开发环境：vscode,node.js,vue.js,webpack

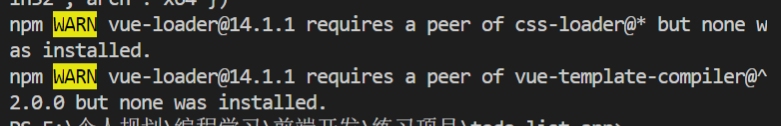
项目初始化

ctrl+` 打开控制台

npm init

npm i webpack vue vue-loaded

npm 出现warn提醒你需要依赖，按照提示进行安装



npm i css-loader vue-template-compiler

**配置webpack加载app.vue文件**

**app.vue文件内容如下：**

<template>

<div id="text">{{text}}</div>

</template>

<script>

export default{

data(){

return {

text: 'abc'

}

}

}

</script>

<style>

#test{

color: red;

}

</style>

创建webpack.config.js文件，配置入口entry，输出output

const path = require('path')

module.exports = {

entry: path.join(\_\_dirname,'src/index.js'),

output: {

filename: 'bundle.js',

path: path.join(\_\_dirname,'dist')

},

module: {

rules: [{

test: /.vue$/,

loader: 'vue-loader'

}]

}

}

创建 index.js文件作为入口

import Vue from 'vue'

import App from './app.vue'

const root = document.createElement('div')

document.body.appendChild(root)

new Vue({

render: (h) => h(App)

}).$mount(root)

配置package.json文件中的scripts命令，添加build

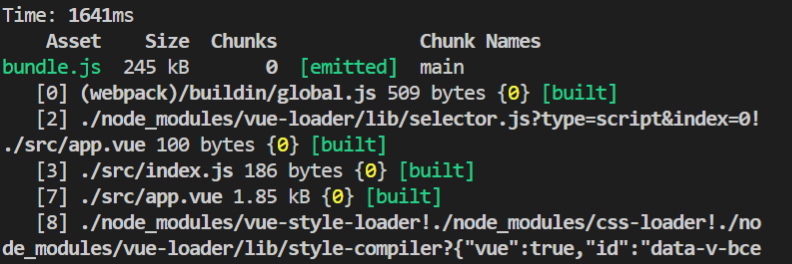
"scripts": {

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

"build": "webpack --config webpack.config.js"

},

在控制台输入npm run build 打包，打包成功后如图



**在webpack.config.js中配置需要打包的文件类型，vue/css/图片。**

url-loader：如果文件大小小于1024，就会转译成代码

module: {

rules: [{

test: /.vue$/,

loader: 'vue-loader'

},

{

test: /.css$/,

use:[

'style-loader',

'css-loader'

]

},

{

test: /\.(gif|jpg|png|svg)$/,

use: [

{

loader: 'url-loader',

options: {

limit: 1024,

name:'[name].[ext]'

}

}

]

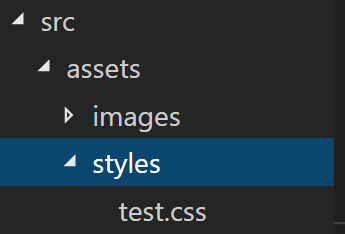
}]

}

安装相应的loader

npm i style-loader css-loader url-loader file-loader

在js代码中import这些非js的内容，创建测试文件test.css。在Index.js中import这些内容。npm run build 测试结果。



import Vue from 'vue'

import App from './app.vue'

import './assets/styles/test.css'

import './assets/images/0.jpg'

const root = document.createElement('div')

document.body.appendChild(root)

new Vue({

render: (h) => h(App)

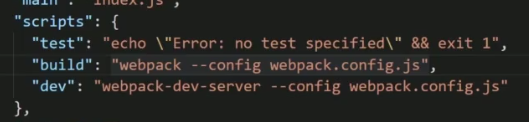
}).$mount(root)

**关于css预处理器。stylus的配置和测试**

**配置webpack-dev-server:专门用在开发环境的打包**

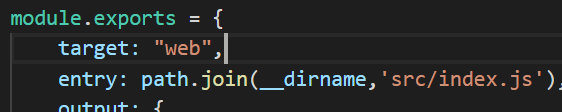
安装 npm i webpack-dev-server

修改package.jsom文件，在build下面添加dev配置



**修改webpack.config.js**

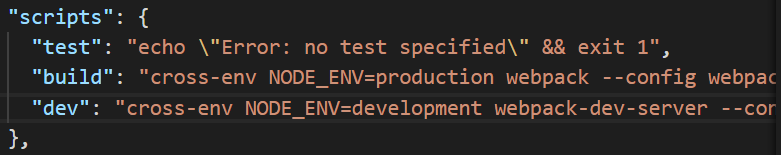
在全局添加target:’web’



