

WEBPACK学习记录，从零搭建vue环境，初试webpack（一）

QQ: 512469231

博主ID: Inory、博丽灵梦

学习之前，默认我们是纯小白，因为博主也是刚刚找到工作的小白，网上有很多文章，但是自己写的文章，总是会在自己迷茫的时候，受到微妙的启发，所以，乐于分享，才是自己进步的开始，如果其他人因为我的文章学会了一小部分知识，我会觉得很高兴！

下个文章写微信小程序吧，下个项目就是小程序。如果项目是pc端的话，我会用webpack搭建环境并使用的，毕竟只有在实际使用才会得到非常大的进步。

话不多说，开始吧，

开局，初始化文件目录

```
npm init - y
```

得到package.json文件和package-lock.json

```
{
  "name": "webpack3",    //你的文件目录名字
  "version": "1.0.0",    //版本号
  "description": "",      //描述
  "main": "index.js",    // - main 字段指定了程序的主入口文件，require('moduleName') 就会加载这个文件。这个字段的默认值是模块根目录下面的 index.js。
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],        //关键字
  "author": "",          //作者
  "license": "ISC",      //你应该为你的模块制定一个协议，让用户知道他们有何权限来使用你的模块，以及使用该模块有哪些限制。最简单的，例如你用BSD-3-Clause 或 MIT之类的协议，如下：{ "license" : "BSD-3-Clause" } 你可以在https://spdx.org/licenses/ 这个地址查阅协议列表 。
  "devDependencies": {
    "webpack": "^4.44.2",
    "webpack-cli": "^3.3.12"
  }
}
```

package-lock.json文件，自己去看这个网址。。太多不做介绍，是人就能看懂<https://www.cnblogs.com/wangweizhang/p/10530294.html>

安装webpack

```
npm install webpack webpack-cli --save-dev
```

安装完后，得到

```
"devDependencies": {
  "webpack": "^4.44.2",
  "webpack-cli": "^3.3.12"
}
```

为了验证我们的webpack是否可以开箱即用，首先创建几个文件体验一下。那我们按照vuecli自带的格式创建几个基础文件吧，格式如下

dist/index.html （自己手动引入文件将要打包的js文件）

src/main.js

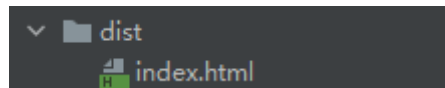
src/assets

src/assets/js/common.js

根据官方的文档，我们的入口文件是main，所以要把package.json的main给干掉，直接删掉就行，然后增加private这个参数，以便确保我们安装包是 `private`(私有的)，并且移除 `main` 入口。这可以防止意外发布你的代码。



```
1 {
2   "name": "webpack3",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "private": true,
7   "scripts": {
8     "test": "echo \\\"Error: no test specified\\\" && exit"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "devDependencies": {
14    "webpack": "^4.44.2",
15    "webpack-cli": "^3.3.12"
16  }
17 }
18
```



```
dist
  index.html
```

```
index.js × package.json × 101 main.js × index.html × common.js ×
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport"
6     content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>Document</title>
9 </head>
10 <body>
11   <script type="module" src="./main.js"></script>
12 </body>
13 </html>
```

开始打包

```
npx webpack
```

你会发现报错。。

为什么呢？因为我也报错了。

我们来看webpack官网

使用一个配置文件

在 webpack v4 中，可以无须任何配置，然而大多数项目会需要很复杂的设置，这就是为什么 webpack 仍然要支持 [配置文件](#)。这比在 terminal(终端) 中手动输入大量命令要高效的多，所以让我们创建一个配置文件：

官网的说明，表示webpack4中，不需要任何的配置文件，也就是说，没有指定入口和出口，直接默认src/index.js，如果你改成其他文件，就会无法进行打包，所以，如果想不需要配置，就要默认index.js，或者自行添加配置文件进行配置，这个到后面我们再来配置。

我们把main.js改成index.js

。。。

继续打包，在打包之前，先删除原来的打包文件，后面我们会让webpack帮我们删除

```
选择管理员: npm
Alternatively, run 'webpack(-cli) --help' for usage info.
Hash: 028f1e6da7ff421ad3bd
Version: webpack 4.44.2
Time: 32ms
Built at: 2020/10/03 下午8:29:57

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode' option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/configuration/mode/

ERROR in Entry module not found: Error: Can't resolve './src' in 'F:\我的项目\webpack3'

F:\我的项目\webpack3>npx webpack
Hash: 9e27099ddc6b6b752718
Version: webpack 4.44.2
Time: 267ms
Built at: 2020/10/03 下午9:07:26
    Asset      Size  Chunks             Chunk Names
  main.js  1.01 KiB       0  [emitted]  main
Entrypoint main = main.js
[0] ./src/index.js + 1 modules 170 bytes {0} [built]
   |   ./src/index.js 50 bytes [built]
   |   ./src/assets/js/common.js 120 bytes [built]

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode' option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/configuration/mode/

F:\我的项目\webpack3>
```

成功啦!!!

我们的webpack已经成功跑起来了!!!

配置文件

刚才我们说到，webpack默认是index.js，所以我们开始改造

在根目录新建webpack.config.js

在这之前，先看会文档<https://webpack.docschina.org/concepts/>

```
const path = require("path");module.exports = {
  entry:{
    ..... //设置入口文件
  },
  output:{
    ..... //设置出口文件
  },
  module:{
    ..... //配置loader，注意使用rules而不是loaders
  },
  plugins:[
    ..... //注意是数组
  ],
  devServer:{
    //我们在这里对webpack-dev-server进行配置
  }
}
```

了解下配置文件应该怎么去写，如果我们直接对着教程抄，其实还是挺迷茫的

在开始前你需要先理解一些核心概念：

- 入口(entry)
- 输出(output)
- loader
- 插件(plugin)
- 模式(mode)
- 浏览器兼容性(browser compatibility)
- 环境(environment)

