# WebPack4.X配置指南

# 一、简介

## （一）WebPack简介

Webpack是一个前端自动化打包工具，根据它的名字也很好理解，web-pack顾名思义就是前端打包工具，它是基于Node和NPM的，所以在安装使用webpack之前，需要安装nodejs，nodejs的版本过低也不行，所以推荐安装nodejs版本为v8.11.2以上，npm版本为v5.6.0以上。

# 二、项目建立

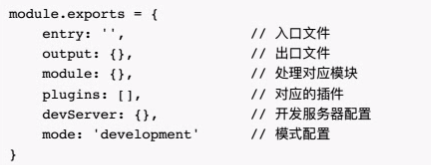
## （一）项目建立

**第一步**：新建一个文件夹，命名为mywebpack;

**第二步**：在cmd命令框中使用cd命令切换到mywebpack目录中，输入npm init或npm init -y,创建package.json文件，需要注意的是如果package.json的名称为webpack可能会报错。

**第三步**：将mywebpack文件夹插入到VSCode等编辑工具中，并在mywebpack根目录下新建一个webpack.config.js文件。

关于webpack.config.js中常见配置项的说明如下：



先做一个默认配置，如下：

const path = require('path');

module.exports = {

entry: path.resolve(\_\_dirname, './src/app.js'), //入口文件

output: {

path: path.resolve(\_\_dirname, './dest'), //打包后的目录

filename: 'dabao.js' //打包后的文件名称

}

}

**注**：可以将上面的配置项拷贝到你的webpack.config.js文件中。

**第四步**：在第三步中的配置内容需要注意两个文件及一个路径：

entry: path.resolve(\_\_dirname, './src/app.js')

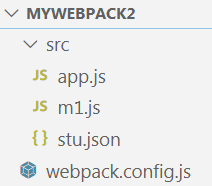
path: path.resolve(\_\_dirname, './dest')

filename: 'dabao.js'

path: path.resolve(\_\_dirname, './dest')及filename: 'dabao.js'这两个配置项的意思是webpack打包完成后会在mywebpack项目根目录中自动创建dest目录，dest目录中自动创建dabao.js文件；

entry: path.resolve(\_\_dirname, './src/app.js')是需要你在mywebpack项目根目录创建一个src文件夹,然后在src文件夹中创建一个app.js、m1.js、stu.json文件。

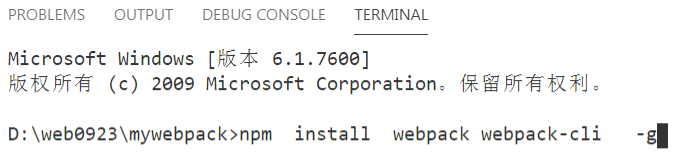
要打包的目录结构及代码如下：



**第五步**：使用webpack打包第四步中的内容则需要使用npm安装webpack和webpack-cli,在cmd命令框中输入：

npm install webpack webpack-cli -g

如下图：



**第六步**：

1、在mywebpack项目根目录下新建src目录，在src目录中新建m1.js文件，m1.js文件内容如下：

exports.xingming = 'lisi';

module.exports.age = 20;

module.exports.fn = function() {

console.log('fn在m1.js文件');

}

2、在mywebpack项目根目录下新建src目录，在src目录中新建stu.json文件，stu.json文件内容如下：

{ "age": 21, "usr": "lisi" }

3、在mywebpack项目根目录下新建src目录，在src目录中新建app.js文件，app.js文件内容如下：

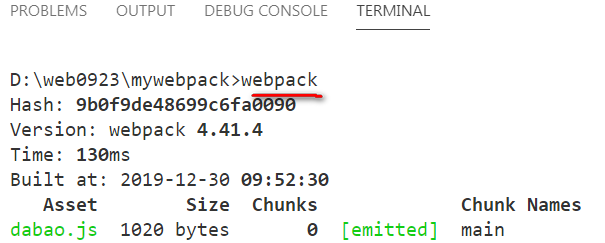
let m1 = require('./m1.js');

console.log(m1);

let stus = require('./stu.json');

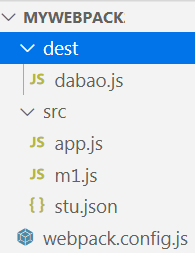
console.log(stus);

在命令行终端中执行webpack命令:



（如果报错 Cannot find module 'html-webpack-plugin' 则在根目录输入cnpm install html-webpack-plugin --save-devcc

