# 1 项目前期准备工作——项目初始化

## 搭建git仓库&本地仓库

### git仓库上的项目如何下载到本地

git clone 远程仓库的SSH地址

### 本地创建的项目如何上传到github？

//与github连接

git init

git remote add origin SSH地址 // 与远程仓库连接

git pull origin master //将master分支下载到本地

//上传项目（github不会上传空文件夹）

git add . //add与点之间有空格

git commit –m ‘描述’

git push

### 如何创建/切换分支？如何将分支内容与master合并

git checkout 分支名 //切换分支

git checkout –b 分支名 //-b表示分支未创建，创建分支并切换到该分支

//合并

### gitignore

如果不小心忘记创建.gitignore文件，而此时项目的代码已经提交到github上，其中包含了一些不想提交的内容，下面介绍一种可行的解决方法。<https://blog.csdn.net/luhu124541/article/details/82048357>

* 在根目录下执行命令 vim .gitignore
* 默认是不可编辑的，按下ins后即可进入编辑模式，输入不想被git追踪的文件规则，如.vscode，/node\_modules/，/dist/等
* 然后退出编辑模式esc或者再次按下INS键，按住shift+zz保存并退出
* 此时根目录下就会生成.gitignore文件

Vim 编辑器(以在命令行下编辑 .gitignore 文件为例)

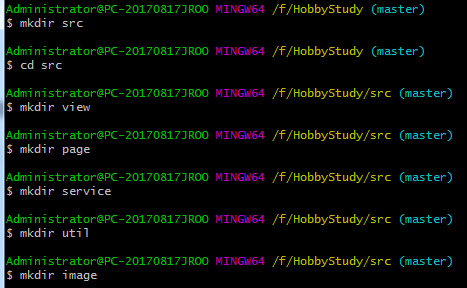
<https://blog.csdn.net/u014490083/article/details/78320055>

### git项目提交

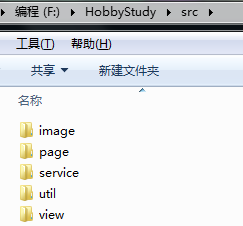
ls //查看当前目录下的文件

## 项目目录

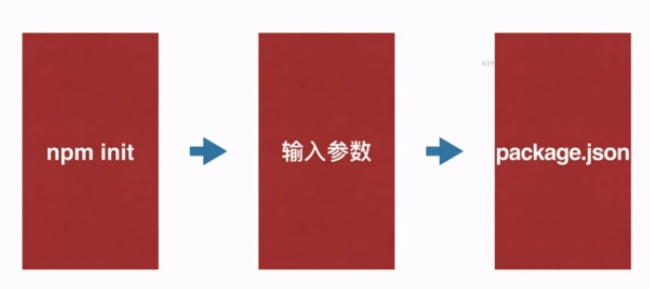
按如下目录依次创建

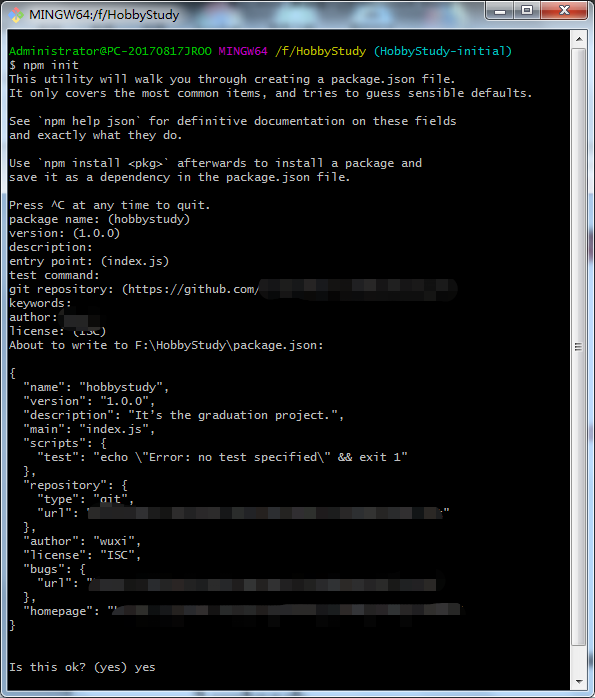


创建好的目录结构如下图



## npm初始化

npm init //这一步完成后会生成package.json文件



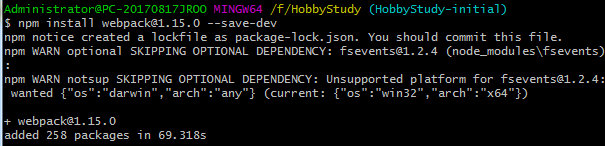
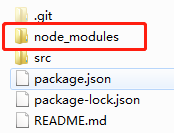
## 如何在项目中安装webpack

