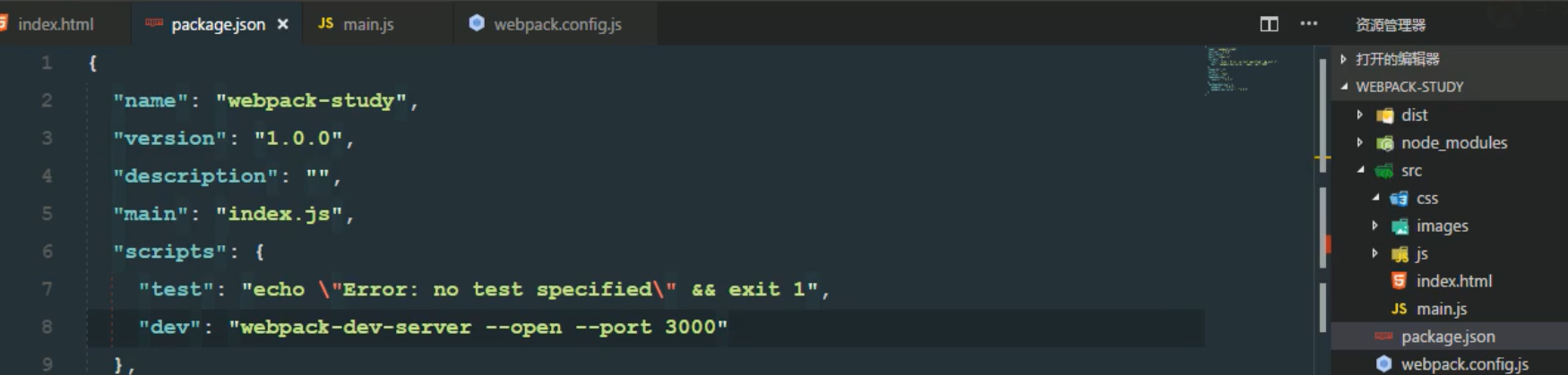
1. webpack-dev-server 帮我们打包生成的bundle.js 文件,并没有放到实际的物理磁盘上;而是直接托管到了电脑的内存中,所以,在我们项目根目录中,根本找不到bundle.js文件
2. 我们可以认为,webpack-dev-server把打包好的文件以一种虚拟的形式,托管到了我们项目的根目录中,虽看不到,但是可以认为和dist src node\_modules
3. index 界面 直接引入 bundle.js



