目录

[一、 基本介绍 1](#_Toc24454)

[二、 安装 1](#_Toc9670)

[三、 命令行编译&打包 2](#_Toc23135)

[npx webpack ./xxx/xxx.xx --mode=xxx 2](#_Toc27126)

[四、 基本配置 2](#_Toc10427)

[1. 5大核心概念 2](#_Toc701)

[2. 打包前清空输出目录 2](#_Toc21753)

[五、 处理样式资源 3](#_Toc5099)

[1. CSS 3](#_Toc29392)

[2. Lcss 3](#_Toc3301)

[3. Sass和Scss 4](#_Toc15500)

[4. Stylus 4](#_Toc3410)

[六、 处理图片资源 4](#_Toc12164)

[1. 优化（图片转Base64） 4](#_Toc12367)

[2. 修改图片的输出目录 4](#_Toc18678)

[七、 处理字体图标资源 5](#_Toc31320)

[指定字体文件的输出目录 5](#_Toc20191)

[八、 处理其他资源 5](#_Toc18232)

[九、 处理JS资源 5](#_Toc11411)

[1. babel：针对js兼容性处理 5](#_Toc16579)

[2. Eslint：针对代码格式检查 6](#_Toc26912)

[十、 处理HTML资源 8](#_Toc27693)

[十一、 开发服务器和自动化 8](#_Toc11717)

[1. 安装webpack-dev-server包 8](#_Toc2867)

[2. 在webpack.config.js中配置 8](#_Toc17656)

[3. 执行指令 9](#_Toc14195)

[十二、 开发/生产模式配置分离 9](#_Toc32468)

[1. 开发模式 9](#_Toc4519)

[2. 生产模式 9](#_Toc3573)

[3. 简化执行命令 9](#_Toc27557)

## 基本介绍

1. Webpack是一个静态资源打包工具
2. 将ES6或Less、Sass等预处理语法编译成浏览器能识别的JS、CSS语法。
3. 压缩代码、处理兼容性、提升性能
4. Webpack输出的文件通常被称为bundle

## 安装

npm i webpack webpack-cli -D

扩展：

npm：下载包

npx：将 node\_modules/.bin下的文件临时添加为环境变量，使命令可执行

i：install

-D：开发环境dev

## 命令行编译&打包

### npx webpack ./xxx/xxx.xx --mode=xxx

会创建dist目录，将编译的文件输出到此目录中

1. **./xxx/xxx.xx：**打包文件的入口
2. **--mode=development：**打包模式

development是开发模式

production是生产环境，生产环境会对代码进行压缩

## 基本配置

### 5大核心概念

1. entry（入口）

指定Webpack从哪个文件开始打包

1. output（输出）

指定Webpakc打包完的文件输出到哪里去，如何命名等

1. loader（加载器）

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

1. plugins（插件）

扩展webpack的功能

1. mode（模式）

主要有两种模式

开发模式：development

生产模式：production

**以上五项均配置在 webpack.config.js 文件中，配置完文件打包命令简化为**

**npx webpack**

### 打包前清空输出目录

output: {

        // 所有文件的输出路径

        // \_\_dirname nodejs的变量，代表当前文件的文件夹目录

        path: path.resolve(\_\_dirname, "dist"), // 绝对路径

        // 入口文件打包输出文件名称

        filename: 'js/main.js',

        // 打包前将path目录清空

        clean: true,

    },

## 处理样式资源

### CSS

1. 到官网找到需要的loader，如处理css需要 css-loader和style-loader
2. 安装需要的包
3. 在webpack.config.js 中配置loader

    // 加载器

    module: {

        rules: [

            // loder的配置

            {

                test: /\.css$/, // 只检测.css文件

                // use执行循序是从右到左（或从下到上）

                use: [

                    // 将js中的css通过创建style标签添加到html文件中

                    "style-loader",

                    // 将css资源编译成commonjs的模块到js中

                    "css-loader",

                ],

            },

        ]