看到官网标记的几个核心概念，大概花5分钟过一遍，不要心浮气躁。。。

好了，虽然你看不懂，但是没关系，我们来实际操作，实际博主也是边学边写文章

编辑器打开webpack.config.js

emmm，应该写什么东西。。。？

回到刚才的思路，我们要改index.js的文件名字，修改成main.js

现在把我们的src/index.js修改成index.js

回忆我们刚刚逛的官网说明，官网上写的入口

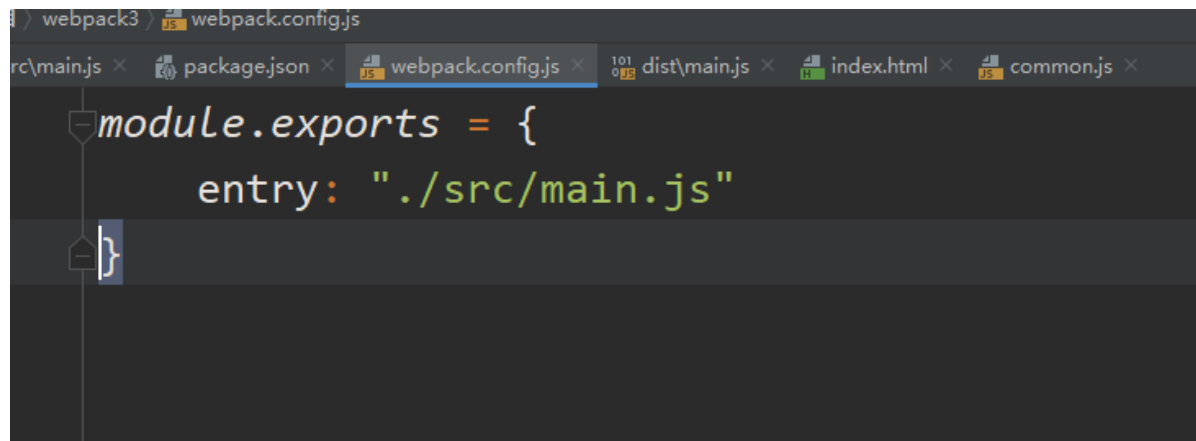
从 webpack v4.0.0 开始，可以不用引入一个配置文件。然而，webpack 仍然还是高度可配置的。在开始前你需要先理解四个核心概念：

- 入口(entry)
- 输出(output)
- loader
- 插件(plugings)

本文档旨在给出这些概念的高度概述，同时提供具体概念的详尽相关用例。

入口就是我们的main.js

那现在开始dong手



```
module.exports = {  
  entry: './src/main.js'  
}
```

运行命令npx webpack

在这之前确保自己把src里面的index文件修改成main

```
F:\我的项目\webpack3>npx webpack
Hash: e7b7eb80ceba179239da
Version: webpack 4.44.2
Time: 72ms
Built at: 2020/10/03 下午9:48:14
    Asset      Size  Chunks             Chunk Names
main.js  1.11 KiB       0  [emitted]  main
Entrypoint main = main.js
[0] ./src/main.js + 1 modules 279 bytes {0} [built]
|   ./src/main.js 50 bytes [built]
|_  ./src/assets/js/common.js 229 bytes [built]

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode' option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/configuration/mode/

F:\我的项目\webpack3>
```

入口文件到这里结束了吗???

没有，如果你往官网下面看，你会发现，还有几个比较精彩的学习点

那就是多入口

JS文件嘛，那肯定是js语法，如果我们改成对象语法，会如何呢？

来动手！

先删除原来的打包文件

```
module.exports = {
  entry: {
    main: "./src/main.js",
    index: "./src/index.js",
    app: "./src/app.js"
  }
}
```

```
<body>
  <script type="module" src="./main.js"></script>
  <script src="app.js"></script>
  <script src="index.js"></script>
</body>
</html>
```

运行命令

```

F:\我的项目\webpack3>npx webpack
Hash: 69cb3603d0e9fd15981b
Version: webpack 4.44.2
Time: 238ms
Built at: 2020/10/03 下午10:01:46
    Asset      Size  Chunks             Chunk Names
  app.js    962 bytes          0  [emitted]  app
  index.js   963 bytes          1  [emitted]  index
  main.js   1.11 KiB          2  [emitted]  main
Entrypoint main = main.js
Entrypoint index = index.js
Entrypoint app = app.js
[0] ./src/index.js 34 bytes {1} [built]
[1] ./src/app.js 32 bytes {0} [built]
[2] ./src/main.js + 1 modules 279 bytes {2} [built]
    |./src/main.js 50 bytes [built]
    |./src/assets/js/common.js 229 bytes [built]

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode' option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/configuration/mode/

F:\我的项目\webpack3>

```

看到dist文件，发现多了3个打包的js文件，说明对象语法是可用的

输出文件output

啥?这webpack只能改个入口? 那也太拉闸了，继续对着官网往下看，来到输出文件

啥是输出文件呢?

就是打包之后的文件

用法(Usage)

在 webpack 中配置 `output` 属性的最低要求是，将它的值设置为一个对象，包括以下两点：

- `filename` 用于输出文件的文件名。
- 目标输出目录 `path` 的绝对路径。

知道这个之后

开始dong手写我们的配置文件

```
module.exports = {  
  entry: {  
    main: "./src/main.js",  
    index: "./src/index.js",  
    app: "./src/app.js"  
  },  
  output: {  
    filename: "index.js"  
  }  
}
```

运行

报错

为什么呢？因为我们有多个入口。

如果你把index和app删了，你会发现会成功运行

因为我们文件没有重复

所以这里我们用到动态属性


```

config.js
module.exports = {
  entry: {
    main: "./src/main.js",
    index: "./src/index.js",
    app: "./src/app.js"
  },
  output: {
    filename: "[name].js"
  }
}

```

运行

```

F:\我的项目\webpack3>npx webpack
Hash: 69cb3603d0e9fd15981b
Version: webpack 4.44.2
Time: 75ms
Built at: 2020/10/03 下午10:26:38
   Asset      Size  Chunks             Chunk Names
  app.js    962 bytes          0  [emitted]      app
 index.js    963 bytes          1  [emitted]      index
 main.js    1.11 KiB          2  [emitted]      main
Entrypoint main = main.js
Entrypoint index = index.js
Entrypoint app = app.js
[0] ./src/index.js 34 bytes {1} [built]
[1] ./src/app.js 32 bytes {0} [built]
[2] ./src/main.js + 1 modules 279 bytes {2} [built]
    | ./src/main.js 50 bytes [built]
    | ./src/assets/js/common.js 229 bytes [built]
WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode' option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/configuration/mode/
F:\我的项目\webpack3>_

```

生产环境和开发环境

在我们日常写代码中，肯定是有生产环境和开发环境的

通俗点，

开发环境：线下给自己看的

生产环境：上线给用户看的

- ✓ 模式(mode)
 - 用法
 - mode: development
 - mode: production

现在我们有问题，应该怎么写？我们只有一个配置文件啊，咋这么麻烦。。

回到官网

mode: development

```
// webpack.development.config.js
module.exports = {
+ mode: 'development'
- plugins: [
-   new webpack.NamedModulesPlugin(),
-   new webpack.DefinePlugin({ "process.env.NODE_ENV": JSON.stringify("development") }),
- ]
}
```

mode: production

```
// webpack.production.config.js
module.exports = {
+ mode: 'production',
- plugins: [
-   new UglifyJsPlugin(/* ... */),
-   new webpack.DefinePlugin({ "process.env.NODE_ENV": JSON.stringify("production") }),
-   new webpack.optimize.ModuleConcatenationPlugin(),
-   new webpack.NoEmitOnErrorsPlugin()
- ]
}
```

官网为我们分开了两个文件，那我们也来试试，新建两个文件

webpack.production.config.js

webpack.development.config.js

名字还挺长的。。。建议直接复制！

建好了

怎么写。。。？

既然是配置文件，那肯定和我们刚刚写的其实是一样的，只需要在配置文件里面加上mode这个字段

```
onfig.js
webpack.config.js x webpack.production.config.js x webpack.development.config.js x
1 module.exports = {
2   mode: "production"
3 }
```

这样就可以了吗???

