# 十分钟上手webpack

## 序言、webpack到底是干什么的

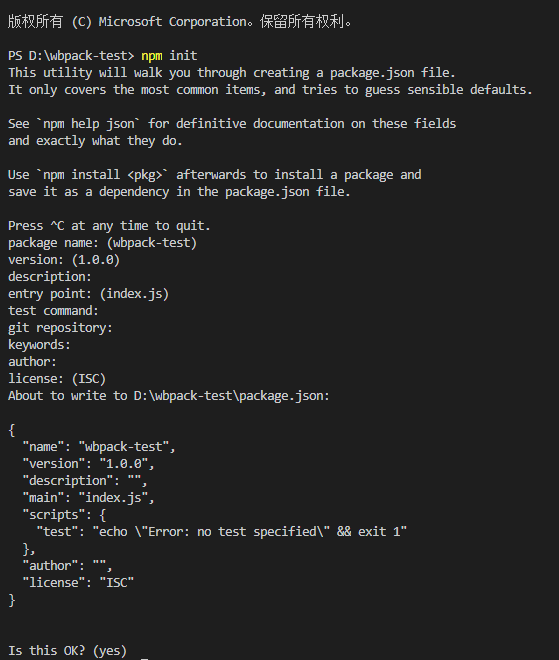
webpack其实就是一个打包工具。它可以将各种当下主流的sass，less，tsx,

vue,jsx,json甚至png,svg等各种格式的文件转化为浏览器可以识别的html，css，js和图片资源文件。将他们按照各种依赖关系，整合在一起，输出到一个bundle文件中。

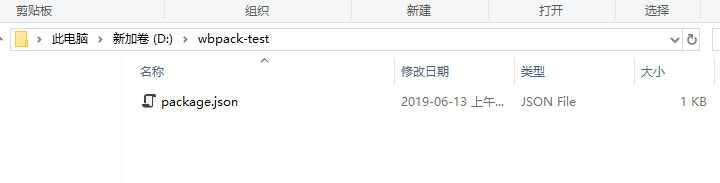
## 创建package.json

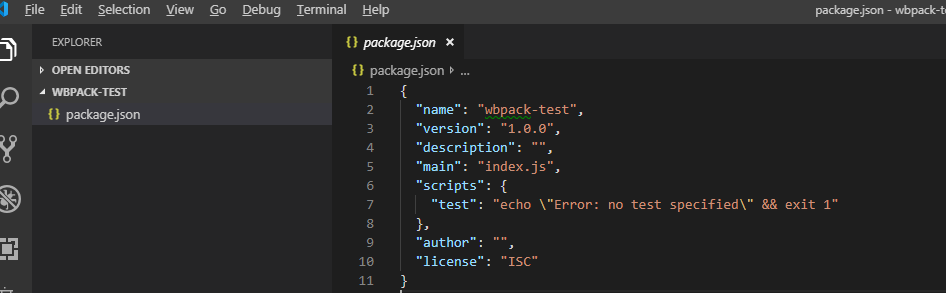
首先本地创建一个webpack-test文件夹，然后打开命令提示符，进入此目录，执行命令

npm init（npm是node.js自带的一个包管理工具，安装node.js就会自动安装npm）。



如上图所示，执行了命令之后，它会让你写一些信息，你可以写，也可以直接使用enter键下一步，使用默认值。最后一句yes之后，就会在webpack-test目录下生成一个package.json文件。如下图