* + 1. webpack-dev-server常用命令

--open 自动打开浏览器 在重启服务器 npm run dev

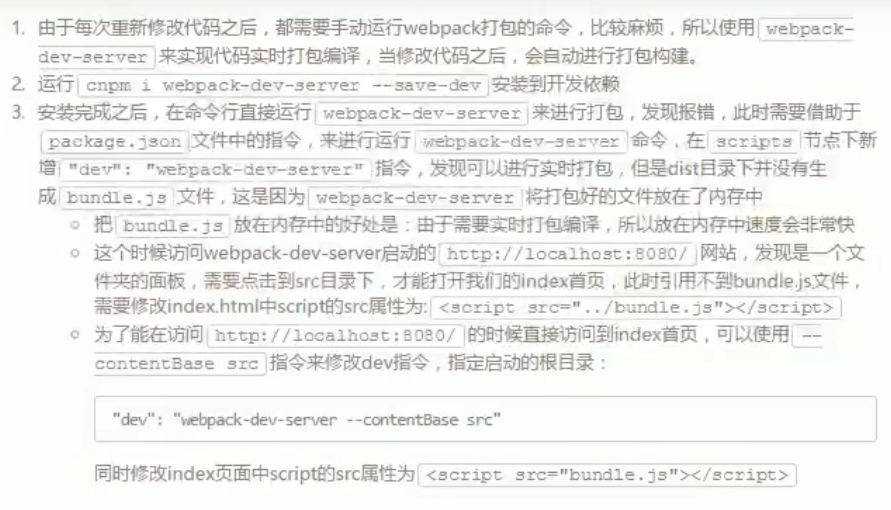
--post 3000 自动打开端口3000



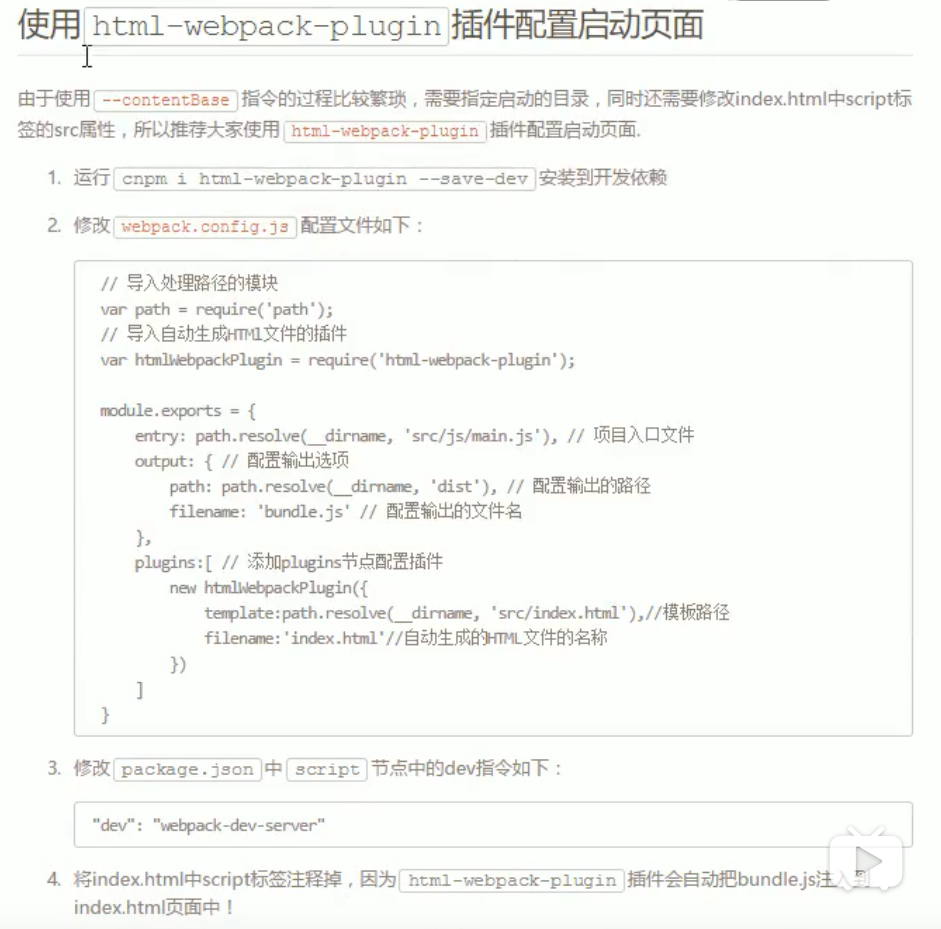
--contenBase src

以src目录

--hot (热更新,热重载)



插件: html-webpack-plugin 插件启动页面



3 loader加载器

3.1 css



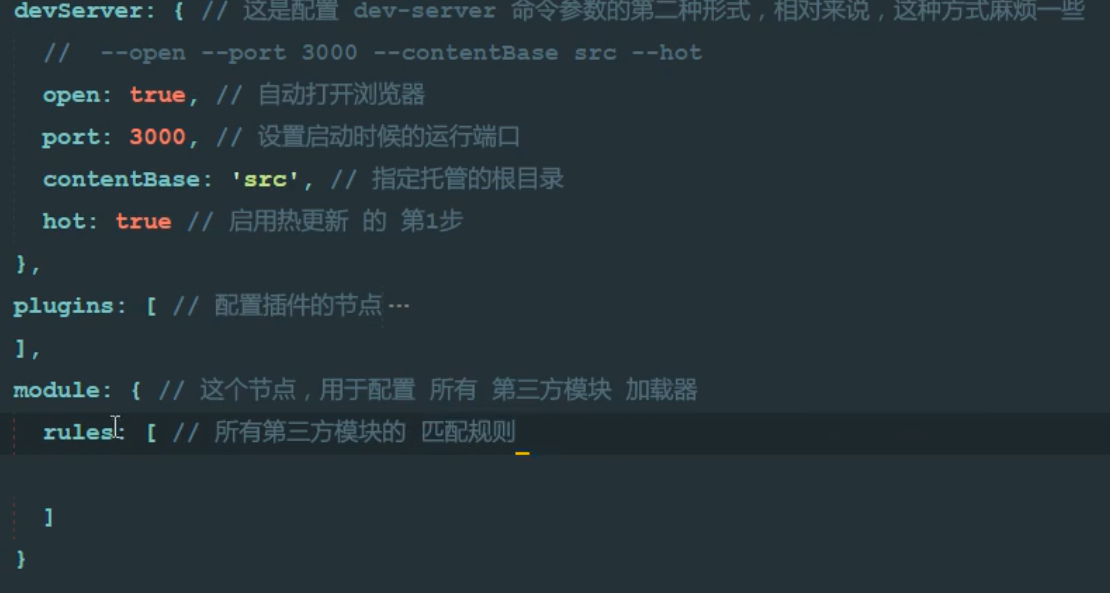
import ‘./css/index.css’

如果 想要打包处理css文件,需要安装

1 cnpm i style-loader css-loader -D

1. 打开webpack.config.js 新增一个配置对象节点

module对象身上有个rules属性,这个属性是一个数组,这个数组中存放了所有第三方文件的匹配和处理规则



里面有个test 就是正则来规定

/\.(css)$/,use:[‘style-loader’, ‘css-loader’]



配置处理.css文件的第三方规则

安装不上

3.2 配置less



处理 less 相关文件

cnpm I less-loader -D

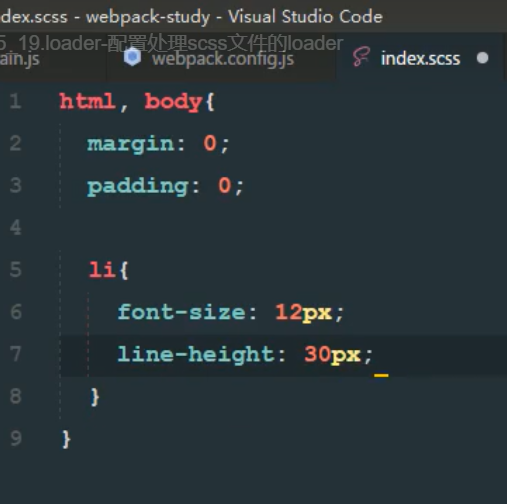
还需要安装 cnpm i less -D

test: /\.less$/,use:[‘style-loader’ , ‘css-loader’,’less-loader’]



3.3 sass配置

新建一个sass文件



导入一下 import ‘./css/sass’

安装 cnpm i sass

安装 [node-sass@4.0.0](mailto:node-sass@4.0.0)

cnpm i node-sass

{test: /\.scss$/, use: [ ‘style-loader’ , ‘css-loader’ , ‘sass-loader’]}

处理.scss文件的第三方loader规则

]}

npm run dev



3.3 复习 webpack的基本使用

npm init -y



4 url-loader 的使用

4.1

先装包,在配置节点

cnpm I sass-loader