回到官网

支持以下字符串值:

选项	描述
development	会将 <code>process.env.NODE_ENV</code> 的值设为 <code>development</code> 。启用 <code>NamedChunksPlugin</code> 和 <code>NamedModulesPlugin</code> 。
production	会将 <code>process.env.NODE_ENV</code> 的值设为 <code>production</code> 。启用 <code>FlagDependencyUsagePlugin</code> , <code>FlagIncludedChunksPlugin</code> , <code>ModuleConcatenationPlugin</code> , <code>NoEmitOnErrorsPlugin</code> , <code>OccurrenceOrderPlugin</code> , <code>SideEffectsFlagPlugin</code> 和 <code>UglifyJsPlugin</code> 。

记住, 只设置 `NODE_ENV`, 则不会自动设置 `mode`。

这一堆英文, 好像有点难懂, 那我们找文档看, 都是啥东西。。。?

以下是文档的链接, 大家可以自己点进去看, 下面就不复制长篇大论了。。

`NamedModulesPlugin` <https://webpack.js.org/migrate/5/#update-outdated-options>

`NamedChunksPlugin` <https://webpack.html.cn/plugins/commons-chunk-plugin.html>

`FlagDependencyUsagePlugin` 删除无用代码的, 其他插件依赖

`FlagIncludedChunksPlugin` <https://blog.csdn.net/hyupeng1006/article/details/89241630>

`ModuleConcatenationPlugin` <https://www.webpackjs.com/plugins/module-concatenation-plugin/>

`NoEmitOnErrorsPlugin` <https://www.webpackjs.com/plugins/no-emit-on-errors-plugin/>

`OccurrenceOrderPlugin` <https://webpack.html.cn/guides/migrating.html>

`SideEffectsFlagPlugin` 移除未使用的模块 https://blog.csdn.net/weixin_34346099/article/details/89140946

`TerserPlugin` https://blog.csdn.net/weixin_34013044/article/details/88671739

登录后复制

FlagIncludedChunksPlugin: 检测并标记模块之间的从属关系。

ModuleConcatenationPlugin: 可以让webpack根据模块间的关系依赖图中, 将所有的模块连接成一个模块。

SideEffectsFlagPlugin: 告诉webpack去清除一个大的模块文件中的未使用的代码, 这个大的文件模块可以是自定义

OccurrenceOrderPlugin: 告诉webpack各个模块间的先后顺序, 这样可以实现最优的构建输出。

TerserPlugin: 它替代了uglifyjs-webpack-plugin插件。它的作用依然是对构建输出的代码进行压缩。

简单介绍一下上面所提到的插件吧

NamedChunksPlugin: 根据文件名称生成稳定chunkid,

NamedModulesPlugin: 热更新时使用的插件, 显示模块的相对路径。

FlagDependencyUsagePlugin: 删除无用代码的, 其他插件依赖

FlagIncludedChunksPlugin: 删除无用代码的, 其他插件依赖

ModuleConcatenationPlugin: 在webpack打包时, 将bundle 内各个模块预编译到一个闭包中, 提升代码在浏览器中的执行速度 (相比之前的打包方式---每个模块都是一个闭包)。

NoEmitOnErrorsPlugin: 在编译出现错误时, 跳过输出阶段。这样可以确保输出资源不会包含错误

OccurrenceOrderPlugin: 比对id的使用频率和分布来得出最短的id分配给使用频率高的模块

SideEffectsFlagPlugin: Tree shaking

UglifyJsPlugin: js代码压缩

这是几个大佬博客上面的截图, 可以简单的看下, 具体的可以自己找文档看一下。

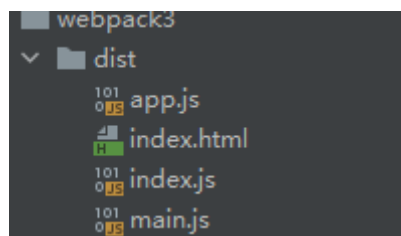
其实大概也就这样子了, 虽然官网写的挺简单的, 我们还是要自己写一遍

```
webpack.config.js x webpack.production.config.js x webpack.development.config.js x
2  module.exports = {
3      mode: "development",
4      entry: {
5          main: "./src/main.js"
6      },
7      output: {
8          filename: "[name].js"
9      },
```

在开发环境下编写了和之前一样的代码

理论上, 他只会打包我们的main.js这个文件, 那么究竟是不是呢?

来尝试下



发现我们的dist出现了3个文件??? 是哪里不对呢, 我们用的是npx webpack这个命令, 但是我们之前一直用的是webpack.config.js这个配置文件, 我们写在webpack.development.config.js这个文件里面, 他是无法识别的, 所以, 我们要让webpack识别这个文件, 所以只能从指令下手, 回到官网

生产环境

- 配置
- NPM Scripts
- 指定 mode
- 压缩(Minification)
- 源码映射(Source Mapping)
- 压缩 CSS
- CLI 替代选项

API 概念 配置 指南 LOADER 迁移 PLUGIN

NPM Scripts

现在, 我们把 `scripts` 重新指向到新配置。让 `npm start` script 中 `webpack-dev-server`, 使用 `webpack.dev.js`, 而让 `npm run build` script 使用 `webpack.prod.js`:

package.json

```
{
  "name": "development",
  "version": "1.0.0",
  "description": "",
  "main": "src/index.js",
  "scripts": {
    - "start": "webpack-dev-server --open",
    + "start": "webpack-dev-server --open --config webpack.dev.js",
    - "build": "webpack"
    + "build": "webpack --config webpack.prod.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "clean-webpack-plugin": "^0.1.17",
    "css-loader": "^0.28.4",
    "csv-loader": "^2.1.1",
    "express": "^4.15.3",
    "file-loader": "^0.11.2",
    "html-webpack-plugin": "^2.29.0",
    "style-loader": "^0.18.2",
    "webpack": "^4.30.0",
    "webpack-dev-middleware": "^1.12.0",
    "webpack-dev-server": "^2.9.1",
    "webpack-merge": "^4.1.0",
    "xml-loader": "^1.2.1"
  }
}
```

这个应该就是我们想要的答案了

打开package.json

```
webpack.config.js x package.json x webpack.production.config.js x webpack.development.config.js x
1 {
2   "name": "webpack3",
3   "version": "1.0.0",
4   "description": "",
5   "private": true,
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   }
}
```

