webpack中文文档-----------》https://doc.webpack-china.org/configuration/entry-context/

**-save-dev是指将依赖的包裹，挂载到本地开发项目中去，可以在package.json文件中得以体现出来。**  
-save是指将包信息添加到dependencies，表示你发布时依赖的包裹。

npm uninstall .....卸载

npm run dev 开发环境下运行的命令，npm run build 是生产环境下运行的命令

在webpack中使用**require(‘文件路径’)**或**import ‘文件路径’** --->效果都是一样的，最终还是以

**\_\_webpack\_require\_\_(‘文件路径’) 的方式引入;因此在webpack中还是推荐使用require(‘路经’)的方式引入**

**零、遇到二次开发的项目**

**①使用npm run dev检查项目中是否含有webpack打包所需要的node\_modules等模块**

**②如果没有，打开package.json查看项目用到的那些模块，然后进行安装所需要的模块**

**③一般情况下少的基础模块：webpack、webpack-cli、html-webpack-plugin、html-loader、img-loader、css-loader**

一、创建一个新项目

①装好node和npm之后,使用webpack -v查看webpack的版本，如果没有安装webpack进入下一步

②使用webpack npm install -g webpack@XXXX在全局开始安装指定版本的webpack

③使用cd 项目文件目录 命令进入项目文件的根目录之后，使用npm install webpack --save安装全局webpack

————》**package.json各个模块的版本**文件 和node\_modules模块文件

和**package-lock.json记录当前状态下实际安装的各个npm package的具体来源和版本号**

④使用npm init +多个回车 命令 初始化一个package.json文件

⑤使用mkdir src 和mkdir dist命令，新建一个src和dist文件夹，放项目的源文件 和打包后的文件

二、使用软件打开项目

①新建一个index.html项目入口文件

②新建项目的webpack配置文件，文件名为webpack.config.js，其简单的配置：

module.exports={

devtool: 'inline-source-map', //追踪错误文件位置

entry:"./src/script/index.js", //打包入口文件

output:{

filename:". /js/bundle.js" //打包后的文件目录

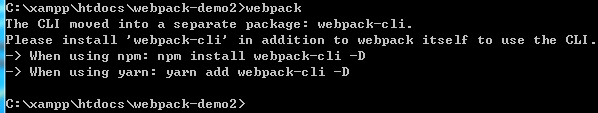
}

}

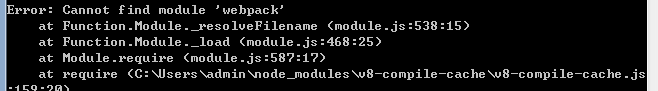
**根据webpack.config.js文件的配置，创建对应的入口文件；**

③在windows系统中运行webpack ---->webpack ;在苹果系统中 -----> npm run webpack 运行webpack

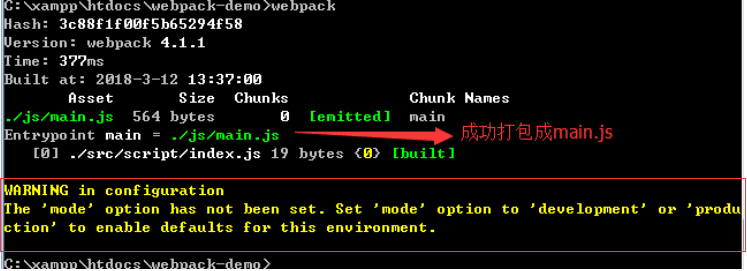
在windows系统中使用webpack命令运行webpack4.10版本的时需要安装webpack-cli



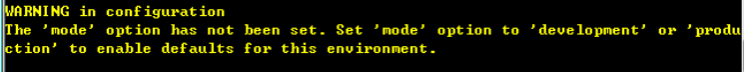
或



现在使用npm install webpack-cli --save-dev安装全局的webpack-cli之后;再次使用webpack进行打包就可以了。



④第③步中报出一个警告：意思就是没有选择开发模式和生产模式



需要在package.js中scripts选项中的添加这样的一段代码:

"scripts": {

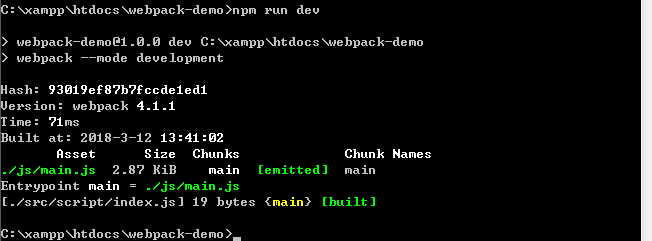
"test": "echo \"Error: no test specified\" && exit 1",

