# webpack 快速入门

1. model开发模式



1. 本地安装
   1. npm install webpack -D
   2. 如果你使用webpack 4+版本，你还需要安装 webpack-cli

npm install webpack-cli -D

* 1. 添加npm的script脚本（package.json）

|  |
| --- |
| "scripts":{  "start":"webpack --config webpack.config.js"  } |

1. 初始化package.json文件

npm init -y

# 处理CSS模块

# webpack的模块和处理sass

# webpack的sass添加c3前缀和sourcemap的处理

1. sourcemap

|  |
| --- |
| **module:{**  **rules:[**  **test:/\.(sc|c|sa)ss$/,**  **use:[**  **'style-loader',{**  **loader:'css-loader',**  **options:{**  **sourceMap:true**  **}**  **},**  **{**  **loader:'sass-loader',**  **options:{**  **sourceMap:true**  **}**  **}**  **]**  **]**  **}** |

1. postcss



|  |
| --- |
| **module:{**  **rules:[**  **test:/\.(sc|c|sa)ss$/,**  **use:[**  **'style-loader',{**  **loader:'css-loader',**  **options:{**  **sourceMap:true**  **}**  **},**  **{**  **loader:'postcss-loader',**  **options:{**  **ident:'postcss',**  **sourceMap:true,**  **plugins:(loader)=>{**  **require('autoprefixer')({browsers:['>0.15% in CN']})**  **}**  **}**  **},**  **{**  **loader:'sass-loader',**  **options:{**  **sourceMap:true**  **}**  **}**  **]**  **]**  **}** |

options中的ident标识微标识符

# css提取成单独的文件

1. 首先css的处理都把mode设置为production。抽取了样式就不能再用style-loader注入到html中了
2. webpack4开始使用：**mini-css-extract-plugin** 插件，
   1. 1-3的版本可以用：**extract-text-webpack-plugin**

|  |
| --- |
| **npm instal1--save-dev mini-css-extract-plugin**  **let** prodConfig = {  **mode**: **'production'**,  **output**: {  **filename**: **'main.[hash].js'**,  **path**: path.*resolve*(\_\_dirname, **'dist'**)  },  **module**: {  **rules**: [  {  **test**: /\.(sc|c|sa)ss$/,  **use**: [  MiniCssExtractPlugin.**loader**, {  **loader**: **'css-loader'**,  **options**: {  **sourceMap**: **true** }  }, {  **loader**: **'postcss-loader'**,  **options**: {  **ident**: **'postcss'**,  **sourceMap**: **true**,  plugins: (loader) => [require(**'autoprefixer'**)({**browsers**: [**'> 0.15% in CN'**]})]  }  }, {  **loader**: **'sass-loader'**,  **options**: {  **sourceMap**: **true** }  }  ]  }  ]  },  **plugins**: [  **new** MiniCssExtractPlugin({  **filename**: **'[name][hash].css'**, *// 设置最终输出的文件名* **chunkFilename**: **'[id][hash].css'** })  ],  **optimization**: {  **minimizer**: [  **new** OptimizeCSSAssetsPlugin({}), *// 压缩CSS插件* **new** UglifyJsPlugin({  **cache**: **true**, **parallel**: **true**, **sourceMap**: **true** *// set to true if you want JS source maps* })  ]  } }; |

# css和js压缩插件的使用

1. 压缩css插件

**optimize-css-assets-webpack-plugin**

1. 压缩JS

**uglifyjs-webpack-plugin，此插件需要一个前提就是：mode:’production’**

|  |
| --- |
| **npm -D uglifyjs-webpack-plugin**  **npm -D optimize-css-assets-webpack-plugin**  **首先require引入**  **optimization: {**  **minimizer: [**  **new OptimizeCSSAssetsPlugin({}), // 压缩CSS插件**  **new UglifyJsPlugin({**  **cache: true, parallel: true, sourceMap: true // set to true if you want JS source maps**  **})**  **]**  **}** |

# webpack解决文件名带哈希值的问题

1. HtmlWebpackPlugin插件
2. **npm instal1--save-dev html-webpack-plugin**

|  |
| --- |
| **首先require引入**  **plugins: [**  **new HtmlWebpackPlugin({**  **title: 'AICODER 全栈线下实习', // 默认值：Webpack App**  **filename: 'index.html', // 默认值： 'index.html'**  **template: path.resolve(\_\_dirname, 'src/main.html'),**  **minify: {**  **collapseWhitespace: true,**  **removeComments: true, // 是否移除注释**  **removeAttributeQuotes: true // 移除属性的引号**  **}**  **}),**  **]** |

