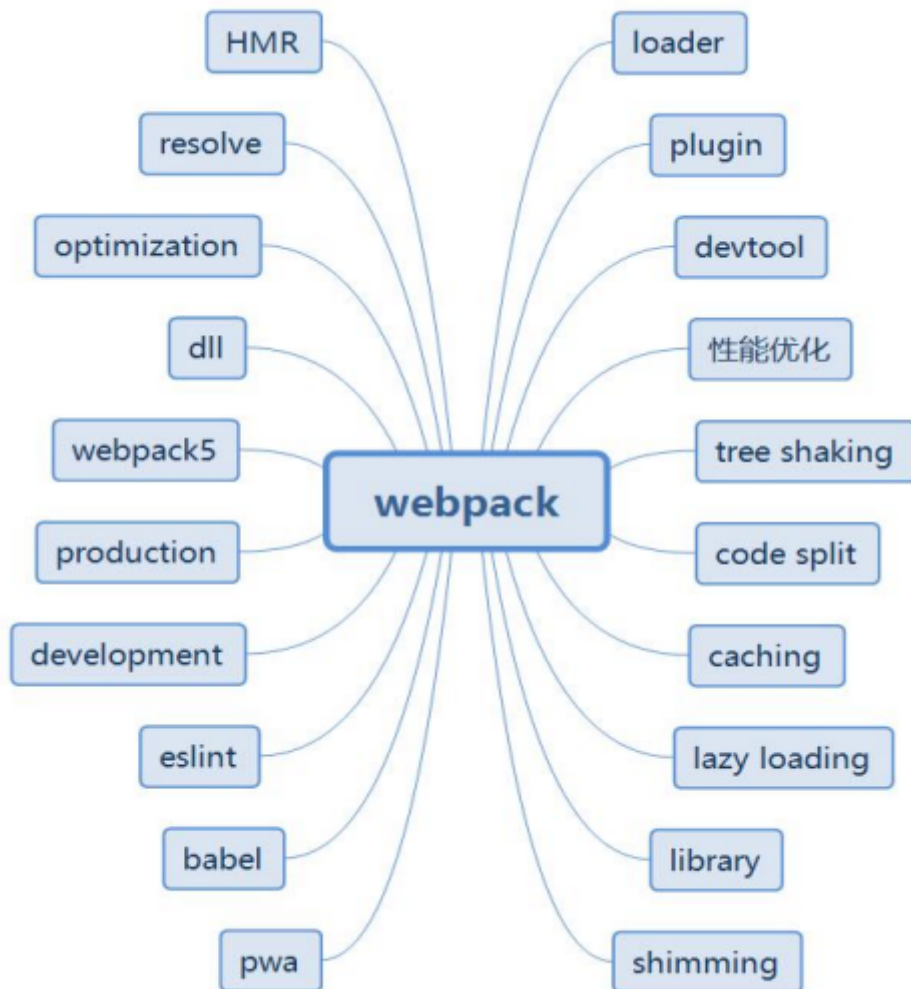


webpack 学习笔记

1. 第一章：webpack 简介

1.1. webpack 知识点

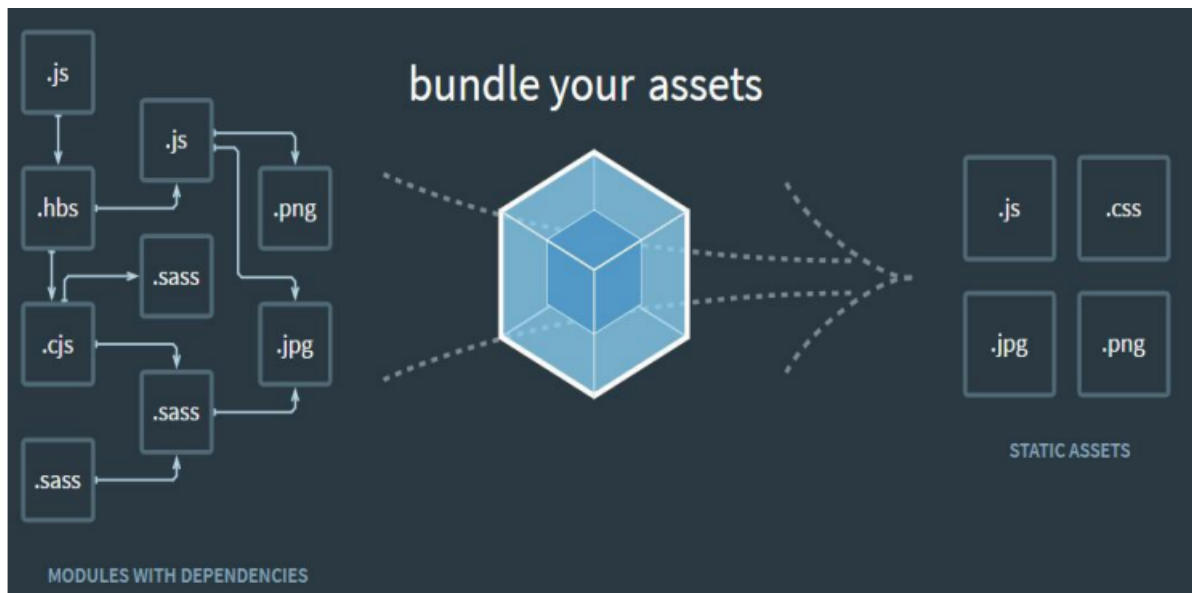


1.2. webpack 是什么

webpack 是一种前端资源构建工具，一个静态模块打包器(module bundler)。

在 webpack 看来，前端的所有资源文件(js/json/css/img/less/...) 都会作为模块处理。

它将根据模块的依赖关系进行静态分析，形成一个 chunk(块)，再打包(对各种静态资源进行处理) 生成对应的静态资源(bundle)。



1.3. webpack 五个核心概念

1.3.1. Entry

- 入口(Entry)指示 webpack 以哪个文件为入口七点开始打包，分析构建内部依赖图。

1.3.2. Output

- 输出(Output)指示 webpack 打包后的资源 bundles 输出到哪里去，以及如何命名。

1.3.3. Loader

- Loader 让 webpack 能够去处理那些非 JavaScript 文件(webpack自身只理解 JavaScript)。

1.3.4. Plugins

- 插件(Plugins)可以用于执行范围更广的任务。插件的范围包括，从打包优化到压缩，一直到重新定义环境的变量等。

1.3.5. Mode

- 模式(Mode)指示 webpack 使用相应模式的配置

选项	描述	特点
development	会将 DefinePlugin 中 process.env.NODE_ENV 的值设置为 development。启用 NamedChunksPlugin 和 NamedModulesPlugin。	能让代码本地调试运行的环境
production	会将 DefinePlugin 中 process.env.NODE_ENV 的值设置为 production。启用 FlagDependencyUsagePlugin, FlagIncludedChunksPlugin, ModuleConcatenationPlugin, NoEmitOnErrorsPlugin, OccurrenceOrderPlugin, SideEffectsFlagPlugin 和 TerserPlugin。	能让代码优化上线运行的环境

2. 第二章：webpack 的初体验

2.1. 初始化配置

1. 初始化package.json

```
1 | npm init
```

2. 下载并安装 webpack

```
1 | ## 先全局安装
2 | npm install webpack webpack-cli -g
3 | ## 再本地安装，-D表示添加到开发依赖
4 | npm install webpack webpack-cli -D
```

2.2. 编译打包应用

1. 创建文件

2. 运行指令

- 开发环境指令：`webpack --entry ./src/js/index.js -o build/js/built.js --mode=development`
 - 说明：webpack 会以 ./src/js/index.js 为入口文件开始打包，打包后输出到 ./build/built.js，整体打包环境是：开发环境(--mode=development)。
 - 功能：webpack 能够编译打包 js 和 json 文件，并且能够将 es6 的模块化语法转换成浏览器能识别的语法。
- 生产环境指令：`webpack --entry src/js/index.js -o build/js/built.js --mode=production`

- 说明：webpack 会以 ./src/js/index.js 为入口文件开始打包，打包后输出到 ./build/built.js，整体打包环境是：生产环境(--mode=production)。
- 功能：在开发配置功能上多一个功能，压缩代码。

3. 结论

- webpack 能够编译 js 和 json 文件，不能处理 css/img 等其他资源。
- 生产环境和开发环境都能够将 es6 模块语法转换成浏览器能识别的语法。
- 生产环境比开发环境多一个压缩代码功能。

4. 问题：

- 不能编译打包 css、img 等文件。
- 不能将 js 的 es6 基本语法转化为 es5 以下语法。

3. 第三章：webpack 开发环境的基本配置

3.1. 创建配置文件

1. 创建文件 webapck.config.js（webpack的配置文件）

- 作用：只是 webpack 干哪些活（当你运行 webpack 指令时，会加载里面的配置
- 所有的构建工具都是基于 nodejs 平台运行的~模块化默认采用 commonjs。

2. 配置内容如下：

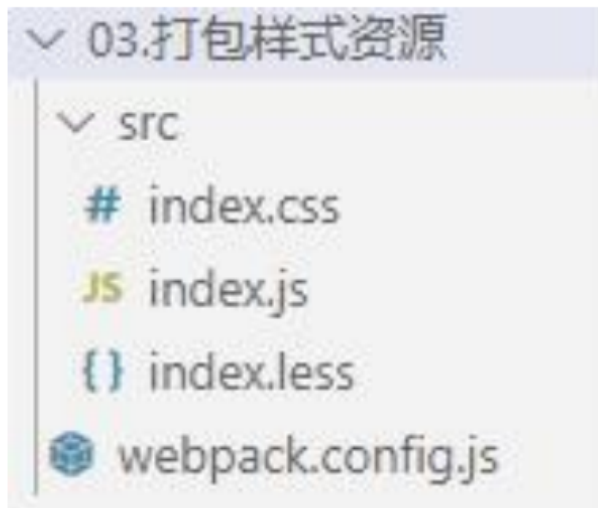
```
1  const { resolve } = require('path') // node 内置核心模块，用来处理路径问题。
2
3  module.exports = {
4    entry: './src/js/index.js', // 入口文件
5    output: { // 输出配置
6      filename: './built.js', // 输出文件名
7      path: resolve(__dirname, 'build/js') // 输出文件路径配置
8    },
9    mode: 'development' // 开发环境
10 }
```

3. 运行指令：webpack

4. 结论：此时功能与上节一致

3.2. 打包样式资源

1. 创建文件



2. 下载安装 loader 包

```
1 | npm i css-loader style-loader less-loader less -D
```

3. 修改配置文件

```
1  // resolve 是用来拼接绝对路径的方法
2  const { resolve } = require('path')
3
4  module.exports = {
5    // webpack 配置
6    // 入口起点
7    entry: './src/index.js',
8    // 输出
9    output: {
10     // 输出文件名
11     filename: 'built.js',
12     // 输出路径
13     // __dirname nodejs的变量，代表当前文件的目录绝对路径
14     path: resolve(__dirname, 'build'),
15   },
16   // loader 的配置
17   module: {
18     rules: [
19       // 详细 loader 配置
20       // 不同文件必须配置不同的 loader 处理
21       {
22         // 匹配哪些文件
23         test: /\.css$/,
24         // 使用哪些 loader 进行处理
25         use: [
26           // use 数组中 loader 执行顺序：从右到左，从下到上 依次执行
27           // 创建 style 标签，将 js 中的样式资源进行插入，添加到 head 中生效
28           'style-loader',
29           // 将css文件变成commonjs模块加载到js中，里面内容是样式字符串
30           'css-loader'
31         ]
32       },
33       {
34         test: /\.less$/,
```

```

35     use: [
36         'style-loader',
37         'css-loader',
38         // 将 less 文件编译成 css 文件
39         // 需要下载 less-loader 和 less
40         'less-loader'
41     ]
42 }
43 ]
44 },
45 // plugins 的配置
46 plugins: [
47     // 详细的 plugins 的配置
48 ],
49 // 模式
50 mode: 'development', // 开发模式
51 // mode: 'production' // 生产模式
52 }

```

4. 运行指令: `webpack`

3.3. 打包 HTML 资源

1. 创建文件



2. 下载安装 plugin 包

```

1 npm install --save-dev html-webpack-plugin
2 ## npm i html-webpack-plugin -D // 简写

```

3. 修改配置文件

```

1  /*
2   loader: 1. 下载 2. 使用 (配置loader)
3   plugin: 1. 下载 2. 引入 3. 使用
4   */
5
6  const { resolve } = require('path')
7  const HtmlWebpackPlugin = require('html-webpack-plugin')
8
9  module.exports = {
10     entry: './src/index.js',
11     output: {

```

```

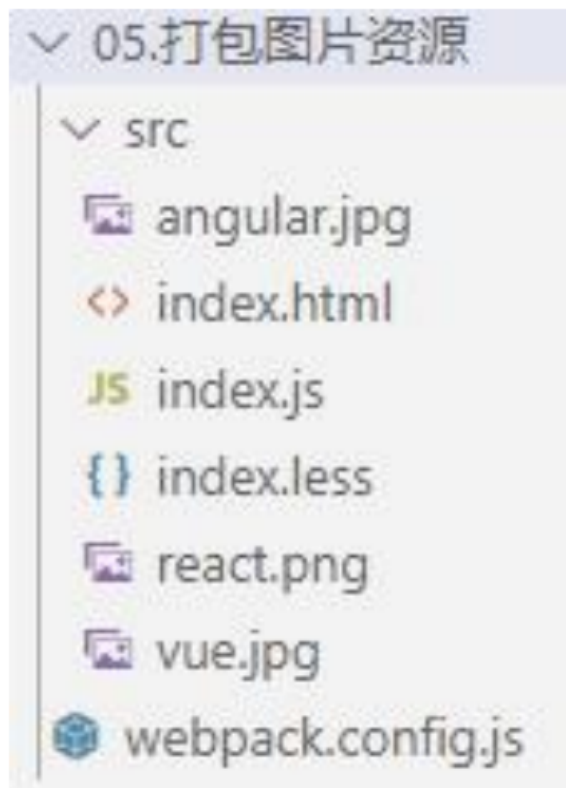
12     filename: 'built.js',
13     path: resolve(__dirname, 'build')
14   },
15   module: {
16     rules: [
17       // loader
18     ]
19   },
20   plugins: [
21     // plugins 的配置
22     // html-webpack-plugin
23     // 功能: 默认会创建一个空的 HTML, 自动引入打包输出的所有资源 (JS/CSS)
24     // 需求: 需要有结构的 HTML 文件
25     new HtmlWebpackPlugin({
26       // 复制 './src/index.html' 文件, 并自动引入打包输出的所有资源 (JS/CSS)
27       template: './src/index.html'
28     })
29   ],
30   mode: 'development'
31 }

