



谢成

webpack
MODULE BUNDLER

为什么要使用webpack

现今的很多网页其实可以看做是功能丰富的应用，它们拥有着复杂的JavaScript代码和一大堆依赖包。为了简化开发的复杂度，前端社区涌现出了很多好的实践方法

模块化，让我们可以把复杂的程序细化为小的文件；
类似于TypeScript这种在JavaScript基础上拓展的开发语言：使我们能够实现目前版本的JavaScript不能直接使用的特性，并且之后还能转换为JavaScript文件使浏览器可以识别；

Scss，less等CSS预处理器

...

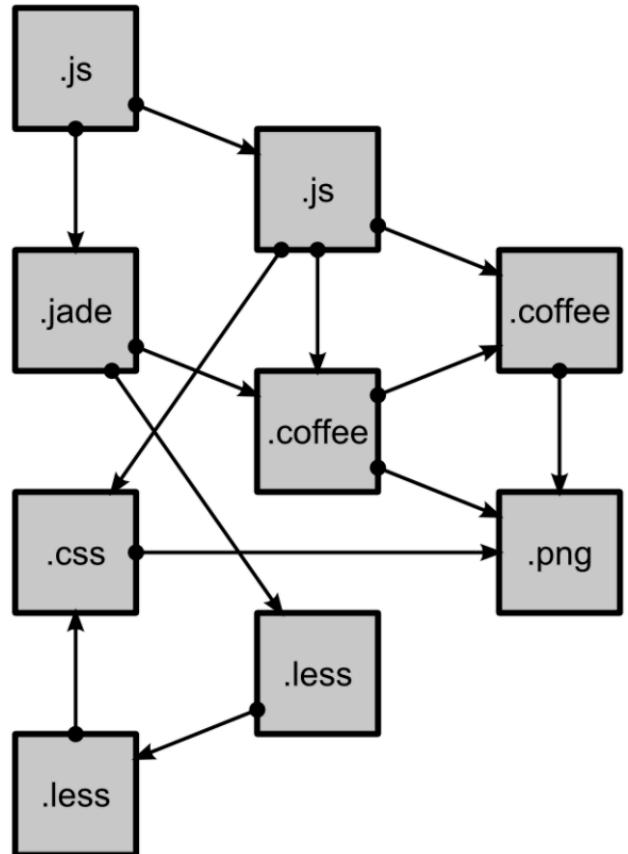
这些改进确实大大的提高了我们的开发效率，但是利用它们开发的文件往往需要进行额外的处理才能让浏览器识别，而手动处理又是非常繁琐的，这就为WebPack类的工具的出现提供了需求。

什么是webpack

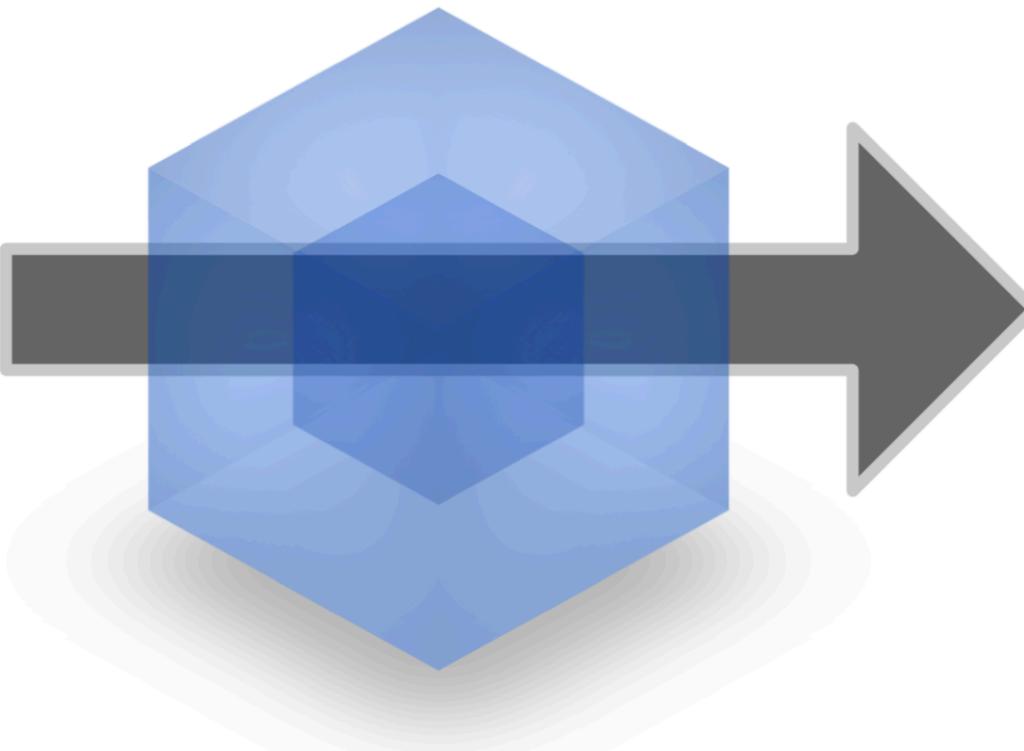
WebPack可以看做是模块打包机：它做的事情是，分析你的项目结构，找到JavaScript模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript等），并将其转换和打包为合适的格式供浏览器使用。