现在我们的script里面啥都没有，我们来自动手写一条指令吧！但是在这之前，我们应该怎么去写命令呢？我们不知道有啥命令给我们写啊？？？

总不能随便乱写吧，回到官网先找下文档

噢噢噢，花5分钟看看文档，这是我在官网找的，直接打开看就行~~~

<https://webpack.docschina.org/configuration/dev-server/#devserver>

<https://webpack.docschina.org/api/cli/>

<https://webpack.docschina.org/configuration/>

是不是感觉舒服了不少？

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "build": "webpack"
},
```

简单来个build，试试我们的命令

```
npm run build
```

```
F:\我的项目\webpack3>npm run build
> webpack3@1.0.0 build F:\我的项目\webpack3
> webpack

Hash: 69cb3603d0e9fd15981b
Version: webpack 4.44.2
Time: 75ms
Built at: 2020/10/04 上午11:31:50
   Asset      Size  Chunks             Chunk Names
  app.js    962 bytes          0  [emitted]  app
  index.js   963 bytes          1  [emitted]  index
  main.js   1.11 KiB          2  [emitted]  main
Entrypoint main = main.js
Entrypoint index = index.js
Entrypoint app = app.js
[0] ./src/index.js 34 bytes {1} [built]
[1] ./src/app.js 32 bytes {0} [built]
[2] ./src/main.js + 1 modules 279 bytes {2} [built]
    |./src/main.js 50 bytes [built]
    |./src/assets/js/common.js 229 bytes [built]

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode' option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/configuration/mode/

F:\我的项目\webpack3>
```

运行成功

那我们来尝试指定配置文件，分别在两个配置文件，写上不同的参数

```
module.exports = {  
  mode: "production",  
  entry: {  
    main: "./src/main.js",  
    app: "./src/app.js"  
  },  
  output: {  
    filename: "[name].js"  
  }  
}
```

```
module.exports = {  
  mode: "development",  
  entry: {  
    main: "./src/main.js"  
  },  
  output: {  
    filename: "[name].js"  
  },  
}
```

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "webpackDev": "webpack --config webpack.development.config.js",  
  "webpackPro": "webpack --config webpack.production.config.js"  
},
```

运行命令,

```
npm run webpackDev
```

```
npm run webpackPro
```

执行成功

```
管理员: npm

> webpack3@1.0.0 webpack F:\我的项目\webpack3
> webpack --config webpack.development.config.js

Hash: 04b31cdcf16e8f072e09
Version: webpack 4.44.2
Time: 59ms
Built at: 2020/10/04 下午12:16:52
   Asset      Size  Chunks             Chunk Names
main.js  4.78 KiB       0  [emitted]  main
Entrypoint main = main.js
[0] ./src/assets/js/common.js 229 bytes {main} [built]
[1] ./src/main.js 50 bytes {main} [built]

F:\我的项目\webpack3> npm run webpackPro

> webpack3@1.0.0 webpackPro F:\我的项目\webpack3
> webpack --config webpack.production.config.js

Hash: 77623c29bb80cf14243f
Version: webpack 4.44.2
Time: 78ms
Built at: 2020/10/04 下午12:18:41
   Asset      Size  Chunks             Chunk Names
app.js    961 bytes       0  [emitted]  app
main.js   1.11 KiB       1  [emitted]  main
Entrypoint main = main.js
Entrypoint app = app.js
[0] ./src/app.js 32 bytes {0} [built]
[1] ./src/main.js + 1 modules 279 bytes {1} [built]
    |   ./src/main.js 50 bytes [built]
    |   ./src/assets/js/common.js 229 bytes [built]

F:\我的项目\webpack3>
```

这样子两个环境就可以了，你会发现，两个环境的包，一个是未压缩的，一个是压缩成一行的

现在又有一个问题，我们两个环境，总不可能写一样的代码吧？总是会有重复的，回到官网，看看有没有相关的文档

<https://webpack.docschina.org/guides/production/>

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

development(开发环境) 和 *production(生产环境)* 这两个环境下的构建目标存在着巨大差异。在开发环境中，我们需要：强大的 source map 和一个有着 live reloading(实时重新加载) 或 hot module replacement(热模块替换) 能力的 localhost server。而生产环境目标则转移至其他方面，关注点在于压缩 bundle、更轻量的 source map、资源优化等，通过这些优化方式改善加载时间。由于要遵循逻辑分离，我们通常建议为每个环境编写彼此独立的 webpack 配置。

虽然，以上我们将生产环境和开发环境做了略微区分，但是，请注意，我们还是会遵循不重复原则 (Don't repeat yourself - DRY)，保留一个 "common(通用)" 配置。为了将这些配置合并在一起，我们将使用一个名为 `webpack-merge` 的工具。此工具会引用 "common" 配置，因此我们不必再在环境特定(environment-specific)的配置中编写重复代码。

我们先从安装 `webpack-merge` 开始，并将之前指南中已经成型的那些代码进行分离：

运行命令 `npm install --save-dev webpack-merge`

随便选一个配置文件写上去，把主配置 `common` 相同的内容删掉

```
const common = require("webpack.config.js");
const merge = require("webpack-merge");
module.exports = merge(common, {
  mode: "production",
  entry: {
    app: "./src/app.js"
  }
});
```

运行我们的打包命令，打包的js文件继承了主配置文件和我们的不同环境的配置文件的内容

大概就是这样子啦

loader

现在我们来处理vue

[回到官网](#)

Loader是什么？

loader

webpack 只能理解 JavaScript 和 JSON 文件，这是 webpack 开箱可用的自带能力。**loader** 让 webpack 能够去处理其他类型的文件，并将它们转换为有效 **模块**，以供应用程序使用，以及被添加到依赖图中。

注意，loader 能够 `import` 导入任何类型的模块（例如 `.css` 文件），这是 webpack 特有的功能，其他打包程序或任务执行器的可能并不支持。我们认为这种语言扩展是很有必要的，因为这可以使开发人员创建出更准确的依赖关系图。

