# 简介

## Webpack是什么

·前端资源构建工具，静态模块打包器

·前端的所有资源文件都会作为模块处理

·将根据模块的依赖关系进行静态分析，打包生成对于的静态资源（bundle）

·所有构建工具都是基于nodejs运行的

·模块化默认采用commonjs

## 五个核心概念

### Entry

·入口

·指示webpack以哪个文件为入口起点开始打包，分析构建内部依赖图

### Output

·输出

·指示webpack打包后的资源bundles输出到哪里去，以及如何命名

### Loader

·让webpack能够去处理哪些非js文件(webpack自身只理解js)

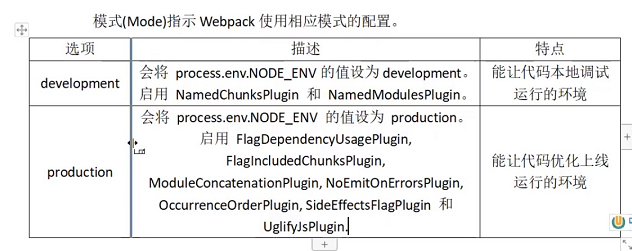
### Plugins

·插件

·用于执行范围更广的任务

·插件范围包括从打包优化和压缩一直到重新定义环境中的变量等

### Mode



# Webpack的初体验

## 打包js和json资源

·npm init

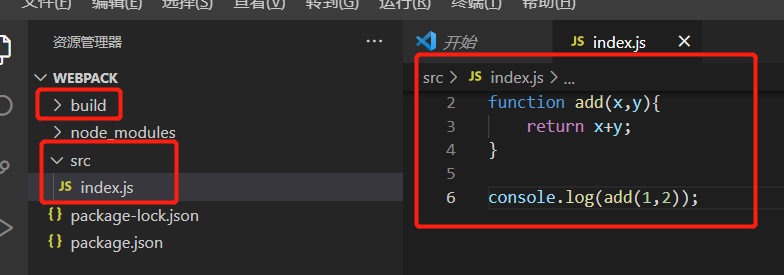
- 初始化package.json配置

·npm i webpack webpack-cli -g

- 安装webpack

·npm i webpack webpack-cli -D

- 将webpack添加到开发依赖



·运行指令：

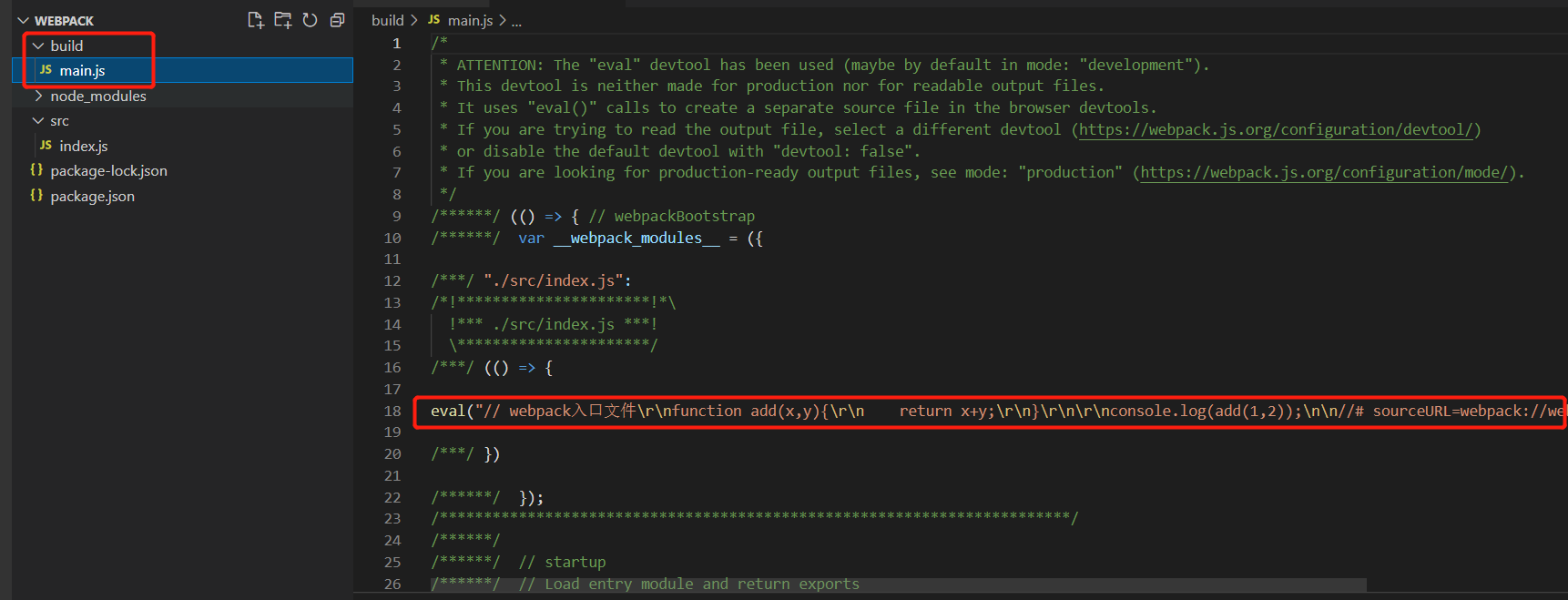
- 开发环境

-- webpack ./src/index.js -o ./build --mode=development

-- webpack 会以 ./src/index.js 为入口文件开始打包，打包后输出到 ./build/main.js文件

-- 整体打包环境是开发环境

-- 生成该文件



- 生产环境

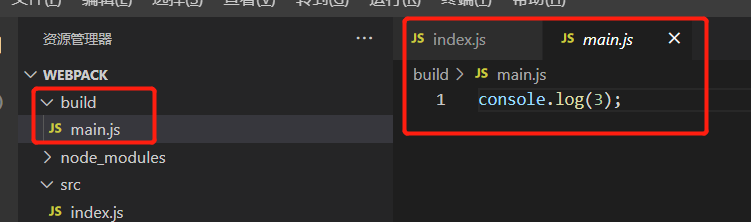
-- webpack ./src/index.js -o ./build --mode=production

-- webpack 会以 ./src/index.js 为入口文件开始打包，打包后输出到 ./build/main.js文件

-- 整体打包环境是生产环境

-- 会压缩js代码

-- 生成该文件



·执行js文件

- node .\build\main.js



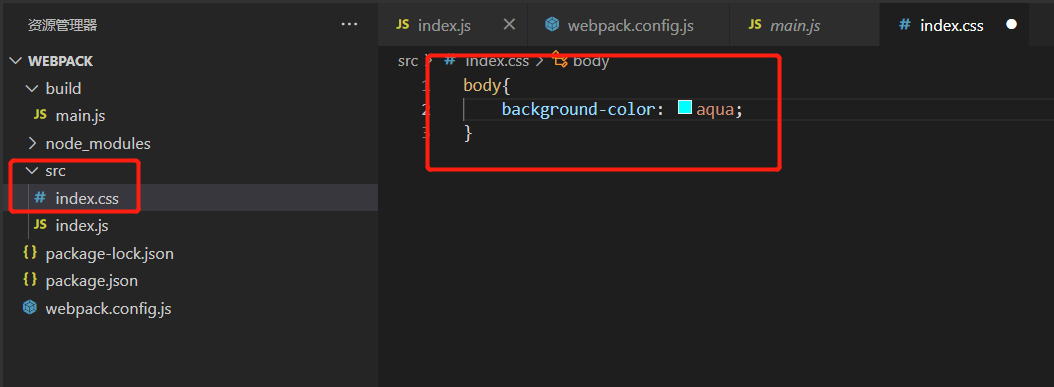
·webpack也可以打包json文件，不能处理css/img等其他文件

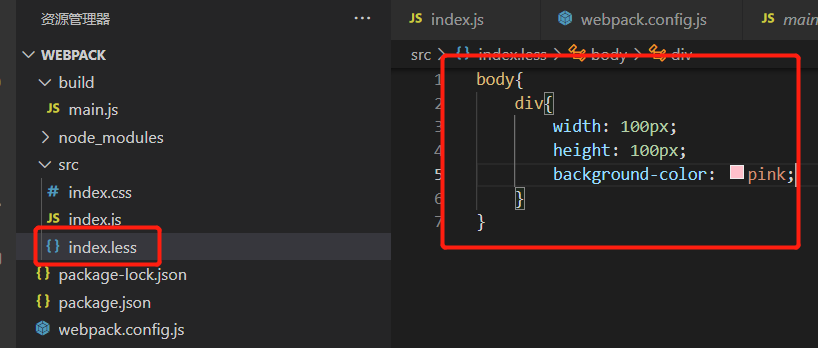
·生产环境和开发环境将ES6模块化编译成浏览器能识别的模块化

·生产环境比开发环境多一个压缩js代码

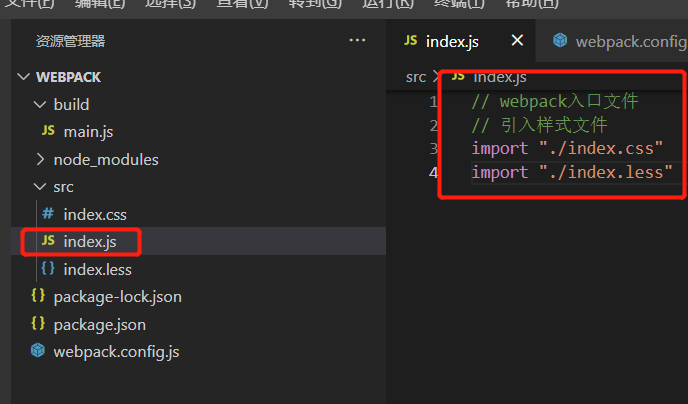
## 打包样式资源

·创建样式文件

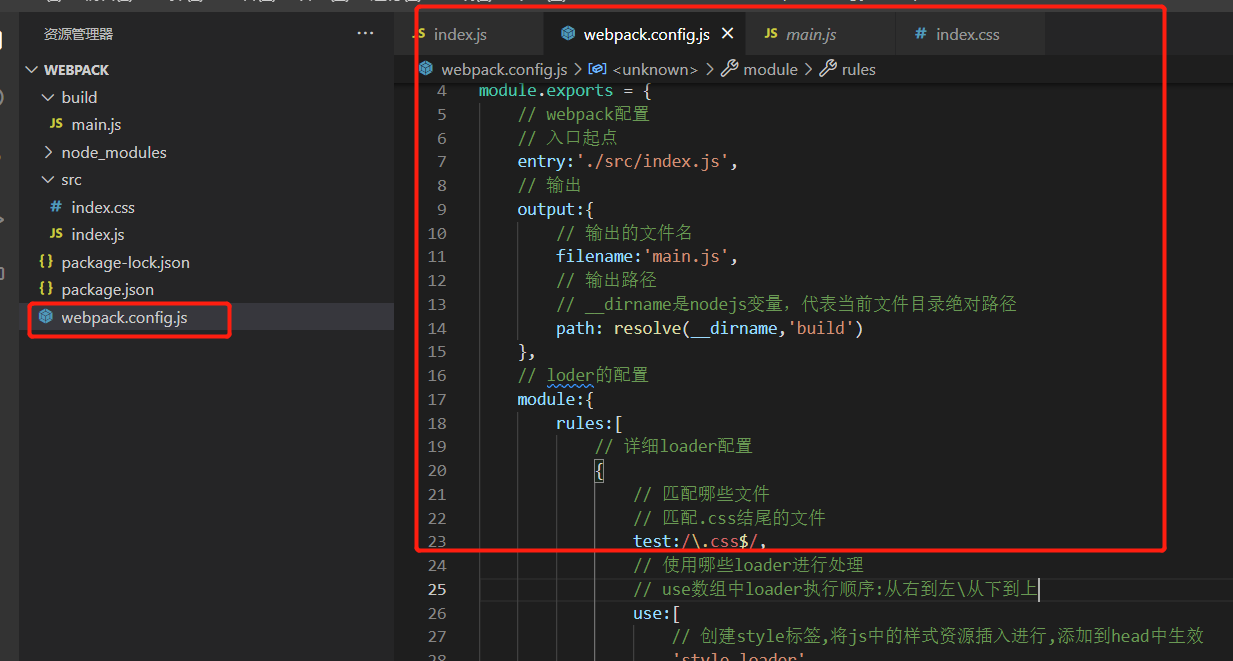




·引入样式资源



·编写webpack配置文件



// webpack配置文件

// resolve用来拼接绝对路径

const {resolve} = require('path');

module.exports = {

    // webpack配置

    // 入口起点

    entry:'./src/index.js',

    // 输出

    output:{

        // 输出的文件名

        filename:'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname,'build')

    },

    // loder的配置

    module:{

        rules:[

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test:/\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use:[

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader'

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test:/\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use:[

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader'

                ]

            }

        ]

    },

    // plugins的配置

    plugins:[

        // 详细的plugins配置

    ],

    // 模式,指定development或production模式

    mode:'development'

}

·下载style-loader、css-loader和less-loader到开发依赖

- npm i style-loader css-loader less-loader -D

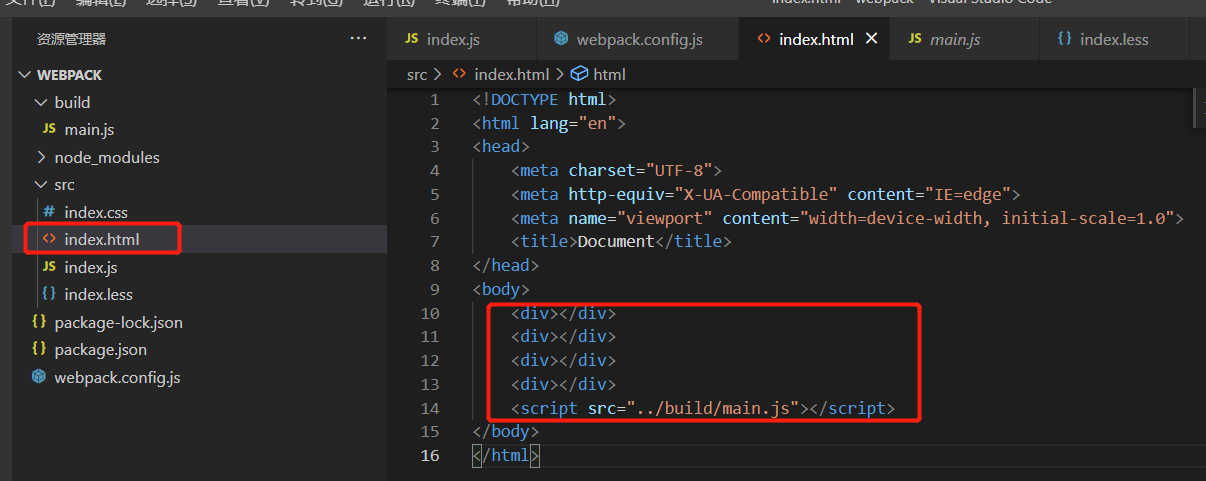
·下载less到开发依赖

- npm i less -D

·执行webpack打包

- webpack

·使用打包后的文件测试有没有生效

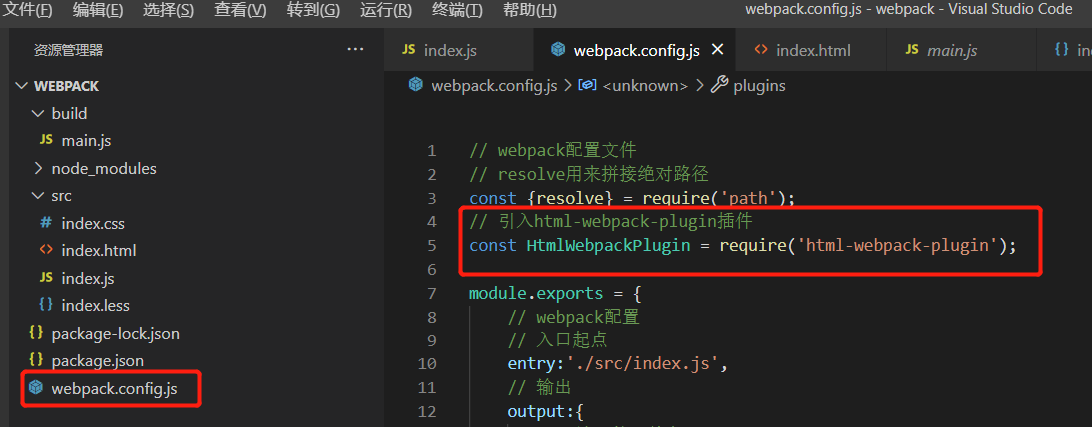


## webpack打包html资源

·下载html-webpack-plugin插件

- npm i html-webpack-plugin -D

·引入插件



// webpack配置文件

// resolve用来拼接绝对路径

const {resolve} = require('path');

// 引入html-webpack-plugin插件

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

module.exports = {

    // webpack配置

    // 入口起点

    entry:'./src/index.js',

    // 输出

    output:{

        // 输出的文件名

        filename:'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname,'build')

    },

    // loder的配置

    module:{

        rules:[

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test:/\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use:[

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader'

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test:/\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use:[

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader'

                ]

            }

        ]

    },

    // plugins的配置

    plugins:[

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

            template:'./src/index.html'

        })

    ],

    // 模式,指定development或production模式

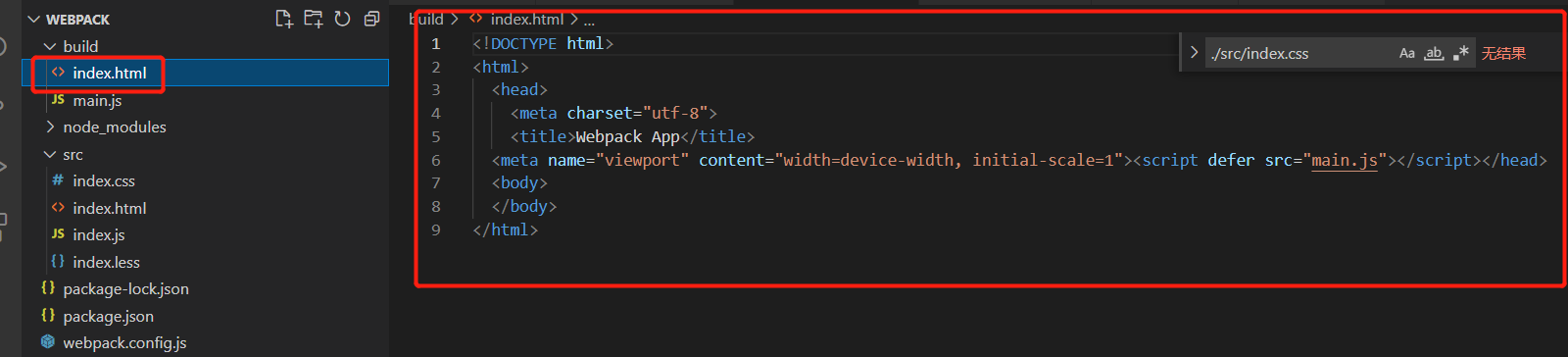
    mode:'development'