    },

### Lcss

1. 安装 less
2. 安装 less-loader
3. 配置到webpack.config.js中

### Sass和Scss

1. 安装 sass
2. 安装 sass-loader
3. 配置到webpack.config.js中

### Stylus

1. 安装 stylus
2. 安装 stylus-loader
3. 配置到webpack.config.js中

## 处理图片资源

Webpack4 处理图片资源时通过 file-loader 和 url-loader 进行处理

Webpack5 将以上两个loader内置到了webpack中

### 优化（图片转Base64）

在module -> rules 中配置

{

                test: /\.(png|jpe?g|gif|webp|svg)$/,

                type: 'asset',

                parser: {

                    dataUrlCondition: {

                        // 小于10kb的图片转base64

                        // 优点：减少请求数量，缺点：体积变大

                        maxSize: 100 \* 1024

                    }

                }

            }

### 修改图片的输出目录

在上方配置中追加

generator: {

                    // 输出图片的名字 hash 哈希值|ext 后缀名|query url传递的参数

                    filename: 'static/images/[hash:10][ext][query]'

                }

## 处理字体图标资源

Webpack5 会自动处理字体文件。

### 指定字体文件的输出目录

{

                test: /\.(ttf|woff2?)$/,

                type: 'asset/resource',

                generator: {

                    // 字体文件的输出名称

                    filename: 'static/fonts/[hash:10][ext][query]'

                }

            },

## 处理其他资源

跟处理文字一样

{

                test: /\.(map3|map4|avi)$/,

                type: 'asset/resource', // resource会原封不动的输出文件

                generator: {

                    // 字体文件的输出名称

                    filename: 'static/media/[hash:10][ext][query]'

                }

            },

## 处理JS资源

Webpack5 默认对js的处理仅限于**模块化**语法，但浏览器不认识任何ES6语法，要处理其他ES6语法就要进行相应配置，如：

### babel：针对js兼容性处理

Babel：主要用于将ES6编写的代码转换成**向后兼容**的JavaScript语法

##### 配置文件（多种写法）

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

##### 配置项

presets 预设

一组babel插件，扩展babel功能

1. @babel/preset-env：一个智能预设，允许使用最新的JavaScript
2. @babel/preset-react：一个用来编译React jsx语法的预设
3. @babel/preset-typescript：一个用来编译TypeScript语法的预设

##### 在webpack.config.js中引入，并在rules项中配置：

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

{

                test: /\.m?js$/,

                // 排除 node\_modules 目录，不进行处理

                exclude: /(node\_modules)/,

                loader: 'babel-loader'

            },

### Eslint：针对代码格式检查

Eslint：可组装的JavaScript和JSX检查工具。

Eslint也是通过配置文件中，写在rulse里的规则对代码进行检查

##### 配置文件（多种写法）

1. .eslintrc.\*：新建文件，位于项目根目录
   1. .eslintrc
   2. .eslintrc.js
   3. .eslintrc.json
2. package.json中eslintConfig：不需要创建文件，在原有文件基础上写

##### 配置项

1. parserOptions 解析选项

module.exports = {

    parserOptions: {

        ecmaVersion: 6, // es 版本

        sourceType: "module", // es module

    }

};

1. rules具体规则

“off” 或 0：关闭规则

“warn” 或 1：开启规则，使用警告级别错误，不会导致程序退出

“error” 或 2：开启规则，使用错误级别，会导致程序退出

module.exports = {

    rules: {

        "no-var": 2, // 不能使用 var 定义变量

    }

};

1. extends继承

可以继承别人的规则（自己写的规则会覆盖继承来的规则），如：

Eslint：eslint:recommended

VueCli：plugin:vue/essential

ReactCli：react-app

module.exports = {

    // 继承 Eslint 规则

    extends: ["eslint:recommended"]

};

##### 在webpack.config.js中引入，并在plugins项中配置

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

new ESLintPlugin({

            // 检测文件的路径

            context: path.resolve(\_\_dirname, 'src')

        }),

## 处理HTML资源

1. 安装 html-webpack-plugin
2. 在 webpack -> plugins 中配置

new HtmlWebpackPlugin({

            /\*

            \*   模板：以public/index.html文件为模板，创建新的html文件

            \*   新的html文件特点：

            \*       1. 结构和原来一致

            \*       2. 自动引入打包输出的资源

            \*/

            template: path.resolve(\_\_dirname, 'public/index.html')

