

# Webpack5 快速入门

---

## 基础配置

### • 前言

#### - 为什么需要打包工具?

开发时，我们会使用框架（React、Vue），ES6 模块化语法，Less/Sass 等 css 预处理器等语法进行开发。

这样的代码要想在浏览器运行必须经过编译成浏览器能识别的 JS、Css 等语法，才能运行。

所以我们需要打包工具帮我们做完这些事。

除此之外，打包工具还能压缩代码、做兼容性处理、提升代码性能等。

#### - 有哪些打包工具?

- Grunt
- Gulp
- Parcel
- Webpack
- Rollup
- Vite
- ...

目前市面上最流量的是 Webpack，所以我们主要以 Webpack 来介绍使用打包工具

### • 基本使用

**Webpack** 是一个静态资源打包工具。

它会以一个或多个文件作为打包的入口，将我们整个项目所有文件编译组合成一个或多个文件输出出去。

输出的文件就是编译好的文件，就可以在浏览器段运行了。

我们将 **Webpack** 输出的文件叫做 **bundle**。

## - 功能介绍

Webpack 本身功能是有限的:

- 开发模式: 仅能编译 JS 中的 ES Module 语法
- 生产模式: 能编译 JS 中的 ES Module 语法, 还能压缩 JS 代码

## - 开始使用

### 1. 资源目录

```
1  webpack_code # 项目根目录 (所有指令必须在这个目录运行)
2      └─ src # 项目源码目录
3          └─ js # js文件目录
4              └─ count.js
5              └─ sum.js
6          └─ main.js # 项目主文件
```

### 2. 创建文件

- count.js

```
1  export default function count(x, y) {
2      return x - y;
3  }
```

- sum.js

```
1  export default function sum( ...args) {
2      return args.reduce((p, c) => p + c, 0);
3  }
```

- main.js

```
1  import count from "./js/count";
2  import sum from "./js/sum";
3
4  console.log(count(2, 1));
5  console.log(sum(1, 2, 3, 4));
```

### 3. 下载依赖

打开终端，来到项目根目录。运行以下指令：

- 初始化 `package.json`

```
1 npm init -y
```

此时会生成一个基础的 `package.json` 文件。

需要注意的是 `package.json` 中 `name` 字段不能叫做 `webpack`，否则下一步会报错

- 下载依赖

```
1 npm i webpack webpack-cli -D
```

### 4. 启用 Webpack

- 开发模式

```
1 npx webpack ./src/main.js --mode=development
```

- 生产模式

```
1 npx webpack ./src/main.js --mode=production
```

`npx webpack`：是用来运行本地安装 `Webpack` 包的。

`./src/main.js`：指定 `Webpack` 从 `main.js` 文件开始打包，不但会打包 `main.js`，还会将其依赖也一起打包进来。

`--mode=xxx`：指定模式（环境）。

### 5. 观察输出文件

默认 `Webpack` 会将文件打包输出到 `dist` 目录下，我们查看 `dist` 目录下文件情况就好了。

### - 小结

`Webpack` 本身功能比较少，只能处理 `js` 资源，一旦遇到 `css` 等其他资源就会报错。

所以我们学习 `Webpack`，就是主要学习如何处理其他资源。

## • 基本配置

在开始使用 `Webpack` 之前，我们需要对 `Webpack` 的配置有一定的认识。

### - 5 大核心概念

#### 1. entry (入口)

指示 Webpack 从哪个文件开始打包

#### 2. output (输出)

指示 Webpack 打包完的文件输出到哪里去，如何命名等

#### 3. loader (加载器)

webpack 本身只能处理 js、json 等资源，其他资源需要借助 loader，Webpack 才能解析

#### 4. plugins (插件)

扩展 Webpack 的功能

#### 5. mode (模式)

主要由两种模式：

- 开发模式：development
- 生产模式：production

### - 准备 Webpack 配置文件

在项目根目录下新建文件：`webpack.config.js`

```
1  module.exports = {
2    // 入口
3    entry: "", // 相对路径
4    // 输出
5    output: {
6      // 文件的输出路径
7      path: "", // 绝对路径
8      // 文件名
9      filename: "",
10   },
11   // 加载器
12   module: {
13     rules: [
14       // loader的配置
15     ],
16   },
17   // 插件
18   plugins: [
19     // 插件的配置
20   ],
```

```
21     // 模式
22     mode: "",
23 };
```

Webpack 是基于 Node.js 运行的，所以采用 Common.js 模块化规范

## - 修改配置文件

### 1. 配置文件

```
1  // Node.js的核心模块，专门用来处理文件路径
2  const path = require("path");
3
4  module.exports = {
5      // 入口
6      // 相对路径和绝对路径都行
7      entry: "./src/main.js",
8      // 输出
9      output: {
10         // path: 文件输出目录，必须是绝对路径
11         // path.resolve()方法返回一个绝对路径
12         // __dirname 当前文件的文件夹绝对路径
13         path: path.resolve(__dirname, "dist"),
14         // filename: 输出文件名
15         filename: "main.js",
16     },
17     // 加载器
18     module: {
19         rules: [],
20     },
21     // 插件
22     plugins: [],
23     // 模式
24     mode: "development", // 开发模式
25 };
```

### 2. 运行指令

```
1  npx webpack
```

此时功能和之前一样，也不能处理样式资源

## - 小结

Webpack 将来都通过 `webpack.config.js` 文件进行配置，来增强 Webpack 的功能

我们后面会以两个模式来分别搭建 Webpack 的配置，先进行开发模式，再完成生产模式

## • 开发模式介绍

开发模式顾名思义就是我们开发代码时使用的模式。

这个模式下我们主要做两件事：

1. 编译代码，使浏览器能识别运行

开发时我们有样式资源、字体图标、图片资源、html 资源等，webpack 默认都不能处理这些资源，所以我们要加载配置来编译这些资源

2. 代码质量检查，树立代码规范

提前检查代码的一些隐患，让代码运行时能更加健壮。

提前检查代码规范和格式，统一团队编码风格，让代码更优雅美观。

## • 处理样式资源

本章节我们学习使用 Webpack 如何处理 Css、Less、Sass、Scss、Styl 样式资源

## - 介绍

Webpack 本身是不能识别样式资源的，所以我们需要借助 Loader 来帮助 Webpack 解析样式资源

我们找 Loader 都应该去官方文档中找到对应的 Loader，然后使用

官方文档找不到的话，可以从社区 Github 中搜索查询

Webpack 官方 Loader 文档：<https://webpack.docschina.org/loaders/>

## - 处理 Css 资源

### 1. 下载包

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

注意：需要下载两个 loader

### 2. 功能介绍

- `css-loader`：负责将 Css 文件编译成 Webpack 能识别的模块
- `style-loader`：会动态创建一个 Style 标签，里面放置 Webpack 中 Css 模块内容

此时样式就会以 Style 标签的形式在页面上生效

### 3. 配置

```
1  const path = require("path");
2
3  module.exports = {
4    entry: "./src/main.js",
5    output: {
6      path: path.resolve(__dirname, "dist"),
7      filename: "main.js",
8    },
9    module: {
10     rules: [
11       {
12         // 用来匹配 .css 结尾的文件
13         test: /\.css$/,
14         // use 数组里面 Loader 执行顺序是从右到左
15         use: ["style-loader", "css-loader"],
16       },
17     ],
18   },
19   plugins: [],
20   mode: "development",
21 };
```

### 4. 添加 Css 资源

- src/css/index.css

```
1  .box1 {
2    width: 100px;
3    height: 100px;
4    background-color: pink;
5  }
```

- src/main.js

```
1  import count from "./js/count";
2  import sum from "./js/sum";
3  // 引入 Css 资源, Webpack才会对其打包
4  import "./css/index.css";
5
6  console.log(count(2, 1));
7  console.log(sum(1, 2, 3, 4));
```

- public/index.html

```

1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>webpack5</title>
8    </head>
9    <body>
10     <h1>Hello Webpack5</h1>
11     <!-- 准备一个使用样式的 DOM 容器 -->
12     <div class="box1"></div>
13     <!-- 引入打包后的js文件, 才能看到效果 -->
14     <script src="../dist/main.js"></script>
15   </body>
16 </html>

```

## 5. 运行指令

```
1  npx webpack
```

打开 index.html 页面查看效果

## - 处理 Less 资源

### 1. 下载包

```
1  npm i less less-loader -D
```

### 2. 功能介绍

- **less-loader** : 负责将 Less 文件编译成 Css 文件

### 3. 配置

```

1  const path = require("path");
2
3  module.exports = {
4    entry: "./src/main.js",
5    output: {
6      path: path.resolve(__dirname, "dist"),
7      filename: "main.js",
8    },
9    module: {
10     rules: [
11       {
12         // 用来匹配 .css 结尾的文件

```



```

13         test: /\.css$/,
14         // use 数组里面 Loader 执行顺序是从右到左
15         use: ["style-loader", "css-loader"],
16     },
17     {
18         test: /\.less$/,
19         use: ["style-loader", "css-loader", "less-loader"],
20     },
21 ],
22 },
23 plugins: [],
24 mode: "development",
25 };

```

#### 4. 添加 Less 资源

- src/less/index.less

```

1  .box2 {
2      width: 100px;
3      height: 100px;
4      background-color: deeppink;
5  }

```

- src/main.js

```

1  import count from "./js/count";
2  import sum from "./js/sum";
3  // 引入资源, Webpack才会对其打包
4  import "./css/index.css";
5  import "./less/index.less";
6
7  console.log(count(2, 1));
8  console.log(sum(1, 2, 3, 4));

```

- public/index.html

```

1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="UTF-8" />
5          <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6          <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7          <title>webpack5</title>
8      </head>
9      <body>

```

```
10     <h1>Hello Webpack5</h1>
11     <div class="box1"></div>
12     <div class="box2"></div>
13     <script src="../dist/main.js"></script>
14   </body>
15 </html>
```

## 5. 运行指令

```
1  npx webpack
```

打开 index.html 页面查看效果

## - 处理 Sass 和 Scss 资源

### 1. 下载包

```
1  npm i sass-loader sass -D
```

注意：需要下载两个

### 2. 功能介绍

- `sass-loader`：负责将 Sass 文件编译成 css 文件
- `sass`：`sass-loader` 依赖 `sass` 进行编译

### 3. 配置

```
1  const path = require("path");
2
3  module.exports = {
4    entry: "./src/main.js",
5    output: {
6      path: path.resolve(__dirname, "dist"),
7      filename: "main.js",
8    },
9    module: {
10     rules: [
11       {
12         // 用来匹配 .css 结尾的文件
13         test: /\.css$/,
14         // use 数组里面 Loader 执行顺序是从右到左
15         use: ["style-loader", "css-loader"],
16       },
17       {
18         test: /\.less$/,
19         use: ["style-loader", "css-loader", "less-loader"],
```

```

20     },
21     {
22       test: /\.s[ac]ss$/,
23       use: ["style-loader", "css-loader", "sass-loader"],
24     },
25   ],
26 },
27 plugins: [],
28 mode: "development",
29 };

```

#### 4. 添加 Sass 资源

- src/sass/index.sass

```

1  /* 可以省略大括号和分号 */
2  .box3
3    width: 100px
4    height: 100px
5    background-color: hotpink

```

- src/sass/index.scss

```

1  .box4 {
2    width: 100px;
3    height: 100px;
4    background-color: lightpink;
5  }

```

- src/main.js

```

1  import count from "./js/count";
2  import sum from "./js/sum";
3  // 引入资源, Webpack才会对其打包
4  import "./css/index.css";
5  import "./less/index.less";
6  import "./sass/index.sass";
7  import "./sass/index.scss";
8
9  console.log(count(2, 1));
10 console.log(sum(1, 2, 3, 4));

```

- public/index.html

```

1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>webpack5</title>
8    </head>
9    <body>
10     <h1>Hello Webpack5</h1>
11     <div class="box1"></div>
12     <div class="box2"></div>
13     <div class="box3"></div>
14     <div class="box4"></div>
15     <script src="../dist/main.js"></script>
16   </body>
17 </html>

```

## 5. 运行指令

```
1  npx webpack
```

打开 index.html 页面查看效果

## - 处理 Styl 资源

### 1. 下载包

```
1  npm i stylus-loader -D
```

### 2. 功能介绍

- **stylus-loader** : 负责将 Styl 文件编译成 Css 文件

### 3. 配置

```

1  const path = require("path");
2
3  module.exports = {
4    entry: "./src/main.js",
5    output: {
6      path: path.resolve(__dirname, "dist"),
7      filename: "main.js",
8    },
9    module: {
10      rules: [
11        {

```

```

12         // 用来匹配 .css 结尾的文件
13         test: /\.css$/,
14         // use 数组里面 Loader 执行顺序是从右到左
15         use: ["style-loader", "css-loader"],
16     },
17     {
18         test: /\.less$/,
19         use: ["style-loader", "css-loader", "less-loader"],
20     },
21     {
22         test: /\.s[ac]ss$/,
23         use: ["style-loader", "css-loader", "sass-loader"],
24     },
25     {
26         test: /\.styl$/,
27         use: ["style-loader", "css-loader", "stylus-loader"],
28     },
29 ],
30 },
31 plugins: [],
32 mode: "development",
33 };

```

#### 4. 添加 Styl 资源

- src/styl/index.styl

```

1  /* 可以省略大括号、分号、冒号 */
2  .box
3      width 100px
4      height 100px
5      background-color pink

```

- src/main.js

```

1  import { add } from "./math";
2  import count from "./js/count";
3  import sum from "./js/sum";
4  // 引入资源, Webpack才会对其打包
5  import "./css/index.css";
6  import "./less/index.less";
7  import "./sass/index.sass";
8  import "./sass/index.scss";
9  import "./styl/index.styl";
10
11  console.log(count(2, 1));
12  console.log(sum(1, 2, 3, 4));

```

- public/index.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>webpack5</title>
8    </head>
9    <body>
10     <h1>Hello Webpack5</h1>
11     <!-- 准备一个使用样式的 DOM 容器 -->
12     <div class="box1"></div>
13     <div class="box2"></div>
14     <div class="box3"></div>
15     <div class="box4"></div>
16     <div class="box5"></div>
17     <script src=" ../dist/main.js"></script>
18   </body>
19 </html>
```

## 5. 运行指令

```
1  npx webpack
```

打开 index.html 页面查看效果

## • 处理图片资源

过去在 Webpack4 时，我们处理图片资源通过 `file-loader` 和 `url-loader` 进行处理

现在 Webpack5 已经将两个 Loader 功能内置到 Webpack 里了，我们只需要简单配置即可处理图片资源

### — 1. 配置

```
1  const path = require("path");
2
3  module.exports = {
4    entry: "./src/main.js",
5    output: {
6      path: path.resolve(__dirname, "dist"),
7      filename: "main.js",
8    },
9    module: {
```

```

10     rules: [
11         {
12             // 用来匹配 .css 结尾的文件
13             test: /\.css$/,
14             // use 数组里面 Loader 执行顺序是从右到左
15             use: ["style-loader", "css-loader"],
16         },
17         {
18             test: /\.less$/,
19             use: ["style-loader", "css-loader", "less-loader"],
20         },
21         {
22             test: /\.s[ac]ss$/,
23             use: ["style-loader", "css-loader", "sass-loader"],
24         },
25         {
26             test: /\.styl$/,
27             use: ["style-loader", "css-loader", "stylus-loader"],
28         },
29         {
30             test: /\.(png|jpe?g|gif|webp)$/,
31             type: "asset",
32         },
33     ],
34 },
35 plugins: [],
36 mode: "development",
37 };

```

## 2. 添加图片资源

- src/images/1.jpeg
- src/images/2.png
- src/images/3.gif

## 3. 使用图片资源

- src/less/index.less

```

1  .box2 {
2      width: 100px;
3      height: 100px;
4      background-image: url("../images/1.jpeg");
5      background-size: cover;
6  }

```

- src/sass/index.sass

```
1  .box3
2    width: 100px
3    height: 100px
4    background-image: url("../images/2.png")
5    background-size: cover
```

- src/styl/index.styl

```
1  .box5
2    width 100px
3    height 100px
4    background-image url("../images/3.gif")
5    background-size cover
```

## 4. 运行指令

```
1  npx webpack
```

打开 index.html 页面查看效果

## 5. 输出资源情况

此时如果查看 dist 目录的话，会发现多了三张图片资源

因为 Webpack 会将所有打包好的资源输出到 dist 目录下

- 为什么样式资源没有呢？

因为经过 `style-loader` 的处理，样式资源打包到 main.js 里面去了，所以没有额外输出出来

## 6. 对图片资源进行优化

将小于某个大小的图片转化成 data URI 形式 (Base64 格式)

```
1  const path = require("path");
2
3  module.exports = {
4    entry: "./src/main.js",
5    output: {
6      path: path.resolve(__dirname, "dist"),
7      filename: "main.js",
8    },
9    module: {
10     rules: [
11       {
12         // 用来匹配 .css 结尾的文件
```



```

13         test: /\.css$/,
14         // use 数组里面 Loader 执行顺序是从右到左
15         use: ["style-loader", "css-loader"],
16     },
17     {
18         test: /\.less$/,
19         use: ["style-loader", "css-loader", "less-loader"],
20     },
21     {
22         test: /\.s[ac]ss$/,
23         use: ["style-loader", "css-loader", "sass-loader"],
24     },
25     {
26         test: /\.styl$/,
27         use: ["style-loader", "css-loader", "stylus-loader"],
28     },
29     {
30         test: /\.(png|jpe?g|gif|webp)$/,
31         type: "asset",
32         parser: {
33             dataUrlCondition: {
34                 maxSize: 10 * 1024 // 小于10kb的图片会被base64处理
35             }
36         }
37     },
38 ],
39 },
40 plugins: [],
41 mode: "development",
42 };

```

- 优点：减少请求数量
- 缺点：体积变得更大

此时输出的图片文件就只有两张，有一张图片以 data URI 形式内置到 js 中了  
(注意：需要将上次打包生成的文件清空，再重新打包才有效果)

## • 修改输出资源的名称和路径

### – 1. 配置

```

1  const path = require("path");
2
3  module.exports = {
4      entry: "./src/main.js",
5      output: {
6          path: path.resolve(__dirname, "dist"),
7          filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
8      },

```

```

9     module: {
10       rules: [
11         {
12           // 用来匹配 .css 结尾的文件
13           test: /\.css$/,
14           // use 数组里面 Loader 执行顺序是从右到左
15           use: ["style-loader", "css-loader"],
16         },
17         {
18           test: /\.less$/,
19           use: ["style-loader", "css-loader", "less-loader"],
20         },
21         {
22           test: /\.s[ac]ss$/,
23           use: ["style-loader", "css-loader", "sass-loader"],
24         },
25         {
26           test: /\.styl$/,
27           use: ["style-loader", "css-loader", "stylus-loader"],
28         },
29         {
30           test: /\.(png|jpe?g|gif|webp)$/,
31           type: "asset",
32           parser: {
33             dataUrlCondition: {
34               maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
35             },
36           },
37           generator: {
38             // 将图片文件输出到 static/imgs 目录中
39             // 将图片文件命名 [hash:8][ext][query]
40             // [hash:8]: hash值取8位
41             // [ext]: 使用之前的文件扩展名
42             // [query]: 添加之前的query参数
43             filename: "static/imgs/[hash:8][ext][query]",
44           },
45         },
46       ],
47     },
48     plugins: [],
49     mode: "development",
50   };

```

## 2. 修改 index.html

```

1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />

```

```

6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>webpack5</title>
8 </head>
9 <body>
10    <h1>Hello Webpack5</h1>
11    <div class="box1"></div>
12    <div class="box2"></div>
13    <div class="box3"></div>
14    <div class="box4"></div>
15    <div class="box5"></div>
16    <!-- 修改 js 资源路径 -->
17    <script src=" ../dist/static/js/main.js"></script>
18 </body>
19 </html>

```

### 3. 运行指令

```
1 npx webpack
```

- 此时输出文件目录：

(注意：需要将上次打包生成的文件清空，再重新打包才有效果)

```

1  └─ dist
2      └─ static
3          └─ imgs
4              └─ 7003350e.png
5          └─ js
6              └─ main.js

```

### • 自动清空上次打包资源

#### 1. 配置

```

1  const path = require("path");
2
3  module.exports = {
4      entry: "./src/main.js",
5      output: {
6          path: path.resolve(__dirname, "dist"),
7          filename: "static/js/main.js",
8          clean: true, // 自动将上次打包目录资源清空
9      },
10     module: {
11         rules: [
12             {

```

```

13     // 用来匹配 .css 结尾的文件
14     test: /\.css$/,
15     // use 数组里面 Loader 执行顺序是从右到左
16     use: ["style-loader", "css-loader"],
17   },
18   {
19     test: /\.less$/,
20     use: ["style-loader", "css-loader", "less-loader"],
21   },
22   {
23     test: /\.s[ac]ss$/,
24     use: ["style-loader", "css-loader", "sass-loader"],
25   },
26   {
27     test: /\.styl$/,
28     use: ["style-loader", "css-loader", "stylus-loader"],
29   },
30   {
31     test: /\.(png|jpe?g|gif|webp)$/,
32     type: "asset",
33     parser: {
34       dataUrlCondition: {
35         maxSize: 40 * 1024, // 小于40kb的图片会被base64处理
36       },
37     },
38     generator: {
39       // 将图片文件输出到 static/imgs 目录中
40       // 将图片文件命名 [hash:8][ext][query]
41       // [hash:8]: hash值取8位
42       // [ext]: 使用之前的文件扩展名
43       // [query]: 添加之前的query参数
44       filename: "static/imgs/[hash:8][ext][query]",
45     },
46   },
47 ],
48 },
49 plugins: [],
50 mode: "development",
51 };

```

## 2. 运行指令

```
1 npx webpack
```

观察 dist 目录资源情况

## • 处理字体图标资源

### – 1. 下载字体图标文件

1. 打开 [阿里巴巴矢量图标库](#)
2. 选择想要的图标添加到购物车，统一下载到本地

### – 2. 添加字体图标资源

- src/fonts/iconfont.ttf
- src/fonts/iconfont.woff
- src/fonts/iconfont.woff2
- src/css/iconfont.css
  - 注意字体文件路径需要修改
- src/main.js

```
1  import { add } from './math';
2  import count from './js/count';
3  import sum from './js/sum';
4  // 引入资源, Webpack才会对其打包
5  import './css/iconfont.css';
6  import './css/index.css';
7  import './less/index.less';
8  import './sass/index.sass';
9  import './sass/index.scss';
10 import './styl/index.styl';
11
12 console.log(count(2, 1));
13 console.log(sum(1, 2, 3, 4));
```

- public/index.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>webpack5</title>
8    </head>
9    <body>
10     <h1>Hello Webpack5</h1>
11     <div class="box1"></div>
12     <div class="box2"></div>
13     <div class="box3"></div>
14     <div class="box4"></div>
```

```

15     <div class="box5"></div>
16     <!-- 使用字体图标 -->
17     <i class="iconfont icon-arrow-down"></i>
18     <i class="iconfont icon-ashbin"></i>
19     <i class="iconfont icon-browse"></i>
20     <script src="../dist/static/js/main.js"></script>
21 </body>
22 </html>

```

### 3. 配置

```

1  const path = require("path");
2
3  module.exports = {
4    entry: "./src/main.js",
5    output: {
6      path: path.resolve(__dirname, "dist"),
7      filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
8      clean: true, // 自动将上次打包目录资源清空
9    },
10   module: {
11     rules: [
12       {
13         // 用来匹配 .css 结尾的文件
14         test: /\.css$/,
15         // use 数组里面 Loader 执行顺序是从右到左
16         use: ["style-loader", "css-loader"],
17       },
18       {
19         test: /\.less$/,
20         use: ["style-loader", "css-loader", "less-loader"],
21       },
22       {
23         test: /\.s[ac]ss$/,
24         use: ["style-loader", "css-loader", "sass-loader"],
25       },
26       {
27         test: /\.styl$/,
28         use: ["style-loader", "css-loader", "stylus-loader"],
29       },
30       {
31         test: /\.?(png|jpe?g|gif|webp)$/,
32         type: "asset",
33         parser: {
34           dataUrlCondition: {
35             maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
36           },
37         },
38       },
39       {
40         generator: {
41           // 将图片文件输出到 static/imgs 目录中

```

```

40         // 将图片文件命名 [hash:8][ext][query]
41         // [hash:8]: hash值取8位
42         // [ext]: 使用之前的文件扩展名
43         // [query]: 添加之前的query参数
44         filename: "static/imgs/[hash:8][ext][query]",
45     },
46 },
47 {
48     test: /\.?(ttf|woff2?)$/,
49     type: "asset/resource",
50     generator: {
51         filename: "static/media/[hash:8][ext][query]",
52     },
53 },
54 ],
55 },
56 plugins: [],
57 mode: "development",
58 };

```

`type: "asset/resource"` 和 `type: "asset"` 的区别:

1. `type: "asset/resource"` 相当于 `file-loader` , 将文件转化成 Webpack 能识别的资源, 其他不做处理
2. `type: "asset"` 相当于 `url-loader` , 将文件转化成 Webpack 能识别的资源, 同时小于某个大小的资源会处理成 data URI 形式

## 4. 运行指令

```
1 npx webpack
```

打开 index.html 页面查看效果

## • 处理其他资源

开发中可能还存在一些其他资源, 如音视频等, 我们也一起处理了

### 1. 配置

```

1  const path = require("path");
2
3  module.exports = {
4      entry: "./src/main.js",
5      output: {
6          path: path.resolve(__dirname, "dist"),
7          filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中

```

```
8     clean: true, // 自动将上次打包目录资源清空
9 },
10 module: {
11     rules: [
12         {
13             // 用来匹配 .css 结尾的文件
14             test: /\.css$/,
15             // use 数组里面 Loader 执行顺序是从右到左
16             use: ["style-loader", "css-loader"],
17         },
18         {
19             test: /\.less$/,
20             use: ["style-loader", "css-loader", "less-loader"],
21         },
22         {
23             test: /\.s[ac]ss$/,
24             use: ["style-loader", "css-loader", "sass-loader"],
25         },
26         {
27             test: /\.styl$/,
28             use: ["style-loader", "css-loader", "stylus-loader"],
29         },
30         {
31             test: /\.?(png|jpe?g|gif|webp)$/,
32             type: "asset",
33             parser: {
34                 dataUrlCondition: {
35                     maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
36                 },
37             },
38             generator: {
39                 // 将图片文件输出到 static/imgs 目录中
40                 // 将图片文件命名 [hash:8][ext][query]
41                 // [hash:8]: hash值取8位
42                 // [ext]: 使用之前的文件扩展名
43                 // [query]: 添加之前的query参数
44                 filename: "static/imgs/[hash:8][ext][query]",
45             },
46         },
47         {
48             test: /\.?(ttf|woff2?|map4|map3|avi)$/,
49             type: "asset/resource",
50             generator: {
51                 filename: "static/media/[hash:8][ext][query]",
52             },
53         },
54     ],
55 },
56 plugins: [],
57 mode: "development",
58 };
```



就是在处理字体图标资源基础上增加其他文件类型，统一处理即可

## 2. 运行指令

```
1 npx webpack
```

打开 index.html 页面查看效果

### • 处理 js 资源

有人可能会问，js 资源 Webpack 不能已经处理了吗，为什么我们还要处理呢？

原因是 Webpack 对 js 处理是有限的，只能编译 js 中 ES 模块化语法，不能编译其他语法，导致 js 不能在 IE 等浏览器运行，所以我们希望做一些兼容性处理。

其次开发中，团队对代码格式是有严格要求的，我们不能由肉眼去检测代码格式，需要使用专业的工具来检测。

- 针对 js 兼容性处理，我们使用 Babel 来完成
- 针对代码格式，我们使用 Eslint 来完成

我们先完成 Eslint，检测代码格式无误后，再由 Babel 做代码兼容性处理

## - Eslint

可组装的 JavaScript 和 JSX 检查工具。

这句话意思就是：它是用来检测 js 和 jsx 语法的工具，可以配置各项功能

我们使用 Eslint，关键是写 Eslint 配置文件，里面写上各种 rules 规则，将来运行 Eslint 时就会以写的规则对代码进行检查

### 1. 配置文件

配置文件由很多种写法：

- `.eslintrc.*`：新建文件，位于项目根目录
  - `.eslintrc`
  - `.eslintrc.js`
  - `.eslintrc.json`
  - 区别在于配置格式不一样
- `package.json` 中 `eslintConfig`：不需要创建文件，在原有文件基础上写

ESLint 会查找和自动读取它们，所以上述配置文件只需要存在一个即可

## 2. 具体配置

我们以 `.eslintrc.js` 配置文件为例：

```
1  module.exports = {
2    // 解析选项
3    parserOptions: {},
4    // 具体检查规则
5    rules: {},
6    // 继承其他规则
7    extends: [],
8    // ...
9    // 其他规则详见: https://eslint.bootcss.com/docs/user-guide/configuring
10 };
```

### 1. parserOptions 解析选项

```
1  parserOptions: {
2    ecmaVersion: 6, // ES 语法版本
3    sourceType: "module", // ES 模块化
4    ecmaFeatures: { // ES 其他特性
5      jsx: true // 如果是 React 项目, 就需要开启 jsx 语法
6    }
7  }
```

### 2. rules 具体规则

- `"off"` 或 `0` - 关闭规则
- `"warn"` 或 `1` - 开启规则, 使用警告级别的错误: `warn` (不会导致程序退出)
- `"error"` 或 `2` - 开启规则, 使用错误级别的错误: `error` (当被触发的时候, 程序会退出)

```
1  rules: {
2    semi: "error", // 禁止使用分号
3    'array-callback-return': 'warn', // 强制数组方法的回调函数中有 return 语句, 否则警告
4    'default-case': [
5      'warn', // 要求 switch 语句中有 default 分支, 否则警告
6      { commentPattern: '^no default$' } // 允许在最后注释 no default, 就不会有警告了
7    ],
8    eqeqeq: [
9      'warn', // 强制使用 === 和 !==, 否则警告
10     'smart' // https://eslint.bootcss.com/docs/rules/eqeqeq#smart 除了少数情况下不会有警告
11   ],
12 }
```

更多规则详见: [规则文档](#)

### 3. extends 继承

开发中一点点写 rules 规则太费劲了, 所以有更好的办法, 继承现有的规则。

现有以下较为有名的规则:

- [Eslint 官方的规则](#): `eslint:recommended`
- [Vue Cli 官方的规则](#): `plugin:vue/essential`
- [React Cli 官方的规则](#): `react-app`

```
1 // 例如在React项目中, 我们可以这样写配置
2 module.exports = {
3   extends: ["react-app"],
4   rules: {
5     // 我们的规则会覆盖掉react-app的规则
6     // 所以想要修改规则直接改就是了
7     eqeqeq: ["warn", "smart"],
8   },
9 };
```

### 3. 在 Webpack 中使用

#### 1. 下载包

```
1 npm i eslint-webpack-plugin eslint -D
```

#### 2. 定义 Eslint 配置文件

- `.eslintrc.js`

```
1 module.exports = {
2   // 继承 Eslint 规则
3   extends: ["eslint:recommended"],
4   env: {
5     node: true, // 启用node中全局变量
6     browser: true, // 启用浏览器中全局变量
7   },
8   parserOptions: {
9     ecmaVersion: 6,
10    sourceType: "module",
11  },
12  rules: {
13    "no-var": 2, // 不能使用 var 定义变量
14  },
15 };
```

### 3. 修改 js 文件代码

- main.js

```
1  import count from "./js/count";
2  import sum from "./js/sum";
3  // 引入资源, Webpack才会对其打包
4  import "./css/iconfont.css";
5  import "./css/index.css";
6  import "./less/index.less";
7  import "./sass/index.sass";
8  import "./sass/index.scss";
9  import "./styl/index.styl";
10
11  var result1 = count(2, 1);
12  console.log(result1);
13  var result2 = sum(1, 2, 3, 4);
14  console.log(result2);
```

### 1. 配置

- webpack.config.js

```
1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3
4  module.exports = {
5    entry: "./src/main.js",
6    output: {
7      path: path.resolve(__dirname, "dist"),
8      filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
9      clean: true, // 自动将上次打包目录资源清空
10   },
11   module: {
12     rules: [
13       {
14         // 用来匹配 .css 结尾的文件
15         test: /\.css$/,
16         // use 数组里面 Loader 执行顺序是从右到左
17         use: ["style-loader", "css-loader"],
18       },
19       {
20         test: /\.less$/,
21         use: ["style-loader", "css-loader", "less-loader"],
22       },
23       {
24         test: /\.s[ac]ss$/,
25         use: ["style-loader", "css-loader", "sass-loader"],
26       },
27     ],
28   },
29   plugins: [
30     new ESLintWebpackPlugin({
31       // 指定 ESLint 配置文件
32       configPath: ".eslintrc.js",
33     })
34   ],
35 }
```

```

27     {
28       test: /\.styl$/,
29       use: ["style-loader", "css-loader", "stylus-loader"],
30     },
31     {
32       test: /\.?(png|jpe?g|gif|webp)$/,
33       type: "asset",
34       parser: {
35         dataUrlCondition: {
36           maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
37         },
38       },
39       generator: {
40         // 将图片文件输出到 static/imgs 目录中
41         // 将图片文件命名 [hash:8][ext][query]
42         // [hash:8]: hash值取8位
43         // [ext]: 使用之前的文件扩展名
44         // [query]: 添加之前的query参数
45         filename: "static/imgs/[hash:8][ext][query]",
46       },
47     },
48     {
49       test: /\.?(ttf|woff2?)$/,
50       type: "asset/resource",
51       generator: {
52         filename: "static/media/[hash:8][ext][query]",
53       },
54     },
55   ],
56 },
57 plugins: [
58   new ESLintWebpackPlugin({
59     // 指定检查文件的根目录
60     context: path.resolve(__dirname, "src"),
61   }),
62 ],
63 mode: "development",
64 };

```

## 5. 运行指令

```
1 npx webpack
```

在控制台查看 Eslint 检查效果

## 4. VSCode Eslint 插件

打开 VSCode，下载 Eslint 插件，即可不用编译就能看到错误，可以提前解决

但是此时就会对项目所有文件默认进行 Eslint 检查了，我们 dist 目录下的打包后文件就会报错。但是我们只需要检查 src 下面的文件，不需要检查 dist 下面的文件。

所以可以使用 Eslint 忽略文件解决。在项目根目录新建下面文件：

- `.eslintignore`

```
1 # 忽略dist目录下所有文件
2 dist
```

## - Babel

JavaScript 编译器。

主要用于将 ES6 语法编写的代码转换为向后兼容的 JavaScript 语法，以便能够运行在当前和旧版本的浏览器或其他环境中

### 1. 配置文件

配置文件由很多种写法：

- `babel.config.*`：新建文件，位于项目根目录
  - `babel.config.js`
  - `babel.config.json`
- `.babelrc.*`：新建文件，位于项目根目录
  - `.babelrc`
  - `.babelrc.js`
  - `.babelrc.json`
- `package.json` 中 `babel`：不需要创建文件，在原有文件基础上写