文件初始默认内容：

这个文件的作用是方便项目转移，这个文件里面会记录很多项目依赖项（后面会有解释），以及以什么方式打开项目的脚本scripts。

## 安装webpack模块

用来记录项目操作流程的package文件有了之后，就要进入正题了，如果之前没有退

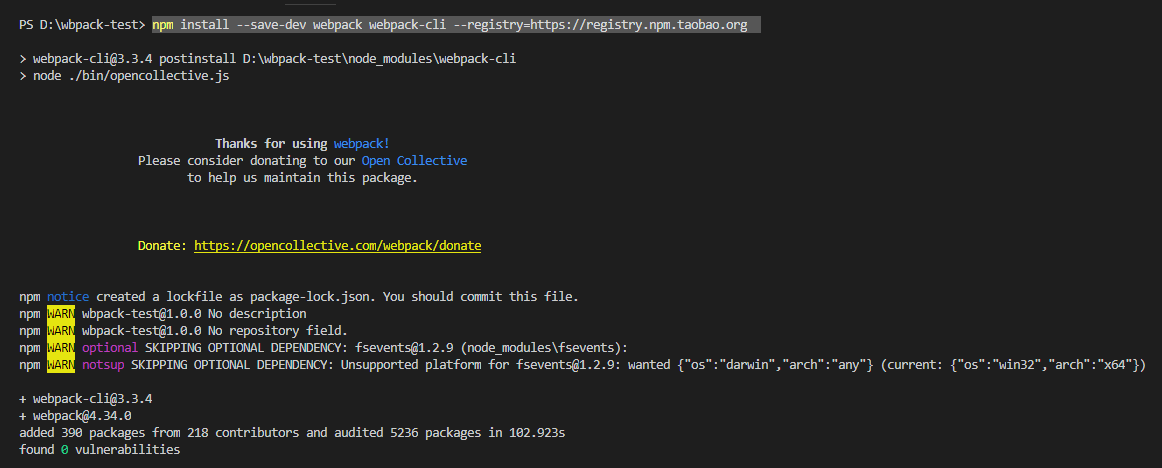
命令行的webpack-test目录的话，执行命令

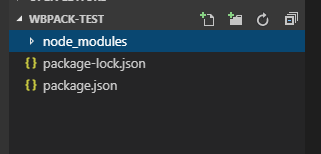
npm install --save-dev webpack webpack-cli --registry=https://registry.npm.taobao.org

（--registry=https://registry.npm.taobao.org 这个配置是为了解决国内用户下载npm模块速度慢，以及有时候会下载失败的可能）

安装webpack和webpack-cli（这两个插件缺一不可，webpack是打包工具，webpack-cli是处理打包命令的webpack自带的方便用户的脚手架）

执行命令成功之后的效果:



这时候目录下会增加一个packjson-lock.json文件和一个node\_modules目录（node\_modules目录是存放npm模块的目录，从npm install的所有模块都自动放在这个目录中，我们上面下载的webpack，和webpack-cli也在这里面

## 创建资源目录src

index.js文件

|  |
| --- |
| import './styles.css';    function component() {  var element = document.createElement('div');  var button = document.createElement('button');  var br = document.createElement('br');  console.log(\_.join(['Hello1', 'providePlugin'], ' '));  button.innerHTML = 'Click me1 lok at the console!';  element.appendChild(br);  element.appendChild(button);  //懒加载，第一次打开页面的时候不会去加载对应的js资源文件，当触发事件的时候才去加载资源文件  // webpackChunkName 这个注释的作用是命名懒加载的资源文件，否则会按照数字递增命名  button.onclick = e => import(/\* webpackChunkName: "math" \*/ './math').then(module => {  var print = module.default;  print();  });  return element;  }  document.body.appendChild(component()); |

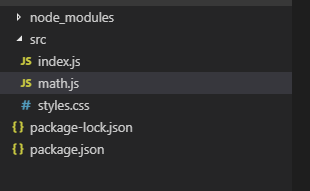
math.js文件

|  |
| --- |
| console.log(  'The print.js module has loaded! See the network tab in dev tools...'  );  export function square(x) {  return x \* x;  }    export function cube(x) {  return x \* x \* x;  }  export default () => {  console.log('Button Clicked: Here\'s "some text"!');  } |

styles.css文件

|  |
| --- |
| body{  background-color: aquamarine  } |

目录截图：



接下来我们通过配置webpack配置文件，然后让index文件里面的内容显示在浏览器上。看到这里，你可能有个疑问：这个目录下面连个html文件都没有，它怎么显示到浏览器上面？假如你看过官网的教程或者其他社区中的一些一步一步的webpac教程，这个文件夹里面一开始确实是有html文件的，但是我们这里是10分钟快速上手，就省略了，原因后面的标题会讲。

