1. 新建src 文件夹 APP.js webpack.config.js文件 配置入口文件和导出文件
2. 在根目录新建导出文件夹 比如 dist
3. Node 服务器 输入 npm init -y 初始化webpack配置
4. npm i -D webpack 安装时需要输入yes 安装webpack-cli
5. 在package.json文件的scripts对象里输入 运行命令 比如："build": "webpack"

6.把webpack.config.js 里的filename的js 文件 引入到index.html 文件就可以输出

配置完成

**安装HTML插件 自动生成HTML文件**

1. npm i -D html-webpack-plugin
2. 在webpack.json 的文件里可以看到html-webpack-plugin 添加进去了
3. 在webpack.congig.js 文件里配置

先声明 const HtmlWebpackPlugln=require('html-webpack-plugin');

在导出里写入plugins:[

new HtmlWebpackPlugln()

]

4.在node里跑npm run build 之后可以在输出文件里dist里看到生成的index.html 文件

**也可以自定义Html文件**

1. 删掉自动生成的index.html文件 在src 文件夹里新建一个自定义的index.html文件
2. 然后在webpack.congig.js里

plugins:[

new HtmlWebpackPlugln(

{

filename:"acc.html",//新建的文件名

template:"src/index.html" //之前自定义的文件名

})]

3.在node里跑npm run build 就可以了 在输出文件里dist里看到生成的acc.html 文件

接着上面：

**Loader 预处理模块**

1. **npm i 先安装依赖**
2. **npm i -S react react-dom 安装react和react-dom**
3. **npm i -D babel-loader babel-core 安装babel-loader和babel-core**
4. **npm i -D babel-preset-react 安装要预设的js 为 react的JSX**
5. 然后在webpack.congig.js里 输入

module:{

rules:[{ //用来放规则

test:/\.js$/, //要匹配的文件为.js结尾的所有js文件

use:[{ //需要用的loader

loader:'babel-loader', //loader名

options:{ // 要预设的js为react 语法 jsx

presets:['react']

}

}]

}]

}

devserver服务器：

1. npm i -D webpack-dev-server

2在package.json的sripts里输入 "start":"webpack-dev-server"

3然后在webpack.congig.js里 输入：

devServer:{

open:true, //自动打开浏览器 默认8080端口

// port:9000 //指定端口

}

4.用npm start 起来

//处理css文件中的url，会自动帮你引入里面要引入的模块；

引入css样式 ：

1. 在webpack.congig.js 的 rules数组里加入以下对象

{

test:/\.css$/,

use:['style-loader','css-loader']

}

1. npm i -D css-loader; 先引入css 再引入style

3.npm i -D style-loader;

4.npm start;

//file-loader:

//1.把你的资源移动到输出目录

//2.返回最终引入资源的url

引入图片：

1.在webpack.congig.js 的 rules数组里加入以下对象

{

test:/\.(png|jpg|gif|pneg)$/,// 输入图片的类型

use:['file-loader']

}

2npm i -D file-loader；

1. npm start

Url-loader的引入 增强版的file-loader

1. npm i -D url-loader;
2. 在webpack.congig.js 的 rules数组里加入以下对象

{

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

use:[

{

loader:'url-loader',

options:{

limit:10000 //10kb以下的图片 用url转成base64编码 其他的用file loader 来打包

}

}

]

}

1. npm start

**引入字体**

1. 在webpack.congig.js 的 rules数组里加入以下对象：

{

test:/\.(ttf|eot|woff|woff2|svg)$/,

use:['file-loader']

}

1. npm start

下载第三方字体 npm i -S font-awesome

引入方法 import 'font-awesome/css/font-awesome.css';

**排除css部分模块化**：

在原有的配置上来改：

原来的：

{

test:/\.css$/,

use:['style-loader','css-loader']

}

修改后的：

{

test:/\.css$/,

use:['style-loader',

{

loader: 'css-loader',

options:{

module:true, //默认不开启模块化

localIdentName:'[name]-[local]\_[hash:base64:6]