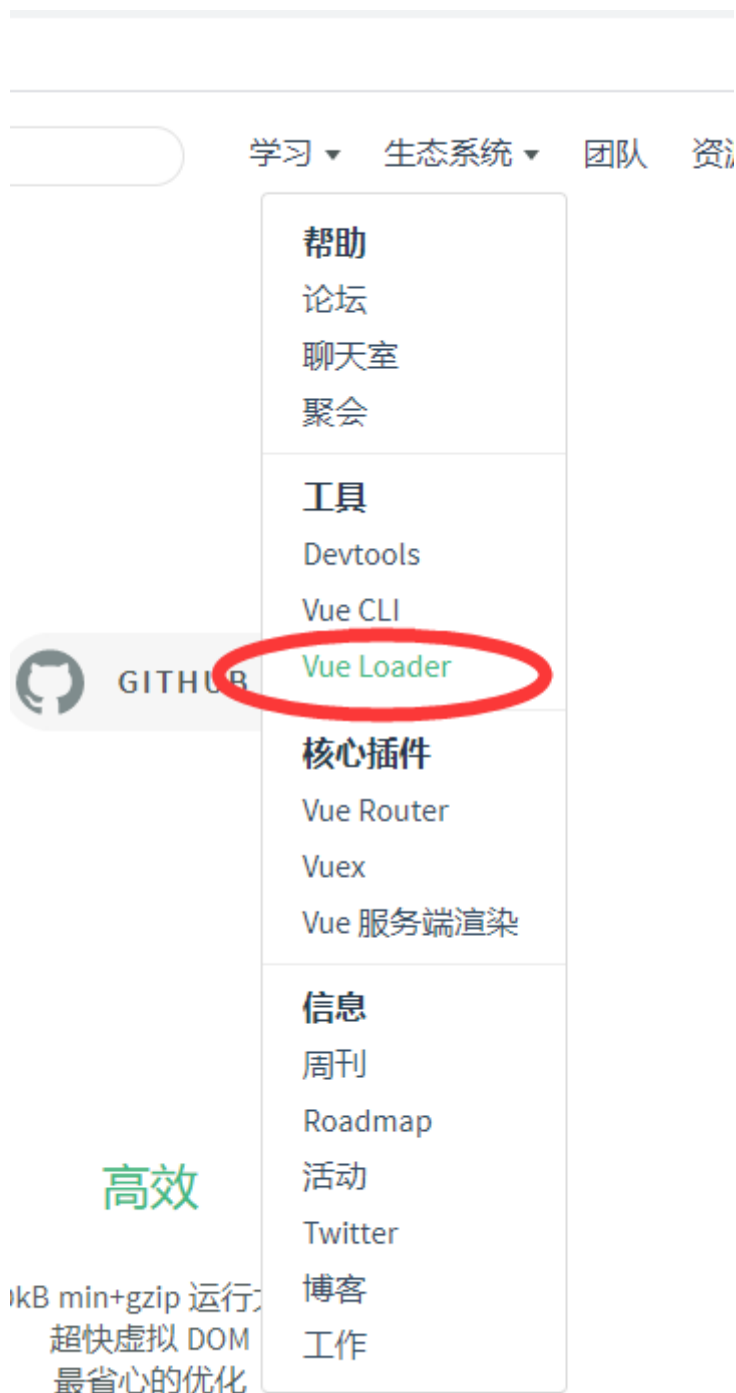
在更高层面，在 webpack 的配置中，**loader** 有两个属性：

1. `test` 属性，识别出哪些文件会被转换。
2. `use` 属性，定义出在进行转换时，应该使用哪个 loader。

接下来，我们处理vue的话，就要用到这个了，因为webpack只支持js和json

那么vue应该用哪一种呢？

vue官网有自己的loader，来到vue官网，点进去导航Vue loader



Vue Loader 是什么？

Vue Loader 是一个 [webpack](#) 的 loader，它允许你以一种名为[单文件组件 \(SFCs\)](#)的格式撰写 Vue 组件：

简单的阅读5分钟

手动设置

安装

你应该将 `vue-loader` 和 `vue-template-compiler` 一起安装——除非你是使用自行 fork 版本的 Vue 模板编译器的高阶用户：

```
npm install -D vue-loader vue-template-compiler
```

sh

主要命令

```
npm install -D vue-loader vue-template-compiler
```

复制粘贴开始

```
WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode' option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/configuration/mode/

F:\我的项目\webpack3>npm install -D vue-loader vue-template-compiler
npm WARN deprecated vue-loader@15.9.3 requires a peer of css-loader@* but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.3 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules\watchpack-chokidar2\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ vue-template-compiler@2.6.12
+ vue-loader@15.9.3
added 23 packages from 48 contributors and audited 418 packages in 9.746s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

F:\我的项目\webpack3>
```

要使用loader，就要在配置文件上面引入插件和写规则

那么应该怎么写呢？

.vue 的文件上之外，请确保在你的 webpack 配置中添加 vue loader 的插件。

```
// webpack.config.js
const VueLoaderPlugin = require('vue-loader/lib/plugin')

module.exports = {
  module: {
    rules: [
      // ... 其它规则
      {
        test: /\.vue$/,
        loader: 'vue-loader'
      }
    ]
  },
  plugins: [
    // 请确保引入这个插件！
    new VueLoaderPlugin()
  ]
}
```

Vue官网给了一个简单的易懂的截图

规则：rules 属性有test, loader

引入：plugins new xxx

安装Vue

```
npm install vue
```

我们开始写vue文件和main

在src里面创建app.vue，安装正常的vue文件编写格式

```
// App.vue
<template>
  <div id="app">
    <p @click="ClickApp">
      {{Msg}}
    </p>
  </div>
</template>
<script>
export default {
  name: "App",
  data(){
    return{
      Msg: "当你看到这个页面，说明HTML部分已经启动成功"
    }
  },
  methods: {
    ClickApp(){
      alert(this.Msg)
    }
  },
  mounted() {
    console.log("当你看到这个页面，说明App.vue的Javascript部分已经启动成功")
  }
}
</script>
```

```
// main.js
import Vue from "vue";
import App from "./App.vue";
new Vue({
  el: "#app",
  render: h => h(App)
})
```

编写webpack.config.js的规则和插件引入

```
const VueLoaderPlugin = require("vue-loader/lib/plugin");
module.exports = {
  entry: {
    main: "./src/main.js"
  },
  output: {
    filename: "[name].js"
  },
  plugins: [
    new VueLoaderPlugin()
  ],
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: "vue-loader"
      }
    ]
  }
}
```

```
}
```

vue和main如果有问题的话，可以用vue-cli创建一遍vue的项目，基本就是这格式。。。

尝试运行一次命令行npm run webpackDev，在这之前，我们把dist文件删除干净，看看会得到什么东西

```
管理员: npm
+ vue@2.6.12
added 1 package from 1 contributor and audited 423 packages in 6.302s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

F:\我的项目\webpack3>npm run webpackDev

> webpack3@1.0.0 webpackDev F:\我的项目\webpack3
> webpack --config webpack.development.config.js

Hash: b1e2fc7e7a7ebcc2b1
Version: webpack 4.44.2
Time: 1220ms
Built at: 2020/10/04 下午4:15:43
   Asset      Size  Chunks             Chunk Names
main.js  260 KiB          0  [emitted]  main
Entrypoint main = main.js
[./node_modules/vue-loader/lib/index.js?!. /src/App.vue?vue&type=script&lang=js&] ./node_modules/vue-loader/lib?vue-l
oader-options!./src/App.vue?vue&type=script&lang=js& 59 bytes {main} [built]
[./node_modules/vue-loader/lib/loaders/templateLoader.js?!. /src/App.vue?vue&type=template&id=7ba5bd90&] ./node_modules/vue-loader/lib/loaders/templateLoader.js?vue-loader-options!./node_modules
/vue-loader/lib?vue-loader-options!./src/App.vue?vue&type=template&id=7ba5bd90& 460 bytes {main} [built]
[./node_modules/webpack/buildin/global.js] (webpack)/buildin/global.js 472 bytes {main} [built]
[./src/App.vue] 1.03 KiB {main} [built]
[./src/App.vue?vue&type=script&lang=js&] 248 bytes {main} [built]
[./src/App.vue?vue&type=template&id=7ba5bd90&] 195 bytes {main} [built]
[./src/main.js] 110 bytes {main} [built]
+ 5 hidden modules

F:\我的项目\webpack3>
```

发现成功了??? 这也太简单了吧，有点奇怪!!