因为这个文件要同时用在开发环境和正式环境，所以要加一个环境判断，在跑npm的时候添加变量标示环境。

安装npm i cross-env 包，使得在不同开发环境下命令相同。

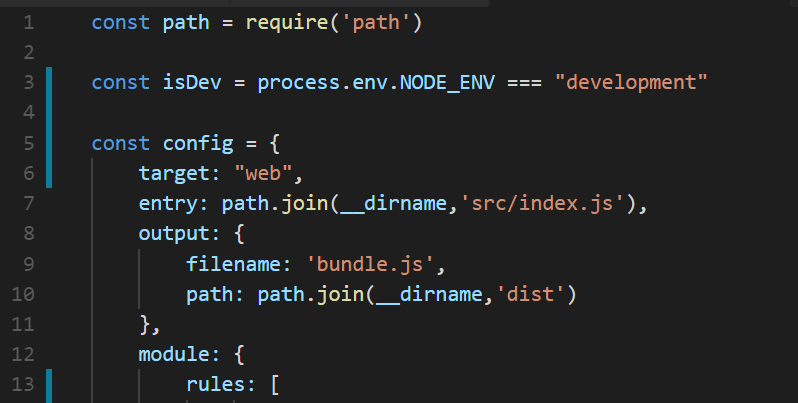
修改package.json文件



在webpack.config.js文件中进行判断。

webpack2之后就可以直接用config.devServer进行配置了。

host:’0.0.0.0’不要直接写localhost，这样别人的电脑就访问不了了

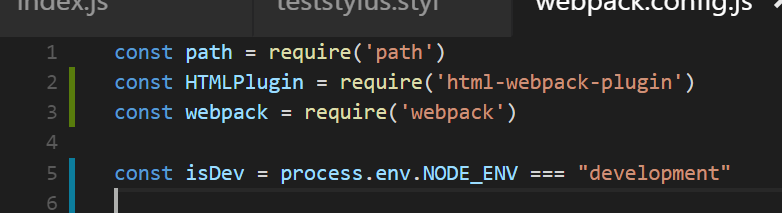


文件最后加上配置



**安装html-webpack-plugin插件使得**html可以**作为入口，自动包含JS**

**npm I html-webpack-plugin**





**配置完成**

**npm run dev**

**如果出现错误，根据提示修改，注意端口是否占用。我后来用了8080端口**

**浏览器访问：localhost:8080,可以查看渲染效果。恩，背景图是我爱豆。**



**最后还要加一些东西**

**historyFallback:{} 因为我们做的是单页应用，所以要加一下地址映射到入口index.html，这个先忽略**

**热加载功能。**

**hot功能，改了一个组件的代码，页面只重新渲染这个组件而不是整个页面进行渲染。**

**一些插件**

**webpack.HotModuleReplacementPlugin() 启动hot功能**

**webpack.NoEmitOnErrorsPlugin() 去除一些不必要的信息**

**浏览器调试代码功能的工具**

**config.devTool = “#cheap-module-evel-source-map”**

**最后配置完成后**



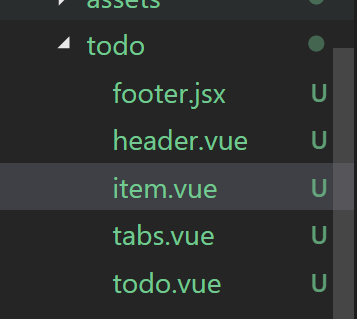
**安装**

**npm i postcss-loader autoprefixer babel-loader babel-core**

**页面实现**

**新建todo文件夹**

**新建各个文件**



**全局样式 global.styl**

**编辑app.vue实现背景虚化 用cover虚化 scoped 只应用在这个组件内**

**main-header**

**组件使用，先import，然后在export中声明使用了这个组件**

**jsx文件写footer，jsx中style要放在外面写，然后import**

**使用jsx更开放功能更强大。使用vue结构更清晰**

**组件，以及父级调用是重点。**

**在组件中进行交互操作，激活父级中的动作，this.$emit(‘父级中的动作’，‘需要从子组件传入到父组件中的参数’)**

**然后再在父级中调用具体操作，具体定义函数形式。**

**所有东西写完之后要进行打包。**

**直接打包存在问题。css文件不能单独打包，只会打包到bubble.js文件中，要单独打包css需要进行配置**

**npm i extract-text-webpack-plugin**

**//webpack.config.js文件**

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

**把非js的静态文件打包到一个文件中**

**删除css配置和styl配置**

{

test: /.css$/,

use:[

'style-loader',

'css-loader'

]

},

**添加开发环境**

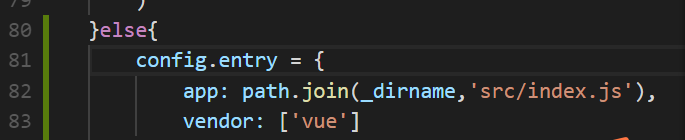


**添加正式环境，这两个不一样，要区分开来。**

**单独打包类库文件。**

**框架代码稳定性高，可以长时间缓存，减少流量和加载，所以单独打包**

**修改entry**



**hash的处理。hash 和chunkhash的区别**

**搭建vue的开发项目。webpack作为开发工具。用了jsx。babel定义。模块体系。静态资源加hash后缀，浏览器长缓存。处理静态资源的依赖关系。**