# [webpack4 多入口，多页面项目构建案例](https://segmentfault.com/a/1190000014984842)

[webpack](https://segmentfault.com/t/webpack/blogs) [webpack-dev-server](https://segmentfault.com/t/webpack-dev-server/blogs)   6.6k 次阅读  ·  读完需要 24 分钟 2018-05-23 发布

趁工作之余从零构建了一个webpack4.x多页面应用程序。过程中也遇到一些坑，就记录下来了。

**webpack核心概念**

* Entry：入口，Webpack 执行构建的第一步将从 Entry 开始。
* Module：模块，在 Webpack 里一切皆模块，一个模块对应着一个文件。Webpack 会从配置的 Entry 开始递归找出所有依赖的模块。
* Chunk：代码块，一个 Chunk 由多个模块组合而成，用于代码合并与分割。
* Loader：模块转换器，用于把模块原内容按照需求转换成新内容。
* Plugin：扩展插件，在 Webpack 构建流程中的特定时机注入扩展逻辑来改变构建结果或做你想要的事情。
* Output：输出结果，在 Webpack 经过一系列处理并得出最终想要的代码后输出结果。

项目的运行主要围绕的就是这几大块

首先看一下构建后目录  
├── build  
│ ├── webpack.base.conf.js  
│ ├── webpack.dev.conf.js  
│ ├── webpack.prod.conf.js  
│ ├── webpack.rules.conf.js  
├── src  
│ ├── css  
│ ├── js  
│ ├── images  
│ ├── assets  
│ ├── pages  
│ │ ├── index  
│ │ │ ├── index.html  
│ │ │ ├── index.js  
│ │ │ └── index.scss  
│ │ ├── login  
│ │ │ ├── index.html  
│ │ │ ├── index.js  
│ │ │ └── index.scss

### 1.多页面的入口文件

修改 webpack.base.conf.js代码entry: {

// 多入口文件 index: ['./src/pages/index/index.js',], login: './src/pages/login/index.js',

},

### 2.配置开发服务器

npm install webpack-dev-server --save-dev//修改 webpack.dev.conf.js代码

devServer: { contentBase: path.join(\_\_dirname, "../dist"), publicPath:'/', host: "127.0.0.1", port: "8089", overlay: true, // 浏览器页面上显示错误

// open: true, // 开启自动打开浏览器

// stats: "errors-only", //stats: "errors-only"表示只打印错误： hot: true // 开启热更新

},//修改package.jsonscripts: {

"dev": "cross-env NODE\_ENV=development webpack-dev-server --config build/webpack.dev.conf.js",

"build": "cross-env NODE\_ENV=production webpack --config build/webpack.prod.conf.js",

"server": "live-server ./ --port=8888"

},

### 3.配置loader

{

test: /\.(css|scss|sass)$/,

// 不分离的写法

// use: ["style-loader", "css-loader",sass-loader"]

// 使用postcss不分离的写法

// use: ["style-loader", "css-loader", sass-loader","postcss-loader"]

// 此处为分离css的写法

/\*use: extractTextPlugin.extract({

fallback: "style-loader",

use: ["css-loader", "sass-loader"],

// css中的基础路径

publicPath: "../"

})\*/

// 此处为使用postcss分离css的写法

use: extractTextPlugin.extract({

fallback: "style-loader",

use: ["css-loader", "sass-loader", "postcss-loader"],

// css中的基础路径

publicPath: "../"

})

},

{

test: /\.js$/,

use: ["babel-loader"],

// 不检查node\_modules下的js文件

exclude: "/node\_modules/"

}, {

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

use: [{

// 需要下载file-loader和url-loader

loader: "url-loader",

options: {

limit: 5 \* 1024,//小于这个时将会已base64位图片打包处理

// 图片文件输出的文件夹

outputPath: "images"

}

}]

}

### 4.从js中分离css

npm **install** **extract**-text-webpack-**plugin** --save-dev

//这个插件在webpack4.x 运行时会有点问题,后面会提到

//修改 webpack.base.conf.js代码

// 分离css插件参数为提取出去的路径**new** extractTextPlugin({

filename: 'css/[name].[hash:8].min.css',

}),

### 5.处理css3属性前缀，消除冗余的css代码并压缩css

//属性自动加前缀

npm install postcss-loader --save-dev //在根目录新建postcss.config.jsmodule.exports = {

plugins: [

require('autoprefixer')//自动添加css前缀

]

};//loader里加入postcss

{

test: /\.(css|scss|sass)$/,

// 此处为使用postcss分离css的写法

use: extractTextPlugin.extract({

fallback: "style-loader",

use: ["css-loader", "sass-loader", "postcss-loader"],

// css中的基础路径

publicPath: "../"

})

}//修改package.json 加入css要兼容浏览器版本的代码"browserslist": [

"defaults",

"not ie < 11",

"last 2 versions",

"> 1%",

"iOS 7",

"last 3 iOS versions"

]//消除冗余css代码 使用glob模块去匹配文件所以还要安装glob模块

npm install purifycss-webpack --save-dev **new** purifyCssWebpack({

paths: glob.sync(path.join(\_\_dirname, "../src/pages/\*/\*.html"))

}),//压缩css

npm install optimize-css-assets-webpack-plugin --save-dev //压缩css//修改 webpack.prod.conf.js代码**new** OptimizeCSSPlugin({

cssProcessorOptions: {

safe: true

}

}),

### 6.清空打包目录

npm **install** clean-webpack-**plugin** --save-dev

//修改 webpack.prod.conf.js代码

//删除dist目录**new** cleanWebpackPlugin(['dist'], {

root: path.resolve(\_\_dirname, '../'), //根目录

// verbose Write **logs** **to** console.

verbose: true, //开启在控制台输出信息

// dry **Use** boolean "true" **to** **test**/emulate delete. (will **not** remove files).