Babel 会查找和自动读取它们，所以以上配置文件只需要存在一个即可

### 2. 具体配置

我们以 `babel.config.js` 配置文件为例：

```
1 module.exports = {
2   // 预设
3   presets: [],
4 };
```

#### 1. presets 预设

简单理解：就是一组 Babel 插件，扩展 Babel 功能

- `@babel/preset-env` : 一个智能预设, 允许您使用最新的 JavaScript。
- `@babel/preset-react` : 一个用来编译 React jsx 语法的预设
- `@babel/preset-typescript` : 一个用来编译 TypeScript 语法的预设

### 3. 在 Webpack 中使用

#### 1. 下载包

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

#### 2. 定义 Babel 配置文件

- `babel.config.js`

```
1 module.exports = {  
2   presets: ["@babel/preset-env"],  
3 };
```

#### 3. 修改 js 文件代码

- `main.js`

```
1 import count from "./js/count";  
2 import sum from "./js/sum";  
3 // 引入资源, Webpack才会对其打包  
4 import "./css/iconfont.css";  
5 import "./css/index.css";  
6 import "./less/index.less";  
7 import "./sass/index.sass";  
8 import "./sass/index.scss";  
9 import "./styl/index.styl";  
10  
11 const result1 = count(2, 1);  
12 console.log(result1);  
13 const result2 = sum(1, 2, 3, 4);  
14 console.log(result2);
```

#### 4. 配置

- `webpack.config.js`

```
1 const path = require("path");  
2 const ESLintWebpackPlugin = require("eslint-webpack-plugin");  
3  
4 module.exports = {  
5   entry: "./src/main.js",
```

```
6   output: {
7     path: path.resolve(__dirname, "dist"),
8     filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
9     clean: true, // 自动将上次打包目录资源清空
10  },
11  module: {
12    rules: [
13      {
14        // 用来匹配 .css 结尾的文件
15        test: /\.css$/,
16        // use 数组里面 Loader 执行顺序是从右到左
17        use: ["style-loader", "css-loader"],
18      },
19      {
20        test: /\.less$/,
21        use: ["style-loader", "css-loader", "less-loader"],
22      },
23      {
24        test: /\.s[ac]ss$/,
25        use: ["style-loader", "css-loader", "sass-loader"],
26      },
27      {
28        test: /\.styl$/,
29        use: ["style-loader", "css-loader", "stylus-loader"],
30      },
31      {
32        test: /\.?(png|jpe?g|gif|webp)$/ ,
33        type: "asset",
34        parser: {
35          dataUrlCondition: {
36            maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
37          },
38        },
39        generator: {
40          // 将图片文件输出到 static/imgs 目录中
41          // 将图片文件命名 [hash:8][ext][query]
42          // [hash:8]: hash值取8位
43          // [ext]: 使用之前的文件扩展名
44          // [query]: 添加之前的query参数
45          filename: "static/imgs/[hash:8][ext][query]",
46        },
47      },
48      {
49        test: /\.?(ttf|woff2?)$/,
50        type: "asset/resource",
51        generator: {
52          filename: "static/media/[hash:8][ext][query]",
53        },
54      },
55      {
56        test: /\.js$/,
```



```

57         exclude: /node_modules/, // 排除node_modules代码不编译
58         loader: "babel-loader",
59     },
60 ],
61 },
62 plugins: [
63     new ESLintWebpackPlugin({
64         // 指定检查文件的根目录
65         context: path.resolve(__dirname, "src"),
66     }),
67 ],
68 mode: "development",
69 };

```

## 5. 运行指令

```
1 npx webpack
```

打开打包后的 `dist/static/js/main.js` 文件查看，会发现箭头函数等 ES6 语法已经转换了

## • 处理 Html 资源

### – 1. 下载包

```
1 npm i html-webpack-plugin -D
```

### – 2. 配置

- webpack.config.js

```

1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4
5  module.exports = {
6      entry: "./src/main.js",
7      output: {
8          path: path.resolve(__dirname, "dist"),
9          filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
10         clean: true, // 自动将上次打包目录资源清空
11     },
12     module: {
13         rules: [
14             {
15                 // 用来匹配 .css 结尾的文件

```

```
16         test: /\.css$/,
17         // use 数组里面 Loader 执行顺序是从右到左
18         use: ["style-loader", "css-loader"],
19     },
20     {
21         test: /\.less$/,
22         use: ["style-loader", "css-loader", "less-loader"],
23     },
24     {
25         test: /\.s[ac]ss$/,
26         use: ["style-loader", "css-loader", "sass-loader"],
27     },
28     {
29         test: /\.styl$/,
30         use: ["style-loader", "css-loader", "stylus-loader"],
31     },
32     {
33         test: /\.?(png|jpe?g|gif|webp)?$/,
34         type: "asset",
35         parser: {
36             dataUrlCondition: {
37                 maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
38             },
39         },
40         generator: {
41             // 将图片文件输出到 static/imgs 目录中
42             // 将图片文件命名 [hash:8][ext][query]
43             // [hash:8]: hash值取8位
44             // [ext]: 使用之前的文件扩展名
45             // [query]: 添加之前的query参数
46             filename: "static/imgs/[hash:8][ext][query]",
47         },
48     },
49     {
50         test: /\.?(ttf|woff2?)$/,
51         type: "asset/resource",
52         generator: {
53             filename: "static/media/[hash:8][ext][query]",
54         },
55     },
56     {
57         test: /\.js$/,
58         exclude: /node_modules/, // 排除node_modules代码不编译
59         loader: "babel-loader",
60     },
61 ],
62 },
63 plugins: [
64     new ESLintWebpackPlugin({
65         // 指定检查文件的根目录
66         context: path.resolve(__dirname, "src"),
```

```

67     }),
68     new HtmlWebpackPlugin({
69       // 以 public/index.html 为模板创建文件
70       // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
71       template: path.resolve(__dirname, "public/index.html"),
72     }),
73   ],
74   mode: "development",
75 };

```

### – 3. 修改 index.html

去掉引入的 js 文件，因为 HtmlWebpackPlugin 会自动引入

```

1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>webpack5</title>
8    </head>
9    <body>
10     <h1>Hello Webpack5</h1>
11     <div class="box1"></div>
12     <div class="box2"></div>
13     <div class="box3"></div>
14     <div class="box4"></div>
15     <div class="box5"></div>
16     <i class="iconfont icon-arrow-down"></i>
17     <i class="iconfont icon-ashbin"></i>
18     <i class="iconfont icon-browse"></i>
19   </body>
20 </html>

```

### – 4. 运行指令

```

1  npx webpack

```

此时 dist 目录就会输出一个 index.html 文件

## • 开发服务器&自动化

每次写完代码都需要手动输入指令才能编译代码，太麻烦了，我们希望一切自动化

## - 1. 下载包

```
1 npm i webpack-dev-server -D
```

## - 2. 配置

- webpack.config.js

```
1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4
5  module.exports = {
6    entry: "./src/main.js",
7    output: {
8      path: path.resolve(__dirname, "dist"),
9      filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
10     clean: true, // 自动将上次打包目录资源清空
11   },
12   module: {
13     rules: [
14       {
15         // 用来匹配 .css 结尾的文件
16         test: /\.css$/,
17         // use 数组里面 Loader 执行顺序是从右到左
18         use: ["style-loader", "css-loader"],
19       },
20       {
21         test: /\.less$/,
22         use: ["style-loader", "css-loader", "less-loader"],
23       },
24       {
25         test: /\.s[ac]ss$/,
26         use: ["style-loader", "css-loader", "sass-loader"],
27       },
28       {
29         test: /\.styl$/,
30         use: ["style-loader", "css-loader", "stylus-loader"],
31       },
32       {
33         test: /\.(png|jpe?g|gif|webp)$/,
34         type: "asset",
35         parser: {
36           dataUrlCondition: {
37             maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
38           },
39         },
40         generator: {
41           // 将图片文件输出到 static/imgs 目录中
```

```

42         // 将图片文件命名 [hash:8][ext][query]
43         // [hash:8]: hash值取8位
44         // [ext]: 使用之前的文件扩展名
45         // [query]: 添加之前的query参数
46         filename: "static/imgs/[hash:8][ext][query]",
47     },
48 },
49 {
50     test: /\. (ttf|woff2?)$/,
51     type: "asset/resource",
52     generator: {
53         filename: "static/media/[hash:8][ext][query]",
54     },
55 },
56 {
57     test: /\.js$/,
58     exclude: /node_modules/, // 排除node_modules代码不编译
59     loader: "babel-loader",
60 },
61 ],
62 },
63 plugins: [
64     new ESLintWebpackPlugin({
65         // 指定检查文件的根目录
66         context: path.resolve(__dirname, "src"),
67     }),
68     new HtmlWebpackPlugin({
69         // 以 public/index.html 为模板创建文件
70         // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
71         template: path.resolve(__dirname, "public/index.html"),
72     }),
73 ],
74 // 开发服务器
75 devServer: {
76     host: "localhost", // 启动服务器域名
77     port: "3000", // 启动服务器端口号
78     open: true, // 是否自动打开浏览器
79 },
80 mode: "development",
81 };

```

### 3. 运行指令

```
1 npx webpack serve
```

#### 注意运行指令发生了变化

并且当你使用开发服务器时，所有代码都会在内存中编译打包，并不会输出到 dist 目录下。

开发时我们只关心代码能运行，有效果即可，至于代码被编译成什么样子，我们并不需要知道。

## • 生产模式介绍

生产模式是开发完成代码后，我们需要得到代码将来部署上线。

这个模式下我们主要对代码进行优化，让其运行性能更好。

优化主要从两个角度出发：

1. 优化代码运行性能
2. 优化代码打包速度

## — 生产模式准备

我们分别准备两个配置文件来放不同的配置

### 1. 文件目录

```
1  |— webpack-test (项目根目录)
2    |— config (Webpack配置文件目录)
3      |— webpack.dev.js(开发模式配置文件)
4      |— webpack.prod.js(生产模式配置文件)
5    |— node_modules (下载包存放目录)
6    |— src (项目源码目录, 除了html其他都在src里面)
7      |— 略
8    |— public (项目html文件)
9      |— index.html
10   |— .eslintrc.js(EsLint配置文件)
11   |— babel.config.js(Babel配置文件)
12   |— package.json (包的依赖管理配置文件)
```

### 2. 修改 webpack.dev.js

因为文件目录变了，所以所有绝对路径需要回退一层目录才能找到对应的文件

```
1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4
5  module.exports = {
6    entry: "./src/main.js",
7    output: {
8      path: undefined, // 开发模式没有输出, 不需要指定输出目录
9      filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
10     // clean: true, // 开发模式没有输出, 不需要清空输出结果
11   },
12   module: {
13     rules: [
14       {
```

```
15         // 用来匹配 .css 结尾的文件
16         test: /\.css$/,
17         // use 数组里面 Loader 执行顺序是从右到左
18         use: ["style-loader", "css-loader"],
19     },
20     {
21         test: /\.less$/,
22         use: ["style-loader", "css-loader", "less-loader"],
23     },
24     {
25         test: /\.s[ac]ss$/,
26         use: ["style-loader", "css-loader", "sass-loader"],
27     },
28     {
29         test: /\.styl$/,
30         use: ["style-loader", "css-loader", "stylus-loader"],
31     },
32     {
33         test: /\.(png|jpe?g|gif|webp)$/,
34         type: "asset",
35         parser: {
36             dataUrlCondition: {
37                 maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
38             },
39         },
40         generator: {
41             // 将图片文件输出到 static/imgs 目录中
42             // 将图片文件命名 [hash:8][ext][query]
43             // [hash:8]: hash值取8位
44             // [ext]: 使用之前的文件扩展名
45             // [query]: 添加之前的query参数
46             filename: "static/imgs/[hash:8][ext][query]",
47         },
48     },
49     {
50         test: /\.(ttf|woff2?)$/,
51         type: "asset/resource",
52         generator: {
53             filename: "static/media/[hash:8][ext][query]",
54         },
55     },
56     {
57         test: /\.js$/,
58         exclude: /node_modules/, // 排除node_modules代码不编译
59         loader: "babel-loader",
60     },
61 ],
62 },
63 plugins: [
64     new ESLintWebpackPlugin({
65         // 指定检查文件的根目录
```

```

66     context: path.resolve(__dirname, "../src"),
67   }),
68   new HtmlWebpackPlugin({
69     // 以 public/index.html 为模板创建文件
70     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
71     template: path.resolve(__dirname, "../public/index.html"),
72   }),
73 ],
74 // 其他省略
75 devServer: {
76   host: "localhost", // 启动服务器域名
77   port: "3000", // 启动服务器端口号
78   open: true, // 是否自动打开浏览器
79 },
80 mode: "development",
81 };

```

运行开发模式的指令:

```

1  npx webpack serve --config ./config/webpack.dev.js

```

### 3. 修改 webpack.prod.js

```

1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4
5  module.exports = {
6    entry: "./src/main.js",
7    output: {
8      path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
9      filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
10     clean: true,
11   },
12   module: {
13     rules: [
14       {
15         // 用来匹配 .css 结尾的文件
16         test: /\.css$/,
17         // use 数组里面 Loader 执行顺序是从右到左
18         use: ["style-loader", "css-loader"],
19       },
20       {
21         test: /\.less$/,
22         use: ["style-loader", "css-loader", "less-loader"],
23       },
24       {
25         test: /\.s[ac]ss$/,

```



```
26     use: ["style-loader", "css-loader", "sass-loader"],
27   },
28   {
29     test: /\.styl$/,
30     use: ["style-loader", "css-loader", "stylus-loader"],
31   },
32   {
33     test: /\.?(png|jpe?g|gif|webp)$/,
34     type: "asset",
35     parser: {
36       dataUrlCondition: {
37         maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
38       },
39     },
40     generator: {
41       // 将图片文件输出到 static/imgs 目录中
42       // 将图片文件命名 [hash:8][ext][query]
43       // [hash:8]: hash值取8位
44       // [ext]: 使用之前的文件扩展名
45       // [query]: 添加之前的query参数
46       filename: "static/imgs/[hash:8][ext][query]",
47     },
48   },
49   {
50     test: /\.?(ttf|woff2?)$/,
51     type: "asset/resource",
52     generator: {
53       filename: "static/media/[hash:8][ext][query]",
54     },
55   },
56   {
57     test: /\.js$/,
58     exclude: /node_modules/, // 排除node_modules代码不编译
59     loader: "babel-loader",
60   },
61 ],
62 },
63 plugins: [
64   new ESLintWebpackPlugin({
65     // 指定检查文件的根目录
66     context: path.resolve(__dirname, "../src"),
67   }),
68   new HtmlWebpackPlugin({
69     // 以 public/index.html 为模板创建文件
70     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
71     template: path.resolve(__dirname, "../public/index.html"),
72   }),
73 ],
74 // devServer: {
75 //   host: "localhost", // 启动服务器域名
76 //   port: "3000", // 启动服务器端口号
```

```
77     // open: true, // 是否自动打开浏览器
78     // },
79     mode: "production",
80   };
```

运行生产模式的指令：

```
1  npx webpack --config ./config/webpack.prod.js
```

#### 4. 配置运行指令

为了方便运行不同模式的指令，我们将指令定义在 package.json 中 scripts 里面

```
1  // package.json
2  {
3    // 其他省略
4    "scripts": {
5      "start": "npm run dev",
6      "dev": "npx webpack serve --config ./config/webpack.dev.js",
7      "build": "npx webpack --config ./config/webpack.prod.js"
8    }
9  }
```

以后启动指令：

- 开发模式： `npm start` 或 `npm run dev`
- 生产模式： `npm run build`

### • Css 处理

#### — 提取 Css 成单独文件

Css 文件目前被打包到 js 文件中，当 js 文件加载时，会创建一个 style 标签来生成样式

这样对于网站来说，会出现闪屏现象，用户体验不好

我们应该是单独的 Css 文件，通过 link 标签加载性能才好

#### 1. 下载包

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

## 2. 配置

- webpack.prod.js

```
1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
5
6  module.exports = {
7    entry: "./src/main.js",
8    output: {
9      path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
10     filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
11     clean: true,
12   },
13   module: {
14     rules: [
15       {
16         // 用来匹配 .css 结尾的文件
17         test: /\.css$/,
18         // use 数组里面 Loader 执行顺序是从右到左
19         use: [MiniCssExtractPlugin.loader, "css-loader"],
20       },
21       {
22         test: /\.less$/,
23         use: [MiniCssExtractPlugin.loader, "css-loader", "less-loader"],
24       },
25       {
26         test: /\.s[ac]ss$/,
27         use: [MiniCssExtractPlugin.loader, "css-loader", "sass-loader"],
28       },
29       {
30         test: /\.styl$/,
31         use: [MiniCssExtractPlugin.loader, "css-loader", "stylus-loader"],
32       },
33       {
34         test: /\.(png|jpe?g|gif|webp)$/,
35         type: "asset",
36         parser: {
37           dataUrlCondition: {
38             maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
39           },
40         },
41         generator: {
42           // 将图片文件输出到 static/imgs 目录中
43           // 将图片文件命名 [hash:8][ext][query]
44           // [hash:8]: hash值取8位
45           // [ext]: 使用之前的文件扩展名
46           // [query]: 添加之前的query参数
47           filename: "static/imgs/[hash:8][ext][query]",
```

```

48     },
49   },
50   {
51     test: /\. (ttf|woff2?) $/,
52     type: "asset/resource",
53     generator: {
54       filename: "static/media/[hash:8][ext][query]",
55     },
56   },
57   {
58     test: /\. js $/,
59     exclude: /node_modules/, // 排除node_modules代码不编译
60     loader: "babel-loader",
61   },
62 ],
63 },
64 plugins: [
65   new ESLintWebpackPlugin({
66     // 指定检查文件的根目录
67     context: path.resolve(__dirname, "../src"),
68   }),
69   new HtmlWebpackPlugin({
70     // 以 public/index.html 为模板创建文件
71     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
72     template: path.resolve(__dirname, "../public/index.html"),
73   }),
74   // 提取css成单独文件
75   new MiniCssExtractPlugin({
76     // 定义输出文件名和目录
77     filename: "static/css/main.css",
78   }),
79 ],
80 // devServer: {
81 //   host: "localhost", // 启动服务器域名
82 //   port: "3000", // 启动服务器端口号
83 //   open: true, // 是否自动打开浏览器
84 // },
85 mode: "production",
86 };

```

### 3. 运行指令

```

1  npm run build

```

## - Css 兼容性处理

### 1. 下载包

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

### 2. 配置

- webpack.prod.js

```
1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
5
6  module.exports = {
7    entry: "./src/main.js",
8    output: {
9      path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
10     filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
11     clean: true,
12   },
13   module: {
14     rules: [
15       {
16         // 用来匹配 .css 结尾的文件
17         test: /\.css$/,
18         // use 数组里面 Loader 执行顺序是从右到左
19         use: [
20           MiniCssExtractPlugin.loader,
21           "css-loader",
22           {
23             loader: "postcss-loader",
24             options: {
25               postcssOptions: {
26                 plugins: [
27                   "postcss-preset-env", // 能解决大多数样式兼容性问题
28                 ],
29               },
30             },
31           },
32         ],
33       },
34       {
35         test: /\.less$/,
36         use: [
37           MiniCssExtractPlugin.loader,
38           "css-loader",
39           {
40             loader: "postcss-loader",
```

```
41     options: {
42       postcssOptions: {
43         plugins: [
44           "postcss-preset-env", // 能解决大多数样式兼容性问题
45         ],
46       },
47     },
48   },
49   "less-loader",
50 ],
51 },
52 {
53   test: /\.s[ac]ss$/,
54   use: [
55     MiniCssExtractPlugin.loader,
56     "css-loader",
57     {
58       loader: "postcss-loader",
59       options: {
60         postcssOptions: {
61           plugins: [
62             "postcss-preset-env", // 能解决大多数样式兼容性问题
63           ],
64         },
65       },
66     },
67     "sass-loader",
68   ],
69 },
70 {
71   test: /\.styl$/,
72   use: [
73     MiniCssExtractPlugin.loader,
74     "css-loader",
75     {
76       loader: "postcss-loader",
77       options: {
78         postcssOptions: {
79           plugins: [
80             "postcss-preset-env", // 能解决大多数样式兼容性问题
81           ],
82         },
83       },
84     },
85     "stylus-loader",
86   ],
87 },
88 {
89   test: /\.?(png|jpe?g|gif|webp)$/,
90   type: "asset",
91   parser: {
```

```

92         dataUrlCondition: {
93             maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
94         },
95     },
96     generator: {
97         // 将图片文件输出到 static/imgs 目录中
98         // 将图片文件命名 [hash:8][ext][query]
99         // [hash:8]: hash值取8位
100        // [ext]: 使用之前的文件扩展名
101        // [query]: 添加之前的query参数
102        filename: "static/imgs/[hash:8][ext][query]",
103    },
104 },
105 {
106     test: /\.?(ttf|woff2?)$/,
107     type: "asset/resource",
108     generator: {
109         filename: "static/media/[hash:8][ext][query]",
110     },
111 },
112 {
113     test: /\.js$/,
114     exclude: /node_modules/, // 排除node_modules代码不编译
115     loader: "babel-loader",
116 },
117 ],
118 },
119 plugins: [
120     new ESLintWebpackPlugin({
121         // 指定检查文件的根目录
122         context: path.resolve(__dirname, "../src"),
123     }),
124     new HtmlWebpackPlugin({
125         // 以 public/index.html 为模板创建文件
126         // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
127         template: path.resolve(__dirname, "../public/index.html"),
128     }),
129     // 提取css成单独文件
130     new MiniCssExtractPlugin({
131         // 定义输出文件名和目录
132         filename: "static/css/main.css",
133     }),
134 ],
135 // devServer: {
136 //     host: "localhost", // 启动服务器域名
137 //     port: "3000", // 启动服务器端口号
138 //     open: true, // 是否自动打开浏览器
139 // },
140 mode: "production",
141 };

```

### 3. 控制兼容性

我们可以在 `package.json` 文件中添加 `browserslist` 来控制样式的兼容性做到什么程度。

```
1  {
2    // 其他省略
3    "browserslist": ["ie ≥ 8"]
4  }
```

想要知道更多的 `browserslist` 配置, 查看[browserslist 文档](#)

以上为了测试兼容性所以设置兼容浏览器 ie8 以上。

实际开发中我们一般不考虑旧版本浏览器了, 所以我们可以这样设置:

```
1  {
2    // 其他省略
3    "browserslist": ["last 2 version", "> 1%", "not dead"]
4  }
```

### 4. 合并配置

- webpack.prod.js

```
1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
5
6  // 获取处理样式的Loaders
7  const getStyleLoaders = (preProcessor) => {
8    return [
9      MiniCssExtractPlugin.loader,
10     "css-loader",
11     {
12       loader: "postcss-loader",
13       options: {
14         postcssOptions: {
15           plugins: [
16             "postcss-preset-env", // 能解决大多数样式兼容性问题
17           ],
18         },
19       },
20     },
21     preProcessor,
22   ].filter(Boolean);
23 };
24
```



```
25 module.exports = {
26   entry: "./src/main.js",
27   output: {
28     path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
29     filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
30     clean: true,
31   },
32   module: {
33     rules: [
34       {
35         // 用来匹配 .css 结尾的文件
36         test: /\.css$/,
37         // use 数组里面 Loader 执行顺序是从右到左
38         use: getStyleLoaders(),
39       },
40       {
41         test: /\.less$/,
42         use: getStyleLoaders("less-loader"),
43       },
44       {
45         test: /\.s[ac]ss$/,
46         use: getStyleLoaders("sass-loader"),
47       },
48       {
49         test: /\.styl$/,
50         use: getStyleLoaders("stylus-loader"),
51       },
52       {
53         test: /\.?(png|jpe?g|gif|webp)$/,
54         type: "asset",
55         parser: {
56           dataUrlCondition: {
57             maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
58           },
59         },
60         generator: {
61           // 将图片文件输出到 static/imgs 目录中
62           // 将图片文件命名 [hash:8][ext][query]
63           // [hash:8]: hash值取8位
64           // [ext]: 使用之前的文件扩展名
65           // [query]: 添加之前的query参数
66           filename: "static/imgs/[hash:8][ext][query]",
67         },
68       },
69       {
70         test: /\.?(ttf|woff2?)$/,
71         type: "asset/resource",
72         generator: {
73           filename: "static/media/[hash:8][ext][query]",
74         },
75       },

```

```

76     {
77         test: /\.js$/,
78         exclude: /node_modules/, // 排除node_modules代码不编译
79         loader: "babel-loader",
80     },
81 ],
82 },
83 plugins: [
84     new ESLintWebpackPlugin({
85         // 指定检查文件的根目录
86         context: path.resolve(__dirname, "../src"),
87     }),
88     new HtmlWebpackPlugin({
89         // 以 public/index.html 为模板创建文件
90         // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
91         template: path.resolve(__dirname, "../public/index.html"),
92     }),
93     // 提取css成单独文件
94     new MiniCssExtractPlugin({
95         // 定义输出文件名和目录
96         filename: "static/css/main.css",
97     }),
98 ],
99 // devServer: {
100 //     host: "localhost", // 启动服务器域名
101 //     port: "3000", // 启动服务器端口号
102 //     open: true, // 是否自动打开浏览器
103 // },
104 mode: "production",
105 };

```

## 5. 运行指令

```
1 npm run build
```

## - Css 压缩

### 1. 下载包

```
1 npm i css-minimizer-webpack-plugin -D
```

### 2. 配置

- webpack.prod.js

```

1 const path = require("path");
2 const ESLintWebpackPlugin = require("eslint-webpack-plugin");

```

```
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
5  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
6
7  // 获取处理样式的Loaders
8  const getStyleLoaders = (preProcessor) => {
9    return [
10      MiniCssExtractPlugin.loader,
11      "css-loader",
12      {
13        loader: "postcss-loader",
14        options: {
15          postcssOptions: {
16            plugins: [
17              "postcss-preset-env", // 能解决大多数样式兼容性问题
18            ],
19          },
20        },
21      },
22      preProcessor,
23    ].filter(Boolean);
24  };
25
26  module.exports = {
27    entry: "./src/main.js",
28    output: {
29      path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
30      filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
31      clean: true,
32    },
33    module: {
34      rules: [
35        {
36          // 用来匹配 .css 结尾的文件
37          test: /\.css$/,
38          // use 数组里面 Loader 执行顺序是从右到左
39          use: getStyleLoaders(),
40        },
41        {
42          test: /\.less$/,
43          use: getStyleLoaders("less-loader"),
44        },
45        {
46          test: /\.s[ac]ss$/,
47          use: getStyleLoaders("sass-loader"),
48        },
49        {
50          test: /\.styl$/,
51          use: getStyleLoaders("stylus-loader"),
52        },
53        {
```

```
54     test: /\. (png|jpe?g|gif|webp)$/ ,
55     type: "asset" ,
56     parser: {
57       dataUrlCondition: {
58         maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
59       },
60     },
61     generator: {
62       // 将图片文件输出到 static/imgs 目录中
63       // 将图片文件命名 [hash:8][ext][query]
64       // [hash:8]: hash值取8位
65       // [ext]: 使用之前的文件扩展名
66       // [query]: 添加之前的query参数
67       filename: "static/imgs/[hash:8][ext][query]" ,
68     },
69   },
70   {
71     test: /\. (ttf|woff2?)$/ ,
72     type: "asset/resource" ,
73     generator: {
74       filename: "static/media/[hash:8][ext][query]" ,
75     },
76   },
77   {
78     test: /\. js$/ ,
79     exclude: /node_modules/ , // 排除node_modules代码不编译
80     loader: "babel-loader" ,
81   },
82 ],
83 },
84 plugins: [
85   new ESLintWebpackPlugin({
86     // 指定检查文件的根目录
87     context: path.resolve(__dirname, ".. /src" ) ,
88   }) ,
89   new HtmlWebpackPlugin({
90     // 以 public/index.html 为模板创建文件
91     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
92     template: path.resolve(__dirname, ".. /public/index.html" ) ,
93   }) ,
94   // 提取css成单独文件
95   new MiniCssExtractPlugin({
96     // 定义输出文件名和目录
97     filename: "static/css/main.css" ,
98   }) ,
99   // css压缩
100   new CssMinimizerPlugin() ,
101 ],
102 // devServer: {
103 //   host: "localhost" , // 启动服务器域名
104 //   port: "3000" , // 启动服务器端口号
```

```
105     // open: true, // 是否自动打开浏览器
106     // },
107     mode: "production",
108   };
```

### 3. 运行指令

```
1  npm run build
```

#### • html 压缩

默认生产模式已经开启了：html 压缩和 js 压缩

不需要额外进行配置

#### • 总结

本章节我们学会了 Webpack 基本使用，掌握了以下功能：

##### 1. 两种开发模式

- 开发模式：代码能编译自动化运行
- 生产模式：代码编译优化输出

##### 2. Webpack 基本功能

- 开发模式：可以编译 ES Module 语法
- 生产模式：可以编译 ES Module 语法，压缩 js 代码

##### 3. Webpack 配置文件

- 5 个核心概念
  - entry
  - output
  - loader
  - plugins
  - mode
- devServer 配置

##### 4. Webpack 脚本指令用法

- `webpack` 直接打包输出
- `webpack serve` 启动开发服务器，内存编译打包没有输出

## 高级优化

## • 介绍

本章节主要介绍 Webpack 高级配置。

所谓高级配置其实就是进行 Webpack 优化，让我们代码在编译/运行时性能更好~

我们会从以下角度来进行优化：

1. 提升开发体验
2. 提升打包构建速度
3. 减少代码体积
4. 优化代码运行性能

## • 提升开发体验

### – SourceMap

#### 为什么

开发时我们运行的代码是经过 webpack 编译后的，例如下面这个样子：

```
1  /*
2   * ATTENTION: The "eval" devtool has been used (maybe by default in mode:
   * "development").
3   * This devtool is neither made for production nor for readable output files.
4   * It uses "eval()" calls to create a separate source file in the browser
   devtools.
5   * If you are trying to read the output file, select a different devtool
   (https://webpack.js.org/configuration/devtool/)
6   * or disable the default devtool with "devtool: false".
7   * If you are looking for production-ready output files, see mode:
   "production" (https://webpack.js.org/configuration/mode/).
8   */
9  /******/ (() => { // webpackBootstrap
10 /******/    "use strict";
11 /******/    var __webpack_modules__ = ({
12
13 /****/ "../node_modules/css-loader/dist/cjs.js!./node_modules/less-
   loader/dist/cjs.js!./src/less/index.less":
14 /*!*****
   *****!\
15     !*** ./node_modules/css-loader/dist/cjs.js!./node_modules/less-
   loader/dist/cjs.js!./src/less/index.less ***!
16
   \*****
   *****/
17 /****/ ((module, __webpack_exports__, __webpack_require__) => {
18
```

```

19  eval("__webpack_require__.r(__webpack_exports__);\n/* harmony export */
    __webpack_require__.d(__webpack_exports__, {\n/* harmony export */
    \"default\": () => (__WEBPACK_DEFAULT_EXPORT__)\n/* harmony export */ });\n/*
    harmony import */ var
    _node_modules_css_loader_dist_runtime_noSourceMaps_js__WEBPACK_IMPORTED_MODULE_0__ =
    __webpack_require__(/*! ../../node_modules/css-loader/dist/runtime/noSourceMaps.js */
    \"./node_modules/css-loader/dist/runtime/noSourceMaps.js\");\n/* harmony import */ var
    _node_modules_css_loader_dist_runtime_noSourceMaps_js__WEBPACK_IMPORTED_MODULE_0___default =
    /*#__PURE__*/ __webpack_require__.n(_node_modules_css_loader_dist_runtime_noSourceMaps_js__WEBPACK_IMPORTED_MODULE_0__);
    \n/* harmony import */ var
    _node_modules_css_loader_dist_runtime_api_js__WEBPACK_IMPORTED_MODULE_1__ =
    __webpack_require__(/*! ../../node_modules/css-loader/dist/runtime/api.js */
    \"./node_modules/css-loader/dist/runtime/api.js\");\n/* harmony import */ var
    _node_modules_css_loader_dist_runtime_api_js__WEBPACK_IMPORTED_MODULE_1___default =
    /*#__PURE__*/ __webpack_require__.n(_node_modules_css_loader_dist_runtime_api_js__WEBPACK_IMPORTED_MODULE_1__);
    \n// Imports\n\n\nvar
    ___CSS_LOADER_EXPORT___ =
    _node_modules_css_loader_dist_runtime_api_js__WEBPACK_IMPORTED_MODULE_1___default()
    ((_node_modules_css_loader_dist_runtime_noSourceMaps_js__WEBPACK_IMPORTED_MODULE_0___default()));
    \n// Module\n___CSS_LOADER_EXPORT___push([module.id,
    \".box2 {\n  width: 100px;\n  height: 100px;\n  background-color:
    deeppink;\n}\", \"\"]);
    \n// Exports\n/* harmony default export */ const
    __WEBPACK_DEFAULT_EXPORT__ = (___CSS_LOADER_EXPORT___);
    \n\n\n//#
    sourceMappingURL=webpack://webpack5/./src/less/index.less?./node_modules/css-loader/dist/cjs.js!./node_modules/less-loader/dist/cjs.js");
20
21  /***/ }),
22  // 其他省略

```

所有 css 和 js 合并成了一个文件，并且多了其他代码。此时如果代码运行出错那么提示代码错误位置我们是看不懂的。一旦将来开发代码文件很多，那么很难去发现错误出现在哪里。

所以我们需要更加准确的错误提示，来帮助我们更好的开发代码。

## 是什么

SourceMap（源代码映射）是一个用来生成源代码与构建后代码——映射的文件的方案。

它会生成一个 xxx.map 文件，里面包含源代码和构建后代码每一行、每一列的映射关系。当构建后代码出错了，会通过 xxx.map 文件，从构建后代码出错位置找到映射后源代码出错位置，从而让浏览器提示源代码文件出错位置，帮助我们更快的找到错误根源。

## 怎么用

通过查看[Webpack DevTool 文档](#)可知，SourceMap 的值有很多种情况。

但实际开发时我们只需要关注两种情况即可：

- 开发模式: `cheap-module-source-map`
  - 优点: 打包编译速度快, 只包含行映射
  - 缺点: 没有列映射

```
1  module.exports = {  
2    // 其他省略  
3    mode: "development",  
4    devtool: "cheap-module-source-map",  
5  };
```