**"dev": "webpack --mode development",**

**"build": "webpack --mode production"**

},

之后：**再运行npm run dev 开发环境下运行的命令，npm run build 是生产环境下运行的命令**



⑤在package.json文件中配置**npm run dev和 npm run build** 打包命令；

"scripts": {

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

"dev": "webpack --mode development",

"build": "webpack --mode production",

"webpack": "webpack --config webpack.config.js --progress --display-module --colors --display-reason"

},

webpack --config webpack.config.js ------>指定webpack的配置文件

--progress --------->显示打包过程

--display-module --------->显示打包的模块

--colors ------------>显示打包后字是彩色的

--display-reason ----------->显示打包的原因

三、当入口是多个时：-------->多页面应用

1)、打包多个入口文件

module.exports={

entry:{ //是个对象

**"entry-a":"./src/script/entry-A.js", //key**

**"entry-b":"./src/script/entry-B.js", //key**

},

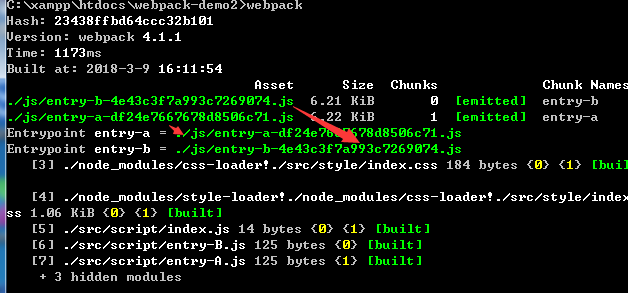
output:{

//这里的name就是上入口对象下是key，chunkhash是入口文件打包后的hash

**filename:"./js/[name]-[chunkhash].js"**

},

}



2)、使用html-webpack-plugin插件 动态生成html文件

---->参考解析https://doc.webpack-china.org/plugins/html-webpack-plugin/#src/components/Sidebar/Sidebar.jsx

①使用**npm install html-webpack-plugin --save-dev --**---》如果出现路径错误，使用npm uninstall webpack -g 卸载webpack 再重新装全局的webpack

②在webpack.config.js文件中这样配置

**var htmlWebpackPlugin=require('html-webpack-plugin');** //1.引用html-webpack-plugin

module.exports={

entry:{ //打包入口文件

"entry-a":"./src/script/entry-A.js",

"entry-b":"./src/script/entry-B.js",

},

output:{ //输出文件

"filename":"./js/[name]-[chunkhash].js",

"publicPath":'eTalker.cc/public/index'

},

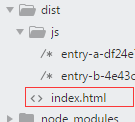
**plugins: [ //引用htmlWebpackPlugin**

**new htmlWebpackPlugin() //2.初始化 html-webpack-plugin**

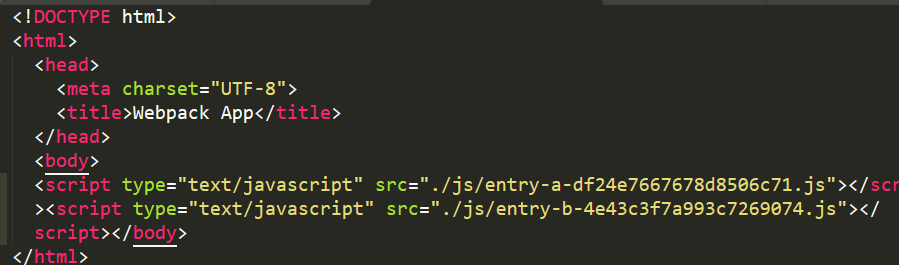
**]**

}

运行webpack 就会动态生成一个index.html



里面引用着打包好的js文件



**③当给htmlWebpackPlugin传参时；**

plugins: [ //引用htmlWebpackPlugin

**new htmlWebpackPlugin({**

**//指向项目的入口文件；这时就不需要在index.html入口文件中引入指定的js了**

**"template": 'index.html' , //模板文件**

**"filename":"index-[hash]-[name].html",//指定打包后的html文件名**

**"inject":' ',//当值为某个标签名时，inject:'head' ---->指定生成js脚本放在./dist/index.html指定的 head标签内 或：值为footer时，生成js脚本就会放在</body>之前,所有body内的元素之后**