npm install webpack –g //全局安装

//在本地项目中安装

npm install [webpack@1.15.0](mailto:webpack@1.15.0) –save–dev

【注】安装1.15.0版本是因为高版本中object，default是保留字，IE8中会出错

## webpack介绍

参考文档：<https://github.com/webpack/docs/wiki/usage>  

### webpack常用命令

webpack

webpack –p // 用于线上发布时的打包，所有文件都会做最小化压缩

webpack --watch //用于监听文件的改变，但不能刷新浏览器，一般用于开发过程

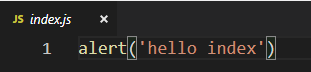
webpack --config webpack.config.js //改变默认配置文件

### webpack-dev-serve



## webpack编译过程

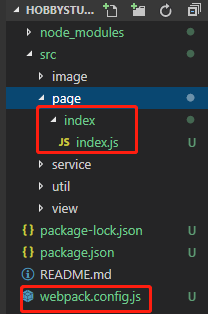
* 在src/page下创建文件夹index/index.js



* 执行webpack {入口文件路径} {输出文件路径}

webpack ./src/page/index/index.js ./dist/app.js

* 根目录下创建webpack.config.js文件



* 在webpack.config.js文件中写，根据项目更改入口文件和输出文件的路径

const path = require('path');

var config = {

entry:{

'index':[ './src/page/index/index.js'],

'login':['./src/page/login/index.js']

},

output: {

path: './dist',

filename: 'js/[name].js'

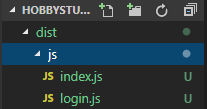
}

};

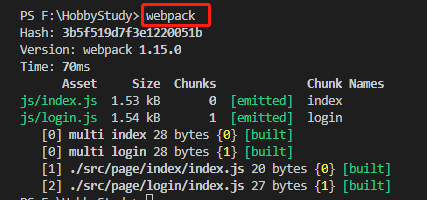
module.exports = config;

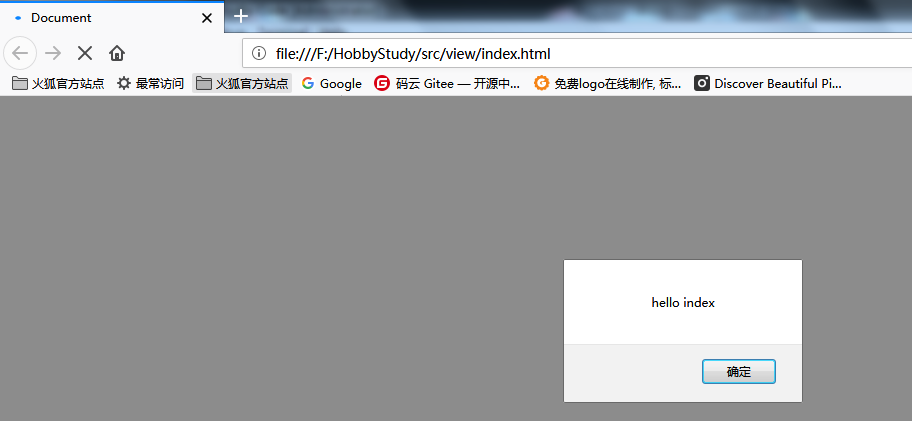
[name].js是因为index.js编译打包成app.js，login/index.js编译打包会把之前的app.js覆盖，为解决这一问题将app换成[name]