得到了main.js

但是，我们要的是，index.html和js文件，可以打开看的页面，但是现在，我们只看到js文件，没给我们打包html，所以我们需要一个可以自动生成html页面的插件，先找找官网的说明

```
// 请确保引入这个插件！
new VueLoaderPlugin()
}
```

这个插件是必须的！ 它的职责是将你定义过的其它规则复制并应用到 .vue 文件里相应语言的块。例如，如果你有一条匹配 /\.js\$/ 的规则，那么它会应用到 .vue 文件里的 <script> 块。

一个更完整的 webpack 配置示例看起来像这样：

也就是说，他会对应我们模块

template: html

script: js(webpack自带)

style: css

知道上面3个条件，那我们就去下载对应的loader和插件即可

首先是html

HtmlWebpackPlugin

该 `HtmlWebpackPlugin` 简化创建HTML文件的提供给您WebPack束。这对于webpack捆绑包特别有用，该捆绑包的文件名中包含哈希值，该哈希值会更改每次编译。您可以让插件为您生成一个HTML文件，使用`lodash`模板提供您自己的模板，或者使用您自己的loader。

安装

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

这是链接：

基本使用：<https://webpack.js.org/plugins/html-webpack-plugin/#root>

可配置项：<https://github.com/jantimon/html-webpack-plugin#options>

自动生成html，应该是符合的，先安装试试

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

安装完之后，我们开始编写配置文件

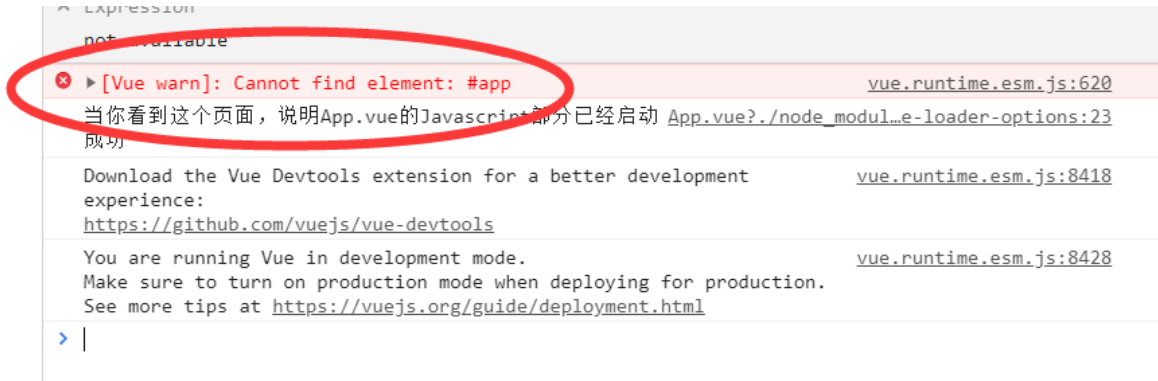
```
const VueLoaderPlugin = require("vue-loader/lib/plugin");
const HtmlWebpackPlugin = require("html-webpack-plugin")
module.exports = {
  entry: {
    main: "./src/main.js"
  },
  output: {
    filename: "[name].js"
  },
  plugins: [
    new VueLoaderPlugin(),
    new HtmlWebpackPlugin()
  ],
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: "vue-loader"
      }
    ]
  }
}
```

运行命令`npm run webpackDev`，记得先删掉原来的文件

```
管理员: npm
F:\我的项目\webpack3>npm run webpackDev
> webpack@4.44.2 webpackDev F:\我的项目\webpack3
> webpack --config webpack.config.js

Hash: 4acfd871ece57a51195
Version: webpack 4.44.2
Time: 494ms
Built at: 2020/10/04 下午6:06:39
    Asset      Size      Chunks             Chunk Names
index.html  225 bytes          0  [emitted]
main.js    260 KiB          0  [emitted]  main
Entrypoint main = main.js
[./node_modules/vue-loader/lib/index.js?!. /src/App.vue?vue&type=script&lang=js&] ./node_modules/vue-loader/lib??vue-loader-options!./src/App.vue?vue&type=script&lang=js& 348 bytes {main} [built]
[./node_modules/vue-loader/lib/loaders/templateLoader.js?!. /src/App.vue?vue&type=template&id=7ba5bd90&] ./node_modules/vue-loader/lib/loaders/templateLoader.js??vue-loader-options!./node_modules/vue-loader/lib??vue-loader-options!./src/App.vue?vue&type=template&id=7ba5bd90& 349 bytes {main} [built]
[./node_modules/webpack/buildin/global.js] (webpack)/buildin/global.js 472 bytes {main} [built]
[./src/App.vue] 1.03 KiB {main} [built]
[./src/App.vue?vue&type=script&lang=js&] 248 bytes {main} [built]
[./src/App.vue?vue&type=template&id=7ba5bd90&] 195 bytes {main} [built]
[./src/main.js] 109 bytes {main} [built]
+ 5 hidden modules
Child HtmlWebpackCompiler:
  1 asset
  Entrypoint HtmlWebpackPlugin_0 = __child-HtmlWebpackPlugin_0
  1 module
F:\我的项目\webpack3>
```

打包完后，我们看到dist文件，多了js文件和一个html文件，打开后发现报错



控制台报错，app没找到??

打开元素控制，发现确实没有元素，他没有帮我们新建div这些，如果你在里面写上div#app,你会发现，我们的vue就运行起来了

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

看到vuecli创建的目录，他自带一个public文件夹，里面有一个index.html，里面是有些div和app的id，所以，按照这个思路，是否可以把public里面的index当作模板直接写入呢？答案是有的

