目录

[Webpack 2](#_Toc527821540)

[webpack是什么？ 2](#_Toc527821541)

[Webpack的作用 2](#_Toc527821542)

[Webpack构成： 2](#_Toc527821543)

[cli: command line interface 命令行界面 2](#_Toc527821544)

[安装webpack 2](#_Toc527821545)

[环境与命令解释 3](#_Toc527821546)

[淘宝镜像安装： 3](#_Toc527821547)

[初体验-跑一跑： 3](#_Toc527821548)

[webpack.config.js //webpack配置文件 3](#_Toc527821549)

[配置文件名字默认webpack.config.js可以改名叫： yys.config.js等 4](#_Toc527821550)

[npm script 配置自定义run 如npm run dev 4](#_Toc527821551)

[webpack4.x – mode 指定代码运行模式 5](#_Toc527821552)

[多文件、 5](#_Toc527821553)

[多入口单出口配置 5](#_Toc527821554)

[多入口多出口配置 5](#_Toc527821555)

[html-webpack-plugin 生成页面插件 6](#_Toc527821556)

[生成多个页面 7](#_Toc527821557)

[多页面分别引入自己的js 7](#_Toc527821558)

[devServer 8](#_Toc527821559)

[运行的服务器（npm run dev）：退出 ctrl+c 停止， Ctrl+d 退出cmd窗口 8](#_Toc527821560)

[webpack-dev-servers 8](#_Toc527821561)

[loaders: 9](#_Toc527821562)

[处理css文件 9](#_Toc527821563)

[压缩打包上线 10](#_Toc527821564)

[webpack4.x 自带压缩模式 10](#_Toc527821565)

[webpack3.x及之前版本使用插件压缩 10](#_Toc527821566)

[图片：（png/jpg/gif） 11](#_Toc527821567)

[默认打包后转成base64 11](#_Toc527821568)

[js引用img 11](#_Toc527821569)

[分离css 12](#_Toc527821570)

[extract-text-webpack-plugin插件 12](#_Toc527821571)

[min-css-plugin (新分离css插件) 12](#_Toc527821572)

[less 预处理器 13](#_Toc527821573)

[分离less 13](#_Toc527821574)

[sass 预处理器 13](#_Toc527821575)

[postCss 预处理 13](#_Toc527821576)

[提取出来module配置 – 使用extractTextPlugin插件 14](#_Toc527821577)

[消除冗余css代码 - purifycss，如：bootstrap只用一部分样式 14](#_Toc527821578)

[调试 15](#_Toc527821579)

[babel 15](#_Toc527821580)

# Webpack

## webpack是什么？

打包工具（模块打包器）

前端工程师必不可少工具

Webpack4.x

## Webpack的作用

1. 打包—把多个文件打包成一个文件如js, 减少服务器请求，减少带宽
2. 转化—如转化 less/sas （需要loader）
3. 优化—SPA（单页面应用程序）盛行，前端项目复杂度高， webpack可以对项目进行优化

## Webpack构成：

1. 入口 entry
2. 出口 output
3. 转化器 loaders
4. 插件 plugins
5. devServer 开发服务器
6. mode 运行的模式/环境

## cli: [command](javascript:;) [line](javascript:;) interface 命令行界面

## 安装webpack

* + npm install webpack –g
  + npm install webpack-cli –g

## 环境与命令解释

**npm install == npm i**

npm init –y //快速生成packge.json文件

npm i jquery vue loader –save //多个包同时下载

npm un jquery == npm uninstall jquery //卸载模块

clear //清除缓存

**开发环境（development）**

就是平时编写代码的环境

//开发环境需要的包

npm i jquery --save-dev == npm i jquery –D

**生产环境（production）**

就是项目开发完毕部署上线环境

npm i jquery –save == npm i jquery –S

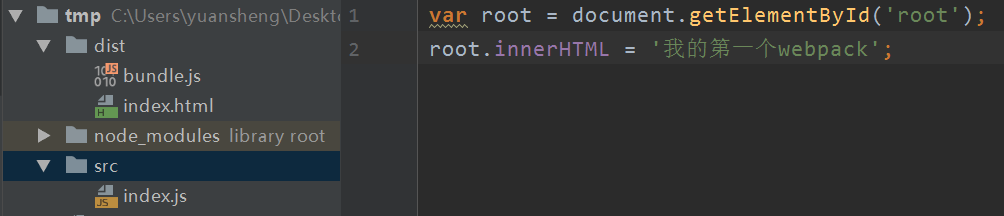
**cmd命令**

**ls //展示当前文件夹目录**

## 淘宝镜像安装：

npm install -g cnpm --registry=https://registry.npm.taobao.org

## 初体验-跑一跑：



**webpack src\index.js –output dist\bundle.js**

## webpack.config.js //webpack配置文件

**const** path = require('path');  
//C:\Users\yuansheng\Desktop\tmp\dist  
console.log(path.resolve(\_\_dirname, 'dist'));  
console.log(\_\_dirname); //C:\Users\yuansheng\Desktop\tmp  
module.exports = {  
 // 入口配置  
 entry: {  
 a: './src/index.js'  
 },  
 // 出口配置  
 output: {  
 */\*\* path: \_\_dirname + '/dist', //①path为绝对路径\*\*/*