// **Default**: false - remove files

dry: false,

}),

### 7. 压缩js

npm **install** uglifyjs-webpack-**plugin** --save-dev

//修改 webpack.prod.conf.js代码

//上线压缩 去除console等信息webpack4.x之后去除了webpack.optimize.UglifyJsPlugin

//https://github.com/mishoo/UglifyJS2/tree/harmony#compress-options**new** UglifyJSPlugin({

uglifyOptions: {

**compress**: {

**warnings**: false,

drop\_debugger: false,

drop\_console: true

}

}

}),

### 8.提取js公共模块

// webpack4里面移除了commonChunksPulgin插件，放在了config.optimization里面,提取js， vendor名字可改optimization: { splitChunks: { cacheGroups: { vendor: {

// test: /\.js$/, test: /[\\/]node\_modules[\\/]/, chunks: "initial", //表示显示块的范围，有三个可选值：initial(初始块)、async(按需加载块)、all(全部块)，默认为all; name: "vendor", //拆分出来块的名字(Chunk Names)，默认由块名和hash值自动生成； enforce: true,

}

}

}

},//项目里配置了自动提取node\_modules里用到的模块如jquery，也可以在原模板里面通过第三方cdn引入，又是另一种配置了。在 webpack.base.conf.js利配置externals后webpack就不会去打包配置模块externals: {

'jquery': 'window.jQuery'

},//externals就是webpack可以不处理应用的某些依赖库，使用externals配置后，依旧可以在代码中通过CMD、AMD或者window/global全局的方式访问。

### 9.复制静态资源

npm **install** copy-webpack-**plugin** --save-dev

//静态资源输出,将src目录下的assets文件夹复制到dist目录下**new** copyWebpackPlugin([{

**from**: path.resolve(\_\_dirname, "../src/assets"),

**to**: './assets',

**ignore**: ['.\*']

}]),

### 10.产出html

npm install html-webpack-plugin --save-dev //修改webpack.base.conf.js代码// 获取html-webpack-plugin参数的方法**var** getHtmlConfig = **function** (name, chunks) {

**return** {

template: `./src/pages/${name}/index.html`,

filename: `${name}.html`,

// favicon: './favicon.ico',

// title: title,

inject: true,

hash: true, //开启hash ?[hash]

chunks: chunks,//页面要引入的包

minify: process.env.NODE\_ENV === "development" ? false : {

removeComments: true, //移除HTML中的注释

collapseWhitespace: true, //折叠空白区域 也就是压缩代码

removeAttributeQuotes: true, //去除属性引用

},

};

};//配置页面**const** htmlArray = [{

\_html: 'index',

title: '首页',

chunks: ['vendor', 'index']//页面用到的vendor模块

},

{

\_html: 'login',

title: '登录',

chunks: ['login']

},

];//自动生成html模板

htmlArray.forEach((element) => {

module.exports.plugins.push(**new** htmlWebpackPlugin(getHtmlConfig(element.\_html, element.chunks)));

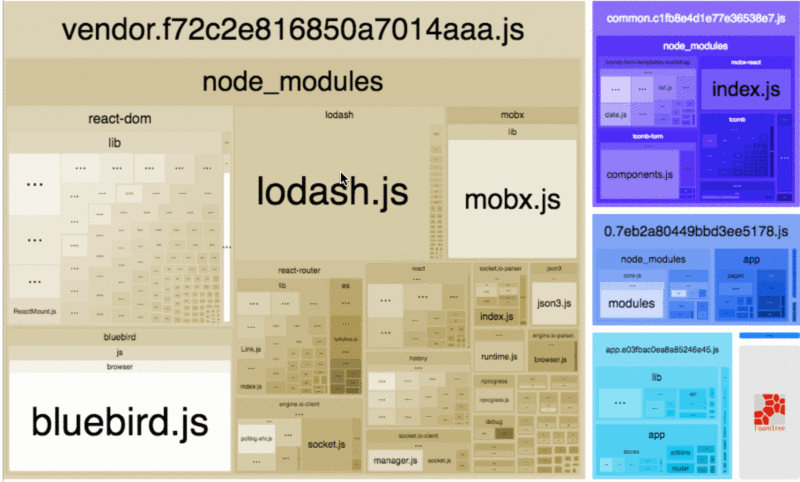
})

### 11.性能优化 高大上的可视化分析模块

npm **install** webpack-bundle-analyzer --save-dev

//修改 webpack.prod.conf.js代码**new** BundleAnalyzerPlugin()

//npm run **build** 后会打开一个页面



通过这个页面可以看到哪些页面是由哪些模块组成的，通过这个可视化页面可以更加方便去定位哪个包臃肿了，然后去优化。

### 报错 & 解决办法

Error: Chunk.entrypoints: Use Chunks.groupsIterable and filter by instanceof Entrypoint instead   
后来发现webpack4不支持extract-text-webpack-plugin 必须下载next版本安装这个插件   
npm install extract-text-webpack-plugin@next  
[https://github.com/webpack-contrib/extract-text-webpack-plugin/issues/701](https://github.com/webpack-contrib/extract-text-webpack-plugin/issues/701" \t "https://segmentfault.com/a/_blank)

### **[项目源码](https://github.com/zhouyupeng/webpack4.x_demo" \t "https://segmentfault.com/a/_blank)**