在命令行终端中执行完webpack命令后，就会在myapp项目根目录中创建一个dest文件夹，并在dest文件夹新建一个dabao.js文件，如下：



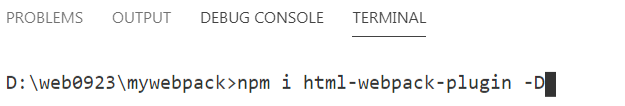
**注**：打包之后的dabao.js文件可以在前端html文件中使用了(比如：可以在dest目录中新建demo.html文件，然后在demo.html文件使用dabao.js文件)，而src/app.js不能直接在前端html文件中使用。

# 三、模块和插件配置安装

（一）配置HTML模板

可以安装一个html-webpack-plugin插件来打包html模板（如:在src目录下新建index.html文件，），并将该模板指向对应的入口文件。

安装html-webpack-plugin插件，如下：



然后在mywebpack项目根目录的src目录中新建index.html,其内容如下：

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

接下来修改webpack.config.js配置文件，增加下面的内容：

const path = require('path');

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

module.exports = {

    entry: path.resolve(\_\_dirname, './src/app.js'), //入口文件

    output: {

        path: path.resolve(\_\_dirname, './dest'), //打包后的目录

        filename: 'dabao.js' //打包后的文件名称

    },

    //设置插件配置项

    plugins: [

        new HtmlWebpackPlugin({

            //在src目录下创建一个index.html页面当作模板来使用

            template: './src/index.html'

        })

    ]

}

接下来执行webpack命令打包，打包完成后在dest目录中会生成一个index.html，**而index.html文件内容会增加一行内容**：

<script type="text/javascript" src="dabao.js"></script>

**注意**：如果在打包过程中报如下的错误：

Error: Cannot find module 'webpack/lib/node/NodeTemplatePlugin

则在mywebpack项目根目录中执行下面命令后再使用webpack命令来打包即可：

npm i webpack -D

（二）引用CSS

要将内部样式表打包完成后输出到dist目录中，需要安装两个加载器：style-loader和css-loader：

npm i style-loader css-loader -D

其中css-loader是负责解析css的，而style-loader则负责将css以内部样式表的形式嵌入到html中，当使用插件拆分css时就不再需要style-loader了，因为是以外部样式表link到html文件中的。

修改webpack.config.js，增加内容如下：

const path = require('path');

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

module.exports = {

    entry: path.resolve(\_\_dirname, './src/app.js'), //入口文件

    output: {

        path: path.resolve(\_\_dirname, './dest'), //打包后的目录

        filename: 'dabao.js' //打包后的文件名称

    },

    //module 当中配置loader

    module: {

        rules: [{

            test: /\.css$/,

            use: ['style-loader', 'css-loader'] //  loader就是要处理对应test规则的文件  loader的use规则，是从右向左处理

        }]

    },

    //设置插件配置项

    plugins: [

        new HtmlWebpackPlugin({

            //在src目录下创建一个index.html页面当作模板来使用

            template: './src/index.html'

        })

    ]

}

在mywebpack项目根目录的src目录中新建common.css文件，其内容如下：

body {

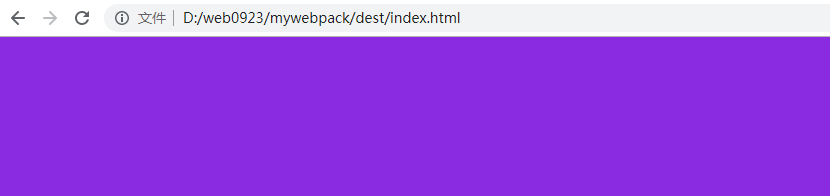
    background-color: blueviolet;

}

再修改mywebpack项目根目录的src目录中app.js文件，增加下面的内容：

require('./common.css');

最后执行webpack命令打包，打包完成后在dest目录中会生成一个index.html，在浏览器中预览index.html，显示如下：

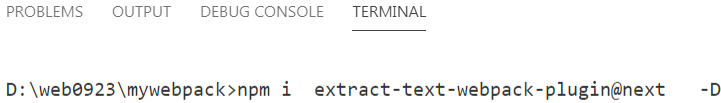


（三）拆分CSS

通过使用extract-text-webpack-plugin插件将刚才的内部样式表拆分为外部样式表。

安装extract-text-webpack-plugin插件：

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



**注**：extract-text-webpack-plugin目前版本不支持webpack4

修改webpack.config.js内容如下：

增加：

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

修改：

    //设置插件配置项

    plugins: [

        new HtmlWebpackPlugin({

            //在src目录下创建一个index.html页面当作模板来使用

            template: './src/index.html'