* **title: 生成的HTML模板的title，如果模板中有设置title的名字，则会忽略这里的设置**
* **filename: 生成的模板文件的名字**
* **template: 模板来源文件**
* **inject: 引入模块的注入位置；取值有true/false/body/head**
* **favicon: 指定页面图标；**

# webpack清理目录插件clean-webpack-plugin应用

1. clean-webpack-plugin
2. npm instal1 clean-webpack-plugin --save-dev

|  |
| --- |
| **plugins: [**  **new HtmlWebpackPlugin({**  **title: 'AICODER 全栈线下实习', // 默认值：Webpack App**  **filename: 'index.html', // 默认值： 'index.html'**  **template: path.resolve(\_\_dirname, 'src/main.html'),**  **minify: {**  **collapseWhitespace: true,**  **removeComments: true, // 是否移除注释**  **removeAttributeQuotes: true // 移除属性的引号**  **}**  **}),**  **new CleanWebpackPlugin(['dist'])**  **]** |

# 图片处理及优化

1. file-loader插件
2. npm install--save-dev file-loader

|  |
| --- |
| **require引入**  **module: {**  **rules: [**  **{**  **test: /\.(png|svg|jpg|gif|jpeg)$/,**  **include: [path.resolve(\_\_dirname, 'src/')],**  **use: [‘filt-loader’}**  **]**  **},** |

1. 对图片进行压缩和优化

**npm install image-webpack-loader--save-dev**

|  |
| --- |
| **require引入**  **module: {**  **rules: [**  **{**  **test: /\.(png|svg|jpg|gif|jpeg)$/,**  **include: [path.resolve(\_\_dirname, 'src/')],**  **use: [**  **‘file-loader’,**  **{**  **loader: 'image-webpack-loader',**  **options: {**  **mozjpeg: {**  **progressive: true,**  **quality: 65**  **},**  **optipng: {**  **enabled: false**  **},**  **pngquant: {**  **quality: '65-90',**  **speed: 4**  **},**  **gifsicle: {**  **interlaced: false**  **},**  **webp: {**  **quality: 75**  **}**  **}**  **}**  **]**  **}**  **]**  **},** |

# 图片base64优化和字体处理

1. url-loader插件
2. npm install--save-dev url-loader

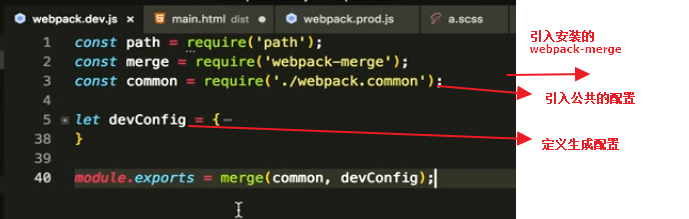
|  |
| --- |
| 首先require引入  **require引入**  **module: {**  **rules: [**  **{**  **test: /\.(png|svg|jpg|gif|jpeg)$/,**  **include: [path.resolve(\_\_dirname, 'src/')],**  **use: [**  **{**  **loader: 'url-loader', //根据图片的大小，优化为base64**  **options: {**  **limit: 10000**  **}**  **}, {**  **loader: 'image-webpack-loader',//先进行图片优化**  **options: {**  **mozjpeg: {**  **progressive: true,**  **quality: 65**  **},**  **optipng: {**  **enabled: false**  **},**  **pngquant: {**  **quality: '65-90',**  **speed: 4**  **},**  **gifsicle: {**  **interlaced: false**  **},**  **webp: {**  **quality: 75**  **}**  **}**  **}**  **]**  **}**  **]**  **},** |

1. 字体的处理

|  |
| --- |
| **module:{**  **rules:[**  **{**  **test:/\.(woff|woff2|eot|ttf|otf)$/,**  **use:[**  **'file-loader'**  **]**  **}**  **]**  **}** |

# 配置合并和提前公共配置

1. webpack-merge插件
2. **npm install--save-dev webpack-merge**



1. **package.js中的script指定配置文件**
   1. **"build": "npx webpack --config webpack.dev.js",**

