# Webpack

使用Webpack需要安装2个依赖包：webpack和webpack-cli。【本机需具备node环境】，CMD执行命令：npm i webpack webpack-cli -g

## 目录介绍

src：项目目录

build:项目输出目录

node\_modules:npm下载的包所存放的位置

## webpack.config.js配置

原生Webpack默认配置仅支持json和js文件的集成打包，不支持样式和图片等其他资源，.less、.css、.img等文件；编写项目的时候又确实需要其他资源，这就需要webpack配置资源解析器，以下先介绍webpack的手动配置文件，再介绍如何配置其他资源解析器。

/\*\*

 \* webpack配置文件，作用：指示webpack干哪些活

 \* 所有构建工具都是基于node.js平台运行的模块化默认采用common.js

\*/

const {resolve} = require("path");

module.exports = {

    mode:"development",//两种模式可选development/production

    entry:"",//入口起点

    output:{

      filename:"",//输出文件名

      path:resolve(\_\_dirname,"") //\_dirname是node.js的变量，代表当前文件目录的绝对路径},

      module:{

        rules:[//详细loader配置

            {

                test:,//正则表达式来验证资源是否符合解析器要求的文件格式

                use:[//use数组中loader执行顺序，从右到左，从下到上执行

]

            }

        ]

},plugins:[//插件

]

}

## 资源解析器配置介绍

### CSS资源解析器配置

注：需要npm安装css-loader和style-loader

test:/\.css$/,

use:[//use数组中loader执行顺序，从右到左，从下到上执行

   "style-loader",//创建style标签，将js中的样式资源插入进行，添加到页面中生效

   "css-loader"//将css文件变成CommonJS模块加载js中，里面的内容是样式字符串

]

以上的配置仅适用于将css代码融入js代码中，希望将css独立出来，需要npm安装mini-css-extract-plugin；这可以将css独立成单独的文件,配置如下。

const MiniCssExtractPlugin = require("mini-css-extract-plugin");

 rules:[

         {

            test:/\.css$/,

            use:[

                //提取js中的css成单独文件

                MiniCssExtractPlugin.loader,

                //将css文件整合到js文件中

                "css-loader"

            ]

         },

]

plugins:[

        new MiniCssExtractPlugin({

            //对输出的文件重命名，所有的css文件都会集中输出到这index.css文件中

            filename:"css/index.css"

        }),

]

### LESS资源解析器配置

注：需要npm安装less和less-loader

test:/\.less$/,

use:[

"style-loader",

"css-loader",

"less-loader"//将less文件编译成css文件

]

### HTML资源解析器配置【plugins配置】

注：需要npm安装html-webpack-plugin

const HtmlWebpackPlugin = require("html-webpack-plugin");//声明

new HtmlWebpackPlugin({

//复制HTML文件，并自动引入打包输出的所有资源，index.html中不需引入js/css文件了

template:"./index.html"

})

### 图片资源解析器配置

注：需要npm安装url-loader和file-loader

{//处理图片资源

     test: /\.(jpg|png|gif)$/,

     loader: "url-loader",//一个依赖项就可以不用谢use，直接写loader

     options: {

//图片大小小于8Kb，就会被base64处理

//优点：减少请求数量（减轻服务器压力）

//缺点：图片体积会更大（文件请求速度更慢）

          limit: 8 \* 1024,

          outputPath:"font"//输出到哪个文件夹下

      }

},{

     test:/\.html$/,//解析html图片文件,[注：需要npm安装html-loader]

     loader:"html-loader"

}

如果需要压缩CSS文件，需要npm安装optimize-css-assets-webpack-plugin插件

const OptimizeCssAssetsWebpackPlugin = require("optimize-css-assets-webpack-plugin");

plugins:[

        new OptimizeCssAssetsWebpackPlugin()

    ]

### 字体解析器配置

注：需要npm安装file-loader

index.js

import "./iconfont.css";//引入字体文件

Webpack.config.js

{ //排除css/js/html资源

     test:/\.(svg|ttf|woff|eot)$/,

     loader:"file-loader",

     options:{

          name:"[hash:10].[ext]",

          outputPath:"font"//输出到哪个文件夹下

    }

}

总结：所有的npm下载的包的配置，均可以到npm官网上查看如何配置使用。

### ESlint检查配置

需要npm安装eslint-loader，eslint，eslint-config-airbnb-base，eslint-plugin-import；

package.json配置

"eslintConfig":{

    "extends":"airbnb-base"

  }

Webpack.config.js配置

{

    /\*\* 配置js的Eslint检查，只检查源代码，不能检查第三方库，所以要使用exclude排除node\_modules中的js

      \* 设置检查规则

      \* 使用airbnb规则

    \*/

     test:/\.js$/,

     exclude:/node\_modules/,

     loader:"eslint-loader",

     options:{fix:true}

}

JavaScript兼容性处理

需要npm安装babel-loader，@babel/preset-env，@babel/core这3个包

{/\* 配置js的Eslint检查，这个检查只检查源代码，不能检查第三方库，所以要使用exclude排查node\_modules中的js依赖，设置检查规则，使用airbnb规则\*/

     test:/\.js$/,

     use:[

      {loader:"eslint-loader",options:{fix:true}},

      {loader:"babel-loader",options:{presets:["@babel/preset-env"]}}

     ],

     exclude:/node\_modules/

}

## Webpack打包

手动打包：CMD中进入到项目文件夹根目录输入:webpack，然后进入到文件夹打开html输出文件即可。此时的项目是完全编译并存储在磁盘中的。

自动打包：利用node的环境启动服务，需要npm安装webpack-dev-server且webpack.config.js中配置如下。【注：自动编译的项目是存储在**内存**中的】

//开发服务器DevServer：自动化（自动编译，自动打开浏览器，自动刷新浏览器）

    devServer:{

        contentBase:resolve(\_\_dirname,"build"),

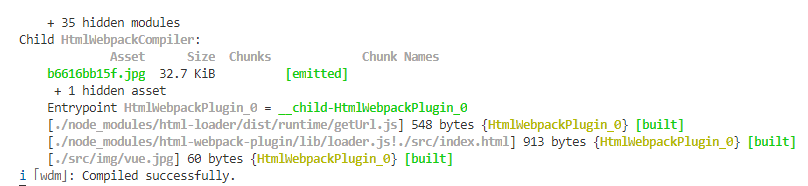
        compress:true,//启动gzip压缩

        port:3000,//端口号

        open:true//打开默认浏览器

    }

在CMD中运行命令：npx webpack-dev-server,成功的结果如下



这个时候，到浏览器地址栏输入localhost:3000（端口号设置为多少填入多少）回车。停止组合键：Ctrl+C。

开启HRM功能（模块更新）

开启HRM功能后，webpack项目就不会每个js模块都执行了，只会执行更新的那个js模块；配置完成后需重启webpack服务。

devServer:{

hot:true

}