目录

[基础 2](#_Toc534454892)

[概念 2](#_Toc534454893)

[命名规则 2](#_Toc534454894)

[chunk 2](#_Toc534454895)

[entry 3](#_Toc534454896)

[output 3](#_Toc534454897)

[module 3](#_Toc534454898)

[loader规则 3](#_Toc534454899)

[rule 3](#_Toc534454900)

[css 4](#_Toc534454901)

[postcss-loader 4](#_Toc534454902)

[功能 4](#_Toc534454903)

[cdn加速 4](#_Toc534454904)

[按需加载 4](#_Toc534454905)

[热更新 5](#_Toc534454906)

[命令行启动方式 5](#_Toc534454907)

[node控制 5](#_Toc534454908)

[拆包 5](#_Toc534454909)

[CommonsChunkPlugin 5](#_Toc534454910)

[SplitChunksPlugin 6](#_Toc534454911)

[分包 7](#_Toc534454912)

[分离css文件 7](#_Toc534454913)

[雪碧图 7](#_Toc534454914)

[loader 8](#_Toc534454915)

[plugins 8](#_Toc534454916)

[CommonsChunkPlugin 8](#_Toc534454917)

[webpack4 9](#_Toc534454918)

[默认配置 9](#_Toc534454919)

[mode 9](#_Toc534454920)

[更新 9](#_Toc534454921)

[webpack-dev-server 9](#_Toc534454922)

[调用 9](#_Toc534454923)

[webpack-dev-server命令行 9](#_Toc534454924)

[nodeapi 9](#_Toc534454925)

[编译出来的代码分析 10](#_Toc534454926)

process.env.NODE\_ENV,这里的process.env.NODE\_ENV不是node中的环境变量,而是webpack.DefinePlugin中定义的全局变量,允许你根据不同的环境执行不同的代码.

# 基础

最好用\_\_dirname或者process.cwd控制目录，因为有些配置的路径是相对工作目录，比如插件，entry等。有些又是相对配置文件所在的目录，比如自己引入的文件。

## 概念

### 命名规则

.[ext] ： 指代源文件后缀。

### chunk

webpack当中配置的入口文件（entry）是chunk，可以理解为entry chunk

入口文件以及它的依赖文件通过code split（代码分割）出来的也是chunk，可以理解为children chunk

通过CommonsChunkPlugin创建出来的文件也是chunk，可以理解为commons chunk

## entry

数组可以用来引入一些前置文件：比如polyfill。

相对目录是当前工作目录(调用命令的目录)，即使配置文件是在子目录里。（所以一般用\_\_dirname确定文件所在目录）

## output

必须有filename

path是绝对路径

chunkFilename和filename同时设置时，结果为 chunkFilename，filename无效。

devtoolModuleFilenameTemplate用于调试时资源映射。默认只是js文件名（可能重名），可以把路径放进字符串里。

devtoolModuleFilenameTemplate: info =>

path.resolve(info.absoluteResourcePath).replace(/\\/g, '/'),

设置publicPath后，所有的引用都会设置的路径，包括js，css，图片等。

## module

### loader规则

！？

### rule

Rule.test 是 Rule.resource.test 的简写。

use必须跟一个数组。字符串的情况可以直接用loader。也可以use:['style-loader']。

test: RegExp | string | function | [(recursive)] | object 字符串必须是一个绝对路径。

loader有两种缩写

正常样式：{use: [{loader:'xxx'}]}

彻底缩写：{loader: 'xxx'}。

一般缩写：{use:['xxx']}

## css

理论上顺序应该是：less-loader css-loader style-loader。但实际不按这个顺序写好像也没问题。

webpack默认是不会单独抽离css文件的。会在js脚本里通过style标签引入(需要借助style-loader)

