webpack

webpack的配置实际上是一个nodejs的脚本。webpack通过这个配置对象来读取相关的一些配置。

# 加载器

vue-loader 默认使用 css-loader 和 Vue模板编译器来编译vue文件的样式和模板。

遇到 <img src url() @import 会被解析为 require引用

如 <img src=”assets/images.png”>,会被编译为

createElement(‘img’, {attrs:{src: require(‘assets/images.png’)}})

file-loader:

能指定从什么地方拷贝资源文件，以及发布后放在哪个目录下，并可以使用版本哈希码来重新发布后的文件来实现增量更新和更好的缓存策略。

url-loader:

能根据你指定的文件大小阈值，判断一个文件是否转换为内联的base-64吗，如果该文件小于阈值，还是使用file-loader来降级处理。（有效减少小文件的http请求）

# 具体配置

## entry

1. ‘./src/main.js’

②多入口，多出口

{

main:’./src/main.js’,

index:’.src/index.js’

}

③多入口，单出口

['./src/main.js','./src/index.js']

## output

output:{

path 指定打包后文件的输出目录

publicPath:’/’ 指定资源文件引用的目录，这个目录会被输出的被引用，如子chunks,图片，字体等。

filename

libraryTarget:’umd’ // AMD 与 commonJS都可以使用（一般为了打包到npm包，可以requre使用）

}

## module

### 作用：

决定了 如何处理项目中不同类型的模块

#### rules

[

{

resource:{

},

issuer:{

}

}

]

匹配规则：  
 resource规则适用于 请求源文件的绝对路径

issuer 规则适用于 声明依赖请求的 源文件的绝对路径

规则条件会根据这两个条件尝试匹配（issuer用的比较少，一般用于某种类型的文件中只能引用某些类型的文件）

### 匹配文件

{ test: ... } 匹配特定条件

{ include: ... } 匹配特定路径

{ exclude: ... } 排除特定路径

{ and: [...] }必须匹配数组中所有条件

{ or: [...] } 匹配数组中任意一个条件

{ not: [...] } 排除匹配数组中所有条件

条件：

字符串：必须以提供的字符串开始，所以是字符串的话，这里我们需要提供绝对路径 】

正则表达式：调用正则的 test 方法来判断匹配 函数：(path) => boolean，返回 true 表示匹配

数组：至少包含一个条件的数组

对象：匹配所有属性值的条件

不解析：

options[‘noParse’] :

### loader应用顺序

一个文件同时匹配多个匹配规则，引用顺序：

****前置 -> 行内 -> 普通 -> 后置****

****行内：****require('json-loader!./file.json')

前置： options[enforce ] : ‘pre’

后置： options[‘enforce’] : ‘post’

模块代码转换的工作由loader来处理：

style-loader

会将css-loader解析的结果转变为js代码，运行时动态插入style标签。

css-loader

解析css代码，处理css中的依赖，例如@import 和 url()等引用外部文件的声明

vue-loader

vue-style-loader

功能与style-loader相似，默认在vue-loader中使用了，一般不用再次配置。

## plugin:

html-webpack-plugin

关联html

传递一个 模板html

extract-text-webpack-plugin 4.x版本？

可以将单独的css文件分离出来。

Uglifyjs-Webpack-Plugin

压缩---默认mode：production配置了

[DefinePlugin](https://link.juejin.im/?target=https://webpack.js.org/plugins/define-plugin/" \t "https://juejin.im/book/5a6abad5518825733c144469/section/_blank)

webpack.[DefinePlugin](https://link.juejin.im/?target=https://webpack.js.org/plugins/define-plugin/" \t "https://juejin.im/book/5a6abad5518825733c144469/section/_blank)