//②resolve--把\_\_dirname和dist合并  
 path: path.resolve(\_\_dirname, 'dist'),

filename: 'bundle.js'  
 },  
 // loaders  
 module: {  
  
 },  
 plugins: [],  
 // 开发服务器  
 devServer: {  
  
 }  
}

## 配置文件名字默认webpack.config.js可以改名叫： yys.config.js等

**如果改名叫： yys.config.js**

运行时： webpack回车=> 改为如下

webpack --config yys.config.js

## npm script 配置自定义run 如npm run dev

npm script:

package.json文件：

"scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1",  
 "build": "webpack –config yys.config.js",  
 "dev": "xxxx",  
 "xxx": "xxx"  
},

## webpack4.x – mode 指定代码运行模式

webpack4.x期望实现0配置但是还未安全实现

1、package.json下的script{}中配置（开发或生产模式）：

“build”: “webpack --mode development”

或

“build”: ”webpack --mode production“

2、也可不采用第一种，在webpack.config.js中配置

module.exports = {  
 mode: 'development', //指定模式

//…

}

## 多文件、

## 多入口单出口配置

module.exports = {  
 // 入口配置  
 entry: {  
 //多个文件 按照顺序依次打包  
 a: ['./src/index.js', './src/index2.js']  
 },

……  
}

## 多入口多出口配置

webpack.config.js

module.exports = {  
 // 入口配置  
 entry: {  
 //多个文件 按照顺序依次打包  
 // a: ['./src/index.js', './src/index2.js']  
 index1: './src/index.js',  
 index2: './src/index2.js'  
 },  
 // 出口配置  
 output: {  
 */\*\* path: \_\_dirname + '/dist', //path为绝对路径\*\*/* path: path.resolve(\_\_dirname, 'dist'), //resolve--把\_\_dirname和dist合并  
 filename: '[name].bundle.js'  
 },

}

index.html



## html-webpack-plugin 生成页面插件

doc: [https://www.npmjs.com/package/html-webpack-plugin#](https://www.npmjs.com/package/html-webpack-plugin)

/\*\*注意： 本插件依赖webpack\*/

1. **安装**
   1. npm i html-webpack-plugin –D
2. **引入**
   1. const HtmlWebpackPlugin = require(‘html-webpack-plugin’)
3. **使用**

plugins: [  
 **new** HtmlWebpackPlugin()  
],

**webpack.config.js配置（压缩、消除缓存、动态标题、模板）**

plugins: [  
 **new** HtmlWebpackPlugin({  
 // 压缩配置  
 minify: {  
 collapseWhitespace: **true**, //压缩空白  
 removeAttributeQuotes: **true** //删除属性双引号  
 },  
 hash: **true**, //动态哈希，消除缓存  
 title: '我配置的标题', //html标题  
 template: './src/index.html' //模板  
 })  
],

**被打包的html模板**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">

<!--对应plugins内的 new HtmlWebpackPlugin 传入 的title-->  
 <title><%= htmlWebpackPlugin.options.title%></title>  
</head>  
<body>  
<div id="root"></div>  
</body>  
</html>

## 生成多个页面

filename: ‘xxx’

## 多页面分别引入自己的js

plugins: [  
 **new** HtmlWebpackPlugin({  
 chunks: ['index1'], //块-与this.entry.index1对应  
 filename: 'index.html',  
 // 压缩配置  
 minify: {  
 collapseWhitespace: **true**, //压缩空白  
 removeAttributeQuotes: **true** //删除属性双引号  
 },  
 hash: **true**, //动态哈希，消除缓存  
 title: '我配置的标题', //html标题  
 template: './src/index.html' //模板  
 }),  
 **new** HtmlWebpackPlugin({  
 chunks: ['index2'],//与this.entry.index2对应  
 filename: 'index2.html',  
 hash:**true**,  
 title: '我配置的标题2',  
 template: './src/index.html'  
 })  
],

