

webpack

安装

打包

webpack.config.js

loader

图片处理

ES6 --> ES5

vue

el 与 template 的关系

webpack plugin

前端模块化方案，webpack 可以将 AMD CMD CommonJS ES6 打包成通用浏览器能够运行的代码
中文文档 <https://www.webpackjs.com/>

安装

webpack 依赖 node 环境

```
npm install webpack@3.6.0 -g
```

打包好的文件在 dist 中。 `dist --> distribution(发布)`

打包

 pack.zip

```
1 // 没有配置的打包，不推荐
2 webpack ./src/main.js ./dist/bundle.js
```

配置 webpack.config.js

补充

package.json 这个是怎么来的？

```
npm init -y
```

 自动会生成这个文件

webpack.config.js

`npm init -y` 生成 package.json 文件

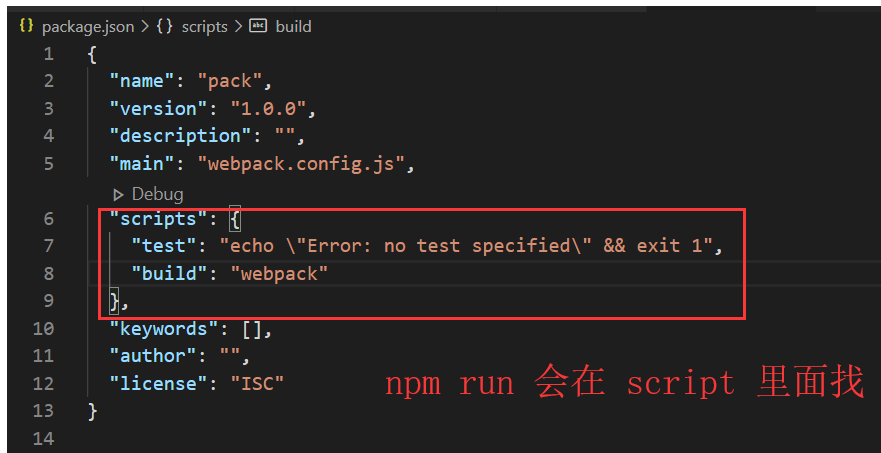
在 script 里面添加脚本

执行 `npm run build`

这里的 webpack 安装开发时依赖就好了

`npm install webpack@3.6.0 --save-dev`

使用 scripts 会优先到本地找 webpack



```
{
  "name": "pack",
  "version": "1.0.0",
  "description": "",
  "main": "webpack.config.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "build": "webpack"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

npm run 会在 script 里面找

```
1 const path = require('path')
2
3 module.exports = {
4   entry: './src/main.js',
5   output: {
6     path: path.resolve(__dirname, 'dist'),
7     filename: 'bundle.js',
8
9     // 任何 url 相关都会加上 publicPath 前缀
10    publicPath: 'dist/',
11  },
12
13  // css
14  module: {
15    rules: [
16      // css
17      {
```

```

18         // 匹配 css 文件的正则表达式
19         test: /\.css$/,
20         // 从右向左加载的
21         use: ['style-loader', 'css-loader']
22     },
23
24     // less
25     {
26         test: /\.less$/,
27         use: [
28             {
29                 loader: 'style-loader',
30             },
31             {
32                 loader: 'css-loader',
33             },
34             {
35                 loader: 'less-loader',
36             },
37         ],
38     },
39
40     // 图片处理
41     {
42         test: /\.(jpg|png|gif|jpeg)$/,
43         use: [
44             {
45                 loader: 'url-loader',
46                 options: {
47                     // 限制大小
48                     // 图片小于 limti 会被编译成 base64 格式
49                     // 图片大于 limit 时会使用 fileloader 加载
50                     limit: 8000,
51
52                     // 原本是32位 hash 改为8位, ext --> extenti
53
54                     // img/ 文件夹
55                     name: 'img/[name].[hash:8].[ext]'
56                 }
57             }
58         ]
59     }
60 }

```

```

57         ],
58     },
59
60     // es6 --> es5
61     {
62         test: /\.js$/,
63         // 排除
64         exclude: /(node_modules|bower_components)/,
65         use: [
66             {
67                 loader: 'babel-loader',
68                 options: {
69                     presets: ['es2015'],
70                 }
71             }
72         ]
73     },
74 ]
75 }
76 }

```

loader

根据需要安装各种 loader 这里用打包 css 为例

```
npm install css-loader --save-dev
```

```
npm install style-loader --save-dev
```

图片处理

```
npm install url-loader --save-dev
```

ES6 --> ES5

ES6 转 ES5 需要借助 babel

```
npm install --save-dev babel-loader@7 babel-core babel-preset-es2015
```

vue

使用vue 的三种方式：

- 1.直接下载
- 2.CDN
- 3.npm install

```
npm install vue --save
```

vue 有两个版本

- runtime-only 代码中不可以有任何 template
- runtime-compiler 代码中可以有 template,因为有 compiler 可以用于编译 template

```
✖ [Vue warn]: You are using the runtime-only build of Vue where the template compiler is not available. Either pre-compile the templates into render functions, or use the compiler-included build.  
(found in <Root>)
```

解决办法，在 webpack 中添加如下配置

```
1 module.export = {  
2   entry: './src/main.js'  
3   output: {  
4     },  
5  
6   module: {  
7     rules: []  
8   },  
9  
10  // 指定 vue 版本  
11  resolve: {  
12    alias: {  
13      'vue$': 'vue/dist/vue.esm.js'  
14    }  
15  }  
16 }
```

el 与 template 的关系

