webpack是一个前端资源构建工具, 一个静态模块打包器

在浏览器中并不能识别less,sass,scss等css预编译器语言

也无法识别import模块语法

补充 npm i xxx -D 开发依赖

所有的构建工具都是基于node运行的 所以模块化默认采用commonjs

## webpack五个核心概念

### Entry

入口 (Entry).webpack以哪个文件为入口开始打包,分析生成构件图

### Output

输出(Output)只是webpack打包后所有资源输出到哪里以及如何命名

### Loader

Loader让webpack能去处理那些非JavaScript文件(webpack自身只能理解JavaScript)

通常情况下一种文件只能由一个loader处理

当一个文件需要被多个loader处理的时候,一定要指定loader执行的先后顺序,比如js文件需要先执行eslint再执行babel

这个时候需要给eslint loader中添加一条属性: **enforce:”pre”** 表示优先执行(加上这个属性会让eslint这个loader放在所有的loader之前执行)

### 4.Plugins

插件(Plugins)可以用于执行更广的任务. 插件的范围包括从打包优化和压缩,一直到重新定义环境中的变量

### 5.Mode

模式(Mode)只是webpack使用相应的模式配置

development 开发模式

production 生产模式



### webpack 运行指令

webpack首先全局安装,然后在项目中安装

具体指令

npm i webpack webpack-cli -g

npm i webpack webpack-cli -D // 表示开发依赖

开发环境: webpack ./src/index.js -o ./build/built.js –mode=development

webpack 会以index为入口文件打包到build/built.js这个文件中 development表示开发打包

生产环境: webpack ./src/index.js -o ./build/built.js –mode= production

webpack 会以index为入口文件打包到build/built.js这个文件中 development表示生产打包(生产环境会压缩js代码)

生产环境和开发环境打包的作用是将ES6的模块化转化为浏览器能识别的代码

注意 webpack无法在不配置的情况下处理css和图片 只能处理js和json

开发环境中代码没有压缩,可以明确的找到这块代码打包自那个文件

生产环境中代码会压缩,正常人无法识别

### webpack打包样式资源(css)

webpack.config.js 目录与src为同一层级

作用: 指示webpack怎么干活,干那些活(当运行webpack的时候会加载里面的配置)

基本配置

var path = require("path")

module.exports = {

    // webpack配置

    // 入口起点

    entry: "./main/index.js",

    // 输出位置

    output: {

        // 输出文件名

        filename: "LidpVideo.js",

        // 输出路径

        // \_\_dirname 当前文件的绝对路径

        path: path.resolve(\_\_dirname,"dist")

    },

    // loader配置

    module: {

        rules: [

            // 详细loader配置

        ]

    },

    // 插件配置

    plugins: [

        // 详细插件配置

    ],

    // 开发模式

    mode: "development"

    // 生产模式

    // mode: "production"

}

#### 处理css文件

css的引入顺序与index.js中的引入顺序一致

处理css的loader名字是 css-loader

module: {

        // 注意 这里的loader都需要下载

        rules: [

            {

                test: /\.css$/,

                use: [

                    "style-loader",

                    "css-loader"

                ]

            },

            {

                test: /\.scss$/,

                use: [

                    "style-loader",

                    "css-loader",

                    "sass-loader"

                ]

            }

        ]

    }

#### 处理HTML文件

loader: 1. 下载 2. 使用(配置loader)

plugins: 1. 下载 2. 引入 3. 使用

处理HTML文件的plugins名字是 html-webpack-plugin

功能: 默认会创建一个空的html,引入打包输出的资源(js/css),输出

var path = require("path")

var HtmlWebpackPlugin =require("html-webpack-plugin")

module.exports = {

    // 入口

    entry: "./src/index.js",

    // 出口

    output: {

        // 输出文件名

        filename: "index.js",

        // 输出路径

        path: path.resolve(\_\_dirname,"dist")

    },

    // 配置loader

    module: {

        // 注意 这里的loader都需要下载

        rules: [

            {

                test: /\.css$/,

                use: [

                    "style-loader",

                    "css-loader"

                ]

            },

            {

                test: /\.scss$/,

                use: [

                    "style-loader",

                    "css-loader",

                    "sass-loader"

                ]

            }

        ]

    },

    plugins: [

        // 插件详细配置

        new HtmlWebpackPlugin({

            // 复制"./src/index.html"文件, 并且自动引入打包输出的资源

            template: "./src/index.html"

        })

    ],

    mode: "development"

}

#### 打包图片资源

图片处理loader名字是: url-loader(依赖file-loader)