如何将编译打包的文件分类，前面加js/就会自动将js的文件放在js文件夹中



entry默认是单入口如何改变成多入口，可以改成对象形式，如上面所示

* 执行命令 webpack 
* 在view中创建index.html，引入

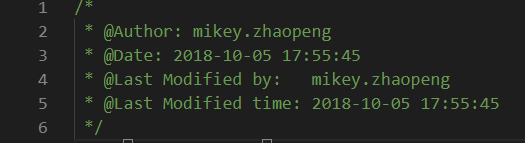
运行如图所示

说明编译成功

## fileheader插件

文件头部注明作者，最后更新日期等信息；

快捷键：ctrl+alt+i



如何修改默认作者

vscode中按cmd+，（逗号）进入用户设置界面

然后输入fileheader，更改作者名，重新启动vscode即可

## 对脚本的处理

### js用什么loader加载

自身带的loader

### 官方文档中的entry只有一个js，多个入口怎么办？



### output里分文件夹存放目标文件，怎么设置？

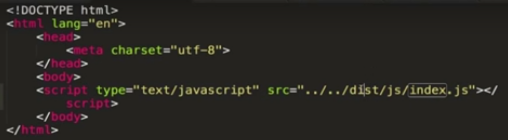


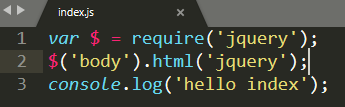
### jQuery引入方法？

* 第一种方法：

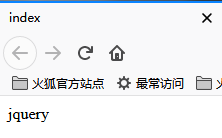
npm install jquery --save

view下创建index.html





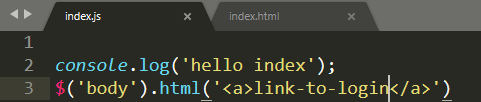
执行webpack



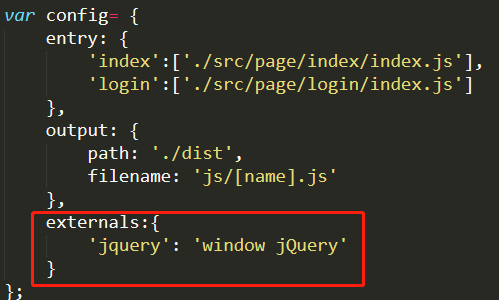
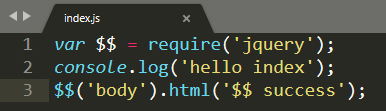
* 第二种

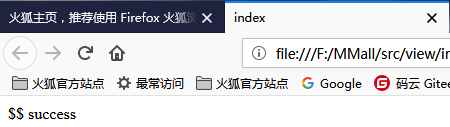
<https://www.bootcdn.cn/jquery/>





* 第三种

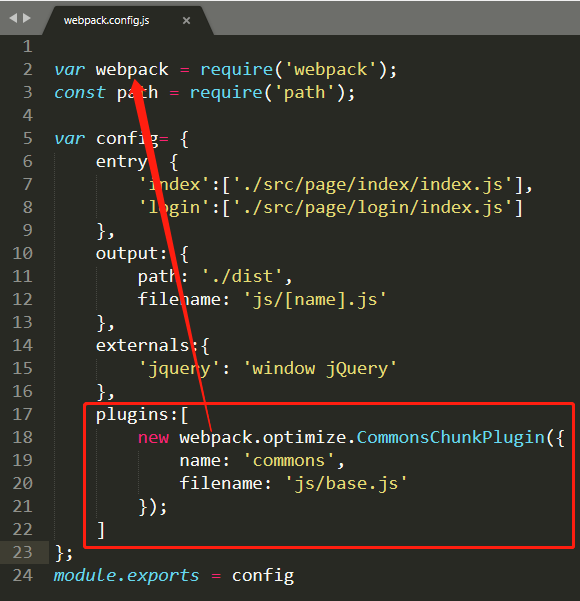
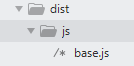
 