clean-webpack-plugin： 删除某些东西（清除）插件

1. 下载
   1. cnpm I clean-webpack-plugin –D
2. 引入
   1. const CleanWebpackPlugin = require(‘clean-webpack-plugin’)
3. 使用
   1. new CleanWebpackPlugin([‘dist’])

plugins: [  
 **new** CleanWebpackPlugin(['dist']), //先删除dist文件夹再重新new HtmlWebpackPlugin  
 **new** HtmlWebpackPlugin({  
 chunks: ['index1'], //与this.entry.index1对应  
 filename: 'index.html',  
 // 压缩配置  
 minify: {  
 collapseWhitespace: **true**, //压缩空白  
 removeAttributeQuotes: **true** //删除属性双引号  
 },  
 hash: **true**, //动态哈希，消除缓存  
 title: '我配置的标题', //html标题  
 template: './src/index.html' //模板  
 }),  
 ],

rm –rf xxx 删除文件

## devServer

官网：<https://webpack.js.org/configuration/dev-server/>

### 运行的服务器（npm run dev）：退出 ctrl+c 停止， Ctrl+d 退出cmd窗口

### webpack-dev-servers

cnpm i webpack-dev-server -D

// 开发服务器  
devServer: {  
 // 设置服务器访问的基本目录  
 contentBase: path.resolve(\_\_dirname, 'dist'),  
 //服务器ip地址，localhost  
 host: 'localhost',  
 //设置端口，  
 port: 8090,  
 open: **true**  //自动打开浏览器}

此时 package.json:

"scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1",  
 "build": "webpack --mode development",  
 "dev": "webpack-dev-server --mode development"  
},

注意: 另一种运行devServer自动打开浏览器的方式：

"dev": "webpack-dev-server –open --mode development"

### 热更新：(如更新一个组件，关联的东西也会跟着更新)

首先: 在webpack.config.js的devServer项设置：

// 开发服务器  
devServer: {

。。。

hot: **true**, //热更新  
}

开启：在webpack.config.js的plugins项设置：

plugins: [  
 **new** webpack.HotModuleReplacementPlugin(),

],

浏览器控制台输出：



## loaders:

加载器、转化器

比如less/scss/ es7 8

### 处理css文件

cnpm i \* - D

style-loader

css-loader

配置：// loaders – use == loader 的写法  
// loaders  
module: {  
 rules: [  
 {  
 test: /\.css$/,  
 // use: ['style-loader', 'css-loader'],  
 // loader: ['style-loader', 'css-loader'],  
 use: [  
 {  
 loader: 'style-loader',  
 },  
 {  
 loader: 'css-loader'  
 }  
  
 ]  
 }  
 ]  
},

## 压缩打包上线

如何压缩：

### webpack4.x 自带压缩模式

mode: 'production', //指定模式, 开发模式为development

### webpack3.x及之前版本使用插件压缩

**uglifyjs-webpack-plugin**

* 1. cnpm i **uglifyjs-webpack-plugin –D**
  2. **const uglify = require(xxx),**
  3. **new ugliufy()**

## 图片：（png/jpg/gif）

## 默认打包后转成base64

**url-loader**

**file-loader**

**cnpm i file-loader url-loader -D**

// loaders  
module: {  
 rules: [  
 {  
 test: /\.(png|jpg|gif)/,  
 use: [{

//url-loader依赖于file-loader所以只引一个  
 loader: "url-loader",

options: {

//大于14000kb采用文件路径加载而不是base64

limit: 14000 ， //打包图片存放的目录  
 outputPath:'images'

}

}]  
 }  
 ]  
},

### js引用img

**import** './css/index.css'  
**import** imgSrc **from** './images/good girl.jpg'  
**let** img = **new** Image();  
img.src=imgSrc;  
img.onload = **function** () {  
 document.body.appendChild(img)  
};

## 分离css

### extract-text-webpack-plugin插件

1. cnpm i extract-text-webpack-plugin -D //稳定版

Error: Chunk.entrypoints: Use Chunks.groupsIterable and filter by instanceof Entrypoint instead

使用以下2安装

// @next 为最新版 如：4.x需要加这个后缀 -- @后面加版本号

cnpm i extract-text-webpack-plugin@next -D

1. 引入模块

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

1. new

**new** ExtractTextPlugin('css/index.css') //打包到css/index.css

1. 在module中配置

module: {  
 rules: [  
 {  
 test: /\.css$/,  
 // use: ['style-loader', 'css-loader'],  
 // loader: ['style-loader', 'css-loader'],  
 use: ExtractTextPlugin.extract({  
 fallback: 'style-loader',  
 use: 'css-loader',  
 //打包后css背景图路径  
 publicPath: '../'  
 })  
 },

]