# 开发环境：启动监控自动编译和启用js的sourcemap

1. 使用inline-source-map，有助于解释说明JS原始出错的位置。不要用于生产环境



1. **devtool:inline-source-map,**

# 开发阶段：启动热更新和代理配置

1. package.js中的script指定配置文件 自动编译不刷新
   1. **"watch": "npx webpack --watch --config webpack.dev.js",**
2. **webpack-dev-server 内存编译，自动编辑刷新**
   1. **npm install--save-dev webpack-dev-server**



|  |
| --- |
| **devServer: {**  **clientLogLevel: 'warning', // 可能的值有 none, error, warning 或者 info（默认值)**  **hot: true, // 启用 webpack 的模块热替换特性, 这个需要配合： webpack.HotModuleReplacementPlugin插件**  **contentBase: path.join(\_\_dirname, "dist"), // 告诉服务器从哪里提供内容， 默认情况下，将使用当前工作目录作为提供内容的目录**  **compress: true, // 一切服务都启用gzip 压缩**  **host: '0.0.0.0', // 指定使用一个 host。默认是 localhost。如果你希望服务器外部可访问 0.0.0.0**  **port: 8080, // 端口**  **open: true, // 是否打开浏览器**  **overlay: { // 出现错误或者警告的时候，是否覆盖页面线上错误消息。**  **warnings: true,**  **errors: true**  **},**  **publicPath: '/', // 此路径下的打包文件可在浏览器中访问。**  **proxy: { // 设置代理**  **"/api": { // 访问api开头的请求，会跳转到 下面的target配置**  **target: "http://192.168.0.102:8080",**  **pathRewrite: {**  **"^/api": "/mockjsdata/5/api"**  **}**  **// /api/getuser => http://192.168.0.102:8080//mockjsdata/5/api/getuser**  **}**  **},**  **quiet: true, // necessary for FriendlyErrorsPlugin. 启用 quiet 后，除了初始启动信息之外的任何内容都不会被打印到控制台。这也意味着来自 webpack 的错误或警告在控制台不可见。**  **watchOptions: { // 监视文件相关的控制选项**  **poll: true, // webpack 使用文件系统(file system)获取文件改动的通知。在某些情况下，不会正常工作。例如，当使用 Network File System (NFS) 时。Vagrant 也有很多问题。在这些情况下，请使用轮询. poll: true。当然 poll也可以设置成毫秒数，比如： poll: 1000**  **ignored: /node\_modules/, // 忽略监控的文件夹，正则**  **aggregateTimeout: 300 // 默认值，当第一个文件更改，会在重新构建前增加延迟**  **}**  **},** |





* 1. **new webpack.NamedModulesPlugin(), // 更容易查看(patch)的依赖**
  2. **new webpack.HotModuleReplacementPlugin() // 替换插件**
  3. **启动webserver: webpack-dev-server –open**
  4. **优化命令：**
     1. **"start": "npx webpack-dev-server --config webpack.dev.js",**

# 代理服务器设置和bable转换及优化

1. 代理服务器设置

|  |
| --- |
| **devServer: {**  **clientLogLevel: 'warning',**  **hot: true, // 启用 webpack 的模块热替换特性, 这个需要配合： webpack.HotModuleReplacementPlugin插件**  **contentBase: path.join(\_\_dirname, "dist"),**  **compress: true, // 一切服务都启用gzip 压缩**  **host: '0.0.0.0', // 指定使用一个 host。默认是 localhost。**  **port: 58080, // 端口**  **open: true, // 是否打开浏览器**  **overlay: { // 出现错误或者警告的时候，是否覆盖页面线上错误消息。**  **warnings: true,**  **errors: true**  **},**  **publicPath: '/', // 此路径下的打包文件可在浏览器中访问。**  **proxy: { // 设置代理**  **"/api": { // 访问api开头的请求，会跳转到 下面的target配置**  **target: "http://192.168.0.102:8080",**  **pathRewrite: {**  **"^/api": "/mockjsdata/5/api"**  **}**  **// /api/getuser => http://192.168.0.102:8080//mockjsdata/5/api/getuser**  **}**  **},**  **quiet: true, //**  **watchOptions: { // 监视文件相关的控制选项**  **poll: true,**  **ignored: /node\_modules/, // 忽略监控的文件夹，正则**  **aggregateTimeout: 300 // 默认值，当第一个文件更改，会在重新构建前增加延迟**  **}**  **},** |

1. JS启用Babel转码
   1. **npm i -D babel-loader babel-core babel-preset-env**
   2. **webpack配置文件**

|  |
| --- |
| **module: {**  **rules: [**  **{**  **test: /\.js$/,**  **exclude: /(node\_modules)/, // 加快编译速度，不包含node\_modules文件夹内容**  **use: {**  **loader: 'babel-loader',**  **options: {**  **cacheDirectory: true //加快编译**  **}**  **}**  **}** |