定义环境变量

ProvidePlugin

webpack.ProvidePlugin

引用某些模块作为应用运行时的变量，而不用每次都require 或 Import

IgnorePlugin

webpack.IgnorePlugin

忽略特定模块中特定文件的引入。

copy-webpack-plugin

复制文件

## resolve

### 作用：

配置模块如何解析。如：import ‘loadash’，这个配置会影响webpack查找lodash的方式。

### options

alias

别名，确保引入更加简单。

extensions

扩展名查询规则

mainFields

修改包查找路径

mainFiles

没有package.json时，默认使用当前目录下index.js。这个也可以配置

## externals

创建一个避免将文件打包到bundle中，而使用script访问。

## devServer

webpack-dev-server

webpack-dev-server --mode development

应当设置 默认development

options:

public:

用于指定静态服务的域名：

如果使用Nginx反向代理，使用该配置指定Nginx配置使用的服务域名。

publicPath:

指定构建好的静态文件在浏览器用什么路径去访问：

默认是 ‘/’

和express中的publicPath是一个东西

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

静态文件位置（这里指定是不经过webpack构建的的静态文件）

publicPath的优先级高于contentBase

proxy

配置代理路径

before

在静态资源中间件处理之前，可以用于拦截部分请求，返回特定内容，

或者实现简单的数据mock

after

在静态资源中间件处理之后，可以用于打印日志或者其它额外工作。

babel

# webpack相关组件

## 开发环境配置组件

**更关注重新加载或热更新替换，具有source map 和 localhost server**

webpack-dev-server

提供一个可热更新的开发服务器

webpack-dev-middleware

是一个容器，它可以把webpack处理后的文件传递给一个服务器，webpack-dev-server内部已经使用了它。也可以作为单独的包使用

const express = require('express');

const webpack = require('webpack');

const webpackDevMiddleware = require('webpack-dev-middleware');

const app = express();

const config = require('./webpack.config.js');

const compiler = webpack(config);

// Tell express to use the webpack-dev-middleware and use the webpack.config.js// configuration file as a base.

app.use(webpackDevMiddleware(compiler, {

publicPath: config.output.publicPath

}));

// Serve the files on port 3000.

app.listen(3000, function () {

console.log('Example app listening on port 3000!\n');

});

## 生产环境配置组件

**更关注更小的bundle**

**webpack-merge**

提供合并配置的功能

# hash与 chunkhash指纹

webpack编译输出文件的配置过程中，可以为文件加入hash指纹，webpack提供两种hash指纹

hash

hash of compilation(编译对象的hash)

compilation对象代表某个版本的资源对应的编译进程。当使用webpack的development中间件时，每次监控到文件有变动，就会创建一个compilation对象。

这个对象包括了当前模块资源，待编译文件，有改动的文件，监听依赖的所有信息。

也就是所任何文件的修改都会导致这个值修改。

chunkhash

chunk的hash（也就是模块的hash）

这个是根据具体模块文件的改动计算所得的hash值。

这个模块引用的其他文件的改动也会被计算

# 关联HTML

## html-webpack-plugin

<https://github.com/jantimon/html-webpack-plugin#configuration>

option {

template: ‘’ // 模板

inject: true | ‘head’ | ‘body’, // bundle 在html中位置

favicon:'./favicon.ico' // favicion 路径 （会复制图标文件到打包文件夹）

}

## Copy-Webpack-Plugin

有时会在模板html中引入 一些公告公共的 js ，css

生产模式下：

new CopyWebpackPlugin([

{

from: path.resolve(\_\_dirname, 'static'),

to: path.resolve(\_\_dirname, 'dist/static'),

ignore: ['.\*']

}

])

这时就需要将这些不经过打包的外部文件使用这个插件放到打包后相应的文件夹中

# 构建CSS

由于css-loader 和 style-loader只是解决了将解析css，处理css文件中依赖 ； 并且打包成js的形式，运行动态添加<style></style>