- 生产模式: `source-map`
  - 优点: 包含行/列映射
  - 缺点: 打包编译速度更慢

```
1  module.exports = {  
2    // 其他省略  
3    mode: "production",  
4    devtool: "source-map",  
5  };
```

## • 提升打包构建速度

### – HotModuleReplacement

#### 为什么

开发时我们修改了其中一个模块代码, Webpack 默认会将所有模块全部重新打包编译, 速度很慢。

所以我们需要做到修改某个模块代码, 就只有这个模块代码需要重新打包编译, 其他模块不变, 这样打包速度就能很快。

#### 是什么

HotModuleReplacement (HMR/热模块替换): 在程序运行中, 替换、添加或删除模块, 而无需重新加载整个页面。

#### 怎么用

##### 1. 基本配置



```

1  module.exports = {
2    // 其他省略
3    devServer: {
4      host: "localhost", // 启动服务器域名
5      port: "3000", // 启动服务器端口号
6      open: true, // 是否自动打开浏览器
7      hot: true, // 开启HMR功能 (只能用于开发环境, 生产环境不需要了)
8    },
9  };

```

此时 css 样式经过 style-loader 处理, 已经具备 HMR 功能了。  
但是 js 还不行。

## 2. JS 配置

```

1  // main.js
2  import count from "./js/count";
3  import sum from "./js/sum";
4  // 引入资源, Webpack才会对其打包
5  import "./css/iconfont.css";
6  import "./css/index.css";
7  import "./less/index.less";
8  import "./sass/index.sass";
9  import "./sass/index.scss";
10 import "./styl/index.styl";
11
12 const result1 = count(2, 1);
13 console.log(result1);
14 const result2 = sum(1, 2, 3, 4);
15 console.log(result2);
16
17 // 判断是否支持HMR功能
18 if (module.hot) {
19   module.hot.accept("./js/count.js", function (count) {
20     const result1 = count(2, 1);
21     console.log(result1);
22   });
23
24   module.hot.accept("./js/sum.js", function (sum) {
25     const result2 = sum(1, 2, 3, 4);
26     console.log(result2);
27   });
28 }

```

上面这样写会很麻烦, 所以实际开发我们会使用其他 loader 来解决。

比如: [vue-loader](#), [react-hot-loader](#)。

## - OneOf

### 为什么

打包时每个文件都会经过所有 loader 处理，虽然因为 `test` 正则原因实际没有处理上，但是都要过一遍。比较慢。

### 是什么

顾名思义就是只能匹配上一个 loader，剩下的就不匹配了。

### 怎么用

```
1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4
5  module.exports = {
6    entry: "./src/main.js",
7    output: {
8      path: undefined, // 开发模式没有输出，不需要指定输出目录
9      filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
10     // clean: true, // 开发模式没有输出，不需要清空输出结果
11   },
12   module: {
13     rules: [
14       {
15         oneOf: [
16           {
17             // 用来匹配 .css 结尾的文件
18             test: /\.css$/,
19             // use 数组里面 Loader 执行顺序是从右到左
20             use: ["style-loader", "css-loader"],
21           },
22           {
23             test: /\.less$/,
24             use: ["style-loader", "css-loader", "less-loader"],
25           },
26           {
27             test: /\.s[ac]ss$/,
28             use: ["style-loader", "css-loader", "sass-loader"],
29           },
30           {
31             test: /\.styl$/,
32             use: ["style-loader", "css-loader", "stylus-loader"],
33           },
34           {
35             test: /\.(png|jpe?g|gif|webp)$/,
36             type: "asset",
37             parser: {
38               dataUrlCondition: {
39                 maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
```

```

40     },
41   },
42   generator: {
43     // 将图片文件输出到 static/imgs 目录中
44     // 将图片文件命名 [hash:8][ext][query]
45     // [hash:8]: hash值取8位
46     // [ext]: 使用之前的文件扩展名
47     // [query]: 添加之前的query参数
48     filename: "static/imgs/[hash:8][ext][query]",
49   },
50 },
51 {
52   test: /\.?(ttf|woff2?)$/,
53   type: "asset/resource",
54   generator: {
55     filename: "static/media/[hash:8][ext][query]",
56   },
57 },
58 {
59   test: /\.js$/,
60   exclude: /node_modules/, // 排除node_modules代码不编译
61   loader: "babel-loader",
62 },
63 ],
64 },
65 ],
66 },
67 plugins: [
68   new ESLintWebpackPlugin({
69     // 指定检查文件的根目录
70     context: path.resolve(__dirname, "../src"),
71   }),
72   new HtmlWebpackPlugin({
73     // 以 public/index.html 为模板创建文件
74     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
75     template: path.resolve(__dirname, "../public/index.html"),
76   }),
77 ],
78 // 开发服务器
79 devServer: {
80   host: "localhost", // 启动服务器域名
81   port: "3000", // 启动服务器端口号
82   open: true, // 是否自动打开浏览器
83   hot: true, // 开启HMR功能
84 },
85 mode: "development",
86 devtool: "cheap-module-source-map",
87 };

```

生产模式也是如此配置。

## - Include/Exclude

### 为什么

开发时我们需要使用第三方的库或插件，所有文件都下载到 node\_modules 中了。而这些文件是不需要编译可以直接使用的。

所以我们在对 js 文件处理时，要排除 node\_modules 下面的文件。

### 是什么

- include

包含，只处理 xxx 文件

- exclude

排除，除了 xxx 文件以外其他文件都处理

### 怎么用

```
1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4
5  module.exports = {
6    entry: "./src/main.js",
7    output: {
8      path: undefined, // 开发模式没有输出，不需要指定输出目录
9      filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
10     // clean: true, // 开发模式没有输出，不需要清空输出结果
11   },
12   module: {
13     rules: [
14       {
15         oneOf: [
16           {
17             // 用来匹配 .css 结尾的文件
18             test: /\.css$/,
19             // use 数组里面 Loader 执行顺序是从右到左
20             use: ["style-loader", "css-loader"],
21           },
22           {
23             test: /\.less$/,
24             use: ["style-loader", "css-loader", "less-loader"],
25           },
26           {
27             test: /\.s[ac]ss$/,
28             use: ["style-loader", "css-loader", "sass-loader"],
29           },
30           {
31             test: /\.styl$/,
32             use: ["style-loader", "css-loader", "stylus-loader"],
```

```
33     },
34     {
35       test: /\. (png|jpe?g|gif|webp)$/ ,
36       type: "asset" ,
37       parser: {
38         dataUrlCondition: {
39           maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
40         },
41       },
42       generator: {
43         // 将图片文件输出到 static/imgs 目录中
44         // 将图片文件命名 [hash:8][ext][query]
45         // [hash:8]: hash值取8位
46         // [ext]: 使用之前的文件扩展名
47         // [query]: 添加之前的query参数
48         filename: "static/imgs/[hash:8][ext][query]" ,
49       },
50     },
51     {
52       test: /\. (ttf|woff2?)$/ ,
53       type: "asset/resource" ,
54       generator: {
55         filename: "static/media/[hash:8][ext][query]" ,
56       },
57     },
58     {
59       test: /\. js$/ ,
60       // exclude: /node_modules/ , // 排除node_modules代码不编译
61       include: path.resolve(__dirname, ".. /src") , // 也可以用包含
62       loader: "babel-loader" ,
63     },
64   ],
65 },
66 ],
67 },
68 plugins: [
69   new ESLintWebpackPlugin({
70     // 指定检查文件的根目录
71     context: path.resolve(__dirname, ".. /src") ,
72     exclude: "node_modules" , // 默认值
73   }) ,
74   new HtmlWebpackPlugin({
75     // 以 public/index.html 为模板创建文件
76     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
77     template: path.resolve(__dirname, ".. /public/index.html") ,
78   }) ,
79 ],
80 // 开发服务器
81 devServer: {
82   host: "localhost" , // 启动服务器域名
83   port: "3000" , // 启动服务器端口号
```

```
84     open: true, // 是否自动打开浏览器
85     hot: true, // 开启HMR功能
86   },
87   mode: "development",
88   devtool: "cheap-module-source-map",
89 };
```

生产模式也是如此配置。

## – Cache

### 为什么

每次打包时 js 文件都要经过 Eslint 检查 和 Babel 编译，速度比较慢。

我们可以缓存之前的 Eslint 检查 和 Babel 编译结果，这样第二次打包时速度就会更快了。

### 是什么

对 Eslint 检查 和 Babel 编译结果进行缓存。

### 怎么用

```
1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4
5  module.exports = {
6    entry: "./src/main.js",
7    output: {
8      path: undefined, // 开发模式没有输出，不需要指定输出目录
9      filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
10     // clean: true, // 开发模式没有输出，不需要清空输出结果
11   },
12   module: {
13     rules: [
14       {
15         oneOf: [
16           {
17             // 用来匹配 .css 结尾的文件
18             test: /\.css$/,
19             // use 数组里面 Loader 执行顺序是从右到左
20             use: ["style-loader", "css-loader"],
21           },
22           {
23             test: /\.less$/,
24             use: ["style-loader", "css-loader", "less-loader"],
25           },
26           {
27             test: /\.s[ac]ss$/,
28             use: ["style-loader", "css-loader", "sass-loader"],
29           },

```

```
30     {
31         test: /\.styl$/,
32         use: ["style-loader", "css-loader", "stylus-loader"],
33     },
34     {
35         test: /\.(png|jpe?g|gif|webp)$/,
36         type: "asset",
37         parser: {
38             dataUrlCondition: {
39                 maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
40             },
41         },
42         generator: {
43             // 将图片文件输出到 static/imgs 目录中
44             // 将图片文件命名 [hash:8][ext][query]
45             // [hash:8]: hash值取8位
46             // [ext]: 使用之前的文件扩展名
47             // [query]: 添加之前的query参数
48             filename: "static/imgs/[hash:8][ext][query]",
49         },
50     },
51     {
52         test: /\.(ttf|woff2?)$/,
53         type: "asset/resource",
54         generator: {
55             filename: "static/media/[hash:8][ext][query]",
56         },
57     },
58     {
59         test: /\.js$/,
60         // exclude: /node_modules/, // 排除node_modules代码不编译
61         include: path.resolve(__dirname, "../src"), // 也可以用包含
62         loader: "babel-loader",
63         options: {
64             cacheDirectory: true, // 开启babel编译缓存
65             cacheCompression: false, // 缓存文件不要压缩
66         },
67     },
68 ],
69 },
70 ],
71 },
72 plugins: [
73     new ESLintWebpackPlugin({
74         // 指定检查文件的根目录
75         context: path.resolve(__dirname, "../src"),
76         exclude: "node_modules", // 默认值
77         cache: true, // 开启缓存
78         // 缓存目录
79         cacheLocation: path.resolve(
80             __dirname,
```

```

81     "../node_modules/.cache/.eslintcache"
82   ),
83   }),
84   new HtmlWebpackPlugin({
85     // 以 public/index.html 为模板创建文件
86     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
87     template: path.resolve(__dirname, "../public/index.html"),
88   }),
89 ],
90 // 开发服务器
91 devServer: {
92   host: "localhost", // 启动服务器域名
93   port: "3000", // 启动服务器端口号
94   open: true, // 是否自动打开浏览器
95   hot: true, // 开启HMR功能
96 },
97 mode: "development",
98 devtool: "cheap-module-source-map",
99 };

```

## — Thread

### 为什么

当项目越来越庞大时，打包速度越来越慢，甚至于需要一个下午才能打包出来代码。这个速度是比较慢的。

我们想要继续提升打包速度，其实就是要提升 js 的打包速度，因为其他文件都比较少。

而对 js 文件处理主要就是 eslint、babel、Terser 三个工具，所以我们要提升它们的运行速度。

我们可以开启多进程同时处理 js 文件，这样速度就比之前的单进程打包更快了。

### 是什么

多进程打包：开启电脑的多个进程同时干一件事，速度更快。

△**需要注意：请仅在特别耗时的操作中使用，因为每个进程启动就有大约为 600ms 左右开销。**

### 怎么用

我们启动进程的数量就是我们 CPU 的核数。

1. 如何获取 CPU 的核数，因为每个电脑都不一样。

```

1  // nodejs核心模块，直接使用
2  const os = require("os");
3  // cpu核数
4  const threads = os.cpus().length;

```

2. 下载包



```
1 npm i thread-loader -D
```

### 3. 使用

```
1  const os = require("os");
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
7  const TerserPlugin = require("terser-webpack-plugin");
8
9  // cpu核数
10 const threads = os.cpus().length;
11
12 // 获取处理样式的Loaders
13 const getStyleLoaders = (preProcessor) => {
14   return [
15     MiniCssExtractPlugin.loader,
16     "css-loader",
17     {
18       loader: "postcss-loader",
19       options: {
20         postcssOptions: {
21           plugins: [
22             "postcss-preset-env", // 能解决大多数样式兼容性问题
23           ],
24         },
25       },
26     },
27     preProcessor,
28   ].filter(Boolean);
29 };
30
31 module.exports = {
32   entry: "./src/main.js",
33   output: {
34     path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
35     filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
36     clean: true,
37   },
38   module: {
39     rules: [
40       {
41         oneOf: [
42           {
43             // 用来匹配 .css 结尾的文件
44             test: /\.css$/,
45             // use 数组里面 Loader 执行顺序是从右到左
```

```
46     use: getStyleLoaders(),
47   },
48   {
49     test: /\.less$/,
50     use: getStyleLoaders("less-loader"),
51   },
52   {
53     test: /\.s[ac]ss$/,
54     use: getStyleLoaders("sass-loader"),
55   },
56   {
57     test: /\.styl$/,
58     use: getStyleLoaders("stylus-loader"),
59   },
60   {
61     test: /\.(png|jpe?g|gif|webp)$/,
62     type: "asset",
63     parser: {
64       dataUrlCondition: {
65         maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
66       },
67     },
68     generator: {
69       // 将图片文件输出到 static/imgs 目录中
70       // 将图片文件命名 [hash:8][ext][query]
71       // [hash:8]: hash值取8位
72       // [ext]: 使用之前的文件扩展名
73       // [query]: 添加之前的query参数
74       filename: "static/imgs/[hash:8][ext][query]",
75     },
76   },
77   {
78     test: /\.?(ttf|woff2?)$/,
79     type: "asset/resource",
80     generator: {
81       filename: "static/media/[hash:8][ext][query]",
82     },
83   },
84   {
85     test: /\.js$/,
86     // exclude: /node_modules/, // 排除node_modules代码不编译
87     include: path.resolve(__dirname, "../src"), // 也可以用包含
88     use: [
89       {
90         loader: "thread-loader", // 开启多进程
91         options: {
92           workers: threads, // 数量
93         },
94       },
95       {
96         loader: "babel-loader",
```

```

97         options: {
98             cacheDirectory: true, // 开启babel编译缓存
99         },
100     },
101 ],
102 },
103 ],
104 },
105 ],
106 },
107 plugins: [
108     new ESLintWebpackPlugin({
109         // 指定检查文件的根目录
110         context: path.resolve(__dirname, "../src"),
111         exclude: "node_modules", // 默认值
112         cache: true, // 开启缓存
113         // 缓存目录
114         cacheLocation: path.resolve(
115             __dirname,
116             "../node_modules/.cache/.eslintcache"
117         ),
118         threads, // 开启多进程
119     }),
120     new HtmlWebpackPlugin({
121         // 以 public/index.html 为模板创建文件
122         // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
123         template: path.resolve(__dirname, "../public/index.html"),
124     }),
125     // 提取css成单独文件
126     new MiniCssExtractPlugin({
127         // 定义输出文件名和目录
128         filename: "static/css/main.css",
129     }),
130     // css压缩
131     // new CssMinimizerPlugin(),
132 ],
133 optimization: {
134     minimize: true,
135     minimizer: [
136         // css压缩也可以写到optimization.minimizer里面, 效果一样的
137         new CssMinimizerPlugin(),
138         // 当生产模式会默认开启TerserPlugin, 但是我们需要进行其他配置, 就要重新写了
139         new TerserPlugin({
140             parallel: threads // 开启多进程
141         })
142     ],
143 },
144 // devServer: {
145 //     host: "localhost", // 启动服务器域名
146 //     port: "3000", // 启动服务器端口号
147 //     open: true, // 是否自动打开浏览器

```

```
148     // },
149     mode: "production",
150     devtool: "source-map",
151   };
```

我们目前打包的内容都很少，所以因为启动进程开销原因，使用多进程打包实际上会显著的让我们打包时间变得很长。

## • 减少代码体积

### – Tree Shaking

#### 为什么

开发时我们定义了一些工具函数库，或者引用第三方工具函数库或组件库。

如果没有特殊处理的话我们打包时会引入整个库，但是实际上可能我们可能只用上极小部分的功能。

这样将整个库都打包进来，体积就太大了。

#### 是什么

`Tree Shaking` 是一个术语，通常用于描述移除 JavaScript 中的没有使用上的代码。

△注意：它依赖 `ES Module`。

#### 怎么用

Webpack 已经默认开启了这个功能，无需其他配置。

### – Babel

#### 为什么

Babel 为编译的每个文件都插入了辅助代码，使代码体积过大！

Babel 对一些公共方法使用了非常小的辅助代码，比如 `_extend`。默认情况下会被添加到每一个需要它的文件中。

你可以将这些辅助代码作为一个独立模块，来避免重复引入。

#### 是什么

`@babel/plugin-transform-runtime`：禁用了 Babel 自动对每个文件的 runtime 注入，而是引入 `@babel/plugin-transform-runtime` 并且使所有辅助代码从这里引用。

#### 怎么用

1. 下载包

```
1  npm i @babel/plugin-transform-runtime -D
```

## 2. 配置

```
1  const os = require("os");
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
7  const TerserPlugin = require("terser-webpack-plugin");
8
9  // cpu核数
10 const threads = os.cpus().length;
11
12 // 获取处理样式的Loaders
13 const getStyleLoaders = (preProcessor) => {
14   return [
15     MiniCssExtractPlugin.loader,
16     "css-loader",
17     {
18       loader: "postcss-loader",
19       options: {
20         postcssOptions: {
21           plugins: [
22             "postcss-preset-env", // 能解决大多数样式兼容性问题
23           ],
24         },
25       },
26     },
27     preProcessor,
28   ].filter(Boolean);
29 };
30
31 module.exports = {
32   entry: "./src/main.js",
33   output: {
34     path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
35     filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
36     clean: true,
37   },
38   module: {
39     rules: [
40       {
41         oneOf: [
42           {
43             // 用来匹配 .css 结尾的文件
44             test: /\.css$/,
45             // use 数组里面 Loader 执行顺序是从右到左
46             use: getStyleLoaders(),
47           },
48           {
49             test: /\.less$/,
```

```
50     use: getStyleLoaders("less-loader"),
51   },
52   {
53     test: /\.s[ac]ss$/,
54     use: getStyleLoaders("sass-loader"),
55   },
56   {
57     test: /\.styl$/,
58     use: getStyleLoaders("stylus-loader"),
59   },
60   {
61     test: /\.(png|jpe?g|gif|webp)$/,
62     type: "asset",
63     parser: {
64       dataUrlCondition: {
65         maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
66       },
67     },
68     generator: {
69       // 将图片文件输出到 static/imgs 目录中
70       // 将图片文件命名 [hash:8][ext][query]
71       // [hash:8]: hash值取8位
72       // [ext]: 使用之前的文件扩展名
73       // [query]: 添加之前的query参数
74       filename: "static/imgs/[hash:8][ext][query]",
75     },
76   },
77   {
78     test: /\.((ttf|woff2?))$/,
79     type: "asset/resource",
80     generator: {
81       filename: "static/media/[hash:8][ext][query]",
82     },
83   },
84   {
85     test: /\.js$/,
86     // exclude: /node_modules/, // 排除node_modules代码不编译
87     include: path.resolve(__dirname, "../src"), // 也可以用包含
88     use: [
89       {
90         loader: "thread-loader", // 开启多进程
91         options: {
92           workers: threads, // 数量
93         },
94       },
95       {
96         loader: "babel-loader",
97         options: {
98           cacheDirectory: true, // 开启babel编译缓存
99           cacheCompression: false, // 缓存文件不要压缩
```

```

100         plugins: ["@babel/plugin-transform-runtime"], // 减少代码体
    积
101     },
102     },
103     ],
104     },
105     ],
106     },
107     ],
108 },
109 plugins: [
110     new ESLintWebpackPlugin({
111         // 指定检查文件的根目录
112         context: path.resolve(__dirname, "../src"),
113         exclude: "node_modules", // 默认值
114         cache: true, // 开启缓存
115         // 缓存目录
116         cacheLocation: path.resolve(
117             __dirname,
118             "../node_modules/.cache/.eslintcache"
119         ),
120         threads, // 开启多进程
121     }),
122     new HtmlWebpackPlugin({
123         // 以 public/index.html 为模板创建文件
124         // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
125         template: path.resolve(__dirname, "../public/index.html"),
126     }),
127     // 提取css成单独文件
128     new MiniCssExtractPlugin({
129         // 定义输出文件名和目录
130         filename: "static/css/main.css",
131     }),
132     // css压缩
133     // new CssMinimizerPlugin(),
134 ],
135 optimization: {
136     minimizer: [
137         // css压缩也可以写到optimization.minimizer里面, 效果一样的
138         new CssMinimizerPlugin(),
139         // 当生产模式会默认开启TerserPlugin, 但是我们需要进行其他配置, 就要重新写了
140         new TerserPlugin({
141             parallel: threads, // 开启多进程
142         }),
143     ]
144 },
145 // devServer: {
146 //     host: "localhost", // 启动服务器域名
147 //     port: "3000", // 启动服务器端口号
148 //     open: true, // 是否自动打开浏览器
149 // },

```

```
150     mode: "production",
151     devtool: "source-map",
152   };
```

## - Image Minimizer

### 为什么

开发如果项目中引用了较多图片，那么图片体积会比较大，将来请求速度比较慢。

我们可以对图片进行压缩，减少图片体积。

△注意：如果项目中图片都是在线链接，那么就不需要了。本地项目静态图片才需要进行压缩。

### 是什么

`image-minimizer-webpack-plugin`：用来压缩图片的插件

### 怎么用

1. 下载包

```
1  npm i image-minimizer-webpack-plugin imagemin -D
```

还有剩下包需要下载，有两种模式：

- 无损压缩

```
1  npm install imagemin-gifsicle imagemin-jpegtran imagemin-optipng imagemin-svgo
   -D
```

- 有损压缩

```
1  npm install imagemin-gifsicle imagemin-mozjpeg imagemin-pngquant imagemin-svgo
   -D
```

“

[有损/无损压缩的区别](#)

2. 配置

我们以无损压缩配置为例：

```
1  const os = require("os");
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
```



```
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
7  const TerserPlugin = require("terser-webpack-plugin");
8  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
9
10 // cpu核数
11 const threads = os.cpus().length;
12
13 // 获取处理样式的Loaders
14 const getStyleLoaders = (preProcessor) => {
15   return [
16     MiniCssExtractPlugin.loader,
17     "css-loader",
18     {
19       loader: "postcss-loader",
20       options: {
21         postcssOptions: {
22           plugins: [
23             "postcss-preset-env", // 能解决大多数样式兼容性问题
24           ],
25         },
26       },
27     },
28     preProcessor,
29   ].filter(Boolean);
30 };
31
32 module.exports = {
33   entry: "./src/main.js",
34   output: {
35     path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
36     filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
37     clean: true,
38   },
39   module: {
40     rules: [
41       {
42         oneOf: [
43           {
44             // 用来匹配 .css 结尾的文件
45             test: /\.css$/,
46             // use 数组里面 Loader 执行顺序是从右到左
47             use: getStyleLoaders(),
48           },
49           {
50             test: /\.less$/,
51             use: getStyleLoaders("less-loader"),
52           },
53           {
54             test: /\.s[ac]ss$/,
```

```

55     use: getStyleLoaders("sass-loader"),
56   },
57   {
58     test: /\.styl$/,
59     use: getStyleLoaders("stylus-loader"),
60   },
61   {
62     test: /\.?(png|jpe?g|gif|svg)$/,
63     type: "asset",
64     parser: {
65       dataUrlCondition: {
66         maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
67       },
68     },
69     generator: {
70       // 将图片文件输出到 static/imgs 目录中
71       // 将图片文件命名 [hash:8][ext][query]
72       // [hash:8]: hash值取8位
73       // [ext]: 使用之前的文件扩展名
74       // [query]: 添加之前的query参数
75       filename: "static/imgs/[hash:8][ext][query]",
76     },
77   },
78   {
79     test: /\.?(ttf|woff2?)$/,
80     type: "asset/resource",
81     generator: {
82       filename: "static/media/[hash:8][ext][query]",
83     },
84   },
85   {
86     test: /\.js$/,
87     // exclude: /node_modules/, // 排除node_modules代码不编译
88     include: path.resolve(__dirname, "../src"), // 也可以用包含
89     use: [
90       {
91         loader: "thread-loader", // 开启多进程
92         options: {
93           workers: threads, // 数量
94         },
95       },
96       {
97         loader: "babel-loader",
98         options: {
99           cacheDirectory: true, // 开启babel编译缓存
100           cacheCompression: false, // 缓存文件不要压缩
101           plugins: ["@babel/plugin-transform-runtime"], // 减少代码体
102         },
103       },
104     ],

```

积

```

105         },
106     ],
107 },
108 ],
109 },
110 plugins: [
111     new ESLintWebpackPlugin({
112         // 指定检查文件的根目录
113         context: path.resolve(__dirname, "../src"),
114         exclude: "node_modules", // 默认值
115         cache: true, // 开启缓存
116         // 缓存目录
117         cacheLocation: path.resolve(
118             __dirname,
119             "../node_modules/.cache/.eslintcache"
120         ),
121         threads, // 开启多进程
122     }),
123     new HtmlWebpackPlugin({
124         // 以 public/index.html 为模板创建文件
125         // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
126         template: path.resolve(__dirname, "../public/index.html"),
127     }),
128     // 提取css成单独文件
129     new MiniCssExtractPlugin({
130         // 定义输出文件名和目录
131         filename: "static/css/main.css",
132     }),
133     // css压缩
134     // new CssMinimizerPlugin(),
135 ],
136 optimization: {
137     minimizer: [
138         // css压缩也可以写到optimization.minimizer里面, 效果一样的
139         new CssMinimizerPlugin(),
140         // 当生产模式会默认开启TerserPlugin, 但是我们需要进行其他配置, 就要重新写了
141         new TerserPlugin({
142             parallel: threads, // 开启多进程
143         }),
144         // 压缩图片
145         new ImageMinimizerPlugin({
146             minimizer: {
147                 implementation: ImageMinimizerPlugin.imageminGenerate,
148                 options: {
149                     plugins: [
150                         ["gifsicle", { interlaced: true }],
151                         ["jpegtran", { progressive: true }],
152                         ["optipng", { optimizationLevel: 5 }],
153                         [
154                             "svgo",
155                             {

```

```

156         plugins: [
157             "preset-default",
158             "prefixIds",
159             {
160                 name: "sortAttrs",
161                 params: {
162                     xmlnsOrder: "alphabetical",
163                 },
164             },
165         ],
166     },
167 ],
168 ],
169 },
170 },
171 }),
172 ],
173 },
174 // devServer: {
175 //   host: "localhost", // 启动服务器域名
176 //   port: "3000", // 启动服务器端口号
177 //   open: true, // 是否自动打开浏览器
178 // },
179 mode: "production",
180 devtool: "source-map",
181 };

```

3. 打包时会出现报错:

```

1  Error: Error with 'src\images\1.jpeg':
   "C:\Users\86176\Desktop\webpack\webpack_code\node_modules\jpegtran-
   bin\vendor\jpegtran.exe"
2  Error with 'src\images\3.gif': spawn
   C:\Users\86176\Desktop\webpack\webpack_code\node_modules\optipng-
   bin\vendor\optipng.exe ENOENT

```

我们需要安装两个文件到 node\_modules 中才能解决, 文件可以从课件中找到:

- jpegtran.exe

需要复制到 node\_modules\jpegtran-bin\vendor 下面

“

[jpegtran 官网地址](#)

- optipng.exe

需要复制到 `node_modules\optipng-bin\vendor` 下面

“

[OptiPNG 官网地址](#)

## • 优化代码运行性能

### — Code Split

#### 为什么

打包代码时会将所有 js 文件打包到一个文件中，体积太大了。我们如果只要渲染首页，就应该只加载首页的 js 文件，其他文件不应该加载。

所以我们需要将打包生成的文件进行代码分割，生成多个 js 文件，渲染哪个页面就只加载某个 js 文件，这样加载的资源就少，速度就更快。

#### 是什么

代码分割（Code Split）主要做了两件事：

1. 分割文件：将打包生成的文件进行分割，生成多个 js 文件。
2. 按需加载：需要哪个文件就加载哪个文件。

#### 怎么用

代码分割实现方式有不同的方式，为了更加方便体现它们之间的差异，我们会分别创建新的文件来演示

##### 1. 多入口

###### 1. 文件目录

```
1  |— public
2  |— src
3  |   |— app.js
4  |   |— main.js
5  |— package.json
6  |— webpack.config.js
```

###### 2. 下载包

```
1  npm i webpack webpack-cli html-webpack-plugin -D
```

###### 3. 新建文件

内容无关紧要，主要观察打包输出的结果

- app.js

```
1 console.log("hello app");
```

- main.js

```
1 console.log("hello main");
```

#### 4. 配置

```
1 // webpack.config.js
2 const path = require("path");
3 const HtmlWebpackPlugin = require("html-webpack-plugin");
4
5 module.exports = {
6   // 单入口
7   // entry: './src/main.js',
8   // 多入口
9   entry: {
10     main: "./src/main.js",
11     app: "./src/app.js",
12   },
13   output: {
14     path: path.resolve(__dirname, "./dist"),
15     // [name]是webpack命名规则，使用chunk的name作为输出的文件名。
16     // 什么是chunk? 打包的资源就是chunk，输出出去叫bundle。
17     // chunk的name是啥呢? 比如: entry中xxx: "./src/xxx.js", name就是xxx。注意是
    前面的xxx, 和文件名无关。
18     // 为什么需要这样命名呢? 如果还是之前写法main.js, 那么打包生成两个js文件都会叫做
    main.js会发生覆盖。(实际上会直接报错的)
19     filename: "js/[name].js",
20     clear: true,
21   },
22   plugins: [
23     new HtmlWebpackPlugin({
24       template: "./public/index.html",
25     }),
26   ],
27   mode: "production",
28 };
```

#### 5. 运行指令

```
1 npx webpack
```

此时在 dist 目录我们能看到输出了两个 js 文件。

总结：配置了几个入口，至少输出几个 js 文件。

## 2. 提取重复代码

如果多入口文件中都引用了同一份代码，我们不希望这份代码被打包到两个文件中，导致代码重复，体积更大。

我们需要提取多入口的重复代码，只打包生成一个 js 文件，其他文件引用它就好。

### 1. 修改文件

- app.js

```
1 import { sum } from "./math";
2
3 console.log("hello app");
4 console.log(sum(1, 2, 3, 4));
```

- main.js

```
1 import { sum } from "./math";
2
3 console.log("hello main");
4 console.log(sum(1, 2, 3, 4, 5));
```

- math.js

```
1 export const sum = (...args) => {
2   return args.reduce((p, c) => p + c, 0);
3 };
```

### 2. 修改配置文件

```
1 // webpack.config.js
2 const path = require("path");
3 const HtmlWebpackPlugin = require("html-webpack-plugin");
4
5 module.exports = {
6   // 单入口
7   // entry: './src/main.js',
8   // 多入口
9   entry: {
10     main: "./src/main.js",
11     app: "./src/app.js",
12   },
13   output: {
```

```

14     path: path.resolve(__dirname, "./dist"),
15     // [name]是webpack命名规则, 使用chunk的name作为输出的文件名。
16     // 什么是chunk? 打包的资源就是chunk, 输出出去叫bundle。
17     // chunk的name是啥呢? 比如: entry中xxx: "./src/xxx.js", name就是xxx。注意是
    前面的xxx, 和文件名无关。
18     // 为什么需要这样命名呢? 如果还是之前写法main.js, 那么打包生成两个js文件都会叫做
    main.js会发生覆盖。(实际上会直接报错的)
19     filename: "js/[name].js",
20     clean: true,
21 },
22 plugins: [
23     new HtmlWebpackPlugin({
24         template: "./public/index.html",
25     }),
26 ],
27 mode: "production",
28 optimization: {
29     // 代码分割配置
30     splitChunks: {
31         chunks: "all", // 对所有模块都进行分割
32         // 以下是默认值
33         // minSize: 20000, // 分割代码最小的大小, 单位是byte
34         // minRemainingSize: 0, // 类似于minSize, 最后确保提取的文件大小不能为0
35         // minChunks: 1, // 至少被引用的次数, 满足条件才会代码分割
36         // maxAsyncRequests: 30, // 按需加载时并行加载的文件的最大数量
37         // maxInitialRequests: 30, // 入口js文件最大并行请求数量
38         // enforceSizeThreshold: 50000, // 超过50kb一定会单独打包 (此时会忽略
    minRemainingSize、maxAsyncRequests、maxInitialRequests)
39         // cacheGroups: { // 组, 哪些模块要打包到一个组
40             // defaultVendors: { // 组名
41                 // test: /[\\/]node_modules[\\/]/, // 需要打包到一起的模块
42                 // priority: -10, // 权重 (越大越高)
43                 // reuseExistingChunk: true, // 如果当前 chunk 包含已从主 bundle 中拆分
    出的模块, 则它将被重用, 而不是生成新的模块
44             // },
45             // default: { // 其他没有写的配置会使用上面的默认值
46                 // minChunks: 2, // 这里的minChunks权重更大
47                 // priority: -20,
48                 // reuseExistingChunk: true,
49             // },
50         // },
51         // 修改配置
52         cacheGroups: {
53             // 组, 哪些模块要打包到一个组
54             // defaultVendors: { // 组名
55                 // test: /[\\/]node_modules[\\/]/, // 需要打包到一起的模块
56                 // priority: -10, // 权重 (越大越高)
57                 // reuseExistingChunk: true, // 如果当前 chunk 包含已从主 bundle 中拆分
    出的模块, 则它将被重用, 而不是生成新的模块
58             // },
59             default: {

```



```

60         // 其他没有写的配置会使用上面的默认值
61         minSize: 0, // 我们定义的文件体积太小了, 所以要改打包的最小文件体积
62         minChunks: 2,
63         priority: -20,
64         reuseExistingChunk: true,
65     },
66 },
67 },
68 },
69 };

```

### 3. 运行指令

```
1 npx webpack
```

此时我们会发现生成 3 个 js 文件, 其中有一个就是提取的公共模块。

### 3. 按需加载, 动态导入

想要实现按需加载, 动态导入模块。还需要额外配置:

