1、全局安装webpack，npm i webpack -g -S

2、初始化npm，npm init -y

3、局部安装webpack

4、设置webpack.config.js文件，设置entery,output

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

output: {

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

filename: 'bundle.js'

},

5、本地安装webpack-dev-server,npm i webpack-dev-server -D，吧打包好的文件放在内存中托管。

本地安装无法在powershell中使用“webpack”命令运行

6、在package.json 中配置 dev: "webpack-dev-server"

7、webpack-dev-server 命令配置

（1）第一种方式

在package.json中的scripts对像中配置如下属性

dev:"webpack-dev-server --open --port 3000 --contentBase src --hot"

（2）第二种配置方式

在webpack.config.js中配置

devServer: {

open: true,//自动打开浏览器

port: 8088,

contentBase: "src", //指定托管的根目录

hot: true //热更新

}

在package.json中的scripts对像中配置如下属性

dev:"webpack-dev-server"

8、启动热更新

const webpack = require('webpack');

plugins: [//配置插件的节点

new webpack.HotModuleReplacementPlugin(),//一个热更新的模块对象,启动热更新的第三步

]

9、安装html-webpack-plugin,在内存中给index.html页面添加script

使用了这个插件之后，我们不用手动添加js引用路径，这个插件会帮我们完成

npm i html-webpack-plugin -D

//导入在内存中生成html的插件

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

plugins: [//配置插件的节点

new webpack.HotModuleReplacementPlugin(),//一个热更新的模块对象,启动热更新的第三步

new htmlWebpackPlugin({

//创建一个在内存中生成html页面的插件

template:path.join(\_\_dirname,'index.html'),//指定模板页面，将来会根据指定的页面路径，去生成内存中的页面.HTML必须放在根目录中

filename:'index.html'//指定在内存中生成的页面的名称

})

]

10、使用import引入css ，必须要安装相应的加载器。

1、安装css-loader 和 style-loader npm i style-loader css-loader -D

2、在webpack.config.js中新增一个配置节点，叫 module,它是一个对象，其中有个rules属性（数组），这个数组中存放了所有第三方文件的匹配和处理规则。

module: {//这个节点用于配置所有第三方模块的加载器

rules: [//匹配规则

{ test: /\.css$/, use: ['style-loader', 'css-loader'] } //配置处理css文件的第三方loader规则

]

}

注意：webpack处理第三方文件类型的过程

1、发现这个要处理的文件不是js文件，然后就去配置文件中找有没有对应的第三方规则。

2、如果找到对应的规则，就会调用对应的loader处理。

3、在调用loader的时候，按照规则设置，从后往前调用。

4、当最后一个loader调用完毕，会把处理的结果，直接交给webpack进行打包合并，最终输出到bundle.js中去。

11、使用less或者sass

安装less和less-loader，less-loader依赖于less

npm i less less-loader -D

填写匹配规则

{ test: /\.less$/, use: ['style-loader', 'css-loader','less-loader'] }, //配置处理less文件的第三方loader规则

安装node-sass 和 sass-loader,sass-loader 内部依赖于 node-sass

npm i node-sass sass-loader -D

{ test: /\.scss$/, use: ['style-loader', 'css-loader','sass-loader'] }, //配置处理sass文件的第三方loader规则

注意：要使用scss,sass不兼容css语法。会报错

12、使用url-loader

默认情况下，webpack无法处理文件中的url地址，不管是图片还是字体库，都无法处理。

安装url-loader 和 file-loader

npm i url-loader file-loader -D

{ test: /\.(jpg|png|gif|bmp|jpeg)$/, use: ['url-loader?limit=1000&name=[hash:8]-[name].[ext]'] },

//处理图片路径的loader规则

//limit给定的值，是限制图片转为base64编码，限制值单位是byte,如果我们引用的图片，大于或等于给定的limit值，则图片不会

//被转为base64格式的字符串，如果图片小于给定的值，则会被转为base64的字符串

另一种配置方式（vue-cli给的配置方式）

{

test: /\.(png|jpe?g|gif|svg)(\?.\*)?$/,

loader: 'url-loader',

options: {

limit: 10000,

name: utils.assetsPath('img/[name].[hash:7].[ext]')

}

}

处理字体文件

{ test: /\.(ttf|eot|svg|woff)$/, use: 'url-loader' },//处理字体文件的loader

另一种配置方式（vue-cli给的配置方式）

{

test: /\.(woff2?|eot|ttf|otf)(\?.\*)?$/,

loader: 'url-loader',

options: {

limit: 10000,

name: utils.assetsPath('fonts/[name].[hash:7].[ext]')

}

}

13、安装配置babel，使其编译es6语法

1、在webpack中，可以运行如下两套命令，安装两套包，去安装Babel相关loader

2、第一套包 npm i babel-core babel-loader babel-plugin-transform-runtime -D

3、第二套包 npm i babel-preset-env babel-preset-stage-0 -D

4、打开webpack.json 配置文件，在module节点下的rules数组中，添加一个新的匹配规则

{ test: /\.js$/, use: 'babel-loader', exclude: /node\_modules/ },//转换es6语法，include包含要处理的文件夹路径

注意：要过滤掉除了其他代码，只把自己写的代码打包进去，使用include 或者 exclude均可以.

如果不排除掉其他js,babel会把node\_modules中所有第三方js文件都打包编译，会消耗很多cpu资源，和内存。

哪怕最终全部运转完了，项目也无法运行

5、在项目的根目录中，新建一个叫做 .babelrc的babel配置文件，这个配置文件，属于JSON格式，所以，在写 .babelrc配置的时候，必须符合JSON的语法规范，不能写注释，字符串必须有双引号包裹。

在babelrc中写如下配置

{

"presets": [

"env",

"stage-0"

],

"plugins": [

"transform-runtime"

]

}

6.如果有报错

Error: Cannot find module '@babel/core'

babel-loader@8 requires Babel 7.x (the package '@babel/core'). If you'd like to use Babel 6.x ('babel-core'), you should install 'babel-loader@7'.

安装一个babel-loader@7就解决

npm i babel-loader@7 -D