}

·打包

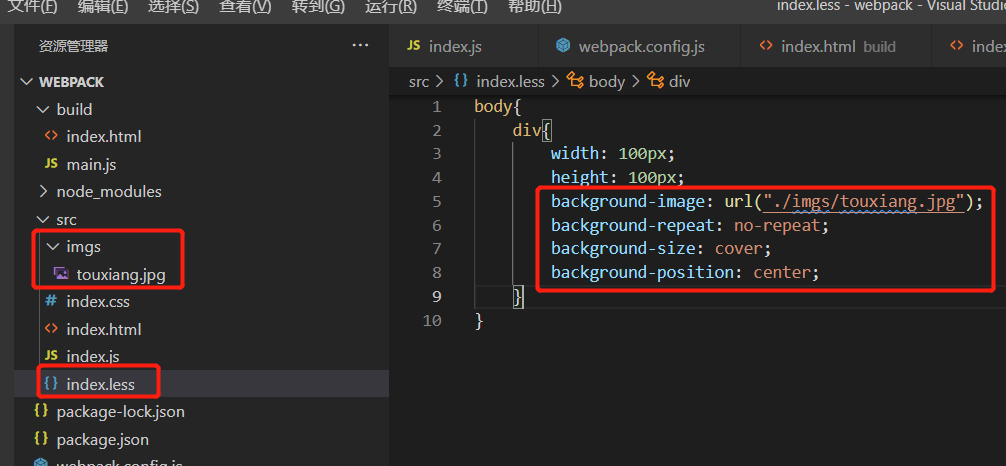
- webpack

·生成以下文件



## 打包图片资源

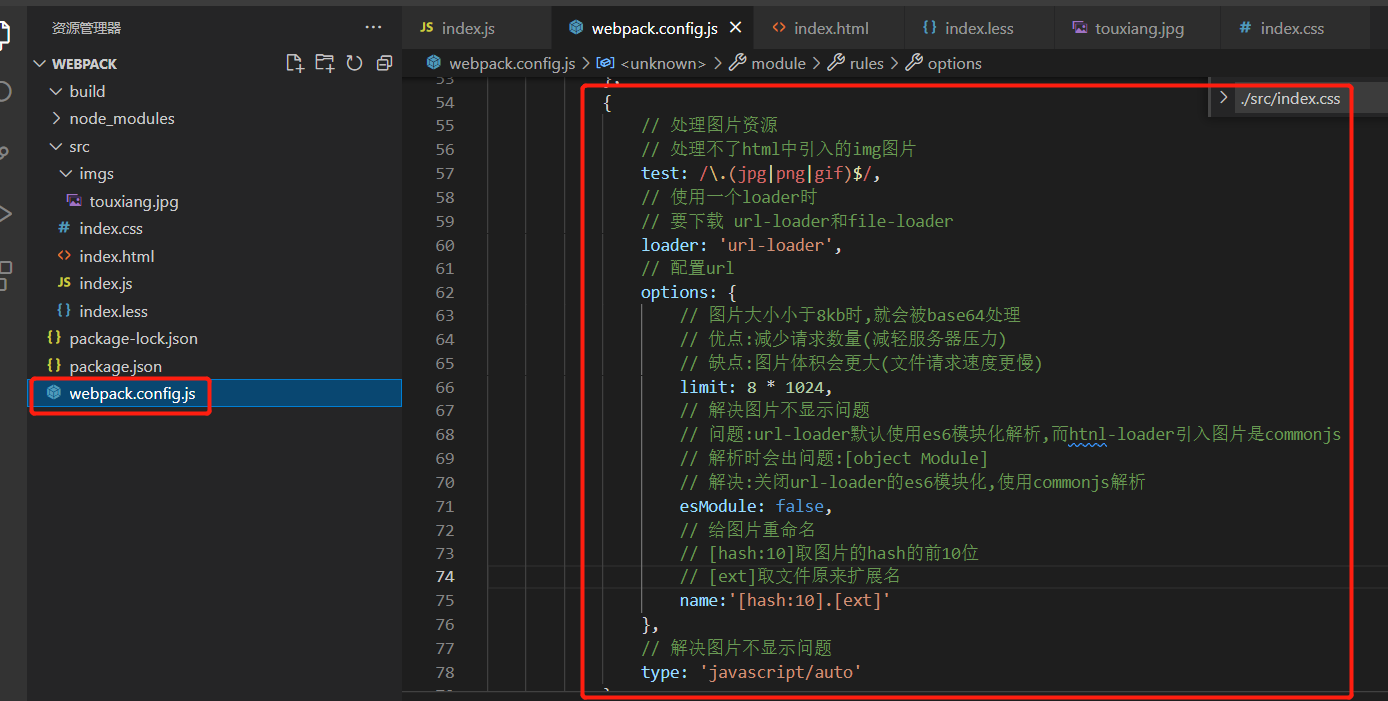
·引入图片资源



·下载url-loader、file-loader和html-loader

- npm i url-loader file-loader html-loader -D

·webpack配置



// webpack配置文件

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

module.exports = {

    // webpack配置

    // 入口起点

    entry: './src/index.js',

    // 输出

    output: {

        // 输出的文件名

        filename: 'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test: /\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader'

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test: /\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader'

                ]

            },

            {

                // 处理图片资源

                // 处理不了html中引入的img图片

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

                // 使用一个loader时

                // 要下载 url-loader和file-loader

                loader: 'url-loader',

                // 配置url

                options: {

                    // 图片大小小于8kb时,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,

                    // 解决图片不显示问题

                    // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                    // 解析时会出问题:[object Module]

                    // 解决:关闭url-loader的es6模块化,使用commonjs解析

                    esModule: false,

                    // 给图片重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name:'[hash:10].[ext]'

                },

                // 解决图片不显示问题

                type: 'javascript/auto'

            },

            // 处理html中引入的图

            {

                test: /\.html$/,

                // 负责引入img,从而能被url-loader打包处理

                loader: 'html-loader'

            }

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

            template: './src/index.html'

        })

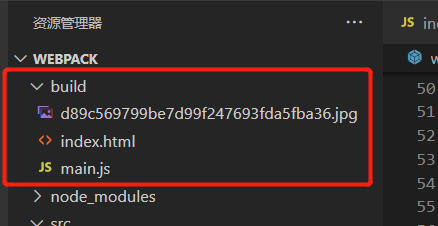
    ],

    // 模式,指定development或production模式

    mode: 'development'

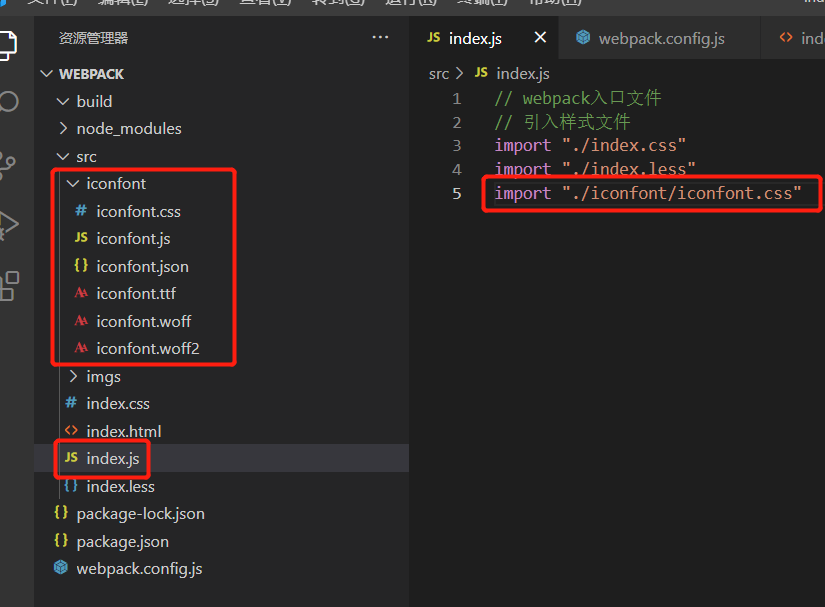
}

·webpack打包

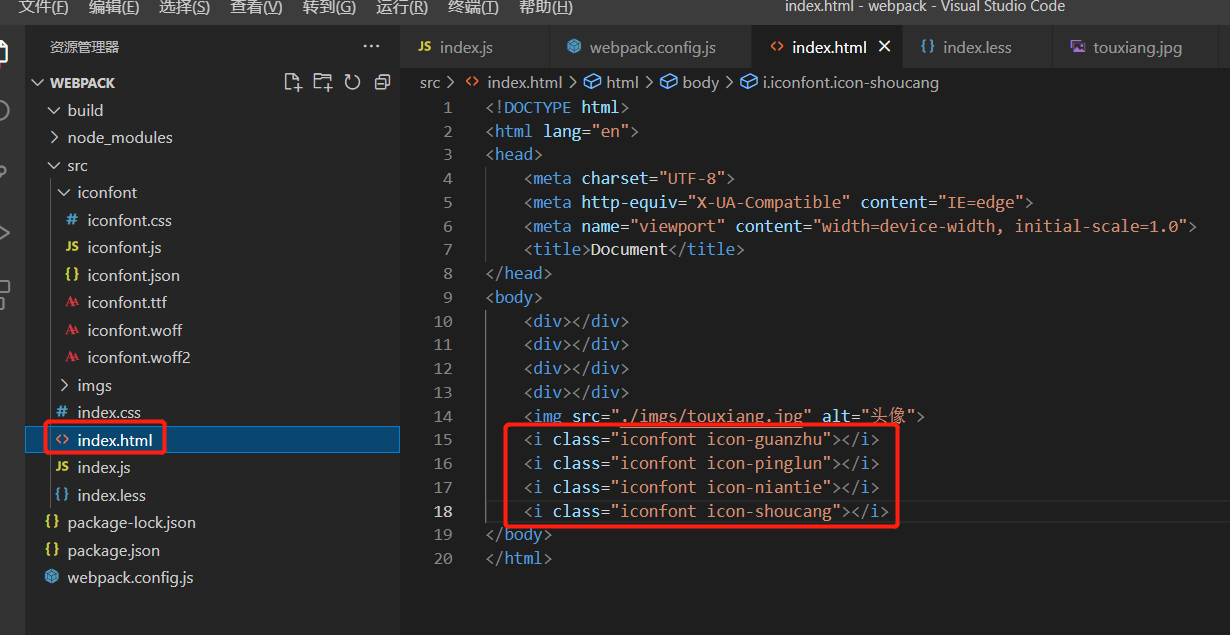


## 打包其他资源

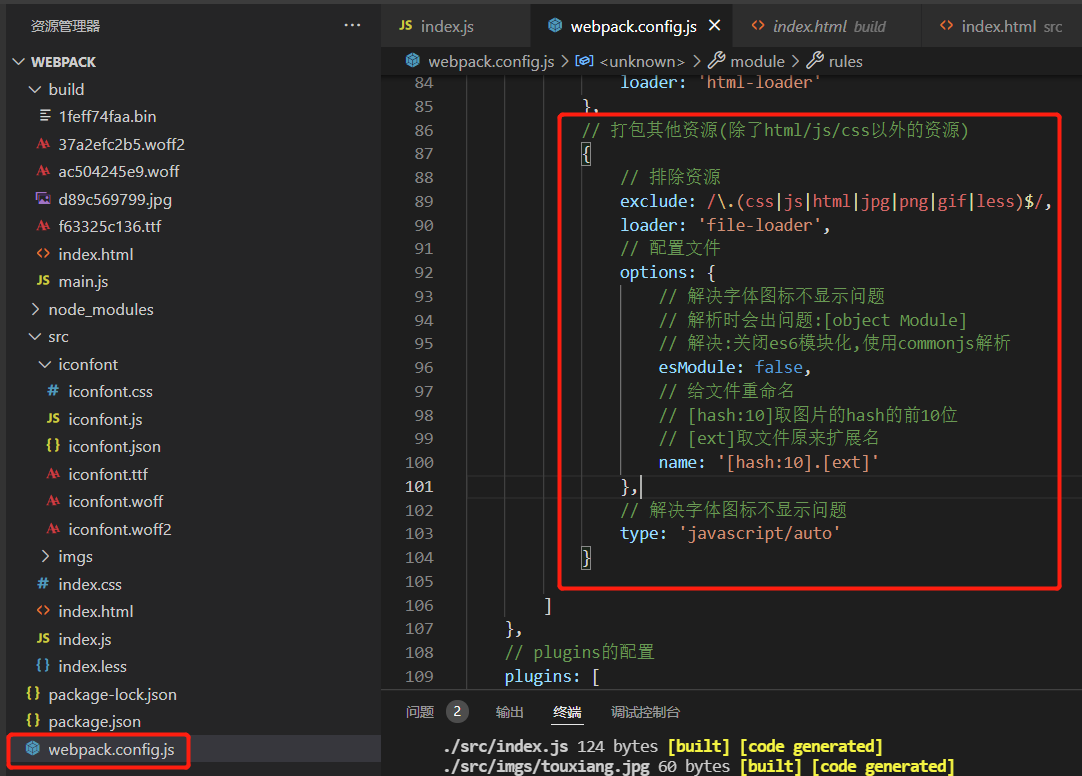
·引入字体图标



·使用字体图标



·配置webpack



// webpack配置文件

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

module.exports = {

    // webpack配置

    // 入口起点

    entry: './src/index.js',

    // 输出

    output: {

        // 输出的文件名

        filename: 'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test: /\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader'

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test: /\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader'

                ]

            },

            {

                // 处理图片资源

                // 处理不了html中引入的img图片

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

                // 使用一个loader时

                // 要下载 url-loader和file-loader

                loader: 'url-loader',

                // 配置url

                options: {

                    // 图片大小小于8kb时,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,

                    // 解决图片不显示问题

                    // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                    // 解析时会出问题:[object Module]

                    // 解决:关闭url-loader的es6模块化,使用commonjs解析

                    esModule: false,

                    // 给图片重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]'

                },

                // 解决图片不显示问题

                type: 'javascript/auto'

            },

            // 处理html中引入的图

            {

                test: /\.html$/,

                // 负责引入img,从而能被url-loader打包处理

                loader: 'html-loader'

            },

            // 打包其他资源(除了html/js/css以外的资源)

            {

                // 排除资源

                exclude: /\.(css|js|html|jpg|png|gif|less)$/,

                loader: 'file-loader',

                // 配置文件

                options: {

                    // 解决字体图标不显示问题

                    // 解析时会出问题:[object Module]

                    // 解决:关闭es6模块化,使用commonjs解析

                    esModule: false,

                    // 给文件重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]'

                },

                // 解决字体图标不显示问题

                type: 'javascript/auto'

            }

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

            template: './src/index.html'

        })

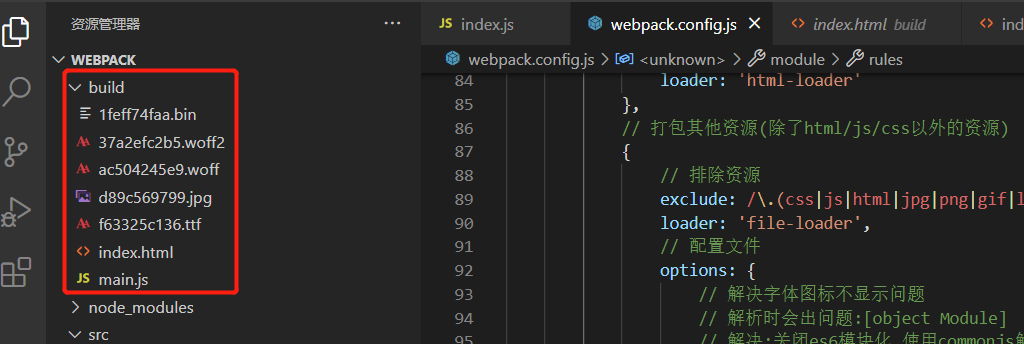
    ],

    // 模式,指定development或production模式

    mode: 'development'

}

·webpack打包



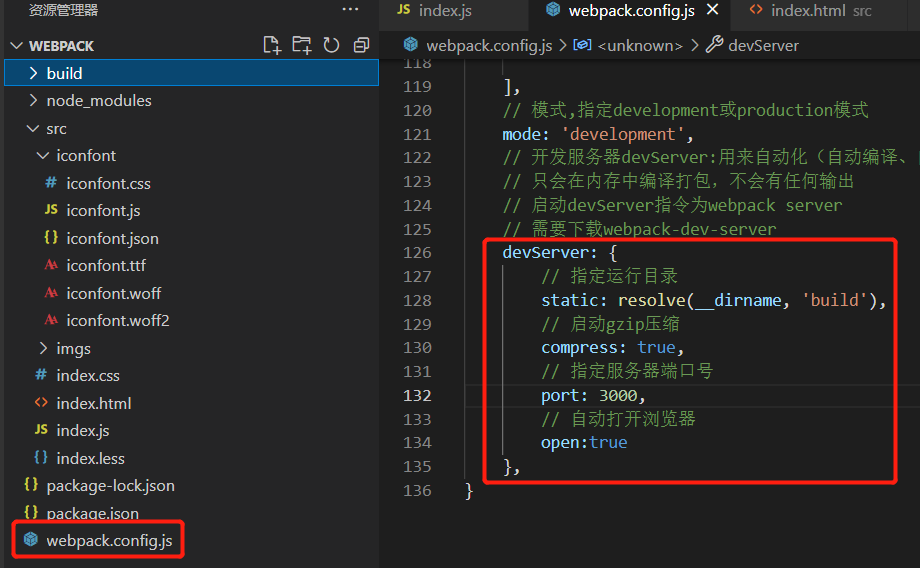
## devServer

·webpack自动化（自动编译、自动打开浏览器、自动刷新浏览器~）

·下载webpack-dev-server

- npm i webpack-dev-server -D

·配置webpack



// webpack配置文件

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

module.exports = {

    // webpack配置

    // 入口起点

    entry: './src/index.js',

    // 输出

    output: {

        // 输出的文件名

        filename: 'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test: /\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader'

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test: /\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader'

                ]

            },

            {

                // 处理图片资源

                // 处理不了html中引入的img图片

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

                // 使用一个loader时

                // 要下载 url-loader和file-loader

                loader: 'url-loader',

                // 配置url

                options: {

                    // 图片大小小于8kb时,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,

                    // 解决图片不显示问题

                    // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                    // 解析时会出问题:[object Module]

                    // 解决:关闭url-loader的es6模块化,使用commonjs解析

                    esModule: false,

                    // 给图片重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]'

                },

                // 解决图片不显示问题

                type: 'javascript/auto'

            },

            // 处理html中引入的图

            {

                test: /\.html$/,

                // 负责引入img,从而能被url-loader打包处理

                loader: 'html-loader'

            },

            // 打包其他资源(除了html/js/css以外的资源)

            {

                // 排除资源

                exclude: /\.(css|js|html|jpg|png|gif|less)$/,

                loader: 'file-loader',

                // 配置文件

                options: {

                    // 解决字体图标不显示问题

                    // 解析时会出问题:[object Module]

                    // 解决:关闭es6模块化,使用commonjs解析

                    esModule: false,

                    // 给文件重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]'

                },

                // 解决字体图标不显示问题

                type: 'javascript/auto'

            }

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

            template: './src/index.html'

        })

    ],

    // 模式,指定development或production模式

    mode: 'development',

    // 开发服务器devServer:用来自动化（自动编译、自动打开浏览器、自动刷新浏览器~）

    // 只会在内存中编译打包，不会有任何输出

    // 启动devServer指令为webpack server

    // 需要下载webpack-dev-server

    devServer: {

        // 指定运行目录

        static: resolve(\_\_dirname, 'build'),

        // 启动gzip压缩

        compress: true,

        // 指定服务器端口号

        port: 3000,

        // 自动打开浏览器

        open:true,

        // 自动刷新

        watchFiles:['./src/index.html']