webpack 3.X版本

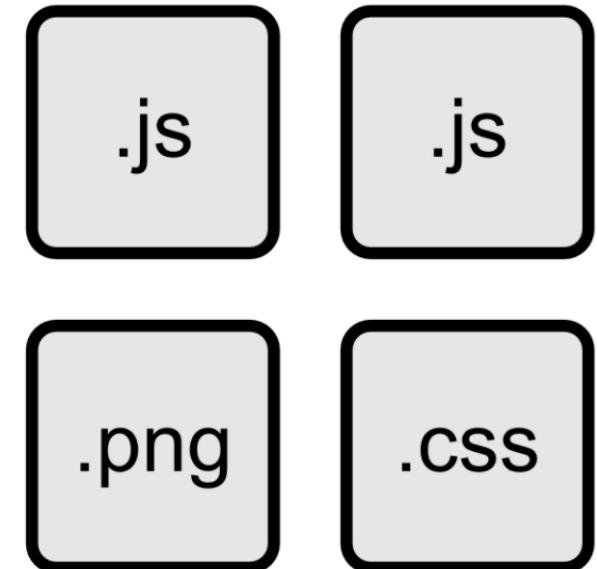
- 打包
- 转换（比如sass、less等）
- 优化



modules
with dependencies



webpack
MODULE BUNDLER



static
assets

英文: <http://webpack.js.org/>

中文: <https://doc.webpack-china.org/>

Github: <https://github.com/webpack/webpack>

安装webpack

安装node

查看版本: node -v

全局安装: npm install -g webpack

查看版本: webpack -v

进入项目目录:

npm init

npm install --save-dev webpack

```
"scripts": {  
  "build": "webpack",  
  "dev": "webpack-dev-server"  
},
```

注意:

① mac需要在命令前面加: sudo

② 如果npm命令安装慢, 可以使用cnpm(<https://npm.taobao.org/>)

Hello World

webpack src/entry.js dist/bundle.js

```
xiechengdeMacBook-Pro:webpack xiecheng$ webpack src/entry.js dist/bundle.js
Hash: ff692be429165e3191cc
Version: webpack 3.8.1
Time: 56ms
          Asset      Size  Chunks             Chunk Names
bundle.js  2.53 kB     0  [emitted]  main
[0] ./src/entry.js 59 bytes {0} [built]
```

入口与出口

webpack.config.js

```
1 const path = require('path');
2 module.exports = {
3   //入口文件的配置项
4   entry: {
5     entry: './src/entry.js',
6     entry2: './src/entry2.js'
7   },
8   //出口文件的配置项
9   output: {
10     path: path.resolve(__dirname, 'dist'),
11     filename: '[name].js'
12   },
13   //模块：例如解读CSS，图片如何转换，压缩
14   module: {},
15   //插件，用于生产模版和各项功能
16   plugins: [],
17   //配置webpack开发服务功能
18   devServer: {}
19 }
```

- **entry:** 配置入口文件的地址，可以是单一入口，也可以是多入口。
- **output:** 配置出口文件的地址，在 webpack2.X版本后，支持多出口配置。
- **module:** 配置模块，主要是解析 CSS和图片转换压缩等功能。
- **plugins:** 配置插件，根据你的需要配置不同功能的插件。
- **devServer:** 配置开发服务功能

webpack命令打包

webpack.config.js

```
17 //配置webpack开发服务功能
18 devServer: {
19     //设置基本目录结构
20     contentBase: path.resolve(__dirname, 'dist'),
21     //服务器的IP地址，可以使用IP也可以使用localhost
22     host: 'localhost',
23     //服务端压缩是否开启
24     compress: true,
25     //配置服务端口号
26     port: 8081
27 }
```