#### 1. 修改文件

- main.js

```

1 console.log("hello main");
2
3 document.getElementById("btn").onclick = function () {
4     // 动态导入 → 实现按需加载
5     // 即使只被引用了一次, 也会代码分割
6     import("./math.js").then(({ sum }) => {
7         alert(sum(1, 2, 3, 4, 5));
8     });
9 };

```

- app.js

```
1 console.log("hello app");
```

- public/index.html

```

1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>Code Split</title>
8    </head>
9    <body>
10     <h1>hello webpack</h1>
11     <button id="btn">计算</button>
12   </body>
13 </html>

```

## 2. 运行指令

```
1  npx webpack
```

我们可以发现，一旦通过 import 动态导入语法导入模块，模块就被代码分割，同时也能按需加载了。

## 4. 单入口

开发时我们可能是单页面应用（SPA），只有一个入口（单入口）。那么我们需要这样配置：

```

1  const path = require("path");
2  const HtmlWebpackPlugin = require("html-webpack-plugin");
3
4  module.exports = {
5    // 单入口
6    entry: "./src/main.js",
7    // 多入口
8    // entry: {
9    //   main: "./src/main.js",
10   //   app: "./src/app.js",
11   // },
12   output: {
13     path: path.resolve(__dirname, "./dist"),
14     // [name]是webpack命名规则，使用chunk的name作为输出的文件名。
15     // 什么是chunk？打包的资源就是chunk，输出出去叫bundle。
16     // chunk的name是啥呢？比如：entry中xxx: "./src/xxx.js", name就是xxx。注意是
    前面的xxx，和文件名无关。
17     // 为什么需要这样命名呢？如果还是之前写法main.js，那么打包生成两个js文件都会叫做
    main.js会发生覆盖。（实际上会直接报错的）
18     filename: "js/[name].js",
19     clean: true,
20   },
21   plugins: [
22     new HtmlWebpackPlugin({

```

```

23     template: "./public/index.html",
24   }),
25 ],
26 mode: "production",
27 optimization: {
28   // 代码分割配置
29   splitChunks: {
30     chunks: "all", // 对所有模块都进行分割
31     // 以下是默认值
32     // minSize: 20000, // 分割代码最小的大小
33     // minRemainingSize: 0, // 类似于minSize, 最后确保提取的文件大小不能为0
34     // minChunks: 1, // 至少被引用的次数, 满足条件才会代码分割
35     // maxAsyncRequests: 30, // 按需加载时并行加载的文件的最大数量
36     // maxInitialRequests: 30, // 入口js文件最大并行请求数量
37     // enforceSizeThreshold: 50000, // 超过50kb一定会单独打包 (此时会忽略
minRemainingSize、maxAsyncRequests、maxInitialRequests)
38     // cacheGroups: { // 组, 哪些模块要打包到一个组
39     //   defaultVendors: { // 组名
40     //     test: /[\\/]node_modules[\\/]/, // 需要打包到一起的模块
41     //     priority: -10, // 权重 (越大越高)
42     //     reuseExistingChunk: true, // 如果当前 chunk 包含已从主 bundle 中拆分
出的模块, 则它将被重用, 而不是生成新的模块
43     //   },
44     //   default: { // 其他没有写的配置会使用上面的默认值
45     //     minChunks: 2, // 这里的minChunks权重更大
46     //     priority: -20,
47     //     reuseExistingChunk: true,
48     //   },
49     // },
50   },
51 };

```

## 5. 更新配置

最终我们会使用单入口+代码分割+动态导入方式来进行配置。更新之前的配置文件。

```

1  // webpack.prod.js
2  const os = require("os");
3  const path = require("path");
4  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
5  const HtmlWebpackPlugin = require("html-webpack-plugin");
6  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
7  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
8  const TerserPlugin = require("terser-webpack-plugin");
9  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
10
11 // cpu核数
12 const threads = os.cpus().length;
13
14 // 获取处理样式的Loaders

```

```
15 const getStyleLoaders = (preProcessor) => {
16   return [
17     MiniCssExtractPlugin.loader,
18     "css-loader",
19     {
20       loader: "postcss-loader",
21       options: {
22         postcssOptions: {
23           plugins: [
24             "postcss-preset-env", // 能解决大多数样式兼容性问题
25           ],
26         },
27       },
28     },
29     preProcessor,
30   ].filter(Boolean);
31 };
32
33 module.exports = {
34   entry: "./src/main.js",
35   output: {
36     path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
37     filename: "static/js/main.js", // 将 js 文件输出到 static/js 目录中
38     clean: true,
39   },
40   module: {
41     rules: [
42       {
43         oneOf: [
44           {
45             // 用来匹配 .css 结尾的文件
46             test: /\.css$/,
47             // use 数组里面 Loader 执行顺序是从右到左
48             use: getStyleLoaders(),
49           },
50           {
51             test: /\.less$/,
52             use: getStyleLoaders("less-loader"),
53           },
54           {
55             test: /\.s[ac]ss$/,
56             use: getStyleLoaders("sass-loader"),
57           },
58           {
59             test: /\.styl$/,
60             use: getStyleLoaders("stylus-loader"),
61           },
62           {
63             test: /\.(png|jpe?g|gif|svg)$/,
64             type: "asset",
65             parser: {
```

```

66         dataUrlCondition: {
67             maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
68         },
69     },
70     generator: {
71         // 将图片文件输出到 static/imgs 目录中
72         // 将图片文件命名 [hash:8][ext][query]
73         // [hash:8]: hash值取8位
74         // [ext]: 使用之前的文件扩展名
75         // [query]: 添加之前的query参数
76         filename: "static/imgs/[hash:8][ext][query]",
77     },
78 },
79 {
80     test: /\.?(ttf|woff2?)$/,
81     type: "asset/resource",
82     generator: {
83         filename: "static/media/[hash:8][ext][query]",
84     },
85 },
86 {
87     test: /\.js$/,
88     // exclude: /node_modules/, // 排除node_modules代码不编译
89     include: path.resolve(__dirname, "../src"), // 也可以用包含
90     use: [
91         {
92             loader: "thread-loader", // 开启多进程
93             options: {
94                 workers: threads, // 数量
95             },
96         },
97         {
98             loader: "babel-loader",
99             options: {
100                 cacheDirectory: true, // 开启babel编译缓存
101                 cacheCompression: false, // 缓存文件不要压缩
102                 plugins: ["@babel/plugin-transform-runtime"], // 减少代码体
103             },
104         },
105     ],
106 },
107 ],
108 },
109 ],
110 },
111 plugins: [
112     new ESLintWebpackPlugin({
113         // 指定检查文件的根目录
114         context: path.resolve(__dirname, "../src"),
115         exclude: "node_modules", // 默认值

```

```
116     cache: true, // 开启缓存
117     // 缓存目录
118     cacheLocation: path.resolve(
119         __dirname,
120         "../node_modules/.cache/.eslintcache"
121     ),
122     threads, // 开启多进程
123 },
124 new HtmlWebpackPlugin({
125     // 以 public/index.html 为模板创建文件
126     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
127     template: path.resolve(__dirname, "../public/index.html"),
128 }),
129 // 提取css成单独文件
130 new MiniCssExtractPlugin({
131     // 定义输出文件名和目录
132     filename: "static/css/main.css",
133 }),
134 // css压缩
135 // new CssMinimizerPlugin(),
136 ],
137 optimization: {
138     minimizer: [
139         // css压缩也可以写到optimization.minimizer里面, 效果一样的
140         new CssMinimizerPlugin(),
141         // 当生产模式会默认开启TerserPlugin, 但是我们需要进行其他配置, 就要重新写了
142         new TerserPlugin({
143             parallel: threads, // 开启多进程
144         }),
145         // 压缩图片
146         new ImageMinimizerPlugin({
147             minimizer: {
148                 implementation: ImageMinimizerPlugin.imageminGenerate,
149                 options: {
150                     plugins: [
151                         ["gifsicle", { interlaced: true }],
152                         ["jpegtran", { progressive: true }],
153                         ["optipng", { optimizationLevel: 5 }],
154                         [
155                             "svgo",
156                             {
157                                 plugins: [
158                                     "preset-default",
159                                     "prefixIds",
160                                     {
161                                         name: "sortAttrs",
162                                         params: {
163                                             xmlnsOrder: "alphabetical",
164                                         },
165                                     },
166                                 ],
167                             },
168                         ],
169                     ],
170                 },
171             },
172         }),
173     ],
174 },
175 },
```

```

167         },
168     ],
169 ],
170 },
171 },
172 }),
173 ],
174 // 代码分割配置
175 splitChunks: {
176     chunks: "all", // 对所有模块都进行分割
177     // 其他内容用默认配置即可
178 },
179 },
180 // devServer: {
181 //     host: "localhost", // 启动服务器域名
182 //     port: "3000", // 启动服务器端口号
183 //     open: true, // 是否自动打开浏览器
184 // },
185 mode: "production",
186 devtool: "source-map",
187 };

```

## 6. 给动态导入文件取名称

### 1. 修改文件

- main.js

```

1  import sum from "./js/sum";
2  // 引入资源, Webpack才会对其打包
3  import "./css/iconfont.css";
4  import "./css/index.css";
5  import "./less/index.less";
6  import "./sass/index.sass";
7  import "./sass/index.scss";
8  import "./styl/index.styl";
9
10 const result2 = sum(1, 2, 3, 4);
11 console.log(result2);
12
13 // 以下代码生产模式下会删除
14 if (module.hot) {
15     module.hot.accept("./js/sum.js", function (sum) {
16         const result2 = sum(1, 2, 3, 4);
17         console.log(result2);
18     });
19 }
20
21 document.getElementById("btn").onClick = function () {
22     // eslint会对动态导入语法报错, 需要修改eslint配置文件

```

```

23     // webpackChunkName: "math": 这是webpack动态导入模块命名的方式
24     // "math"将来就会作为[name]的值显示。
25     import(/* webpackChunkName: "math" */ "./js/math.js").then(({ count }) => {
26         console.log(count(2, 1));
27     });
28 };

```

## 2. eslint 配置

- 下载包

```
1  npm i eslint-plugin-import -D
```

- 配置

```

1  // .eslintrc.js
2  module.exports = {
3      // 继承 ESLint 规则
4      extends: ["eslint:recommended"],
5      env: {
6          node: true, // 启用node中全局变量
7          browser: true, // 启用浏览器中全局变量
8      },
9      plugins: ["import"], // 解决动态导入import语法报错问题 → 实际使用eslint-
      plugin-import的规则解决的
10     parserOptions: {
11         ecmaVersion: 6,
12         sourceType: "module",
13     },
14     rules: {
15         "no-var": 2, // 不能使用 var 定义变量
16     },
17 };

```

## 1. 统一命名配置

```

1  const os = require("os");
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
7  const TerserPlugin = require("terser-webpack-plugin");
8  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
9
10 // cpu核数

```



```
11  const threads = os.cpus().length;
12
13  // 获取处理样式的Loaders
14  const getStyleLoaders = (preProcessor) => {
15    return [
16      MiniCssExtractPlugin.loader,
17      "css-loader",
18      {
19        loader: "postcss-loader",
20        options: {
21          postcssOptions: {
22            plugins: [
23              "postcss-preset-env", // 能解决大多数样式兼容性问题
24            ],
25          },
26        },
27      },
28      preProcessor,
29    ].filter(Boolean);
30  };
31
32  module.exports = {
33    entry: "./src/main.js",
34    output: {
35      path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
36      filename: "static/js/[name].js", // 入口文件打包输出资源命名方式
37      chunkFilename: "static/js/[name].chunk.js", // 动态导入输出资源命名方式
38      assetModuleFilename: "static/media/[name].[hash][ext]", // 图片、字体等资
源命名方式 (注意用hash)
39      clean: true,
40    },
41    module: {
42      rules: [
43        {
44          oneOf: [
45            {
46              // 用来匹配 .css 结尾的文件
47              test: /\.css$/,
48              // use 数组里面 Loader 执行顺序是从右到左
49              use: getStyleLoaders(),
50            },
51            {
52              test: /\.less$/,
53              use: getStyleLoaders("less-loader"),
54            },
55            {
56              test: /\.s[ac]ss$/,
57              use: getStyleLoaders("sass-loader"),
58            },
59            {
60              test: /\.styl$/,
```

```

61     use: getStyleLoaders("stylus-loader"),
62   },
63   {
64     test: /\.?(png|jpe?g|gif|svg)$/,
65     type: "asset",
66     parser: {
67       dataUrlCondition: {
68         maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
69       },
70     },
71     // generator: {
72     //   // 将图片文件输出到 static/imgs 目录中
73     //   // 将图片文件命名 [hash:8][ext][query]
74     //   // [hash:8]: hash值取8位
75     //   // [ext]: 使用之前的文件扩展名
76     //   // [query]: 添加之前的query参数
77     //   filename: "static/imgs/[hash:8][ext][query]",
78     // },
79   },
80   {
81     test: /\.?(ttf|woff2?)$/,
82     type: "asset/resource",
83     // generator: {
84     //   filename: "static/media/[hash:8][ext][query]",
85     // },
86   },
87   {
88     test: /\.js$/,
89     // exclude: /node_modules/, // 排除node_modules代码不编译
90     include: path.resolve(__dirname, "../src"), // 也可以用包含
91     use: [
92       {
93         loader: "thread-loader", // 开启多进程
94         options: {
95           workers: threads, // 数量
96         },
97       },
98       {
99         loader: "babel-loader",
100        options: {
101          cacheDirectory: true, // 开启babel编译缓存
102          cacheCompression: false, // 缓存文件不要压缩
103          plugins: ["@babel/plugin-transform-runtime"], // 减少代码体
104        },
105      },
106    ],
107  },
108 ],
109 },
110 ],

```

积

```
111 },
112 plugins: [
113   new ESLintWebpackPlugin({
114     // 指定检查文件的根目录
115     context: path.resolve(__dirname, "../src"),
116     exclude: "node_modules", // 默认值
117     cache: true, // 开启缓存
118     // 缓存目录
119     cacheLocation: path.resolve(
120       __dirname,
121       "../node_modules/.cache/.eslintcache"
122     ),
123     threads, // 开启多进程
124   }),
125   new HtmlWebpackPlugin({
126     // 以 public/index.html 为模板创建文件
127     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
128     template: path.resolve(__dirname, "../public/index.html"),
129   }),
130   // 提取css成单独文件
131   new MiniCssExtractPlugin({
132     // 定义输出文件名和目录
133     filename: "static/css/[name].css",
134     chunkFilename: "static/css/[name].chunk.css",
135   }),
136   // css压缩
137   // new CssMinimizerPlugin(),
138 ],
139 optimization: {
140   minimizer: [
141     // css压缩也可以写到optimization.minimizer里面, 效果一样的
142     new CssMinimizerPlugin(),
143     // 当生产模式会默认开启TerserPlugin, 但是我们需要进行其他配置, 就要重新写了
144     new TerserPlugin({
145       parallel: threads, // 开启多进程
146     }),
147     // 压缩图片
148     new ImageMinimizerPlugin({
149       minimizer: {
150         implementation: ImageMinimizerPlugin.imageminGenerate,
151         options: {
152           plugins: [
153             ["gifsicle", { interlaced: true }],
154             ["jpegtran", { progressive: true }],
155             ["optipng", { optimizationLevel: 5 }],
156             [
157               "svgo",
158               {
159                 plugins: [
160                   "preset-default",
161                   "prefixIds",
```

```

162         {
163             name: "sortAttrs",
164             params: {
165                 xmlnsOrder: "alphabetical",
166             },
167         },
168     ],
169 },
170 ],
171 ],
172 },
173 },
174 }),
175 ],
176 // 代码分割配置
177 splitChunks: {
178     chunks: "all", // 对所有模块都进行分割
179     // 其他内容用默认配置即可
180 },
181 },
182 // devServer: {
183 //     host: "localhost", // 启动服务器域名
184 //     port: "3000", // 启动服务器端口号
185 //     open: true, // 是否自动打开浏览器
186 // },
187 mode: "production",
188 devtool: "source-map",
189 };

```

### 3. 运行指令

```
1 npx webpack
```

观察打包输出 js 文件名称。

#### - Preload / Prefetch

##### 为什么

我们前面已经做了代码分割，同时会使用 import 动态导入语法来进行代码按需加载（我们也叫懒加载，比如路由懒加载就是这样实现的）。

但是加载速度还不够好，比如：是用户点击按钮时才加载这个资源的，如果资源体积很大，那么用户会感觉到明显卡顿效果。

我们想在浏览器空闲时间，加载后续需要使用的资源。我们就需要用上 `Preload` 或 `Prefetch` 技术。

## 是什么

- `Preload`：告诉浏览器立即加载资源。
- `Prefetch`：告诉浏览器在空闲时才开始加载资源。

它们共同点：

- 都只会加载资源，并不执行。
- 都有缓存。

它们区别：

- `Preload` 加载优先级高，`Prefetch` 加载优先级低。
- `Preload` 只能加载当前页面需要使用的资源，`Prefetch` 可以加载当前页面资源，也可以加载下一个页面需要使用的资源。

总结：

- 当前页面优先级高的资源用 `Preload` 加载。
- 下一个页面需要使用的资源用 `Prefetch` 加载。

它们的问题：兼容性较差。

- 我们可以去 [Can I Use](#) 网站查询 API 的兼容性问题。
- `Preload` 相对于 `Prefetch` 兼容性好一点。

## 怎么用

### 1. 下载包

```
1 npm i @vue/preload-webpack-plugin -D
```

### 2. 配置 webpack.prod.js

```
1  const os = require("os");
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
7  const TerserPlugin = require("terser-webpack-plugin");
8  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
9  const PreloadWebpackPlugin = require("@vue/preload-webpack-plugin");
10
11  // cpu核数
12  const threads = os.cpus().length;
13
14  // 获取处理样式的Loaders
15  const getStyleLoaders = (preProcessor) => {
16    return [
```

```
17     MiniCssExtractPlugin.loader,
18     "css-loader",
19     {
20       loader: "postcss-loader",
21       options: {
22         postcssOptions: {
23           plugins: [
24             "postcss-preset-env", // 能解决大多数样式兼容性问题
25           ],
26         },
27       },
28     },
29     preProcessor,
30   ].filter(Boolean);
31 };
32
33 module.exports = {
34   entry: "./src/main.js",
35   output: {
36     path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
37     filename: "static/js/[name].js", // 入口文件打包输出资源命名方式
38     chunkFilename: "static/js/[name].chunk.js", // 动态导入输出资源命名方式
39     assetModuleFilename: "static/media/[name].[hash][ext]", // 图片、字体等资
源命名方式 (注意用hash)
40     clean: true,
41   },
42   module: {
43     rules: [
44       {
45         oneOf: [
46           {
47             // 用来匹配 .css 结尾的文件
48             test: /\.css$/,
49             // use 数组里面 Loader 执行顺序是从右到左
50             use: getStyleLoaders(),
51           },
52           {
53             test: /\.less$/,
54             use: getStyleLoaders("less-loader"),
55           },
56           {
57             test: /\.s[ac]ss$/,
58             use: getStyleLoaders("sass-loader"),
59           },
60           {
61             test: /\.styl$/,
62             use: getStyleLoaders("stylus-loader"),
63           },
64           {
65             test: /\.?(png|jpe?g|gif|svg)$/,
66             type: "asset",
```

```

67         parser: {
68             dataUrlCondition: {
69                 maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
70             },
71         },
72         // generator: {
73         //     // 将图片文件输出到 static/imgs 目录中
74         //     // 将图片文件命名 [hash:8][ext][query]
75         //     // [hash:8]: hash值取8位
76         //     // [ext]: 使用之前的文件扩展名
77         //     // [query]: 添加之前的query参数
78         //     filename: "static/imgs/[hash:8][ext][query]",
79         // },
80     },
81     {
82         test: /\.?(ttf|woff2?)$/,
83         type: "asset/resource",
84         // generator: {
85         //     filename: "static/media/[hash:8][ext][query]",
86         // },
87     },
88     {
89         test: /\.js$/,
90         // exclude: /node_modules/, // 排除node_modules代码不编译
91         include: path.resolve(__dirname, "../src"), // 也可以用包含
92         use: [
93             {
94                 loader: "thread-loader", // 开启多进程
95                 options: {
96                     workers: threads, // 数量
97                 },
98             },
99             {
100                 loader: "babel-loader",
101                 options: {
102                     cacheDirectory: true, // 开启babel编译缓存
103                     cacheCompression: false, // 缓存文件不要压缩
104                     plugins: ["@babel/plugin-transform-runtime"], // 减少代码体
105                 },
106             },
107         ],
108     },
109 ],
110 },
111 ],
112 },
113 plugins: [
114     new ESLintWebpackPlugin({
115         // 指定检查文件的根目录
116         context: path.resolve(__dirname, "../src"),

```

```
117     exclude: "node_modules", // 默认值
118     cache: true, // 开启缓存
119     // 缓存目录
120     cacheLocation: path.resolve(
121         __dirname,
122         "../node_modules/.cache/.eslintcache"
123     ),
124     threads, // 开启多进程
125 },
126 new HtmlWebpackPlugin({
127     // 以 public/index.html 为模板创建文件
128     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
129     template: path.resolve(__dirname, "../public/index.html"),
130 }),
131 // 提取css成单独文件
132 new MiniCssExtractPlugin({
133     // 定义输出文件名和目录
134     filename: "static/css/[name].css",
135     chunkFilename: "static/css/[name].chunk.css",
136 }),
137 // css压缩
138 // new CssMinimizerPlugin(),
139 new PreloadWebpackPlugin({
140     rel: "preload", // preload兼容性更好
141     as: "script",
142     // rel: 'prefetch' // prefetch兼容性更差
143 }),
144 ],
145 optimization: {
146     minimizer: [
147         // css压缩也可以写到optimization.minimizer里面, 效果一样的
148         new CssMinimizerPlugin(),
149         // 当生产模式会默认开启TerserPlugin, 但是我们需要进行其他配置, 就要重新写了
150         new TerserPlugin({
151             parallel: threads, // 开启多进程
152         }),
153         // 压缩图片
154         new ImageMinimizerPlugin({
155             minimizer: {
156                 implementation: ImageMinimizerPlugin.imageminGenerate,
157                 options: {
158                     plugins: [
159                         ["gifsicle", { interlaced: true }],
160                         ["jpegtran", { progressive: true }],
161                         ["optipng", { optimizationLevel: 5 }],
162                     ],
163                     "svgo",
164                     {
165                         plugins: [
166                             "preset-default",
167                             "prefixIds",
```



```

168         {
169             name: "sortAttrs",
170             params: {
171                 xmlnsOrder: "alphabetical",
172             },
173         },
174     ],
175 },
176 ],
177 ],
178 },
179 },
180 }),
181 ],
182 // 代码分割配置
183 splitChunks: {
184     chunks: "all", // 对所有模块都进行分割
185     // 其他内容用默认配置即可
186 },
187 },
188 // devServer: {
189 //     host: "localhost", // 启动服务器域名
190 //     port: "3000", // 启动服务器端口号
191 //     open: true, // 是否自动打开浏览器
192 // },
193 mode: "production",
194 devtool: "source-map",
195 };

```

## — Network Cache

### 为什么

将来开发时我们对静态资源会使用缓存来优化，这样浏览器第二次请求资源就能读取缓存了，速度很快。

但是这样的话就会有一个问题，因为前后输出的文件名是一样的，都叫 main.js，一旦将来发布新版本，因为文件名没有变化导致浏览器会直接读取缓存，不会加载新资源，项目也就没法更新了。

所以我们从文件名入手，确保更新前后文件名不一样，这样就可以做缓存了。

### 是什么

它们都会生成一个唯一的 hash 值。

- fullhash (webpack4 是 hash)