    },

}

·启动devServer

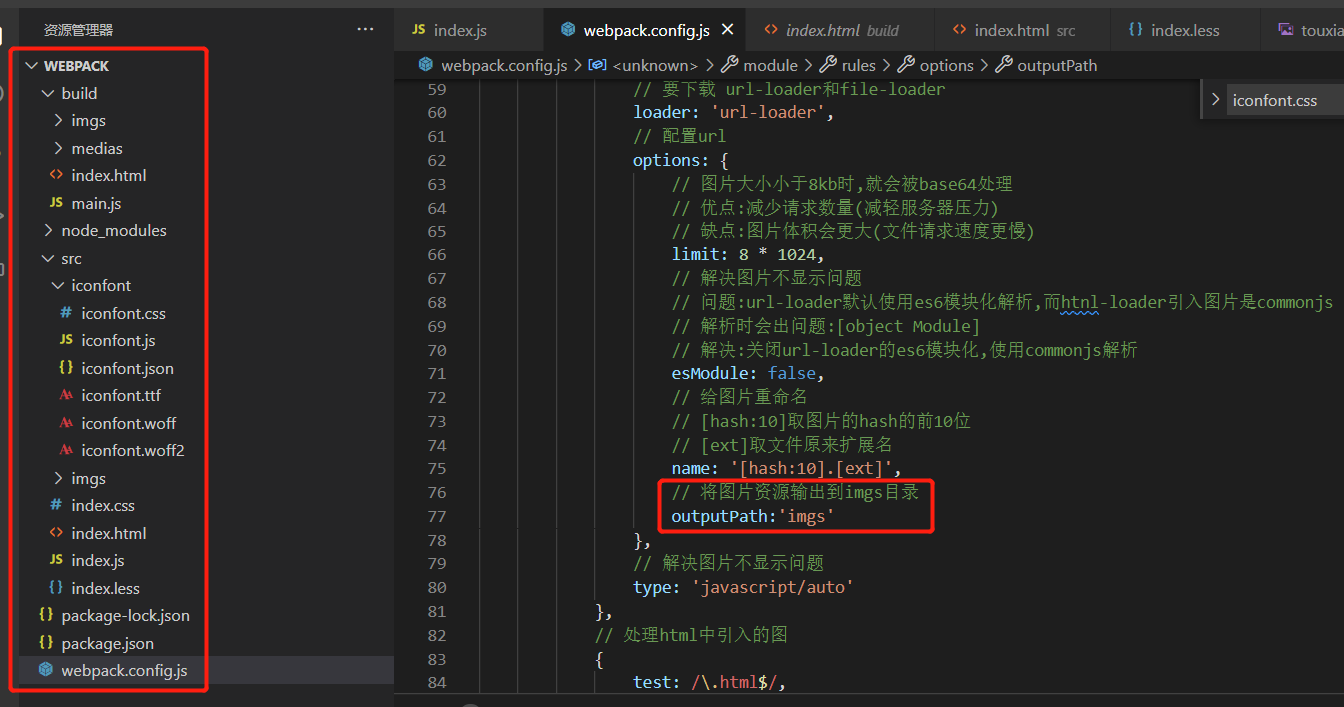
- webpack server

·打开浏览器输入http://localhost:3000/

# Webpack开发环境基本配置

·能让代码运行

·把以上代码整合在一起



// webpack配置文件

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

module.exports = {

    // webpack配置

    // 入口起点

    entry: './src/index.js',

    // 输出

    output: {

        // 输出的文件名

        filename: 'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test: /\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader'

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test: /\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    'style-loader',

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader'

                ]

            },

            {

                // 处理图片资源

                // 处理不了html中引入的img图片

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

                // 使用一个loader时

                // 要下载 url-loader和file-loader

                loader: 'url-loader',

                // 配置url

                options: {

                    // 图片大小小于8kb时,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,

                    // 解决图片不显示问题

                    // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                    // 解析时会出问题:[object Module]

                    // 解决:关闭url-loader的es6模块化,使用commonjs解析

                    esModule: false,

                    // 给图片重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到imgs目录

                    outputPath:'imgs'

                },

                // 解决图片不显示问题

                type: 'javascript/auto'

            },

            // 处理html中引入的图

            {

                test: /\.html$/,

                // 负责引入img,从而能被url-loader打包处理

                loader: 'html-loader'

            },

            // 打包其他资源(除了html/js/css以外的资源)

            {

                // 排除资源

                exclude: /\.(css|js|html|jpg|png|gif|less)$/,

                loader: 'file-loader',

                // 配置文件

                options: {

                    // 解决字体图标不显示问题

                    // 解析时会出问题:[object Module]

                    // 解决:关闭es6模块化,使用commonjs解析

                    esModule: false,

                    // 给文件重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到medias目录

                    outputPath:'medias'

                },

                // 解决字体图标不显示问题

                type: 'javascript/auto'

            }

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

            template: './src/index.html'

        })

    ],

    // 模式,指定development或production模式

    mode: 'development',

    // 开发服务器devServer:用来自动化（自动编译、自动打开浏览器、自动刷新浏览器~）

    // 只会在内存中编译打包，不会有任何输出

    // 启动devServer指令为webpack server

    // 需要下载webpack-dev-server

    devServer: {

        // 指定运行目录

        static: resolve(\_\_dirname, 'build'),

        // 启动gzip压缩

        compress: true,

        // 指定服务器端口号

        port: 3000,

        // 自动打开浏览器

        open:true

    },

}

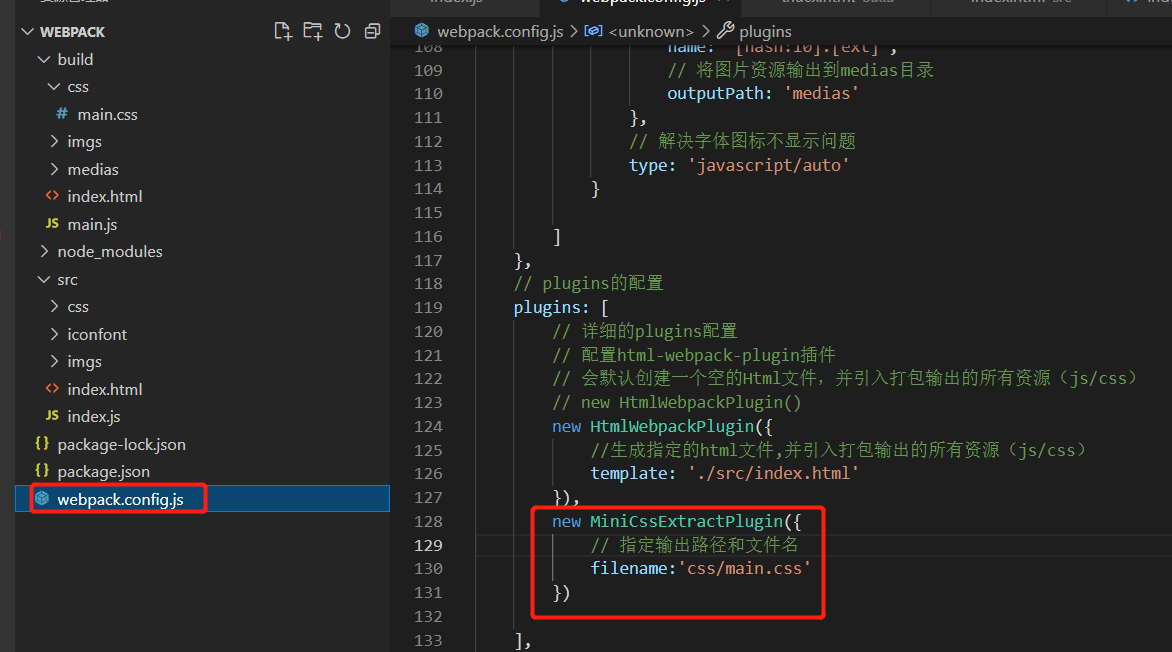
# Webpack生产环境的基本配置

## 提取css成单独文件

·下载mini-css-extract-plugin

- npm i mini-css-extract-plugin -D

·配置webpack



// webpack配置文件

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

// 引入mini-css-extract-plugin插件

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

module.exports = {

    // webpack配置

    // 入口起点

    entry: './src/index.js',

    // 输出

    output: {

        // 输出的文件名

        filename: 'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test: /\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader'

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test: /\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader'

                ]

            },

            {

                // 处理图片资源

                // 处理不了html中引入的img图片

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

                // 使用一个loader时

                // 要下载 url-loader和file-loader

                loader: 'url-loader',

                // 配置url

                options: {

                    // 图片大小小于8kb时,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,

                    // 解决图片不显示问题

                    // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                    // 解析时会出问题:[object Module]

                    // 解决:关闭url-loader的es6模块化,使用commonjs解析

                    esModule: false,

                    // 给图片重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到imgs目录

                    outputPath: 'imgs'

                },

                // 解决图片不显示问题

                type: 'javascript/auto'

            },

            // 处理html中引入的图

            {

                test: /\.html$/,

                // 负责引入img,从而能被url-loader打包处理

                loader: 'html-loader'

            },

            // 打包其他资源(除了html/js/css以外的资源)

            {

                // 排除资源

                exclude: /\.(css|js|html|jpg|png|gif|less)$/,

                loader: 'file-loader',

                // 配置文件

                options: {

                    // 解决字体图标不显示问题

                    // 解析时会出问题:[object Module]

                    // 解决:关闭es6模块化,使用commonjs解析

                    esModule: false,

                    // 给文件重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到medias目录

                    outputPath: 'medias'

                },

                // 解决字体图标不显示问题

                type: 'javascript/auto'

            }

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

            template: './src/index.html'

        }),

        new MiniCssExtractPlugin({

            // 指定输出路径和文件名

            filename:'css/main.css'

        })

    ],

    // 模式,指定development或production模式

    mode: 'development',

    // 开发服务器devServer:用来自动化（自动编译、自动打开浏览器、自动刷新浏览器~）

    // 只会在内存中编译打包，不会有任何输出

    // 启动devServer指令为webpack server

    // 需要下载webpack-dev-server

    devServer: {

        // 指定运行目录

        static: resolve(\_\_dirname, 'build'),

        // 启动gzip压缩

        compress: true,

        // 指定服务器端口号

        port: 3000,

        // 自动打开浏览器

        open: true

    },

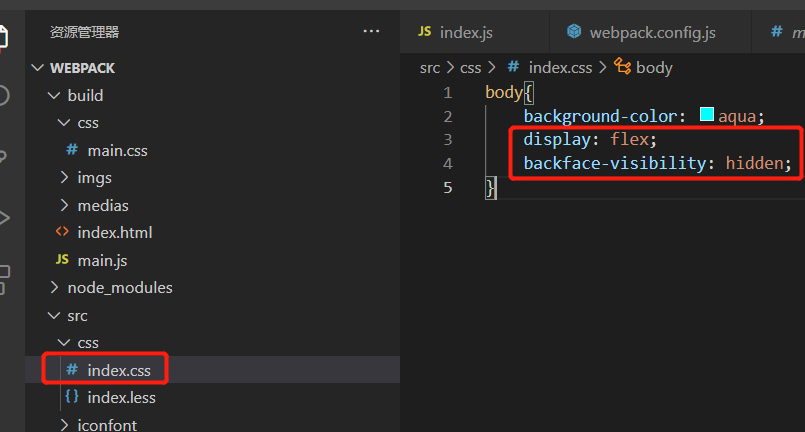
}

## Css兼容性处理

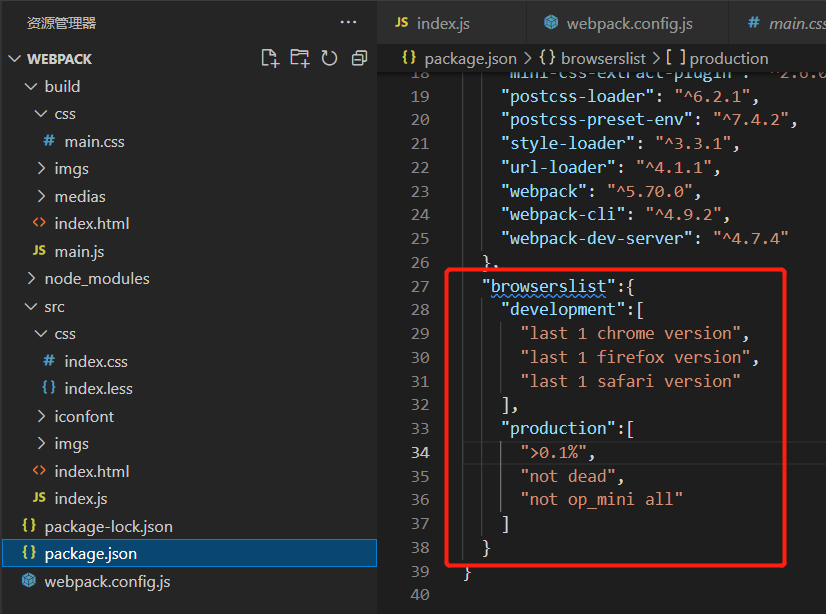
·下载postcss-loader postcss-preset-env

- npm i postcss-loader postcss-preset-env -D

·设置有兼容性问题的样式



·配置package.json



  "browserslist":{

    "development":[

      "last 1 chrome version",

      "last 1 firefox version",

      "last 1 safari version"

    ],

    "production":[

      ">0.1%",

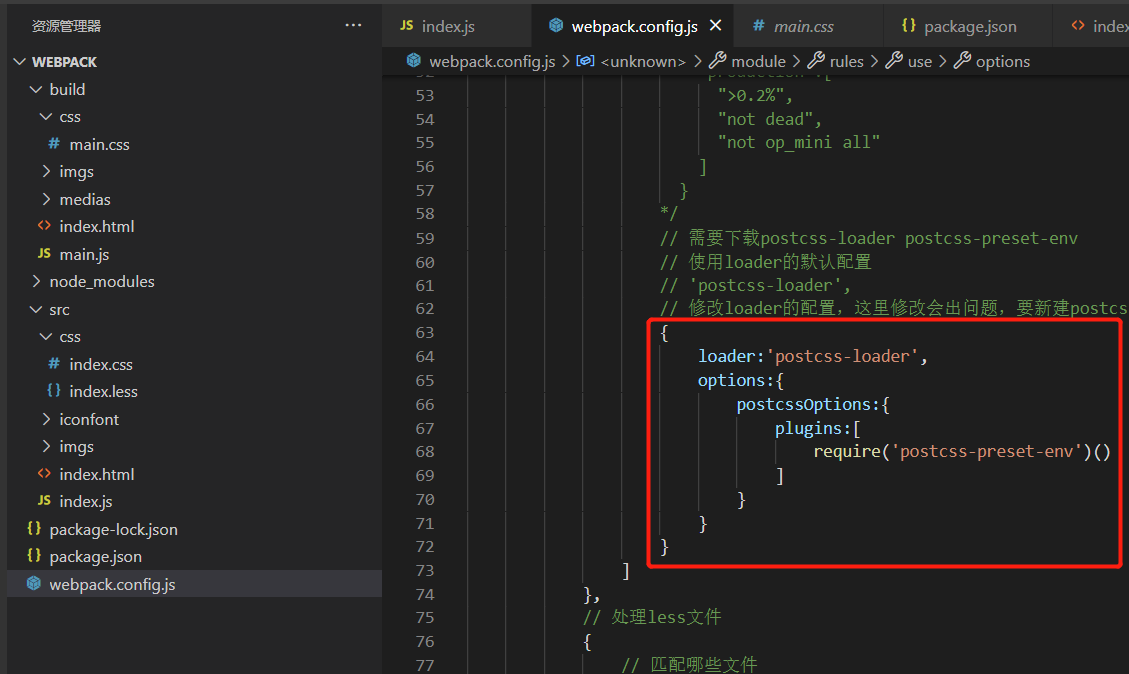
      "not dead",

      "not op\_mini all"

    ]

  }

·配置webpack



// webpack配置文件

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

// 引入mini-css-extract-plugin插件

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

// 设置nodejs环境变量

process.env.NODE\_ENV = 'development';

module.exports = {

    // webpack配置

    // 入口起点

    entry: './src/index.js',

    // 输出

    output: {

        // 输出的文件名

        filename: 'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test: /\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // css兼容性处理：postcss --> postcss-loader postcss-preset-env

                    // 帮postcss找到package.json中browserslist里面的配置。通过配置加载指定的css兼容性样式

                    /\*

                    "browserslist":{

                        // 开发环境 --> 设置node环境变量:process.env.NODE\_ENV = development

                        "development":[

                          "last 1 chrome version",

                          "last 1 firefox version",

                          "last 1 safari version"

                        ],

                        生产环境：默认

                        "production":[

                          ">0.2%",

                          "not dead",

                          "not op\_mini all"

                        ]

                      }

                    \*/

                    // 需要下载postcss-loader postcss-preset-env

                    // 使用loader的默认配置

                    // 'postcss-loader',

                    // 修改loader的配置，这里修改会出问题，要新建postcss.config.js文件修改才行

                    {

                        loader:'postcss-loader',

                        options:{

                            postcssOptions:{

                                plugins:[

                                    require('postcss-preset-env')()

                                ]

                            }

                        }

                    }

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test: /\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader',

                    {

                        loader:'postcss-loader',

                        options:{

                            postcssOptions:{

                                plugins:[

                                    require('postcss-preset-env')()

                                ]

                            }

                        }

                    }

                ]

            },

            {

                // 处理图片资源

                // 处理不了html中引入的img图片

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

                // 使用一个loader时

                // 要下载 url-loader和file-loader

                loader: 'url-loader',

                // 配置url

                options: {

                    // 图片大小小于8kb时,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,

                    // 解决图片不显示问题

                    // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                    // 解析时会出问题:[object Module]

                    // 解决:关闭url-loader的es6模块化,使用commonjs解析

                    esModule: false,

                    // 给图片重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到imgs目录

                    outputPath: 'imgs'

                },

                // 解决图片不显示问题

                type: 'javascript/auto'

            },

            // 处理html中引入的图

            {

                test: /\.html$/,

                // 负责引入img,从而能被url-loader打包处理

                loader: 'html-loader'

            },

            // 打包其他资源(除了html/js/css以外的资源)

            {

                // 排除资源

                exclude: /\.(css|js|html|jpg|png|gif|less)$/,

                loader: 'file-loader',

                // 配置文件

                options: {

                    // 解决字体图标不显示问题

                    // 解析时会出问题:[object Module]

                    // 解决:关闭es6模块化,使用commonjs解析

                    esModule: false,

                    // 给文件重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到medias目录

                    outputPath: 'medias'

                },

                // 解决字体图标不显示问题

                type: 'javascript/auto'

            }

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

            template: './src/index.html'

        }),

        new MiniCssExtractPlugin({

            // 指定输出路径和文件名

            filename:'css/main.css'

        })