需要下载两个包分别是: url-loader file-loader

var path = require("path")

var HtmlWebpackPlugin = require("html-webpack-plugin")

module.exports = {

  // 入口

  entry: "./src/index.js",

  // 出口

  output: {

    // 输出文件名

    filename: "index.js",

    // 输出路径

    path: path.resolve(\_\_dirname, "dist")

  },

  // 配置loader

  module: {

    // 注意 这里的loader都需要下载

    rules: [

      {

        test: /\.css$/,

        use: [

          "style-loader",

          "css-loader"

        ]

      },

      {

        test: /\.scss$/,

        use: [

          "style-loader",

          "css-loader",

          "sass-loader"

        ]

      },

        // 配置图片处理

        // 会有一个问题,可以处理css中的图片资源,但是无法处理html中的图片资源,所以在下方需要再引入一个loader

      {

        test: /.(jpg|png|gif)$/,

        // 此处需要下载 file-loader  url-loader

        use: [

            {

                loader:  "url-loader",

                options: {

                    limit: 8 \* 1024,

                    // 因为html-loader引入图片采用的是commonjs(require)  而 url-loader采用的是es6(import)

                    // 所以解析是会出现个问题  html中的<img src="[object Module]" alt="">

                    // 解决方法 : 关闭url-loader中的es6解析

                    esModule:false

                }

            }

        ]

      },

        // html-loader 处理html中的image文件,负责image文件的引入 从而能够被url-loader处理

        {

            test:/.html$/,

            use:"html-loader"

        }

    ]

  },

  plugins: [

    // 插件详细配置

    new HtmlWebpackPlugin({

      // 复制"./src/index.html"文件, 并且自动引入打包输出的资源

      template: "./src/index.html"

    })

  ],

  mode: "development"

}

#### 打包其他资源

rules:[

...,

{

// 也可以用test

// 排除

exclude:/\.(css|html|js)/,

loader:”file-loader”,

options:{

// 修改输出文件名

name:”[hash:10].[ext]”

}

}

]

#### devServer开发服务器(热编译)

自动编译,自动打开浏览器,自动刷新浏览器

启动devServer指令为 npx webpack-dev-server

需要下载 webpack-dev-server

npm i webpack-dev-server-D

var path = require("path")

var HtmlWebpackPlugin =require("html-webpack-plugin")

module.exports = {

    // 入口

    entry: "./src/index.js",

    // 出口

    output: {

        // 输出文件名

        filename: "index.js",

        // 输出路径

        path: path.resolve(\_\_dirname,"dist")

    },

    // 配置loader

    module: {

        // 注意 这里的loader都需要下载

        rules: [

            {

                test: /\.css$/,

                use: [

                    "style-loader",

                    "css-loader"

                ]

            },

            {

                test: /\.scss$/,

                use: [

                    "style-loader",

                    "css-loader",

                    "sass-loader"

                ]

            }

        ]

    },

    plugins: [

        // 插件详细配置

        new HtmlWebpackPlugin({

            // 复制"./src/index.html"文件, 并且自动引入打包输出的资源

            template: "./src/index.html"

        })

    ],

    mode: "development",

    // 开发服务器 热编译 自动刷新

    // 只会在内存中编译打包,不会有本地输出

    // 启动devServer指令为 npx webpack-dev-server

    // 需要下载 webpack-dev-server

    devServer: {

        // 端口号

        port:8000,

        // 项目构建后的路径

        contentBase:path.resolve(\_\_dirname,"dist"),

        // 启用g-zip压缩

        compress:true,

        // 自动打开浏览器

        open:true

    }

}

### css兼容处理

package.json

{

  "scripts": {

    "dev": "npx webpack-dev-server",

    "build": "webpack --mode development",

    "build2": "webpack --mode production"

  },

  "browserslist":{

    "development":[

      "last 1 chrome version",

      "last 1 firefox version",

      "last 1 safari version"

    ],

    "production":[

      ">0.01%",

      "not dead",

      "not op\_mini all"

    ]

  }

}

webpack.config.js

// 下载插件  mini-css-extract-plugin 本插件为导出css为单独文件

/\*

\* module:{

\*   rules:[

\*       {

\*           test:/\.css$/,

\*           use:[

\*               // 因为style-loader会将css-loader处理过后的代码插入到html中

\*               // 所以此处不适用style-loader

\*               // 使用 MiniCssExtractPlugin.loader 代替style-loader  作用为提取js中的css到单独文件中 并且会引入到html中

\*               MiniCssExtractPlugin.loader

\*               ,"css-loader"

\*           ]

\*       }

\*   ]

\* }

\*

\*

\* \*/

var {resolve} = require("path")

var HtmlWebpackPlugin = require("html-webpack-plugin")

var MiniCssExtractPlugin = require("mini-css-extract-plugin")