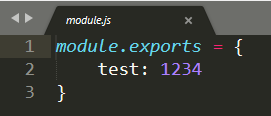
### 提取出公共模块怎么处理？



<https://webpack.docschina.org/plugins/commons-chunk-plugin/>

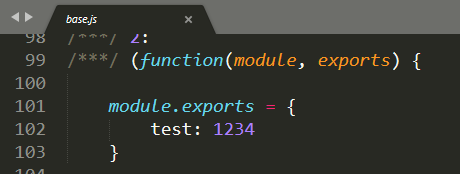
在src/page下创建module.js



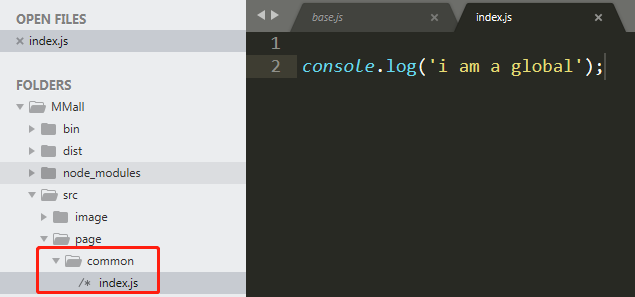
在index.js以及login/index.js中引入

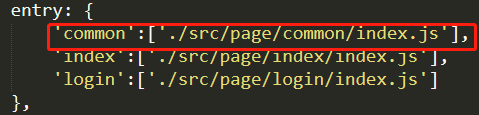


执行webpack后，dist/base.js中就会出现

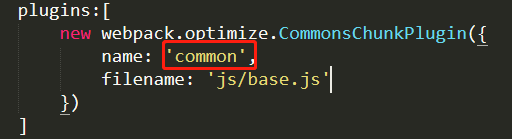


如果全局引入，则在每个文件中都require太麻烦

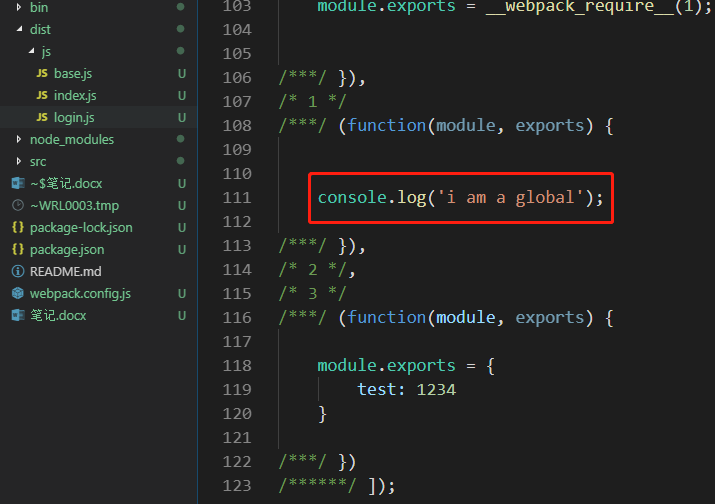




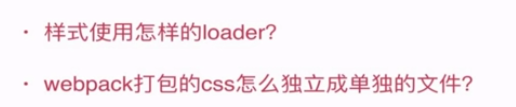
此时也会在dist/js中打包一个common，如果想将common打包到base.js中如何做？



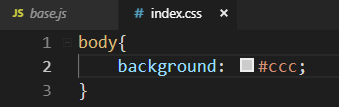
commons改为common

删除dist，执行webpack 

## 对样式的处理



在index文件夹下新建文件index.css



搜索“webpack对css的处理” <https://www.cnblogs.com/liuyt/p/5810042.html>

module: {

loaders: [

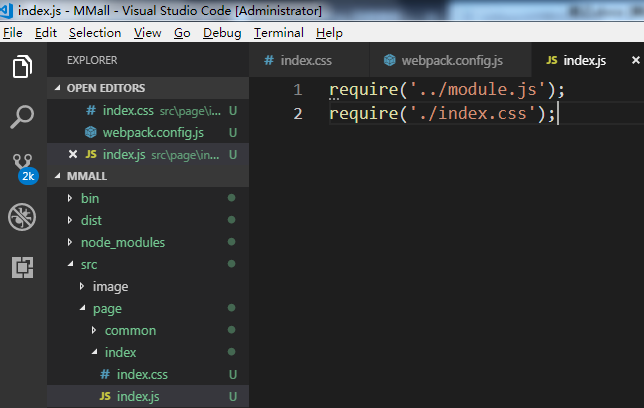
{ test: /\.css$/, loader: "style-loader!css-loader" }

]

},

在webpack.config.js中插入上述代码，style-loader!css-loader代表从右往左执行，先执行css-loader再执行style-loader

引入css

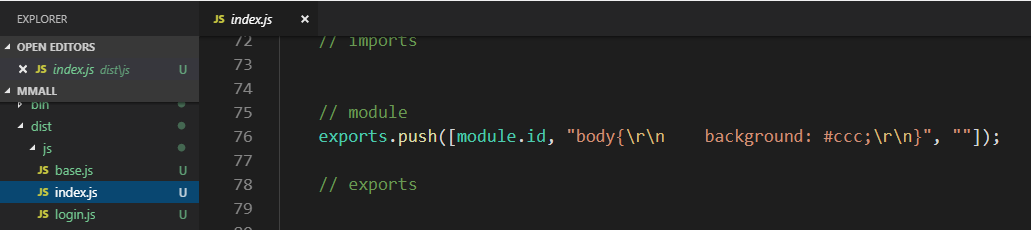


webpack会报错，因为没有安装这些loader

npm install css-loader style-loader --save-dev

查看package.json





此时打包后会吧css放到index.js中，但这样会影响css的加载，等js加载完才会加载css则会有一段空白，那么如何将css单独打包

搜索“webpack css单独打包”