    ],

    // 模式,指定development或production模式

    mode: 'development',

    // 开发服务器devServer:用来自动化（自动编译、自动打开浏览器、自动刷新浏览器~）

    // 只会在内存中编译打包，不会有任何输出

    // 启动devServer指令为webpack server

    // 需要下载webpack-dev-server

    devServer: {

        // 指定运行目录

        static: resolve(\_\_dirname, 'build'),

        // 启动gzip压缩

        compress: true,

        // 指定服务器端口号

        port: 3000,

        // 自动打开浏览器

        open: true

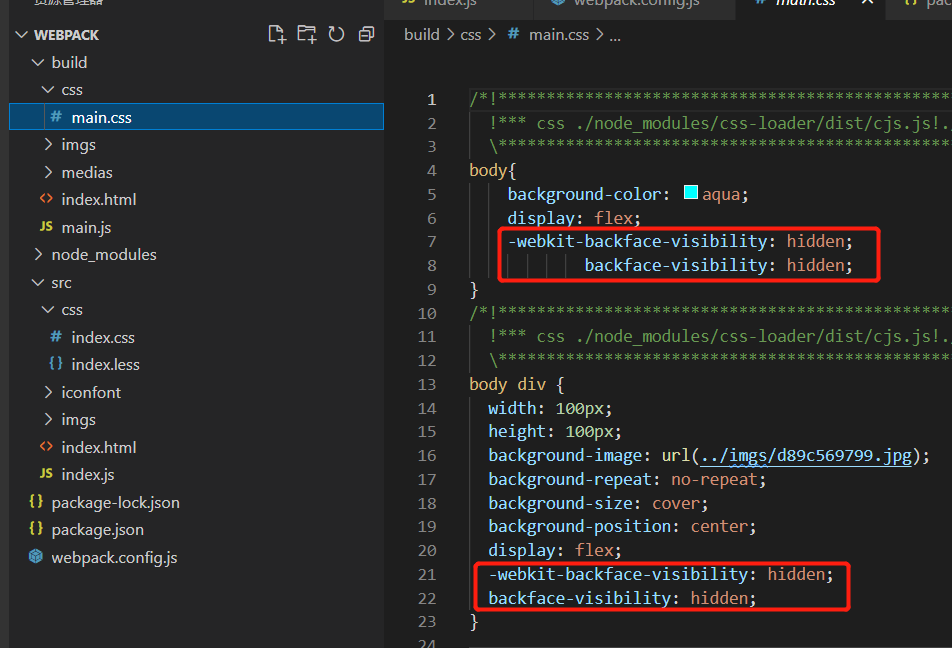
    },

}

·打包webpack

- webpack

·查看生成后的css文件

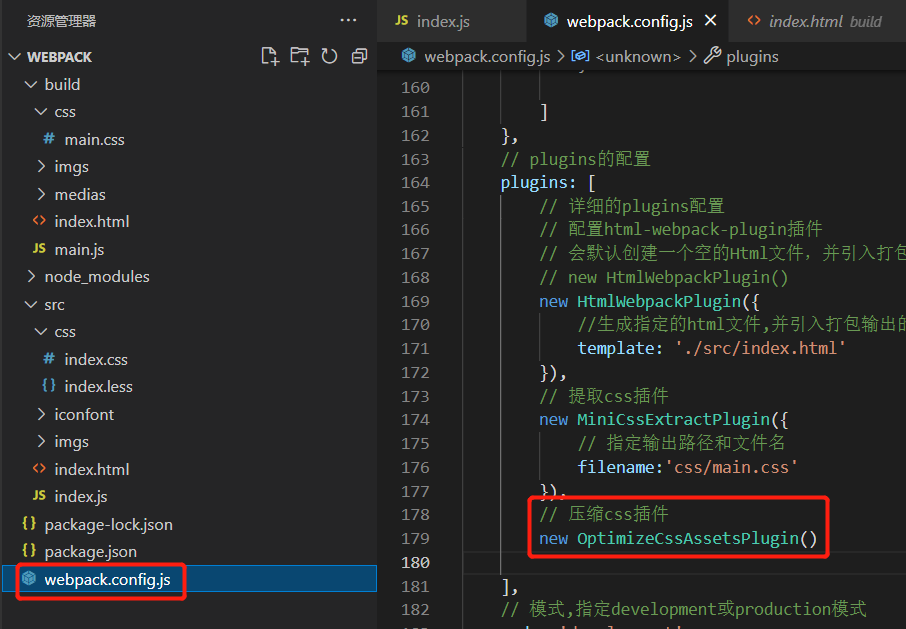


## 压缩css

·下载optimize-css-assets-webpack-plugin插件

- npm i optimize-css-assets-webpack-plugin -D

·配置webpack



// webpack配置文件

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

// 引入mini-css-extract-plugin插件

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

// 设置nodejs环境变量

process.env.NODE\_ENV = 'development';

// 引入optimize-css-assets-webpack-plugin

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

module.exports = {

    // webpack配置

    // 入口起点

    entry: './src/index.js',

    // 输出

    output: {

        // 输出的文件名

        filename: 'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test: /\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // css兼容性处理：postcss --> postcss-loader postcss-preset-env

                    // 帮postcss找到package.json中browserslist里面的配置。通过配置加载指定的css兼容性样式

                    /\*

                    "browserslist":{

                        // 开发环境 --> 设置node环境变量:process.env.NODE\_ENV = development

                        "development":[

                          "last 1 chrome version",

                          "last 1 firefox version",

                          "last 1 safari version"

                        ],

                        生产环境：默认

                        "production":[

                          ">0.2%",

                          "not dead",

                          "not op\_mini all"

                        ]

                      }

                    \*/

                    // 需要下载postcss-loader postcss-preset-env

                    // 使用loader的默认配置

                    // 'postcss-loader',

                    // 修改loader的配置，这里修改会出问题，要新建postcss.config.js文件修改才行

                    {

                        loader:'postcss-loader',

                        options:{

                            postcssOptions:{

                                plugins:[

                                    require('postcss-preset-env')()

                                ]

                            }

                        }

                    }

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test: /\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader',

                    {

                        loader:'postcss-loader',

                        options:{

                            postcssOptions:{

                                plugins:[

                                    require('postcss-preset-env')()

                                ]

                            }

                        }

                    }

                ]

            },

            {

                // 处理图片资源

                // 处理不了html中引入的img图片

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

                // 使用一个loader时

                // 要下载 url-loader和file-loader

                loader: 'url-loader',

                // 配置url

                options: {

                    // 图片大小小于8kb时,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,

                    // 解决图片不显示问题

                    // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                    // 解析时会出问题:[object Module]

                    // 解决:关闭url-loader的es6模块化,使用commonjs解析

                    esModule: false,

                    // 给图片重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到imgs目录

                    outputPath: 'imgs'

                },

                // 解决图片不显示问题

                type: 'javascript/auto'

            },

            // 处理html中引入的图

            {

                test: /\.html$/,

                // 负责引入img,从而能被url-loader打包处理

                loader: 'html-loader'

            },

            // 打包其他资源(除了html/js/css以外的资源)

            {

                // 排除资源

                exclude: /\.(css|js|html|jpg|png|gif|less)$/,

                loader: 'file-loader',

                // 配置文件

                options: {

                    // 解决字体图标不显示问题

                    // 解析时会出问题:[object Module]

                    // 解决:关闭es6模块化,使用commonjs解析

                    esModule: false,

                    // 给文件重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到medias目录

                    outputPath: 'medias'

                },

                // 解决字体图标不显示问题

                type: 'javascript/auto'

            }

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

            template: './src/index.html'

        }),

        // 提取css插件

        new MiniCssExtractPlugin({

            // 指定输出路径和文件名

            filename:'css/main.css'

        }),

        // 压缩css插件

        new OptimizeCssAssetsPlugin()

    ],

    // 模式,指定development或production模式

    mode: 'development',

    // 开发服务器devServer:用来自动化（自动编译、自动打开浏览器、自动刷新浏览器~）

    // 只会在内存中编译打包，不会有任何输出

    // 启动devServer指令为webpack server

    // 需要下载webpack-dev-server

    devServer: {

        // 指定运行目录

        static: resolve(\_\_dirname, 'build'),

        // 启动gzip压缩

        compress: true,

        // 指定服务器端口号

        port: 3000,

        // 自动打开浏览器

        open: true

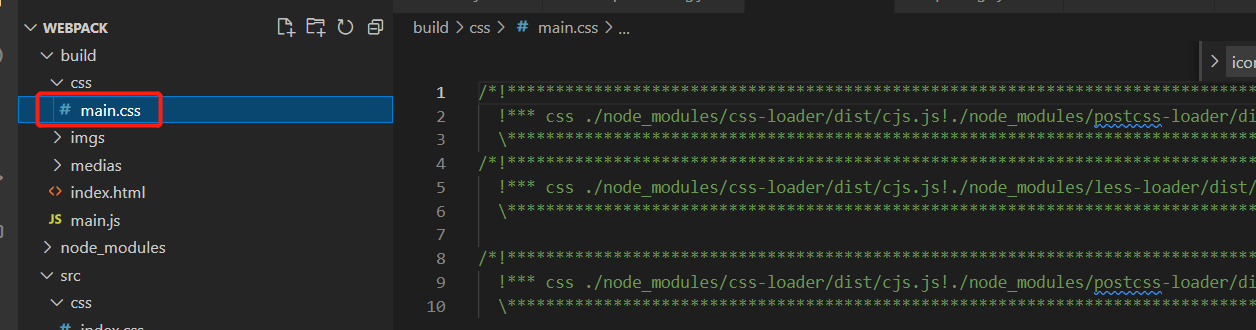
    },

}

·打包webpack

- webpack

·查看生成的css文件



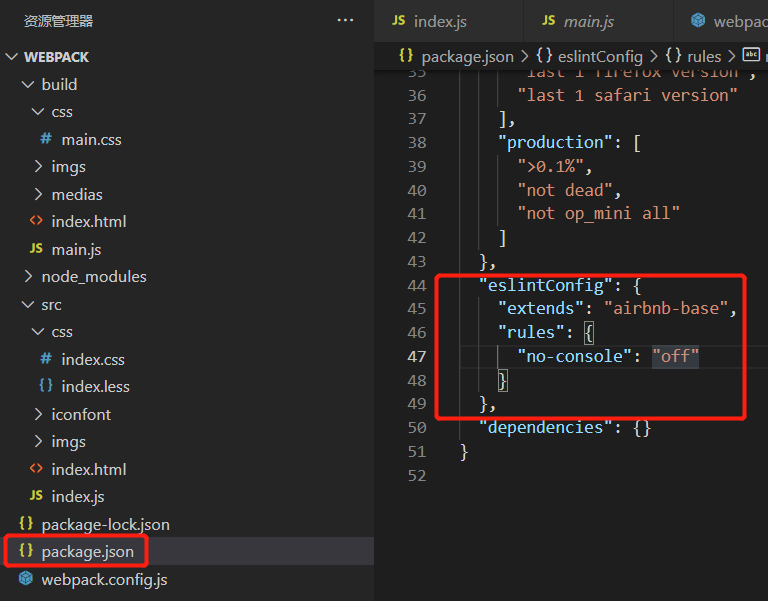
## Js语法检查eslint

·作用：检查js语法是否规范，可以自动修复不规范的语法

·下载eslint eslint-config-airbnb-base eslint-plugin-import

- npm i eslint eslint-config-airbnb-base eslint-plugin-import -D

·配置package.json



  "eslintConfig": {

    "extends": "airbnb-base",

    "rules": {

      "no-console": "off"

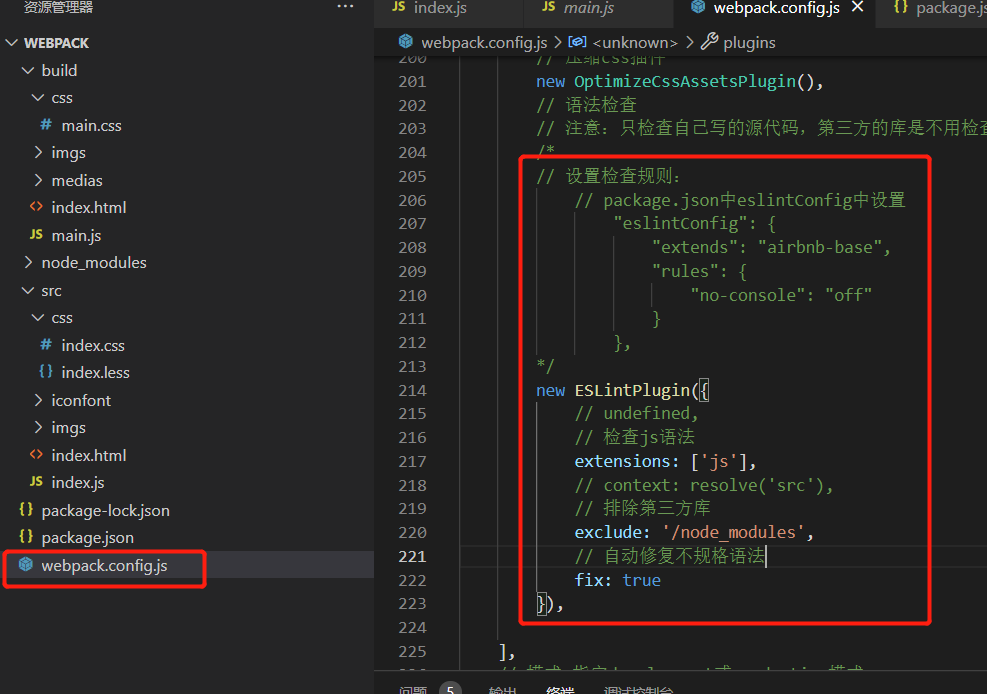
    }

  },

·下载eslint-webpack-plugin

- npm i eslint-webpack-plugin -D

·配置webpack



// webpack配置文件

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

// 引入mini-css-extract-plugin插件

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

// 设置nodejs环境变量

process.env.NODE\_ENV = 'development';

// 引入optimize-css-assets-webpack-plugin

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

// 引入eslint-webpack-plugin

const ESLintPlugin = require('eslint-webpack-plugin');

module.exports = {

    // webpack配置

    // 入口起点

    entry: './src/index.js',

    // 输出

    output: {

        // 输出的文件名

        filename: 'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test: /\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // css兼容性处理：postcss --> postcss-loader postcss-preset-env

                    // 帮postcss找到package.json中browserslist里面的配置。通过配置加载指定的css兼容性样式

                    /\*

                    "browserslist":{

                        // 开发环境 --> 设置node环境变量:process.env.NODE\_ENV = development

                        "development":[

                          "last 1 chrome version",

                          "last 1 firefox version",

                          "last 1 safari version"

                        ],

                        生产环境：默认

                        "production":[

                          ">0.2%",

                          "not dead",

                          "not op\_mini all"

                        ]

                      }

                    \*/

                    // 需要下载postcss-loader postcss-preset-env

                    // 使用loader的默认配置

                    // 'postcss-loader',

                    // 修改loader的配置，这里修改会出问题，要新建postcss.config.js文件修改才行

                    {

                        loader: 'postcss-loader',

                        options: {

                            postcssOptions: {

                                plugins: [

                                    require('postcss-preset-env')()

                                ]

                            }

                        }

                    }

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test: /\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader',

                    {

                        loader: 'postcss-loader',

                        options: {

                            postcssOptions: {

                                plugins: [

                                    require('postcss-preset-env')()

                                ]

                            }

                        }

                    }

                ]

            },

            {

                // 处理图片资源

                // 处理不了html中引入的img图片

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

                // 使用一个loader时

                // 要下载 url-loader和file-loader

                loader: 'url-loader',

                // 配置url

                options: {

                    // 图片大小小于8kb时,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,

                    // 解决图片不显示问题

                    // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                    // 解析时会出问题:[object Module]

                    // 解决:关闭url-loader的es6模块化,使用commonjs解析

                    esModule: false,

                    // 给图片重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到imgs目录

                    outputPath: 'imgs'

                },

                // 解决图片不显示问题

                type: 'javascript/auto'

            },

            // 处理html中引入的图

            {

                test: /\.html$/,

                // 负责引入img,从而能被url-loader打包处理

                loader: 'html-loader'

            },

            // 打包其他资源(除了html/js/css以外的资源)

            {

                // 排除资源

                exclude: /\.(css|js|html|jpg|png|gif|less)$/,

                loader: 'file-loader',

                // 配置文件

                options: {

                    // 解决字体图标不显示问题

                    // 解析时会出问题:[object Module]

                    // 解决:关闭es6模块化,使用commonjs解析

                    esModule: false,

                    // 给文件重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到medias目录

                    outputPath: 'medias'

                },

                // 解决字体图标不显示问题

                type: 'javascript/auto'

            },

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

            template: './src/index.html'

        }),

        // 提取css插件

        new MiniCssExtractPlugin({

            // 指定输出路径和文件名

            filename: 'css/main.css'

        }),

        // 压缩css插件

        new OptimizeCssAssetsPlugin(),

        // 语法检查

        // 注意：只检查自己写的源代码，第三方的库是不用检查的

        /\*

        // 设置检查规则：

            // package.json中eslintConfig中设置

                "eslintConfig": {

                    "extends": "airbnb-base",

                    "rules": {

                        "no-console": "off"

                    }

                },

        \*/

        new ESLintPlugin({

            // undefined,

            // 检查js语法

            extensions: ['js'],

            // context: resolve('src'),

            // 排除第三方库

            exclude: '/node\_modules',

            // 自动修复不规格语法

            fix: true

        }),

    ],

    // 模式,指定development或production模式

    mode: 'development',

    // 开发服务器devServer:用来自动化（自动编译、自动打开浏览器、自动刷新浏览器~）

    // 只会在内存中编译打包，不会有任何输出

    // 启动devServer指令为webpack server

    // 需要下载webpack-dev-server

    devServer: {

        // 指定运行目录

        static: resolve(\_\_dirname, 'build'),

        // 启动gzip压缩

        compress: true,

        // 指定服务器端口号

        port: 3000,

        // 自动打开浏览器

        open: true,

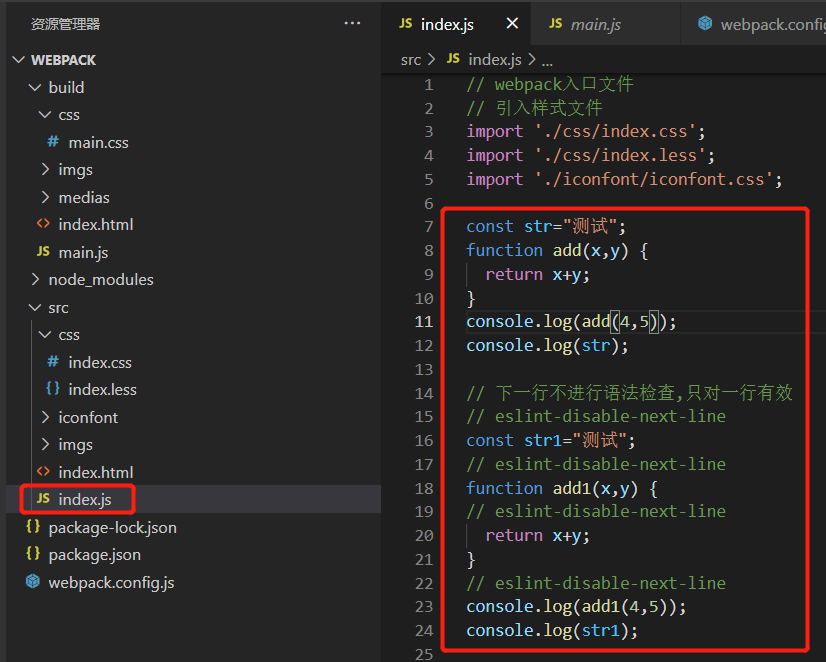
        // 自动刷新

        watchFiles: ['./src/index.html']