**'title':'is Good',**

**"minify":{**

**//minify中的参数参考: https://github.com/kangax/html-minifier#options-quick-reference**

**"removeComments":true, //删除html注释**

**" collapseWhitespace":true, //删除html空格**

**}**

**})**

]

在项目的入口文件index.html或模板文件中，可以使用**<%=**  htmlWebpackPlugin.options.title  **%>** 这种 格式；来获取**new htmlWebpackPlugin**中的一些参数；

遍历：htmlWebpackPlugin.options

<% for(var key in htmlWebpackPlugin.files ){ %>

<%= key %> -----><%= htmlWebpackPlugin.files[key] %>

<% } %>

<script src="<%= htmlWebpackPlugin.files.js[0] %>" > </script>

<script src="<%= htmlWebpackPlugin.files.js[1] %>" > </script>

<hr>

<% for(var key in htmlWebpackPlugin.options ){ %>

<%= key %> -----><%= htmlWebpackPlugin.options[key] %>

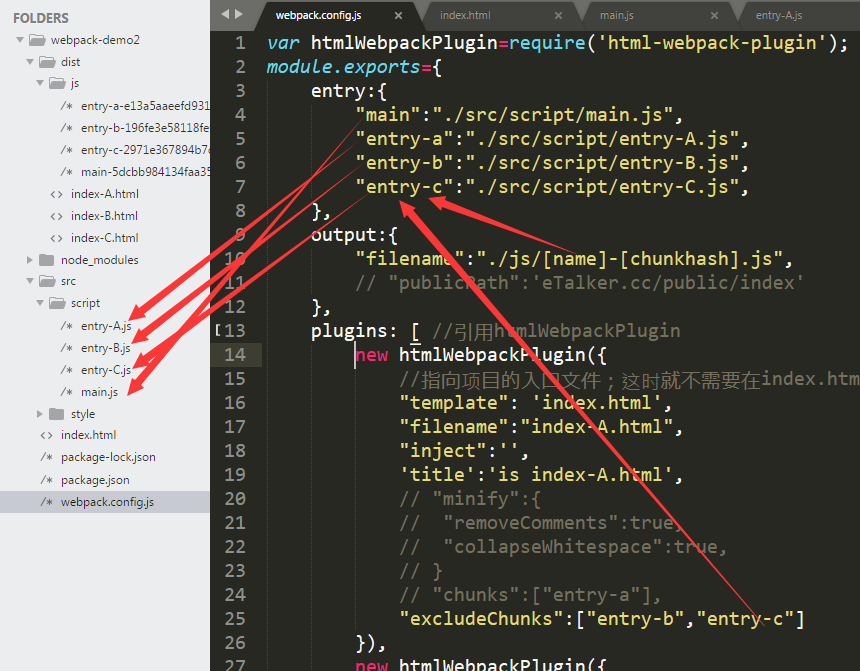
<% } %>

这样就可以在项目的入口文件或模板文件中，指定的位置添加指定的js文件了.

④同一个html模板，生成多个不同的html页面，并且让不同的页面还引用自己的js文件

|  |  |
| --- | --- |
| **var htmlWebpackPlugin=require('html-webpack-plugin');**  module.exports={  entry:{ "entry-a":"./src/script/entry-A.js",  "entry-b":"./src/script/entry-B.js",  "entry-c":"./src/script/entry-C.js",  },  output:{  "filename":"./js/[name]-[chunkhash].js",  // "publicPath":'eTalker.cc/public/index'  },  plugins: [  **new htmlWebpackPlugin({**  "template": 'index.html',  "filename":"index-A.html",  "inject":'footer',  'title':'is index-A.html',  // "minify":{  // "removeComments":true,  // "collapseWhitespace":true,  // }  // "chunks":["entry-a"],  "excludeChunks":["entry-b","entry-c"]  }),  **new htmlWebpackPlugin**({  "template": 'index.html',  "filename":"index-B.html",  "inject":'footer',  'title':'is index-B.html',  // "minify":{  // "removeComments":true,  // "collapseWhitespace":true,  // }  // "chunks":["entry-b"],  "excludeChunks":["entry-a","entry-c"]  }), | **new htmlWebpackPlugin**({  "template": 'index.html',  "filename":"index-C.html",  "inject":'footer',  'title':'is index-C.html',  // "minify":{  // "removeComments":true,  // "collapseWhitespace":true,  // }  // "chunks":["entry-c"],  "excludeChunks":["entry-a","entry-b"]    })  ]  }  **chunks**是用来指定页面引用指定的js文件的；  **excludeChunks**是用来指定页面不能引用那些js文件  Html模板：  <!DOCTYPE html>  <html>  <head>  <title><%= htmlWebpackPlugin.options.title%></title>  <meta charset="utf-8">  </head>  <body>  <% for(var key in htmlWebpackPlugin.files ){ %>  <%= key %> <%= htmlWebpackPlugin.files[key] %>  <% } %> <hr>  <% for(var key in htmlWebpackPlugin.options ){ %>  <%= key %> <%= htmlWebpackPlugin.options[key] %>  <% } %>  </body>  </html> |