您可以将配置选项的哈希值传递给 `html-webpack-plugin`。允许的键如下：

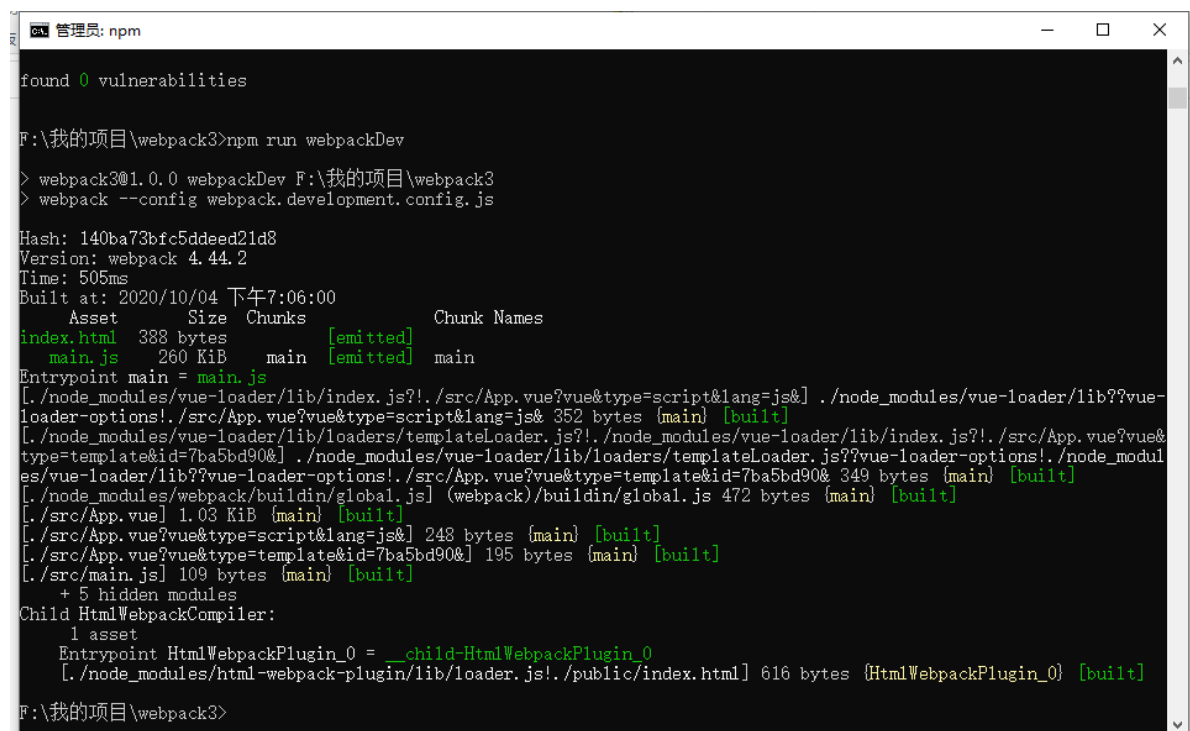
名称	类型	默认	描述
<code>title</code>	<code>{String}</code>	Webpack App	用于生成的HTML文档的标题
<code>filename</code>	<code>{String}</code>	<code>'index.html'</code>	写入HTML的文件。默认为 <code>index.html</code> 。您可以在这里指定一个子目录太（如： <code>assets/admin.html</code> ）
<code>template</code>	<code>{String}</code>	<code>''</code>	webpack 模板的相对或绝对路径。默认情况下，它将使用 <code>(src/index.ejs 如果存在)</code> 。请参阅 文档 以了解详细信息

filename是写入html，而template是模板，我们可以使用这个来进行

```
const VueLoaderPlugin = require("vue-loader/lib/plugin");
const HtmlWebpackPlugin = require("html-webpack-plugin")
const path = require("path")
module.exports = {
  entry: {
    main: "./src/main.js"
  },
  output: {
    filename: "[name].js"
  },
  plugins: [
    new VueLoaderPlugin(),
    new HtmlWebpackPlugin({
      filename: 'index.html',
      template: path.resolve(__dirname, './public/index.html'),
      title: 'blog'
    })
  ],
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: "vue-loader"
      }
    ]
  }
}
```

可能大家会发现怎么多了一个path，这个是node的路径表示方式，__dirname 代表的是当前模块路径，如果不用这个，你就要写绝对路径，所以还是直接用node的方法好一点

我们删除dist里面的东西再来运行一次命令



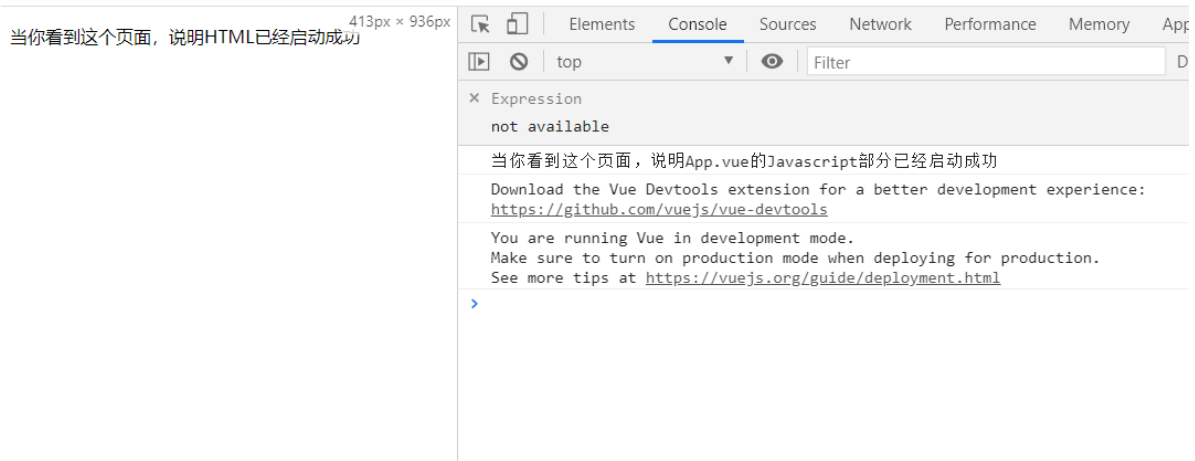
```
管理员: npm
found 0 vulnerabilities

F:\我的项目\webpack3>npm run webpackDev

> webpack3@1.0.0 webpackDev F:\我的项目\webpack3
> webpack --config webpack.development.config.js

Hash: 140ba73bfc5ddeed21d8
Version: webpack 4.44.2
Time: 505ms
Built at: 2020/10/04 下午7:06:00
    Asset      Size  Chunks             Chunk Names
index.html  388 bytes               [emitted]
main.js    260 KiB             [emitted]  main
Entrypoint main = main.js
[./node_modules/vue-loader/lib/index.js?!. /src/App.vue?vue&type=script&lang=js&] ./node_modules/vue-loader/lib??vue-loader-options!./src/App.vue?vue&type=script&lang=js& 352 bytes {main} [built]
[./node_modules/vue-loader/lib/loaders/templateLoader.js?!. /node_modules/vue-loader/lib/index.js?!. /src/App.vue?vue&type=template&id=7ba5bd90&] ./node_modules/vue-loader/lib/loaders/templateLoader.js??vue-loader-options!./node_modules/vue-loader/lib??vue-loader-options!./src/App.vue?vue&type=template&id=7ba5bd90& 349 bytes {main} [built]
[./node_modules/webpack/buildin/global.js] (webpack)/buildin/global.js 472 bytes {main} [built]
[./src/App.vue] 1.03 KiB {main} [built]
[./src/App.vue?vue&type=script&lang=js&] 248 bytes {main} [built]
[./src/App.vue?vue&type=template&id=7ba5bd90&] 195 bytes {main} [built]
[./src/main.js] 109 bytes {main} [built]
+ 5 hidden modules
Child HtmlWebpackCompiler:
  1 asset
  Entrypoint HtmlWebpackPlugin_0 = __child-HtmlWebpackPlugin_0
    [./node_modules/html-webpack-plugin/lib/loader.js!./public/index.html] 616 bytes {HtmlWebpackPlugin_0} [built]

F:\我的项目\webpack3>
```