    },

}

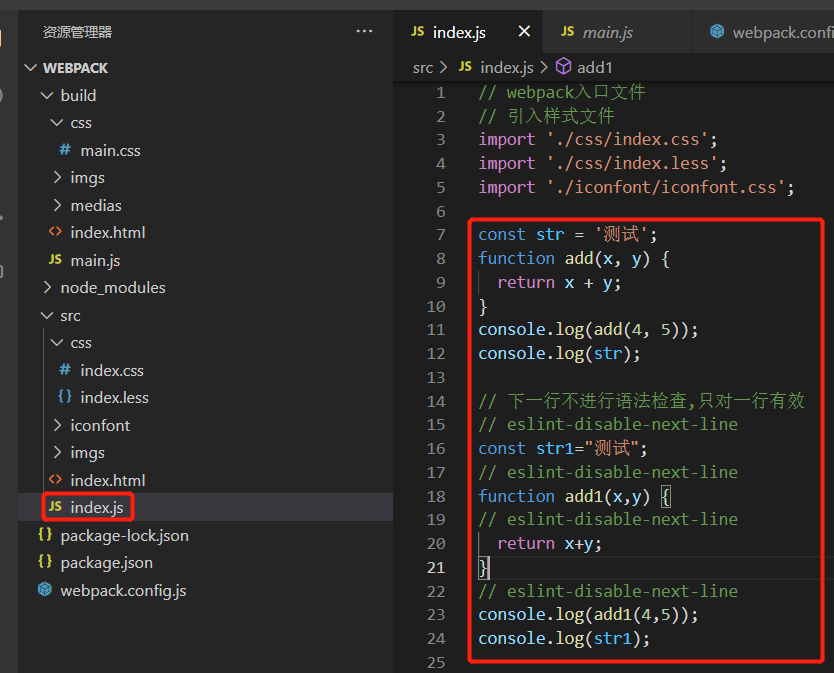
·打包前



·打包webpack

- webpack

·打包后



## Js兼容性处理

·作用：使js代码兼容大多数浏览器

·下载babel-loader @babel/preset-env @babel/core

- npm i babel-loader @babel/preset-env @babel/core -D

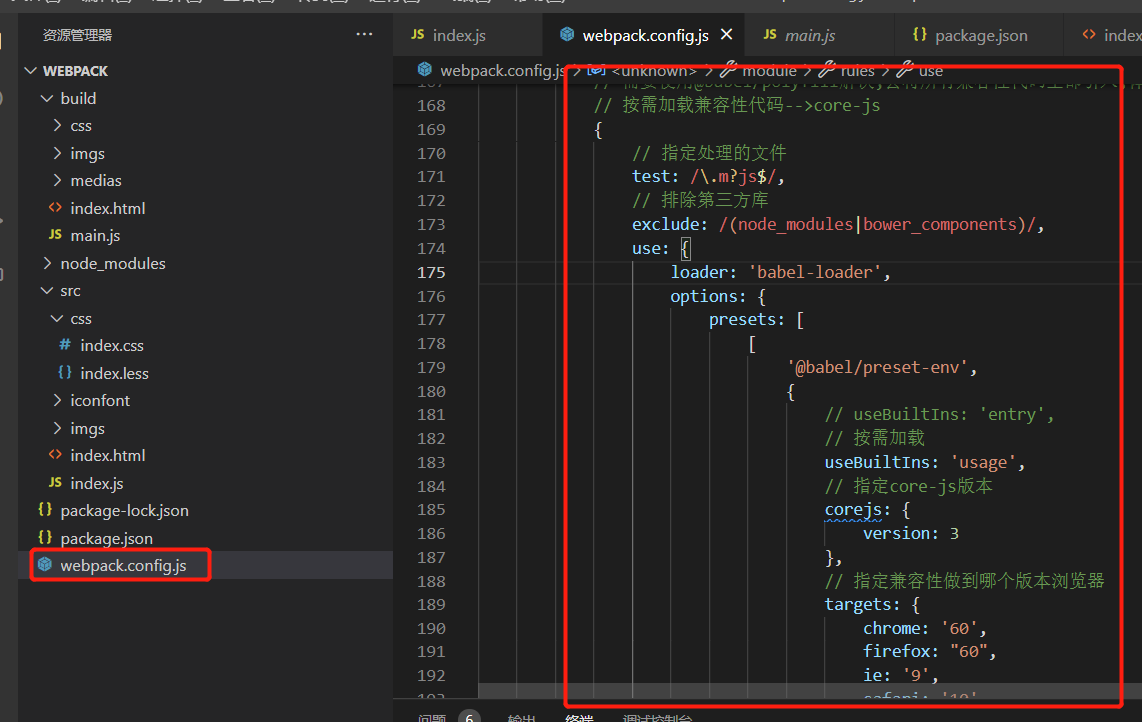
·下载@babel/polyfill @babel/plugin-transform-runtime

- npm i @babel/polyfill @babel/plugin-transform-runtime -D

·下载core-js

- npm i core-js -D

·配置webpack



// webpack配置文件

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

// 引入mini-css-extract-plugin插件

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

// 设置nodejs环境变量

process.env.NODE\_ENV = 'development';

// 引入optimize-css-assets-webpack-plugin

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

// 引入eslint-webpack-plugin

const ESLintPlugin = require('eslint-webpack-plugin');

module.exports = {

    // webpack配置

    // 入口起点

    entry: './src/index.js',

    // 引入@babel/polyfill

    // entry: ['@babel/polyfill','./src/index.js'],

    // 输出

    output: {

        // 输出的文件名

        filename: 'main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test: /\.css$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // css兼容性处理：postcss --> postcss-loader postcss-preset-env

                    // 帮postcss找到package.json中browserslist里面的配置。通过配置加载指定的css兼容性样式

                    /\*

                    "browserslist":{

                        // 开发环境 --> 设置node环境变量:process.env.NODE\_ENV = development

                        "development":[

                          "last 1 chrome version",

                          "last 1 firefox version",

                          "last 1 safari version"

                        ],

                        生产环境：默认

                        "production":[

                          ">0.2%",

                          "not dead",

                          "not op\_mini all"

                        ]

                      }

                    \*/

                    // 需要下载postcss-loader postcss-preset-env

                    // 使用loader的默认配置

                    // 'postcss-loader',

                    // 修改loader的配置，这里修改会出问题，要新建postcss.config.js文件修改才行

                    {

                        loader: 'postcss-loader',

                        options: {

                            postcssOptions: {

                                plugins: [

                                    require('postcss-preset-env')()

                                ]

                            }

                        }

                    }

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test: /\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    'css-loader',

                    // 将less文件编译成css文件

                    'less-loader',

                    {

                        loader: 'postcss-loader',

                        options: {

                            postcssOptions: {

                                plugins: [

                                    require('postcss-preset-env')()

                                ]

                            }

                        }

                    }

                ]

            },

            {

                // 处理图片资源

                // 处理不了html中引入的img图片

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

                // 使用一个loader时

                // 要下载 url-loader和file-loader

                loader: 'url-loader',

                // 配置url

                options: {

                    // 图片大小小于8kb时,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,

                    // 解决图片不显示问题

                    // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                    // 解析时会出问题:[object Module]

                    // 解决:关闭url-loader的es6模块化,使用commonjs解析

                    esModule: false,

                    // 给图片重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到imgs目录

                    outputPath: 'imgs'

                },

                // 解决图片不显示问题

                type: 'javascript/auto'

            },

            // 处理html中引入的图

            {

                test: /\.html$/,

                // 负责引入img,从而能被url-loader打包处理

                loader: 'html-loader'

            },

            // 打包其他资源(除了html/js/css以外的资源)

            {

                // 排除资源

                exclude: /\.(css|js|html|jpg|png|gif|less)$/,

                loader: 'file-loader',

                // 配置文件

                options: {

                    // 解决字体图标不显示问题

                    // 解析时会出问题:[object Module]

                    // 解决:关闭es6模块化,使用commonjs解析

                    esModule: false,

                    // 给文件重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到medias目录

                    outputPath: 'medias'

                },

                // 解决字体图标不显示问题

                type: 'javascript/auto'

            },

            // js兼容性处理

            // 需要下载babel-loader @babel/preset-env @babel/core

            // 问题:只能转换基本语法,promise等语法不能转换

            // 需要使用@babel/polyfill解决,会将所有兼容性代码全部引入,体积大

            // 按需加载兼容性代码-->core-js

            {

                // 指定处理的文件

                test: /\.m?js$/,

                // 排除第三方库

                exclude: /(node\_modules|bower\_components)/,

                use: {

                    loader: 'babel-loader',

                    options: {

                        presets: [

                            [

                                '@babel/preset-env',

                                {

                                    // useBuiltIns: 'entry',

                                    // 按需加载

                                    useBuiltIns: 'usage',

                                    // 指定core-js版本

                                    corejs: {

                                        version: 3

                                    },

                                    // 指定兼容性做到哪个版本浏览器

                                    targets: {

                                        chrome: '60',

                                        firefox: "60",

                                        ie: '9',

                                        safari: '10',

                                        edge: '17'

                                    }

                                }

                            ]

                        ],

                        //利用 @babel/plugin-transform-runtime 插件还能以沙箱垫片的方式防止污染全局， 并抽离公共的 helper function , 以节省代码的冗余

                        "plugins": ["@babel/plugin-transform-runtime"]

                    }

                }

            }

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

            template: './src/index.html'

        }),

        // 提取css插件

        new MiniCssExtractPlugin({

            // 指定输出路径和文件名

            filename: 'css/main.css'

        }),

        // 压缩css插件

        new OptimizeCssAssetsPlugin(),

        // 语法检查

        // 注意：只检查自己写的源代码，第三方的库是不用检查的

        /\*

        // 设置检查规则：

            // package.json中eslintConfig中设置

                "eslintConfig": {

                    "extends": "airbnb-base",

                    "rules": {

                        "no-console": "off"

                    }

                },

        \*/

        new ESLintPlugin({

            // undefined,

            // 检查js语法

            extensions: ['js'],

            // context: resolve('src'),

            // 排除第三方库

            exclude: '/node\_modules',

            // 自动修复不规格语法

            fix: true

        }),

    ],

    // 模式,指定development或production模式

    mode: 'development',

    // 开发服务器devServer:用来自动化（自动编译、自动打开浏览器、自动刷新浏览器~）

    // 只会在内存中编译打包，不会有任何输出

    // 启动devServer指令为webpack server

    // 需要下载webpack-dev-server

    devServer: {

        // 指定运行目录

        static: resolve(\_\_dirname, 'build'),

        // 启动gzip压缩

        compress: true,

        // 指定服务器端口号

        port: 3000,

        // 自动打开浏览器

        open: true,

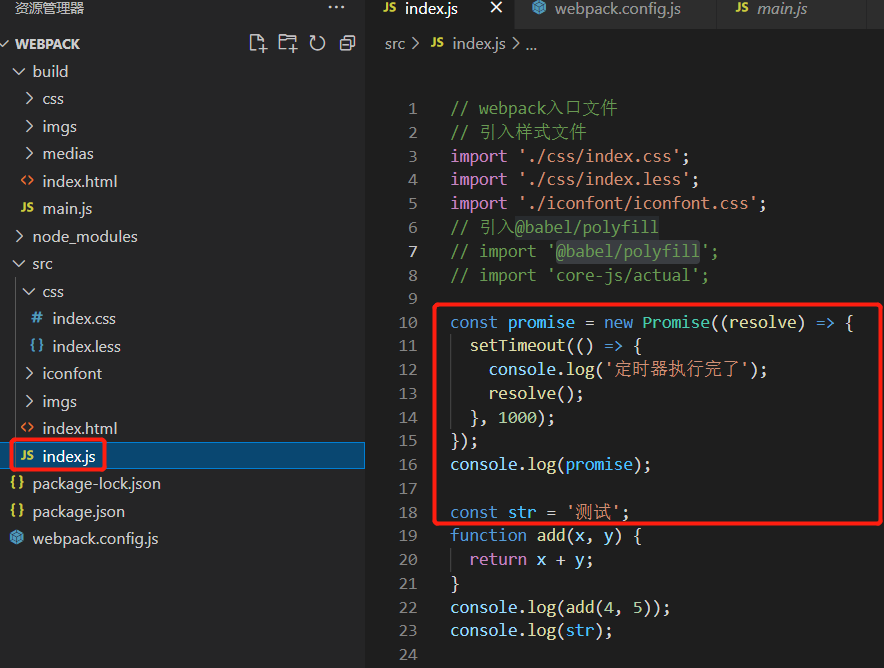
        // 自动刷新

        watchFiles: ['./src/index.html']