```

1 import Vue from 'vue'
2
3 new Vue({
4   el: '#app',
5   template: `
6     <div>
7       <h2>{{ message }}</h2>
8       <button></button>
9     </div>
10  `,
11  data: {
12    message: 'Hello Webpack',
13  }
14 })

```

template 最终在编译的时候会去替换 el 选中的部分
但是写成 template 挂载类上也很难看。。。如何解决呢？

第一次优化，拆成组件的形式

```

1 import Vue from 'vue'
2
3 const App = {
4   template: `
5     <div>
6       <h2>{{ message }}</h2>
7       <button>按钮</button>
8     </div>
9   `,
10  data() {
11    return {
12      message: 'Hello Webpack',
13    }
14  }
15 }
16
17 new Vue({

```

```

18     el: '#app',
19     template: `<App/>`,
20     components: {
21         App,
22     }
23 })

```

第二次优化

创建 vue/app.js 将组件单独的拆分成一个文件

```

1 export default {
2     template: `
3         <div>
4             <h2>{{ message }}</h2>
5             <button>按钮</button>
6         </div>
7     `,
8     data() {
9         return {
10             message: 'Hello Webpack',
11         }
12     }
13 }

```

```

1 import Vue from 'vue'
2 import App from './vue/app'
3
4 new Vue({
5     el: '#app',
6     template: `<App/>`,
7     components: {
8         App,
9     }
10 })

```

第三次优化

让模板 template 与 对象分离

创建 vue/App.vue

```
1 <template>
2   <div>
3     <h2>{{ message }}</h2>
4     <h2>{{ msg }}</h2>
5     <button>按钮</button>
6   </div>
7 </template>
8
9 <script>
10 export default {
11   data() {
12     return {
13       message: 'Hello Webpack',
14       msg: '好好学习，天天向上',
15     }
16   }
17 }
18 </script>
19
20 <style scoped>
21
22 </style>
```

```
1 import Vue from 'vue'
2 import App from './vue/App.vue'
3
4 new Vue({
5   el: '#app',
6   template: `<App/>`,
7   components: {
8     App,
```



```
9     }  
10  })
```

这个时候需要配置 webpack 来编译 .vue 文件

```
npm install vue-loader vue-template-compiler --save-dev
```

遇到问题降低一下版本，新版本需要装额外的插件

```
"vue-loader": "^13.0.0"
```

webpack 配置 vue-loader

```
1 // .vue 文件编译  
2 {  
3   test: /\.vue$/,  
4   use: ['vue-loader']  
5 }
```

webpack plugin

插件一般是对 webpack 现有的一些功能进行扩充。

npm 进行插件安装

版权声明

banner --> 横幅

```
1 const webpack = require('webpack')  
2  
3 module.export = {  
4   ...  
5   plugins: [  
6     new webpack.BannerPlugin('版权 xxx 所有')  
7   ]  
8 }
```

打包 index.html 文件到 dist

```
npm install html-webpack-plugin --save-dev
```

```
"html-webpack-plugin": "^3.2.0",
```

报错请安装指定的版本

src/index.html

```
<div id="app">
```

```
</div>
```

script 会自动插入，所以之前配置的 publicPath 应该注释掉

```
1 const webpack = require('webpack')
2 const HTMLWebpackPlugin = require('html-webpack-plugin')
3
4 module.export = {
5   ...
6   plugins: [
7     new webpack.BannerPlugin('版权 xxx 所有'),
8     new HTMLWebpackPlugin({
9       template: 'index.html'
10    }),
11   ]
12 }
```

对 js 文件进行压缩

```
npm install uglifyjs-webpack-plugin@1.1.1 --save-dev
```

```
1 const webpack = require('webpack')
2 const HTMLWebpackPlugin = require('html-webpack-plugin')
3 const UglifyjsWebpackPlugin = require('uglifyjs-webpack-plugin')
4
5 module.export = {
6   ...
7   plugins: [
8     // 版权声明，这个插件是 webpack 自带的
9     new webpack.BannerPlugin('版权 xxx 所有'),
10    // 打包 html
```

```

11     new HTMLWebpackPlugin({
12         template: 'index.html'
13     }),
14     // 压缩 js
15     new UglifyJsWebpackPlugin(),
16 ]
17 }

```

搭建本地服务器

为了解决每次修改后都需要重新 build 的问题，可以基于 node.js express 框架 搭建一个服务器，实现修改后浏览器自动刷新。

```
npm install --save-dev webpack-dev-server@2.9.1
```

在 webpack 中配置 devServer

contentBase: 编译好的静态文件，默认是根目录，这里填写 './dist'

port: 端口

inline: 页面实时刷新

historyApiFallback: 在 SPA 页面中，依赖 html5 的 history 模式

```

1 module.export = {
2     ...
3     devServer: {
4         contentBase: './dist',
5         inline: true,
6         port: 8080,
7     }
8 }
9
10
11 // 这个地方还要配置一下 package.json script
12
13 "dev": webpack-dev-server,

```

```
npm run dev
```

 开启服务

webpack 配置分离

分成开发时配置、编译时配置

将配置分成 基础、开发、生产 配置，新建文件夹 build

build/base.config.js

build/dev.config.js

build/prod.config.js

配置合并

```
npm install webpack-merge --save-dev
```

修改 package.json script 配置文件

```
1 "dev": "webpack-dev-server --config ./build/dev.config.js"
2 "build": "webpack --config ./build/prod.config.js"
```

 [pack.zip](#)