## webpack配置文件

1. webpack-test根目录下增加三个文件

webpack.common.js

|  |
| --- |
| const path = require('path');  const { CleanWebpackPlugin } = require('clean-webpack-plugin');  const HtmlWebpackPlugin = require('html-webpack-plugin');  const webpack = require('webpack');  module.exports = {  entry: {  index: './src/index.js'  },  output: {  filename: '[name].[hash].js',  path: path.resolve(\_\_dirname, 'dist'),  // publicPath:'/' //这个路径会用于出口的html文件引入关联文件的路劲  },  optimization: {  // 提取公用代码块，优化打包大小必备  splitChunks: {  cacheGroups: {  commons: {  name: "commons",//公用代码块名称  chunks: "initial",  minChunks: 1  }  }  }  },  plugins: [  new CleanWebpackPlugin(),  new HtmlWebpackPlugin({  title: 'Production'  }),  new webpack.ProvidePlugin({  \_: 'lodash' //全局变量整个lodash  // join: ['lodash', 'join'] //全局join 全部用lodash里面的join方法  })  ],  module: {  rules: [  {  test: /\.css$/,  use: ['style-loader', 'css-loader']  },  {  test: /\.(png|svg|jpg|gif)$/,  use: ['file-loader']  },  {  test: /\.(woff|woff2|eot|ttf|otf)$/,  use: ['file-loader']  }  ]  }  }; |

webpack.dev.js

|  |
| --- |
| const common = require('./webpack.common.js');  const merge = require('webpack-merge');  const webpack=require('webpack');  module.exports =env => {  // Use env.<YOUR VARIABLE> here:  console.log('来了') // 'local'  console.log('Production: ') // true  return merge(common, {  mode:'development',  devtool: 'inline-source-map',//开启这个可以在开发环境中调试代码  devServer: {  contentBase: "./dist",  compress: true,  port: 3000,  hot:true  },  plugins:[  new webpack.NamedModulesPlugin(),  new webpack.HotModuleReplacementPlugin()//这两个插件用于开发环境时，修改保存代码之后页面自动刷新  ]  });  } |

webpack.prod.js

|  |
| --- |
| const common = require('./webpack.common.js');  const merge = require('webpack-merge');  // const UglifyJsPlugin =require('uglifyjs-webpack-plugin');  module.exports = merge(common, {  devtool: 'source-map',//开启这个可以在生产环境中调试代码  // plugins:[  // new UglifyJsPlugin({ //尝试了一下代码压缩，但是会报错还没有找到原因  // sourceMap: false  // })  // ],  mode:'production'  }); |

然后在package.json文件里面增加两个scripts

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

"start": "webpack-dev-server --open --config webpack.dev.js",

此时package.json的完整内容是：

{

"name": "wbpack-test",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

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

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

"start": "webpack-dev-server --open --config webpack.dev.js"

},

"author": "",

"license": "ISC",

"devDependencies": {

"webpack": "^4.34.0",

"webpack-cli": "^3.3.4"