每次修改任何一个文件，所有文件名的 hash 至都将改变。所以一旦修改了任何一个文件，整个项目的文件缓存都将失效。

- chunkhash

根据不同的入口文件(Entry)进行依赖文件解析、构建对应的 chunk，生成对应的哈希值。我们 js 和 css 是同一个引入，会共享一个 hash 值。

- contenthash

根据文件内容生成 hash 值，只有文件内容变化了，hash 值才会变化。所有文件 hash 值是独享且不同的。

## 怎么用

```
1  const os = require("os");
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
7  const TerserPlugin = require("terser-webpack-plugin");
8  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
9  const PreloadWebpackPlugin = require("@vue/preload-webpack-plugin");
10
11  // cpu核数
12  const threads = os.cpus().length;
13
14  // 获取处理样式的Loaders
15  const getStyleLoaders = (preProcessor) => {
16    return [
17      MiniCssExtractPlugin.loader,
18      "css-loader",
19      {
20        loader: "postcss-loader",
21        options: {
22          postcssOptions: {
23            plugins: [
24              "postcss-preset-env", // 能解决大多数样式兼容性问题
25            ],
26          },
27        },
28      },
29      preProcessor,
30    ].filter(Boolean);
31  };
32
33  module.exports = {
34    entry: "./src/main.js",
35    output: {
36      path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
37      // [contenthash:8]使用contenthash, 取8位长度
38      filename: "static/js/[name].[contenthash:8].js", // 入口文件打包输出资源命名方式
39      chunkFilename: "static/js/[name].[contenthash:8].chunk.js", // 动态导入输出资源命名方式
40    },
41  };
42
43  // 生产模式打包
44  module.exports = {
45    mode: "production",
46    entry: "./src/main.js",
47    output: {
48      path: path.resolve(__dirname, "../dist"),
49      filename: "static/js/[name].[contenthash:8].js",
50      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
51    },
52    plugins: [
53      new ESLintWebpackPlugin(),
54      new HtmlWebpackPlugin(),
55      new MiniCssExtractPlugin({
56        // 生产模式打包
57        extract: true,
58      }),
59      new TerserPlugin({
60        // 生产模式打包
61        terserOptions: {
62          compress: {
63            // 生产模式打包
64            drop_console: true,
65            drop_debugger: true,
66          },
67        },
68      }),
69      new ImageMinimizerPlugin({
70        // 生产模式打包
71        minimizerOptions: {
72          // 生产模式打包
73          removeMetadata: true,
74          // 生产模式打包
75          removeSizeAttribute: true,
76        },
77      }),
78      new PreloadWebpackPlugin(),
79    ],
80    optimization: {
81      // 生产模式打包
82      minimize: true,
83      // 生产模式打包
84      minimizer: [
85        new CssMinimizerPlugin(),
86      ],
87    },
88  };
89
90  // 生产模式打包
91  module.exports = {
92    mode: "production",
93    entry: "./src/main.js",
94    output: {
95      path: path.resolve(__dirname, "../dist"),
96      filename: "static/js/[name].[contenthash:8].js",
97      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
98    },
99    plugins: [
100      new ESLintWebpackPlugin(),
101      new HtmlWebpackPlugin(),
102      new MiniCssExtractPlugin({
103        // 生产模式打包
104        extract: true,
105      }),
106      new TerserPlugin({
107        // 生产模式打包
108        terserOptions: {
109          compress: {
110            // 生产模式打包
111            drop_console: true,
112            drop_debugger: true,
113          },
114        },
115      }),
116      new ImageMinimizerPlugin({
117        // 生产模式打包
118        minimizerOptions: {
119          // 生产模式打包
120          removeMetadata: true,
121          // 生产模式打包
122          removeSizeAttribute: true,
123        },
124      }),
125      new PreloadWebpackPlugin(),
126    ],
127    optimization: {
128      // 生产模式打包
129      minimize: true,
130      // 生产模式打包
131      minimizer: [
132        new CssMinimizerPlugin(),
133      ],
134    },
135  };
136
137  // 生产模式打包
138  module.exports = {
139    mode: "production",
140    entry: "./src/main.js",
141    output: {
142      path: path.resolve(__dirname, "../dist"),
143      filename: "static/js/[name].[contenthash:8].js",
144      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
145    },
146    plugins: [
147      new ESLintWebpackPlugin(),
148      new HtmlWebpackPlugin(),
149      new MiniCssExtractPlugin({
150        // 生产模式打包
151        extract: true,
152      }),
153      new TerserPlugin({
154        // 生产模式打包
155        terserOptions: {
156          compress: {
157            // 生产模式打包
158            drop_console: true,
159            drop_debugger: true,
160          },
161        },
162      }),
163      new ImageMinimizerPlugin({
164        // 生产模式打包
165        minimizerOptions: {
166          // 生产模式打包
167          removeMetadata: true,
168          // 生产模式打包
169          removeSizeAttribute: true,
170        },
171      }),
172      new PreloadWebpackPlugin(),
173    ],
174    optimization: {
175      // 生产模式打包
176      minimize: true,
177      // 生产模式打包
178      minimizer: [
179        new CssMinimizerPlugin(),
180      ],
181    },
182  };
183
184  // 生产模式打包
185  module.exports = {
186    mode: "production",
187    entry: "./src/main.js",
188    output: {
189      path: path.resolve(__dirname, "../dist"),
190      filename: "static/js/[name].[contenthash:8].js",
191      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
192    },
193    plugins: [
194      new ESLintWebpackPlugin(),
195      new HtmlWebpackPlugin(),
196      new MiniCssExtractPlugin({
197        // 生产模式打包
198        extract: true,
199      }),
200      new TerserPlugin({
201        // 生产模式打包
202        terserOptions: {
203          compress: {
204            // 生产模式打包
205            drop_console: true,
206            drop_debugger: true,
207          },
208        },
209      }),
210      new ImageMinimizerPlugin({
211        // 生产模式打包
212        minimizerOptions: {
213          // 生产模式打包
214          removeMetadata: true,
215          // 生产模式打包
216          removeSizeAttribute: true,
217        },
218      }),
219      new PreloadWebpackPlugin(),
220    ],
221    optimization: {
222      // 生产模式打包
223      minimize: true,
224      // 生产模式打包
225      minimizer: [
226        new CssMinimizerPlugin(),
227      ],
228    },
229  };
230
231  // 生产模式打包
232  module.exports = {
233    mode: "production",
234    entry: "./src/main.js",
235    output: {
236      path: path.resolve(__dirname, "../dist"),
237      filename: "static/js/[name].[contenthash:8].js",
238      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
239    },
240    plugins: [
241      new ESLintWebpackPlugin(),
242      new HtmlWebpackPlugin(),
243      new MiniCssExtractPlugin({
244        // 生产模式打包
245        extract: true,
246      }),
247      new TerserPlugin({
248        // 生产模式打包
249        terserOptions: {
250          compress: {
251            // 生产模式打包
252            drop_console: true,
253            drop_debugger: true,
254          },
255        },
256      }),
257      new ImageMinimizerPlugin({
258        // 生产模式打包
259        minimizerOptions: {
260          // 生产模式打包
261          removeMetadata: true,
262          // 生产模式打包
263          removeSizeAttribute: true,
264        },
265      }),
266      new PreloadWebpackPlugin(),
267    ],
268    optimization: {
269      // 生产模式打包
270      minimize: true,
271      // 生产模式打包
272      minimizer: [
273        new CssMinimizerPlugin(),
274      ],
275    },
276  };
277
278  // 生产模式打包
279  module.exports = {
280    mode: "production",
281    entry: "./src/main.js",
282    output: {
283      path: path.resolve(__dirname, "../dist"),
284      filename: "static/js/[name].[contenthash:8].js",
285      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
286    },
287    plugins: [
288      new ESLintWebpackPlugin(),
289      new HtmlWebpackPlugin(),
290      new MiniCssExtractPlugin({
291        // 生产模式打包
292        extract: true,
293      }),
294      new TerserPlugin({
295        // 生产模式打包
296        terserOptions: {
297          compress: {
298            // 生产模式打包
299            drop_console: true,
300            drop_debugger: true,
301          },
302        },
303      }),
304      new ImageMinimizerPlugin({
305        // 生产模式打包
306        minimizerOptions: {
307          // 生产模式打包
308          removeMetadata: true,
309          // 生产模式打包
310          removeSizeAttribute: true,
311        },
312      }),
313      new PreloadWebpackPlugin(),
314    ],
315    optimization: {
316      // 生产模式打包
317      minimize: true,
318      // 生产模式打包
319      minimizer: [
320        new CssMinimizerPlugin(),
321      ],
322    },
323  };
324
325  // 生产模式打包
326  module.exports = {
327    mode: "production",
328    entry: "./src/main.js",
329    output: {
330      path: path.resolve(__dirname, "../dist"),
331      filename: "static/js/[name].[contenthash:8].js",
332      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
333    },
334    plugins: [
335      new ESLintWebpackPlugin(),
336      new HtmlWebpackPlugin(),
337      new MiniCssExtractPlugin({
338        // 生产模式打包
339        extract: true,
340      }),
341      new TerserPlugin({
342        // 生产模式打包
343        terserOptions: {
344          compress: {
345            // 生产模式打包
346            drop_console: true,
347            drop_debugger: true,
348          },
349        },
350      }),
351      new ImageMinimizerPlugin({
352        // 生产模式打包
353        minimizerOptions: {
354          // 生产模式打包
355          removeMetadata: true,
356          // 生产模式打包
357          removeSizeAttribute: true,
358        },
359      }),
360      new PreloadWebpackPlugin(),
361    ],
362    optimization: {
363      // 生产模式打包
364      minimize: true,
365      // 生产模式打包
366      minimizer: [
367        new CssMinimizerPlugin(),
368      ],
369    },
370  };
371
372  // 生产模式打包
373  module.exports = {
374    mode: "production",
375    entry: "./src/main.js",
376    output: {
377      path: path.resolve(__dirname, "../dist"),
378      filename: "static/js/[name].[contenthash:8].js",
379      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
380    },
381    plugins: [
382      new ESLintWebpackPlugin(),
383      new HtmlWebpackPlugin(),
384      new MiniCssExtractPlugin({
385        // 生产模式打包
386        extract: true,
387      }),
388      new TerserPlugin({
389        // 生产模式打包
390        terserOptions: {
391          compress: {
392            // 生产模式打包
393            drop_console: true,
394            drop_debugger: true,
395          },
396        },
397      }),
398      new ImageMinimizerPlugin({
399        // 生产模式打包
400        minimizerOptions: {
401          // 生产模式打包
402          removeMetadata: true,
403          // 生产模式打包
404          removeSizeAttribute: true,
405        },
406      }),
407      new PreloadWebpackPlugin(),
408    ],
409    optimization: {
410      // 生产模式打包
411      minimize: true,
412      // 生产模式打包
413      minimizer: [
414        new CssMinimizerPlugin(),
415      ],
416    },
417  };
418
419  // 生产模式打包
420  module.exports = {
421    mode: "production",
422    entry: "./src/main.js",
423    output: {
424      path: path.resolve(__dirname, "../dist"),
425      filename: "static/js/[name].[contenthash:8].js",
426      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
427    },
428    plugins: [
429      new ESLintWebpackPlugin(),
430      new HtmlWebpackPlugin(),
431      new MiniCssExtractPlugin({
432        // 生产模式打包
433        extract: true,
434      }),
435      new TerserPlugin({
436        // 生产模式打包
437        terserOptions: {
438          compress: {
439            // 生产模式打包
440            drop_console: true,
441            drop_debugger: true,
442          },
443        },
444      }),
445      new ImageMinimizerPlugin({
446        // 生产模式打包
447        minimizerOptions: {
448          // 生产模式打包
449          removeMetadata: true,
450          // 生产模式打包
451          removeSizeAttribute: true,
452        },
453      }),
454      new PreloadWebpackPlugin(),
455    ],
456    optimization: {
457      // 生产模式打包
458      minimize: true,
459      // 生产模式打包
460      minimizer: [
461        new CssMinimizerPlugin(),
462      ],
463    },
464  };
465
466  // 生产模式打包
467  module.exports = {
468    mode: "production",
469    entry: "./src/main.js",
470    output: {
471      path: path.resolve(__dirname, "../dist"),
472      filename: "static/js/[name].[contenthash:8].js",
473      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
474    },
475    plugins: [
476      new ESLintWebpackPlugin(),
477      new HtmlWebpackPlugin(),
478      new MiniCssExtractPlugin({
479        // 生产模式打包
480        extract: true,
481      }),
479      new TerserPlugin({
480        // 生产模式打包
481        terserOptions: {
482          compress: {
483            // 生产模式打包
484            drop_console: true,
485            drop_debugger: true,
486          },
487        },
488      }),
489      new ImageMinimizerPlugin({
490        // 生产模式打包
491        minimizerOptions: {
492          // 生产模式打包
493          removeMetadata: true,
494          // 生产模式打包
495          removeSizeAttribute: true,
496        },
497      }),
498      new PreloadWebpackPlugin(),
499    ],
500    optimization: {
501      // 生产模式打包
502      minimize: true,
503      // 生产模式打包
504      minimizer: [
505        new CssMinimizerPlugin(),
506      ],
507    },
508  };
509
510  // 生产模式打包
511  module.exports = {
512    mode: "production",
513    entry: "./src/main.js",
514    output: {
515      path: path.resolve(__dirname, "../dist"),
516      filename: "static/js/[name].[contenthash:8].js",
517      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
518    },
519    plugins: [
520      new ESLintWebpackPlugin(),
521      new HtmlWebpackPlugin(),
522      new MiniCssExtractPlugin({
523        // 生产模式打包
524        extract: true,
525      }),
526      new TerserPlugin({
527        // 生产模式打包
528        terserOptions: {
529          compress: {
530            // 生产模式打包
531            drop_console: true,
532            drop_debugger: true,
533          },
534        },
535      }),
536      new ImageMinimizerPlugin({
537        // 生产模式打包
538        minimizerOptions: {
539          // 生产模式打包
540          removeMetadata: true,
541          // 生产模式打包
542          removeSizeAttribute: true,
543        },
544      }),
545      new PreloadWebpackPlugin(),
546    ],
547    optimization: {
548      // 生产模式打包
549      minimize: true,
550      // 生产模式打包
551      minimizer: [
552        new CssMinimizerPlugin(),
553      ],
554    },
555  };
556
557  // 生产模式打包
558  module.exports = {
559    mode: "production",
560    entry: "./src/main.js",
561    output: {
562      path: path.resolve(__dirname, "../dist"),
563      filename: "static/js/[name].[contenthash:8].js",
564      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
565    },
566    plugins: [
567      new ESLintWebpackPlugin(),
568      new HtmlWebpackPlugin(),
569      new MiniCssExtractPlugin({
570        // 生产模式打包
571        extract: true,
572      }),
573      new TerserPlugin({
574        // 生产模式打包
575        terserOptions: {
576          compress: {
577            // 生产模式打包
578            drop_console: true,
579            drop_debugger: true,
580          },
581        },
582      }),
583      new ImageMinimizerPlugin({
584        // 生产模式打包
585        minimizerOptions: {
586          // 生产模式打包
587          removeMetadata: true,
588          // 生产模式打包
589          removeSizeAttribute: true,
590        },
591      }),
592      new PreloadWebpackPlugin(),
593    ],
594    optimization: {
595      // 生产模式打包
596      minimize: true,
597      // 生产模式打包
598      minimizer: [
599        new CssMinimizerPlugin(),
600      ],
601    },
602  };
603
604  // 生产模式打包
605  module.exports = {
606    mode: "production",
607    entry: "./src/main.js",
608    output: {
609      path: path.resolve(__dirname, "../dist"),
610      filename: "static/js/[name].[contenthash:8].js",
611      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
612    },
613    plugins: [
614      new ESLintWebpackPlugin(),
615      new HtmlWebpackPlugin(),
616      new MiniCssExtractPlugin({
617        // 生产模式打包
618        extract: true,
619      }),
620      new TerserPlugin({
621        // 生产模式打包
622        terserOptions: {
623          compress: {
624            // 生产模式打包
625            drop_console: true,
626            drop_debugger: true,
627          },
628        },
629      }),
630      new ImageMinimizerPlugin({
631        // 生产模式打包
632        minimizerOptions: {
633          // 生产模式打包
634          removeMetadata: true,
635          // 生产模式打包
636          removeSizeAttribute: true,
637        },
638      }),
639      new PreloadWebpackPlugin(),
640    ],
641    optimization: {
642      // 生产模式打包
643      minimize: true,
644      // 生产模式打包
645      minimizer: [
646        new CssMinimizerPlugin(),
647      ],
648    },
649  };
650
651  // 生产模式打包
652  module.exports = {
653    mode: "production",
654    entry: "./src/main.js",
655    output: {
656      path: path.resolve(__dirname, "../dist"),
657      filename: "static/js/[name].[contenthash:8].js",
658      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
659    },
660    plugins: [
661      new ESLintWebpackPlugin(),
662      new HtmlWebpackPlugin(),
663      new MiniCssExtractPlugin({
664        // 生产模式打包
665        extract: true,
666      }),
667      new TerserPlugin({
668        // 生产模式打包
669        terserOptions: {
670          compress: {
671            // 生产模式打包
672            drop_console: true,
673            drop_debugger: true,
674          },
675        },
676      }),
677      new ImageMinimizerPlugin({
678        // 生产模式打包
679        minimizerOptions: {
680          // 生产模式打包
681          removeMetadata: true,
682          // 生产模式打包
683          removeSizeAttribute: true,
684        },
685      }),
686      new PreloadWebpackPlugin(),
687    ],
688    optimization: {
689      // 生产模式打包
690      minimize: true,
691      // 生产模式打包
692      minimizer: [
693        new CssMinimizerPlugin(),
694      ],
695    },
696  };
697
698  // 生产模式打包
699  module.exports = {
700    mode: "production",
701    entry: "./src/main.js",
702    output: {
703      path: path.resolve(__dirname, "../dist"),
704      filename: "static/js/[name].[contenthash:8].js",
705      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
706    },
707    plugins: [
708      new ESLintWebpackPlugin(),
709      new HtmlWebpackPlugin(),
710      new MiniCssExtractPlugin({
711        // 生产模式打包
712        extract: true,
713      }),
714      new TerserPlugin({
715        // 生产模式打包
716        terserOptions: {
717          compress: {
718            // 生产模式打包
719            drop_console: true,
720            drop_debugger: true,
721          },
722        },
723      }),
724      new ImageMinimizerPlugin({
725        // 生产模式打包
726        minimizerOptions: {
727          // 生产模式打包
728          removeMetadata: true,
729          // 生产模式打包
730          removeSizeAttribute: true,
731        },
732      }),
733      new PreloadWebpackPlugin(),
734    ],
735    optimization: {
736      // 生产模式打包
737      minimize: true,
738      // 生产模式打包
739      minimizer: [
740        new CssMinimizerPlugin(),
741      ],
742    },
743  };
744
745  // 生产模式打包
746  module.exports = {
747    mode: "production",
748    entry: "./src/main.js",
749    output: {
750      path: path.resolve(__dirname, "../dist"),
751      filename: "static/js/[name].[contenthash:8].js",
752      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
753    },
754    plugins: [
755      new ESLintWebpackPlugin(),
756      new HtmlWebpackPlugin(),
757      new MiniCssExtractPlugin({
758        // 生产模式打包
759        extract: true,
760      }),
759      new TerserPlugin({
760        // 生产模式打包
761        terserOptions: {
762          compress: {
763            // 生产模式打包
764            drop_console: true,
765            drop_debugger: true,
766          },
767        },
768      }),
769      new ImageMinimizerPlugin({
770        // 生产模式打包
771        minimizerOptions: {
772          // 生产模式打包
773          removeMetadata: true,
774          // 生产模式打包
775          removeSizeAttribute: true,
776        },
777      }),
778      new PreloadWebpackPlugin(),
779    ],
780    optimization: {
781      // 生产模式打包
782      minimize: true,
783      // 生产模式打包
784      minimizer: [
785        new CssMinimizerPlugin(),
786      ],
787    },
788  };
789
790  // 生产模式打包
791  module.exports = {
792    mode: "production",
793    entry: "./src/main.js",
794    output: {
795      path: path.resolve(__dirname, "../dist"),
796      filename: "static/js/[name].[contenthash:8].js",
797      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
798    },
799    plugins: [
800      new ESLintWebpackPlugin(),
801      new HtmlWebpackPlugin(),
802      new MiniCssExtractPlugin({
803        // 生产模式打包
804        extract: true,
805      }),
806      new TerserPlugin({
807        // 生产模式打包
808        terserOptions: {
809          compress: {
810            // 生产模式打包
811            drop_console: true,
812            drop_debugger: true,
813          },
814        },
815      }),
816      new ImageMinimizerPlugin({
817        // 生产模式打包
818        minimizerOptions: {
819          // 生产模式打包
820          removeMetadata: true,
821          // 生产模式打包
822          removeSizeAttribute: true,
823        },
824      }),
825      new PreloadWebpackPlugin(),
826    ],
827    optimization: {
828      // 生产模式打包
829      minimize: true,
830      // 生产模式打包
831      minimizer: [
832        new CssMinimizerPlugin(),
833      ],
834    },
835  };
836
837  // 生产模式打包
838  module.exports = {
839    mode: "production",
840    entry: "./src/main.js",
841    output: {
842      path: path.resolve(__dirname, "../dist"),
843      filename: "static/js/[name].[contenthash:8].js",
844      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
845    },
846    plugins: [
847      new ESLintWebpackPlugin(),
848      new HtmlWebpackPlugin(),
849      new MiniCssExtractPlugin({
850        // 生产模式打包
851        extract: true,
852      }),
853      new TerserPlugin({
854        // 生产模式打包
855        terserOptions: {
856          compress: {
857            // 生产模式打包
858            drop_console: true,
859            drop_debugger: true,
860          },
861        },
862      }),
863      new ImageMinimizerPlugin({
864        // 生产模式打包
865        minimizerOptions: {
866          // 生产模式打包
867          removeMetadata: true,
868          // 生产模式打包
869          removeSizeAttribute: true,
870        },
871      }),
872      new PreloadWebpackPlugin(),
873    ],
874    optimization: {
875      // 生产模式打包
876      minimize: true,
877      // 生产模式打包
878      minimizer: [
879        new CssMinimizerPlugin(),
880      ],
881    },
882  };
883
884  // 生产模式打包
885  module.exports = {
886    mode: "production",
887    entry: "./src/main.js",
888    output: {
889      path: path.resolve(__dirname, "../dist"),
890      filename: "static/js/[name].[contenthash:8].js",
891      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
892    },
893    plugins: [
894      new ESLintWebpackPlugin(),
895      new HtmlWebpackPlugin(),
896      new MiniCssExtractPlugin({
897        // 生产模式打包
898        extract: true,
899      }),
900      new TerserPlugin({
901        // 生产模式打包
902        terserOptions: {
903          compress: {
904            // 生产模式打包
905            drop_console: true,
906            drop_debugger: true,
907          },
908        },
909      }),
910      new ImageMinimizerPlugin({
911        // 生产模式打包
912        minimizerOptions: {
913          // 生产模式打包
914          removeMetadata: true,
915          // 生产模式打包
916          removeSizeAttribute: true,
917        },
918      }),
919      new PreloadWebpackPlugin(),
920    ],
921    optimization: {
922      // 生产模式打包
923      minimize: true,
924      // 生产模式打包
925      minimizer: [
926        new CssMinimizerPlugin(),
927      ],
928    },
929  };
930
931  // 生产模式打包
932  module.exports = {
933    mode: "production",
934    entry: "./src/main.js",
935    output: {
936      path: path.resolve(__dirname, "../dist"),
937      filename: "static/js/[name].[contenthash:8].js",
938      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
939    },
940    plugins: [
941      new ESLintWebpackPlugin(),
942      new HtmlWebpackPlugin(),
943      new MiniCssExtractPlugin({
944        // 生产模式打包
945        extract: true,
946      }),
947      new TerserPlugin({
948        // 生产模式打包
949        terserOptions: {
950          compress: {
951            // 生产模式打包
952            drop_console: true,
953            drop_debugger: true,
954          },
955        },
956      }),
957      new ImageMinimizerPlugin({
958        // 生产模式打包
959        minimizerOptions: {
960          // 生产模式打包
961          removeMetadata: true,
962          // 生产模式打包
963          removeSizeAttribute: true,
964        },
965      }),
966      new PreloadWebpackPlugin(),
967    ],
968    optimization: {
969      // 生产模式打包
970      minimize: true,
971      // 生产模式打包
972      minimizer: [
973        new CssMinimizerPlugin(),
974      ],
975    },
976  };
977
978  // 生产模式打包
979  module.exports = {
980    mode: "production",
981    entry: "./src/main.js",
982    output: {
983      path: path.resolve(__dirname, "../dist"),
984      filename: "static/js/[name].[contenthash:8].js",
985      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
986    },
987    plugins: [
988      new ESLintWebpackPlugin(),
989      new HtmlWebpackPlugin(),
990      new MiniCssExtractPlugin({
991        // 生产模式打包
992        extract: true,
993      }),
994      new TerserPlugin({
995        // 生产模式打包
996        terserOptions: {
997          compress: {
998            // 生产模式打包
999            drop_console: true,
1000            drop_debugger: true,
1001          },
1002        },
1003      }),
1004      new ImageMinimizerPlugin({
1005        // 生产模式打包
1006        minimizerOptions: {
1007          // 生产模式打包
1008          removeMetadata: true,
1009          // 生产模式打包
1010          removeSizeAttribute: true,
1011        },
1012      }),
1013      new PreloadWebpackPlugin(),
1014    ],
1015    optimization: {
1016      // 生产模式打包
1017      minimize: true,
1018      // 生产模式打包
1019      minimizer: [
1020        new CssMinimizerPlugin(),
1021      ],
1022    },
1023  };
1024
1025  // 生产模式打包
1026  module.exports = {
1027    mode: "production",
1028    entry: "./src/main.js",
1029    output: {
1030      path: path.resolve(__dirname, "../dist"),
1031      filename: "static/js/[name].[contenthash:8].js",
1032      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
1033    },
1034    plugins: [
1035      new ESLintWebpackPlugin(),
1036      new HtmlWebpackPlugin(),
1037      new MiniCssExtractPlugin({
1038        // 生产模式打包
1039        extract: true,
1040      }),
1041      new TerserPlugin({
1042        // 生产模式打包
1043        terserOptions: {
1044          compress: {
1045            // 生产模式打包
1046            drop_console: true,
1047            drop_debugger: true,
1048          },
1049        },
1050      }),
1051      new ImageMinimizerPlugin({
1052        // 生产模式打包
1053        minimizerOptions: {
1054          // 生产模式打包
1055          removeMetadata: true,
1056          // 生产模式打包
1057          removeSizeAttribute: true,
1058        },
1059      }),
1060      new PreloadWebpackPlugin(),
1061    ],
1062    optimization: {
1063      // 生产模式打包
1064      minimize: true,
1065      // 生产模式打包
1066      minimizer: [
1067        new CssMinimizerPlugin(),
1068      ],
1069    },
1070  };
1071
1072  // 生产模式打包
1073  module.exports = {
1074    mode: "production",
1075    entry: "./src/main.js",
1076    output: {
1077      path: path.resolve(__dirname, "../dist"),
1078      filename: "static/js/[name].[contenthash:8].js",
1079      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
1080    },
1081    plugins: [
1082      new ESLintWebpackPlugin(),
1083      new HtmlWebpackPlugin(),
1084      new MiniCssExtractPlugin({
1085        // 生产模式打包
1086        extract: true,
1087      }),
1088      new TerserPlugin({
1089        // 生产模式打包
1090        terserOptions: {
1091          compress: {
1092            // 生产模式打包
1093            drop_console: true,
1094            drop_debugger: true,
1095          },
1096        },
1097      }),
1098      new ImageMinimizerPlugin({
1099        // 生产模式打包
1100        minimizerOptions: {
1101          // 生产模式打包
1102          removeMetadata: true,
1103          // 生产模式打包
1104          removeSizeAttribute: true,
1105        },
1106      }),
1107      new PreloadWebpackPlugin(),
1108    ],
1109    optimization: {
1110      // 生产模式打包
1111      minimize: true,
1112      // 生产模式打包
1113      minimizer: [
1114        new CssMinimizerPlugin(),
1115      ],
1116    },
1117  };
1118
1119  // 生产模式打包
1120  module.exports = {
1121    mode: "production",
1122    entry: "./src/main.js",
1123    output: {
1124      path: path.resolve(__dirname, "../dist"),
1125      filename: "static/js/[name].[contenthash:8].js",
1126      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
1127    },
1128    plugins: [
1129      new ESLintWebpackPlugin(),
1130      new HtmlWebpackPlugin(),
1131      new MiniCssExtractPlugin({
1132        // 生产模式打包
1133        extract: true,
1134      }),
1135      new TerserPlugin({
1136        // 生产模式打包
1137        terserOptions: {
1138          compress: {
1139            // 生产模式打包
1140            drop_console: true,
1141            drop_debugger: true,
1142          },
1143        },
1144      }),
1145      new ImageMinimizerPlugin({
1146        // 生产模式打包
1147        minimizerOptions: {
1148          // 生产模式打包
1149          removeMetadata: true,
1150          // 生产模式打包
1151          removeSizeAttribute: true,
1152        },
1153      }),
1154      new PreloadWebpackPlugin(),
1155    ],
1156    optimization: {
1157      // 生产模式打包
1158      minimize: true,
1159      // 生产模式打包
1160      minimizer: [
1161        new CssMinimizerPlugin(),
1162      ],
1163    },
1164  };
1165
1166  // 生产模式打包
1167  module.exports = {
1168    mode: "production",
1169    entry: "./src/main.js",
1170    output: {
1171      path: path.resolve(__dirname, "../dist"),
1172      filename: "static/js/[name].[contenthash:8].js",
1173      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
1174    },
1175    plugins: [
1176      new ESLintWebpackPlugin(),
1177      new HtmlWebpackPlugin(),
1178      new MiniCssExtractPlugin({
1179        // 生产模式打包
1180        extract: true,
1181      }),
1182      new TerserPlugin({
1183        // 生产模式打包
1184        terserOptions: {
1185          compress: {
1186            // 生产模式打包
1187            drop_console: true,
1188            drop_debugger: true,
1189          },
1190        },
1191      }),
1192      new ImageMinimizerPlugin({
1193        // 生产模式打包
1194        minimizerOptions: {
1195          // 生产模式打包
1196          removeMetadata: true,
1197          // 生产模式打包
1198          removeSizeAttribute: true,
1199        },
1200      }),
1201      new PreloadWebpackPlugin(),
1202    ],
1203    optimization: {
1204      // 生产模式打包
1205      minimize: true,
1206      // 生产模式打包
1207      minimizer: [
1208        new CssMinimizerPlugin(),
1209      ],
1210    },
1211  };
1212
1213  // 生产模式打包
1214  module.exports = {
1215    mode: "production",
1216    entry: "./src/main.js",
1217    output: {
1218      path: path.resolve(__dirname, "../dist"),
1219      filename: "static/js/[name].[contenthash:8].js",
1220      chunkFilename: "static/js/[name].[contenthash:8].chunk.js",
1221    },
1222    plugins: [
1223      new ESLintWebpackPlugin(),
1224      new HtmlWebpackPlugin(),
1225      new MiniCssExtractPlugin({
1226        // 生产模式打包
1227        extract: true,
1228      }),
1229      new TerserPlugin({
1230        // 生产模式打包
1231        terserOptions: {
1232          compress: {
1233            // 生产模式打包
1234            drop_console: true,
1235            drop_debugger: true,
1236          },
1237        },
1238      }),
1239      new ImageMinimizerPlugin({
1240        // 生产模式打包
1241        minimizerOptions: {
1242          // 生产模式打包
1243          removeMetadata: true,
1244          // 生产模式打包
1245          removeSizeAttribute: true,
1246        },
1247      }),
1248      new PreloadWebpackPlugin(),
1249    ],
1250    optimization: {
1251      // 生产模式打包
1252      minimize: true,
1253      // 生产模式打包
```

```
40     assetModuleFilename: "static/media/[name].[hash][ext]", // 图片、字体等资
源命名方式 (注意用hash)
41     clean: true,
42   },
43   module: {
44     rules: [
45       {
46         oneOf: [
47           {
48             // 用来匹配 .css 结尾的文件
49             test: /\.css$/,
50             // use 数组里面 Loader 执行顺序是从右到左
51             use: getStyleLoaders(),
52           },
53           {
54             test: /\.less$/,
55             use: getStyleLoaders("less-loader"),
56           },
57           {
58             test: /\.s[ac]ss$/,
59             use: getStyleLoaders("sass-loader"),
60           },
61           {
62             test: /\.styl$/,
63             use: getStyleLoaders("stylus-loader"),
64           },
65           {
66             test: /\.(png|jpe?g|gif|svg)$/,
67             type: "asset",
68             parser: {
69               dataUrlCondition: {
70                 maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
71               },
72             },
73             // generator: {
74             //   // 将图片文件输出到 static/imgs 目录中
75             //   // 将图片文件命名 [hash:8][ext][query]
76             //   // [hash:8]: hash值取8位
77             //   // [ext]: 使用之前的文件扩展名
78             //   // [query]: 添加之前的query参数
79             //   filename: "static/imgs/[hash:8][ext][query]",
80             // },
81           },
82           {
83             test: /\.((ttf|woff2?))$/,
84             type: "asset/resource",
85             // generator: {
86             //   filename: "static/media/[hash:8][ext][query]",
87             // },
88           },
89         ],
90       },
91     ],
92   },
93 },
```

```

90         test: /\.js$/,
91         // exclude: /node_modules/, // 排除node_modules代码不编译
92         include: path.resolve(__dirname, "../src"), // 也可以用包含
93         use: [
94             {
95                 loader: "thread-loader", // 开启多进程
96                 options: {
97                     workers: threads, // 数量
98                 },
99             },
100             {
101                 loader: "babel-loader",
102                 options: {
103                     cacheDirectory: true, // 开启babel编译缓存
104                     cacheCompression: false, // 缓存文件不要压缩
105                     plugins: ["@babel/plugin-transform-runtime"], // 减少代码体
106                 },
107             },
108         ],
109     },
110 ],
111 },
112 ],
113 },
114 plugins: [
115     new ESLintWebpackPlugin({
116         // 指定检查文件的根目录
117         context: path.resolve(__dirname, "../src"),
118         exclude: "node_modules", // 默认值
119         cache: true, // 开启缓存
120         // 缓存目录
121         cacheLocation: path.resolve(
122             __dirname,
123             "../node_modules/.cache/.eslintcache"
124         ),
125         threads, // 开启多进程
126     }),
127     new HtmlWebpackPlugin({
128         // 以 public/index.html 为模板创建文件
129         // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
130         template: path.resolve(__dirname, "../public/index.html"),
131     }),
132     // 提取css成单独文件
133     new MiniCssExtractPlugin({
134         // 定义输出文件名和目录
135         filename: "static/css/[name].[contenthash:8].css",
136         chunkFilename: "static/css/[name].[contenthash:8].chunk.css",
137     }),
138     // css压缩
139     // new CssMinimizerPlugin(),

```

```
140     new PreloadWebpackPlugin({
141       rel: "preload", // preload兼容性更好
142       as: "script",
143       // rel: 'prefetch' // prefetch兼容性更差
144     }),
145   ],
146   optimization: {
147     minimizer: [
148       // css压缩也可以写到optimization.minimizer里面, 效果一样的
149       new CssMinimizerPlugin(),
150       // 当生产模式会默认开启TerserPlugin, 但是我们需要进行其他配置, 就要重新写了
151       new TerserPlugin({
152         parallel: threads, // 开启多进程
153       }),
154       // 压缩图片
155       new ImageMinimizerPlugin({
156         minimizer: {
157           implementation: ImageMinimizerPlugin.imageminGenerate,
158           options: {
159             plugins: [
160               ["gifsicle", { interlaced: true }],
161               ["jpegtran", { progressive: true }],
162               ["optipng", { optimizationLevel: 5 }],
163               [
164                 "svgo",
165                 {
166                   plugins: [
167                     "preset-default",
168                     "prefixIds",
169                     {
170                       name: "sortAttrs",
171                       params: {
172                         xmlnsOrder: "alphabetical",
173                       },
174                     },
175                   ],
176                 },
177               ],
178             ],
179           },
180         },
181       }),
182     ],
183     // 代码分割配置
184     splitChunks: {
185       chunks: "all", // 对所有模块都进行分割
186       // 其他内容用默认配置即可
187     },
188   },
189   // devServer: {
190     // host: "localhost", // 启动服务器域名
```

```

191     //   port: "3000", // 启动服务器端口号
192     //   open: true, // 是否自动打开浏览器
193     // },
194     mode: "production",
195     devtool: "source-map",
196   };

```

- 问题：

当我们修改 math.js 文件再重新打包的时候，因为 contenthash 原因，math.js 文件 hash 值发生了变化（这是正常的）。

但是 main.js 文件的 hash 值也发生了变化，这会导致 main.js 的缓存失效。明明我们只修改 math.js，为什么 main.js 也会变身变化呢？

- 原因：

- 更新前：math.xxx.js, main.js 引用的 math.xxx.js
- 更新后：math.yyy.js, main.js 引用的 math.yyy.js, 文件名发生了变化，间接导致 main.js 也发生了变化

- 解决：

将 hash 值单独保管在一个 runtime 文件中。

我们最终输出三个文件：main、math、runtime。当 math 文件发送变化，变化的是 math 和 runtime 文件，main 不变。

runtime 文件只保存文件的 hash 值和它们与文件关系，整个文件体积就比较小，所以变化重新请求的代价也小。

```

1  const os = require("os");
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
7  const TerserPlugin = require("terser-webpack-plugin");
8  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
9  const PreloadWebpackPlugin = require("@vue/preload-webpack-plugin");
10
11  // cpu核数
12  const threads = os.cpus().length;
13
14  // 获取处理样式的Loaders
15  const getStyleLoaders = (preProcessor) => {
16    return [
17      MiniCssExtractPlugin.loader,
18      "css-loader",
19      {
20        loader: "postcss-loader",

```

```
21     options: {
22       postcssOptions: {
23         plugins: [
24           "postcss-preset-env", // 能解决大多数样式兼容性问题
25         ],
26       },
27     },
28   },
29   preProcessor,
30 ].filter(Boolean);
31 };
32
33 module.exports = {
34   entry: "./src/main.js",
35   output: {
36     path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
37     // [contenthash:8]使用contenthash, 取8位长度
38     filename: "static/js/[name].[contenthash:8].js", // 入口文件打包输出资源命名方式
39     chunkFilename: "static/js/[name].[contenthash:8].chunk.js", // 动态导入输出资源命名方式
40     assetModuleFilename: "static/media/[name].[hash][ext]", // 图片、字体等资源命名方式 (注意用hash)
41     clean: true,
42   },
43   module: {
44     rules: [
45       {
46         oneOf: [
47           {
48             // 用来匹配 .css 结尾的文件
49             test: /\.css$/,
50             // use 数组里面 Loader 执行顺序是从右到左
51             use: getStyleLoaders(),
52           },
53           {
54             test: /\.less$/,
55             use: getStyleLoaders("less-loader"),
56           },
57           {
58             test: /\.s[ac]ss$/,
59             use: getStyleLoaders("sass-loader"),
60           },
61           {
62             test: /\.styl$/,
63             use: getStyleLoaders("stylus-loader"),
64           },
65           {
66             test: /\.(png|jpe?g|gif|svg)$/,
67             type: "asset",
68             parser: {
```

```

69         dataUrlCondition: {
70             maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
71         },
72     },
73     // generator: {
74     //     // 将图片文件输出到 static/imgs 目录中
75     //     // 将图片文件命名 [hash:8][ext][query]
76     //     // [hash:8]: hash值取8位
77     //     // [ext]: 使用之前的文件扩展名
78     //     // [query]: 添加之前的query参数
79     //     filename: "static/imgs/[hash:8][ext][query]",
80     // },
81 },
82 {
83     test: /\.?(ttf|woff2?)$/,
84     type: "asset/resource",
85     // generator: {
86     //     filename: "static/media/[hash:8][ext][query]",
87     // },
88 },
89 {
90     test: /\.js$/,
91     // exclude: /node_modules/, // 排除node_modules代码不编译
92     include: path.resolve(__dirname, "../src"), // 也可以用包含
93     use: [
94         {
95             loader: "thread-loader", // 开启多进程
96             options: {
97                 workers: threads, // 数量
98             },
99         },
100         {
101             loader: "babel-loader",
102             options: {
103                 cacheDirectory: true, // 开启babel编译缓存
104                 cacheCompression: false, // 缓存文件不要压缩
105                 plugins: ["@babel/plugin-transform-runtime"], // 减少代码体
106             },
107         },
108     ],
109 },
110 ],
111 },
112 ],
113 },
114 plugins: [
115     new ESLintWebpackPlugin({
116         // 指定检查文件的根目录
117         context: path.resolve(__dirname, "../src"),
118         exclude: "node_modules", // 默认值

```





```

170             name: "sortAttrs",
171             params: {
172                 xmlnsOrder: "alphabetical",
173             },
174         },
175     ],
176 },
177 ],
178 ],
179 },
180 },
181 }),
182 ],
183 // 代码分割配置
184 splitChunks: {
185     chunks: "all", // 对所有模块都进行分割
186     // 其他内容用默认配置即可
187 },
188 // 提取runtime文件
189 runtimeChunk: {
190     name: (entrypoint) => `runtime-${entrypoint.name}`, // runtime文件命名
191     规则
192 },
193 // devServer: {
194 //     host: "localhost", // 启动服务器域名
195 //     port: "3000", // 启动服务器端口号
196 //     open: true, // 是否自动打开浏览器
197 // },
198 mode: "production",
199 devtool: "source-map",
200 };

```

## Core-js

### 为什么

过去我们使用 babel 对 js 代码进行了兼容性处理，其中使用@babel/preset-env 智能预设来处理兼容性问题。

它可将 ES6 的一些语法进行编译转换，比如箭头函数、点点点运算符等。但是如果是 async 函数、promise 对象、数组的一些方法（includes）等，它没办法处理。

所以此时我们 js 代码仍然存在兼容性问题，一旦遇到低版本浏览器会直接报错。所以我们想要将 js 兼容性问题彻底解决

### 是什么

core-js 是专门用来做 ES6 以及以上 API 的 polyfill。

polyfill 翻译过来叫做垫片/补丁。就是用社区上提供的一段代码，让我们在不兼容某些新特性的浏览器上，使用该新特性。

## 怎么用

### 1. 修改 main.js

```
1  import count from "./js/count";
2  import sum from "./js/sum";
3  // 引入资源, Webpack才会对其打包
4  import "./css/iconfont.css";
5  import "./css/index.css";
6  import "./less/index.less";
7  import "./sass/index.sass";
8  import "./sass/index.scss";
9  import "./styl/index.styl";
10
11  const result1 = count(2, 1);
12  console.log(result1);
13  const result2 = sum(1, 2, 3, 4);
14  console.log(result2);
15  // 添加promise代码
16  const promise = Promise.resolve();
17  promise.then(() => {
18    console.log("hello promise");
19  });
```

此时 ESLint 会对 Promise 报错。

### 2. 修改配置文件

- 下载包

```
1  npm i @babel/eslint-parser -D
```

- .eslintrc.js

```
1  module.exports = {
2    // 继承 ESLint 规则
3    extends: ["eslint:recommended"],
4    parser: "@babel/eslint-parser", // 支持最新的最终 ECMAScript 标准
5    env: {
6      node: true, // 启用node中全局变量
7      browser: true, // 启用浏览器中全局变量
8    },
9    plugins: ["import"], // 解决动态导入import语法报错问题 → 实际使用eslint-plugin-import的规则解决的
10   parserOptions: {
11     ecmaVersion: 6, // es6
12     sourceType: "module", // es module
13   },
```

```
14     rules: {
15       "no-var": 2, // 不能使用 var 定义变量
16     },
17   };
```

### 3. 运行指令

```
1  npm run build
```

此时观察打包输出的 js 文件，我们发现 Promise 语法并没有编译转换，所以我们需要使用 `core-js` 来进行 `polyfill`。

### 4. 使用 `core-js`

- 下载包

```
1  npm i core-js
```

- 手动全部引入

```
1  import "core-js";
2  import count from "./js/count";
3  import sum from "./js/sum";
4  // 引入资源, Webpack才会对其打包
5  import "./css/iconfont.css";
6  import "./css/index.css";
7  import "./less/index.less";
8  import "./sass/index.sass";
9  import "./sass/index.scss";
10 import "./styl/index.styl";
11
12 const result1 = count(2, 1);
13 console.log(result1);
14 const result2 = sum(1, 2, 3, 4);
15 console.log(result2);
16 // 添加promise代码
17 const promise = Promise.resolve();
18 promise.then(() => {
19   console.log("hello promise");
20 });
```

这样引入会将所有兼容性代码全部引入，体积太大了。我们只想引入 promise 的 `polyfill`。

- 手动按需引入

```

1  import "core-js/es/promise";
2  import count from "./js/count";
3  import sum from "./js/sum";
4  // 引入资源, Webpack才会对其打包
5  import "./css/iconfont.css";
6  import "./css/index.css";
7  import "./less/index.less";
8  import "./sass/index.sass";
9  import "./sass/index.scss";
10 import "./styl/index.styl";
11
12 const result1 = count(2, 1);
13 console.log(result1);
14 const result2 = sum(1, 2, 3, 4);
15 console.log(result2);
16 // 添加promise代码
17 const promise = Promise.resolve();
18 promise.then(() => {
19     console.log("hello promise");
20 });