```

4. 运行指令: `webpack`

3.4. 打包图片资源

1. 创建文件



2. 下载安装 loader 包

```

1 npm install --save-dev html-loader url-loader file-loader
2 ## --save-dev 简写是 -D

```

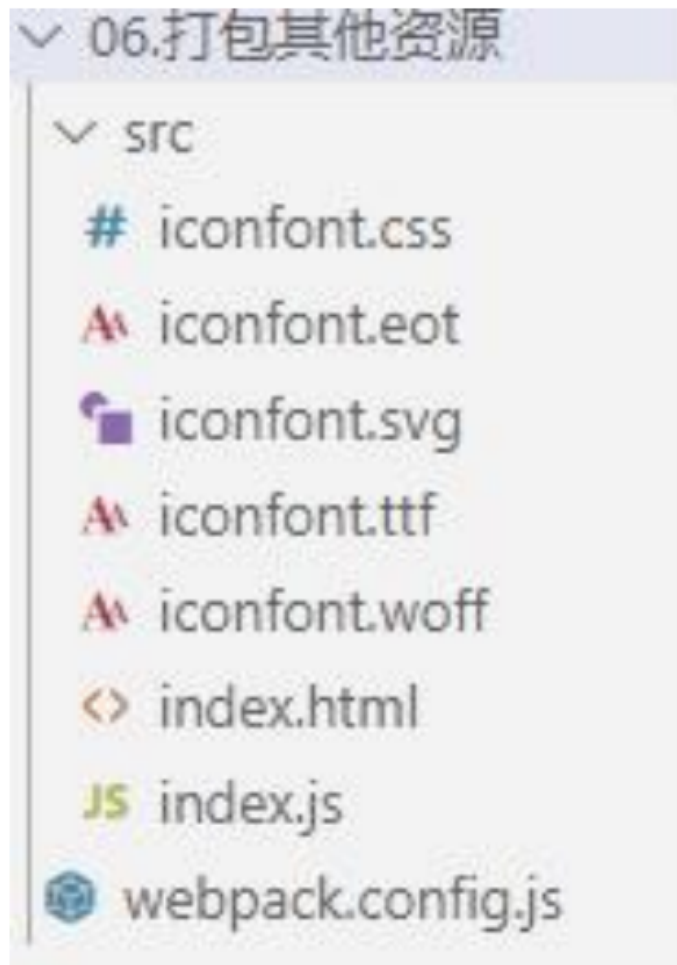
3. 修改配置文件

```
1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  module.exports = {
5    entry: './src/index.js',
6    output: {
7      filename: 'built.js',
8      path: resolve(__dirname, 'build')
9    },
10   module: {
11     rules: [
12       {
13         test: /\.less$/,
14         // 要使用多个 loader 处理用 use, 单个直接用 loader
15         use: ['style-loader', 'css-loader', 'less-loader']
16       },
17       {
18         // 问题: 默认处理不了 html 中的 img 图片
19         // 处理图片资源
20         test: /\. (jpg|png|gif)$/ ,
21         type: 'asset',
22         generator: {
23           // 给图片进行重命名
24           // [hash:10]取图片的 hash 的前 10 位
25           // [ext]取文件原来的扩展名
26           filename: 'img/[name]_[contenthash:6][ext]'
27         },
28         parser: {
29           dataUrlCondition: {
30             // 图片大小小于 8kb, 就会被 base64 处理
31             // 优点: 减少请求数量 (减轻服务器压力)
32             // 缺点: 图片体积会更大 (文件请求速度更慢)
33             maxSize: 200 * 1024
34           }
35         }
36       },
37       {
38         test: /\.html$/,
39         // 处理 html 文件的 img 图片 (负责引入 img, 从而能被 url-loader 进行处理)
40         loader: 'html-loader'
41       }
42     ]
43   },
44   plugins: [
45     new HtmlWebpackPlugin({
46       template: './src/index.html'
47     })
48   ],
49   mode: 'development'
50 }
```

4. 运行指令: webpack

3.5. 打包其他资源

1. 创建文件



2. 修改配置文件

```
1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  module.exports = {
5    entry: './src/index.js',
6    output: {
7      filename: 'built.js',
8      path: resolve(__dirname, 'build')
9    },
10   module: {
11     rules: [
12       /* {
13         // webpack 4, 已弃用
14         exclude: /\. (css|js|html|less) $/,
15         loader: 'file-loader',
16       }, */
17       {
18         // webpack 5 用法
19         test: /\. (eot|ttf|woff|svg) $/,
20         type: 'asset/resource',
21         generator: {
22           filename: 'font/[name]_[contenthash:6] [ext]'
23         }
24       }
25     ]
26   }
27 }
```

```

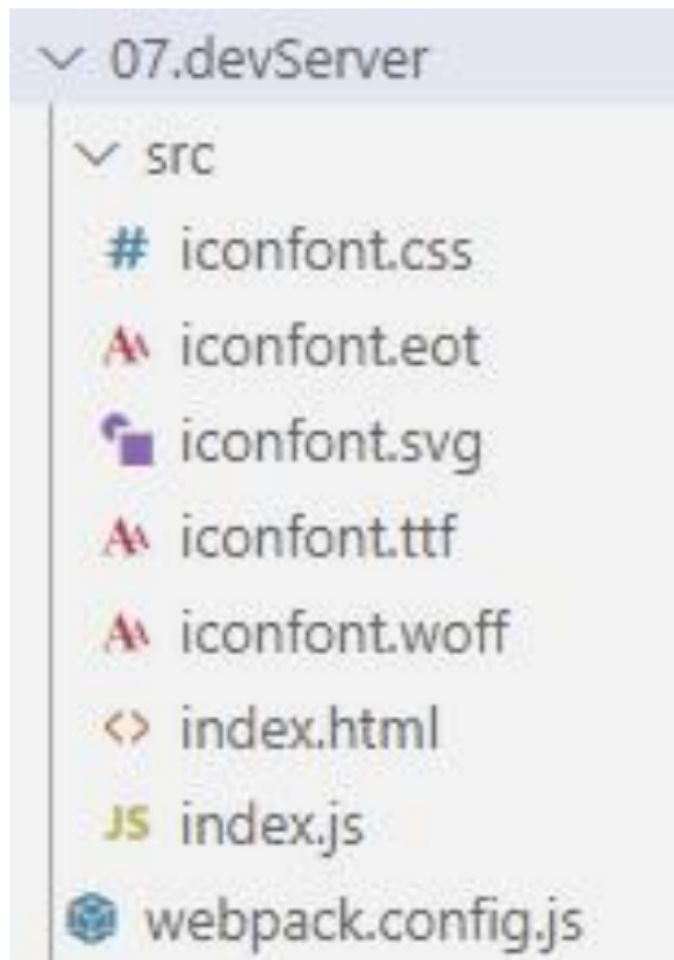
24     }
25   ]
26 },
27 plugins: [
28   new HtmlWebpackPlugin({
29     template: './src/index.html'
30   })
31 ],
32 mode: 'development'
33 }

```

3. 运行指令: `webpack`

3.6. devServer

1. 创建文件



2. 修改配置文件

```

1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  module.exports = {
5    entry: './src/index.js',
6    output: {
7      filename: 'built.js',
8      path: resolve(__dirname, 'build')
9    },

```

```

10 module: {
11   rules: [
12     {
13       test: /\.css$/,
14       use: ['style-loader', 'css-loader']
15     },
16     // 打包其他资源（除了 html/js/css 资源意外的资源）
17     {
18       // 排除 css/js/html 资源
19       exclude: /\. (css|js|html|less)$/,
20       loader: 'file-loader',
21       options: {
22         name: '[hash:10].[ext]'
23       }
24     }
25   ],
26 },
27 plugins: [
28   new HtmlWebpackPlugin({
29     template: './src/index.html'
30   })
31 ],
32 mode: 'development',
33
34 // 开发服务器 devServer（热重载）：用来自动化（自动编译，自动打开浏览器，自动刷新浏览器）
35 // 特点：只会在内存中编译打包，不会有任何输出到本地代码
36 // 启动 devServer 指令为：npx webpack serve
37 devServer: {
38   // 运行代码的目录
39   static: {
40     directory: resolve(__dirname, 'build'),
41   },
42   // 启动 gzip 压缩
43   compress: true,
44   // 端口号
45   port: 3000,
46   // 自动打开浏览器
47   open: true,
48 },
49 }

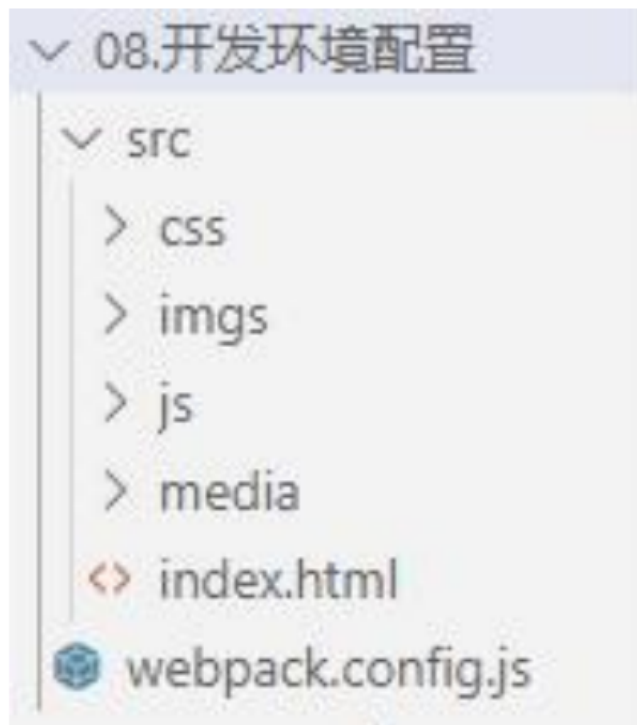
```

3. 运行指令： `npx webpack-dev-server`

3.7. 开发环境配置

- 注意：用到的 loader 包 跟 plugin 插件都要提前下载好

1. 创建文件



修改配置文件

```
1  /*
2     开发环境配置：能让代码运行
3     运行项目指令：
4     webpack 会打包结果输出出去
5     npx webpack serve 只会在内存中编译打包，没有输出
6  */
7
8  const HtmlWebpackPlugin = require('html-webpack-plugin')
9  const { resolve } = require('path')
10
11  module.exports = {
12    entry: './src/js/index.js',
13    output: {
14      filename: 'js/built.js',
15      path: resolve(__dirname, 'build')
16    },
17    module: {
18      rules: [
19        {
20          test: /\.css$/,
21          use: ['style-loader', 'css-loader']
22        },
23        {
24          test: /\.less$/,
25          use: ['style-loader', 'css-loader', 'less-loader']
26        },
27        {
28          // 处理图片资源
29          test: /\.?(jpg|png|jpeg|gif)$/i,
30          type: 'asset',
31          generator: {
32            filename: 'imgs/[name]_[contenthash:6][ext]'
33          },
34          parser: {
```

```

35     dataUrlCondition: {
36       maxSize: 8 * 1024
37     }
38   }
39 },
40 {
41   // 处理html中的 图片资源
42   test: /\.html$/,
43   loader: 'html-loader'
44 },
45 {
46   // 处理字体资源
47   test: /\.(eot|ttf|woff|svg)$/,
48   type: 'asset/resource',
49   generator: {
50     outputPath: 'media'
51   }
52 }
53 ]
54 },
55 plugins: [
56   new HtmlWebpackPlugin({
57     template: './src/index.html'
58   })
59 ],
60 mode: 'development',
61 devServer: {
62   static: {
63     directory: resolve(__dirname, 'build')
64   },
65   compress: true,
66   port: 5000,
67   open: true
68 }
69 }

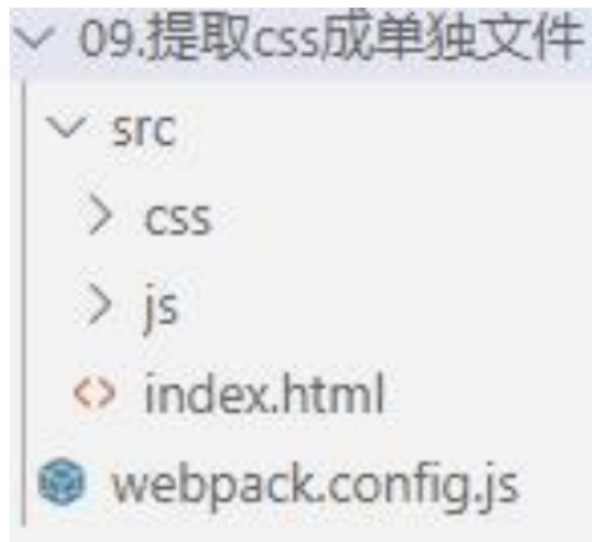
```

3. 运行指令: `npx webpack serve`

4. 第四章：webpack 生产环境的基本配置

4.1. 提取 css 成单独文件

1. 创建文件



2. 下载安装包

3. 下载插件

```
1 | npm install --save-dev mini-css-extract-plugin
```

4. 修改配置文件

```
1 | const { resolve } = require('path')
2 | const HtmlWebpackPlugin = require('html-webpack-plugin')
3 | const MiniCssExtractPlugin = require('mini-css-extract-plugin')
4 |
5 | module.exports = {
6 |   entry: './src/js/index.js',
7 |   output: {
8 |     filename: 'js/built.js',
9 |     path: resolve(__dirname, 'build')
10 |  },
11 |   module: {
12 |     rules: [
13 |       {
14 |         test: /\.css$/,
15 |         use: [
16 |           // 创建 style 标签，将样式放入
17 |           // 'style-loader',
18 |           // 这个 loader 取代 style-loader。作用：提取 js 中的 css 成单独文件
19 |           MiniCssExtractPlugin.loader,
20 |           // 将 css 文件整合到 js 文件中
21 |           'css-loader'
22 |         ]
23 |       },
24 |       {
25 |         test: /\.?(jpg|png|gif)$/i,
26 |         type: 'asset'
27 |       }
28 |     ]
29 |   },
30 |   plugins: [
31 |     new HtmlWebpackPlugin({
32 |       template: './src/index.html'
```

```

33     }),
34     // 将css提取成单独文件
35     new MiniCssExtractPlugin({
36         filename: 'css/built.css'
37     })
38 ],
39 mode: 'development'
40 }