    },

}

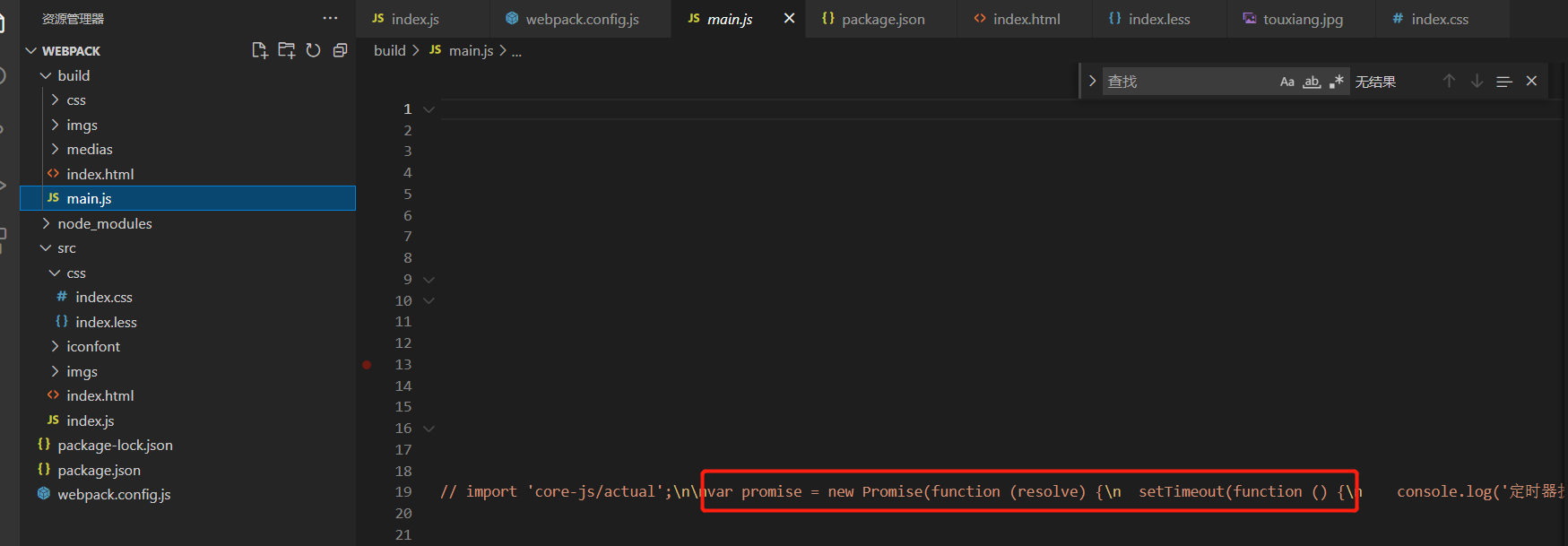
·源代码



·打包webpack

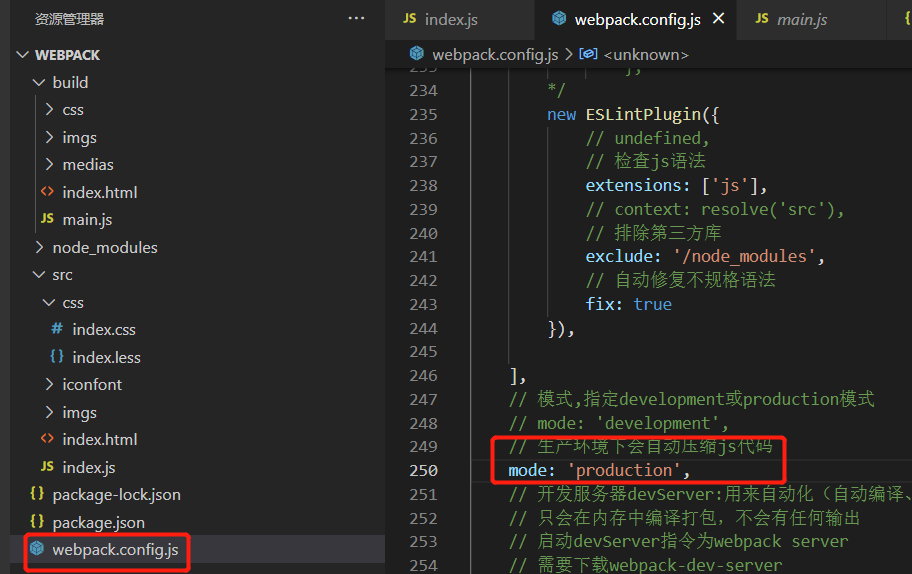
- webpack

·打包后代码



## js压缩

·生产环境下会自动压缩js代码



·打包webpack

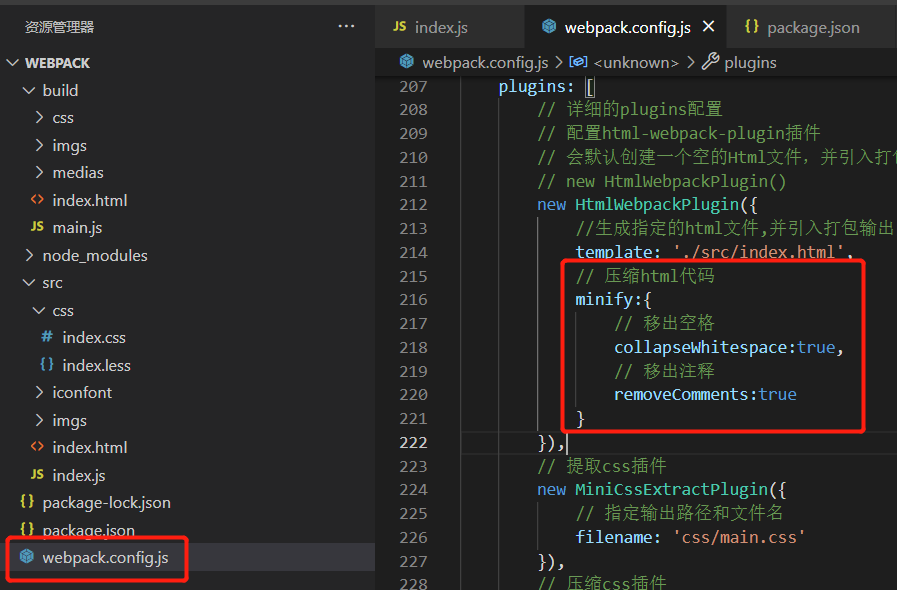
- webpack

·打包后的js文件

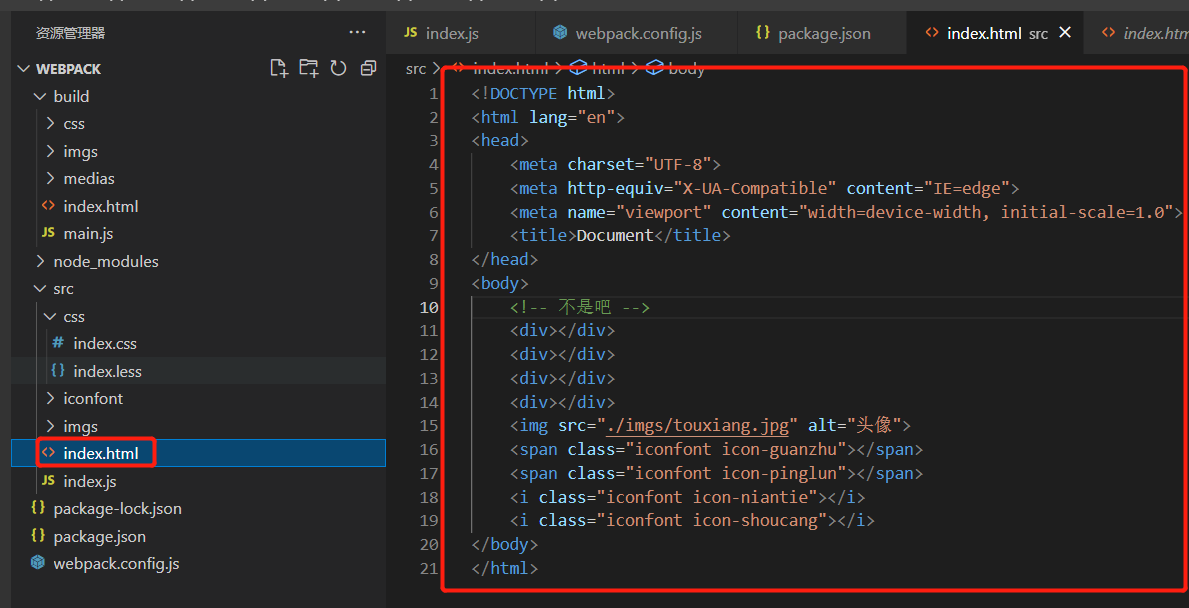


## html压缩

·配置webpack



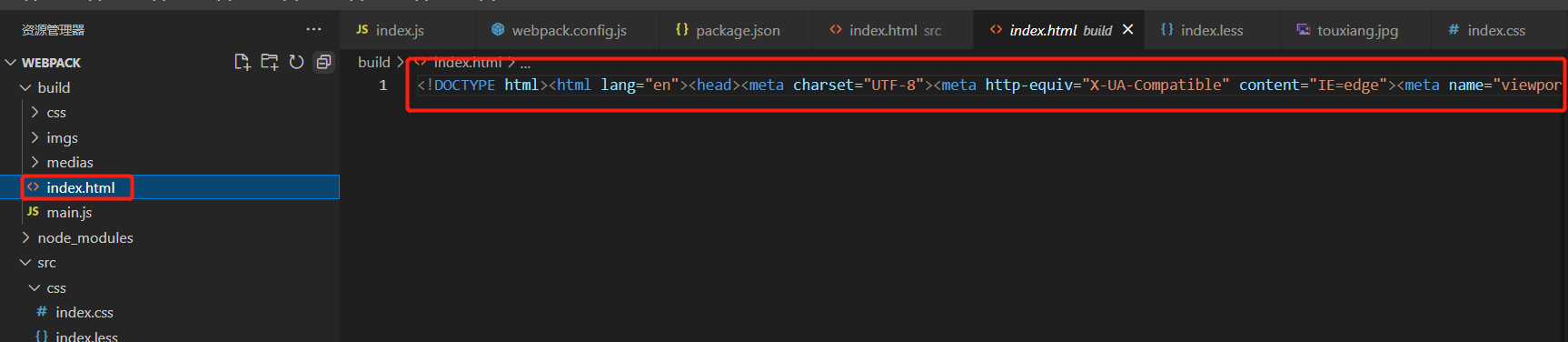
·源代码



·打包webpack

- webpack

·打包后

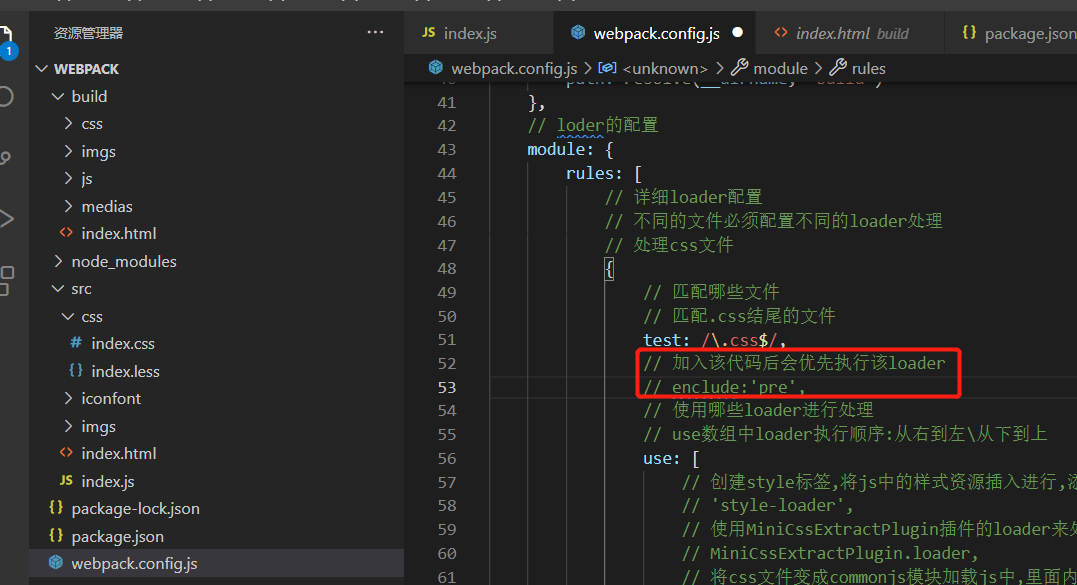


## 生产环境基本配置

·下载之前学过的模块

·正常来讲，一个文件只能被一个loader处理，当一个文件要被多个loader处理，一定要指定loader执行的先后顺序

- 比如先执行eslint再执行babel



·配置webpack

// webpack配置文件

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

// 引入mini-css-extract-plugin插件

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

// 设置nodejs环境变量

process.env.NODE\_ENV = 'development';

// 引入optimize-css-assets-webpack-plugin

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

// 引入eslint-webpack-plugin

const ESLintPlugin = require('eslint-webpack-plugin');

// 复用loader

const commonCSSLoader = [

    MiniCssExtractPlugin.loader,

    'css-loader',

    {

        loader: 'postcss-loader',

        options: {

            postcssOptions: {

                plugins: [

                    require('postcss-preset-env')()

                ]

            }

        }

    }];

module.exports = {

    // webpack配置

    // 入口起点

    entry: './src/index.js',

    // 引入@babel/polyfill

    // entry: ['@babel/polyfill','./src/index.js'],

    // 输出

    output: {

        // 输出的文件名

        filename: 'js/main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            // 详细loader配置

            // 不同的文件必须配置不同的loader处理

            // 处理css文件

            {

                // 匹配哪些文件

                // 匹配.css结尾的文件

                test: /\.css$/,

                // 加入该代码后会优先执行该loader

                // enclude:'pre',

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    // MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    // 'css-loader',

                    // css兼容性处理：postcss --> postcss-loader postcss-preset-env

                    // 帮postcss找到package.json中browserslist里面的配置。通过配置加载指定的css兼容性样式

                    /\*

                    "browserslist":{

                        // 开发环境 --> 设置node环境变量:process.env.NODE\_ENV = development

                        "development":[

                          "last 1 chrome version",

                          "last 1 firefox version",

                          "last 1 safari version"

                        ],

                        生产环境：默认

                        "production":[

                          ">0.2%",

                          "not dead",

                          "not op\_mini all"

                        ]

                      }

                    \*/

                    // 需要下载postcss-loader postcss-preset-env

                    // 使用loader的默认配置

                    // 'postcss-loader',

                    // 修改loader的配置，这里修改会出问题，要新建postcss.config.js文件修改才行

                    // {

                    //     loader: 'postcss-loader',

                    //     options: {

                    //         postcssOptions: {

                    //             plugins: [

                    //                 require('postcss-preset-env')()

                    //             ]

                    //         }

                    //     }

                    // }

                    // 直接使用复用cssloader

                    ...commonCSSLoader

                ]

            },

            // 处理less文件

            {

                // 匹配哪些文件

                // 匹配.less结尾的文件

                test: /\.less$/,

                // 使用哪些loader进行处理

                // use数组中loader执行顺序:从右到左\从下到上

                use: [

                    // 这个一定要在less-loader前面

                    ...commonCSSLoader,

                    // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                    // 'style-loader',

                    // 使用MiniCssExtractPlugin插件的loader来处理

                    // MiniCssExtractPlugin.loader,

                    // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                    // 'css-loader',

                    // 将less文件编译成css文件

                    'less-loader'

                    // {

                    //     loader: 'postcss-loader',

                    //     options: {

                    //         postcssOptions: {

                    //             plugins: [

                    //                 require('postcss-preset-env')()

                    //             ]

                    //         }

                    //     }

                    // }

                    // 直接使用复用cssloader

                ]

            },

            {

                // 处理图片资源

                // 处理不了html中引入的img图片

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

                // 使用一个loader时

                // 要下载 url-loader和file-loader

                loader: 'url-loader',

                // 配置url

                options: {

                    // 图片大小小于8kb时,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,

                    // 解决图片不显示问题

                    // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                    // 解析时会出问题:[object Module]

                    // 解决:关闭url-loader的es6模块化,使用commonjs解析

                    esModule: false,

                    // 给图片重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到imgs目录

                    outputPath: 'imgs'

                },

                // 解决图片不显示问题

                type: 'javascript/auto'

            },

            // 处理html中引入的图

            {

                test: /\.html$/,

                // 负责引入img,从而能被url-loader打包处理

                loader: 'html-loader'

            },

            // 打包其他资源(除了html/js/css以外的资源)

            {

                // 排除资源

                exclude: /\.(css|js|html|jpg|png|gif|less)$/,

                loader: 'file-loader',

                // 配置文件

                options: {

                    // 解决字体图标不显示问题

                    // 解析时会出问题:[object Module]

                    // 解决:关闭es6模块化,使用commonjs解析

                    esModule: false,

                    // 给文件重命名

                    // [hash:10]取图片的hash的前10位

                    // [ext]取文件原来扩展名

                    name: '[hash:10].[ext]',

                    // 将图片资源输出到medias目录

                    outputPath: 'medias'

                },

                // 解决字体图标不显示问题

                type: 'javascript/auto'

            },

            // js兼容性处理

            // 需要下载babel-loader @babel/preset-env @babel/core

            // 问题:只能转换基本语法,promise等语法不能转换

            // 需要使用@babel/polyfill解决,会将所有兼容性代码全部引入,体积大

            // 按需加载兼容性代码-->core-js

            {

                // 指定处理的文件

                test: /\.m?js$/,

                // 排除第三方库

                exclude: /(node\_modules|bower\_components)/,

                use: {

                    loader: 'babel-loader',

                    options: {

                        presets: [

                            [

                                '@babel/preset-env',

                                {

                                    // useBuiltIns: 'entry',

                                    // 按需加载

                                    useBuiltIns: 'usage',

                                    // 指定core-js版本

                                    corejs: {

                                        version: 3

                                    },

                                    // 指定兼容性做到哪个版本浏览器

                                    targets: {

                                        chrome: '60',

                                        firefox: "60",

                                        ie: '9',

                                        safari: '10',

                                        edge: '17'

                                    }

                                }

                            ]

                        ],

                        //利用 @babel/plugin-transform-runtime 插件还能以沙箱垫片的方式防止污染全局， 并抽离公共的 helper function , 以节省代码的冗余

                        "plugins": ["@babel/plugin-transform-runtime"]

                    }

                }

            }

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

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

            // 压缩html代码

            minify: {

                // 移出空格

                collapseWhitespace: true,

                // 移出注释

                removeComments: true

            }

        }),

        // 提取css插件

        new MiniCssExtractPlugin({

            // 指定输出路径和文件名

            filename: 'css/main.css'

        }),

        // 压缩css插件

        new OptimizeCssAssetsPlugin(),

        // 语法检查

        // 注意：只检查自己写的源代码，第三方的库是不用检查的

        /\*

        // 设置检查规则：

            // package.json中eslintConfig中设置

                "eslintConfig": {

                    "extends": "airbnb-base",

                    "rules": {

                        "no-console": "off"

                    }

                },

        \*/

        new ESLintPlugin({

            // undefined,

            // 检查js语法

            extensions: ['js'],

            // context: resolve('src'),

            // 排除第三方库

            exclude: '/node\_modules',

            // 自动修复不规格语法

            fix: true

        }),

    ],

    // 模式,指定development或production模式

    // mode: 'development',

    // 生产环境下会自动压缩js代码

    mode: 'production',

    // 开发服务器devServer:用来自动化（自动编译、自动打开浏览器、自动刷新浏览器~）

    // 只会在内存中编译打包，不会有任何输出

    // 启动devServer指令为webpack server

    // 需要下载webpack-dev-server

    devServer: {

        // 指定运行目录

        static: resolve(\_\_dirname, 'build'),

        // 启动gzip压缩

        compress: true,

        // 指定服务器端口号

        port: 3000,

        // 自动打开浏览器

        open: true,

        // 自动刷新

        watchFiles: ['./src/index.html']

    },

}

·配置package.json

{

  "name": "webpack\_test",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "",

  "license": "ISC",

  "devDependencies": {

    "@babel/core": "^7.17.5",

    "@babel/plugin-transform-runtime": "^7.17.0",

    "@babel/polyfill": "^7.12.1",

    "@babel/preset-env": "^7.16.11",

    "babel-loader": "^8.2.3",

    "core-js": "^3.21.1",

    "css-loader": "^6.6.0",

    "eslint": "^8.10.0",

    "eslint-config-airbnb-base": "^15.0.0",

    "eslint-plugin-import": "^2.25.4",

    "eslint-webpack-plugin": "^3.1.1",

    "file-loader": "^6.2.0",

    "html-loader": "^3.1.0",

    "html-webpack-plugin": "^5.5.0",

    "less": "^4.1.2",

    "less-loader": "^10.2.0",

    "mini-css-extract-plugin": "^2.6.0",

    "optimize-css-assets-webpack-plugin": "^6.0.1",

    "postcss-loader": "^6.2.1",

    "postcss-preset-env": "^7.4.2",

    "style-loader": "^3.3.1",

    "url-loader": "^4.1.1",

    "webpack": "^5.70.0",

    "webpack-cli": "^4.9.2",

    "webpack-dev-server": "^4.7.4"

  },

  "browserslist": {

    "development": [

      "last 1 chrome version",

      "last 1 firefox version",

      "last 1 safari version"

    ],

    "production": [

      ">0.1%",

      "not dead",

      "not op\_mini all"

    ]

  },

  "eslintConfig": {

    "extends": "airbnb-base",

    "rules": {

      "no-console": "off"

    }

  },

  "dependencies": {}

}

·打包webpack

- webpack

# Webpack优化配置

## 性能优化介绍

### 开发环境性能优化

·优化打包构建速度

·优化代码调试

### 生产环境性能优化

·优化打包构建速度

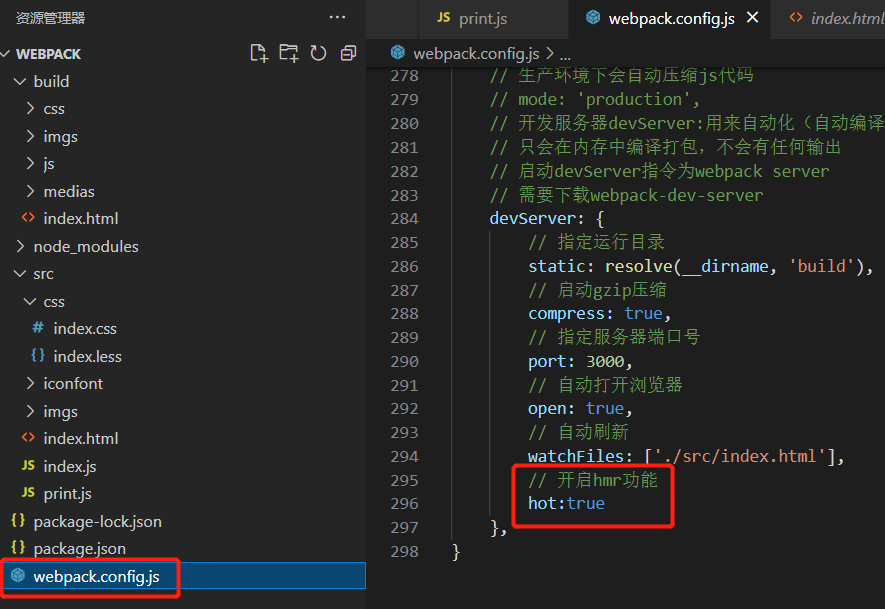
·优化代码运行性能

## HMR

·hot module replacement 热模块替换

·只加载所修改的模块，优化打包构建速度

·开启HMR



·样式文件

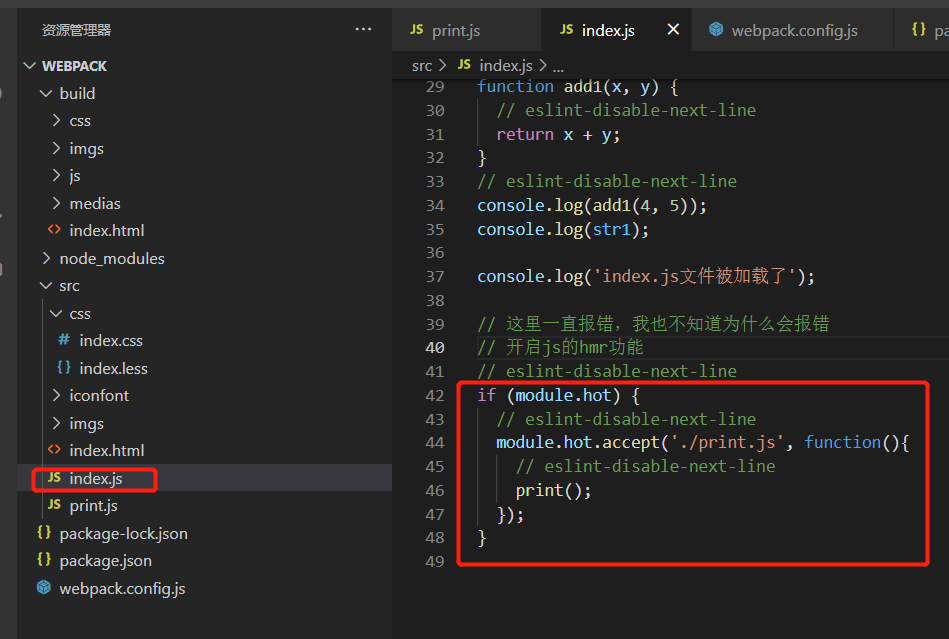
- 可以使用HMR功能

- 因为sty-loader内部实现了

·js文件

- 默认没有使用HMR功能

- 开启HMR：需要修改js代码，添加支持HMR功能的代码

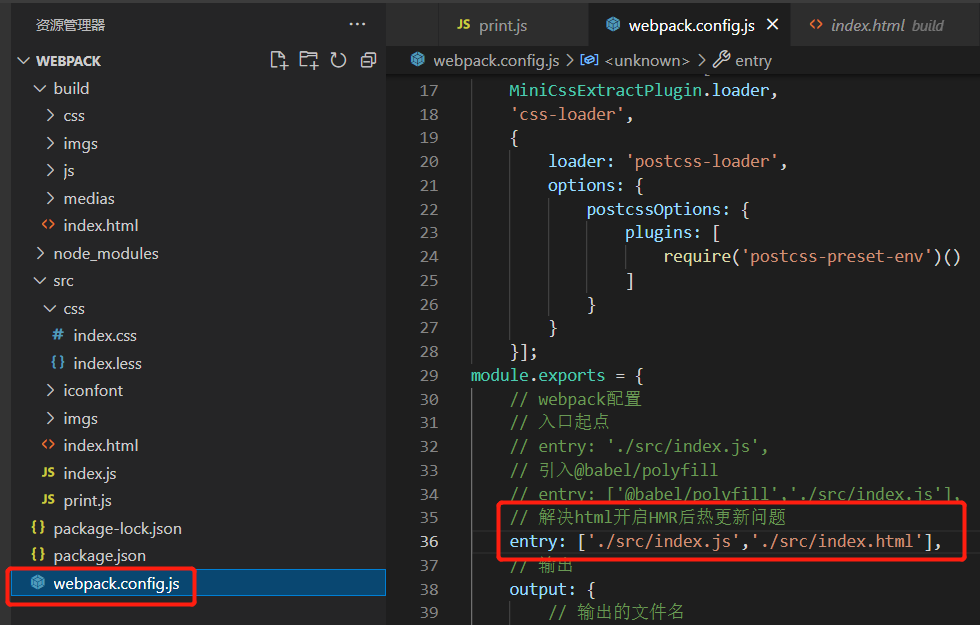


·html文件

- 默认没有使用HMR功能,只要一个html文件，也不需要使用HMR功能

- 同时会导致问题：html文件不会自动更新

- 解决：修改entry入口，将html文件引入，但还是，没有使用HMR功能

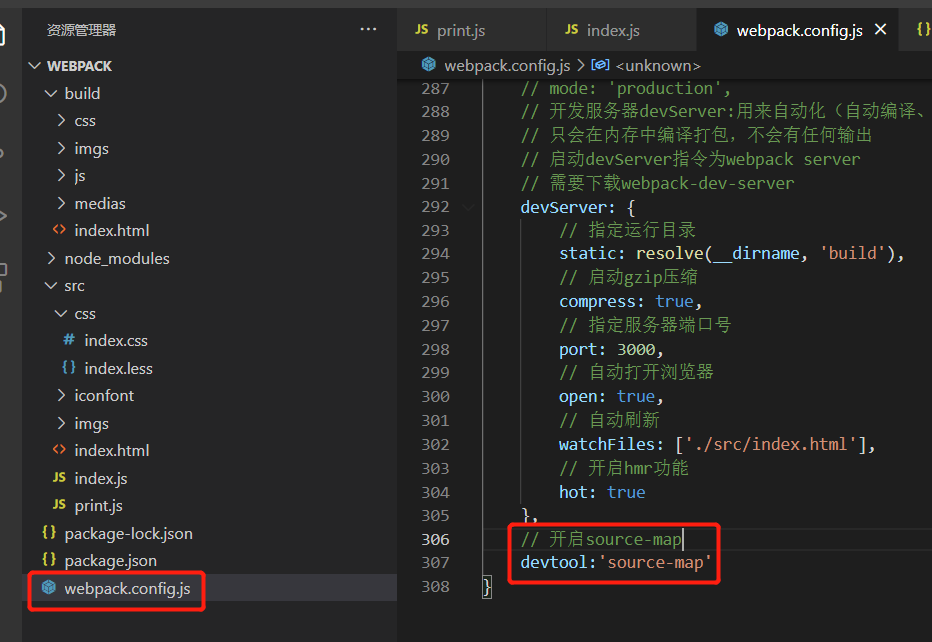


## source-map

·一种提供源代码到构建后代码映射技术

- 如果构建后代码出错，通过映射可以追踪源代码错误

·开启source-map



    // 开启source-map

    // 默认外部

    // 有错误代码准确信息和源代码的错误位置

    devtool:'source-map'