}

}

],

exclude:[ //排除不用开启模块化的文件

path.resolve(\_\_dirname,'node\_modules'),

path.resolve(\_\_dirname,'src/common')

]

},

{

test:/\.css$/,

use:['style-loader','css-loader'],

include:[ //全局的

path.resolve(\_\_dirname,'node\_modules'),

path.resolve(\_\_dirname,'src/common')

]

},

**配置less 和Scss**

1. **npm i -D sass-loader node-sass；**
2. 在webpack.congig.js 的 rules数组里加入以下对象：

//未开启模块化的配置

//未开启模块化的引入方式 如：import './common/style/main.scss';

{

test:/\.scss$/,

use:['style-loader','css-loader','sass-loader']

},

// 开启模块化 的配置

//开启模块化的引入方式 如：import style from './common/style/main.scss';

{

test:/\.scss$/,

use:['style-loader',

{

loader:'css-loader',

options:{ //开启模块化

module:true, //默认不开启模块化

localIdentName:'[name]-[local]\_[hash:base64:6]

}

},

'sass-loader']

},

1. npm start

**配置less 和Scss 需要和css-loader和style-loader来转化**

**优化以上less和Scss的配置如下：**

**Scss优化之后的配置如下：**

{

test:/\.scss$/,

use:['style-loader',

{

loader: 'css-loader',

options:{

module:true, //默认不开启模块化

localIdentName:'[name]-[local]\_[hash:base64:6]'

}

},

'sass-loader'

],

exclude:[ //排除不用开启模块化的文件

path.resolve(\_\_dirname,'node\_modules'),

path.resolve(\_\_dirname,'src/common')

]

},

{

test:/\.scss$/,

use:['style-loader','css-loader','sass-loader'],

include:[ //全局的

path.resolve(\_\_dirname,'node\_modules'),

path.resolve(\_\_dirname,'src/common')

]

},

**Less优化之后的配置如下：**

**安装 npm i -D less-loader less；**

{

test:/\.less$/,

use:['style-loader',

{

loader: 'css-loader',

options:{

module:true, //默认不开启模块化

localIdentName:'[name]-[local]\_[hash:base64:6]'

}

},

'less-loader'

],

exclude:[ //排除不用开启模块化的文件

path.resolve(\_\_dirname,'node\_modules'),

path.resolve(\_\_dirname,'src/common')

]

},

{

test:/\.less$/,

use:['style-loader','css-loader','less-loader'],

include:[ //全局的

path.resolve(\_\_dirname,'node\_modules'),

path.resolve(\_\_dirname,'src/common')

]

},

**babel 强大的script编译器**

1. **在新的文件里安装babel npm init**
2. **npm i -D babel-cli**

**Webpack.config.js的配置：**

**//导出一个node模块，供webpack去调用，导出是一个对象，改对象设置webpack运行过程中的一些配置参数**

**module.expots{**

**//应用的入口文件，webpack会分析这个入口文件中导入的其他文件，并进行处理，最后进行打包**

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

**//设置打包后的文件输出配置**

**output:{**

**path：**'./dist'，

fileName：'bundle.js'

**}**,

//配置loader

Module:{

Rules:[

//每个规则就是一个对象

{

//当加载的是该类型的文件

test：/\.txt$/,

//使用什么loader去处理

}

]

**}**

Webpack的几项主要功能：

[Webpack](https://webpack.github.io/)是一个模块打包工具（module bundler），以下列出Webpack的几项主要功能： webpack loader，webpack-dev-server

* 将CSS，图片与其他资源打包
* 打包之前预处理（Less，CoffeeScript，JSX，ES6等）档案
* 依条目文件不同，把.js分拆为多个.js档案
* 整合丰富的Loader可以使用（Webpack本身仅能处理JavaScript模块，其余档案如：CSS，Image需要载入不同Loader进行处理）