```

5. 运行指令: `webpack`

4.2. css 兼容性处理

1. 创建文件



2. 下载 loader

```

1 npm install --save-dev postcss-loader postcss-preset-env
2 ## --save-dev 相当于 -D

```

3. 修改配置文件

webpack.config.js:

```

1 const { resolve } = require('path')
2 const HtmlWebpackPlugin = require('html-webpack-plugin')
3 const MiniCssExtractPlugin = require('mini-css-extract-plugin')
4
5 // 设置 nodejs 环境变量
6 // process.env.NODE_ENV = 'development'
7
8 module.exports = {
9     entry: './src/js/index.js',
10    output: {
11        filename: 'js/built.js',
12        path: resolve(__dirname, 'build')
13    },
14    module: {

```

```

15     rules: [
16       {
17         test: /\.css$/,
18         use: [
19           MiniCssExtractPlugin.loader,
20           'css-loader',
21           /*
22            css 兼容性处理: postcss --> postcss-loader postcss-preset-env
23
24            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
25            // 更多的browserslist配置可以去github搜索 关键字`browserslist`
26            "browserslist": {
27              // 开发环境 --> 设置nodejs环境变量: process.env.NODE_ENV =
development
28              "development": [
29                "last 1 chrome version", // 兼容最近的一个 chrome 版本
30                "last 1 firefox version",
31                "last 1 safari version"
32              ],
33              // 生产环境: 默认是看生产环境
34              "production": [
35                ">0.2%",
36                "not dead",
37                "not op_mini all"
38              ]
39            }
40          */
41          // 使用 loader 的默认配置
42          // 'postcss-loader'
43          // 修改loader的配置（写成对象的形式）
44          {
45            loader: 'postcss-loader',
46            options: {
47              postcssOptions: {
48                plugins: [require('postcss-preset-env')()],
49              },
50            },
51          },
52        ]
53      }
54    ],
55    plugins: [
56      new HtmlWebpackPlugin({
57        template: './src/index.html'
58      }),
59      new MiniCssExtractPlugin({
60        filename: 'css/built.css'
61      })
62    ],
63    mode: 'development'
64  }
65 }

```

4. 修改 package.json

```
1  "browserslist": {
```



```

2 // 开发环境 --> 设置nodejs环境变量: process.env.NODE_ENV = development, 不设置
  默认是production
3 "development": [
4   "last 1 chrome version", // 兼容最近的一个 chrome 版本
5   "last 1 firefox version",
6   "last 1 safari version"
7 ],
8 // 生产环境: 默认是看生产环境
9 "production": [
10  ">0.2%",
11  "not dead",
12  "not op_mini all"
13 ]
14 }

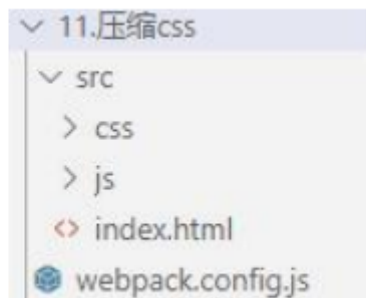
```

- 更多的browserslist配置可以去github搜索 关键字

4. 运行指令: `webpack`

4.3. 压缩 css

1. 创建文件



2. 下载安装包

```
1 | npm install optimize-css-assets-webpack-plugin -D
```

3. 修改配置文件

```

1 const { resolve } = require('path')
2 const HtmlWebpackPlugin = require('html-webpack-plugin')
3 const MiniCssExtractPlugin = require('mini-css-extract-plugin')
4 const CssMinimizerWebpackPlugin = require('css-minimizer-webpack-plugin')
5
6 // 设置 nodejs 环境变量
7 process.env.NODE_ENV = 'development'
8
9 module.exports = {
10   entry: './src/js/index.js',
11   output: {
12     filename: 'js/built.js',
13     path: resolve(__dirname, 'build'),
14   },
15   module: {
16     rules: [

```

```

17     {
18         test: /\.css$/,
19         use: [
20             // 插件 style 标签，将样式放入
21             // 'style-loader',
22             // 这个 loader 取代 style-loader 。作用：提取js中的css成单独文件
23             MiniCssExtractPlugin.loader,
24             // 将css 文件整合到js文件中
25             'css-loader',
26
27             // 使用loader默认配置
28             // 直接: loader: 'postcss-loader',
29             // 修改loader配置，需要写到对象里面
30             {
31                 loader: 'postcss-loader',
32                 options: {
33                     postcssOptions: {
34                         plugins: ['postcss-preset-env'],
35                     },
36                 },
37             },
38         ],
39     },
40     {
41         test: /\. (jpg|png|gif) $/,
42         type: 'asset',
43     },
44 ],
45 },
46 plugins: [
47     new HtmlWebpackPlugin({
48         template: './src/index.html',
49     }),
50     new MiniCssExtractPlugin({
51         filename: 'css/built.css',
52     }),
53     // 压缩css
54     new CssMinimizerWebpackPlugin(),
55 ],
56 mode: 'development',
57 }

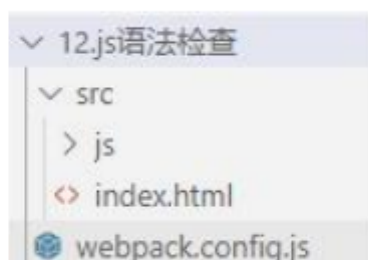
```

4. 运行指令： `webpack`

4.4. js 语法检查 (eslint)

1. 创建文件

!



2. 下载安装包

```
1 | npm install eslint-loader eslint eslint-config-airbnb-base eslint-plugin-import -D
```

3. 修改配置文件

```
1 | const { resolve } = require('path')
2 | const HtmlWebpackPlugin = require('html-webpack-plugin')
3 | const MiniCssExtractPlugin = require('mini-css-extract-plugin')
4 | const OptimizeCssAssetsWebpackPlugin = require('optimize-css-assets-webpack-plugin')
5 |
6 | // 设置 nodejs 环境变量
7 | // process.env.NODE_ENV
8 |
9 | module.exports = {
10 |   entry: './src/js/index.js',
11 |   output: {
12 |     filename: 'js/built.js',
13 |     path: resolve(__dirname, 'build')
14 |   },
15 |   module: {
16 |     rules: [
17 |       /*语法检查: eslint-loader eslint
18 |        注意: 只检查自己写的源代码, 第三方的库是不用检查的
19 |        设置检查规则:
20 |        package.json 中 eslintConfig 中设置~
21 |        "eslintConfig": {
22 |          "extends": "airbnb-base"
23 |        }
24 |        airbnb --> eslint-config-airbnb-base eslint-plugin-import eslint
25 |       */
26 |       {
27 |         test: /\.js$/,
28 |         exclude: /node_modules/,
29 |         loader: 'eslint-loader',
30 |         options: {
31 |           // 自动修复 eslint 的错误
32 |           fix: true
33 |         }
34 |       }
35 |     ]
36 |   },
37 |   plugins: [
38 |     new HtmlWebpackPlugin({
39 |       template: './src/index.html'
40 |     }),
41 |     new MiniCssExtractPlugin({
42 |       filename: 'css/built.css'
43 |     }),
44 |     new OptimizeCssAssetsWebpackPlugin()
45 |   ],
46 |   mode: 'development'
47 | }
```

打包时，在js文件中，不推荐出现console.log，可以写成

```
1 // index.js
2 // 下一行 eslint 所有规则都失效（下一行不进行eslint检查）
3 // eslint-disable-next-line
4 console.log('innn')
```

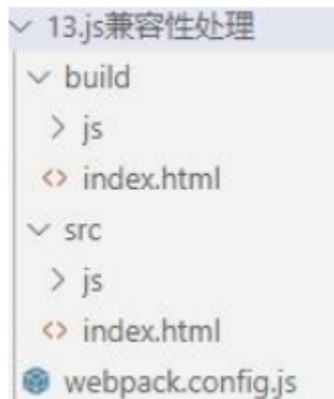
4. 配置 package.json

```
1 "eslintConfig": {
2   "extends": "airbnb-base",
3   "env": {
4     "browser": true
5   }
6 }
```

5. 运行指令：webpack

4.5. js 兼容性处理

1. 创建文件



2. 下载安装包

```
1 npm install babel-loader @babel/core @babel/preset-env @babel/polyfill
  core-js
```

3. 修改配置文件

```
1 const { resolve } = require('path')
2 const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4 module.exports = {
5   entry: './src/js/index.js',
6   output: {
7     filename: 'js/built.js',
8     path: resolve(__dirname, 'build')
9   },
10  module: {
11    rules: [
12      /*
13      js兼容性处理: babel-loader @babel/core
```

```

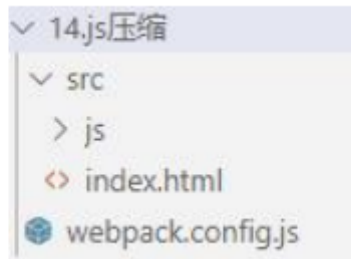
14      1. 基本js兼容性处理 --> @babel/preset-env
15      问题：只能转换基本语法，如promise等高级语法不能转换
16      2. 全部兼容性处理 --> @babel/polyfill
17      用法：直接在js文件中引入：import '@babel/polyfill'
18      问题：我只要解决部分兼容性问题，但是将所有兼容性代码全部引入，体积太大了~
19      3. 需要做兼容性处理的就做：按需加载 --> corejs
20      使用第三种方案的时候，不能在用第二种方案，必须注释掉
21      总结：我们是结合第一种（基本兼容性处理）和第三种方法（高级语法兼容性处理）完成兼容
    型处理，第二种体积太大了一般不考虑
22      */
23      {
24          test: /\.js$/,
25          exclude: /node_modules/,
26          loader: 'babel-loader',
27          options: {
28              // 预设：指示 babel 做怎么样的兼容性处理
29              presets: [
30                  '@babel/preset-env',
31                  {
32                      // 按需加载
33                      useBuiltIns: 'usage',
34                      // 指定 core-js 版本
35                      corejs: {
36                          version: 3
37                      },
38                      // 指定兼容性做到哪个版本浏览器
39                      targets: {
40                          chrome: '60',
41                          firefox: '60',
42                          ie: '9',
43                          safari: '10',
44                          edge: '17'
45                      }
46                  }
47              ]
48          }
49      }
50  ],
51 },
52 plugins: [
53     new HtmlWebpackPlugin({
54         template: './src/index.html'
55     })
56 ],
57 mode: 'development'
58 }

```

4. 运行命令： `webpack`

4.6. js 压缩

1. 创建文件

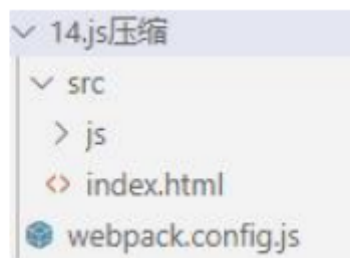


2. 修改配置文件

```
1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  module.exports = {
5    entry: './src/js/index.js',
6    output: {
7      filename: 'js/built.js',
8      path: resolve(__dirname, 'build')
9    },
10   plugins: [
11     new HtmlWebpackPlugin({
12       template: './src/index.html'
13     })
14   ],
15   // 生产环境下会自动压缩js代码
16   // 生产环境下会自动加载很多插件，其中的 uglifyPlugin(现在用terser插件) 就是用来压缩
   js 代码的
17   mode: 'production'
18 }
```

4.7. HTML 压缩

1. 创建文件



2. 修改配置文件

```
1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  module.exports = {
5    entry: './src/js/index.js',
6    output: {
7      filename: 'js/built.js',
8      path: resolve(__dirname, 'build')
9    },
10   plugins: [
```

```

11     new HtmlWebpackPlugin({
12       template: './src/index.html',
13       // 压缩 html 代码
14       minify: {
15         // 移除空格
16         collapseWhitespace: true,
17         // 移除注释
18         removeComments: true
19       }
20     })
21   ],
22   mode: 'production'
23 }

```

4.8. 生产环境配置

1. 创建文件



2. 修改配置文件（用到的插件和loader要提前下载好）

```

1  const { resolve } = require('path')
2  const MiniCssExtractPlugin = require('mini-css-extract-plugin')
3  const OptimizeCssAssetsWebpackPlugin = require('optimize-css-assets-webpack-plugin') // css压缩
4  const HtmlWebpackPlugin = require('html-webpack-plugin') // 生成并引入 index.html
5
6  // 定义 nodejs 环境变量：决定使用 browserslist 的哪个环境
7  process.env.NODE_ENV = 'production'
8
9  // 复用 loader
10 const commonCssLoader = [
11   MiniCssExtractPlugin.loader,
12   'css-loader',
13   {
14     // css兼容性处理
15     // 还需要在 package.json中定义browserslist
16     loader: 'postcss-loader',
17     options: {
18       postcssOptions: {
19         plugins: [require('postcss-preset-env')()]
20       }
21     }
22   }
23 ]
24
25 module.exports = {
26   entry: './src/js/index.js',
27   output: {
28     filename: 'js/built.js',

```

```

29     path: resolve(__dirname, 'build')
30   },
31   module: {
32     rules: [
33       {
34         test: /\.css$/,
35         use: [...commonCssLoader]
36       },
37       {
38         test: /\.less$/,
39         use: [...commonCssLoader, 'less-loader']
40       },
41     /*
42      正常来讲，一个文件只能被一个 loader 处理。
43      当一个文件要被多个 loader 处理，那么一定要指定 loader 执行的先后顺序：先执行
44      eslint 再执行 babel
45      */
46     {
47       // js 语法检查
48       // 在 package.json 中 eslintConfig --> airbnb
49       test: /\.js$/,
50       exclude: /node_modules/,
51       // 优先执行
52       enforce: 'pre',
53       loader: 'eslint-loader',
54       options: {
55         // 自动修正
56         fix: true
57       },
58     },
59     {
60       // js 兼容性处理
61       test: /\.js$/,
62       exclude: /node_modules/,
63       loader: 'babel-loader',
64       options: {
65         presets: [
66           [
67             '@babel/preset-env',
68             {
69               // 按需加载
70               useBuiltIns: 'usage',
71               corejs: {version: 3},
72               // 要兼容的浏览器版本
73               targets: {
74                 chrome: '60',
75                 firefox: '50'
76               }
77             }
78           ]
79         ]
80       },
81     },
82     {
83       // 处理图片资源
84       test: /\.(jpg|png|gif|jpeg)$/i,
85       type: 'asset',
86       generator: {