* 1. **在项目的根目录下，添加babel的配置文件：.babelrc**

|  |
| --- |
| **{**  **"presets": ["env"],**  **}** |

* 1. **引入babel runtime作为独立模块，避免重复引入**

|  |
| --- |
| **安装：**  **npm install babel-plugin-transform-runtime --save-dev**  **npm install babel-runtime- -save //必须在生产环境依赖**  **wepack.comfig.js中设置**  **rules: [**  **{**  **//’transform-runtime’ 插件告诉Babel要引入runtime来代替注入**  **test: /\.js$/,**  **exclude: /(node\_modules|bower\_components)/, // 加快编译速度，不包含node\_modules文件夹内容**  **use: {**  **loader: 'babel-loader',**  **options: {**  **cacheDirectory: true**  **}**  **}**  **}**  **]**  **修改.babelrc文件**  **{**  **"presets": ["env"],**  **"plugins": [**  **["transform-runtime", {**  **"helpers": true,**  **"polyfill": true,**  **"regenerator": true,**  **"moduleName": "babel-runtime"**  **}]**  **]**  **}** |

# eslint校验配置

1. 安装

|  |
| --- |
| **npm install eslint--save-dev**  **npm install eslint-loader--save-dev**  **#以下是用到的额外的需要安装的eslint的解释器、校验规则等**  **npm i -D babel-eslint standard** |

1. eslint可以直接放到webpack配置中，也可以在项目根目录的**.eslintrc**中

|  |
| --- |
| **webpack中**  **rules: [**  **{**  **//’transform-runtime’ 插件告诉Babel要引入runtime来代替注入**  **test: /\.js$/,**  **exclude: /(node\_modules|bower\_components)/, // 加快编译速度，不包含node\_modules文件夹内容**  **use: [**  **{},**  **……**  **{**  **loader: eslint-loader',**  **options: {**  **fix:true //不符合规则时自动修复**  **}**  **}**  **]**  **]** |

按照webpack中rule的执行顺序是从下到上的，所以eslint应该放置在use数组的最后

1. **.eslintrc.js文件**

|  |
| --- |
| **// .eslintrc.js**  **// https://eslint.org/docs/user-guide/configuring**  **module.exports = {**  **root: true,**  **parserOptions: {**  **parser: 'babel-eslint'**  **},**  **env: {**  **browser: true**  **},**  **extends: [**  **// https://github.com/standard/standard/blob/master/docs/RULES-en.md**  **'standard'**  **],**  **globals: {**  **NODE\_ENV: false**  **},**  **rules: {**  **// allow async-await**  **'generator-star-spacing': 'off',**  **// allow debugger during development**  **'no-debugger': process.env.NODE\_ENV === 'production' ? 'error' : 'off',**  **// 添加，分号必须**  **semi: ['error', 'always'],**  **'no-unexpected-multiline': 'off',**  **'space-before-function-paren': ['error', 'never'],**  **// 'quotes': ["error", "double", { "avoidEscape": true }]**  **quotes: [**  **'error',**  **'single',**  **{**  **avoidEscape: true**  **}**  **]**  **}**  **};** |

1. **再添加一个.eslintignore文件**

|  |
| --- |
| **/dist/**  **/node\_modules/**  **/\*.js** |

1. **打开eslint.autoFIxOnSave：true,出现错误自动修复**

# 模块解析后缀和别名配置详解

1. 创建import或require的别名，确保模块的引入变得简单
2. webpack.config.js根目录下

|  |
| --- |
| **resolve: {**  **alias: { // 配置别名**  **'@': path.resolve(\_\_dirname, 'src/')**  **},**  **extensions: [".js", ".vue", ".json"] // 默认值: [".js",".json"] 模块名字可以省略的后缀名**  **},** |

**示例：import{d,e,f} from ‘@/b’**

# 模块的外部依赖配置

1. esternals配置选项提供了从输出的bundle中排除依赖的方法
2. 例如从CDN引入jQuery，而不是把它打包
3. 首先引入一个CDN上的jQuery。
4. 在webpack的根目录中

|  |
| --- |
| **externals: { // 把一个模块做成外部依赖，不会打包到 js文件中。**  **jquery: 'jQuery',**  **lodash: '\_'**  **},** |

不同的配置方式

|  |
| --- |
| **externals: {**  **react: 'react',**  **},**  **或**  **externals: {**  **lodash: {**  **commonjs:'lodash',**  **amd:'lodash',**  **root:'\_'//指向全局变量**  **}**  **},**  **或**  **externals: {**  **subtract: {**  **root:["math","subtract"]//相当于window.math.substract**  **}**  **},** |

1. 使用方式

|  |
| --- |
| **import $ from ‘jquery’;//不会打包到bundle中** |

# 打包分析报表插件与优化总结

1. **开发环境下：webpack-bundle-analyzer插件**
2. **npm install--save-dev webpack-bundle-analyzer**

|  |
| --- |
| **const BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin;**    **module.exports = {**  **plugins: [**  **new BundleAnalyzerPlugin()**  **]**  **}** |