        }),

        //css拆分后会把css文件放到dest目录下的css/styles.css中

**new ExtractTextWebpackPlugin('css/style.css')**

    ],

    //module 当中配置loader

    module: {

        rules: [{

            test: /\.css$/,

            //use: ['style-loader', 'css-loader'] //  loader就是要处理对应test规则的文件  loader的use规则，是从右向左处理

**use: ExtractTextWebpackPlugin.extract({ use: 'css-loader' }) //将css用link方式引入就不需要style-loader了**

        }]

    }

最后执行webpack命令打包，打包完成后在dest目录中会创建css目录，在css目录中会生成style.css文件，并且在dest目录生成的index.html会引入style.css文件，如下：



**注**：也可以在src目录中新建demo.css文件，将在该文件中增加如下内容：

\* {

    padding: 0px 0px;

Margin: 0px 0px;

}

然后修改app.js文件并增加：require('./demo.css')， 这样webpack打包时可以将多个css文件内容合并成一个css文件中；

（四）引用Less

Less在前端的使用方式有两种：一种是引入less.js文件来动态编译less代码，这样可以降低程序的响应时间；另一种是使用工具或lessc来手动编译转换less代码为css; 现在可以在webpack中配置less，让它变得自动起来。

安装：

npm i less less-loader -D

修改mywebpack项目根目录中的src目录下的index.html文件，在<body>标签中增加下面的内容：

    <div class="container">

        <div class="header">

            <p>less效果</p>

        </div>

    </div>

在mywebpack项目根目录中的src目录下新建less目录，在less目录中新建index.less文件，其内容如下：

.container{

    .header{

        p{

            color: blueviolet;

        }

    }

}

最后修改mywebpack项目根目录中的src目录下app.js文件，增加下面的内容：

require('./less/index.less');

执行webpack命令打包，打包完成后在dest目录中会创建css目录，在css目录中会生成style.css文件，并把index.less文件内容合并到style.css文件中，如下：



（五）打包CSS中的图片

css中引用图片主要使用file-loader和url-loader，其中file-loader主要解决引用路径问题，url-loader主要解决图片的加载优化问题，可以设置一个阈值，比如将9k以下的图片转换为data-url的形式直接引用到html文件中，从而减少请求次数，以达到优化目的。

安装：

npm i url-loader file-loader -D

修改webpack.config.js配置文件：

        module:  {

        rules:  [{

                test:   /\.css$/,

                //use:  ['style-loader',  'css-loader']  //  loader就是要处理对应test规则的文件  loader的use规则，是从右向左处理

                use:  ExtractTextWebpackPlugin.extract({  use:   'css-loader'  })  //将css用link方式引入就不需要style-loader了

            },

**{**

**test: /\.(png|jpg|gif)$/,**

**use: [{**

**loader: 'url-loader',**

**options: {**

**limit: 8192**

**}**

**}**]

            }]

    }

修改mywebpack项目根目录中的src目录下的common.css文件，增加下面的内容：

.div1 {

    background: url('./3333.jpg') no-repeat;

}

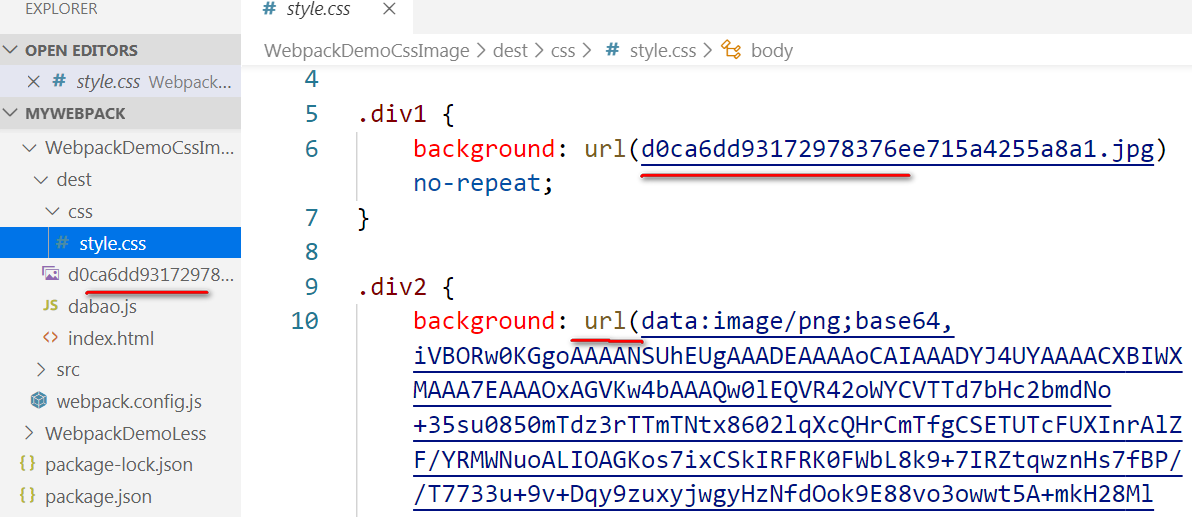
.div2 {

    background: url('./123.png') no-repeat;