```



```

86         filename: '[name]_[contenthash:6][ext]'
87     },
88     parser: {
89         dataUrlCondition: {
90             maxSize: 10 * 1024
91         }
92     },
93 },
94 {
95     // 处理 html 中的图片
96     test: /\.html$/,
97     loader: 'html-loader'
98 },
99 {
100     // 处理其他资源
101     exclude: /\. (js|css|less|html|jpg|png|gif|jpeg)$/,
102     type: 'asset/resource',
103     generator: {
104         filename: 'media/[name]_[contenthash:6][ext]'
105     }
106 }
107 ]
108 },
109 plugins: [
110     // 打包html资源
111     new HtmlWebpackPlugin({
112         template: './src/index.html',
113         // 压缩html
114         minify: {
115             collapseWhitespace: true,
116             removeComments: true
117         }
118     }),
119     // 提取css成单独文件
120     new MiniCssExtractPlugin({
121         filename: 'css/built.css'
122     }),
123     // 压缩 css
124     new CssMinimizerWebpackPlugin()
125 ],
126 mode: 'production'
127 }

```

3. 运行指令: `webpack`

5. 第五章: webpack 优化配置

5.1. webpack性能优化

- 开发环境性能优化
- 生产环境性能优化

5.1.1. 开发环境性能优化

- 优化打包构建速度（HMR）
 - 问题：只要修改了一个模块，重新构建时其他模块也会重新构建
 - HMR（热模块替换）：构建时如果只有一个模块发生变化，只会重新构建这一个模块，而其他模块会用它之前的缓存
- 优化代码调试（source-map）
 - source-map

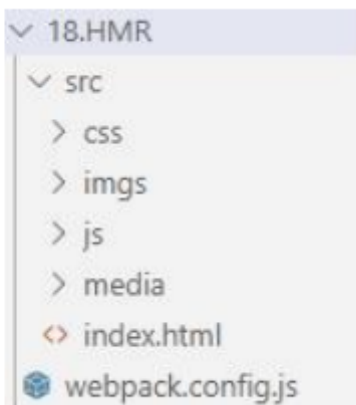
5.1.2. 生产环境性能优化

- 优化打包构建速度
 - oneOf
 - babel 缓存
 - 多进程打包
 - externals(完全不打包，通过链接引入)
 - dll(打包一次，之后就不打包了)
- 优化代码运行的性能
 - 缓存（文件资源）
 - hash
 - chunkhash
 - contenthash
 - tree shaking（树摇）
 - code split（代码分割）
 - 可以配合 dll 把node_modules库中的各种第三方库都分
 - 懒加载 / 预加载
 - pwa（离线可访问技术）

5.2. HMR（模块热更新）

说明：使我们代码在更新的时候不是全部更新，而是只更新变化的部分

1. 创建文件



2. 修改配置文件

```
1  /*
2    HMR: hot module replacement 热模块替换 / 模块热更新
3    作用：一个模块发生变化，只会重新打包这一个模块（而不是打包所有模块）
4    极大提升构建速度
5
6    样式文件：可以使用 HMR 功能，因为 style-loader 内部实现了
7    js文件：默认不能使用 HMR 功能 --> 需要修改 js 代码，在js文件中添加支持HMR功能的
    代码
8
9    注意：HMR功能对js的处理，只能处理非入口js文件的其他文件。
10   if(module.hot) {
11     // 一旦 module.hot 为 true，说明开启了 HMR 功能。 --> 让HMR功能代码生效
12     module.hot.accept('./print.js', function () { // 方法会监听print.js
13       文件的变化，一旦发生变化，其他模块不会重新打包构建。
14       // 会执行后面的回调函数
15       print()
16     })
17   }
18   html文件：默认不能使用 HMR 功能，同时会导致问题：html 文件不能热更新了~（不用做
19   HMR功能）
20   解决：修改entry入口，将 html 文件引入
21 */
22
23 const { resolve } = require('path')
24 const HtmlWebpackPlugin = require('html-webpack-plugin')
25
26 module.exports = {
27   entry: ['./src/js/index.js', './src/index.html'],
28   output: {
29     filename: 'js/built.js',
30     path: resolve(__dirname, 'build')
31   },
32   module: {
33     rules: [
34       // loader 配置
35       {
36         // 处理 less 资源
37         test: /\.less$/,
38         use: ['style-loader', 'css-loader', 'less-loader']
39       },
40       {
41         // 处理 css 资源
42         test: /\.css$/,
43         use: ['style-loader', 'css-loader']
44       },
45       {
46         // 处理图片资源
47         test: /\.(jpg|png|gif)$/,
48         loader: 'url-loader',
49         options: {
50           limit: 8 * 1024,
51           name: '[hash:10].[ext]',
52           // 关闭 es6 模块化
53           esModule: false,
54           outputPath: 'imgs'
55         }
56       }
57     ]
58   }
59 }
```

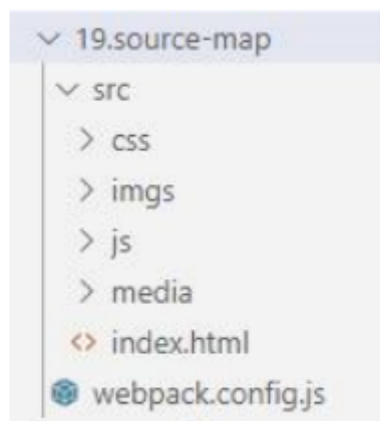
```

53     },
54     {
55         // 处理 html 中 img 资源
56         test: /\.html$/,
57         loader: 'html-loader'
58     },
59     {
60         // 处理其他资源
61         exclude: /\. (html|js|css|less|jpg|png|gif)/,
62         loader: 'file-loader',
63         options: {
64             name: '[hash:10].[ext]',
65             outputPath: 'media'
66         }
67     }
68 ],
69 },
70 plugins: [
71     // plugins 的配置
72     new HtmlWebpackPlugin({
73         template: './src/index.html'
74     })
75 ],
76 mode: 'development',
77 devServer: {
78     static: {
79         directory: resolve(__dirname, 'build'),
80     },
81     compress: true,
82     port: 5000,
83     open: true,
84     // 开启 HMR 功能
85     // 当修改了webpack配置，新配置要想生效，必须重启webpack服务
86     // hot: true, // webpack5 默认开启
87 },
88 }

```

5.3. source-map (代码映射)

1. 创建文件



2. 修改配置文件

```
1  const HtmlWebpackPlugin = require('html-webpack-plugin')
2  const { resolve } = require('path')
3
4  module.exports = {
5    entry: ['./src/js/index.js', './src/index.html'],
6    output: {
7      filename: 'js/built.js',
8      path: resolve(__dirname, 'build'),
9    },
10   module: {
11     rules: [
12       {
13         test: /\.css$/,
14         use: ['style-loader', 'css-loader'],
15       },
16       {
17         test: /\.less$/,
18         use: ['style-loader', 'css-loader', 'less-loader'],
19       },
20       {
21         // 处理图片资源
22         test: /\..(jpg|png|jpeg|gif)$/i,
23         type: 'asset',
24         generator: {
25           filename: 'imgs/[name]_[contenthash:6][ext]',
26         },
27         parser: {
28           dataUrlCondition: {
29             maxSize: 8 * 1024,
30           },
31         },
32       },
33       {
34         // 处理html中的 图片资源
35         test: /\.html$/,
36         loader: 'html-loader',
37       },
38       {
39         // 处理字体资源
40         test: /\..(eot|ttf|woff|svg)$/i,
41         type: 'asset/resource',
42         generator: {
43           outputPath: 'media',
44         },
45       },
46     ],
47   },
48   plugins: [
49     new HtmlWebpackPlugin({
50       template: './src/index.html',
51     }),
52   ],
53   mode: 'development',
54   devServer: {
55     static: {
56       directory: resolve(__dirname, 'build'),
57     },
58     compress: true,
```

```

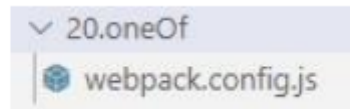
59     port: 5000,
60     open: true,
61     hot: true, // webpack5 默认开启
62   },
63   devtool: 'eval-source-map',
64 }
65
66 /*
67   source-map: 一种提供源代码到构建后代码映射的技术（如果构建后代码出错了，通过映射可以追踪到源代码错误）
68
69   [inline-|hidden-|eval-][nosources-][cheap-[module-]]source-map
70
71   source-map: 外部
72     错误代码的准确信息 和 源代码的
73   inline-source-map: 内联
74     只生成一个内联source-map
75     错误代码的准确信息 和 源代码的错误位置
76   hidden-source-map: 外部
77     错误代码的错误原因，但是没有错误位置
78     不能追踪到源代码错误，只能提示到构建后代码的错误位置
79   eval-source-map: 内联
80     每一个文件都生成一个对应的source-map，都在eval
81     错误代码的准确信息 和 源代码的错误位置
82   nosources-source-map: 外部
83     错误代码的准确信息，但是没有任何源代码信息
84   cheap-source-map: 外部
85     错误代码的准确信息 和 源代码的错误位置
86     只能精确到行
87   cheap-module-source-map: 外部
88     错误代码的准确信息 和 源代码的错误位置
89     module会将loader的source-map加入
90
91   内联 和 外部的区别：1. 外部生成了文件，内联没有 2. 内联构建速度更快
92
93   开发环境：速度快，调式更友好
94     速度快(eval>inline>cheap>...)
95       eval-cheap-source-map
96       eval-source-map
97     调试更友好
98       source-map
99       cheap-module-source-map
100      cheap-source-map
101
102    --> eval-source-map(开发环境一般用这个) / eval-cheap-module-source-map
103
104   生产环境：源代码要不要隐藏？调试要不要更友好
105     // 内联会让代码体积变大，所以在生产环境不用内联
106     nosources-source-map 全部隐藏
107     hidden-source-map 只隐藏源代码，会提示构建后代码错误
108
109    --> source-map(生产环境一般用这个) / cheap-module-source-map
110
111  */
112

```

3. 运行指令：webpack

5.4. oneOf (loader 限单次匹配)

1. 创建文件



2. 修改配置文件

```
1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3  const MiniCssExtractPlugin = require('mini-css-extract-plugin')
4  const CssMinimizerWebpackPlugin = require('css-minimizer-webpack-plugin')
5
6  // 定义nodejs环境变量: 决定使用browserslist的哪个环境
7  process.env.NODE_ENV = 'production'
8
9  // 复用loader
10 const commonCssloader = [
11   MiniCssExtractPlugin.loader,
12   'css-loader',
13   {
14     // css兼容性处理
15     // 还需要再 package.json中定义browserslist
16     loader: 'postcss-loader',
17     options: {
18       postcssOptions: {
19         plugins: [require('postcss-preset-env')()],
20       },
21     },
22   },
23 ]
24
25 module.exports = {
26   entry: './src/js/index.js',
27   output: {
28     filename: 'js/built.js',
29     path: resolve(__dirname, 'build')
30   },
31   module: {
32     rules: [
33       {
34         // 在package.json中设置eslintConfig --> airbnb
35         test: /\.js$/,
36         exclude: /node_modules/,
37         // 优先执行
38         enforce: 'pre',
39         loader: 'eslint-loader', // eslint 语法检查
40         options: {
41           fix: true
42         }
43       },
44     ]
45   }
46 }
```

```

45 // 一下loader只会匹配一个，解决每个文件都会被全部loader过一遍（相当于找到直接
break掉）
46 // 不能有两个配置处理同一类型的文件
47 oneOf: [
48   {
49     test: /\.css$/,
50     use: [...commonCssloader]
51   },
52   {
53     test: /\.less$/,
54     use: [...commonCssloader, 'less-loader']
55   },
56   /*
57   正常来讲，一个文件只能被一个loader处理。
58   当一个文件要被多个loader处理，那么一定要指定loader执行的先后顺序
59   */
60
61   {
62     // js兼容性处理
63     test: /\.js$/,
64     exclude: /node_modules/,
65     loader: 'babel-loader',
66     options: {
67       presets: [
68         [
69           '@babel/preset-env',
70           {
71             useBuiltIns: 'usage',
72             corejs: {
73               version: 3
74             },
75             targets: {
76               chrome: '60',
77               firefox: '50',
78               ie: '9',
79               safari: '10'
80             }
81           }
82         ]
83       ]
84     }
85   },
86   {
87     // 处理图片资源
88     test: /\.(jpg|png|gif|jpeg)$/i,
89     type: 'asset',
90     generator: {
91       filename: '[name]_[contenthash:6][ext]'
92     },
93     parser: {
94       dataUrlCondition: {
95         maxSize: 10 * 1024
96       }
97     }
98   },
99   {
100     // 处理html 中的图片资源
101     test: /\.html$/,

```



```

102         loader: 'html-loader'
103     },
104     {
105         // 处理其他资源
106         exclude: /\. (js|css|less|html|jpg|png|gif|jpeg)$/,
107         type: 'asset/resource',
108         generator: {
109             filename: 'media/[name]_[contenthash:6][ext]'
110         }
111     }
112 ]
113 }
114 ]
115 },
116 plugins: [
117     // 打包html资源
118     new HtmlWebpackPlugin({
119         template: './src/index.html',
120         // 压缩html
121         minify: {
122             collapseWhitespace: true,
123             removeComments: true
124         }
125     }),
126     // 提取css成单独文件
127     new MiniCssExtractPlugin({
128         filename: 'css/built.css'
129     }),
130     // 压缩 css
131     new CssMinimizerWebpackPlugin()
132 ],
133 mode: 'production'
134 }
135

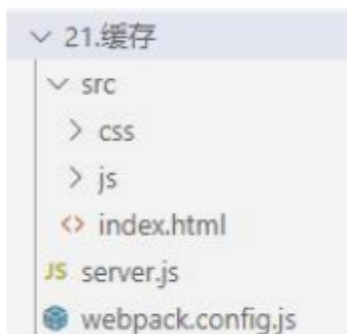
```

