1. 初始化项目

npm init -y

1. 安装webpack和webpack-cli

npm i webpack --save-dev

npm i webpack-cli --save-dev

1. 修改package.json文件

"scripts": {

"build": "webpack"//执行运行项目

}

1. 创建入口文件(webpack4 more配置entry（入口src/index.js 和output（出口） dist/main.js)

/src/index.js

1. 修改package.json文件

"dev": "webpack --mode development",

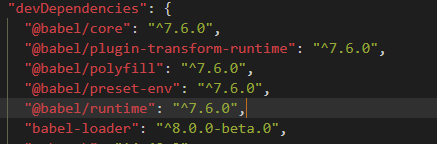
"build": "webpack --mode production"

1. Webpack4配置总结



1. 项目配置
   1. js的处理：转换 ES6 代码，解决浏览器兼容问题(配置babel-loader)

安装：



npm install babel-loader@8.0.0-beta.0 @babel/core @babel/preset-env

npm install --save @babel/polyfill

npm install --save-dev @babel/plugin-transform-runtime

npm install --save @babel/runtime

添加babelloader配置文件/.babelrc

{

  "presets": [

      "@babel/preset-env"

  ],

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

}

添加项目配置文件/webpack.config.js

const path = require('path');

module.exports = {

  entry: [

    "@babel/polyfill",

    path.join(\_\_dirname, './src/index.js')

  ],

  module: {

    rules: [

      {

        test: /\.js$/,

        exclude: /node\_modules/,

        loader: 'babel-loader',

        options: {}

      }

    ]

  }

}

* 1. css的处理：编译css，自动添加前缀，抽取css到独立文件(编译css，自动添加前缀，抽取css到独立文件)

npm i mini-css-extract-plugin css-loader --save-dev(postcss-loader(抽取css到独立文件)

npm i style-loader postcss-loader --save-dev (用于添加浏览器前缀)

npm install --save-dev autoprefixer

创建postcss配置文件/postcss.config.js

module.exports = {

  plugins: {

      autoprefixer: {}

  }

}

修改webpack.config.js文件

const path = require('path');

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

module.exports = (env, argv) => {

  const devMode = argv.mode !== 'production'

  return {

    module: {

      entry: [

        "@babel/polyfill",

        path.join(\_\_dirname, './src/index.js')

      ],

      rules: [

        {

          test: /\.js$/,

          exclude: /node\_modules/,

          loader: 'babel-loader',

          options: {}

        },

        {

          test: /\.css$/,

          use: [

            devMode ? 'style-loader' : MiniCssExtractPlugin.loader,

            'css-loader',

            'postcss-loader'

          ]

        }

      ],

    },

    plugins: [

      new MiniCssExtractPlugin({

        filename: "[name].css",

        chunkFilename: "[id].css"

      })

    ]

  }

}

* 1. html的处理：复制并压缩html文件（复制并压缩html文件 html-webpack-plugin）

npm i html-webpack-plugin html-loader --save-dev

Webpack.congfig.json中配置

const HtmlWebPackPlugin = require("html-webpack-plugin");

module.exports = {

module: {

rules: [

// ...,

{

test: /\.html$/,

use: [{

loader: "html-loader",

options: {

minimize: true

}

}]

}

]

},

plugins: [

new HtmlWebPackPlugin({

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

filename: "./index.html"

})

]

};

* 1. dist的清理：打包前清理源目录文件（clean-webpack-plugin）

npm install clean-webpack-plugin --save-dev

配置：

const CleanWebpackPlugin = require('clean-webpack-plugin');

module.exports = {

plugins: [

new CleanWebpackPlugin(['dist']),

]

};

* 1. assets的处理：静态资源处理（file-loader）

npm install file-loader --save-dev

配置

* 1. server的启用：development 模式下启动服务器并实时刷新（webpack-dev-server）

npm i webpack-dev-server --save-dev

package.json中配置

"scripts": {

"start": "webpack-dev-server --mode development --open",

"build": "webpack --mode production"

}

webpack环境变量配置

创建环境变量文件：/config/index.js

环境变量配置：

const config = {

  production: {

    ENV\_CONFIG: '"prod"',

    BASE\_API: '"http://localhost:4200"'

  },

  development: {

    ENV\_CONFIG: '"dev"',

    BASE\_API: '"http://localhost:4200"'

  }

};

module.exports = config;

Webpack.config.js中引入

const configEnv = require('./config')

使用webpack.DefinePlugin配置全局变量

导入webpack

const webpack = require('webpack')

在plugins数组中添加

new webpack.DefinePlugin({

        'process.env': configEnv[argv.mode]

      })

使用：console.log(process.env.ENV\_CONFIG)

1. Elslint

安装

Npm install eslint eslint-plugin-prettier eslint-config-prettier babel-eslint prettier eslint-plugin-html --save-dev

react校验

npm install eslint-plugin-react@latest --save-dev

npm install babel-eslint --save-dev

新建eslint配置文件.eslintrc.js

module.exports = {

  "root": true,

  "extends": [

    'prettier',

    'eslint:recommended',

   ],

  "plugins": ['prettier'],

  "rules": {

    "prettier/prettier": "error",

    "no-console": process.env.NODE\_ENV === "production" ? "error" : "off",

    'no-debugger': process.env.NODE\_ENV === 'production' ? 'error' : 'off',

  },

  "parserOptions" : {

    "ecmaVersion": 6,

"sourceType": "module"

  },

  "env": { // 指定脚本的运行环境。每种环境都有一组特定的预定义全局变量

    "browser": true,

    "node": true,

    "es6": true,

    "commonjs": true

  },

  "parser": "babel-eslint", //指定一个不同的解析器, 默认esprima

}

新建prettier 配置文件.prettierrc

{

  "printWidth": 100,

  "singleQuote": true,

  "semi": false

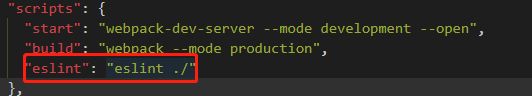
}

1. [eslint 配合 git （husky）](https://www.cnblogs.com/zzd0916/p/10254893.html)

安装

npm install husky --save-dev

在package.json scripts下面加入esliint的命令验证 eslint的规则



接下来在package.json 下面加入 husky 的配置项

"husky": {

    "hooks": {

      "pre-commit": "echo 'husky' && npm run eslint"

    }

  },

1. 安装lint-staged(确保没有错误被提交)

安装

npm install --save-dev lint-staged@next

在package.json中配置

"husky": {

    "hooks": {

      "pre-commit": "lint-staged"

    }

  },

  "lint-staged": {

    "\*.{js,jsx,ts,tsx}": [

      "pretty-quick --staged",

      "eslint --fix",

      "git add"

    ]

  },

快速提交

npm install --save-dev prettier pretty-quick

多环境配置cross-env

安装

npm i --save-dev cross-env

使用：根据生产环境格式提交代码

"husky": {

    "hooks": {

      "pre-commit": "cross-env NODE\_ENV=prod lint-staged"

    }

  },

1. Vscode中 按照eslint 格式保存

新建/.vscode/setting.json

配置

{

  // 保存时自动格式化

  "eslint.autoFixOnSave": true,

  // 用来配置作用的文件类型

  "eslint.validate": [

    "javascript",

    "javascriptreact",

    {

      "language": "html",

      "autoFix": true

    },

    {

      "language": "jsx",

      "autoFix": true

    }

  ]

}

1. 错误

lint-staged的问题删除重新安装依赖，也可能husky也存在问题，也删除重新安装