        }),

## 开发服务器和自动化

### 安装webpack-dev-server包

**npm i webpack-dev-server -D**

webpack-dev-server 可以实现自动编译，刷新

### 在webpack.config.js中配置

与五大核心平级

devServer: {

        host: 'localhost', // 服务器域名

        port: '3001', // 服务器端口

        open: true, // 是否自动打开浏览器

    }

### 执行指令

**npx webpack serve**

只用执行一次，之后修改代码不用再手动编译，会在内存中编译打包，不会在dist中输出。

## 开发/生产模式配置分离

在项目根目录创建config文件夹，文件夹中存放两个webpack配置文件

**webpack.dev.js：**开发模式配置文件

**webpack.prod.js：**生产模式配置文件

### 开发模式

* 1. 开发模式没有输出，所以output -> path 可以是undefined
  2. 开发模式不需要清空dist，所以不需要配置 output -> clean
  3. mode: “development”
  4. 命令：npx webpack serve --config ./config/webpack.dev.js

### 生产模式

* 1. 生产模式不需要配置devServer
  2. mode: ”production”
  3. 命令：npx webpack --config ./config/webpack.prod.js

### 简化执行命令

在package.json中配置

  "scripts": {

    "dev": "webpack serve --config ./config/webpack.dev.js",

    "build": "webpack --config ./config/webpack.prod.js"

  },

指令变成：npm run dev/build

## CSS处理（生产环境配置）

### 提取CSS成单文件

CSS文件被打包到js文件中，当js文件加载时，会创建一个style标签来生成样式

这种方式加载页面时，会出现闪屏现象，用户体验不好

通过link加载单独的CSS文件性能较好

1. 下载包

npm i mini-css-extract-plugin -D

1. 配置
2. 在webpack.prod.js中引入MiniCssExtractPlugin

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

1. 将所有 ”style-loader” 改为 MiniCssExtractPlugin.loader
2. 在plugins中配置

new MiniCssExtractPlugin({

            filename: 'static/css/main.css'

        }),

注：编译后index.htm会自动引入css和js文件，是应为HtmlWebpackPlugin 会自动引入打包生成的资源。

### CSS兼容性处理

1. 下载包

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

**postcss-loader依赖于postcss**

**postcss需要通过postcss-preset-env解决兼容性问题**

1. 配置
2. 在 package.json 中设置样式的兼容级别，常用配置如下：

"browserslist": [

    "last 2 version", // 只兼容最近两个版本

    "> 1%", // 只兼容 99% 的浏览器，忽略冷门的浏览器

    "not dead" // 忽略已经不存在的浏览器

  ]

1. 在webpack.prod.js中配置postcss

配置要写在 “css-loader” 之后 “预处理css-loader” 之前，如：

{

                test: /\.less$/,

                /\* 只能使用1个loader \*/

                // loader: 'xxx',

                /\* 可以使用多个loader \*/

                use: [

                    MiniCssExtractPlugin.loader,

                    'css-loader',

                    // postcss 解决css兼容性

                    {

                        loader: 'postcss-loader',

                        options: {

                            postcssOptions: {

                                plugins: [

                                    // 能解决大多数样式兼容性问题

                                    "postcss-preset-env",

                                ]

                            }

                        }

                    },

                    // 将 less 编译成 css 文件

                    'less-loader',

                ],

            },

### 封装样式loader函数

##### 封装

function getStyleLoader(pre) {

    return [

        // 将js中的css通过创建style标签添加到html文件中

        MiniCssExtractPlugin.loader,

        // 将css资源编译成commonjs的模块到js中

        "css-loader",

        {

            loader: 'postcss-loader',

            options: {

                postcssOptions: {

                    plugins: [

                        "postcss-preset-env", // 能解决大多数样式兼容性问题

                    ]

                }

            }

        },

        pre

    ].filter(Boolean);

    // 过滤 undefined、null、空字符串等 == false的变量

}

##### 调用

{

                // .s a或c ss 文件

                test: /\.s[ac]ss$/,

                // 'sass-loader' 参数会被添加进配置

                use: getStyleLoader('sass-loader'),

            },

### CSS压缩

1. 下载包

npm install css-minimizer-webpack-plugin --save-dev

## html和js压缩

开启生产模式，默认html和js都会进行压缩