}

}

## 开始打包

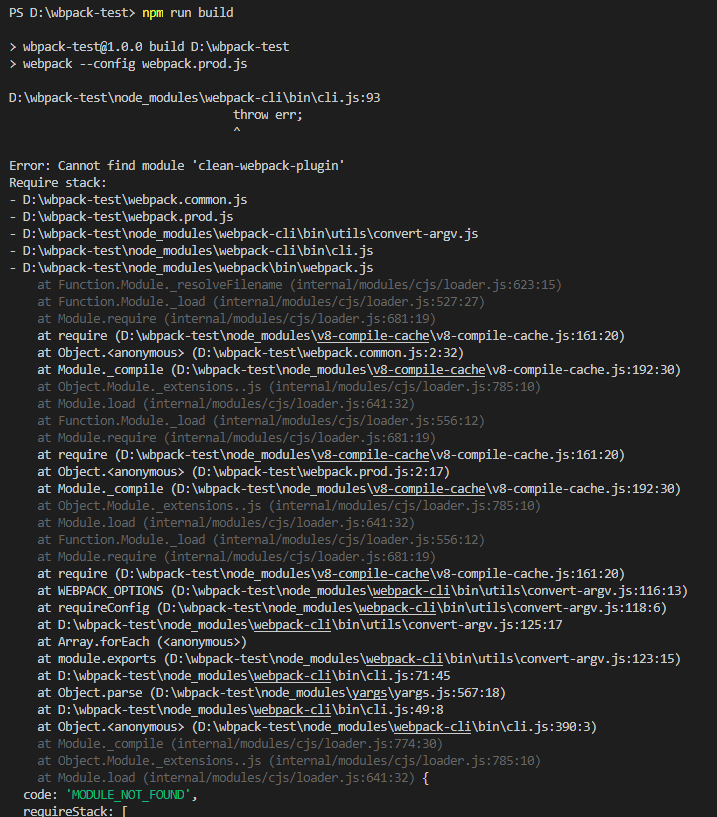
经过前面这么多的铺垫，终于到了我们的最主要的打包环节，话不多说，直接开始。

还是用命令提示符进入到webpack-test目录，然后运行命令:

npm run build (这个命令会告诉webpack打包工具去执行package.json里面的scripts对象的build的脚本，也就是说实际执行的是webpack --config webpack.prod.js) 。

这个命令还有一个问题就是 –config webpack.prod.js，为何会有这个选项呢，这是因为webpack打包工具会使用webpack.config.js这个文件当作默认配置文件，我们如果不想使用这个默认配置文件，可以使用--config这个配置，更改webpack的配置项文件。

下面是运行效果:



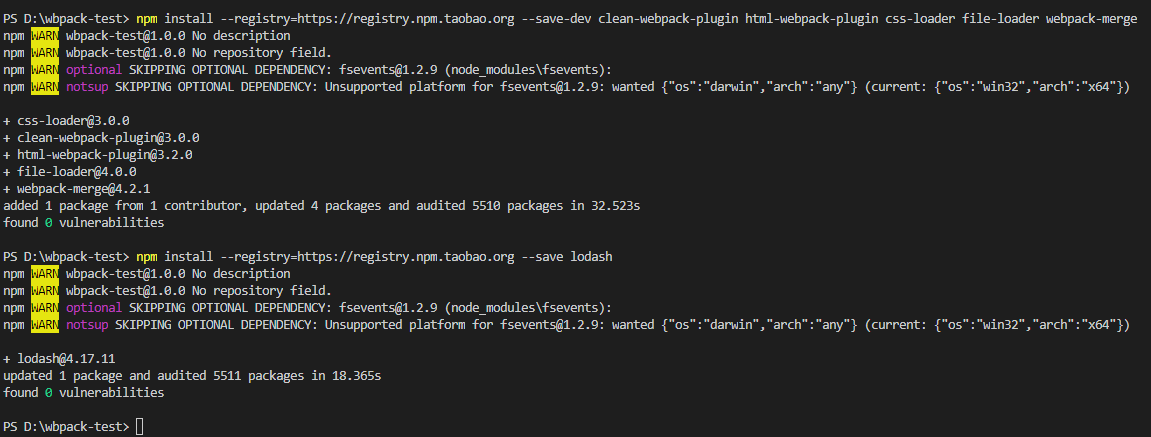
这么一长串东西，一看就是报错，为什么会报错呢，是因为代码里面的一些依赖项模块没有安装，所以下面我们来安装好我们配置项和代码里面各种依赖。