⑤让公用的js分别放置在不同的页面中使用，并且让不同的页面还引用自己的js文件



|  |  |
| --- | --- |
| Webpack.config.js  var htmlWebpackPlugin=require('html-webpack-plugin');  module.exports={  entry:{  "main":"./src/script/main.js",  "entry-a":"./src/script/entry-A.js",  "entry-b":"./src/script/entry-B.js",  "entry-c":"./src/script/entry-C.js", },  output:{  "filename":"./js/[name]-[chunkhash].js",  // "publicPath":'eTalker.cc/public/index' },  plugins: [  new htmlWebpackPlugin({  "template": 'index.html',  "filename":"index-A.html",  "inject":'',  'title':'is index-A.html',  // "minify":{ "removeComments":true,  // "collapseWhitespace":true, } ,  // "chunks":["entry-a"],  "excludeChunks":["entry-b","entry-c"]  }),  new htmlWebpackPlugin({  "template": 'index.html',  "filename":"index-B.html",  "inject":'',  'title':'is index-B.html',  "excludeChunks":["entry-a","entry-c"]  }), | new htmlWebpackPlugin({  "template": 'index.html',  "filename":"index-C.html",  "inject":'',  'title':'is index-C.html',  "excludeChunks":["entry-a","entry-b"]  })  ]  }  **同一个HTML文件**  <!DOCTYPE html>  <html>  <head>  <title><%= htmlWebpackPlugin.options.title%></title>  <meta charset="utf-8">  </head>  <body>  <script>  <%= compilation.assets[  htmlWebpackPlugin.files.chunks.main.entry.substr(  htmlWebpackPlugin.files.publicPath.length)].source() %>  </script>  <% for(var k in htmlWebpackPlugin.files.chunks ){ %>  <% if(k!=='main'){ %>  <script src="  <%= htmlWebpackPlugin.files.chunks[k].entry%>  "></script>  <% } %>  <% } %>  </body>  </html> |

⑥不同html模板，生成多个不同的html页面:

只需要修改htmlWebpackPlugin中的template值定的html模板即可

1. 处理其他的文件

--->众多loader  **https://doc.webpack-china.org/loaders 或 https://www.npmjs.com/package/img-loader**

1）处理js文件 ------->需要安装babel-loader babel-core

①npm install --save-dev babel-loader babel-core //官网http://babeljs.io/docs/setup/#installation

②需要在webpack.config.js中配置

module: {

rules: [

{ test: /\.js$/, exclude:"/node\_modules/", include:"/src/",  **loader: "babel-loader"** }

]

}

exclude ---->是不让那些文件使用babel-loader ；include ---->是让那些文件使用babel-loader ；

|  |  |
| --- | --- |
|  |  |

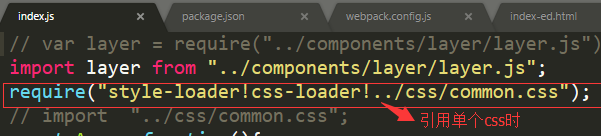
③使用**require('路径')** || **import var from ‘路径’** || **impor ‘路径’**方法,在打包的入口文件index.js中引用即可。 2）处理样式文件

①使用css-loader和style-loader 处理css文件

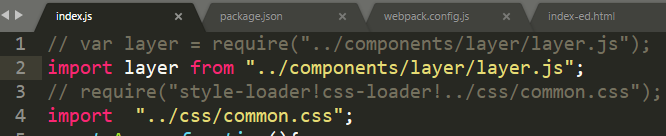
1.使用npm install css-loader style-loader --save-dev命令安装css-loader和style-loader