<https://www.cnblogs.com/zhongxia/p/5834089.html>

1. 安装extract-text-webpack-plugin

// use npm

npm install extract-text-webpack-plugin@1.1.0 --save-dev

2. 配置

加载器里面写好插件的配置（使用什么加载器），在webpack的 plugins 里面设置抽离出来的CSS文件名叫什么。

var webpack = require('webpack');

const path = require('path');

var ExtractTextPlugin = require('extract-text-webpack-plugin');

var config= {

entry: {

'common':['./src/page/common/index.js'],

'index':['./src/page/index/index.js'],

'login':['./src/page/login/index.js']

},

output: {

path: './dist',

filename: 'js/[name].js'

},

externals:{

    'jquery': 'window jQuery'

},

module: {

loaders: [

{ test: /\.css$/, loader: ExtractTextPlugin.extract('style-loader', 'css-loader','less-loader') }

]

},

plugins:[

new webpack.optimize.CommonsChunkPlugin({

name: 'common',

filename: 'js/base.js'

}),

new ExtractTextPlugin("css/[name].css")

]

};

module.exports = config

## html-webpack-plugin

<https://www.npmjs.com/package/html-webpack-plugin>

npm i --save-dev html-webpack-plugin

webpack.config.js

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

var config= {

plugins:[

//打包一个html文件

new HtmlWebpackPlugin({

template: './src/view/index.html',

filename: 'view/index.html',

inject: true,

hash: true,

chunks: ['common','index']

}),

new HtmlWebpackPlugin({

template: './src/view/login.html',

filename: 'view/login.html',

inject: true,

hash: true,

chunks: ['common','login']

})

]

};

module.exports = config

如果按这样写，多个页面代码很冗余，所以将其进行函数封装

//获取html-webpack-plugin参数的方法

var getHtmlConfig = function(name){

return {

template: './src/view/' + name + '.html',

filename: 'view/' + name + '.html',

inject: true,

hash: true,

chunks: ['common', name]

}

};

new HtmlWebpackPlugin(getHtmlConfig('index')),

new HtmlWebpackPlugin(getHtmlConfig('login'))

## jQuery如何引入不需要每次都输入？

安装html-loader

npm install html-loader –save-dev

在src/view下新建文件夹layout存放公共文件，里面新建head.html

//head.html

<head>

<meta charset="UTF-8">

</head>

在index.html中将head标签删除，然后添加<%= require(./layout/head.html)%>

## 处理图片，文字字体等

在 webpack 中引入图片需要依赖 url-loader 这个加载器。

npm install url-loader --save-dev

npm install file-loader –save-dev

参考文档：

<https://blog.csdn.net/x550392236/article/details/78427388>

module: {

loaders: [

{

test: /\.(gif|png|jpg|woff|svg|eot|ttf)\??.\*$/,

loader: 'url-loader?limit=8192&name=images/[hash:8].[name].[ext]'

}

]

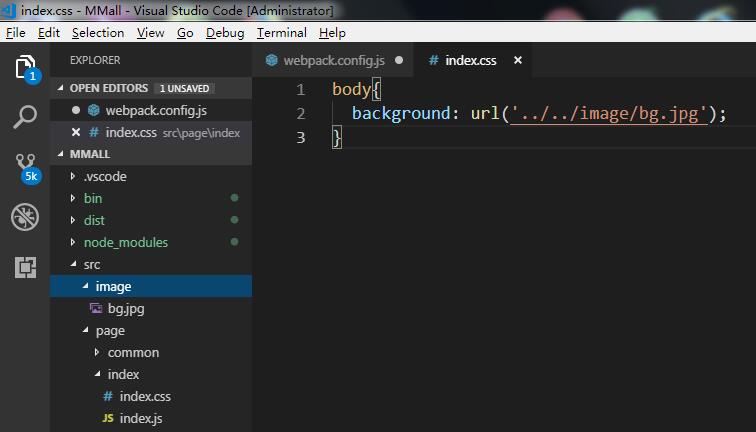
}

test 属性代表可以匹配的图片类型，除了 png、jpg 之外也可以添加 gif 等，以竖线隔开即开。

loader 后面 limit 字段代表图片打包限制，这个限制并不是说超过了就不能打包，而是指当图片大小小于限制时会自动转成 base64 码引用。上例中大于8192字节的图片正常打包，小于8192字节的图片以 base64 的方式引用。