3. 运行指令: `webpack`

5.5. 缓存

- babel 缓存
- 文件资源缓存

1. 创建文件



2. 修改配置文件

```

1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3  const MiniCssExtractPlugin = require('mini-css-extract-plugin')
4  const CssMinimizerWebpackPlugin = require('css-minimizer-webpack-plugin')
5
6  /*
7   缓存:
8   babel缓存
9       cacheDirectory: true
10      --> 让第二次打包更快
11  文件资源缓存
12      hash: 每次webpack构建时会生成一个唯一的hash值。
13      问题: 因为js和css同时使用一个hash值。
14      如果重新打包, 会导致所有的缓存失效。(可能我却只改动一个文件)
15      chunkhash: 根据chunk生成的hash值。如果打包来源于同一个chunk, 那么hash值一样
16      问题: js和css的hash值还是一样的
17      因为css是在js中被引入的, 所以同属于一个chunk
18      contenthash: 根据文件的内容生成hash值。不同文件hash值一定不一样
19      --> 让代码上线运行缓存更好使用
20  */
21
22  // 定义nodejs环境变量: 决定使用browserslist的哪个环境
23  process.env.NODE_ENV = 'production'
24
25  // 复用loader
26  const commonCssloader = [
27    MiniCssExtractPlugin.loader,
28    'css-loader',
29    {
30      // css兼容性处理
31      // 还需要再 package.json中定义browserslist
32      loader: 'postcss-loader',
33      options: {
34        postcssOptions: {
35          plugins: [require('postcss-preset-env')()]
36        }
37      }
38    }
39  ]
40
41  module.exports = {
42    entry: './src/js/index.js',
43    output: {
44      filename: 'js/built.[contenthash:10].js',
45      path: resolve(__dirname, 'build')
46    },
47    module: {
48      rules: [
49        {
50          // 在package.json中设置eslintConfig --> airbnb
51          test: /\.js$/,
52          exclude: /node_modules/,
53          // 优先执行
54          enforce: 'pre',
55          loader: 'eslint-loader', // eslint 语法检查
56          options: {
57            fix: true
58          }

```

```

59     },
60     {
61         // 以下loader只会匹配一个，解决每个文件都会被全部loader过一遍（相当于找到直接
break掉）
62         // 不能有两个配置处理同一类型的文件
63         oneOf: [
64             {
65                 test: /\.css$/,
66                 use: [...commonCssloader]
67             },
68             {
69                 test: /\.less$/,
70                 use: [...commonCssloader, 'less-loader']
71             },
72             /*
73             正常来讲，一个文件只能被一个loader处理。
74             当一个文件要被多个loader处理，那么一定要指定loader执行的先后顺序
75             */
76
77             {
78                 // js兼容性处理
79                 test: /\.js$/,
80                 exclude: /node_modules/,
81                 loader: 'babel-loader',
82                 options: {
83                     presets: [
84                         [
85                             '@babel/preset-env',
86                             {
87                                 useBuiltIns: 'usage',
88                                 corejs: {
89                                     version: 3
90                                 },
91                                 targets: {
92                                     chrome: '60',
93                                     firefox: '50',
94                                     ie: '9',
95                                     safari: '10'
96                                 }
97                             }
98                         ]
99                     ],
100                     // 开启babel缓存，第二次构建时，会读取之前的缓存
101                     cachedDirectory: true
102                 }
103             },
104             {
105                 // 处理图片资源
106                 test: /\.(jpg|png|gif|jpeg)$/i,
107                 type: 'asset',
108                 generator: {
109                     filename: '[name]_[contenthash:6][ext]'
110                 },
111                 parser: {
112                     dataUrlCondition: {
113                         maxSize: 10 * 1024
114                     }
115                 }
116             }
117         ]
118     }
119 }

```

```

116     },
117     {
118         // 处理html 中的图片资源
119         test: /\.html$/,
120         loader: 'html-loader'
121     },
122     {
123         // 处理其他资源
124         exclude: /\. (js|css|less|html|jpg|png|gif|jpeg)$/,
125         type: 'asset/resource',
126         generator: {
127             filename: 'media/[name]_[contenthash:6][ext]'
128         }
129     }
130 ]
131 }
132 ]
133 },
134 plugins: [
135     // 打包html资源
136     new HtmlWebpackPlugin({
137         template: './src/index.html',
138         // 压缩html
139         minify: {
140             collapseWhitespace: true,
141             removeComments: true
142         }
143     }),
144     // 提取css成单独文件
145     new MiniCssExtractPlugin({
146         filename: 'css/built.[contenthash:10].css'
147     }),
148     // 压缩 css
149     new CssMinimizerWebpackPlugin()
150 ],
151 mode: 'production',
152 devtool: 'source-map'
153 }
154

```

package.json

```

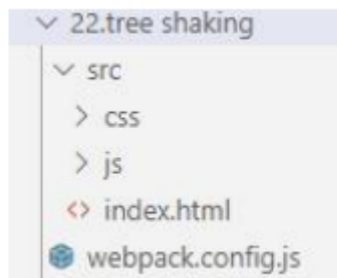
1  "eslintConfig": {
2    "extends": "airbnb-base",
3    "env": {
4      "browser": true
5    }
6  },

```

3. 运行指令: `webpack`

5.6. tree shaking (树摇)

1. 创建文件



2. 修改配置文件

```
1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3  const MiniCssExtractPlugin = require('mini-css-extract-plugin')
4  const CssMinimizerWebpackPlugin = require('css-minimizer-webpack-plugin')
5
6  /*
7   tree shaking: 去除无用代码
8   前提: 1. 必须使用ES6模块化 2. 开启production环境
9   作用: 减少代码体积
10
11   在package.json中配置
12   "sideEffects": false 所有代码都没有副作用 (都可以进行tree shaking)
13   问题: 可能会把css / @babel/polyfill (副作用) 文件干掉
14   "sideEffects": [*.css]
15  */
16
17  // 定义nodejs环境变量: 决定使用browserslist的哪个环境
18  process.env.NODE_ENV = 'production'
19
20  // 复用loader
21  const commonCssloader = [
22    MiniCssExtractPlugin.loader,
23    'css-loader',
24    {
25      // css兼容性处理
26      // 还需要再 package.json中定义browserslist
27      loader: 'postcss-loader',
28      options: {
29        postcssOptions: {
30          plugins: [require('postcss-preset-env')()]
31        }
32      }
33    }
34  ]
35
36  module.exports = {
37    entry: './src/js/index.js',
38    output: {
39      filename: 'js/built.[contenthash:10].js',
40      path: resolve(__dirname, 'build')
41    },
42    module: {
```

```

43 rules: [
44   {
45     // 在package.json中设置eslintConfig --> airbnb
46     test: /\.js$/,
47     exclude: /node_modules/,
48     // 优先执行
49     enforce: 'pre',
50     loader: 'eslint-loader', // eslint 语法检查
51     options: {
52       fix: true
53     }
54   },
55   {
56     // 以下loader只会匹配一个，解决每个文件都会被全部loader过一遍（相当于找到直接
break掉）
57     // 不能有两个配置处理同一类型的文件
58     oneOf: [
59       {
60         test: /\.css$/,
61         use: [...commonCssloader]
62       },
63       {
64         test: /\.less$/,
65         use: [...commonCssloader, 'less-loader']
66       },
67       /*
68         正常来讲，一个文件只能被一个loader处理。
69         当一个文件要被多个loader处理，那么一定要指定loader执行的先后顺序
70         */
71
72       {
73         // js兼容性处理
74         test: /\.js$/,
75         exclude: /node_modules/,
76         loader: 'babel-loader',
77         options: {
78           presets: [
79             [
80               '@babel/preset-env',
81               {
82                 useBuiltIns: 'usage',
83                 corejs: {
84                   version: 3
85                 },
86                 targets: {
87                   chrome: '60',
88                   firefox: '50',
89                   ie: '9',
90                   safari: '10'
91                 }
82               }
83             ]
94           ],
95           // 开启babel缓存，第二次构建时，会读取之前的缓存
96           cacheDirectory: true
97         }
98       },
99     ]

```

```

100         // 处理图片资源
101         test: /\. (jpg|png|gif|jpeg)$/i,
102         type: 'asset',
103         generator: {
104             filename: '[name]_[contenthash:6][ext]'
105         },
106         parser: {
107             dataUrlCondition: {
108                 maxSize: 10 * 1024
109             }
110         }
111     },
112     {
113         // 处理html 中的图片资源
114         test: /\.html$/,
115         loader: 'html-loader'
116     },
117     {
118         // 处理其他资源
119         exclude: /\. (js|css|less|html|jpg|png|gif|jpeg)$/i,
120         type: 'asset/resource',
121         generator: {
122             filename: 'media/[name]_[contenthash:6][ext]'
123         }
124     }
125 ]
126 }
127 ]
128 },
129 plugins: [
130     // 打包html资源
131     new HtmlWebpackPlugin({
132         template: './src/index.html',
133         // 压缩html
134         minify: {
135             collapseWhitespace: true,
136             removeComments: true
137         }
138     }),
139     // 提取css成单独文件
140     new MiniCssExtractPlugin({
141         filename: 'css/built.[contenthash:10].css'
142     }),
143     // 压缩 css
144     new CssMinimizerWebpackPlugin()
145 ],
146 mode: 'production'
147 // devtool: 'inline-source-map'
148 }
149 const { resolve } = require('path')
150 const HtmlWebpackPlugin = require('html-webpack-plugin')
151 const MiniCssExtractPlugin = require('mini-css-extract-plugin')
152 const CssMinimizerWebpackPlugin = require('css-minimizer-webpack-plugin')
153
154 /*
155 tree shaking: 去除无用代码
156 前提: 1. 必须使用ES6模块化 2. 开启production环境
157 作用: 减少代码体积

```

```

158
159     在package.json中配置
160     "sideEffects": false  所有代码都没有副作用（都可以进行tree shaking）
161     问题：可能会把css / @babel/polyfill（副作用）文件干掉
162     "sideEffects": [*.css]
163 */
164
165 // 定义nodejs环境变量：决定使用browserslist的哪个环境
166 process.env.NODE_ENV = 'production'
167
168 // 复用loader
169 const commonCssloader = [
170   MiniCssExtractPlugin.loader,
171   'css-loader',
172   {
173     // css兼容性处理
174     // 还需要再 package.json中定义browserslist
175     loader: 'postcss-loader',
176     options: {
177       postcssOptions: {
178         plugins: [require('postcss-preset-env')()]
179       }
180     }
181   }
182 ]
183
184 module.exports = {
185   entry: './src/js/index.js',
186   output: {
187     filename: 'js/built.[contenthash:10].js',
188     path: resolve(__dirname, 'build')
189   },
190   module: {
191     rules: [
192       {
193         // 在package.json中设置eslintConfig --> airbnb
194         test: /\.js$/,
195         exclude: /node_modules/,
196         // 优先执行
197         enforce: 'pre',
198         loader: 'eslint-loader', // eslint 语法检查
199         options: {
200           fix: true
201         }
202       },
203       {
204         // 以下loader只会匹配一个，解决每个文件都会被全部loader过一遍（相当于找到直接
205         break掉）
206         // 不能有两个配置处理同一类型的文件
207         oneOf: [
208           {
209             test: /\.css$/,
210             use: [...commonCssloader]
211           },
212           {
213             test: /\.less$/,
214             use: [...commonCssloader, 'less-loader']
215           }
216         ]
217       }
218     ]
219   }
220 }

```



```

215      /*
216      正常来讲，一个文件只能被一个loader处理。
217      当一个文件要被多个loader处理，那么一定要指定loader执行的先后顺序
218      */
219
220      {
221        // js兼容性处理
222        test: /\.js$/,
223        exclude: /node_modules/,
224        loader: 'babel-loader',
225        options: {
226          presets: [
227            [
228              '@babel/preset-env',
229              {
230                useBuiltIns: 'usage',
231                corejs: {
232                  version: 3
233                },
234                targets: {
235                  chrome: '60',
236                  firefox: '50',
237                  ie: '9',
238                  safari: '10'
239                }
240              }
241            ]
242          ],
243          // 开启babel缓存，第二次构建时，会读取之前的缓存
244          cacheDirectory: true
245        }
246      },
247      {
248        // 处理图片资源
249        test: /\.jpg|png$/,
250        const HtmlWebpackPlugin = require('html-webpack-plugin')
251        const MiniCssExtractPlugin = require('mini-css-extract-plugin')
252        const CssMinimizerWebpackPlugin = require('css-minimizer-webpack-plugin')
253
254      /*
255      tree shaking: 去除无用代码
256      前提: 1. 必须使用ES6模块化 2. 开启production环境
257      作用: 减少代码体积
258
259      在package.json中配置
260      "sideEffects": false 所有代码都没有副作用（都可以进行tree shaking）
261      问题: 可能会把css / @babel/polyfill（副作用）文件干掉
262      "sideEffects": [*.css]
263      */
264
265      // 定义nodejs环境变量: 决定使用browserslist的哪个环境
266      process.env.NODE_ENV = 'production'
267
268      // 复用loader
269      const commonCssloader = [
270        MiniCssExtractPlugin.loader,
271        'css-loader',
272        {

```

```

273 // css兼容性处理
274 // 还需要再 package.json中定义browserslist
275 loader: 'postcss-loader',
276 options: {
277   postcssOptions: {
278     plugins: [require('postcss-preset-env')()]
279   }
280 }
281 }
282 ]
283
284 module.exports = {
285   entry: './src/js/index.js',
286   output: {
287     filename: 'js/built.[contenthash:10].js',
288     path: resolve(__dirname, 'build')
289   },
290   module: {
291     rules: [
292       {
293         // 在package.json中设置eslintConfig --> airbnb
294         test: /\.js$/,
295         exclude: /node_modules/,
296         // 优先执行
297         enforce: 'pre',
298         loader: 'eslint-loader', // eslint 语法检查
299         options: {
300           fix: true
301         }
302       },
303       {
304         // 以下loader只会匹配一个，解决每个文件都会被全部loader过一遍（相当于找到直接
305         // break掉）
306         // 不能有两个配置处理同一类型的文件
307         oneOf: [
308           {
309             test: /\.css$/,
310             use: [...commonCssloader]
311           },
312           {
313             test: /\.less$/,
314             use: [...commonCssloader, 'less-loader']
315           },
316           /*
317           正常来讲，一个文件只能被一个loader处理。
318           当一个文件要被多个loader处理，那么一定要指定loader执行的先后顺序
319           */
320           {
321             // js兼容性处理
322             test: /\.js$/,
323             exclude: /node_modules/,
324             loader: 'babel-loader',
325             options: {
326               presets: [
327                 [
328                   '@babel/preset-env',
329                   {