2.在index.js文件中引用

使用require('style-loader!css-loader!文件相对路径'); \*注意\* 先有style-loader后有css-loader



或（推荐）



3.配置webpack.config.js

module: {

rules: [

{ test: /\.js$/,

exclude:path.resolve(\_\_dirname,"/node\_modules/"),

include:path.resolve(\_\_dirname,"/src/"),

loader: "babel-loader" },

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

]

}

②使用postcss-loader自动处理css3中样式的前缀

1.安装：npm install postcss-loader --save-dev

2.安装：npm install autoprefixer --save-dev

3.配置webpack.config.js

module: {

rules: [

{ test: /\.js$/,exclude:path.resolve(\_\_dirname,"/node\_modules/"),

include:path.resolve(\_\_dirname,"/src/"),loader: "babel-loader" },

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

]

}

1. 新建并配置postcss.config.js **module.exports = {**

**plugins:[**

**require('autoprefixer')({**

**browsers:['last 5 versions']**

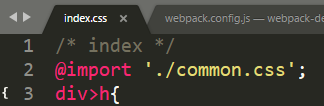
**}),**

**]**

**}**

③使用importloaders=1配置webpack.config.js处理@import引入的css文件

1.使用@import “./new.css”在index.css引入css文件



2.配置webpack.config.js

module: {

rules: [ { test: /\.js$/,

exclude:path.resolve(\_\_dirname,"/node\_modules/"),

include:path.resolve(\_\_dirname,"/src/"),

loader: "babel-loader" },

**{ test: /\.css$/,loader: "style-loader!css-loader?importLoaders=1!postcss-loader"}**

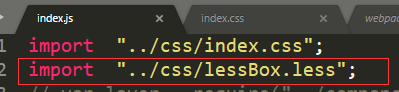
]

}

④使用less-loader处理less文件

1.安装 npm install less-loader --save-dev less

2.在index.js中引入less文件



3.配置webpack.config.js

module: {

rules: [ { test: /\.js$/,

exclude:path.resolve(\_\_dirname,"/node\_modules/"),

include:"/src/",

loader: "babel-loader" },

**{test:/\.less/,loader:"style-loader!css-loader!less-loader!postcss-loader"}**

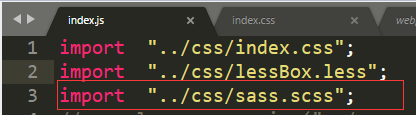
]

}

⑤使用sass-loader处理scss文件

1.安装 npm install sass-loader node-sass webpack --save-dev

2.在index.js中引入scss文件



3.配置webpack.config.js

module: { rules: [ { test: /\.js$/,

exclude:path.resolve(\_\_dirname,"/node\_modules/"),

include:path.resolve(\_\_dirname,"/src/"),

loader: "babel-loader" },

{test:/\.css$/,loader:"style-loader!css-loader?importLoaders=1!postcss-loader"},

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

**{test:/\.scss$/,loader:"style-loader!css-loader!sass-loader!postcss-loader"},**

]

}

1. 处理模板文件：

①处理html模板

1.安装html-loader npm i -D html-loader

2.新建layer.html模板文件，并在模板逻文件layer.js中引用和抛出

**import tpl from "./layer.html";**

function layer(){

return {

'name':"这是layer组建js",

'tpl':tpl

}

}

**export default layer;**

3.在入口文件index.js中;收入html字符串模板

import Layer from "../components/layer/layer.js";

const App = function(){

var dom=document.querySelector('#app');

var layer = new Layer();

dom.innerHTML = layer.tpl;

}

new App();

4.配置webpack.config.js

module: {

rules: [

{ test: /\.js$/,

exclude:path.resolve(\_\_dirname,"/node\_modules/"),

include:path.resolve(\_\_dirname,"/src/"),

loader: "babel-loader" },

{test:/\.css$/,loader:"style-loader!css-loader?importLoaders=1!postcss-loader"},

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

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

**{test: /\.html$/,loader:"html-loader"},**

]