    // 内联(只生成一个内联source-map)

    // 有错误代码准确信息和源代码的错误位置

    // devtool:'inline-source-map'

    // 外部

    // 有错误代码的错误原因，没有源代码的错误位置，只能提示到构建后代码的错误位置

    // devtool:'hidden-source-map'

    // 内联(每一个文件都生成对应的source-map，都在eval)

    // 有错误代码准确信息和源代码的错误位置

    // devtool:'eval-source-map'

    // 外部

    // 有错误代码准确信息，但是没有任何源代码信息

    // devtool:'nosources-source-map'

    // 外部

    // 有错误代码准确信息和源代码的错误位置，但只能精确到行

    // devtool:'cheap-source-map'

    // 外部

    // 有错误代码准确信息和源代码的错误位置

    // module会将loader的sourmap加入

    // devtool:'cheap-module-source-map'

    /\*

        内联和外部的区别

            外部生成了文件，内联没有

            内联构建速度更快，内联会让代码体积变大，所以在生产环境不用内联

    \*/

    // 开发环境：速度快，调试更友好

        // 速度：eval>inline>cheap

            // eval-cheap-source-map

            // 其次eval-source-map

        // 调试友好

            // source-map

            // cheap-module-source-map

            // cheap-source-map

        // 用eval-source-map / eval-cheap-source-map

    // 生产环境：源代码要不要隐藏？调试要不要友好

        // 要不要隐藏

            // nosources-source-map 全部隐藏

            // hidden-source-map 只隐藏源代码

        // 调试要不要友好

            // source-map

        // 使用source-map / cheap-module-source-map

}

## oneOf

·配置webpack

// webpack配置文件

// 当修改了该配置文件一定要重启服务，才能生效

// resolve用来拼接绝对路径

const { resolve } = require('path');

// 引入html-webpack-plugin插件

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

// 引入mini-css-extract-plugin插件

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

// 设置nodejs环境变量

process.env.NODE\_ENV = 'development';

// 引入optimize-css-assets-webpack-plugin

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

// 引入eslint-webpack-plugin

const ESLintPlugin = require('eslint-webpack-plugin');

// 引入webpack，以便添加HMR插件

// const webpack = require('webpack');

// 复用loader

const commonCSSLoader = [

    MiniCssExtractPlugin.loader,

    'css-loader',

    {

        loader: 'postcss-loader',

        options: {

            postcssOptions: {

                plugins: [

                    require('postcss-preset-env')()

                ]

            }

        }

    }];

module.exports = {

    // webpack配置

    // 入口起点

    // entry: './src/index.js',

    // 引入@babel/polyfill

    // entry: ['@babel/polyfill','./src/index.js'],

    // 解决html开启HMR后热更新问题

    entry: ['./src/index.js', './src/index.html'],

    // 输出

    output: {

        // 输出的文件名

        filename: 'js/main.js',

        // 输出路径

        // \_\_dirname是nodejs变量，代表当前文件目录绝对路径

        path: resolve(\_\_dirname, 'build')

    },

    // loder的配置

    module: {

        rules: [

            {

                // 以下loader只会匹配一个

                // 注意：不能有两个配置处理同一种类型文件

                oneOf: [

                    // 详细loader配置

                    // 不同的文件必须配置不同的loader处理

                    // 处理css文件

                    {

                        // 匹配哪些文件

                        // 匹配.css结尾的文件

                        test: /\.css$/,

                        // 加入该代码后会优先执行该loader

                        // enclude:'pre',

                        // 使用哪些loader进行处理

                        // use数组中loader执行顺序:从右到左\从下到上

                        use: [

                            // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                            // 'style-loader',

                            // 使用MiniCssExtractPlugin插件的loader来处理

                            // MiniCssExtractPlugin.loader,

                            // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                            // 'css-loader',

                            // css兼容性处理：postcss --> postcss-loader postcss-preset-env

                            // 帮postcss找到package.json中browserslist里面的配置。通过配置加载指定的css兼容性样式

                            /\*

                            "browserslist":{

                                // 开发环境 --> 设置node环境变量:process.env.NODE\_ENV = development

                                "development":[

                                  "last 1 chrome version",

                                  "last 1 firefox version",

                                  "last 1 safari version"

                                ],

                                生产环境：默认

                                "production":[

                                  ">0.2%",

                                  "not dead",

                                  "not op\_mini all"

                                ]

                              }

                            \*/

                            // 需要下载postcss-loader postcss-preset-env

                            // 使用loader的默认配置

                            // 'postcss-loader',

                            // 修改loader的配置，这里修改会出问题，要新建postcss.config.js文件修改才行

                            // {

                            //     loader: 'postcss-loader',

                            //     options: {

                            //         postcssOptions: {

                            //             plugins: [

                            //                 require('postcss-preset-env')()

                            //             ]

                            //         }

                            //     }

                            // }

                            // 直接使用复用cssloader

                            ...commonCSSLoader

                        ]

                    },

                    // 处理less文件

                    {

                        // 匹配哪些文件

                        // 匹配.less结尾的文件

                        test: /\.less$/,

                        // 使用哪些loader进行处理

                        // use数组中loader执行顺序:从右到左\从下到上

                        use: [

                            // 这个一定要在less-loader前面

                            ...commonCSSLoader,

                            // 创建style标签,将js中的样式资源插入进行,添加到head中生效

                            // 'style-loader',

                            // 使用MiniCssExtractPlugin插件的loader来处理

                            // MiniCssExtractPlugin.loader,

                            // 将css文件变成commonjs模块加载js中,里面内容是样式字符串

                            // 'css-loader',

                            // 将less文件编译成css文件

                            'less-loader'

                            // {

                            //     loader: 'postcss-loader',

                            //     options: {

                            //         postcssOptions: {

                            //             plugins: [

                            //                 require('postcss-preset-env')()

                            //             ]

                            //         }

                            //     }

                            // }

                            // 直接使用复用cssloader

                        ]

                    },

                    {

                        // 处理图片资源

                        // 处理不了html中引入的img图片

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

                        // 使用一个loader时

                        // 要下载 url-loader和file-loader

                        loader: 'url-loader',

                        // 配置url

                        options: {

                            // 图片大小小于8kb时,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,

                            // 解决图片不显示问题

                            // 问题:url-loader默认使用es6模块化解析,而htnl-loader引入图片是commonjs

                            // 解析时会出问题:[object Module]

                            // 解决:关闭url-loader的es6模块化,使用commonjs解析

                            esModule: false,

                            // 给图片重命名

                            // [hash:10]取图片的hash的前10位

                            // [ext]取文件原来扩展名

                            name: '[hash:10].[ext]',

                            // 将图片资源输出到imgs目录

                            outputPath: 'imgs'

                        },

                        // 解决图片不显示问题

                        type: 'javascript/auto'

                    },

                    // 处理html中引入的图

                    {

                        test: /\.html$/,

                        // 负责引入img,从而能被url-loader打包处理

                        loader: 'html-loader'

                    },

                    // 打包其他资源(除了html/js/css以外的资源)

                    {

                        // 排除资源

                        exclude: /\.(css|js|html|jpg|png|gif|less)$/,

                        loader: 'file-loader',

                        // 配置文件

                        options: {

                            // 解决字体图标不显示问题

                            // 解析时会出问题:[object Module]

                            // 解决:关闭es6模块化,使用commonjs解析

                            esModule: false,

                            // 给文件重命名

                            // [hash:10]取图片的hash的前10位

                            // [ext]取文件原来扩展名

                            name: '[hash:10].[ext]',

                            // 将图片资源输出到medias目录

                            outputPath: 'medias'

                        },

                        // 解决字体图标不显示问题

                        type: 'javascript/auto'

                    },

                    // js兼容性处理

                    // 需要下载babel-loader @babel/preset-env @babel/core

                    // 问题:只能转换基本语法,promise等语法不能转换

                    // 需要使用@babel/polyfill解决,会将所有兼容性代码全部引入,体积大

                    // 按需加载兼容性代码-->core-js

                    {

                        // 指定处理的文件

                        test: /\.m?js$/,

                        // 排除第三方库

                        exclude: /(node\_modules|bower\_components)/,

                        use: {

                            loader: 'babel-loader',

                            options: {

                                presets: [

                                    [

                                        '@babel/preset-env',

                                        {

                                            // useBuiltIns: 'entry',

                                            // 按需加载

                                            useBuiltIns: 'usage',

                                            // 指定core-js版本

                                            corejs: {

                                                version: 3

                                            },

                                            // 指定兼容性做到哪个版本浏览器

                                            targets: {

                                                chrome: '60',

                                                firefox: "60",

                                                ie: '9',

                                                safari: '10',

                                                edge: '17'

                                            }

                                        }

                                    ]

                                ],

                                //利用 @babel/plugin-transform-runtime 插件还能以沙箱垫片的方式防止污染全局， 并抽离公共的 helper function , 以节省代码的冗余

                                "plugins": ["@babel/plugin-transform-runtime"]

                            }

                        }

                    }

                ]

            }

        ]

    },

    // plugins的配置

    plugins: [

        // 详细的plugins配置

        // 配置html-webpack-plugin插件

        // 会默认创建一个空的Html文件，并引入打包输出的所有资源（js/css）

        // new HtmlWebpackPlugin()

        new HtmlWebpackPlugin({

            //生成指定的html文件,并引入打包输出的所有资源（js/css）

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

            // 压缩html代码

            minify: {

                // 移出空格

                collapseWhitespace: true,

                // 移出注释

                removeComments: true

            }

        }),

        // 提取css插件

        new MiniCssExtractPlugin({

            // 指定输出路径和文件名

            filename: 'css/main.css'

        }),

        // 压缩css插件

        new OptimizeCssAssetsPlugin(),

        // 语法检查

        // 注意：只检查自己写的源代码，第三方的库是不用检查的

        /\*

        // 设置检查规则：

            // package.json中eslintConfig中设置

                "eslintConfig": {

                    "extends": "airbnb-base",

                    "rules": {

                        "no-console": "off"

                    }

                },

        \*/

        new ESLintPlugin({

            // undefined,

            // 检查js语法

            extensions: ['js'],

            // context: resolve('src'),

            // 排除第三方库

            exclude: '/node\_modules',

            // 自动修复不规格语法

            fix: true

        }),

        // new webpack.HotModuleReplacementPlugin(),

        // new webpack.NamedModulesPlugin(),

        // new webpack.HotModuleReplacementPlugin()

    ],

    // 模式,指定development或production模式

    mode: 'development',

    // 生产环境下会自动压缩js代码

    // mode: 'production',

    // 开发服务器devServer:用来自动化（自动编译、自动打开浏览器、自动刷新浏览器~）

    // 只会在内存中编译打包，不会有任何输出

    // 启动devServer指令为webpack server

    // 需要下载webpack-dev-server

    devServer: {

        // 指定运行目录

        static: resolve(\_\_dirname, 'build'),

        // 启动gzip压缩

        compress: true,

        // 指定服务器端口号

        port: 3000,

        // 自动打开浏览器

        open: true,

        // 自动刷新

        watchFiles: ['./src/index.html'],

        // 开启hmr功能

        hot: true

    },

    // 开启source-map

    // 默认外部

    // 有错误代码准确信息和源代码的错误位置

    devtool: 'source-map'

    // 内联(只生成一个内联source-map)

    // 有错误代码准确信息和源代码的错误位置

    // devtool:'inline-source-map'

    // 外部

    // 有错误代码的错误原因，没有源代码的错误位置，只能提示到构建后代码的错误位置

    // devtool:'hidden-source-map'

    // 内联(每一个文件都生成对应的source-map，都在eval)

    // 有错误代码准确信息和源代码的错误位置

    // devtool:'eval-source-map'

    // 外部

    // 有错误代码准确信息，但是没有任何源代码信息

    // devtool:'nosources-source-map'

    // 外部

    // 有错误代码准确信息和源代码的错误位置，但只能精确到行

    // devtool:'cheap-source-map'

    // 外部

    // 有错误代码准确信息和源代码的错误位置

    // module会将loader的sourmap加入

    // devtool:'cheap-module-source-map'

    /\*

        内联和外部的区别

            外部生成了文件，内联没有

            内联构建速度更快，内联会让代码体积变大，所以在生产环境不用内联

    \*/

    // 开发环境：速度快，调试更友好

    // 速度：eval>inline>cheap

    // eval-cheap-source-map

    // 其次eval-source-map

    // 调试友好

    // source-map

    // cheap-module-source-map

    // cheap-source-map

    // 用eval-source-map / eval-cheap-source-map

    // 生产环境：源代码要不要隐藏？调试要不要友好

    // 要不要隐藏

    // nosources-source-map 全部隐藏

    // hidden-source-map 只隐藏源代码

    // 调试要不要友好

    // source-map

    // 使用source-map / cheap-module-source-map

}

## 缓存

·babel缓存

- 加快打包构建速度

- cacheDirectory:true

·文件资源缓存

- 避免强缓存

- hash：每次webpack构建时会生成一个唯一的hash值。

- filename: 'js/main.[hash:10].js'

- filename: 'css/main.[hash:10].css'

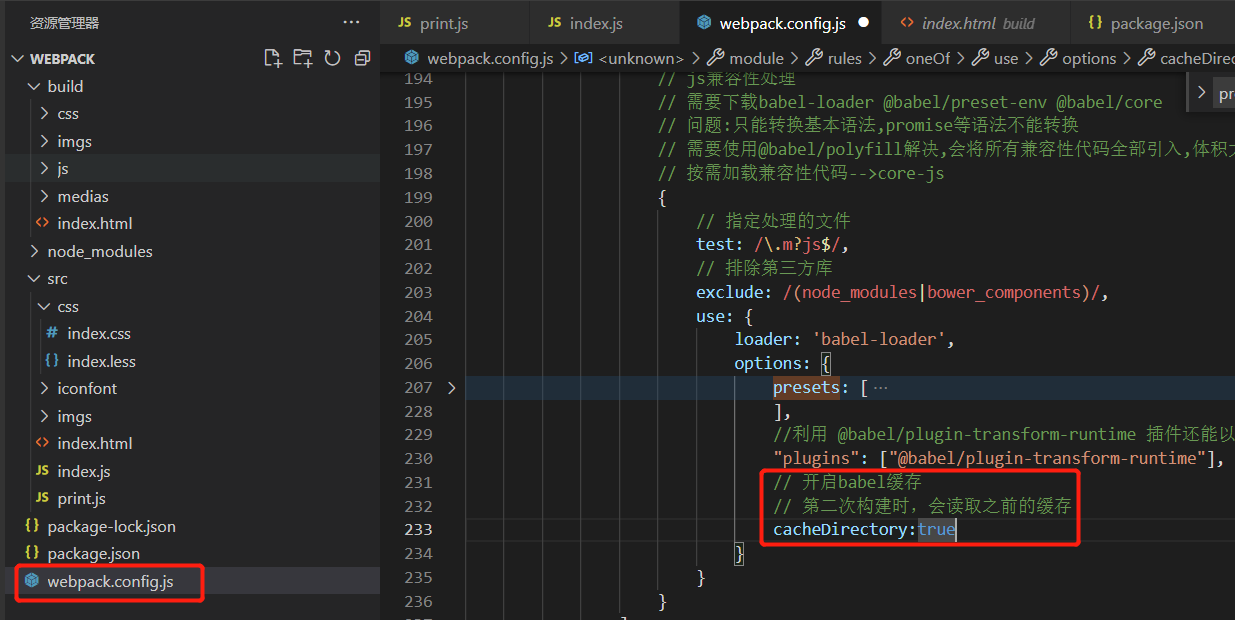
- 问题：因为js和css同时使用一个hash值。如果重新打包，会导致所有缓存失效。

- chunkhash: 根据chunk生成hash值。如果打包来源于同一个chunk，那么hash值一样

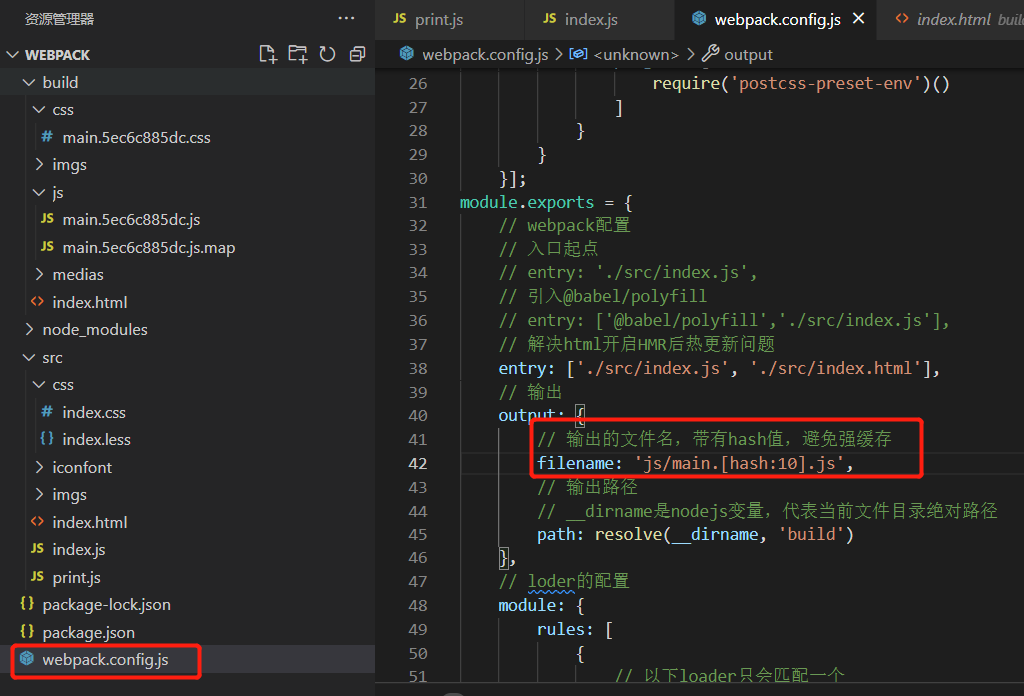
- 问题：js和css还是一样的，因为css是在js中被引入的，所以同属于一个chunk