// process.env.NODE\_ENV = "development"

module.exports = {

    entry: "./src/index.js",

    output: {

        filename: "js/build.js",

        path: resolve(\_\_dirname,"dist")

    },

    module: {

        rules: [

            {

                test: /\.css$/,

                use:[

                    MiniCssExtractPlugin.loader,

                    "css-loader",

                    /\*\*

                     \* css兼容性处理需要用到 postcss

                     \* 在websocket中使用的话需要用到 postcss-loader

                     \* 还需要一个插件 postcss-preset-env  这个插件可以帮助postcss识别使用环境 从而精确到兼容ie几

                     \*

                     \*

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

                        // 可以去GitHub上面找  browserslist 搜详细配置

                      "browserslist":{

                            // 开发环境

                            // 需要设置nodejs的环境变量

                            // 在本文件的上方

                            // process.env.NODE\_ENV = "development"

                          "development":[

                              "last 1 chrome version",  // 最后一个版本的谷歌浏览器

                              "last 1 firefox version", // 最后一个版本的火狐浏览器

                              "last 1 safari version"   // 最后一个版本的苹果浏览器

                          ],

                          // 生产环境  默认看生产环境

                          "production":[

                              ">0.2%", // 用户>0.2%的浏览器

                              "not dead",  // 不要已经去世的浏览器

                              "not op\_mini all"  // 不要op\_mini浏览器全部版本

                          ]

                      }

                     \*/

                    {

                        // 修改loader的配置

                        loader: "postcss-loader",

                        options: {

                            ident:"postcss",  // 固定值

                            plugins:()=>[

                                // postcss的插件

                                require("postcss-preset-env")()

                            ]

                        }

                    },

                ]

            }

        ]

    },

    plugins: [

        new HtmlWebpackPlugin({

            template: "./src/index.html"

        }),

        new MiniCssExtractPlugin({

            filename: "css/dist.css"

        })

    ],

    mode: "production",

    devServer: {

        port:3000,

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

        compress:true,

        // 自动打开浏览器

        open:true

    }

}

### 压缩css

插件名 optimize-css-assets-webpack-plugin

plugins: [

        new HtmlWebpackPlugin({

            template: "./src/index.html"

        }),

        new MiniCssExtractPlugin({

            filename: "css/dist.css"

        }),

        //压缩css

        new OptimizeCssassetsWebpackPlugin()

    ],

### js语法检查 ESLint

语法检查:可以检查常见的语法错误,统一代码风格

eslint-loader eslint

设置检查规则:

在package.json中eslintConfig中设置

推荐使用airbnb规则

eslint eslint-config-airbnb-base eslint-plugin-import

在loader中写

{

test: /\.js$/,

// 这里不检查node\_modules下的文件

exclude:/node\_modules/

loader:”eslint-loader”,

options:{

// 自动修复eslint的错误

fix:true

}

}

package.json中添加

“eslintConfig”:{

“extends”:”airbnb-base”

}

如果希望eslint的规则对下一行失效

添加如下注释则会忽略

// eslint-disable-next-line

### js兼容性处理

兼容处理需要使用到 babel-loader @babel/preset-env @babel/core @babel/polyfill core-js

babel-loader js兼容处理的loader

@babel/preset-env 基本兼容

@babel/core 上面两个包的依赖项

@babel/polyfill 做全部语法兼容用

core-js 做按需加载用

webpack.config.js

{

test:/\.js$/,

exclude:/node\_modules/,

loader:”babel-loader”,

options:{

// 预设: 只是babel做怎样的兼容性处理 下方是基本兼容性处理

// 只能做普通的兼容性处理 无法处理如promise等等

// 全部兼容性处理 使用 @babel/polyfill

// 这个包会将所有兼容处理引入,体积太大

// 所以我们需要做按需加载 corejs包

presets: [

// 注意此处的中括号

["@babel/preset-env",

{

// 按需加载

useBuiltIns: "usage",

// 指定corejs版本

corejs: {

version: 3

},

// 指定浏览器版本

targets: {

chrome: "60",

firefox: "60",

ie: "9",

safari: "10",

edge: "17"

}

}]

]

}

### js和html压缩

webpack.config.js

js压缩: 将mode改为production // 生产环境会自动压缩js代码

plugins: [

        new HtmlWebpackPlugin({

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

            minify: {

                // 移除空格

                collapseWhitespace: true,

                // 移除注释

                removeComments: true,

            },

        }),

    ]

### 性能优化

开发环境性能优化

优化webpack的打包构建速度

优化代码调试功能

生产环境性能优化

优化webpack打包构建速度

优化代码运行的性能