```

```

330         useBuiltIns: 'usage',
331         corejs: {
332             version: 3
333         },
334         targets: {
335             chrome: '60',
336             firefox: '50',
337             ie: '9',
338             safari: '10'
339         }
340     },
341 ]
342 ],
343 // 开启babel缓存, 第二次构建时, 会读取之前的缓存
344 cacheDirectory: true
345 }
346 },
347 {
348     // 处理图片资源
349     test: /\. (jpg|png|gif|jpeg)$/i,
350     type: 'asset',
351     generator: {
352         filename: '[name]_[contenthash:6][ext]'
353     },
354     parser: {
355         dataUrlCondition: {
356             maxSize: 10 * 1024
357         }
358     }
359 },
360 {
361     // 处理html 中的图片资源
362     test: /\.html$/,
363     loader: 'html-loader'
364 },
365 {
366     // 处理其他资源
367     exclude: /\. (js|css|less|html|jpg|png|gif|jpeg)$/,
368     type: 'asset/resource',
369     generator: {
370         filename: 'media/[name]_[contenthash:6][ext]'
371     }
372 }
373 ]
374 }
375 ]
376 },
377 plugins: [
378     // 打包html资源
379     new HtmlWebpackPlugin({
380         template: './src/index.html',
381         // 压缩html
382         minify: {
383             collapseWhitespace: true,
384             removeComments: true
385         }
386     }),
387     // 提取css成单独文件

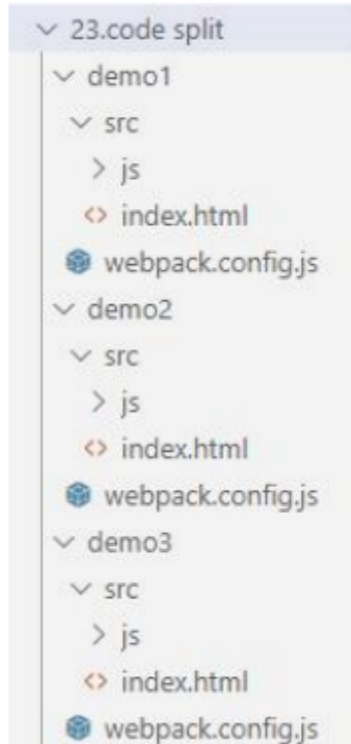
```

```
388     new MiniCssExtractPlugin({
389       filename: 'css/built.[contenthash:10].css'
390     }),
391     // 压缩 css
392     new CssMinimizerWebpackPlugin()
393   ],
394   mode: 'production'
395   // devtool: 'inline-source-map'
396 }
397 g|gif|jpeg)$/i,
398     type: 'asset',
399     generator: {
400       filename: '[name]_[contenthash:6][ext]'
401     },
402     parser: {
403       dataUrlCondition: {
404         maxSize: 10 * 1024
405       }
406     }
407   },
408   {
409     // 处理html 中的图片资源
410     test: /\.html$/,
411     loader: 'html-loader'
412   },
413   {
414     // 处理其他资源
415     exclude: /\. (js|css|less|html|jpg|png|gif|jpeg)$/,
416     type: 'asset/resource',
417     generator: {
418       filename: 'media/[name]_[contenthash:6][ext]'
419     }
420   }
421 ]
422 }
423 ]
424 },
425 plugins: [
426   // 打包html资源
427   new HtmlWebpackPlugin({
428     template: './src/index.html',
429     // 压缩html
430     minify: {
431       collapseWhitespace: true,
432       removeComments: true
433     }
434   }),
435   // 提取css成单独文件
436   new MiniCssExtractPlugin({
437     filename: 'css/built.[contenthash:10].css'
438   }),
439   // 压缩 css
440   new CssMinimizerWebpackPlugin()
441 ],
442 mode: 'production'
443 // devtool: 'inline-source-map'
444 }
445
```

3. 运行指令: `webpack`

5.7. code split (代码分割)

1. 创建文件



2. 修改配置文件

◦ 修改 demo1 配置文件

```
1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  // 定义nodejs环境变量: 决定使用browserslist的哪个环境
5  process.env.NODE_ENV = 'production'
6
7  module.exports = {
8    // 单入口
9    // entry: './src/js/index.js',
10   // 多入口: 有一个入口, 最终输出就有一个bundle
11   entry: {
12     index: './src/js/index.js',
13     test: './src/js/test.js'
14   },
15   output: {
16     // [name]: 取文件名
17     filename: 'js/[name].[contenthash:10].js',
18     path: resolve(__dirname, 'build')
19   },
20   plugins: [
21     // 打包html资源
22     new HtmlWebpackPlugin({
23       template: './src/index.html',
24       // 压缩html
```

```

25     minify: {
26         collapsewhitespace: true,
27         removeComments: true
28     }
29 })
30 ],
31 mode: 'production'
32 }

```

- 修改 demo2 配置文件（多入口模式最终方案）

```

1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  // 定义nodejs环境变量：决定使用browserslist的哪个环境
5  process.env.NODE_ENV = 'production'
6
7  module.exports = {
8      // 单入口
9      entry: './src/js/index.js',
10     // entry: {
11     //     index: './src/js/index.js',
12     //     test: './src/js/test.js'
13     // },
14     output: {
15         // [name]: 取文件名
16         filename: 'js/[name].[contenthash:10].js',
17         path: resolve(__dirname, 'build')
18     },
19     plugins: [
20         // 打包html资源
21         new HtmlWebpackPlugin({
22             template: './src/index.html',
23             // 压缩html
24             minify: {
25                 collapsewhitespace: true,
26                 removeComments: true
27             }
28         })
29     ],
30     /*
31     可以将node_module中代码单独打包成一个chunk最终输出
32     自动分析多入口文件chunk中，有没有公共的文件（）。如果有会打包成单独一个chunk，不
    重复打包
33     */
34     optimization: {
35         splitChunks: {
36             chunks: 'all'
37         }
38     },
39     mode: 'production'
40 }

```

- 修改 demo3 配置文件（单入口模式）
 - 通过 js 代码，让某个文件被单独打包成一个 chunk
 - 再结合 optimization 配置实现代码分割

```

1  /*
2     通过js代码，让某个文件单独打包成一个chunk
3     import动态导入语法：配合optimization配置，能将某个文件单独打包，实现代码分割
4  */
5  import(/* webpackChunkName: 'test' */ './test')
6    .then(({ mul, count }) => {
7      // 文件加载成功
8      // eslint-disable-next-line
9      console.log(mul(2, 5))
10   })
11   .catch(() => {
12     // eslint-disable-next-line
13     console.log('文件加载失败')
14   })
15

```

```

1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  /*
5     通过js代码，让某个文件单独打包成一个chunk
6     import动态导入语法：配合optimization配置，能将某个文件单独打包，实现代码分割
7  */
8
9  // 定义nodejs环境变量：决定使用browserslist的哪个环境
10 process.env.NODE_ENV = 'production'
11
12 module.exports = {
13   // 单入口
14   entry: './src/js/index.js',
15   output: {
16     // [name]：取文件名
17     filename: 'js/[name].[contenthash:10].js',
18     path: resolve(__dirname, 'build'),
19   },
20   plugins: [
21     // 打包html资源
22     new HtmlWebpackPlugin({
23       template: './src/index.html',
24       // 压缩html
25       minify: {
26         collapseWhitespace: true,
27         removeComments: true,
28       },
29     }),
30   ],
31   /*
32     可以将node_module中代码单独打包成一个chunk最终输出
33     自动分析多入口文件chunk中，有没有公共的文件。如果有会打包成单独一个chunk，不重复
34     打包
35  */
36   optimization: {
37     splitChunks: {
38       chunks: 'all',
39     },
40   },
41 }

```

```

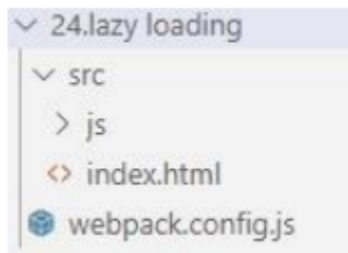
39   },
40   mode: 'production',
41 }

```

3. 运行指令: `webpack`

5.8. lazy loading (懒加载)

1. 创建文件



2. 修改配置文件

```

1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  module.exports = {
5    // 单入口
6    entry: './src/js/index.js',
7    output: {
8      filename: 'js/[name].[contenthash:10].js',
9      path: resolve(__dirname, 'build')
10   },
11   plugins: [
12     new HtmlWebpackPlugin({
13       template: './src/index.html',
14       minify: {
15         collapseWhitespace: true,
16         removeComments: true
17       }
18     })
19   ],
20   optimization: {
21     splitChunks: {
22       chunks: 'all'
23     }
24   },
25   mode: 'production'
26 }

```

3. index.js (懒加载和预加载代码)

```

1  console.log('index.js文件被加载了~')
2
3  import { mul } from './test'
4
5  document.getElementById('btn').onclick = function () {
6    // console.log(mul(4, 5)) // mul 是 test 模块中的一个乘法计算的函数方法

```



```

7 // 懒加载：当文件需要使用时才加载
8 // import(/* webpackChunkName: 'test' */'./test').then(({mul}) => {
9 //   console.log(mul(4, 5))
10 // })
11
12 // 预加载 prefetch --> webpackPrefetch: true
13 //   会在使用之前，提前加载 js 文件
14 //   正常加载可以认为是并行加载（同一时间加载多个文件 预加载 prefetch：等其他资源加
    载完毕，浏览器空闲了，再偷偷加载
15 //
16 import(/* webpackChunkName: 'test', webpackPrefetch: true
    */'./test').then(({mul}) => {
17   console.log(mul(4, 5))
18 })
19 }

```

4. 运行指令： `webpack`

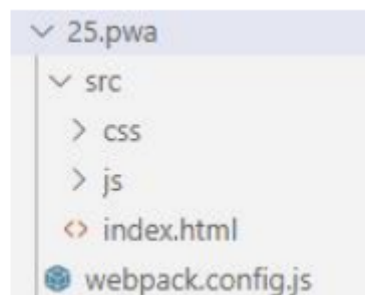
5. 懒加载和预加载的区别：（正常加载可以认为是并行加载）

- 懒加载：当文件需要使用时才加载
- 预加载（兼容性较差）：会在使用之前，提前加载 js 文件，等其他资源加载完毕，浏览器空闲了，再偷偷加载

5.9. PWA（离线加载）

- PWA：渐进式网络开发应用程序（离线加载）

1. 创建文件



2. 下载安装包

```
1 | npm install wordbox-webpack-plugin -D
```

3. 修改配置文件

```

1 const { resolve } = require('path')
2 const HtmlWebpackPlugin = require('html-webpack-plugin')
3 const MiniCssExtractPlugin = require('mini-css-extract-plugin')
4 const CssMinimizerWebpackPlugin = require('css-minimizer-webpack-plugin')
5 const workboxWebpackPlugin = require('workbox-webpack-plugin')
6
7 /*
8   PWA：渐进式网络开发应用程序（离线可访问）
9   workbox --> workbox-webpack-plugin
10 */
11

```

```

12 // 定义nodejs环境变量：决定使用browserslist的哪个环境
13 process.env.NODE_ENV = 'production'
14
15 // 复用loader
16 const commonCssloader = [
17   MiniCssExtractPlugin.loader,
18   'css-loader',
19   {
20     // css兼容性处理
21     // 还需要再 package.json中定义browserslist
22     loader: 'postcss-loader',
23     options: {
24       postcssOptions: {
25         plugins: [require('postcss-preset-env')()]
26       }
27     }
28   }
29 ]
30
31 module.exports = {
32   entry: './src/js/index.js',
33   output: {
34     filename: 'js/built.[contenthash:10].js',
35     path: resolve(__dirname, 'build')
36   },
37   module: {
38     rules: [
39       {
40         // 在package.json中设置eslintConfig --> airbnb
41         test: /\.js$/,
42         exclude: /node_modules/,
43         // 优先执行
44         enforce: 'pre',
45         loader: 'eslint-loader', // eslint 语法检查
46         options: {
47           fix: true
48         }
49       },
50       {
51         // 以下loader只会匹配一个，解决每个文件都会被全部loader过一遍（相当于找到直接
52         // break掉）
53         // 不能有两个配置处理同一类型的文件
54         oneOf: [
55           {
56             test: /\.css$/,
57             use: [...commonCssloader]
58           },
59           {
60             test: /\.less$/,
61             use: [...commonCssloader, 'less-loader']
62           },
63           /*
64             正常来讲，一个文件只能被一个loader处理。
65             当一个文件要被多个loader处理，那么一定要指定loader执行的先后顺序
66             */
67           {
68             // js兼容性处理

```

```

69     test: /\.js$/,
70     exclude: /node_modules/,
71     loader: 'babel-loader',
72     options: {
73       presets: [
74         [
75           '@babel/preset-env',
76           {
77             useBuiltIns: 'usage',
78             corejs: {
79               version: 3
80             },
81             targets: {
82               chrome: '60',
83               firefox: '50',
84               ie: '9',
85               safari: '10'
86             }
87           }
88         ]
89       ],
90       // 开启babel缓存, 第二次构建时, 会读取之前的缓存
91       cacheDirectory: true
92     }
93   },
94   {
95     // 处理图片资源
96     test: /\. (jpg|png|gif|jpeg)$/i,
97     type: 'asset',
98     generator: {
99       filename: '[name]_[contenthash:6][ext]'
100     },
101     parser: {
102       dataUrlCondition: {
103         maxSize: 10 * 1024
104       }
105     }
106   },
107   {
108     // 处理html 中的图片资源
109     test: /\.html$/,
110     loader: 'html-loader'
111   },
112   {
113     // 处理其他资源
114     exclude: /\. (js|css|less|html|jpg|png|gif|jpeg)$/i,
115     type: 'asset/resource',
116     generator: {
117       filename: 'media/[name]_[contenthash:6][ext]'
118     }
119   }
120 ]
121 }
122 ]
123 },
124 plugins: [
125   // 打包html资源
126   new HtmlWebpackPlugin({

```

```

127     template: './src/index.html',
128     // 压缩html
129     minify: {
130         collapseWhitespace: true,
131         removeComments: true
132     }
133   }),
134   // 提取css成单独文件
135   new MiniCssExtractPlugin({
136     filename: 'css/built.[contenthash:10].css'
137   }),
138   // 压缩 css
139   new CssMinimizerWebpackPlugin(),
140   new WorkboxWebpackPlugin.GenerateSW({
141     /*
142     1. 帮助serviceworker快速启动
143     2. 删除旧的 serviceworker
144
145     生成一个 serviceworker 配置文件，并在入口文件中注册serviceworker
146     */
147     clientsClaim: true,
148     skipWaiting: true
149   })
150 ],
151 mode: 'production'
152 // devtool: 'inline-source-map'
153 }
154

```

4. 入口 index.js 中

```

1  import { mul } from './test'
2  import './css/index.css'
3
4  function sum(...args) {
5    return args.reduce((p, c) => p + c, 0)
6  }
7
8
9  /*
10   1. eslint不认识 window、navigator 全局变量
11     解决: 需要修改 package.json 中 eslintConfig 配置
12     "env": {
13       "browser": true // 支持浏览器端全局变量，如果要支持node则改成"node": true
14     }
15   2. serviceworker(SW) 代码必须运行在服务器上
16     --> nodejs
17     -->
18     npm i server -g
19     server -s build 启动服务器，将 build 目录下所有资源作为静态资源暴露出去
20   */
21   // 注册 serviceworker
22   // 处理兼容性问题
23   if ('serviceWorker' in navigator) {
24     window.addEventListener('load', () => {
25       navigator.serviceWorker.register('/service-worker.js')
26       .then(() => {