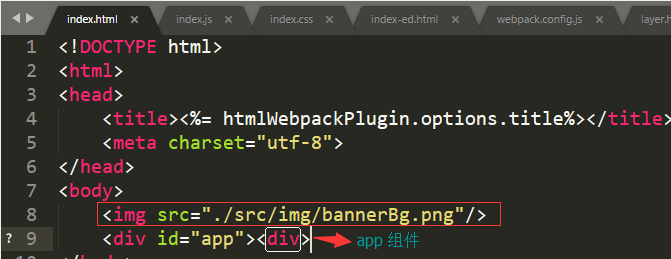
}

1. 处理.png|.jpg|.gif|结尾的图像文件

①使用file-loader对图像最基本的处理

1.安装 file-loader npm install --save-dev file-loader

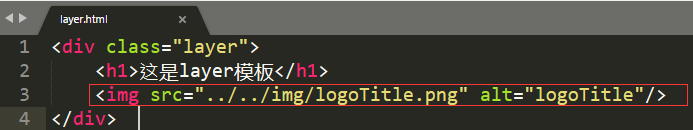
2.在项目的文件index.html中直接以相对路径的形式引用 ------>注意:要在组件之前使用



或者是在css文件中引用图片：



或者是在组件内直接以相对路径的形式引用：



3.配置webpack.config.js

module: {

rules: [

{ test: /\.js$/,

exclude:path.resolve(\_\_dirname,"/node\_modules/"),

include:path.resolve(\_\_dirname,"/src/"),

loader: "babel-loader" },

{test:/\.css$/,loader:"style-loader!css-loader?importLoaders=1!postcss-loader"},

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

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

{test:/\.html$/,loader:"html-loader"},

**{test:/\.(png|jpg|gif)$/i,loader:'file-loader', options:{name:'dist/[name]-min.[ext]'} }**

]

}

options中的参数：https://doc.webpack-china.org/loaders/file-loader/

②使用url-loader 对文件进行选择性的压缩成base64编码格式的文件

**\*/ limit-->当文件小于limit指定的大小时，就会转成base64编码,否则就会交给file-loader进行打包处理/\***

**图片转成base64编码后的文件，浏览器就不会发送htpp请求该图片了。**

1.安装url-loader npm install --save-dev url-loader

2.配置webpack.config.js

module: {

rules: [

{ test: /\.js$/,

exclude:path.resolve(\_\_dirname,"/node\_modules/"),

include:path.resolve(\_\_dirname,"/src/"),

loader: "babel-loader" },

{test:/\.css$/,loader:"style-loader!css-loader?importLoaders=1!postcss-loader"},

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

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

{test:/\.html$/,loader:"html-loader"},

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

//loader:'file-loader',

**loader:'url-loader',**

options:{

**limit: 20000,** //20k （以字节为单位）

name:'img/[name]-min.[ext]',

}

}

]

}

③使用img-loader对图片进行压缩

1.安装img-loader npm install img-loader --save-dev

2.配置webpack.config.js

module: {

rules: [

{ test: /\.js$/,

exclude:path.resolve(\_\_dirname,"/node\_modules/"),

include:path.resolve(\_\_dirname,"/src/"),

loader: "babel-loader" },

{test:/\.css$/,loader:"style-loader!css-loader?importLoaders=1!postcss-loader"},

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

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

{test:/\.html$/,loader:"html-loader"},

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

loaders:[

'url-loader?limit=30000&name=img/[name]-min.[ext]',

**'img-loader'**

]

}

]

}

**五、使用webpack-server进行打包；**

①使用npm i webpack-dev-server --save-dev命令安装一个全局的 webpack-dev-server 服务，

并挂在到本地项目中，在配置文件package.json中可以看到-->



②在package.json文件中配置start启动项； ----->配置默认的是localhost下的8080端口

③配置好第②步后，可以在终端先使用npm run dev 或npm run build进行打包，之后运行npm start 命令

"scripts": {

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

"dev": "webpack --mode development",

"build": "webpack --mode production" --watch, //实时监听内容是否发生改变

"webpack": "webpack --config webpack.config.js --progress --display-module --colors --display-reason",

"start": "webpack-dev-server --entry ./src/script/index.js --output-filename ./dist/js/main.js"

},

并在浏览器中运行http://localhost:8080/ ----->--watch监听当内容是否发生改变，若是发生改变， webpack会重新打包--->之后浏览器再次刷新即可

④在webpack.config.js中配置打包后默认运行的端口号



以上配置完之后，可以先运行npm run dev 在运行npm start

**i ｢wds｣: Project is running at http://localhost:3000/**

**i ｢wds｣: webpack output is served from /**

webpack安装技巧：

一次性可以安装多个loader

npm i style-loader css-loader postcss-loader autoprefixer html-loader less-loader sass-loader file-loader url-loader img-loader --save-dev



安装结果：