```

只引入打包 promise 的 `polyfill`，打包体积更小。但是将来如果还想使用其他语法，我需要手动引入库很麻烦。

- 自动按需引入
  - main.js

```

1  import count from "./js/count";
2  import sum from "./js/sum";
3  // 引入资源, Webpack才会对其打包
4  import "./css/iconfont.css";
5  import "./css/index.css";
6  import "./less/index.less";
7  import "./sass/index.sass";
8  import "./sass/index.scss";
9  import "./styl/index.styl";
10
11 const result1 = count(2, 1);
12 console.log(result1);
13 const result2 = sum(1, 2, 3, 4);
14 console.log(result2);
15 // 添加promise代码
16 const promise = Promise.resolve();
17 promise.then(() => {
18     console.log("hello promise");
19 });

```

- babel.config.js

```

1  module.exports = {
2    // 智能预设: 能够编译ES6语法
3    presets: [
4      [
5        "@babel/preset-env",
6        // 按需加载core-js的polyfill
7        { useBuiltIns: "usage", corejs: { version: "3", proposals: true } }
8      ],
9    ],
10 };

```

此时就会自动根据我们代码中使用的语法，来按需加载相应的 `polyfill` 了。

## - PWA

### 为什么

开发 Web App 项目，项目一旦处于网络离线情况，就没法访问了。

我们希望给项目提供离线体验。

### 是什么

渐进式网络应用程序(progressive web application - PWA): 是一种可以提供类似于 native app(原生应用程序) 体验的 Web App 的技术。

其中最重要的是，在 **离线(offline)** 时应用程序能够继续运行功能。

内部通过 Service Workers 技术实现的。

### 怎么用

#### 1. 下载包

```
1  npm i workbox-webpack-plugin -D
```

#### 2. 修改配置文件

```

1  const os = require("os");
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
7  const TerserPlugin = require("terser-webpack-plugin");
8  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
9  const PreloadWebpackPlugin = require("@vue/preload-webpack-plugin");
10 const WorkboxPlugin = require("workbox-webpack-plugin");
11

```

```
12 // cpu核数
13 const threads = os.cpus().length;
14
15 // 获取处理样式的Loaders
16 const getStyleLoaders = (preProcessor) => {
17   return [
18     MiniCssExtractPlugin.loader,
19     "css-loader",
20     {
21       loader: "postcss-loader",
22       options: {
23         postcssOptions: {
24           plugins: [
25             "postcss-preset-env", // 能解决大多数样式兼容性问题
26           ],
27         },
28       },
29     ],
30     preProcessor,
31   ].filter(Boolean);
32 };
33
34 module.exports = {
35   entry: "./src/main.js",
36   output: {
37     path: path.resolve(__dirname, "../dist"), // 生产模式需要输出
38     // [contenthash:8]使用contenthash, 取8位长度
39     filename: "static/js/[name].[contenthash:8].js", // 入口文件打包输出资源命名方式
40     chunkFilename: "static/js/[name].[contenthash:8].chunk.js", // 动态导入输出资源命名方式
41     assetModuleFilename: "static/media/[name].[hash][ext]", // 图片、字体等资源命名方式 (注意用hash)
42     clean: true,
43   },
44   module: {
45     rules: [
46       {
47         oneOf: [
48           {
49             // 用来匹配 .css 结尾的文件
50             test: /\.css$/,
51             // use 数组里面 Loader 执行顺序是从右到左
52             use: getStyleLoaders(),
53           },
54           {
55             test: /\.less$/,
56             use: getStyleLoaders("less-loader"),
57           },
58           {
59             test: /\.s[ac]ss$/,
```

```

60     use: getStyleLoaders("sass-loader"),
61   },
62   {
63     test: /\.styl$/,
64     use: getStyleLoaders("stylus-loader"),
65   },
66   {
67     test: /\.?(png|jpe?g|gif|svg)$/,
68     type: "asset",
69     parser: {
70       dataUrlCondition: {
71         maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
72       },
73     },
74     // generator: {
75     //   // 将图片文件输出到 static/imgs 目录中
76     //   // 将图片文件命名 [hash:8][ext][query]
77     //   // [hash:8]: hash值取8位
78     //   // [ext]: 使用之前的文件扩展名
79     //   // [query]: 添加之前的query参数
80     //   filename: "static/imgs/[hash:8][ext][query]",
81     // },
82   },
83   {
84     test: /\.?(ttf|woff2?)$/,
85     type: "asset/resource",
86     // generator: {
87     //   filename: "static/media/[hash:8][ext][query]",
88     // },
89   },
90   {
91     test: /\.js$/,
92     // exclude: /node_modules/, // 排除node_modules代码不编译
93     include: path.resolve(__dirname, "../src"), // 也可以用包含
94     use: [
95       {
96         loader: "thread-loader", // 开启多进程
97         options: {
98           workers: threads, // 数量
99         },
100       },
101       {
102         loader: "babel-loader",
103         options: {
104           cacheDirectory: true, // 开启babel编译缓存
105           cacheCompression: false, // 缓存文件不要压缩
106           plugins: ["@babel/plugin-transform-runtime"], // 减少代码体
107         },
108       },
109     ],

```

积



```
110     },
111   ],
112 },
113 ],
114 },
115 plugins: [
116   new ESLintWebpackPlugin({
117     // 指定检查文件的根目录
118     context: path.resolve(__dirname, "../src"),
119     exclude: "node_modules", // 默认值
120     cache: true, // 开启缓存
121     // 缓存目录
122     cacheLocation: path.resolve(
123       __dirname,
124       "../node_modules/.cache/.eslintcache"
125     ),
126     threads, // 开启多进程
127   }),
128   new HtmlWebpackPlugin({
129     // 以 public/index.html 为模板创建文件
130     // 新的html文件有两个特点: 1. 内容和源文件一致 2. 自动引入打包生成的js等资源
131     template: path.resolve(__dirname, "../public/index.html"),
132   }),
133   // 提取css成单独文件
134   new MiniCssExtractPlugin({
135     // 定义输出文件名和目录
136     filename: "static/css/[name].[contenthash:8].css",
137     chunkFilename: "static/css/[name].[contenthash:8].chunk.css",
138   }),
139   // css压缩
140   // new CssMinimizerPlugin(),
141   new PreloadWebpackPlugin({
142     rel: "preload", // preload兼容性更好
143     as: "script",
144     // rel: 'prefetch' // prefetch兼容性更差
145   }),
146   new WorkboxPlugin.GenerateSW({
147     // 这些选项帮助快速启用 ServiceWorkers
148     // 不允许遗留任何“旧的” ServiceWorkers
149     clientsClaim: true,
150     skipWaiting: true,
151   }),
152 ],
153 optimization: {
154   minimizer: [
155     // css压缩也可以写到optimization.minimizer里面, 效果一样的
156     new CssMinimizerPlugin(),
157     // 当生产模式会默认开启TerserPlugin, 但是我们需要进行其他配置, 就要重新写了
158     new TerserPlugin({
159       parallel: threads, // 开启多进程
160     }),
161   ],
162 },
163 ],
164 },
165 ],
166 },
167 ],
168 },
169 ],
170 ],
171 ],
172 ],
173 ],
174 ],
175 ],
176 ],
177 ],
178 ],
179 ],
180 ],
181 ],
182 ],
183 ],
184 ],
185 ],
186 ],
187 ],
188 ],
189 ],
190 ],
191 ],
192 ],
193 ],
194 ],
195 ],
196 ],
197 ],
198 ],
199 ],
200 ],
201 ],
202 ],
203 ],
204 ],
205 ],
206 ],
207 ],
208 ],
209 ],
210 ],
211 ],
212 ],
213 ],
214 ],
215 ],
216 ],
217 ],
218 ],
219 ],
220 ],
221 ],
222 ],
223 ],
224 ],
225 ],
226 ],
227 ],
228 ],
229 ],
230 ],
231 ],
232 ],
233 ],
234 ],
235 ],
236 ],
237 ],
238 ],
239 ],
240 ],
241 ],
242 ],
243 ],
244 ],
245 ],
246 ],
247 ],
248 ],
249 ],
250 ],
251 ],
252 ],
253 ],
254 ],
255 ],
256 ],
257 ],
258 ],
259 ],
260 ],
261 ],
262 ],
263 ],
264 ],
265 ],
266 ],
267 ],
268 ],
269 ],
270 ],
271 ],
272 ],
273 ],
274 ],
275 ],
276 ],
277 ],
278 ],
279 ],
280 ],
281 ],
282 ],
283 ],
284 ],
285 ],
286 ],
287 ],
288 ],
289 ],
290 ],
291 ],
292 ],
293 ],
294 ],
295 ],
296 ],
297 ],
298 ],
299 ],
300 ],
301 ],
302 ],
303 ],
304 ],
305 ],
306 ],
307 ],
308 ],
309 ],
310 ],
311 ],
312 ],
313 ],
314 ],
315 ],
316 ],
317 ],
318 ],
319 ],
320 ],
321 ],
322 ],
323 ],
324 ],
325 ],
326 ],
327 ],
328 ],
329 ],
330 ],
331 ],
332 ],
333 ],
334 ],
335 ],
336 ],
337 ],
338 ],
339 ],
340 ],
341 ],
342 ],
343 ],
344 ],
345 ],
346 ],
347 ],
348 ],
349 ],
350 ],
351 ],
352 ],
353 ],
354 ],
355 ],
356 ],
357 ],
358 ],
359 ],
360 ],
361 ],
362 ],
363 ],
364 ],
365 ],
366 ],
367 ],
368 ],
369 ],
370 ],
371 ],
372 ],
373 ],
374 ],
375 ],
376 ],
377 ],
378 ],
379 ],
380 ],
381 ],
382 ],
383 ],
384 ],
385 ],
386 ],
387 ],
388 ],
389 ],
390 ],
391 ],
392 ],
393 ],
394 ],
395 ],
396 ],
397 ],
398 ],
399 ],
400 ],
401 ],
402 ],
403 ],
404 ],
405 ],
406 ],
407 ],
408 ],
409 ],
410 ],
411 ],
412 ],
413 ],
414 ],
415 ],
416 ],
417 ],
418 ],
419 ],
420 ],
421 ],
422 ],
423 ],
424 ],
425 ],
426 ],
427 ],
428 ],
429 ],
430 ],
431 ],
432 ],
433 ],
434 ],
435 ],
436 ],
437 ],
438 ],
439 ],
440 ],
441 ],
442 ],
443 ],
444 ],
445 ],
446 ],
447 ],
448 ],
449 ],
450 ],
451 ],
452 ],
453 ],
454 ],
455 ],
456 ],
457 ],
458 ],
459 ],
460 ],
461 ],
462 ],
463 ],
464 ],
465 ],
466 ],
467 ],
468 ],
469 ],
470 ],
471 ],
472 ],
473 ],
474 ],
475 ],
476 ],
477 ],
478 ],
479 ],
480 ],
481 ],
482 ],
483 ],
484 ],
485 ],
486 ],
487 ],
488 ],
489 ],
490 ],
491 ],
492 ],
493 ],
494 ],
495 ],
496 ],
497 ],
498 ],
499 ],
500 ],
501 ],
502 ],
503 ],
504 ],
505 ],
506 ],
507 ],
508 ],
509 ],
510 ],
511 ],
512 ],
513 ],
514 ],
515 ],
516 ],
517 ],
518 ],
519 ],
520 ],
521 ],
522 ],
523 ],
524 ],
525 ],
526 ],
527 ],
528 ],
529 ],
530 ],
531 ],
532 ],
533 ],
534 ],
535 ],
536 ],
537 ],
538 ],
539 ],
540 ],
541 ],
542 ],
543 ],
544 ],
545 ],
546 ],
547 ],
548 ],
549 ],
550 ],
551 ],
552 ],
553 ],
554 ],
555 ],
556 ],
557 ],
558 ],
559 ],
560 ],
561 ],
562 ],
563 ],
564 ],
565 ],
566 ],
567 ],
568 ],
569 ],
570 ],
571 ],
572 ],
573 ],
574 ],
575 ],
576 ],
577 ],
578 ],
579 ],
580 ],
581 ],
582 ],
583 ],
584 ],
585 ],
586 ],
587 ],
588 ],
589 ],
590 ],
591 ],
592 ],
593 ],
594 ],
595 ],
596 ],
597 ],
598 ],
599 ],
600 ],
601 ],
602 ],
603 ],
604 ],
605 ],
606 ],
607 ],
608 ],
609 ],
610 ],
611 ],
612 ],
613 ],
614 ],
615 ],
616 ],
617 ],
618 ],
619 ],
620 ],
621 ],
622 ],
623 ],
624 ],
625 ],
626 ],
627 ],
628 ],
629 ],
630 ],
631 ],
632 ],
633 ],
634 ],
635 ],
636 ],
637 ],
638 ],
639 ],
640 ],
641 ],
642 ],
643 ],
644 ],
645 ],
646 ],
647 ],
648 ],
649 ],
650 ],
651 ],
652 ],
653 ],
654 ],
655 ],
656 ],
657 ],
658 ],
659 ],
660 ],
661 ],
662 ],
663 ],
664 ],
665 ],
666 ],
667 ],
668 ],
669 ],
670 ],
671 ],
672 ],
673 ],
674 ],
675 ],
676 ],
677 ],
678 ],
679 ],
680 ],
681 ],
682 ],
683 ],
684 ],
685 ],
686 ],
687 ],
688 ],
689 ],
690 ],
691 ],
692 ],
693 ],
694 ],
695 ],
696 ],
697 ],
698 ],
699 ],
700 ],
701 ],
702 ],
703 ],
704 ],
705 ],
706 ],
707 ],
708 ],
709 ],
710 ],
711 ],
712 ],
713 ],
714 ],
715 ],
716 ],
717 ],
718 ],
719 ],
720 ],
721 ],
722 ],
723 ],
724 ],
725 ],
726 ],
727 ],
728 ],
729 ],
730 ],
731 ],
732 ],
733 ],
734 ],
735 ],
736 ],
737 ],
738 ],
739 ],
740 ],
741 ],
742 ],
743 ],
744 ],
745 ],
746 ],
747 ],
748 ],
749 ],
750 ],
751 ],
752 ],
753 ],
754 ],
755 ],
756 ],
757 ],
758 ],
759 ],
760 ],
761 ],
762 ],
763 ],
764 ],
765 ],
766 ],
767 ],
768 ],
769 ],
770 ],
771 ],
772 ],
773 ],
774 ],
775 ],
776 ],
777 ],
778 ],
779 ],
780 ],
781 ],
782 ],
783 ],
784 ],
785 ],
786 ],
787 ],
788 ],
789 ],
790 ],
791 ],
792 ],
793 ],
794 ],
795 ],
796 ],
797 ],
798 ],
799 ],
800 ],
801 ],
802 ],
803 ],
804 ],
805 ],
806 ],
807 ],
808 ],
809 ],
810 ],
811 ],
812 ],
813 ],
814 ],
815 ],
816 ],
817 ],
818 ],
819 ],
820 ],
821 ],
822 ],
823 ],
824 ],
825 ],
826 ],
827 ],
828 ],
829 ],
830 ],
831 ],
832 ],
833 ],
834 ],
835 ],
836 ],
837 ],
838 ],
839 ],
840 ],
841 ],
842 ],
843 ],
844 ],
845 ],
846 ],
847 ],
848 ],
849 ],
850 ],
851 ],
852 ],
853 ],
854 ],
855 ],
856 ],
857 ],
858 ],
859 ],
860 ],
861 ],
862 ],
863 ],
864 ],
865 ],
866 ],
867 ],
868 ],
869 ],
870 ],
871 ],
872 ],
873 ],
874 ],
875 ],
876 ],
877 ],
878 ],
879 ],
880 ],
881 ],
882 ],
883 ],
884 ],
885 ],
886 ],
887 ],
888 ],
889 ],
890 ],
891 ],
892 ],
893 ],
894 ],
895 ],
896 ],
897 ],
898 ],
899 ],
900 ],
901 ],
902 ],
903 ],
904 ],
905 ],
906 ],
907 ],
908 ],
909 ],
910 ],
911 ],
912 ],
913 ],
914 ],
915 ],
916 ],
917 ],
918 ],
919 ],
920 ],
921 ],
922 ],
923 ],
924 ],
925 ],
926 ],
927 ],
928 ],
929 ],
930 ],
931 ],
932 ],
933 ],
934 ],
935 ],
936 ],
937 ],
938 ],
939 ],
940 ],
941 ],
942 ],
943 ],
944 ],
945 ],
946 ],
947 ],
948 ],
949 ],
950 ],
951 ],
952 ],
953 ],
954 ],
955 ],
956 ],
957 ],
958 ],
959 ],
960 ],
961 ],
962 ],
963 ],
964 ],
965 ],
966 ],
967 ],
968 ],
969 ],
970 ],
971 ],
972 ],
973 ],
974 ],
975 ],
976 ],
977 ],
978 ],
979 ],
980 ],
981 ],
982 ],
983 ],
984 ],
985 ],
986 ],
987 ],
988 ],
989 ],
990 ],
991 ],
992 ],
993 ],
994 ],
995 ],
996 ],
997 ],
998 ],
999 ],
1000 ],
```

```

161     // 压缩图片
162     new ImageMinimizerPlugin({
163         minimizer: {
164             implementation: ImageMinimizerPlugin.imageminGenerate,
165             options: {
166                 plugins: [
167                     ["gifsicle", { interlaced: true }],
168                     ["jpegtran", { progressive: true }],
169                     ["optipng", { optimizationLevel: 5 }],
170                     [
171                         "svgo",
172                         {
173                             plugins: [
174                                 "preset-default",
175                                 "prefixIds",
176                                 {
177                                     name: "sortAttrs",
178                                     params: {
179                                         xmlnsOrder: "alphabetical",
180                                     },
181                                 },
182                             ],
183                         },
184                     ],
185                 ],
186             },
187         },
188     }),
189 ],
190 // 代码分割配置
191 splitChunks: {
192     chunks: "all", // 对所有模块都进行分割
193     // 其他内容用默认配置即可
194 },
195 },
196 // devServer: {
197 //     host: "localhost", // 启动服务器域名
198 //     port: "3000", // 启动服务器端口号
199 //     open: true, // 是否自动打开浏览器
200 // },
201 mode: "production",
202 devtool: "source-map",
203 };

```

### 3. 修改 main.js

```

1  import count from "./js/count";
2  import sum from "./js/sum";
3  // 引入资源, Webpack才会对其打包
4  import "./css/iconfont.css";

```

```

5   import "./css/index.css";
6   import "./less/index.less";
7   import "./sass/index.sass";
8   import "./sass/index.scss";
9   import "./styl/index.styl";
10
11  const result1 = count(2, 1);
12  console.log(result1);
13  const result2 = sum(1, 2, 3, 4);
14  console.log(result2);
15  // 添加promise代码
16  const promise = Promise.resolve();
17  promise.then(() => {
18    console.log("hello promise");
19  });
20
21  const arr = [1, 2, 3, 4, 5];
22  console.log(arr.includes(5));
23
24  if ("serviceWorker" in navigator) {
25    window.addEventListener("load", () => {
26      navigator.serviceWorker
27        .register("/service-worker.js")
28        .then((registration) => {
29          console.log("SW registered: ", registration);
30        })
31        .catch((registrationError) => {
32          console.log("SW registration failed: ", registrationError);
33        });
34    });
35  }

```

#### 4. 运行指令

```
1  npm run build
```

此时如果直接通过 VSCode 访问打包后页面，在浏览器控制台会发现 `SW registration failed`。

因为我们打开的访问路径是：`http://127.0.0.1:5500/dist/index.html`。此时页面会去请求 `service-worker.js` 文件，请求路径是：`http://127.0.0.1:5500/service-worker.js`，这样找不到会 404。

实际 `service-worker.js` 文件路径是：`http://127.0.0.1:5500/dist/service-worker.js`。

#### 5. 解决路径问题

- 下载包

```
1 npm i serve -g
```

serve 也是用来启动开发服务器来部署代码查看效果的。

- 运行指令

```
1 serve dist
```

此时通过 serve 启动的服务器我们 service-worker 就能注册成功了。

## • 总结

我们从 4 个角度对 webpack 和代码进行了优化：

### 1. 提升开发体验

- 使用 `Source Map` 让开发或上线时代码报错能有更加准确的错误提示。

### 2. 提升 webpack 提升打包构建速度

- 使用 `HotModuleReplacement` 让开发时只重新编译打包更新变化了的代码，不变的代码使用缓存，从而使更新速度更快。
- 使用 `OneOf` 让资源文件一旦被某个 loader 处理了，就不会继续遍历了，打包速度更快。
- 使用 `Include/Exclude` 排除或只检测某些文件，处理的文件更少，速度更快。
- 使用 `Cache` 对 eslint 和 babel 处理的结果进行缓存，让第二次打包速度更快。
- 使用 `Thread` 多进程处理 eslint 和 babel 任务，速度更快。（需要注意的是，进程启动通信都有开销的，要在比较多代码处理时使用才有效果）

### 3. 减少代码体积

- 使用 `Tree Shaking` 剔除了没有使用的多余代码，让代码体积更小。
- 使用 `@babel/plugin-transform-runtime` 插件对 babel 进行处理，让辅助代码从中引入，而不是每个文件都生成辅助代码，从而体积更小。
- 使用 `Image Minimizer` 对项目中的图片进行压缩，体积更小，请求速度更快。（需要注意的是，如果项目中图片都是在线链接，那么就不需要了。本地项目静态图片才需要进行压缩。）

### 4. 优化代码运行性能

- 使用 `Code Split` 对代码进行分割成多个 js 文件，从而使单个文件体积更小，并行加载 js 速度更快。并通过 import 动态导入语法进行按需加载，从而达到需要使用时才加载该资源，不用时不加载资源。
- 使用 `Preload / Prefetch` 对代码进行提前加载，等未来需要使用时就能直接使用，从而用户体验更好。
- 使用 `Network Cache` 能对输出资源文件进行更好的命名，将来好做缓存，从而用户体验更好。

- 使用 `Core-js` 对 js 进行兼容性处理，让我们代码能运行在低版本浏览器。
- 使用 `PWA` 能让代码离线也能访问，从而提升用户体验。

## 项目配置

### • React 脚手架

#### – 开发模式配置

```
1  // webpack.dev.js
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const ReactRefreshWebpackPlugin = require("@pmmmwh/react-refresh-webpack-
  plugin");
6  const CopyPlugin = require("copy-webpack-plugin");
7
8  const getStyleLoaders = (preProcessor) => {
9    return [
10      "style-loader",
11      "css-loader",
12      {
13        loader: "postcss-loader",
14        options: {
15          postcssOptions: {
16            plugins: [
17              "postcss-preset-env", // 能解决大多数样式兼容性问题
18            ],
19          },
20        },
21      },
22      preProcessor,
23    ].filter(Boolean);
24  };
25
26  module.exports = {
27    entry: "./src/main.js",
28    output: {
29      path: undefined,
30      filename: "static/js/[name].js",
31      chunkFilename: "static/js/[name].chunk.js",
32      assetModuleFilename: "static/js/[hash:10][ext][query]",
33    },
34    module: {
35      rules: [
36        {
37          oneOf: [
38            {
```

```
39         // 用来匹配 .css 结尾的文件
40         test: /\.css$/,
41         // use 数组里面 Loader 执行顺序是从右到左
42         use: getStyleLoaders(),
43     },
44     {
45         test: /\.less$/,
46         use: getStyleLoaders("less-loader"),
47     },
48     {
49         test: /\.s[ac]ss$/,
50         use: getStyleLoaders("sass-loader"),
51     },
52     {
53         test: /\.styl$/,
54         use: getStyleLoaders("stylus-loader"),
55     },
56     {
57         test: /\.?(png|jpe?g|gif|svg)$/,
58         type: "asset",
59         parser: {
60             dataUrlCondition: {
61                 maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
62             },
63         },
64     },
65     {
66         test: /\.?(ttf|woff2?)$/,
67         type: "asset/resource",
68     },
69     {
70         test: /\.?(jsx|js)$/,
71         include: path.resolve(__dirname, "../src"),
72         loader: "babel-loader",
73         options: {
74             cacheDirectory: true,
75             cacheCompression: false,
76             plugins: [
77                 // "@babel/plugin-transform-runtime", // presets中包含了
78                 "react-refresh/babel", // 开启js的HMR功能
79             ],
80         },
81     },
82 ],
83 },
84 ],
85 },
86 plugins: [
87     new ESLintWebpackPlugin({
88         context: path.resolve(__dirname, "../src"),
89         exclude: "node_modules",
```

```
90     cache: true,
91     cacheLocation: path.resolve(
92       __dirname,
93       "../node_modules/.cache/.eslintcache"
94     ),
95   }),
96   new HtmlWebpackPlugin({
97     template: path.resolve(__dirname, "../public/index.html"),
98   }),
99   new ReactRefreshWebpackPlugin(), // 解决js的HMR功能运行时全局变量的问题
100   // 将public下面的资源复制到dist目录去 (除了index.html)
101   new CopyPlugin({
102     patterns: [
103       {
104         from: path.resolve(__dirname, "../public"),
105         to: path.resolve(__dirname, "../dist"),
106         toType: "dir",
107         noErrorOnMissing: true, // 不生成错误
108         globOptions: {
109           // 忽略文件
110           ignore: ["**/index.html"],
111         },
112         info: {
113           // 跳过terser压缩js
114           minimized: true,
115         },
116       },
117     ],
118   }),
119 ],
120 optimization: {
121   splitChunks: {
122     chunks: "all",
123   },
124   runtimeChunk: {
125     name: (entrypoint) => `runtime-${entrypoint.name}`,
126   },
127 },
128 resolve: {
129   extensions: [".jsx", ".js", ".json"], // 自动补全文件扩展名, 让jsx可以使用
130 },
131 devServer: {
132   open: true,
133   host: "localhost",
134   port: 3000,
135   hot: true,
136   compress: true,
137   historyApiFallback: true, // 解决react-router刷新404问题
138 },
139 mode: "development",
140 devtool: "cheap-module-source-map",
```

```
141   };
```

## - 生产模式配置

```
1  // webpack.prod.js
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const TerserWebpackPlugin = require("terser-webpack-plugin");
7  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
8  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
9  const CopyPlugin = require("copy-webpack-plugin");
10
11  const getStyleLoaders = (preProcessor) => {
12    return [
13      MiniCssExtractPlugin.loader,
14      "css-loader",
15      {
16        loader: "postcss-loader",
17        options: {
18          postcssOptions: {
19            plugins: [
20              "postcss-preset-env", // 能解决大多数样式兼容性问题
21            ],
22          },
23        },
24      },
25      preProcessor,
26    ].filter(Boolean);
27  };
28
29  module.exports = {
30    entry: "./src/main.js",
31    output: {
32      path: path.resolve(__dirname, "../dist"),
33      filename: "static/js/[name].[contenthash:10].js",
34      chunkFilename: "static/js/[name].[contenthash:10].chunk.js",
35      assetModuleFilename: "static/js/[hash:10][ext][query]",
36      clean: true,
37    },
38    module: {
39      rules: [
40        {
41          oneOf: [
42            {
43              // 用来匹配 .css 结尾的文件
44              test: /\.css$/,
45              // use 数组里面 Loader 执行顺序是从右到左
46              use: getStyleLoaders(),
```



```
47     },
48     {
49       test: /\.less$/,
50       use: getStyleLoaders("less-loader"),
51     },
52     {
53       test: /\.s[ac]ss$/,
54       use: getStyleLoaders("sass-loader"),
55     },
56     {
57       test: /\.styl$/,
58       use: getStyleLoaders("stylus-loader"),
59     },
60     {
61       test: /\.?(png|jpe?g|gif|svg)$/,
62       type: "asset",
63       parser: {
64         dataUrlCondition: {
65           maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
66         },
67       },
68     },
69     {
70       test: /\.?(ttf|woff2?)$/,
71       type: "asset/resource",
72     },
73     {
74       test: /\.?(jsx|js)$/,
75       include: path.resolve(__dirname, "../src"),
76       loader: "babel-loader",
77       options: {
78         cacheDirectory: true,
79         cacheCompression: false,
80         plugins: [
81           // "@babel/plugin-transform-runtime" // presets中包含了
82         ],
83       },
84     },
85   ],
86 },
87 ],
88 },
89 plugins: [
90   new ESLintWebpackPlugin({
91     context: path.resolve(__dirname, "../src"),
92     exclude: "node_modules",
93     cache: true,
94     cacheLocation: path.resolve(
95       __dirname,
96       "../node_modules/.cache/.eslintcache"
97     ),
98   })
99 ],
```



```

149         },
150     },
151 ],
152 },
153 ],
154 ],
155 },
156 },
157 }),
158 ],
159 splitChunks: {
160     chunks: "all",
161 },
162 runtimeChunk: {
163     name: (entrypoint) => `runtime~${entrypoint.name}`,
164 },
165 },
166 resolve: {
167     extensions: [".jsx", ".js", ".json"],
168 },
169 mode: "production",
170 devtool: "source-map",
171 };

```

## — 其他配置

- package.json

```

1  {
2    "name": "react-cli",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "start": "npm run dev",
8      "dev": "cross-env NODE_ENV=development webpack serve --config
./config/webpack.dev.js",
9      "build": "cross-env NODE_ENV=production webpack --config
./config/webpack.prod.js"
10   },
11   "keywords": [],
12   "author": "",
13   "license": "ISC",
14   "devDependencies": {
15     "@babel/core": "^7.17.10",
16     "@pmmmwh/react-refresh-webpack-plugin": "^0.5.5",
17     "babel-loader": "^8.2.5",
18     "babel-preset-react-app": "^10.0.1",
19     "copy-webpack-plugin": "^10.2.4",
20     "cross-env": "^7.0.3",

```

```

21   "css-loader": "^6.7.1",
22   "css-minimizer-webpack-plugin": "^3.4.1",
23   "eslint-config-react-app": "^7.0.1",
24   "eslint-webpack-plugin": "^3.1.1",
25   "html-webpack-plugin": "^5.5.0",
26   "image-minimizer-webpack-plugin": "^3.2.3",
27   "imagemin": "^8.0.1",
28   "imagemin-gifsicle": "^7.0.0",
29   "imagemin-jpegtran": "^7.0.0",
30   "imagemin-optipng": "^8.0.0",
31   "imagemin-svgo": "^10.0.1",
32   "less-loader": "^10.2.0",
33   "mini-css-extract-plugin": "^2.6.0",
34   "postcss-loader": "^6.2.1",
35   "postcss-preset-env": "^7.5.0",
36   "react-refresh": "^0.13.0",
37   "sass-loader": "^12.6.0",
38   "style-loader": "^3.3.1",
39   "stylus-loader": "^6.2.0",
40   "webpack": "^5.72.0",
41   "webpack-cli": "^4.9.2",
42   "webpack-dev-server": "^4.9.0"
43 },
44 "dependencies": {
45   "antd": "^4.20.2",
46   "react": "^18.1.0",
47   "react-dom": "^18.1.0",
48   "react-router-dom": "^6.3.0"
49 },
50 "browserslist": ["last 2 version", "> 1%", "not dead"]
51 }

```

- .eslintrc.js

```

1  module.exports = {
2    extends: ["react-app"], // 继承 react 官方规则
3    parserOptions: {
4      babelOptions: {
5        presets: [
6          // 解决页面报错问题
7          ["babel-preset-react-app", false],
8          "babel-preset-react-app/prod",
9        ],
10     },
11   },
12 };

```

- babel.config.js

```

1  module.exports = {
2    // 使用react官方规则
3    presets: ["react-app"],
4  };

```

## — 合并开发和生产配置

- webpack.config.js

```

1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
5  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
6  const TerserWebpackPlugin = require("terser-webpack-plugin");
7  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
8  const ReactRefreshWebpackPlugin = require("@pmmmwh/react-refresh-webpack-
  plugin");
9
10 // 需要通过 cross-env 定义环境变量
11 const isProduction = process.env.NODE_ENV === "production";
12
13 const getStyleLoaders = (preProcessor) => {
14   return [
15     isProduction ? MiniCssExtractPlugin.loader : "style-loader",
16     "css-loader",
17     {
18       loader: "postcss-loader",
19       options: {
20         postcssOptions: {
21           plugins: [
22             "postcss-preset-env", // 能解决大多数样式兼容性问题
23           ],
24         },
25       },
26     },
27     preProcessor,
28   ].filter(Boolean);
29 };
30
31 module.exports = {
32   entry: "./src/main.js",
33   output: {
34     path: isProduction ? path.resolve(__dirname, "../dist") : undefined,
35     filename: isProduction
36       ? "static/js/[name].[contenthash:10].js"
37       : "static/js/[name].js",
38     chunkFilename: isProduction
39       ? "static/js/[name].[contenthash:10].chunk.js"

```

```
40     : "static/js/[name].chunk.js",
41     assetModuleFilename: "static/js/[hash:10][ext][query]",
42     clean: true,
43 },
44 module: {
45     rules: [
46         {
47             oneOf: [
48                 {
49                     // 用来匹配 .css 结尾的文件
50                     test: /\.css$/,
51                     // use 数组里面 Loader 执行顺序是从右到左
52                     use: getStyleLoaders(),
53                 },
54                 {
55                     test: /\.less$/,
56                     use: getStyleLoaders("less-loader"),
57                 },
58                 {
59                     test: /\.s[ac]ss$/,
60                     use: getStyleLoaders("sass-loader"),
61                 },
62                 {
63                     test: /\.styl$/,
64                     use: getStyleLoaders("stylus-loader"),
65                 },
66                 {
67                     test: /\.(png|jpe?g|gif|svg)$/,
68                     type: "asset",
69                     parser: {
70                         dataUrlCondition: {
71                             maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
72                         },
73                     },
74                 },
75                 {
76                     test: /\.(ttf|woff2?)$/,
77                     type: "asset/resource",
78                 },
79                 {
80                     test: /\.(jsx|js)$/,
81                     include: path.resolve(__dirname, ".. /src"),
82                     loader: "babel-loader",
83                     options: {
84                         cacheDirectory: true, // 开启babel编译缓存
85                         cacheCompression: false, // 缓存文件不要压缩
86                         plugins: [
87                             // "@babel/plugin-transform-runtime", // presets中包含了
88                             !isProduction && "react-refresh/babel",
89                         ].filter(Boolean),
90                     },
91                 },
92             ],
93         },
94     ],
95 },
```

```

91         },
92     ],
93 },
94 ],
95 },
96 plugins: [
97     new ESLintWebpackPlugin({
98         extensions: [".js", ".jsx"],
99         context: path.resolve(__dirname, "../src"),
100         exclude: "node_modules",
101         cache: true,
102         cacheLocation: path.resolve(
103             __dirname,
104             "../node_modules/.cache/.eslintcache"
105         ),
106     }),
107     new HtmlWebpackPlugin({
108         template: path.resolve(__dirname, "../public/index.html"),
109     }),
110     isProduction &&
111     new MiniCssExtractPlugin({
112         filename: "static/css/[name].[contenthash:10].css",
113         chunkFilename: "static/css/[name].[contenthash:10].chunk.css",
114     }),
115     !isProduction && new ReactRefreshWebpackPlugin(),
116 ].filter(Boolean),
117 optimization: {
118     minimize: isProduction,
119     // 压缩的操作
120     minimizer: [
121         // 压缩css
122         new CssMinimizerPlugin(),
123         // 压缩js
124         new TerserWebpackPlugin(),
125         // 压缩图片
126         new ImageMinimizerPlugin({
127             minimizer: {
128                 implementation: ImageMinimizerPlugin.imageminGenerate,
129                 options: {
130                     plugins: [
131                         ["gifsicle", { interlaced: true }],
132                         ["jpegtran", { progressive: true }],
133                         ["optipng", { optimizationLevel: 5 }],
134                         [
135                             "svgo",
136                             {
137                                 plugins: [
138                                     "preset-default",
139                                     "prefixIds",
140                                 ],
141                                 name: "sortAttrs",