安装webpack-dev-server: npm install --save-dev webpack-dev-server

webpack-dev-server (原因: 服务装到了node_modules里面, 找不到)

package.json

```
6  "scripts": {  
7    "server": "webpack-dev-server"  
8  },
```

npm run server

注意：

- ① server可以改
- ② webpackV3.5以后的版本直接支持热更新，之前的版本需要另配置

HTML文件打包

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

```
3 const HtmlWebpackPlugin= require('html-webpack-plugin');
```

```
24 //插件，用于生产模版和各项功能
25 plugins:[
26     // new UglifyJsPlugin(),
27     new HtmlWebpackPlugin({
28         minify:{
29             removeAttributeQuotes:true
30         },
31         hash:true,
32         template:'./src/index.html'
33     })
34 ];
```

- **minify:** 是对html文件进行压缩，去掉属性的双引号
- **hash:** 为了开发中js有缓存效果，所以加入hash，这样可以有效避免缓存JS
- **template:** 是要打包的html模版路径和文件名称

CSS文件打包

在入口js文件中： `import css from './css/index.css';`

style-loader: 用来处理css文件中的url()等， url挂在到js中

css-loader: 用来将css插入到页面的style标签

安装**style-loader**: `npm install --save-dev style-loader`

安装**css-loader**: `npm install --save-dev css-loader`

```
13 //模块：例如解读CSS,图片如何转换, 压缩
14 module: {
15     rules: [
16         {
17             test: /\.css$/,
18             use: ['style-loader', 'css-loader']
19         }
20     ],
21 }
```

<https://github.com/webpack-contrib/extract-text-webpack-plugin>
安装: npm install --save-dev extract-text-webpack-plugin

```
4 const ExtractTextPlugin = require("extract-text-webpack-plugin");
```

```
module: {
  rules: [
    {
      test: /\.css$/,
      // use: ['style-loader', 'css-loader']
      use: ExtractTextPlugin.extract({
        fallback: "style-loader",
        use: "css-loader"
      })
    },
  ]}
```

```
plugins: [
  // new UglifyJsPlugin(),
  new HtmlWebpackPlugin({
    minify: {
      removeAttributeQuotes: true
    },
    hash: true,
    template: './src/index.html'
  }),
  new ExtractTextPlugin("css/index.css")
],
```

css文件会分离出来，但如果图片不是base64而是一个图片文件，这时候就会出现路径问题，需要配置publicPath来解决

```
12 //出口文件的配置项
13 output: {
14     path: path.resolve(__dirname, 'dist'),
15     filename: '[name].js',
16     publicPath: 'http://localhost:8081/'  
17 }
```

js代码压缩

webpack.config.js

```
2 const UglifyJsPlugin = require('uglifyjs-webpack-plugin');  
23 //插件，用于生产模版和各项功能  
24 plugins: [  
25     new UglifyJsPlugin()  
26 ],
```

npm run build命令

```
ERROR in entry.js from UglifyJs  
Unexpected token: name (urlParts) [entry.js:317,4]
```

```
ERROR in entry2.js from UglifyJs  
Unexpected token: name (urlParts) [entry2.js:317,4]
```

npm run dev

- 开发环境
- 生产环境

CSS中引用图片

```
background-image: url(..../images/logo.png);
```

npm install --save-dev file-loader url-loader

```
19 module: {
20   rules: [
21     {
22       test: /\.css$/,
23       // use: ['style-loader', 'css-loader']
24       use: ExtractTextPlugin.extract({
25         fallback: "style-loader",
26         use: "css-loader"
27       })
28     }, {
29       test: /\.(png|jpg|gif)/,
30       use: [
31         {
32           loader: 'url-loader',
33           options: {
34             limit: 500,
35             outputPath: 'images/'
36           }
37         }
38       ]
39     },
40   ],
41 }
```

- **test: \.(png|jpg|gif)** /是匹配图片文件后缀名称。
- **use:** 是指定使用的loader和loader的配置参数。
- **limit:** 是把小于**500000B**的文件打成Base64的格式，写入JS。

HTML中的图片打包

```
<div id="div3">  
    
</div>
```

<https://github.com/wzsxyz/html-withimg-loader>

npm install html-withimg-loader --save-dev

```
}, {  
  test: /\.(\htm|html)$/i,  
  loader: 'html-withimg-loader'  
}
```

Sass打包和分离

npm install --save-dev node-sass sass-loader

```
}, {  
  test: /\.scss/,  
  use: [  
    {  
      loader: 'style-loader'  
    },  
    {  
      loader: 'css-loader'  
    },  
    {  
      loader: 'sass-loader'  
    }  
  ]  
}
```

在js文件中导入scss文件

```
2 import sass from './css/common.scss';
```

```
use: ExtractTextPlugin.extract({  
  use: [  
    {  
      loader: "css-loader"  
    }, {  
      loader: "sass-loader"  
    }],  
  fallback: "style-loader"  
})
```

<https://github.com/postcss/postcss-loader>

npm install --save-dev postcss-loader autoprefixer
新建文件： postcss.config.js

The image shows a code editor with two snippets of `postcss.config.js` code side-by-side.