```

```

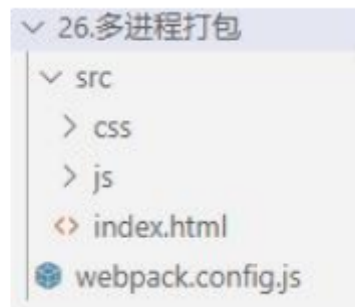
27     console.log('SW注册成功了')
28   })
29   .catch(() => {
30     console.log('SW注册失败了')
31   })
32 })
33 }

```

5. 运行指令: `webpack`

5.10. 多进程打包

1. 创建文件



2. 下载安装包

```
1 | npm install thread-loader -D
```

3. 修改配置文件

```

1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3  const MiniCssExtractPlugin = require('mini-css-extract-plugin')
4  const CssMinimizerWebpackPlugin = require('css-minimizer-webpack-plugin')
5  const WorkboxWebpackPlugin = require('workbox-webpack-plugin')
6
7  /*
8   PWA: 渐进式网络开发应用程序（离线可访问）
9   workbox --> workbox-webpack-plugin
10 */
11
12 // 定义nodejs环境变量：决定使用browserslist的哪个环境
13 process.env.NODE_ENV = 'production'
14
15 // 复用loader
16 const commonCssLoader = [
17   MiniCssExtractPlugin.loader,
18   'css-loader',
19   {
20     // css兼容性处理
21     // 还需要再 package.json中定义browserslist
22     loader: 'postcss-loader',
23     options: {
24       postcssOptions: {
25         plugins: [require('postcss-preset-env')()],

```

```

26     },
27   },
28 },
29 ]
30
31 module.exports = {
32   entry: './src/js/index.js',
33   output: {
34     filename: 'js/built.[contenthash:10].js',
35     path: resolve(__dirname, 'build'),
36   },
37   module: {
38     rules: [
39       {
40         // 在package.json中设置eslintConfig --> airbnb
41         test: /\.js$/,
42         exclude: /node_modules/,
43         // 优先执行
44         enforce: 'pre',
45         loader: 'eslint-loader', // eslint 语法检查
46         options: {
47           fix: true,
48         },
49       },
50       {
51         // 以下loader只会匹配一个，解决每个文件都会被全部loader过一遍（相当于找到直接
52         // break掉）
53         // 不能有两个配置处理同一类型的文件
54         oneOf: [
55           {
56             test: /\.css$/,
57             use: [...commonCssloader],
58           },
59           {
60             test: /\.less$/,
61             use: [...commonCssloader, 'less-loader'],
62           },
63           /*
64             正常来讲，一个文件只能被一个loader处理。
65             当一个文件要被多个loader处理，那么一定要指定loader执行的先后顺序
66             */
67           {
68             // js兼容性处理
69             test: /\.js$/,
70             exclude: /node_modules/,
71             use: [
72               /*
73                 开启多进程打包
74                 进程启动大概为600ms，进程通信也有开销。
75                 只有工作消耗时间比较长，才需要多进程打包
76               */
77               {
78                 loader: 'thread-loader',
79                 options: {
80                   workers: 2 // 进程2个
81                 }
82               },

```

```

83     {
84         loader: 'babel-loader',
85         options: {
86             presets: [
87                 [
88                     '@babel/preset-env',
89                     {
90                         useBuiltIns: 'usage',
91                         corejs: {
92                             version: 3,
93                         },
94                         targets: {
95                             chrome: '60',
96                             firefox: '50',
97                             ie: '9',
98                             safari: '10',
99                         },
100                     },
101                 ],
102             ],
103             // 开启babel缓存，第二次构建时，会读取之前的缓存
104             cacheDirectory: true,
105         },
106     },
107 ],
108 },
109 {
110     // 处理图片资源
111     test: /\. (jpg|png|gif|jpeg) $/i,
112     type: 'asset',
113     generator: {
114         filename: '[name]_[contenthash:6][ext]',
115     },
116     parser: {
117         dataUrlCondition: {
118             maxSize: 10 * 1024,
119         },
120     },
121 },
122 {
123     // 处理html 中的图片资源
124     test: /\.html$/,
125     loader: 'html-loader',
126 },
127 {
128     // 处理其他资源
129     exclude: /\. (js|css|less|html|jpg|png|gif|jpeg) $/,
130     type: 'asset/resource',
131     generator: {
132         filename: 'media/[name]_[contenthash:6][ext]',
133     },
134 },
135 ],
136 },
137 ],
138 },
139 plugins: [
140     // 打包html资源

```

```

141     new HtmlWebpackPlugin({
142       template: './src/index.html',
143       // 压缩html
144       minify: {
145         collapseWhitespace: true,
146         removeComments: true,
147       },
148     }),
149     // 提取css成单独文件
150     new MiniCssExtractPlugin({
151       filename: 'css/built.[contenthash:10].css',
152     }),
153     // 压缩 css
154     new CssMinimizerWebpackPlugin(),
155     new WorkboxWebpackPlugin.GenerateSW({
156       /*
157        1. 帮助serviceworker快速启动
158        2. 删除旧的 serviceworker
159
160        生成一个 serviceworker 配置文件，并在入口文件中注册serviceworker
161       */
162       clientsClaim: true,
163       skipWaiting: true,
164     }),
165   ],
166   mode: 'production',
167   // devtool: 'inline-source-map'
168 }
169

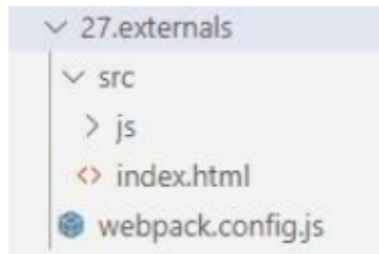
```

4. 运行指令: `webpack`

5.11. externals (防止某些包打包至bundle中)

- 规定某些包，不打包进来，而是通过链接引入
- 在需要用到的地方需要引入 (index.html 中通过链接引入)

1. 创建文件



2. 修改配置文件

```

1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  module.exports = {
5    entry: './src/index.js',
6    output: {
7      filename: 'js/built.js',

```



```

8     path: resolve(__dirname, 'build')
9   },
10  plugins: [
11    new HtmlWebpackPlugin({
12      template: './src/index.html'
13    })
14  ],
15  mode: 'production',
16  externals: {
17    // 拒绝 jQuery 被打包进来
18    // 忽略库名: 'npm包名'
19    jquery: 'jQuery'
20  }
21 }

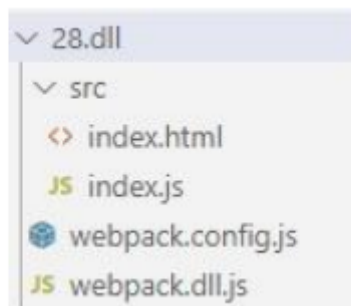
```

3. 运行指令: `webpack`

5.12. dll (动态链接库)

说明: dll 类似 externals 会指示 webpack 哪些库不参与打包, 不同的是 dll 会单独的对某些库进行单独打包, 将多个库打包成一个 chunk, dll 需要打包一次, 之后就不需要打包了, externals 是始终不打包

1. 创建文件



2. 写 webpack.dll.js 文件

```

1  /*
2   使用dll技术, 对某些库 (第三方库: jQuery、react、vue) 进行单独打包
3   当你运行 webpack 时, 默认查找 webpack.config.js 配置文件
4   需求: 需要运行 webpack.dll.js 文件
5   --> webpack --config webpack.dll.js
6  */
7  const { resolve } = require('path')
8  const webpack = require('webpack')
9
10 module.exports = {
11   entry: {
12     // 最终打包生成的[name] --> jquery
13     // ['jquery'] --> 要打包的库是jquery
14     jquery: ['jquery'],
15   },
16   output: {
17     filename: '[name].js',
18     path: resolve(__dirname, 'dll'),
19     library: '[name]_[hash]', // 打包的库向外暴露出去的内容叫什么名字
20   },
21   plugins: [

```

```

22 // 打包生成一个 manifest.json --> 提供jquery映射
23 new webpack.DllPlugin({
24   name: '[name]_[hash]', // 映射库的暴露的内容名称
25   path: resolve(__dirname, 'dll/manifest.json'), // 输出文件路径
26 },
27 ],
28 mode: 'production',
29 }

```

3. 运行指令: `webpack --config webpack.dll.js` --> 打包指定库并生成 dll/manifest.json

4. 修改配置文件

```

1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3  const webpack = require('webpack')
4  const AddAssetHtmlWebpackPlugin = require('add-asset-html-webpack-plugin')
5
6  module.exports = {
7    entry: './src/index.js',
8    output: {
9      filename: './src/index.js',
10     path: resolve(__dirname, 'build')
11   },
12   plugins: [
13     new HtmlWebpackPlugin({
14       template: './src/index.html'
15     }),
16     // 告诉 webpack 哪些库不参与打包, 同时使用时的名称也得变~
17     new webpack.DllReferencePlugin({
18       manifest: resolve(__dirname, 'dll/manifest.json')
19     }),
20     // 将某个文件打包输出出去, 并在 html 中自动引入该资源
21     new AddAssetHtmlWebpackPlugin({
22       filepath: resolve(__dirname, 'dll/jquery.js')
23     })
24     mode: 'production'
25   ]
26 }

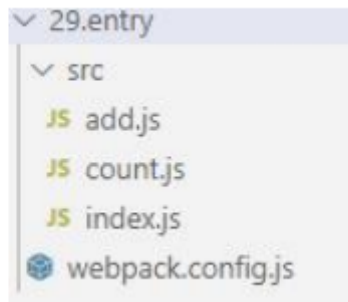
```

5. 运行指令: `webpack`

6. 第六章: webpack 配置详情

6.1. entry

1. 创建文件



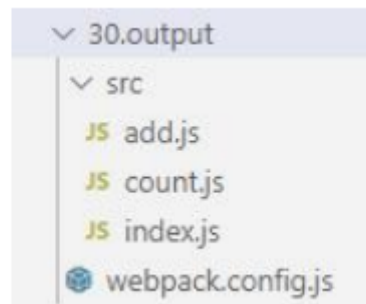
2. 修改配置文件

```
1  const HtmlWebpackPlugin = require('html-webpack-plugin')
2  const { resolve } = require('path')
3
4  /*
5   entry: 入口起点
6   1. string --> './src/index.js'
7       单入口
8       打包形成一个chunk。输出一个bundle文件
9       此时chunk的名称默认是 main
10  2. array --> ['./src/index.js', './src/add.js']
11      多入口
12      所有入口文件最终只会形成一个chunk，输出出去只有一个bundle
13      --> 只有在HMR功能中让html热更新生效才使用
14  3. object --> {key: value, key: value}
15      多入口
16      有几个文件就形成几个chunk，输出几个bundle
17      此时chunk名称是key
18
19      --> 特殊用法(dll中用到):
20      entry: {
21          所有入口文件最终只会形成一个chunk，输出出去只有一个bundle
22          // 把 './src/index.js', './src/count.js' 打包到index.js
23          中, './src/add.js' 单独打包
24          index: ['./src/index.js', './src/count.js'],
25          // 形成一个chunk，输出一个bundle
26          add: './src/add.js'
27      },
28  */
29  module.exports = {
30      entry: {
31          index: './src/index.js',
32          add: './src/add.js',
33      },
34      output: {
35          filename: '[name].js',
36          path: resolve(__dirname, 'build'),
37      },
38      module: {
39          rules: [],
40      },
41      plugins: [new HtmlWebpackPlugin()],
42      mode: 'development',
43  }
44
```

3. 运行指令: `webpack`

6.2. output

1. 创建文件



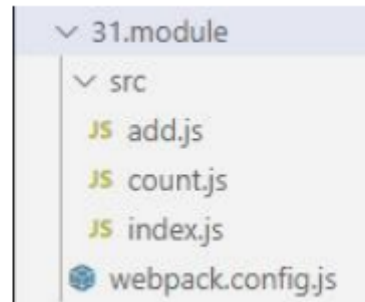
2. 修改配置文件

```
1  const HtmlWebpackPlugin = require('html-webpack-plugin')
2  const { resolve } = require('path')
3
4  module.exports = {
5    entry: './src/index.js',
6    output: {
7      // 文件名称 (指定目录+名称)
8      filename: 'js/[name].js',
9      // 输出文件目录 (将来所有资源输出的公共目录)
10     path: resolve(__dirname, 'build'),
11     environment: {
12       // 告诉webpack不使用箭头函数
13       arrowFunction: false,
14       // 告诉 webpack 不使用 const
15       const: false
16     },
17     // 所有资源引入公共路径前缀 --> 'imgs/a.jpg' --> '/imgs/a.jpg'
18     publicPath: '/',
19     // 非入口chunk的名称
20     chunkFilename: '[name].chunk.js',
21     // 整个库向外暴露的变量名, 结合dll使用
22     library: '[name]',
23     // libraryTarget: 'window', // 变量名添加到哪个上 browser
24     // libraryTarget: 'global', // 变量名添加到哪个上 node
25     libraryTarget: 'commonjs',
26   },
27   module: {
28     rules: [],
29   },
30   plugins: [new HtmlWebpackPlugin()],
31   mode: 'development',
32 }
```

3. 运行指令: `webpack`

6.3. module

1. 创建文件



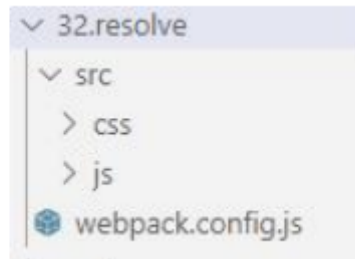
2. 修改配置文件

```
1  const HtmlWebpackPlugin = require('html-webpack-plugin')
2  const { resolve } = require('path')
3
4  module.exports = {
5    entry: './src/index.js',
6    output: {
7      filename: 'js/[name].js',
8      path: resolve(__dirname, 'build'),
9    },
10   module: {
11     rules: [
12       // loader 的配置
13       {
14         test: /\.css$/,
15         // 多个loader用use
16         use: ['style-loader', 'css-loader'],
17       },
18       {
19         test: /\.js$/,
20         // 排除node_modules下的js文件
21         exclude: /node_modules/,
22         // 只检查src下的js文件
23         include: resolve(__dirname, 'src'),
24         // 优先执行
25         enforce: 'pre',
26         // 延后执行
27         enforce: 'post',
28         // 单个loader用loader
29         loader: 'eslint-loader',
30         // loader的配置选项
31         options: {}
32       },
33       {
34         // 以下配置只会生效一个
35         oneOf: [],
36       },
37     ],
38   },
39   plugins: [new HtmlWebpackPlugin()],
40   mode: 'development',
41 }
```