而没有直接转换为独立的css文件

extract-text-webpack-plugin@next

目前稳定版支持webpack3，使用@版支持webpack4

webpack设计核心概念

模块打包器

webpack 处理应用程序是，会递归构建一个依赖关系图。

四个核心概念：

入口 输出 loader 插件

入口：根上下文或app第一个启动文件

出口：将所有资源（assets）归拢后，output描述webpack如何处理归拢在一起的代码。

loader:webpack把所有文件（css, html ,scss, jpg,etc...）都作为模块处理。然而webpack只理解JavaScript。 loader在文件被添加到依赖图中时，将其转为模块。

webpack命令参数

--progress : 显示构建进度

webpack-dev-server 命令参数（部分配置参数可以直接在命令行写入，有些参数则只能通过命令行写入）

--allowed-hosts :所有域都可以访问

--bonjour

--color

配置参数

**configuration Languages**

支持多种语法编写配置文件。

需要安装一些依赖

支持 TypeScript

CoffeScript

babel

jsx

**configuration Types**

exporting a function

为了区分development 和 production build，至少需要两个参数。

module.exports = {+module.exports = function(env, argv) {

return { devtool: env.production ? 'source-maps' : 'eval',

plugins: [

new webpack.optimize.UglifyJsPlugin({+ compress: argv['optimize-minimize'] // only if -p or --optimize-minimize were passed

})

] };

};

exporting a promise

module.exports = () => {

return new Promise((resolve, reject) => {

setTimeout(() => {

resolve({

entry: './app.js',

/\* ... \*/

})

}, 5000)

})}

exporting multiple configurations

module.exports = [{

output: {

filename: './dist-amd.js',

libraryTarget: 'amd'

},

entry: './app.js',}, {

output: {

filename: './dist-commonjs.js',

libraryTarget: 'commonjs'

},

entry: './app.js',}]

**entry [strring | object {key:string,key:(function:()=> string ]**

默认使用当前文件夹，让你的配置独立于当前工作文件夹

**动态入口 【】**

entry: () => new Promise((resolve) => resolve(['./demo', './demo2']))

**context 【string】**

默认使用当前文件夹，让你的配置独立于当前工作文件夹，上下文时入口文件所在的目录绝对路径。

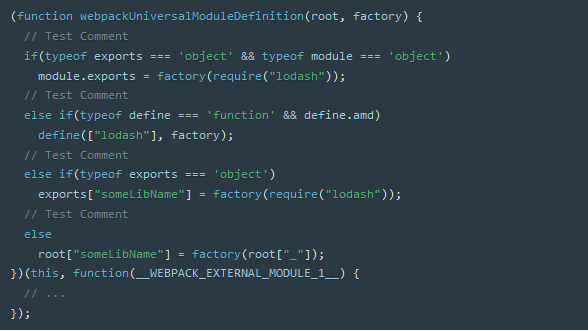
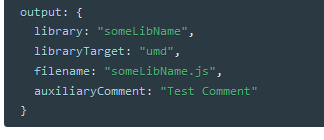
**name 【string】**

默认使用当前文件夹，让你的配置独立于当前工作文件夹

**output**

auxiliaryComment [string | object]

与output.library和output。libraryTarget一起使用，允许用户向导出容器中插入注释



## 

chunkFilename [string]

chunkLoadTimeout [string]

chunk请求到期时间（默认：120000）

crossOriginLoading [string | boolean]

只用于 target为web（编译的模块用于浏览器环境）。使用了通过script标签的jsonp来加载需要加载的chunk

值： false（默认），‘anonymous’(不带凭据的启用跨域加载) , ‘use-credentials’ (带凭据的启动跨域加载)

**devServer**

webpack-dev-server的配置参数

after [function]

在服务器内部的其他所有中间件之后提供执行自定义中间件的能力

allowedHosts 【array】 cli

允许访问Dev server的百名单

before [function]

在其他中间件之前提供执行自定义中间件的能力。

before(app){

app.get('/some/path', function(req, res) {

res.json({ custom: 'response' });

});}

bonjour [boolean] cli

广播服务

clientLogLevel 【string】

控制日志输出

‘none’

‘error’

‘warning’