### min-css-plugin (新分离css插件)

<https://www.npmjs.com/package/mini-css-extract-plugin>

new MiniCssExtractPlugin({

filename: ‘css/index.css’

})

注意：该插件背景图片可能存在路径问题

## less 预处理器

1. cnpm i less less-loader -D
2. module中配置

{  
 test: /\.less$/,  
 use: ['style-loader', 'css-loader', 'less-loader']  
},

1. 在打包的index.js中引入

**import** './less/a.less'

### 分离less

{  
 test: /\.less$/,  
 use: ExtractTextPlugin.extract({  
 fallback: 'style-loader',  
 use: ['css-loader', 'less-loader'],  
 publicPath: '../'  
 })  
},

## sass 预处理器

1、cnpm i node-sass sass-loader –D

2、module中配置

{  
 test: /\.sass|\.scss/,  
 use: ExtractTextPlugin.extract({  
 fallback: 'style-loader',  
 use: ['css-loader', 'sass-loader'],  
 publicPath: '../'  
 })  
},

1. index.js中引入a.sass

**import** './sass/a.sass'

## postCss 预处理

专门处理css平台

//postcss-loader 依赖autoprefixer

1. cnpm i postcss-loader autoprefixer -D
2. 准备postcss.config.js 配置postCss

module.exports={  
 plugins: [  
 require('autoprefixer')  
 ]  
};

1. webpack.config.js 的module中配置

use: [  
 {loader: "style-loader"},  
 {loader: "css-loader"},  
 {loader: "postcss-loader"},  
]

1. 结果：

#root {  
 color: yellow;  
 -webkit-transform: rotate(45deg); //自动生成c3适配  
 transform: rotate(45deg);  
}

### 提取出来module配置 – 使用extractTextPlugin插件

use: ExtractTextPlugin.extract({  
 fallback: 'style-loader',  
 use: ['css-loader', 'postcss-loader'],  
 //打包后css背景图路径  
 publicPath: '../'  
})

## 消除冗余css代码 - purifycss，如：bootstrap只用一部分样式

1、cnpm i purifycss-webpack purify-css -D

2、webpack.config.js引入插件

**const** PurifyCssWebpack= require('purifycss-webpack');

1. 需要用到额外的包

glob

**const** glob = require('glob'); //与PurifyCssWebpack配合使用-找路径

1. plugins中配置

**new** PurifyCssWebpack({  
 paths: glob.sync(path.join(\_\_dirname, 'src/\*.html'))  
})

## 调试

webpack4.x开启调试：

--mode development （npm run dev配） 或 mode: development （module配）

mode: 'development', //指定模式, 开发模式为develop 4.x用

webpack3.x 开启调试：

在module中加

devtool: ‘source-map’

devtool: 'source-map', //3.x之前用

## babel 待续

-babel用来编译js

-让你很轻松的使用Esnext转化

-jsx

安装 babel-core bable-loader babel-preset-env

cnpm i babel-loader babel-core babel-preset-env -D

## 使用模块

和node一模一样

导出： module.exports = xxx

引入： require(‘./xxx’)

## 在webpack中使用json

webpack.config.json中定义：

{  
 "entry": "./src/index.js"  
}

webpack.config.js(与webpack.config.json平级)中使用：

**const** json = require('./webpack.config.json');

在module.exports中配置

entry: {  
 index1: json.entry,  
 },

## copy-webpack-plugin – 静态资源输出- 复制资源

1. 下载：cnpm i copy-webpack-plugin -D
2. 引入：

**const** CopyWebpackPlugin = require('copy-webpack-plugin'); //静态资源输出

1. 使用：

**new** CopyWebpackPlugin([  
 {  
 from: path.resolve(\_\_dirname, 'src/assets'),  
 to: './public'  
 }  
])

## 使用第三方库

**两种方式：**

**第一种：**

直接npm 下载 cnpm i jquery -D

**import** $ **from** 'jquery'  
$('#root').css({  
 display: "block",  
 width: "100px",  
 height: "50px",  
 background: '#e4393c'  
})

**第二种：**

### providePlugin – webpack自带插件- 分离打包js

const webpack = require(‘webpack’)

在plugin里面使用

**new** webpack.ProvidePlugin({  
 $: 'jquery'  
})

**第一种和第二种方式区别：**

第一种如果没有使用jquery也会打包进去，

第二种如果没有使用jquery 不会打包进去，消除冗余

## 提取第三方库或打包为单独js文件

**webpack3.x之前：**

CommonsChunkPlugin（4.x removed）

new webpack.optimize.CommonsChunkPlugin({

name: ‘jquery’

})

**webpack4.x:**

在webpack.config.js中module.exports增加配置项