3. 运行指令: `webpack`

6.4. resolve

1. 创建文件



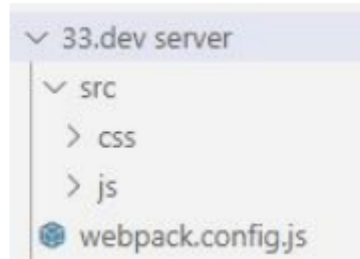
2. 修改配置文件

```
1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  module.exports = {
5    entry: './src/js/index.js',
6    output: {
7      filename: 'js/[name].js',
8      path: resolve(__dirname, 'build')
9    },
10   module: {
11     rules: [
12       {
13         test: /\.css$/,
14         use: ['style-loader', 'css-loader']
15       }
16     ]
17   },
18   plugins: [new HtmlWebpackPlugin()],
19   mode: 'development',
20   // 解析模块的规则
21   resolve: {
22     // 配置解析模块路径别名: 优点是简写路径, 缺点是路径没有提示
23     alias: {
24       $css: resolve(__dirname, 'src/css')
25     },
26     // 配置省略文件路径的后缀名
27     extensions: ['.js', 'json', '.jsx'],
28     // 告诉 webpack 解析模块去找哪个目录
29     modules: [resolve(__dirname, '../..../node_modules'), 'node_modules']
30   }
31 }
```

3. 运行指令: `webpack`

6.5. devServer

1. 创建文件



2. 修改配置文件

```
1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3
4  module.exports = {
5    entry: './src/js/index.js',
6    output: {
7      filename: 'js/[name].js',
8      path: resolve(__dirname, 'build')
9    },
10   module: {
11     rules: [
12       {
13         test: /\.css$/,
14         use: ['style-loader', 'css-loader']
15       }
16     ]
17   },
18   plugins: [new HtmlWebpackPlugin()],
19   mode: 'development',
20   // 解析模块的规则
21   resolve: {
22     // 配置解析模块路径别名: 优点是简写路径, 缺点是路径没有提示
23     alias: {
24       $css: resolve(__dirname, 'src/css')
25     },
26     // 配置省略文件路径的后缀名
27     extensions: ['.js', 'json', '.jsx', '.css'],
28     // 告诉 webpack 解析模块去找哪个目录
29     modules: [resolve(__dirname, '../..../node_modules'), 'node_modules']
30   },
31   devServer: {
32     // 运行代码的目录
33     contentBase: resolve(__dirname, 'build'),
34     // 监视 contentBase 目录下的所有文件, 一旦文件变化就会 reload
35     watchContentBase: true,
36     watchOptions: {
37       // 忽略文件
38       ignored: /node_modules/
39     },
40     // 启动 gzip 压缩
41     compress: true,
42     // 端口号
43     port: 5000,
```

```

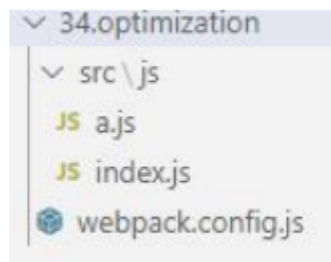
44 // 域名
45 host: 'localhost',
46 // 自动打开浏览器
47 open: true,
48 // 自动开启 HMR 功能
49 hot: true,
50 // 不要显示启动服务器日志信息
51 clientLogLevel: 'none',
52 // 除了一些基本启动信息以外，其他内容都不要提示
53 quiet: true,
54 // 如果出错了，不要全屏提示~
55 overlay: false,
56 // 服务器代理 --> 解决开发环境跨域问题
57 proxy: {
58   // 一旦 devServer(5000) 服务器接收到 /api/xxx 的请求，就会把请求转发到另外一个
    服务器(3000)
59   '/api': {
60     target: 'http://localhost:3000',
61     // 发送请求时，请求路径重写：将 /api/xxx --> /xxx (去掉/api)
62     pathRewrite: {
63       '^/api': ''
64     }
65   }
66 }
67 }
68 }

```

3. 运行指令： `webapck`

6.6. optimization

1. 创建文件



2. 修改配置文件

```

1  const { resolve } = require('path')
2  const HtmlWebpackPlugin = require('html-webpack-plugin')
3  // 版本到4.26以上之后，js的压缩不再用UglifyPlugin，而是用 TerserWebpackPlugin压缩
4  const TerserWebpackPlugin = require('terser-webpack-plugin')
5
6  module.exports = {
7    entry: './src/js/index.js',
8    output: {
9      filename: 'js/[name].[contenthash:10].js',
10     path: resolve(__dirname, 'build'),
11     chunkFilename: 'js/[name].[contenthash:10]_chunk.js'
12   },

```



```

13 module: {
14   rules: [
15     {
16       test: /\.css$/,
17       use: ['style-loader', 'css-loader']
18     }
19   ]
20 },
21 plugins: [new HtmlWebpackPlugin()],
22 mode: 'production',
23 resolve: {
24   alias: {
25     $css: resolve(__dirname, 'src/css')
26   },
27   extensions: ['.js', '.json', '.jsx', '.css'],
28   modules: [resolve(__dirname, '../..../node_modules'), 'node_modules']
29 },
30 optimization: {
31   splitChunks: {
32     chunks: 'all'
33     // 默认值，可以不写~
34     /* minSize: 30 * 1024, // 分割的chunk最小为30kb
35     maxSize: 0, // 最大没有限制
36     minChunks: 1, // 要提取的chunk最少被引用1次
37     maxAsyncRequest: 5, // 按需加载时并行加载的文件的最大数量
38     maxInitialRequest: 3, // 入口js文件最大并行请求数量
39     automaticNameDelimiter: '~', // 名称连接符
40     name: true, // 可以使用命名规则
41     cacheGroup: { // 分割chunk 的组
42       // node_modules 中的文件会被打包到 vendors 组的 chunk 中。 --
43       >vendors~xxx.js
44       // 满足上面规则，如：大小超过 30kb，至少被引用一次。
45       vendors: {
46         test: /[\\/]node_modules[\\/]$/,
47         // 优先级
48         priority: -10
49       },
50       default: {
51         minChunks: 2, // 要提取的chunk最少被引用2次
52         // 优先级
53         priority: -20,
54         // 如果当前要打包的模块，和之前已经被提取的模块时同一个，就会服用，而不是重新
55         打包模块
56         reuseExistingChunk: true
57       }
58     } */
59   },
60   // 将当前模块记录其他模块的 hash 单独打包为一个文件 runtime
61   // 解决：修改 a 文件导致 b 文件的 contenthash 变化
62   runtimeChunk: {
63     name: entrypoint => `runtime-${entrypoint.name}`
64   },
65   minimizer: [
66     // 配置生产环境的压缩方案：js 和 css
67     new TerserWebpackPlugin({
68       // 开启缓存
69       cache: true,
70       // 开启多进程打包

```

```
69         parallel: true,  
70         // 启动 source-map  
71         sourceMap: true  
72     })  
73 ]  
74 }  
75 }
```

7. 第七章：webpack5 介绍和使用

此版本重点关注以下内容：

- 通过持久缓存提高构建性能；
- 使用更好的算法和默认值来改善长期缓存；
- 通过更好的树摇（tree shaking）和代码生成来改善捆绑包大小；
- 清除怪异状态的内部解构，同时在 v4 中实现功能而不引入任何重大更改；
- 通过引入重大更改来为将来的功能做准备，以使我们能够尽可能长时间使用 v5。

7.1. 下载

```
1 | npm i webpack@next webpack-cli -D
```

7.2. 自动删除 Node.js Polyfills

早期，webpack 的目标是允许在浏览器中运行大多数 node.js 模块，但是模块格局发生了变化，许多模块用途现在主要是为前端目的而编写的。webpack <= 4 附带了许多 node.js 核心模块的 polyfill，一旦模块使用了任何核心模块（即 crypto 模块），这些模块就会自动应用。

尽管这使使用为 node.js 编写的模块变得容易，但它会将这些巨大的 polyfill 添加到包中。在许多情况下，这些 polyfill 是不必要的。

webpack 5 会自动停止填充这些核心模块，并专注于与前端兼容的模块。

迁移：

- 尽可能尝试使用与前端兼容的模块。
- 可以为 node.js 核心模块手动添加一个 polyfill。错误消息提示如何实现该目标。

7.3. Chunk 和 模块 ID

添加了用于长期缓存的新算法。在生产模式下默认情况下启用这些功能。

```
1 | chunkIds: "deterministic", moduleIds: "deterministic"
```

7.4. Chunk ID

你可以不使用 `import(/* webpackChunkName: "name" */ "module")` 在开发环境来为 chunk 命名，生产环境还是有必要的

webpack 内部有 chunk 命名规则，不再是以 id(0, 1, 2) 命名了

7.5. Tree Shaking

1. webpack 现在能够处理对嵌套模块的 tree shaking

```
1 // inner.js
2 export const a = 1
3 export const b = 2
4
5 // module.js
6 import * as inner from './inner'
7 export { inner }
8
9 // user.js
10 import * as module from './module'
11 console.log(module.inner.a)
```

在生产环境中，inner 模块暴露的 `b` 会被删除

2. webpack 现在能够处理好多个模块之前的关系

```
1 import { something } from './something'
2
3 function usingSomething() {
4   return something
5 }
6
7 export function test() {
8   return usingSomething()
9 }
```

当设置了 `"sideEffects": false` 时，一旦发现 `test` 方法没有使用，不但删除 `test`，还会删除 `"./something"`

3. webpack 现在能处理对 Commonjs 的 tree shaking

7.6. Output

webpack 4 默认只能输出 ES5 代码

webpack 5 开始新增一个属性 `output.ecmaVersion`，可以设置 ES5 和 ES6 / ES2015 代码。

如： `output.ecmaVersion: 2015`

7.7. SplitChunk

```
1 // webpack 4
2 minSize: 30000;
```

```
1 // webpack 5
2 minSize: {
3   javascript: 30000,
4   style: 50000,
5 }
```

7.8. Caching

```
1 // 配置缓存
2 cache: {
3   // 磁盘缓存
4   type: 'filesystem',
5   buildDependencies: {
6     // 当配置修改时，缓存失效
7     config: [__filename]
8   }
9 }
```

缓存将存储到 `node_modules/.cache/webpack`

7.9. 监视输出文件

之前 webpack 总是在第一次构建时输出全部文件，但是监视重新构建时会只更新修改的文件。

此次更新在第一次构建时会找到输出文件看是否有变化，从而决定要不要输出全部文件。

7.10. 默认值

- `entry: './src/index.js'`
- `output.path: path.resolve(__dirname, 'dist')`
- `output.filename: '[name].js'`

7.11. 更多内容

<https://github.com/webpack/changelog-v5>

7.12. webpack 5 处理图片文件

```
1 // 含转换base64的功能 "asset", (等价于 file + url 两个loader)
2 {
3   test: /\.?(jpe?g|png|gif|svg)$/,
4   type: 'asset',
5   generator: {
6     filename: 'img/[name]_[hash].[ext]',
7   },
8   parser: {
9     dataUrlCondition: {
10       maxSize: 100 * 1024 //(表示100kb以下的文件转换成base64编码)
11     }
12   }
13 }
```

7.13. 打包项目中的静态资源

资源模块类型(asset module type), 通过添加 4 种新的模块类型, 来替换之前使用的一些 loader:

- `asset/resource` 发送一个单独的文件并导出 URL。之前通过使用 `file-loader` 实现。
- `asset/inline` 导出一个资源的 data URI(base64格式)。之前通过使用 `url-loader` 实现。
- `asset/source` 导出资源的源代码。之前通过使用 `raw-loader` 实现。
- `asset` 在导出一个 data URI 和发送一个单独的文件之间自动选择。之前通过使用 `url-loader`, 并且配置资源体积限制实现。

7.13.1. asset/resource

- 配置loader

```
1 {
2   test: /\.?(png|jpe?g|gif)$/,
3   type: "asset/resource",
4 }
```

- 使用

```
1 // index.js
2 import reactUrl from '../imgs/react.jpg'
3 document.querySelector('.react').src = reactUrl
```

- 让图片输出到指定目录

```

1  output: {
2    ...,
3    assetModuleFilename: "images/[hash:8][ext]",
4  }
5  或:
6  {
7    test: /\. (png|jpe?g|gif|svg)$/ ,
8    type: "asset/resource",
9    // generator只适用于asset和asset/resource
10   generator: {
11     filename: 'imgs/[name][hash][ext]',
12   },
13 }

```

- 运行指令: `webpack`

7.13.2. asset/inline

- 配置 loader

```

1  {
2    test: /\.svg$/,
3    type: 'asset/inline'
4  }

```

- 使用
- 使用

```

1  // index.js
2  import wexin from '../imgs/微信.svg'
3  document.querySelector('.wexin').src = wexin

```

- 运行指令: `webpack`

7.13.3. asset/source

- 配置 loader

```

1  {
2    test: /\.txt/,
3    type: 'asset/source',
4  }

```

- 使用

```

1  //hello.txt
2  hello中的文本
3
4  // index.js
5  import text from '../assets/hello.txt' //text就是hello.txt的内容
6  document.querySelector('#txt').innerText = text

```

- 运行指令: `webpack`

7.13.4. asset/source

在asset/resource和asset/inline之间自动选择

- 配置 loader

```
1 {
2   test: /\. (png|jpe?g|gif|svg)$/,
3   type: "asset",
4   parser: {
5     dataUrlCondition: {
6       maxSize: 3 * 1024, // 小于3kb以下的图片会被打包成base64格式
7     },
8   },
9 }
```

- 使用

```
1 // index.js
2 import reactUrl from '../imgs/react.jpg'
3 document.querySelector('.react').src = reactUrl
```

- 让图片输出到指定目录

```
1 output: {
2   ...,
3   assetModuleFilename: "images/[hash:8][ext]",
4 }
5 或:
6 {
7   test: /\. (png|jpe?g|gif|svg)$/,
8   type: "asset",
9   // generator只适用于asset和asset/resource
10  generator: {
11     filename: 'imgs/[name][hash][ext]',
12   },
13 }
```

- 运行指令: `webpack`