}

**注**：将3333.jpg、123.png图片拷贝到mywebpack项目根目录中的src目录里。

执行webpack命令打包，打包完成后在dest目录中会生成3333.jpg图片打包后的文件，123.png图片小于9K而不会生成实体图片，只会将该图片自动转换成base64格式。



（六）压缩JS代码

在webpack.config.js文件中新增：  
   mode: 'production' //默认是production表示压缩js代码

mode: 'development' //development表示不压缩代码

（七）压缩CSS代码

CSS代码压缩需要安装插件：optimize-css-assets-webpack-plugin

安装：

npm i optimize-css-assets-webpack-plugin --save-dev

修改webpack.config.js文件，增加下面的配置项：

const OptimizeCssAssetsPlugin= require('optimize-css-assets-webpack-plugin');

和

new OptimizeCssAssetsPlugin({

assetNameRegExp: /\.css$/g,

cssProcessor:require('cssnano'),

cssProcessorOptions:{safe:true,discardComments:{removeAll:true}},

canPrint:true

})

（八）压缩HTML代码

Html代码压缩不用额外安装插件及模块，修改webpack.config.js中的html插件为：

        new HtmlWebpackPlugin({

            //在src目录下创建一个index.html页面当作模板来使用

            template: './src/index.html',

minify:{

removeAttributeQuotes:true,

removeComments:true,

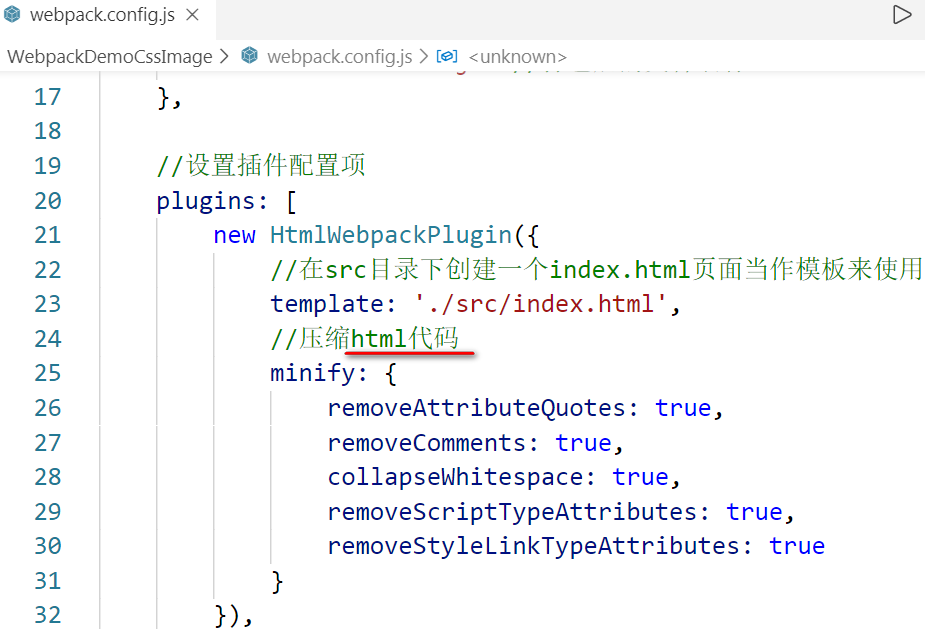
collapseWhitespace:true,

removeScriptTypeAttributes:true,

removeStyleLinkTypeAttributes:true

}

        }),



（九）webpack热更新

每次运行和更新代码都要输入webpack命令重新打包，太麻烦了，使用webpack热更新可以解决该问题，每次改动代码后，只需要保存代码并刷新页面即可。

安装

npm i webpack-dev-server -D

修改webpack.config.js配置文件，增加下面的内容：

const Webpack = require('webpack');

new Webpack.HotModuleReplacementPlugin(), //webpack热更新插件

//新增热更新配置：

devServer:{

//设置服务器访问的基本目录

contentBase:path.resolve(\_\_dirname,'dest'),

//设置服务器ip地址，可以是localhost

host:'localhost',

//设置端口

port:8090,

//设置自动打开浏览器

open:true,

//设置热更新

hot:true

}

webpack.config.js配置文件内容如下：



最后使用webpack-dev-server --mode development来启动：