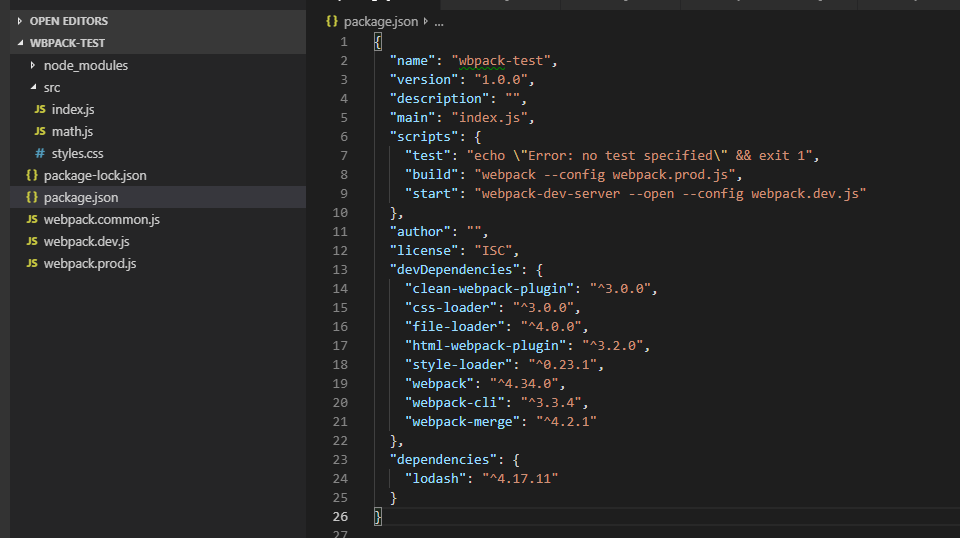
检查了一下代码和配置项文件，发现缺少的模块，然后我们执行脚本安装它们

install --registry=https://registry.npm.taobao.org --save-dev clean-webpack-plugin html-webpack-plugin css-loader file-loader style-loader webpack-merge

上面的脚本安装的是devDependencies

install --registry=https://registry.npm.taobao.org --save lodash

上面的脚本安装的是dependencies

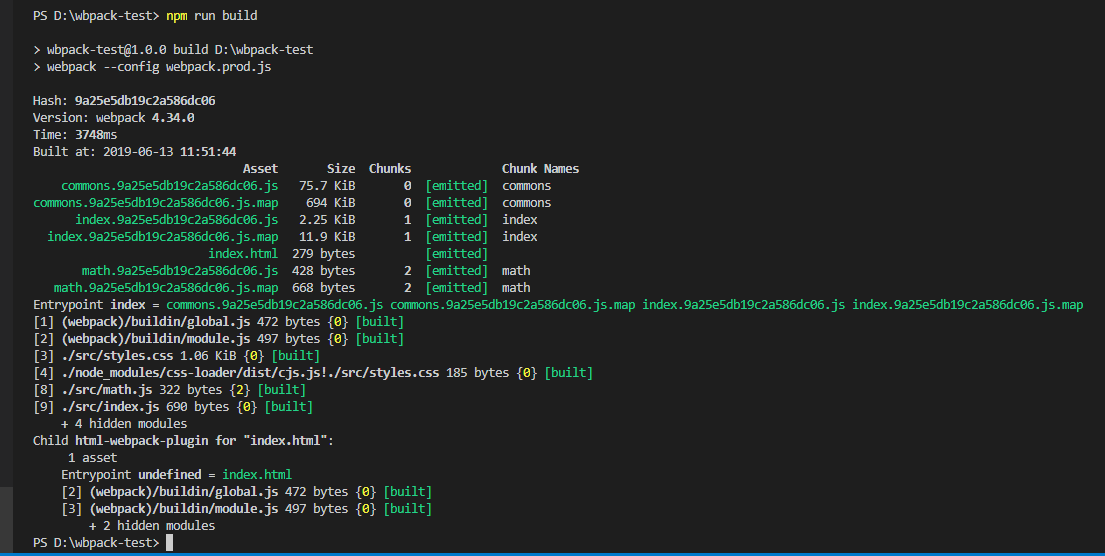
安装完之后package.json文件现在是这样

**devDependencie和 dependencies 的区别**

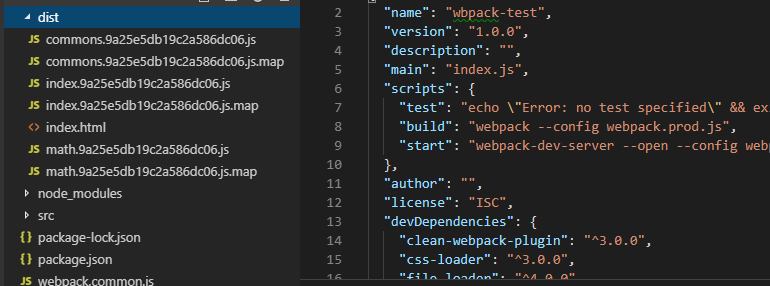
使用 --save-dev 安装的 插件，被写入到 package.json 文件的devDependencies 域里面去，而使用 --save 安装的插件，则是被写入dependencies 区块里面去。devDependencies  里面的插件只用于开发环境，不用于生产环境，dependencies  是需要发布到生产环境的。