```

```

142             params: {
143                 xmlnsOrder: "alphabetical",
144             },
145         },
146     ],
147 },
148 ],
149 ],
150 },
151 },
152 })),
153 ],
154 // 代码分割配置
155 splitChunks: {
156     chunks: "all",
157     // 其他都用默认值
158 },
159 runtimeChunk: {
160     name: (entrypoint) => `runtime~${entrypoint.name}`,
161 },
162 },
163 resolve: {
164     extensions: [".jsx", ".js", ".json"],
165 },
166 devServer: {
167     open: true,
168     host: "localhost",
169     port: 3000,
170     hot: true,
171     compress: true,
172     historyApiFallback: true,
173 },
174 mode: isProduction ? "production" : "development",
175 devtool: isProduction ? "source-map" : "cheap-module-source-map",
176 };

```

- 修改运行指令 package.json

```

1  {
2      "name": "react-cli",
3      "version": "1.0.0",
4      "description": "",
5      "main": "index.js",
6      "scripts": {
7          "start": "npm run dev",
8          "dev": "cross-env NODE_ENV=development webpack serve --config
./config/webpack.config.js",
9          "build": "cross-env NODE_ENV=production webpack --config
./config/webpack.config.js"
10     },

```



```

11   "keywords": [],
12   "author": "",
13   "license": "ISC",
14   "devDependencies": {
15     "@babel/core": "^7.17.10",
16     "@pmmmwh/react-refresh-webpack-plugin": "^0.5.5",
17     "babel-loader": "^8.2.5",
18     "babel-preset-react-app": "^10.0.1",
19     "cross-env": "^7.0.3",
20     "css-loader": "^6.7.1",
21     "css-minimizer-webpack-plugin": "^3.4.1",
22     "eslint-config-react-app": "^7.0.1",
23     "eslint-webpack-plugin": "^3.1.1",
24     "html-webpack-plugin": "^5.5.0",
25     "image-minimizer-webpack-plugin": "^3.2.3",
26     "imagemin": "^8.0.1",
27     "imagemin-gifsicle": "^7.0.0",
28     "imagemin-jpegtran": "^7.0.0",
29     "imagemin-optipng": "^8.0.0",
30     "imagemin-svgo": "^10.0.1",
31     "less-loader": "^10.2.0",
32     "mini-css-extract-plugin": "^2.6.0",
33     "react-refresh": "^0.13.0",
34     "sass-loader": "^12.6.0",
35     "style-loader": "^3.3.1",
36     "stylus-loader": "^6.2.0",
37     "webpack": "^5.72.0",
38     "webpack-cli": "^4.9.2",
39     "webpack-dev-server": "^4.9.0"
40   },
41   "dependencies": {
42     "react": "^18.1.0",
43     "react-dom": "^18.1.0",
44     "react-router-dom": "^6.3.0"
45   },
46   "browserslist": ["last 2 version", "> 1%", "not dead"]
47 }

```

## — 优化配置

```

1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
5  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
6  const TerserWebpackPlugin = require("terser-webpack-plugin");
7  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
8  const ReactRefreshWebpackPlugin = require("@pmmmwh/react-refresh-webpack-
9  plugin");
10 const CopyPlugin = require("copy-webpack-plugin");

```

```

10
11 const isProduction = process.env.NODE_ENV === "production";
12
13 const getStyleLoaders = (preProcessor) => {
14   return [
15     isProduction ? MiniCssExtractPlugin.loader : "style-loader",
16     "css-loader",
17     {
18       loader: "postcss-loader",
19       options: {
20         postcssOptions: {
21           plugins: [
22             "postcss-preset-env",
23           ],
24         },
25       },
26     },
27     preProcessor && {
28       loader: preProcessor,
29       options:
30         preProcessor === "less-loader"
31         ? {
32           // antd的自定义主题
33           lessOptions: {
34             modifyVars: {
35               // 其他主题色: https://ant.design/docs/react/customize-
theme-cn
36               "@primary-color": "#1DA57A", // 全局主色
37             },
38             javascriptEnabled: true,
39           },
40         }
41         : {},
42     },
43   ].filter(Boolean);
44 };
45
46 module.exports = {
47   entry: "./src/main.js",
48   output: {
49     path: isProduction ? path.resolve(__dirname, "../dist") : undefined,
50     filename: isProduction
51       ? "static/js/[name].[contenthash:10].js"
52       : "static/js/[name].js",
53     chunkFilename: isProduction
54       ? "static/js/[name].[contenthash:10].chunk.js"
55       : "static/js/[name].chunk.js",
56     assetModuleFilename: "static/js/[hash:10][ext][query]",
57     clean: true,
58   },
59   module: {

```

```
60     rules: [
61       {
62         oneOf: [
63           {
64             test: /\.css$/,
65             use: getStyleLoaders(),
66           },
67           {
68             test: /\.less$/,
69             use: getStyleLoaders("less-loader"),
70           },
71           {
72             test: /\.s[ac]ss$/,
73             use: getStyleLoaders("sass-loader"),
74           },
75           {
76             test: /\.styl$/,
77             use: getStyleLoaders("stylus-loader"),
78           },
79           {
80             test: /\.?(png|jpe?g|gif|svg)$/ ,
81             type: "asset",
82             parser: {
83               dataUrlCondition: {
84                 maxSize: 10 * 1024,
85               },
86             },
87           },
88           {
89             test: /\.?(ttf|woff2?)$/ ,
90             type: "asset/resource",
91           },
92           {
93             test: /\.?(jsx|js)$/ ,
94             include: path.resolve(__dirname, "../src"),
95             loader: "babel-loader",
96             options: {
97               cacheDirectory: true,
98               cacheCompression: false,
99               plugins: [
100                 // "@babel/plugin-transform-runtime", // presets中包含了
101                 !isProduction && "react-refresh/babel",
102               ].filter(Boolean),
103             },
104           },
105         ],
106       },
107     ],
108   },
109   plugins: [
110     new ESLintWebpackPlugin({
```

```
111     extensions: [".js", ".jsx"],
112     context: path.resolve(__dirname, "../src"),
113     exclude: "node_modules",
114     cache: true,
115     cacheLocation: path.resolve(
116       __dirname,
117       "../node_modules/.cache/.eslintcache"
118     ),
119   })),
120   new HtmlWebpackPlugin({
121     template: path.resolve(__dirname, "../public/index.html"),
122   }),
123   isProduction &&
124     new MiniCssExtractPlugin({
125       filename: "static/css/[name].[contenthash:10].css",
126       chunkFilename: "static/css/[name].[contenthash:10].chunk.css",
127     }),
128   !isProduction && new ReactRefreshWebpackPlugin(),
129   // 将public下面的资源复制到dist目录去 (除了index.html)
130   new CopyPlugin({
131     patterns: [
132       {
133         from: path.resolve(__dirname, "../public"),
134         to: path.resolve(__dirname, "../dist"),
135         toType: "dir",
136         noErrorOnMissing: true, // 不生成错误
137         globOptions: {
138           // 忽略文件
139           ignore: ["**/index.html"],
140         },
141         info: {
142           // 跳过terser压缩js
143           minimized: true,
144         },
145       },
146     ],
147   }),
148 ].filter(Boolean),
149 optimization: {
150   minimize: isProduction,
151   // 压缩的操作
152   minimizer: [
153     // 压缩css
154     new CssMinimizerPlugin(),
155     // 压缩js
156     new TerserWebpackPlugin(),
157     // 压缩图片
158     new ImageMinimizerPlugin({
159       minimizer: {
160         implementation: ImageMinimizerPlugin.imageminGenerate,
161         options: {
```

```

162         plugins: [
163             ["gifsicle", { interlaced: true }],
164             ["jpegtran", { progressive: true }],
165             ["optipng", { optimizationLevel: 5 }],
166             [
167                 "svgo",
168                 {
169                     plugins: [
170                         "preset-default",
171                         "prefixIds",
172                         {
173                             name: "sortAttrs",
174                             params: {
175                                 xmlnsOrder: "alphabetical",
176                             },
177                         },
178                     ],
179                 },
180             ],
181         ],
182     },
183 },
184 }),
185 ],
186 // 代码分割配置
187 splitChunks: {
188     chunks: "all",
189     cacheGroups: {
190         // layouts通常是admin项目的主体布局组件，所有路由组件都要使用的
191         // 可以单独打包，从而复用
192         // 如果项目中没有，请删除
193         layouts: {
194             name: "layouts",
195             test: path.resolve(__dirname, "../src/layouts"),
196             priority: 40,
197         },
198         // 如果项目中使用antd，此时将所有node_modules打包在一起，那么打包输出文件会比
199         // 较大。
200         // 所以我们将node_modules中比较大的模块单独打包，从而并行加载速度更好
201         // 如果项目中没有，请删除
202         antd: {
203             name: "chunk-antd",
204             test: /[\\/]node_modules[\\/]antd(.*)/,
205             priority: 30,
206         },
207         // 将react相关的库单独打包，减少node_modules的chunk体积。
208         react: {
209             name: "react",
210             test: /[\\/]node_modules[\\/]react(.*)?[\\/]$/,
211             chunks: "initial",
212             priority: 20,

```

```

212     },
213     libs: {
214       name: "chunk-libs",
215       test: /[\\/]node_modules[\\/]$/,
216       priority: 10, // 权重最低, 优先考虑前面内容
217       chunks: "initial",
218     },
219   },
220 },
221 runtimeChunk: {
222   name: (entrypoint) => `runtime~${entrypoint.name}`,
223 },
224 },
225 resolve: {
226   extensions: [".jsx", ".js", ".json"],
227 },
228 devServer: {
229   open: true,
230   host: "localhost",
231   port: 3000,
232   hot: true,
233   compress: true,
234   historyApiFallback: true,
235 },
236 mode: isProduction ? "production" : "development",
237 devtool: isProduction ? "source-map" : "cheap-module-source-map",
238 performance: false, // 关闭性能分析, 提示速度
239 };

```

## • Vue 脚手架

### – 开发模式配置

```

1  // webpack.dev.js
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const { VueLoaderPlugin } = require("vue-loader");
6  const { DefinePlugin } = require("webpack");
7  const CopyPlugin = require("copy-webpack-plugin");
8
9  const getStyleLoaders = (preProcessor) => {
10    return [
11      "vue-style-loader",
12      "css-loader",
13      {
14        loader: "postcss-loader",
15        options: {
16          postcssOptions: {

```

```
17     plugins: [
18         "postcss-preset-env", // 能解决大多数样式兼容性问题
19     ],
20     },
21     },
22     },
23     preProcessor,
24 ].filter(Boolean);
25 };
26
27 module.exports = {
28     entry: "./src/main.js",
29     output: {
30         path: undefined,
31         filename: "static/js/[name].js",
32         chunkFilename: "static/js/[name].chunk.js",
33         assetModuleFilename: "static/js/[hash:10][ext][query]",
34     },
35     module: {
36         rules: [
37             {
38                 // 用来匹配 .css 结尾的文件
39                 test: /\.css$/,
40                 // use 数组里面 Loader 执行顺序是从右到左
41                 use: getStyleLoaders(),
42             },
43             {
44                 test: /\.less$/,
45                 use: getStyleLoaders("less-loader"),
46             },
47             {
48                 test: /\.s[ac]ss$/,
49                 use: getStyleLoaders("sass-loader"),
50             },
51             {
52                 test: /\.styl$/,
53                 use: getStyleLoaders("stylus-loader"),
54             },
55             {
56                 test: /\.?(png|jpe?g|gif|svg)$/,
57                 type: "asset",
58                 parser: {
59                     dataUrlCondition: {
60                         maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
61                     },
62                 },
63             },
64             {
65                 test: /\.?(ttf|woff2?)$/,
66                 type: "asset/resource",
67             },
```

```
68     {
69       test: /\.(\.jsx|js)$/,
70       include: path.resolve(__dirname, "../src"),
71       loader: "babel-loader",
72       options: {
73         cacheDirectory: true,
74         cacheCompression: false,
75         plugins: [
76           // "@babel/plugin-transform-runtime" // presets中包含了
77         ],
78       },
79     },
80     // vue-loader不支持oneOf
81     {
82       test: /\.vue$/,
83       loader: "vue-loader", // 内部会给vue文件注入HMR功能代码
84       options: {
85         // 开启缓存
86         cacheDirectory: path.resolve(
87           __dirname,
88           "node_modules/.cache/vue-loader"
89         ),
90       },
91     },
92   ],
93 },
94 plugins: [
95   new ESLintWebpackPlugin({
96     context: path.resolve(__dirname, "../src"),
97     exclude: "node_modules",
98     cache: true,
99     cacheLocation: path.resolve(
100       __dirname,
101       "../node_modules/.cache/.eslintcache"
102     ),
103   }),
104   new HtmlWebpackPlugin({
105     template: path.resolve(__dirname, "../public/index.html"),
106   }),
107   new CopyPlugin({
108     patterns: [
109       {
110         from: path.resolve(__dirname, "../public"),
111         to: path.resolve(__dirname, "../dist"),
112         toType: "dir",
113         noErrorOnMissing: true,
114         globOptions: {
115           ignore: ["**/index.html"],
116         },
117         info: {
118           minimized: true,
```



```

119     },
120   },
121 ],
122 )),
123   new VueLoaderPlugin(),
124   // 解决页面警告
125   new DefinePlugin({
126     __VUE_OPTIONS_API__: "true",
127     __VUE_PROD_DEVTOOLS__: "false",
128   }),
129 ],
130   optimization: {
131     splitChunks: {
132       chunks: "all",
133     },
134     runtimeChunk: {
135       name: (entrypoint) => `runtime-${entrypoint.name}`,
136     },
137   },
138   resolve: {
139     extensions: [".vue", ".js", ".json"], // 自动补全文件扩展名, 让vue可以使用
140   },
141   devServer: {
142     open: true,
143     host: "localhost",
144     port: 3000,
145     hot: true,
146     compress: true,
147     historyApiFallback: true, // 解决vue-router刷新404问题
148   },
149   mode: "development",
150   devtool: "cheap-module-source-map",
151 };

```

## — 生产模式配置

```

1  // webpack.prod.js
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
7  const TerserWebpackPlugin = require("terser-webpack-plugin");
8  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
9  const { VueLoaderPlugin } = require("vue-loader");
10 const { DefinePlugin } = require("webpack");
11
12 const getStyleLoaders = (preProcessor) => {
13   return [
14     MiniCssExtractPlugin.loader,

```

```
15     "css-loader",
16     {
17         loader: "postcss-loader",
18         options: {
19             postcssOptions: {
20                 plugins: [
21                     "postcss-preset-env", // 能解决大多数样式兼容性问题
22                 ],
23             },
24         },
25     },
26     preProcessor,
27 ].filter(Boolean);
28 };
29
30 module.exports = {
31     entry: "./src/main.js",
32     output: {
33         path: undefined,
34         filename: "static/js/[name].[contenthash:10].js",
35         chunkFilename: "static/js/[name].[contenthash:10].chunk.js",
36         assetModuleFilename: "static/js/[hash:10][ext][query]",
37         clean: true,
38     },
39     module: {
40         rules: [
41             {
42                 // 用来匹配 .css 结尾的文件
43                 test: /\.css$/,
44                 // use 数组里面 Loader 执行顺序是从右到左
45                 use: getStyleLoaders(),
46             },
47             {
48                 test: /\.less$/,
49                 use: getStyleLoaders("less-loader"),
50             },
51             {
52                 test: /\.s[ac]ss$/,
53                 use: getStyleLoaders("sass-loader"),
54             },
55             {
56                 test: /\.styl$/,
57                 use: getStyleLoaders("stylus-loader"),
58             },
59             {
60                 test: /\.(png|jpe?g|gif|svg)$/,
61                 type: "asset",
62                 parser: {
63                     dataUrlCondition: {
64                         maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
65                     },
66                 },
67             },
68         ],
69     },
70 }
```

```
66     },
67   },
68   {
69     test: /\.(\.ttf|woff2?)$/,
70     type: "asset/resource",
71   },
72   {
73     test: /\.(\.jsx|\.js)$/,
74     include: path.resolve(__dirname, "../src"),
75     loader: "babel-loader",
76     options: {
77       cacheDirectory: true,
78       cacheCompression: false,
79       plugins: [
80         // "@babel/plugin-transform-runtime" // presets中包含了
81       ],
82     },
83   },
84   // vue-loader不支持oneOf
85   {
86     test: /\.vue$/,
87     loader: "vue-loader", // 内部会给vue文件注入HMR功能代码
88     options: {
89       // 开启缓存
90       cacheDirectory: path.resolve(
91         __dirname,
92         "node_modules/.cache/vue-loader"
93       ),
94     },
95   },
96 ],
97 },
98 plugins: [
99   new ESLintWebpackPlugin({
100     context: path.resolve(__dirname, "../src"),
101     exclude: "node_modules",
102     cache: true,
103     cacheLocation: path.resolve(
104       __dirname,
105       "../node_modules/.cache/.eslintcache"
106     ),
107   }),
108   new HtmlWebpackPlugin({
109     template: path.resolve(__dirname, "../public/index.html"),
110   }),
111   new CopyPlugin({
112     patterns: [
113       {
114         from: path.resolve(__dirname, "../public"),
115         to: path.resolve(__dirname, "../dist"),
116         toType: "dir",
```

```
117         noErrorOnMissing: true,
118         globOptions: {
119             ignore: ["**/index.html"],
120         },
121         info: {
122             minimized: true,
123         },
124     },
125 ],
126 }),
127 new MiniCssExtractPlugin({
128     filename: "static/css/[name].[contenthash:10].css",
129     chunkFilename: "static/css/[name].[contenthash:10].chunk.css",
130 }),
131 new VueLoaderPlugin(),
132 new DefinePlugin({
133     __VUE_OPTIONS_API__: "true",
134     __VUE_PROD_DEVTOOLS__: "false",
135 }),
136 ],
137 optimization: {
138     // 压缩的操作
139     minimizer: [
140         new CssMinimizerPlugin(),
141         new TerserWebpackPlugin(),
142         new ImageMinimizerPlugin({
143             minimizer: {
144                 implementation: ImageMinimizerPlugin.imageminGenerate,
145                 options: {
146                     plugins: [
147                         ["gifsicle", { interlaced: true }],
148                         ["jpegtran", { progressive: true }],
149                         ["optipng", { optimizationLevel: 5 }],
150                         [
151                             "svgo",
152                             {
153                                 plugins: [
154                                     "preset-default",
155                                     "prefixIds",
156                                     {
157                                         name: "sortAttrs",
158                                         params: {
159                                             xmlnsOrder: "alphabetical",
160                                         },
161                                     },
162                                 ],
163                             },
164                         ],
165                     ],
166                 },
167             },
168         )
169     ]
170 }
```

```

168     }},
169   ],
170   splitChunks: {
171     chunks: "all",
172   },
173   runtimeChunk: {
174     name: (entrypoint) => `runtime~${entrypoint.name}`,
175   },
176 },
177 resolve: {
178   extensions: [".vue", ".js", ".json"],
179 },
180 mode: "production",
181 devtool: "source-map",
182 };

```

## — 其他配置

- package.json

```

1  {
2    "name": "vue-cli",
3    "version": "1.0.0",
4    "description": "",
5    "main": "main.js",
6    "scripts": {
7      "start": "npm run dev",
8      "dev": "cross-env NODE_ENV=development webpack serve --config
./config/webpack.dev.js",
9      "build": "cross-env NODE_ENV=production webpack --config
./config/webpack.prod.js"
10   },
11   "keywords": [],
12   "author": "",
13   "license": "ISC",
14   "devDependencies": {
15     "@babel/core": "^7.17.10",
16     "@babel/eslint-parser": "^7.17.0",
17     "@vue/cli-plugin-babel": "^5.0.4",
18     "babel-loader": "^8.2.5",
19     "copy-webpack-plugin": "^10.2.4",
20     "cross-env": "^7.0.3",
21     "css-loader": "^6.7.1",
22     "css-minimizer-webpack-plugin": "^3.4.1",
23     "eslint-plugin-vue": "^8.7.1",
24     "eslint-webpack-plugin": "^3.1.1",
25     "html-webpack-plugin": "^5.5.0",
26     "image-minimizer-webpack-plugin": "^3.2.3",
27     "imagemin": "^8.0.1",
28     "imagemin-gifsicle": "^7.0.0",

```

```

29   "imagemin-jpegtran": "^7.0.0",
30   "imagemin-optipng": "^8.0.0",
31   "imagemin-svgo": "^10.0.1",
32   "less-loader": "^10.2.0",
33   "mini-css-extract-plugin": "^2.6.0",
34   "postcss-preset-env": "^7.5.0",
35   "sass-loader": "^12.6.0",
36   "stylus-loader": "^6.2.0",
37   "vue-loader": "^17.0.0",
38   "vue-style-loader": "^4.1.3",
39   "vue-template-compiler": "^2.6.14",
40   "webpack": "^5.72.0",
41   "webpack-cli": "^4.9.2",
42   "webpack-dev-server": "^4.9.0"
43 },
44 "dependencies": {
45   "vue": "^3.2.33",
46   "vue-router": "^4.0.15"
47 },
48 "browserslist": ["last 2 version", "> 1%", "not dead"]
49 }

```

- .eslintrc.js

```

1  module.exports = {
2    root: true,
3    env: {
4      node: true,
5    },
6    extends: ["plugin:vue/vue3-essential", "eslint:recommended"],
7    parserOptions: {
8      parser: "@babel/eslint-parser",
9    },
10 };

```

- babel.config.js

```

1  module.exports = {
2    presets: ["@vue/cli-plugin-babel/preset"],
3  };

```

## – 合并开发和生产配置

```

1  // webpack.config.js
2  const path = require("path");
3  const ESLintWebpackPlugin = require("eslint-webpack-plugin");

```

```
4  const HtmlWebpackPlugin = require("html-webpack-plugin");
5  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
6  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");
7  const TerserWebpackPlugin = require("terser-webpack-plugin");
8  const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
9  const { VueLoaderPlugin } = require("vue-loader");
10 const { DefinePlugin } = require("webpack");
11 const CopyPlugin = require("copy-webpack-plugin");
12
13 // 需要通过 cross-env 定义环境变量
14 const isProduction = process.env.NODE_ENV === "production";
15
16 const getStyleLoaders = (preProcessor) => {
17   return [
18     isProduction ? MiniCssExtractPlugin.loader : "vue-style-loader",
19     "css-loader",
20     {
21       loader: "postcss-loader",
22       options: {
23         postcssOptions: {
24           plugins: ["postcss-preset-env"],
25         },
26       },
27     },
28     preProcessor,
29   ].filter(Boolean);
30 };
31
32 module.exports = {
33   entry: "./src/main.js",
34   output: {
35     path: isProduction ? path.resolve(__dirname, "../dist") : undefined,
36     filename: isProduction
37       ? "static/js/[name].[contenthash:10].js"
38       : "static/js/[name].js",
39     chunkFilename: isProduction
40       ? "static/js/[name].[contenthash:10].chunk.js"
41       : "static/js/[name].chunk.js",
42     assetModuleFilename: "static/js/[hash:10][ext][query]",
43     clean: true,
44   },
45   module: {
46     rules: [
47       {
48         // 用来匹配 .css 结尾的文件
49         test: /\.css$/,
50         // use 数组里面 Loader 执行顺序是从右到左
51         use: getStyleLoaders(),
52       },
53       {
54         test: /\.less$/,
```

```
55     use: getStyleLoaders("less-loader"),
56   },
57   {
58     test: /\.s[ac]ss$/,
59     use: getStyleLoaders("sass-loader"),
60   },
61   {
62     test: /\.styl$/,
63     use: getStyleLoaders("stylus-loader"),
64   },
65   {
66     test: /\..(png|jpe?g|gif|svg)$/,
67     type: "asset",
68     parser: {
69       dataUrlCondition: {
70         maxSize: 10 * 1024, // 小于10kb的图片会被base64处理
71       },
72     },
73   },
74   {
75     test: /\..(ttf|woff2?)$/,
76     type: "asset/resource",
77   },
78   {
79     test: /\..(jsx|js)$/,
80     include: path.resolve(__dirname, "../src"),
81     loader: "babel-loader",
82     options: {
83       cacheDirectory: true,
84       cacheCompression: false,
85       plugins: [
86         // "@babel/plugin-transform-runtime" // presets中包含了
87       ],
88     },
89   },
90   // vue-loader不支持oneOf
91   {
92     test: /\.vue$/,
93     loader: "vue-loader", // 内部会给vue文件注入HMR功能代码
94     options: {
95       // 开启缓存
96       cacheDirectory: path.resolve(
97         __dirname,
98         "node_modules/.cache/vue-loader"
99       ),
100     },
101   },
102 ],
103 },
104 plugins: [
105   new ESLintWebpackPlugin({
```



```
106     context: path.resolve(__dirname, "../src"),
107     exclude: "node_modules",
108     cache: true,
109     cacheLocation: path.resolve(
110         __dirname,
111         "../node_modules/.cache/.eslintcache"
112     ),
113   }),
114   new HtmlWebpackPlugin({
115     template: path.resolve(__dirname, "../public/index.html"),
116   }),
117   new CopyPlugin({
118     patterns: [
119       {
120         from: path.resolve(__dirname, "../public"),
121         to: path.resolve(__dirname, "../dist"),
122         toType: "dir",
123         noErrorOnMissing: true,
124         globOptions: {
125           ignore: ["**/index.html"],
126         },
127         info: {
128           minimized: true,
129         },
130       },
131     ],
132   }),
133   isProduction &&
134     new MiniCssExtractPlugin({
135       filename: "static/css/[name].[contenthash:10].css",
136       chunkFilename: "static/css/[name].[contenthash:10].chunk.css",
137     }),
138   new VueLoaderPlugin(),
139   new DefinePlugin({
140     __VUE_OPTIONS_API__: "true",
141     __VUE_PROD_DEVTOOLS__: "false",
142   }),
143 ].filter(Boolean),
144 optimization: {
145   minimize: isProduction,
146   // 压缩的操作
147   minimizer: [
148     new CssMinimizerPlugin(),
149     new TerserWebpackPlugin(),
150     new ImageMinimizerPlugin({
151       minimizer: {
152         implementation: ImageMinimizerPlugin.imageminGenerate,
153         options: {
154           plugins: [
155             ["gifsicle", { interlaced: true }],
156             ["jpegtran", { progressive: true }],
```

```

157         ["optipng", { optimizationLevel: 5 }],
158         [
159             "svgo",
160             {
161                 plugins: [
162                     "preset-default",
163                     "prefixIds",
164                     {
165                         name: "sortAttrs",
166                         params: {
167                             xmlnsOrder: "alphabetical",
168                         },
169                     },
170                 ],
171             },
172         ],
173     ],
174 },
175 },
176 }),
177 ],
178 splitChunks: {
179     chunks: "all",
180 },
181 runtimeChunk: {
182     name: (entrypoint) => `runtime~${entrypoint.name}`,
183 },
184 },
185 resolve: {
186     extensions: [".vue", ".js", ".json"],
187 },
188 devServer: {
189     open: true,
190     host: "localhost",
191     port: 3000,
192     hot: true,
193     compress: true,
194     historyApiFallback: true, // 解决vue-router刷新404问题
195 },
196 mode: isProduction ? "production" : "development",
197 devtool: isProduction ? "source-map" : "cheap-module-source-map",
198 };

```

## — 优化配置

```

1  const path = require("path");
2  const ESLintWebpackPlugin = require("eslint-webpack-plugin");
3  const HtmlWebpackPlugin = require("html-webpack-plugin");
4  const MiniCssExtractPlugin = require("mini-css-extract-plugin");
5  const CssMinimizerPlugin = require("css-minimizer-webpack-plugin");

```

```

6   const ImageMinimizerPlugin = require("image-minimizer-webpack-plugin");
7   const TerserWebpackPlugin = require("terser-webpack-plugin");
8   const CopyPlugin = require("copy-webpack-plugin");
9   const { VueLoaderPlugin } = require("vue-loader");
10  const { DefinePlugin } = require("webpack");
11  const AutoImport = require("unplugin-auto-import/webpack");
12  const Components = require("unplugin-vue-components/webpack");
13  const { ElementPlusResolver } = require("unplugin-vue-components/resolvers");
14  // 需要通过 cross-env 定义环境变量
15  const isProduction = process.env.NODE_ENV === "production";
16
17  const getStyleLoaders = (preProcessor) => {
18    return [
19      isProduction ? MiniCssExtractPlugin.loader : "vue-style-loader",
20      "css-loader",
21      {
22        loader: "postcss-loader",
23        options: {
24          postcssOptions: {
25            plugins: ["postcss-preset-env"],
26          },
27        },
28      },
29      preProcessor && {
30        loader: preProcessor,
31        options: {
32          preProcessor === "sass-loader"
33            ? {
34              // 自定义主题: 自动引入我们定义的scss文件
35              additionalData: `@use "@styles/element/index.scss" as *;`,
36            }
37            : {},
38        },
39      ].filter(Boolean);
40  };
41
42  module.exports = {
43    entry: "./src/main.js",
44    output: {
45      path: isProduction ? path.resolve(__dirname, "../dist") : undefined,
46      filename: isProduction
47        ? "static/js/[name].[contenthash:10].js"
48        : "static/js/[name].js",
49      chunkFilename: isProduction
50        ? "static/js/[name].[contenthash:10].chunk.js"
51        : "static/js/[name].chunk.js",
52      assetModuleFilename: "static/js/[hash:10][ext][query]",
53      clean: true,
54    },
55    module: {

```

```
56     rules: [
57       {
58         test: /\.css$/,
59         use: getStyleLoaders(),
60       },
61       {
62         test: /\.less$/,
63         use: getStyleLoaders("less-loader"),
64       },
65       {
66         test: /\.s[ac]ss$/,
67         use: getStyleLoaders("sass-loader"),
68       },
69       {
70         test: /\.styl$/,
71         use: getStyleLoaders("stylus-loader"),
72       },
73       {
74         test: /\.?(png|jpe?g|gif|svg)$/,
75         type: "asset",
76         parser: {
77           dataUrlCondition: {
78             maxSize: 10 * 1024,
79           },
80         },
81       },
82       {
83         test: /\.?(ttf|woff2?)$/,
84         type: "asset/resource",
85       },
86       {
87         test: /\.?(jsx|js)$/,
88         include: path.resolve(__dirname, "../src"),
89         loader: "babel-loader",
90         options: {
91           cacheDirectory: true,
92           cacheCompression: false,
93           plugins: [
94             // "@babel/plugin-transform-runtime" // presets中包含了
95           ],
96         },
97       },
98       // vue-loader不支持oneOf
99       {
100         test: /\.vue$/,
101         loader: "vue-loader", // 内部会给vue文件注入HMR功能代码
102         options: {
103           // 开启缓存
104           cacheDirectory: path.resolve(
105             __dirname,
106             "node_modules/.cache/vue-loader"
```

```

107         },
108     },
109 },
110 ],
111 },
112 plugins: [
113     new ESLintWebpackPlugin({
114         context: path.resolve(__dirname, "../src"),
115         exclude: "node_modules",
116         cache: true,
117         cacheLocation: path.resolve(
118             __dirname,
119             "../node_modules/.cache/.eslintcache"
120         ),
121     }),
122     new HtmlWebpackPlugin({
123         template: path.resolve(__dirname, "../public/index.html"),
124     }),
125     new CopyPlugin({
126         patterns: [
127             {
128                 from: path.resolve(__dirname, "../public"),
129                 to: path.resolve(__dirname, "../dist"),
130                 toType: "dir",
131                 noErrorOnMissing: true,
132                 globOptions: {
133                     ignore: ["**/index.html"],
134                 },
135                 info: {
136                     minimized: true,
137                 },
138             },
139         ],
140     }),
141     isProduction &&
142     new MiniCssExtractPlugin({
143         filename: "static/css/[name].[contenthash:10].css",
144         chunkFilename: "static/css/[name].[contenthash:10].chunk.css",
145     }),
146     new VueLoaderPlugin(),
147     new DefinePlugin({
148         __VUE_OPTIONS_API__: "true",
149         __VUE_PROD_DEVTOOLS__: "false",
150     }),
151     // 按需加载element-plus组件样式
152     AutoImport({
153         resolvers: [ElementPlusResolver()],
154     }),
155     Components({
156         resolvers: [
157             ElementPlusResolver({

```

```

158         importStyle: "sass", // 自定义主题
159     }),
160 ],
161 }),
162 ].filter(Boolean),
163 optimization: {
164     minimize: isProduction,
165     // 压缩的操作
166     minimizer: [
167         new CssMinimizerPlugin(),
168         new TerserWebpackPlugin(),
169         new ImageMinimizerPlugin({
170             minimizer: {
171                 implementation: ImageMinimizerPlugin.imageminGenerate,
172                 options: {
173                     plugins: [
174                         ["gifsicle", { interlaced: true }],
175                         ["jpegtran", { progressive: true }],
176                         ["optipng", { optimizationLevel: 5 }],
177                         [
178                             "svgo",
179                             {
180                                 plugins: [
181                                     "preset-default",
182                                     "prefixIds",
183                                     {
184                                         name: "sortAttrs",
185                                         params: {
186                                             xmlnsOrder: "alphabetical",
187                                         },
188                                     },
189                                 ],
190                             },
191                         ],
192                     ],
193                 },
194             },
195         }),
196     ],
197     splitChunks: {
198         chunks: "all",
199         cacheGroups: {
200             // layouts通常是admin项目的主体布局组件，所有路由组件都要使用的
201             // 可以单独打包，从而复用
202             // 如果项目中没有，请删除
203             layouts: {
204                 name: "layouts",
205                 test: path.resolve(__dirname, "../src/layouts"),
206                 priority: 40,
207             },