Left Snippet:

```
④ postcss.config.js x
1 module.exports = {
2   plugins: [
3     require('autoprefixer')
4   ]
5 };
```

Right Snippet:

```
module: {
  rules: [
    {
      test: /\.css$/,
      // use: ['style-loader', 'css-loader']
      use: ExtractTextPlugin.extract({
        fallback: "style-loader",
        use: [
          {
            loader: "css-loader",
            options: {importLoaders: 1}
          },
          'postcss-loader'
        ],
        }
      })
    }
  }
};
```

清除未使用的css

<https://github.com/webpack-contrib/purifycss-webpack>

npm install --save-dev purifycss-webpack purify-css

```
2 const glob = require('glob');
```

```
6 const PurifyCSSPlugin = require("purifycss-webpack");
```

```
new PurifyCSSPlugin({
  paths: glob.sync(path.join(__dirname, 'src/*.html')),
})
```

babel

npm install --save-dev babel-core babel-loader babel-preset-es2015
babel-preset-react

```
}, {  
  test: /\.jsx|js$/,
  use: {
    loader: 'babel-loader',
    options: {
      presets: [
        "es2015", "react"
      ]
    }
  },
  exclude: /node_modules/
}
```

babel-preset-env

打包注释

```
new webpack.BannerPlugin('成哥所有，翻版必究！')
```

模块化配置

```
▲ └── webpack_config
      └── JS entry_webpack.js
```

```
1 const entry = {
2   entry: './src/entry.js'
3 };
4 module.exports = entry;
```

```
8 const entry = require("./webpack_config/entry_webpack.js");
```

```
module.exports = {
  //入口文件的配置项
  entry: entry,
  //出口文件的配置项
};
```

开发环境与生产环境

`devDependencies` 存放测试代码依赖的包或构建工具的包

`dependencies` 存放项目或组件代码中依赖到的

安装全部项目依赖包: `npm install`

安装生产环境依赖包: `npm install --production`

`npm install jquery --save`

打包第三方类库1

npm install --save jquery

3 import \$ from 'jquery';

9 const webpack = require("webpack");

```
new webpack.ProvidePlugin({
  $: "jquery"
})
```

打包第三方类库2

JS entry webpack.js ×

```
1 const entry = {  
2   entry: './src/entry.js',  
3   jquery: 'jquery'  
4 };  
5 module.exports = entry;
```

```
new webpack.optimize.CommonsChunkPlugin({  
  //name对应入口文件中的名字  
  name: 'jquery',  
  //把文件打包到哪里，是一个路径  
  filename: "assets/js/jquery.js",  
  //最小打包的文件模块数，这里直接写2就好  
  minChunks: 2  
})
```

打包第三方类库3

多个第三方类库： 比如 `npm install --save vue`

```
new webpack.optimize.CommonsChunkPlugin({
    //name对应入口文件中的名字
    name: ['jquery', 'vue'],
    //把文件打包到哪里，是一个路径
    filename: "assets/js/[name].js",
    //最小打包的文件模块数，这里直接写2就好
    minChunks: 2
})
```

资源拷贝

npm install --save-dev copy-webpack-plugin

```
9 var CopyWebpackPlugin = require('copy-webpack-plugin');
```

```
new CopyWebpackPlugin([
  {
    from: __dirname + '/src/public',
    to: './public'
}])
```

Json文件

根目录下创建json文件：

```
{ } author.json x  
1 {  
2   "name": "xiecheng",  
3   "age": 30,  
4   "company": "weichuang"  
5 }
```

```
JS entry.js x  
12 let json = require('../author.json');  
13 document.querySelector('#json').innerHTML = `作者: ${json.name}, 年龄: ${json.age}, 公司: ${json.company}`;
```

watch

webpack --watch

```
109     watchOptions: {  
110         //检测修改的时间，以毫秒为单位  
111         poll: 1000,  
112         //防止重复保存而发生重复编译错误。这里设置的500是半秒内重复保存，不进行打包操作  
113         aggregateTimeout: 500,  
114         //不监听的目录  
115         ignored: /node_modules/,  
116     }
```



Thank you

谢

谢

观

看