比如我们写一个项目要依赖于jQuery，没有这个包的依赖运行就会报错，这时候就把这个依赖写入dependencies ；而我们使用的一些构建工具比如glup、webpack这些只是在开发中使用的包，上线以后就和他们没关系了，所以将它写入devDependencies。

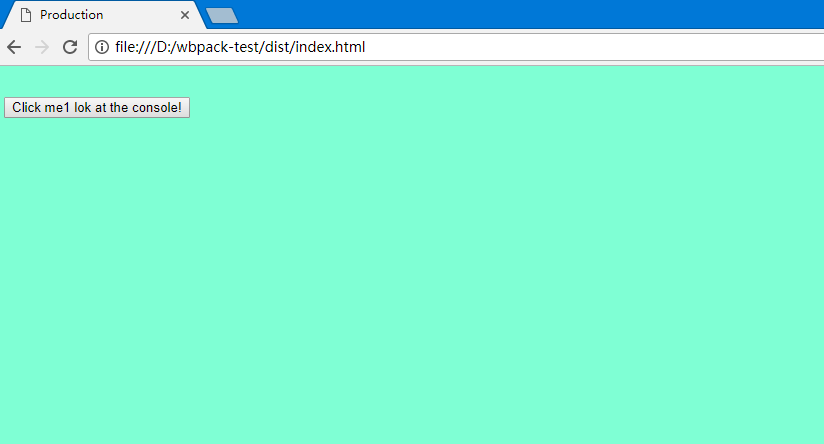
这时候再运行命令

npm run build

可以看到打包成功了，目录下也多出了打包之后的dist文件夹



打包完之后，要能用才行啊，浏览器中输入<file:///D:/wbpack-test/dist/index.html>



## 多入口文件打包

我们在src文件夹下面新增一个print.js文件

print.js

console.log(

'The print.js module has loaded! See the network tab in dev tools...'

);

Webpack.common.js更改

entry: {

index: './src/index.js',

print:'./src/print.js'

},

output: {

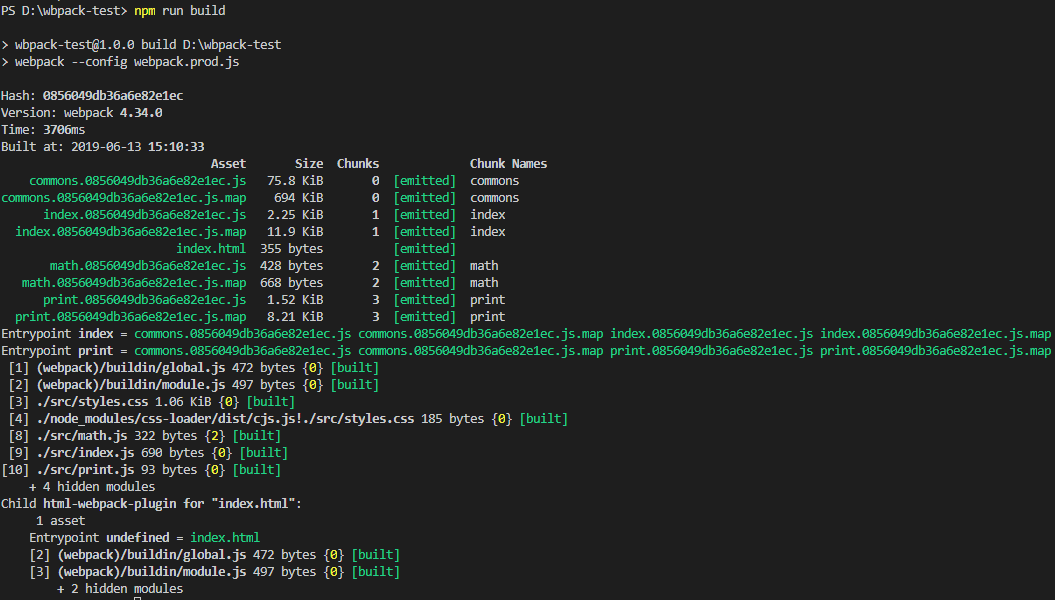
filename: '[name].[hash].js',

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

// publicPath:'/' //这个路径会用于出口的html文件引入关联文件的路劲

},

接下来运行命令

npm run build

可以看出，print.js也已经输出到dist文件夹了。