### postcss-loader

## 功能

## cdn加速

设置publicPath。一次性设置了所有在html，js里直接引入的资源（包括js，css，图片。）

抽离css文件。

被css资源文件引入的资源地址

为引用文件加上hash值。（js，css，图片）

## 按需加载

使用import()即可，一般配合router，webpack本身支持打包成多个单独的文件，可以通过router引用。

老版本的babel不支持import()，需要引入bael-plugin-syntax-dynamic-import，现在不需要了。

原理:通过script标签去异步加载

## 热更新

live reload: 保存后自动刷新浏览器。

hmr: 无刷新更新代码，一般仅限css修改可以。

以chunk为单位进行更新。

### 命令行启动方式

只需要在命令行里加上--watch --hot即可，甚至不需要再配置文件里配置。

### node控制

重点：

要在入口时运行webpack的两个服务。

entry: ['webpack-dev-server/client?http://localhost:9000/',

'webpack/hot/dev-server', path.resolve(\_\_dirname, '../js/index.js')],

同时配置watch，hot。

**new** *webpack*.*HotModuleReplacementPlugin*()

同时引入上面这个插件。

## 拆包

### CommonsChunkPlugin

webpack.optimize.CommonsChunkPlugin

chunks

name

webpack4已经被移除了

### SplitChunksPlugin

webpack.optimize.SplitChunksPlugin

w4默认配置：

module.exports = {

//...

optimization: {

splitChunks: {

chunks: 'async',

minSize: 30000,

maxSize: 0,

minChunks: 1,

maxAsyncRequests: 5,

maxInitialRequests: 3,

automaticNameDelimiter: '~',

name: true,

cacheGroups: {

vendors: {

test: /[\\/]node\_modules[\\/]/,

priority: -10

},

default: {

minChunks: 2,

priority: -20,

reuseExistingChunk: true

}

}

}

}

};

新的代码块被共享或者来自node\_modules文件夹

chunks: "initial" | "async" | "all" | ((*chunk*: *compilation*.*Chunk*) => *boolean*);

默认 entry 的 chunk 不会被拆分。chunks='all’代表着我们拆分chunk时也包含entry。

initial: 入口之间共享的代码

name: *boolean* | *string* | ((...*args*: *any*[]) => *any*);

给一个字符串的话，拆分的chunk会合并成对应字符串。

缓存组：

一个缓存组可以理解成一个输出文件

默认将所有来源于node\_modules的模块分配到叫做'venders'的缓存组,所有引用超过两次的模块分配到'default'缓存组.

缓存组属性

name

true值将会自动根据切割之前的代码块和缓存组键值(key)自动分配命名,否则就需要传入一个String或者function。而且定义name以后，所有的模块会被放进一个文件里。

## 分包

npm run build -- type=a

在build文件里，通过process.argv获取到type，打不同的渠道包。注意argv是一个数组，每个值是每一项的全部，比如"type=a"，需要自己split。

## 分离css文件

## 雪碧图

webpack-spritesmith

src.cwd指定输入目录

target.image： 输出图片地址

target.css: 输出css文件,里面有每一个icon的css样式

customTemplates.custom\_format: 每一个css样式的结构，值为一个函数，返回css输出的css文件。比如" .ico { background-image: url(I) } \n .ico-hover { background-image: url(I) }"

sprite:{name,width,height,offset\_x,offset\_y}

只需要在dev模式下使用，生成的文件在prod打包时一样可以用。

每次运行都会生成新的图片吗？还是有变化时才会重新生成？

每次运行时都会重新生成。所以run start的时间比较长

# loader

输入输出可以是二进制文件，也可以是文本文件（默认）。

# plugins

一般需要装： 生成html的，注入变量的，拆分chunk的

生产环境： 压缩，分离css，

原理：输入一个options选项，返回一个对象，里面需要有一个apply方法。webpack会调这个方法，所以plugins最好用class的方式写。apply会传入compiler对象。在apply函数里通过compiler.plugin(eventname, callback)监听对应的事件。

## CommonsChunkPlugin

filename

'static/common.[chunkhash:6].js',

这里的static是相对于打包出来的目录比如dist的子目录。

# webpack4

## 默认配置

entry: 'src/index.js'

output: dist/main.js

## mode

## 更新

SplitChunkPlugin,取代CommonsChunkPlugin

# webpack-dev-server

调用这个命令时，也会查找webpack.config.js命令。

## 调用

### webpack-dev-server命令行

执行命令行时，会读webpack配置文件的devServer配置。

3.x增加了一个log参数。

底层通过express实现。

### nodeapi

如果你通过 Node.js API 来使用 dev-server， devServer 中的选项将被忽略。将选项作为第二个参数传入： new WebpackDevServer(compiler, {...})。

const compiler = webpack(webpackConfig)

const serverConfig = {

contentBase: config.distPath,

compress: true,

port: 9000,

hot: true,

}

const server = new webpackDevServer(compiler, serverConfig);

server.listen(9000);

# 编译出来的代码分析

用到了Symbol是什么鬼？