‘info’

color CLI only

compress 【boolean】 cli

对于任何服务是都开启gzip

contentBase [boolean | string | array] cli

静态资源路径，支持多出口。

contentBase: [path.join(\_\_dirname, "public"), path.join(\_\_dirname, "assets")]

disableHostCheck [boolean] cli

设置这个，绕过主机检测。不推荐（容易受到DNS重新绑定攻击）。

filename 【string】

减少编译操作在懒惰模式中。

在懒惰模式，默认：所有的请求结果都在一个新的编译。

有filename属性, 仅仅被请求的文件被编译。

header 【object】

向所有响应添加头部

historyApiFallback [boolean | object] cli

替换404响应

historyApiFallback: {

rewrites: [

{ from: /^\/$/, to: '/views/landing.html' },

{ from: /^\/subpage/, to: '/views/subpage.html' },

{ from: /./, to: '/views/404.html' }

]}

host 【string】 cli

如果你想外部可以使用，设置host 0.0.0.0

hot [boolean] cli

热更新

hotOnly [boolena] cli

以防build失败的情况下使得热更新可用。

https [boolean | object] cli

使得服务支持http2的HTTPS

index [string]

设置index file

info 【boolean】 cli only

输出cli信息

inline 【boolean】 cli

是否开启 iframe mode

inline mode:一个脚本在包中处理重载

iframe mode : 状态栏里展示build信息

lazy [boolean] cli

服务仅仅在请求时再编译。意味着webpack不再监控任何文件的改变

watchOptins 无效--在使用lazy mode时

noInfo [boolena]

webpack 启动和每次保存时的 包信息将会被隐藏，

而错误和警告会被展示

open 【boolean】cli

服务器将会打开浏览器

openPage [string]

在打开浏览器时，指定导航到的页面

overlay [boolean | object]

默认情况下不可用，打开这个，可以在浏览器中展示编译错误和警告

pfx [string] cli

当使用cli时，指向数字证书（ssl）的地址。并且这个数字证书应该是字节流的格式（bytestream）

pfxPassphrase [string] cli

SSL pfx证书的密码

port 【number】 cli

端口

proxy 【object】

代理一些url是有用的, 当你拥有一个单独的后端开发服务器，并且你想在同一域下发送api请求

dev-server 将会使用一个强大的 http-proxy-middleware 包

当后端没有合法的证书，可以设置绕过他

通过写一个回调，绕过代理。

支持多路径

progress 【boolean】 cli only

输出进度

public 【string】 cli

当使用 inline模式时，并且你正在使用代理服务，服务器不总是知道需要链接到哪。

publicPath 【string】

1. 确保publicPath 都以/作为开始和结束

在这个的路径下，在浏览器的打包后的文件是可用的（可以链接到文件的）

使用全部的url，这是必要的在热更新替换时。（http:localhost:8080）

quiet [boolean] cli

除了开启信息，其他信息不被写出

setup [function]

不赞成使用

socket 【string】 cli

The Unix socket to listen to (instead of a host).

staticOptions [object]

静态资源高级配置（contentBase为string时可以使用）

配置参考express

stats [string | object]

精确控制console的内容

stdin [boolean] cli only

当标准输入结束时，关闭服务

useLocalIp [boolean] cli

让浏览器可以用本地的ip打开

watchContentBase [boolean] cli

监控contentBase中的文件，文件改变重载。

默认不使用

watchOptions [object]

webpack是使用文件系统来通知文件改变的，当使用网络文件系统（NFS） 构建工具（vagrant）有在这里有很多问题。可以使用这个参数。

# webpack代码分割

<http://www.css88.com/doc/webpack2/guides/code-splitting-require/>

## **require.ensure()**

webpack 在编译时，会静态地解析代码中的 require.ensure()，同时将模块添加到一个分开的 chunk 当中。这个新的 chunk 会被 webpack 通过 jsonp 来按需加载。

# webpack封装工具

cooking