```

```

208 // 如果项目中使用element-plus, 此时将所有node_modules打包在一起, 那么打包输出文件会比较大。
209 // 所以我们将node_modules中比较大的模块单独打包, 从而并行加载速度更好
210 // 如果项目中没有, 请删除
211 elementUI: {
212   name: "chunk-elementPlus",
213   test: /[\\/]node_modules[\\/]_?element-plus(.*)/,
214   priority: 30,
215 },
216 // 将vue相关的库单独打包, 减少node_modules的chunk体积。
217 vue: {
218   name: "vue",
219   test: /[\\/]node_modules[\\/]vue(.*)[\\/]$/,
220   chunks: "initial",
221   priority: 20,
222 },
223 libs: {
224   name: "chunk-libs",
225   test: /[\\/]node_modules[\\/]$/,
226   priority: 10, // 权重最低, 优先考虑前面内容
227   chunks: "initial",
228 },
229 },
230 },
231 runtimeChunk: {
232   name: (entrypoint) => `runtime~${entrypoint.name}`,
233 },
234 },
235 resolve: {
236   extensions: [".vue", ".js", ".json"],
237   alias: {
238     // 路径别名
239     "@": path.resolve(__dirname, "../src"),
240   },
241 },
242 devServer: {
243   open: true,
244   host: "localhost",
245   port: 3000,
246   hot: true,
247   compress: true,
248   historyApiFallback: true, // 解决vue-router刷新404问题
249 },
250 mode: isProduction ? "production" : "development",
251 devtool: isProduction ? "source-map" : "cheap-module-source-map",
252 performance: false,
253 };

```

## 原理分析

## • Loader 原理

### – loader 概念

帮助 webpack 将不同类型的文件转换为 webpack 可识别的模块。

### – loader 执行顺序

#### 1. 分类

- pre: 前置 loader
- normal: 普通 loader
- inline: 内联 loader
- post: 后置 loader

#### 2. 执行顺序

- 4 类 loader 的执行优先级为: `pre > normal > inline > post` 。
- 相同优先级的 loader 执行顺序为: `从右到左, 从下到上` 。

例如:

```
1  // 此时loader执行顺序: loader3 - loader2 - loader1
2  module: {
3    rules: [
4      {
5        test: /\.js$/,
6        loader: "loader1",
7      },
8      {
9        test: /\.js$/,
10       loader: "loader2",
11     },
12     {
13       test: /\.js$/,
14       loader: "loader3",
15     },
16   ],
17 }
```

```
1  // 此时loader执行顺序: loader1 - loader2 - loader3
2  module: {
3    rules: [
4      {
5        enforce: "pre",
6        test: /\.js$/,
7        loader: "loader1",
8      },
9      {
```



```

10     // 没有enforce就是normal
11     test: /\.js$/,
12     loader: "loader2",
13   },
14   {
15     enforce: "post",
16     test: /\.js$/,
17     loader: "loader3",
18   },
19 ],
20 },

```

### 3. 使用 loader 的方式

- 配置方式：在 `webpack.config.js` 文件中指定 loader。（pre、normal、post loader）
- 内联方式：在每个 `import` 语句中显式指定 loader。（inline loader）

### 4. inline loader

用法： `import Styles from 'style-loader!css-loader?modules!./styles.css';`

含义：

- 使用 `css-loader` 和 `style-loader` 处理 `styles.css` 文件
- 通过 `!` 将资源中的 loader 分开

`inline loader` 可以通过添加不同前缀，跳过其他类型 loader。

- `!` 跳过 normal loader。

`import Styles from '!style-loader!css-loader?modules!./styles.css';`

- `-!` 跳过 pre 和 normal loader。

`import Styles from '-!style-loader!css-loader?modules!./styles.css';`

- `!!` 跳过 pre、normal 和 post loader。

`import Styles from '!!style-loader!css-loader?modules!./styles.css';`

## • 开发一个 loader

### - 1. 最简单的 loader

```

1  // loaders/loader1.js
2  module.exports = function loader1(content) {
3    console.log("hello loader");
4    return content;
5  };

```

它接受要处理的源码作为参数，输出转换后的 js 代码。

## – 2. loader 接受的参数

- `content` 源文件的内容
- `map` SourceMap 数据
- `meta` 数据，可以是任何内容

## • loader 分类

### – 1. 同步 loader

```
1 module.exports = function (content, map, meta) {  
2   return content;  
3 };
```

`this.callback` 方法则更灵活，因为它允许传递多个参数，而不仅仅是 `content`。

```
1 module.exports = function (content, map, meta) {  
2   // 传递map, 让source-map不中断  
3   // 传递meta, 让下一个loader接收到其他参数  
4   this.callback(null, content, map, meta);  
5   return; // 当调用 callback() 函数时, 总是返回 undefined  
6 };
```

### – 2. 异步 loader

```
1 module.exports = function (content, map, meta) {  
2   const callback = this.async();  
3   // 进行异步操作  
4   setTimeout(() => {  
5     callback(null, result, map, meta);  
6   }, 1000);  
7 };
```

“

由于同步计算过于耗时，在 Node.js 这样的单线程环境下进行此操作并不是好的方案，我们建议尽可能地使你的 loader 异步化。但如果计算量很小，同步 loader 也是可以的。

### – 3. Raw Loader

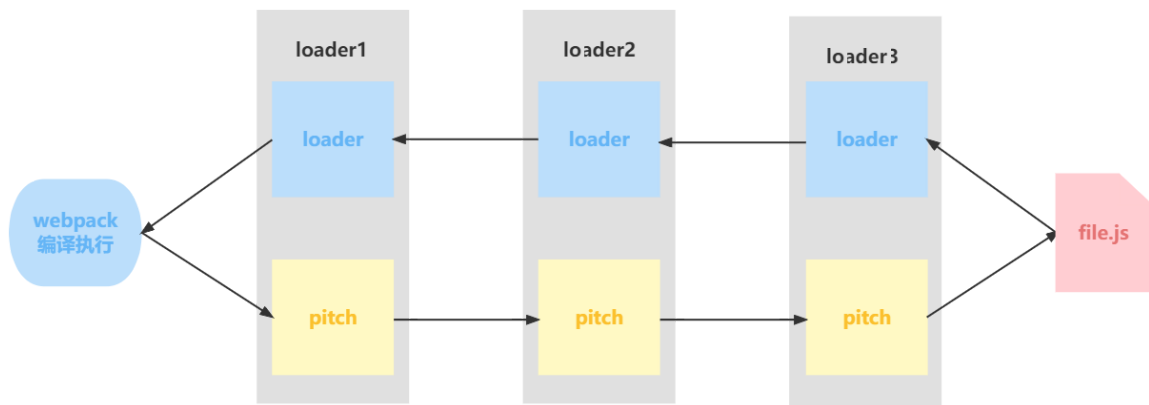
默认情况下，资源文件会被转化为 UTF-8 字符串，然后传给 loader。通过设置 raw 为 true，loader 可以接收原始的 Buffer。

```
1 module.exports = function (content) {
2   // content是一个Buffer数据
3   return content;
4 };
5 module.exports.raw = true; // 开启 Raw Loader
```

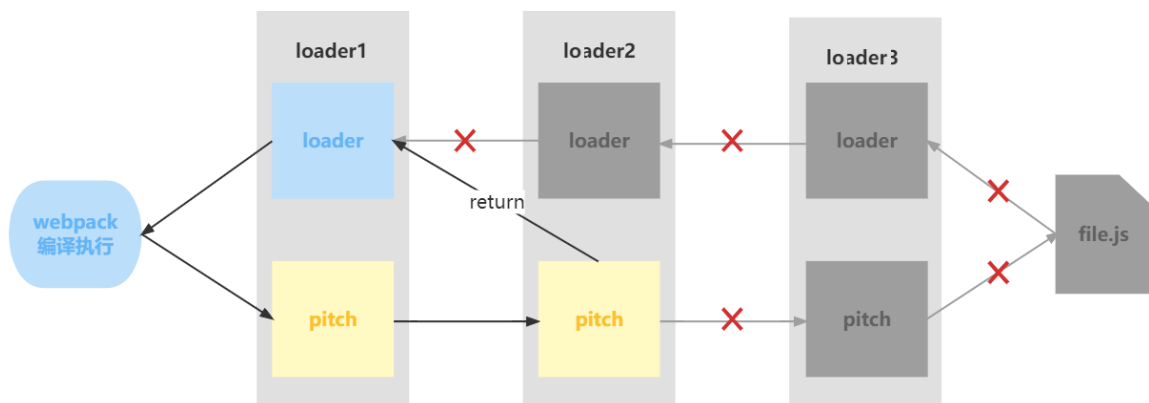
### – 4. Pitching Loader

```
1 module.exports = function (content) {
2   return content;
3 };
4 module.exports.pitch = function (remainingRequest, precedingRequest, data) {
5   console.log("do somethings");
6 };
```

webpack 会先从左到右执行 loader 链中的每个 loader 上的 pitch 方法（如果有），然后再从右到左执行 loader 链中的每个 loader 上的普通 loader 方法。



在这个过程中如果任何 pitch 有返回值，则 loader 链被阻断。webpack 会跳过后面所有的 pitch 和 loader，直接进入上一个 loader。



## • loader API

方法名	含义	用法
this.async	异步回调 loader。返回 this.callback	const callback = this.async()
this.callback	可以同步或者异步调用的并返回多个结果的函数	this.callback(err, content, sourceMap?, meta?)
this.getOptions(schema)	获取 loader 的 options	this.getOptions(schema)
this.emitFile	产生一个文件	this.emitFile(name, content, sourceMap)
this.utils.contextify	返回一个相对路径	this.utils.contextify(context, request)
this.utils.absolutify	返回一个绝对路径	this.utils.absolutify(context, request)

“

更多文档，请查阅 [webpack 官方 loader api 文档](#)

## • 手写 clean-log-loader

作用：用来清理 js 代码中的 `console.log`

```
1 // loaders/clean-log-loader.js
2 module.exports = function cleanLogLoader(content) {
3   // 将console.log替换为空
4   return content.replace(/console\.log\(.*\);?/g, "");
5 };
```

## • 手写 banner-loader

作用：给 js 代码添加文本注释

- loaders/banner-loader/index.js

```
1 const schema = require("./schema.json");
2
3 module.exports = function (content) {
4   // 获取loader的options, 同时对options内容进行校验
5   // schema是options的校验规则 (符合 JSON schema 规则)
6   const options = this.getOptions(schema);
```

```

7
8   const prefix = `
9     /*
10    * Author: ${options.author}
11    */
12   `;
13
14   return `${prefix} \n ${content}`;
15 };

```

- loaders/banner-loader/schema.json

```

1  {
2    "type": "object",
3    "properties": {
4      "author": {
5        "type": "string"
6      }
7    },
8    "additionalProperties": false
9  }

```

## • 手写 babel-loader

作用：编译 js 代码，将 ES6+语法编译成 ES5-语法。

- 下载依赖

```

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

```

- loaders/babel-loader/index.js

```

1  const schema = require("./schema.json");
2  const babel = require("@babel/core");
3
4  module.exports = function (content) {
5    const options = this.getOptions(schema);
6    // 使用异步loader
7    const callback = this.async();
8    // 使用babel对js代码进行编译
9    babel.transform(content, options, function (err, result) {
10      callback(err, result.code);
11    });
12  };

```

- loaders/banner-loader/schema.json

```
1  {
2    "type": "object",
3    "properties": {
4      "presets": {
5        "type": "array"
6      }
7    },
8    "additionalProperties": true
9  }
```

## • 手写 file-loader

作用：将文件原封不动输出出去

- 下载包

```
1  npm i loader-utils -D
```

- loaders/file-loader.js

```
1  const loaderUtils = require("loader-utils");
2
3  function fileLoader(content) {
4    // 根据文件内容生产一个新的文件名称
5    const filename = loaderUtils.interpolateName(this, "[hash].[ext]", {
6      content,
7    });
8    // 输出文件
9    this.emitFile(filename, content);
10   // 暴露出去, 给js引用。
11   // 记得加上''
12   return `export default '${filename}'`;
13 }
14
15 // loader 解决的是二进制的内容
16 // 图片是 Buffer 数据
17 fileLoader.raw = true;
18
19 module.exports = fileLoader;
```

- loader 配置

```

1  {
2    test: /\.?(png|jpe?g|gif)$/ ,
3    loader: "./loaders/file-loader.js",
4    type: "javascript/auto", // 解决图片重复打包问题
5  },

```

## • 手写 style-loader

作用：动态创建 style 标签，插入 js 中的样式代码，使样式生效。

- loaders/style-loader.js

```

1  const styleLoader = () => {};
2
3  styleLoader.pitch = function (remainingRequest) {
4    /*
5      remainingRequest: C:\Users\86176\Desktop\source\node_modules\css-
6      loader\dist\cjs.js!C:\Users\86176\Desktop\source\src\css\index.css
7      这里是inline loader用法，代表后面还有一个css-loader等待处理
8
9      最终我们需要将remainingRequest中的路径转化成相对路径，webpack才能处理
10     希望得到： ../.. /node_modules/css-loader/dist/cjs.js!./index.css
11
12     所以：需要将绝对路径转化成相对路径
13     要求：
14       1. 必须是相对路径
15       2. 相对路径必须以 ./ 或 ../ 开头
16       3. 相对路径的路径分隔符必须是 / ， 不能是 \
17     */
18     const relativeRequest = remainingRequest
19       .split("!")
20       .map((part) => {
21         // 将路径转化为相对路径
22         const relativePath = this.utils.contextify(this.context, part);
23         return relativePath;
24       })
25       .join("!");
26
27     /*
28       !! ${relativeRequest}
29       relativeRequest: ../.. /node_modules/css-loader/dist/cjs.js!./index.css
30       relativeRequest是inline loader用法，代表要处理的index.css资源，使用css-
31       loader处理
32       !! 代表禁用所有配置的loader，只使用inline loader。（也就是外面我们style-loader
33       和css-loader），它们被禁用了，只是用我们指定的inline loader，也就是css-loader
34
35       import style from "!!${relativeRequest}"
36       引入css-loader处理后的css文件
37       为什么需要css-loader处理css文件，不是我们直接读取css文件使用呢？

```

```

35     因为可能存在@import导入css语法，这些语法就要通过css-loader解析才能变成一个css文件，否则我们引入的css资源会缺少
36     const styleEl = document.createElement('style')
37     动态创建style标签
38     styleEl.innerHTML = style
39     将style标签内容设置为处理后的css代码
40     document.head.appendChild(styleEl)
41     添加到head中生效
42     */
43     const script = `
44         import style from "!!${relativeRequest}"
45         const styleEl = document.createElement('style')
46         styleEl.innerHTML = style
47         document.head.appendChild(styleEl)
48     `;
49
50     // style-loader是第一个loader，由于return导致熔断，所以其他loader不执行了（不管是normal还是pitch）
51     return script;
52 };
53
54 module.exports = styleLoader;

```

## Plugin 原理

### • Plugin 的作用

通过插件我们可以扩展 webpack，加入自定义的构建行为，使 webpack 可以执行更广泛的任务，拥有更强的构建能力。

### • Plugin 工作原理

“

webpack 就像一条生产线，要经过一系列处理流程后才能将源文件转换成输出结果。这条生产线上的每个处理流程的职责都是单一的，多个流程之间存在依赖关系，只有完成当前处理后才能交给下一个流程去处理。

插件就像是一个插入到生产线中的一个功能，在特定的时机对生产线上的资源做处理。webpack 通过 Tapable 来组织这条复杂的生产线。webpack 在运行过程中会广播事件，插件只需要监听它所关心的事件，就能加入到这条生产线中，去改变生产线的运作。

webpack 的事件流机制保证了插件的有序性，使得整个系统扩展性很好。

——「深入浅出 Webpack」

站在代码逻辑的角度就是：webpack 在编译代码过程中，会触发一系列 Tapable 钩子事件，插件所做的，就是找到相应的钩子，往上面挂上自己的任务，也就是注册事件，这样，当 webpack 构建的时候，插件注册的事件就会随着钩子的触发而执行了。



## • Webpack 内部的钩子

### - 什么是钩子

钩子的本质就是：事件。为了方便我们直接介入和控制编译过程，webpack 把编译过程中触发的各类关键事件封装成事件接口暴露了出来。这些接口被很形象地称做：`hooks`（钩子）。开发插件，离不开这些钩子。

### - Tappable

`Tappable` 为 webpack 提供了统一的插件接口（钩子）类型定义，它是 webpack 的核心功能库。webpack 中目前有十种 `hooks`，在 `Tappable` 源码中可以看到，他们是：

```
1 // https://github.com/webpack/tappable/blob/master/lib/index.js
2 exports.SyncHook = require("./SyncHook");
3 exports.SyncBailHook = require("./SyncBailHook");
4 exports.SyncWaterfallHook = require("./SyncWaterfallHook");
5 exports.SyncLoopHook = require("./SyncLoopHook");
6 exports.AsyncParallelHook = require("./AsyncParallelHook");
7 exports.AsyncParallelBailHook = require("./AsyncParallelBailHook");
8 exports.AsyncSeriesHook = require("./AsyncSeriesHook");
9 exports.AsyncSeriesBailHook = require("./AsyncSeriesBailHook");
10 exports.AsyncSeriesLoopHook = require("./AsyncSeriesLoopHook");
11 exports.AsyncSeriesWaterfallHook = require("./AsyncSeriesWaterfallHook");
12 exports.HookMap = require("./HookMap");
13 exports.MultiHook = require("./MultiHook");
```

`Tappable` 还统一暴露了三个方法给插件，用于注入不同类型的自定义构建行为：

- `tap`：可以注册同步钩子和异步钩子。
- `tapAsync`：回调方式注册异步钩子。
- `tapPromise`：Promise 方式注册异步钩子。

## • Plugin 构建对象

### - Compiler

compiler 对象中保存着完整的 Webpack 环境配置，每次启动 webpack 构建时它都是一个独一无二，仅仅会创建一次的对象。

这个对象会在首次启动 Webpack 时创建，我们可以通过 compiler 对象上访问到 Webpack 的主环境配置，比如 loader、plugin 等等配置信息。

它有以下主要属性：

- `compiler.options` 可以访问本次启动 webpack 时候所有的配置文件，包括但不限于 loaders、entry、output、plugin 等等完整配置信息。

- `compiler.inputFileSystem` 和 `compiler.outputFileSystem` 可以进行文件操作，相当于 Nodejs 中 fs。
- `compiler.hooks` 可以注册 tapable 的不同种类 Hook，从而可以在 compiler 生命周期中植入不同的逻辑。

“

[compiler hooks 文档](#)

## - Compilation

compilation 对象代表一次资源的构建，compilation 实例能够访问所有的模块和它们的依赖。

一个 compilation 对象会对构建依赖图中所有模块，进行编译。在编译阶段，模块会被加载(load)、封存(seal)、优化(optimize)、分块(chunk)、哈希(hash)和重新创建(restore)。

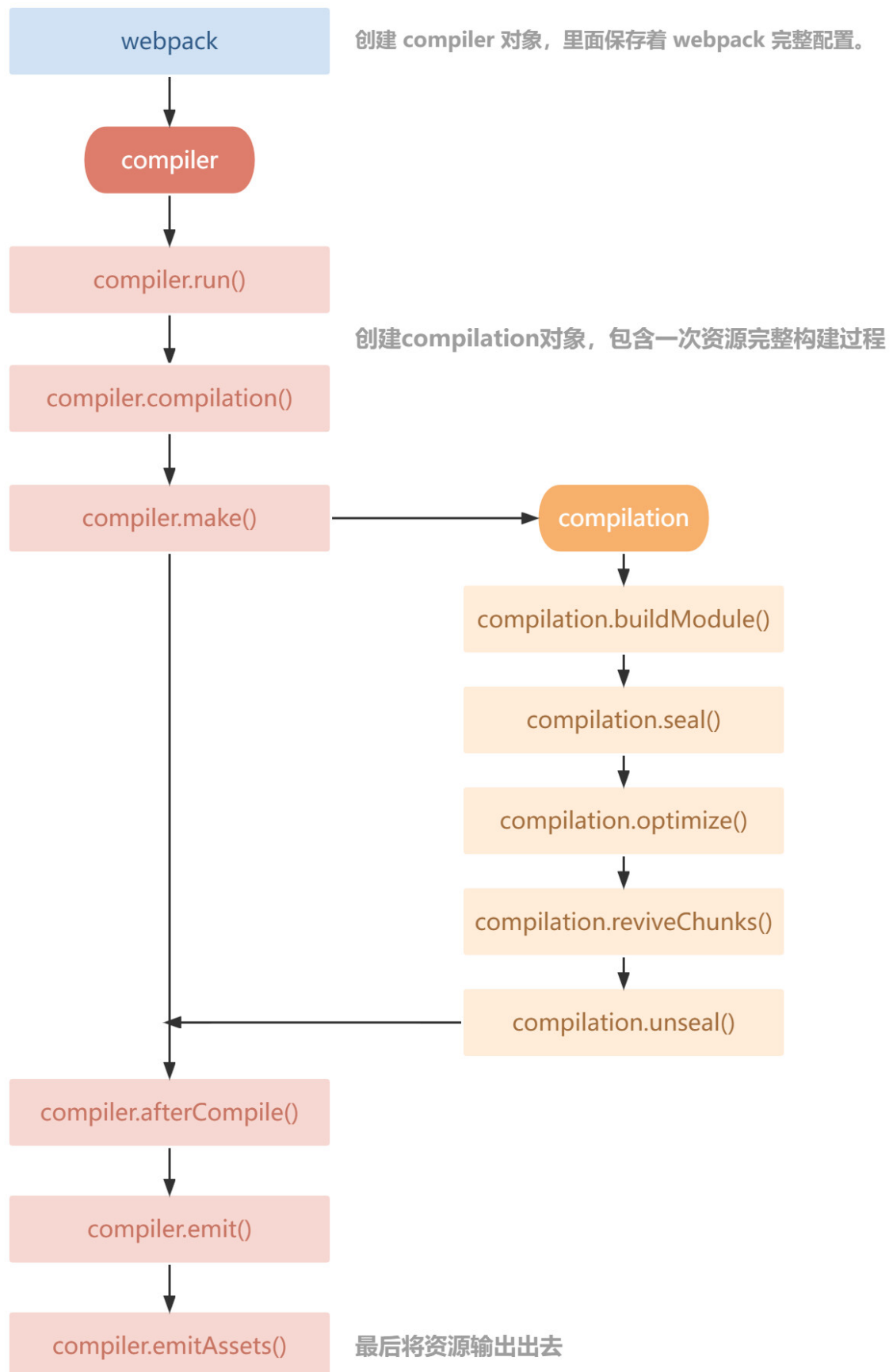
它有以下主要属性：

- `compilation.modules` 可以访问所有模块，打包的每一个文件都是一个模块。
- `compilation.chunks` chunk 即是多个 modules 组成而来的一个代码块。入口文件引入的资源组成一个 chunk，通过代码分割的模块又是另外的 chunk。
- `compilation.assets` 可以访问本次打包生成所有文件的结果。
- `compilation.hooks` 可以注册 tapable 的不同种类 Hook，用于在 compilation 编译模块阶段进行逻辑添加以及修改。

“

[compilation hooks 文档](#)

## - 生命周期简图



## • 开发一个插件

### – 最简单的插件

- plugins/test-plugin.js

```
1  class TestPlugin {
2    constructor() {
3      console.log("TestPlugin constructor()");
4    }
5    // 1. webpack读取配置时, new TestPlugin(), 会执行插件 constructor 方法
6    // 2. webpack创建 compiler 对象
7    // 3. 遍历所有插件, 调用插件的 apply 方法
8    apply(compiler) {
9      console.log("TestPlugin apply()");
10   }
11 }
12
13 module.exports = TestPlugin;
```

### – 注册 hook

```
1  class TestPlugin {
2    constructor() {
3      console.log("TestPlugin constructor()");
4    }
5    // 1. webpack读取配置时, new TestPlugin(), 会执行插件 constructor 方法
6    // 2. webpack创建 compiler 对象
7    // 3. 遍历所有插件, 调用插件的 apply 方法
8    apply(compiler) {
9      console.log("TestPlugin apply()");
10
11      // 从文档可知, compile hook 是 SyncHook, 也就是同步钩子, 只能用tap注册
12      compiler.hooks.compile.tap("TestPlugin", (compilationParams) => {
13        console.log("compiler.compile()");
14      });
15
16      // 从文档可知, make 是 AsyncParallelHook, 也就是异步并行钩子, 特点就是异步任务同时执行
17      // 可以使用 tap、tapAsync、tapPromise 注册。
18      // 如果使用tap注册的话, 进行异步操作是不会等待异步操作执行完成的。
19      compiler.hooks.make.tap("TestPlugin", (compilation) => {
20        setTimeout(() => {
21          console.log("compiler.make() 111");
22        }, 2000);
23      });
24
25      // 使用tapAsync、tapPromise注册, 进行异步操作会等异步操作做完再继续往下执行
26      compiler.hooks.make.tapAsync("TestPlugin", (compilation, callback) => {
```

```

27     setTimeout(() => {
28         console.log("compiler.make() 222");
29         // 必须调用
30         callback();
31     }, 1000);
32 });
33
34 compiler.hooks.make.tapPromise("TestPlugin", (compilation) => {
35     console.log("compiler.make() 333");
36     // 必须返回promise
37     return new Promise((resolve) => {
38         resolve();
39     });
40 });
41
42 // 从文档可知, emit 是 AsyncSeriesHook, 也就是异步串行钩子, 特点就是异步任务顺序执行
43 compiler.hooks.emit.tapAsync("TestPlugin", (compilation, callback) => {
44     setTimeout(() => {
45         console.log("compiler.emit() 111");
46         callback();
47     }, 3000);
48 });
49
50 compiler.hooks.emit.tapAsync("TestPlugin", (compilation, callback) => {
51     setTimeout(() => {
52         console.log("compiler.emit() 222");
53         callback();
54     }, 2000);
55 });
56
57 compiler.hooks.emit.tapAsync("TestPlugin", (compilation, callback) => {
58     setTimeout(() => {
59         console.log("compiler.emit() 333");
60         callback();
61     }, 1000);
62 });
63 }
64 }
65
66 module.exports = TestPlugin;

```

## - 启动调试

通过调试查看 `compiler` 和 `compilation` 对象数据情况。

### 1. package.json 配置指令

```

1  {
2      "name": "source",

```

```

3   "version": "1.0.0",
4   "scripts": {
5     "debug": "node --inspect-brk ./node_modules/webpack-cli/bin/cli.js"
6   },
7   "keywords": [],
8   "author": "xiongjian",
9   "license": "ISC",
10  "devDependencies": {
11    "@babel/core": "^7.17.10",
12    "@babel/preset-env": "^7.17.10",
13    "css-loader": "^6.7.1",
14    "loader-utils": "^3.2.0",
15    "webpack": "^5.72.0",
16    "webpack-cli": "^4.9.2"
17  }
18 }

```

## 2. 运行指令

```
1  npm run debug
```

此时控制台输出以下内容：

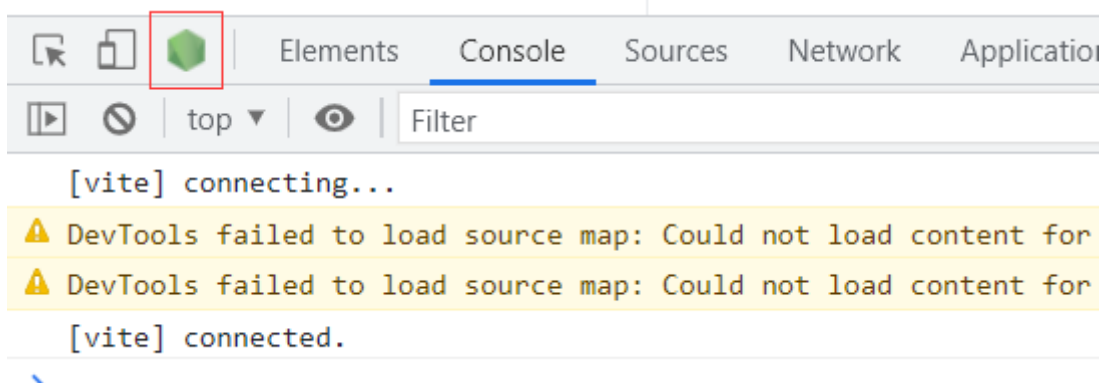
```

1  PS C:\Users\86176\Desktop\source> npm run debug
2
3  > source@1.0.0 debug
4  > node --inspect-brk ./node_modules/webpack-cli/bin/cli.js
5
6  Debugger listening on ws://127.0.0.1:9229/629ea097-7b52-4011-93a7-02f83c75c797
7  For help, see: https://nodejs.org/en/docs/inspecto

```

## 3. 打开 Chrome 浏览器，F12 打开浏览器调试控制台。

此时控制台会显示一个绿色的图标



## 4. 点击绿色的图标进入调试模式。

5. 在需要调试代码处用 `debugger` 打断点，代码就会停止运行，从而调试查看数据情况。

## • BannerWebpackPlugin

1. 作用：给打包输出文件添加注释。

2. 开发思路：

- 需要打包输出前添加注释：需要使用 `compiler.hooks.emit` 钩子，它是打包输出前触发。
- 如何获取打包输出的资源？ `compilation.assets` 可以获取所有即将输出的资源文件。

3. 实现：

```
1 // plugins/banner-webpack-plugin.js
2 class BannerWebpackPlugin {
3   constructor(options = {}) {
4     this.options = options;
5   }
6
7   apply(compiler) {
8     // 需要处理文件
9     const extensions = ["js", "css"];
10
11    // emit是异步串行钩子
12    compiler.hooks.emit.tapAsync("BannerWebpackPlugin", (compilation,
13    callback) => {
14      // compilation.assets包含所有即将输出的资源
15      // 通过过滤只保留需要处理的文件
16      const assetPaths = Object.keys(compilation.assets).filter((path) => {
17        const splitted = path.split(".");
18        return extensions.includes(splitted[splitted.length - 1]);
19      });
20
21      assetPaths.forEach((assetPath) => {
22        const asset = compilation.assets[assetPath];
23
24        const source = `/*
25 * Author: ${this.options.author}
26 */\n${asset.source()}`;
27
28        // 覆盖资源
29        compilation.assets[assetPath] = {
30          // 资源内容
31          source() {
32            return source;
33          },
34          // 资源大小
35          size() {
36            return source.length;
37          },
38        };
39      });
40    });
41  }
```

```

40     callback();
41   });
42 }
43 }
44
45 module.exports = BannerWebpackPlugin;

```

## • CleanWebpackPlugin

1. 作用：在 webpack 打包输出前将上次打包内容清空。

2. 开发思路：

- 如何在打包输出前执行？需要使用 `compiler.hooks.emit` 钩子，它是打包输出前触发。
- 如何清空上次打包内容？
  - 获取打包输出目录：通过 `compiler` 对象。
  - 通过文件操作清空内容：通过 `compiler.outputFileSystem` 操作文件。

3. 实现：

```

1  // plugins/clean-webpack-plugin.js
2  class CleanWebpackPlugin {
3    apply(compiler) {
4      // 获取操作文件的对象
5      const fs = compiler.outputFileSystem;
6      // emit是异步串行钩子
7      compiler.hooks.emit.tapAsync("CleanWebpackPlugin", (compilation,
8      callback) => {
9        // 获取输出文件目录
10       const outputPath = compiler.options.output.path;
11       // 删除目录所有文件
12       const err = this.removeFiles(fs, outputPath);
13       // 执行成功err为undefined, 执行失败err就是错误原因
14       callback(err);
15     });
16   }
17
18   removeFiles(fs, path) {
19     try {
20       // 读取当前目录下所有文件
21       const files = fs.readdirSync(path);
22
23       // 遍历文件, 删除
24       files.forEach((file) => {
25         // 获取文件完整路径
26         const filePath = `${path}/${file}`;
27         // 分析文件
28         const fileStat = fs.statSync(filePath);
29         // 判断是否是文件夹

```



```

29         if (fileStat.isDirectory()) {
30             // 是文件夹需要递归遍历删除下面所有文件
31             this.removeFiles(fs, filePath);
32         } else {
33             // 不是文件夹就是文件，直接删除
34             fs.unlinkSync(filePath);
35         }
36     });
37
38     // 最后删除当前目录
39     fs.rmdirSync(path);
40 } catch (e) {
41     // 将产生的错误返回出去
42     return e;
43 }
44 }
45 }
46
47 module.exports = CleanWebpackPlugin;

```

- **AnalyzeWebpackPlugin**

1. 作用：分析 webpack 打包资源大小，并输出分析文件。
2. 开发思路：
  - 在哪做？ `compiler.hooks.emit`，它是在打包输出前触发，我们需要分析资源大小同时添加上分析后的 md 文件。
3. 实现：

```

20         size() {
21             return source.length;
22         },
23     };
24 });
25 }
26 }
27
28 module.exports = AnalyzeWebpackPlugin;

```

## • InlineChunkWebpackPlugin

1. 作用: webpack 打包生成的 runtime 文件太小了, 额外发送请求性能不好, 所以需要将其内联到 js 中, 从而减少请求数量。

2. 开发思路:

- 我们需要借助 `html-webpack-plugin` 来实现
  - 在 `html-webpack-plugin` 输出 index.html 前将内联 runtime 注入进去
  - 删除多余的 runtime 文件
- 如何操作 `html-webpack-plugin`? [官方文档](#)

3. 实现:

```

1  // plugins/inline-chunk-webpack-plugin.js
2  const HtmlWebpackPlugin = require("safe-require")("html-webpack-plugin");
3
4  class InlineChunkWebpackPlugin {
5      constructor(tests) {
6          this.tests = tests;
7      }
8
9      apply(compiler) {
10         compiler.hooks.compilation.tap("InlineChunkWebpackPlugin", (compilation)
11         => {
12             const hooks = HtmlWebpackPlugin.getHooks(compilation);
13
14             hooks.alterAssetTagGroups.tap("InlineChunkWebpackPlugin", (assets) => {
15                 assets.headTags = this.getInlineTag(assets.headTags,
16                 compilation.assets);
17                 assets.bodyTags = this.getInlineTag(assets.bodyTags,
18                 compilation.assets);
19             });
20
21             hooks.afterEmit.tap("InlineChunkHtmlPlugin", () => {
22                 Object.keys(compilation.assets).forEach((assetName) => {
23                     if (this.tests.some((test) => assetName.match(test))) {
24                         delete compilation.assets[assetName];
25                     }
26                 });
27             });
28         });
29     }
30 }

```

```
23     });
24   });
25 });
26 }
27
28 getInlineTag(tags, assets) {
29   return tags.map((tag) => {
30     if (tag.tagName !== "script") return tag;
31
32     const scriptName = tag.attributes.src;
33
34     if (!this.tests.some((test) => scriptName.match(test))) return tag;
35
36     return { tagName: "script", innerHTML: assets[scriptName].source(),
closeTag: true };
37   });
38 }
39 }
40
41 module.exports = InlineChunkWebpackPlugin;
```