url-loader 后面除了 limit 字段，还可以通过 name 字段来指定图片打包的目录与文件名：

上例中的 name 字段指定了在打包根目录（output.path）下生成名为 images 的文件夹，并在原图片名前加上8位 hash 值。

【bug】![d:\Documents\Tencent Files\371081268\Image\C2C\[TA4ODHCS4PQE~PHE](VTJB.jpg](data:image/jpeg;base64,)

## webpack-dev-server

### 安装和配置

npm install webpack-dev-server@1.16.5 --save-dev

npm install [webpack-dev-server@1.16.5](mailto:webpack-dev-server@1.16.5) –g

### 应用场景和使用方法

entry: {

'common':['./src/page/common/index.js',

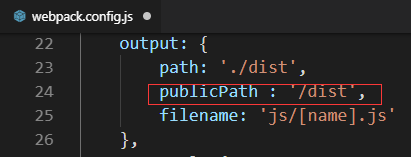
'webpack-dev-server/client?http://localhost8088/'

],

}

执行

webpack-dev-server --inline --port8088



### 环境变量的设置和读取方法

var WEBPACK\_ENV = process.env.WEBPACK\_ENV || 'dev';

console.log(WEBPACK\_ENV);

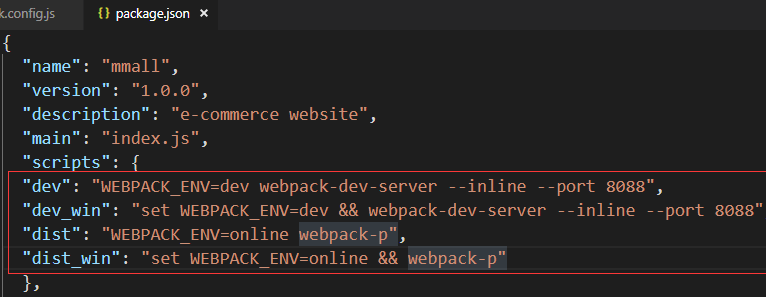
if('dev' === WEBPACK\_ENV){

config.entry.common.push('webpack-dev-server/client?http://localhost8088/');

}

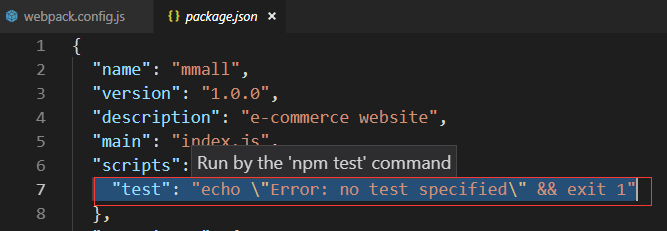
执行命令

WEBPACK\_ENV=dev webpack-dev-server --inline –port 8088（mac系统）



### webpack命令和npm自定义命令的结合

上述代码太长不容易记住，在package.json文件中做修改，替换test



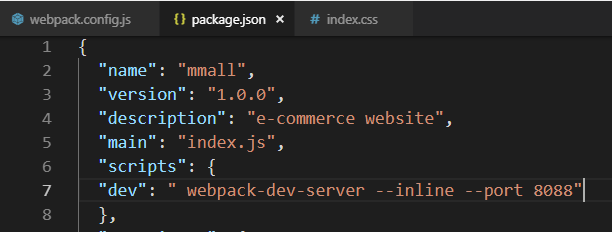
"dev": "WEBPACK\_ENV=dev webpack-dev-server --inline --port 8088",

"dev\_win": "set WEBPACK\_ENV=dev && webpack-dev-server --inline --port 8088",

"dist": "WEBPACK\_ENV=online webpack -p",

"dist\_win": "set WEBPACK\_ENV=online && webpack -p"

此时可以执行命令npm run dev

【bug】

"dev": " webpack-dev-server --inline --port 8088"

此时运行npm run dev可以成功运行