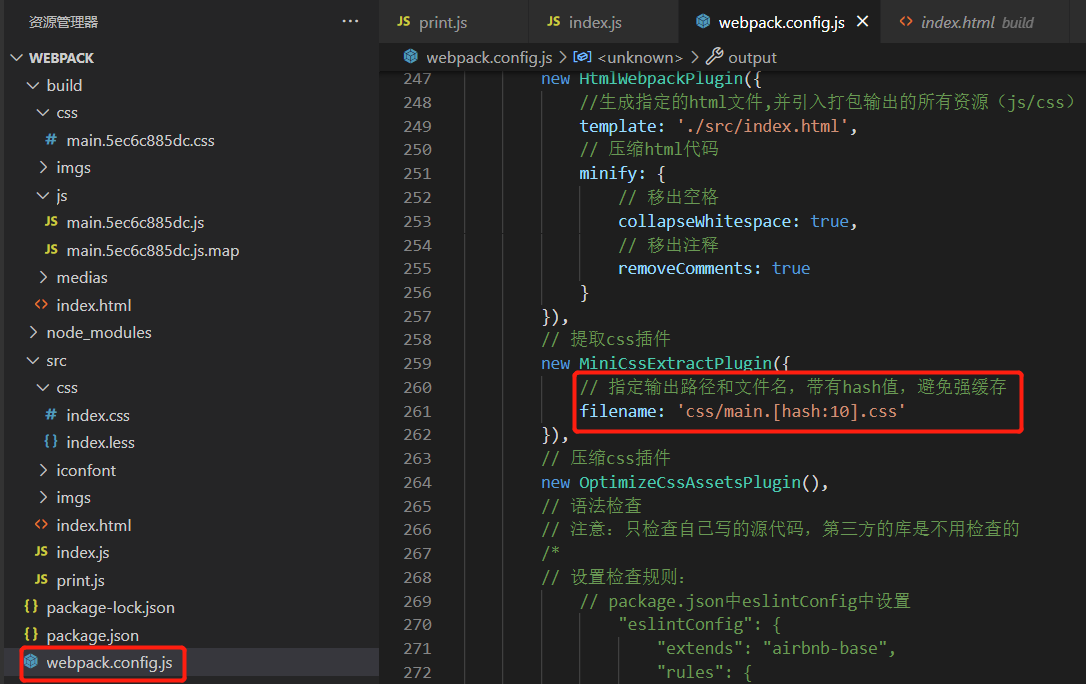
- contenthash：根据文件的内容生产hash值。不同的文件hash不同

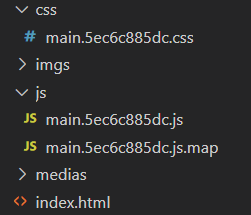
·开启babel缓存



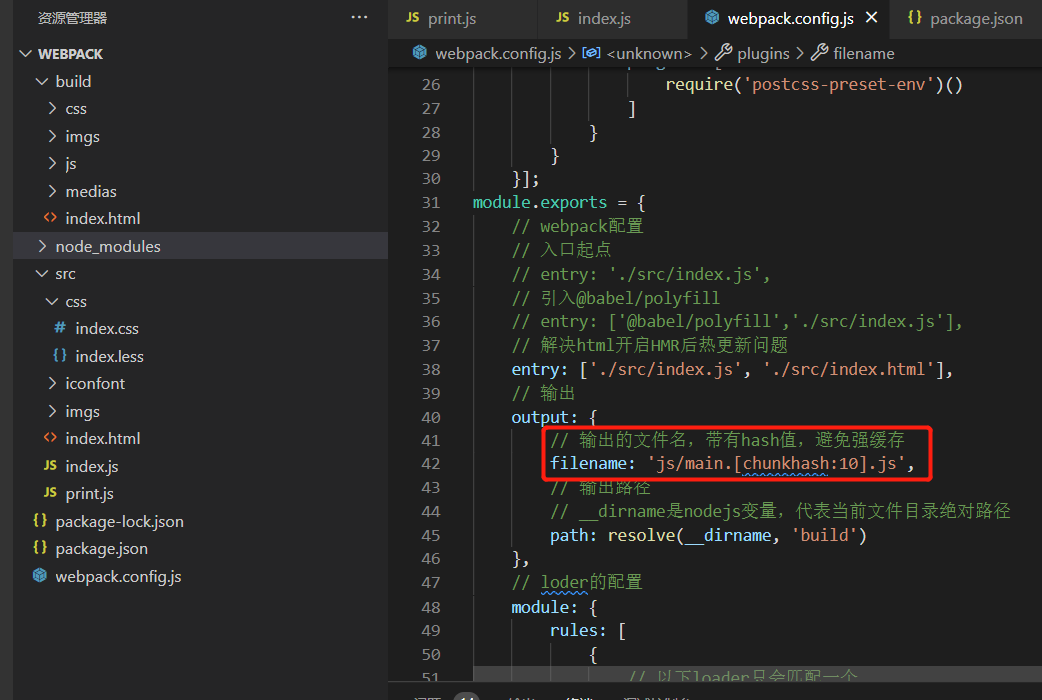
·使用hash避免强缓存

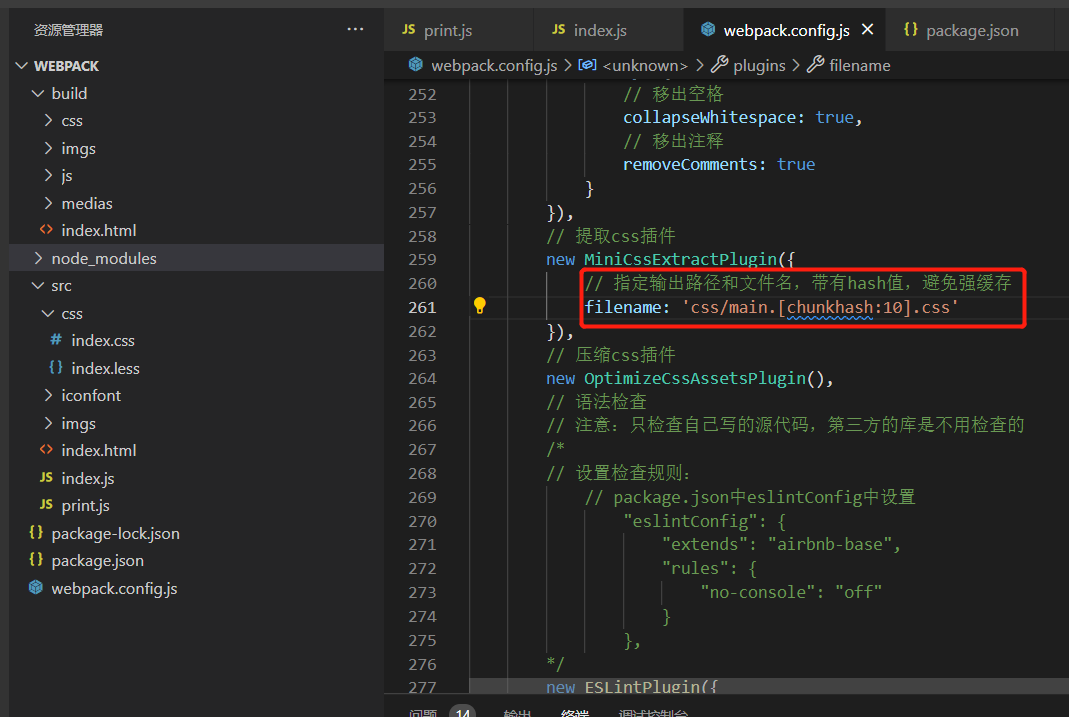


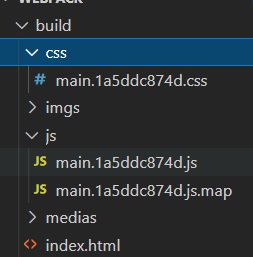




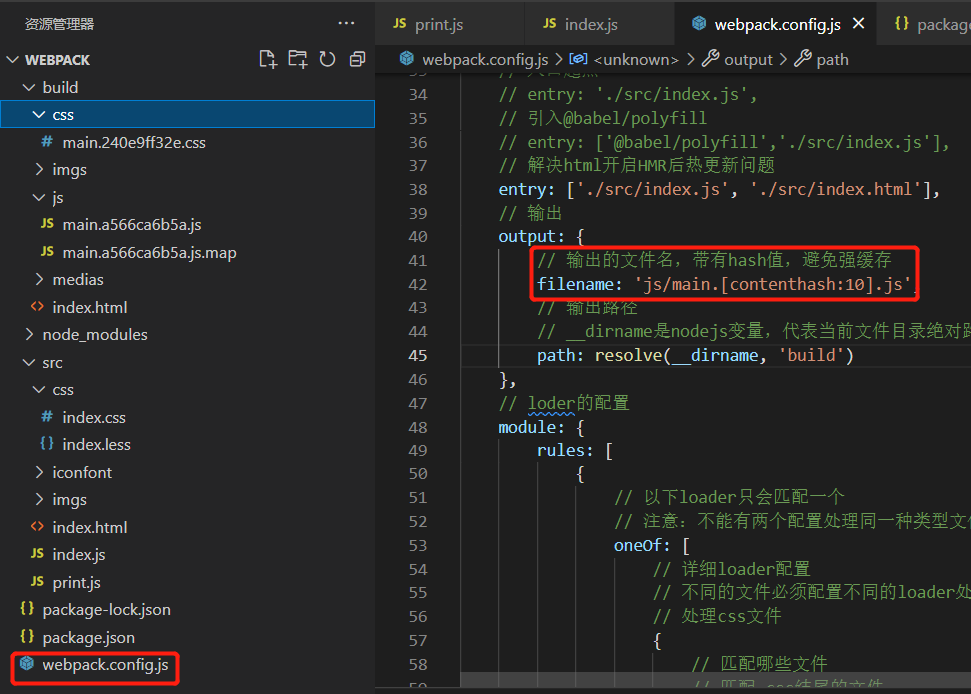
·使用chunkhash避免强缓存

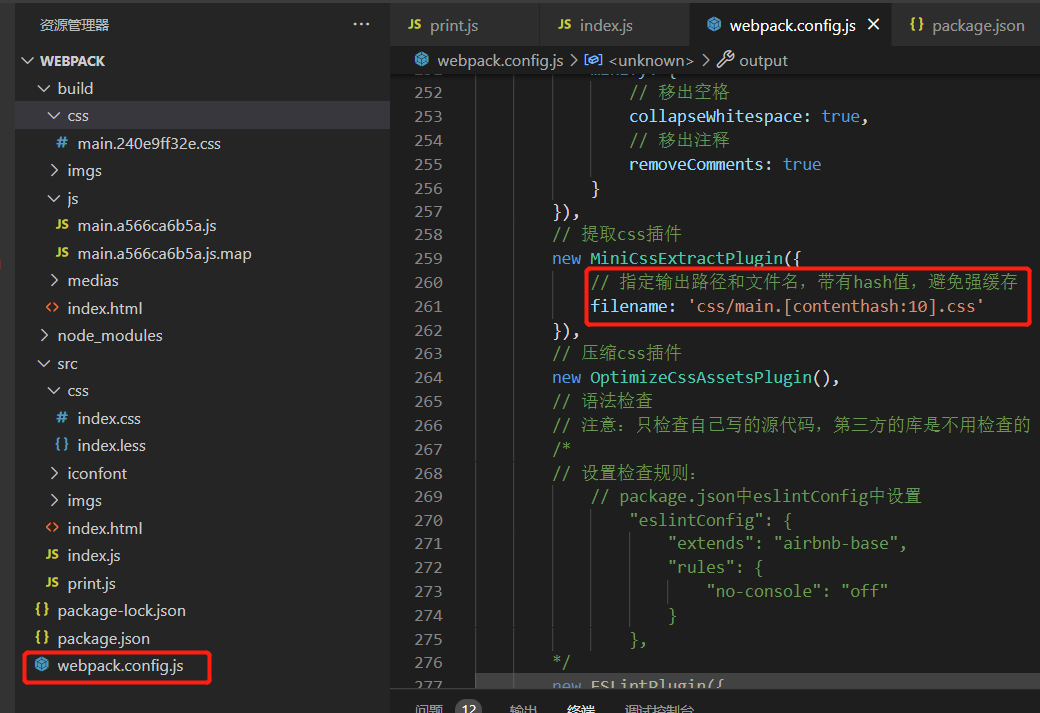


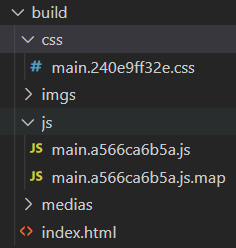




·contenthash避免强缓存







## tree shaking

·去除无用代码，减少代码体积

·前提

- 必须使用es6模块化

- 开启production环境

·在package.json中配置

- "sideEffects":false

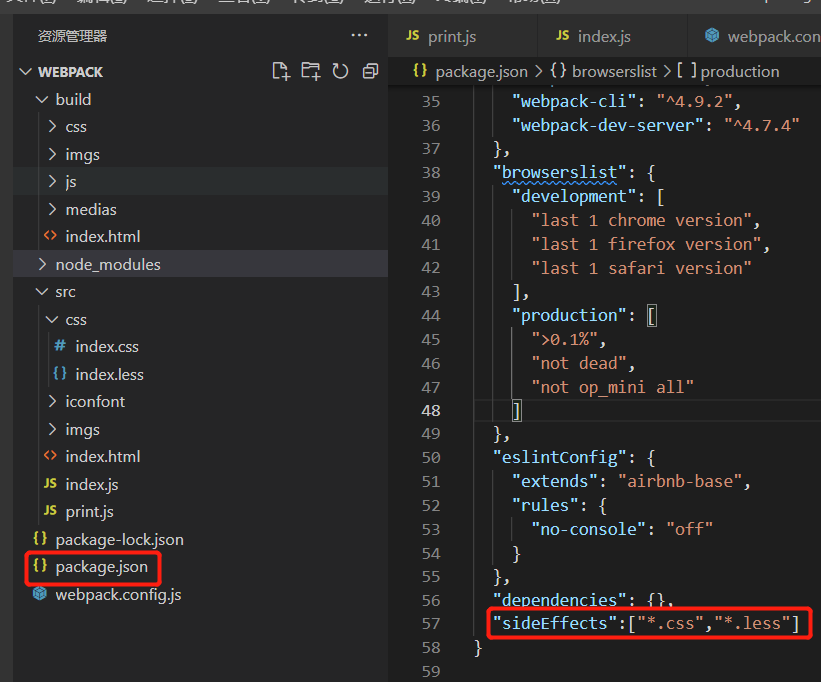
- 所有代码都没有副作用（都可以进行tree shaking）

- 问题：可能会把css / @babel/polyfill文件干掉

- 解决

- "sideEffects":["\*.css","\*.less"]

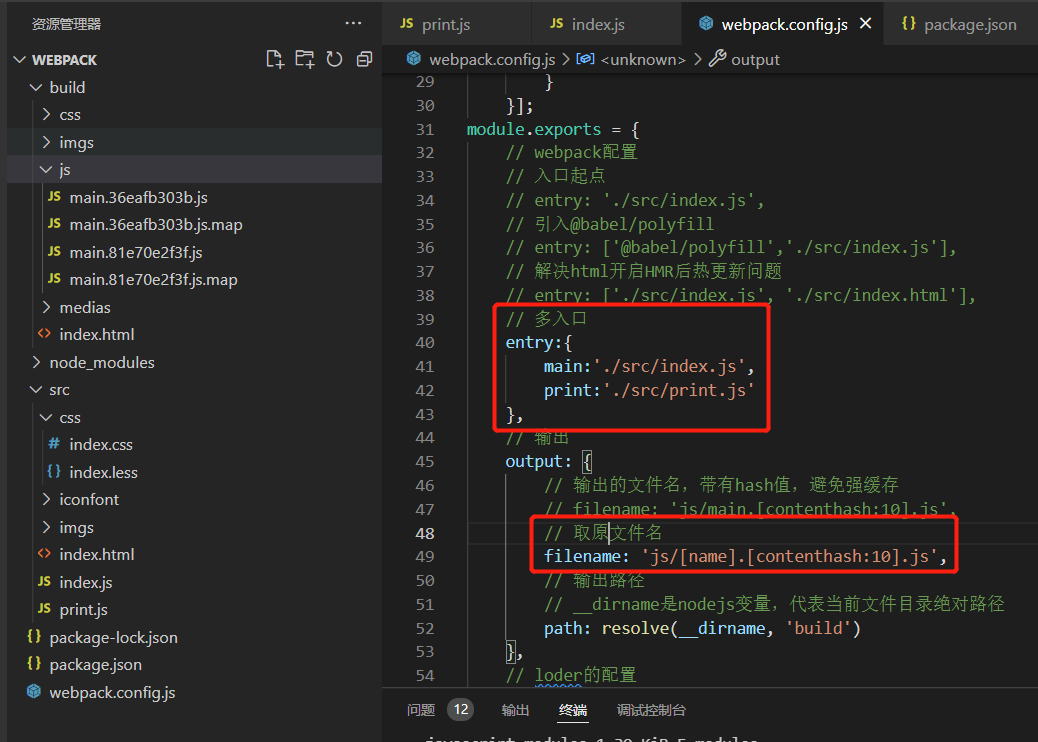
- 不处理css资源

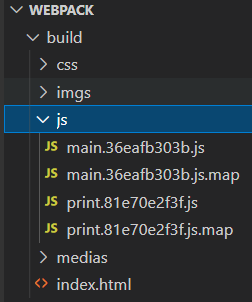


## code split

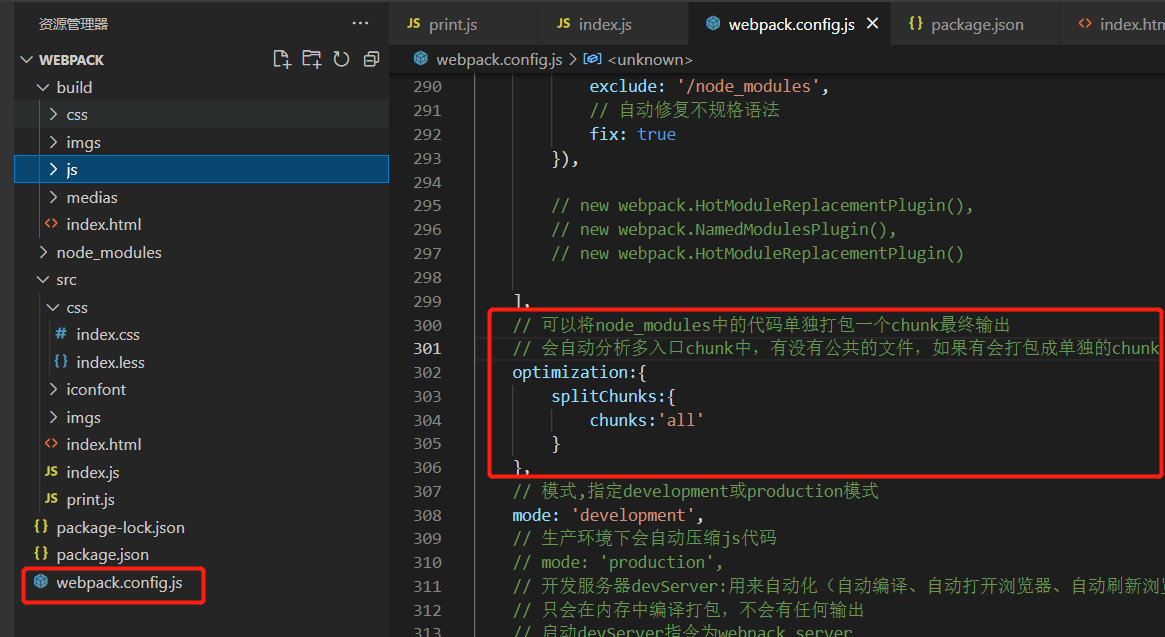
·打包生成多个js文件

### 方式一





## 方式二



方式三

·要关闭语法检查

# Webpack原理分析

# Webpack5介绍