运行成功了！

这里就差不多告一段落了，但是我们还有一个缺陷，那就是每次都要手动删除文件才能进行命令，我们来看看官网的方法，我已经弄好链接

点进去官网链接：<https://webpack.docschina.org/guides/output-management/#cleaning-up-the-dist-folder>

API 概念 配置 指南 装载机 迁移 插入

清理 /dist 文件夹

您可能已经注意到，由于遗留了之前的指南和代码示例，我们的 /dist 文件夹出现相当杂乱。webpack 将生成文件并放置在 /dist 文件夹中，但是它不会追踪某些文件是实际在项目中用到的。

通常比较推荐的做法是，在每次合并前清理 /dist 文件夹，这样只会生成用到的文件。让我们实现这个需求。

`clean-webpack-plugin` 是一个流行的清理插件，安装和配置它。

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

webpack.config.js

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
+ const { CleanWebpackPlugin } = require('clean-webpack-plugin');

module.exports = {
  entry: {
    app: './src/index.js',
    print: './src/print.js',
  },
  plugins: [
+   new CleanWebpackPlugin(),
    new HtmlWebpackPlugin({
      title: '管理输出',
    }),
  ],
  output: {
    filename: '[name].bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
};
```

现在，执行 `npm run build`，检查 /dist 文件夹。如果一切顺利，现在只会看到合并后生成的文件，而没有旧文件！

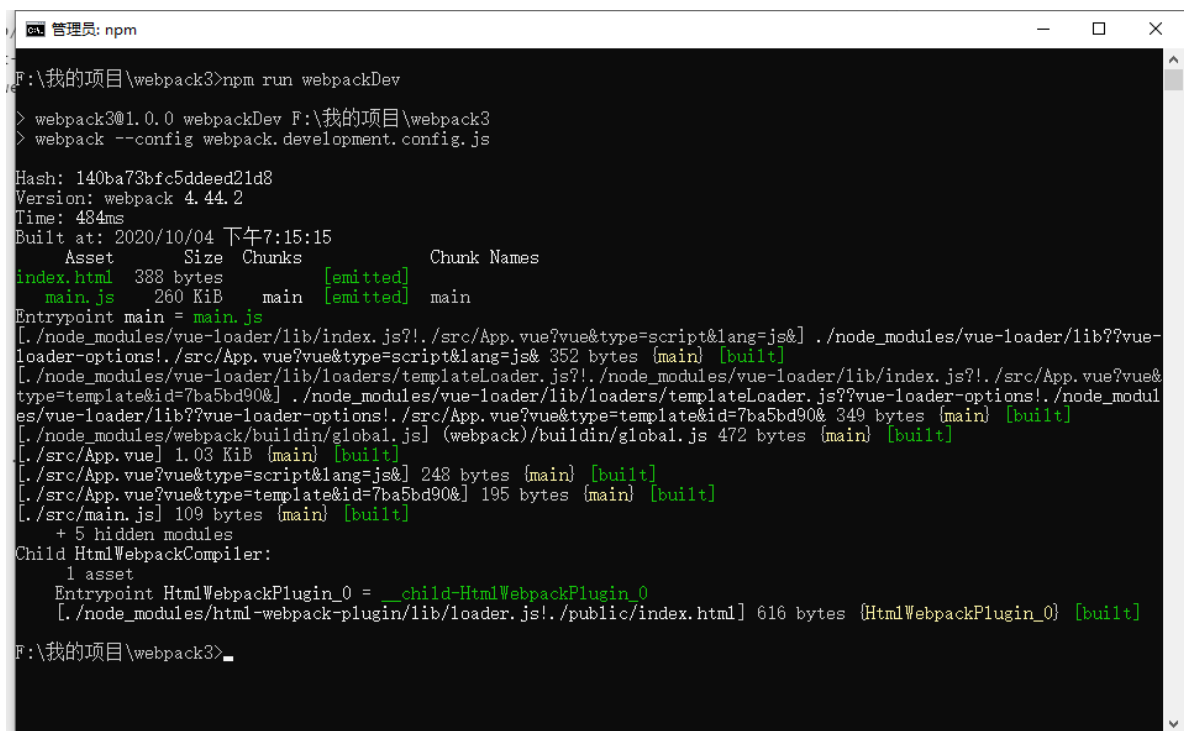
```
npm install --save-dev clean-webpack-plugin
```

安装这个插件

```
{
  "name": "webpack3",
  "version": "1.0.0",
  "description": "",
  "private": true,
  "scripts": {
    "webpackDev": "webpack --config webpack.development.config.js",
    "webpackPro": "webpack --config webpack.production.config.js"
  },
  "keywords": [],
  "author": "博丽灵梦",
  "license": "ISC",
  "devDependencies": {
    "clean-webpack-plugin": "^3.0.0",
    "html-webpack-plugin": "^4.5.0",
    "vue-loader": "^15.9.3",
    "vue-template-compiler": "^2.6.12",
    "webpack": "^4.44.2",
    "webpack-cli": "^3.3.12",
    "webpack-merge": "^5.1.4"
  },
  "dependencies": {
    "vue": "^2.6.12"
  }
}
```

```
const VueLoaderPlugin = require("vue-loader/lib/plugin");
const HtmlWebpackPlugin = require("html-webpack-plugin");
const { CleanWebpackPlugin } = require("clean-webpack-plugin");
const path = require("path");
module.exports = {
  entry: {
    main: "./src/main.js"
  },
  output: {
    filename: "[name].js"
  },
  plugins: [
    new VueLoaderPlugin(),
    new HtmlWebpackPlugin({
      filename: 'index.html',
      template: path.resolve(__dirname, './public/index.html'),
      title: 'blog'
    }),
    new CleanWebpackPlugin()
  ],
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: "vue-loader"
      }
    ]
  }
}
```

```
}  
}
```



```
F:\我的项目\webpack3>npm run webpackDev  
  
> webpack3@1.0.0 webpackDev F:\我的项目\webpack3  
> webpack --config webpack.development.config.js  
  
Hash: 140ba73bfc5ddeed21d8  
Version: webpack 4.44.2  
Time: 484ms  
Built at: 2020/10/04 下午7:15:15  


| Asset      | Size      | Chunks         | Chunk Names |
|------------|-----------|----------------|-------------|
| index.html | 388 bytes | [emitted]      |             |
| main.js    | 260 KiB   | main [emitted] | main        |

  
Entrypoint main = main.js  
[./node_modules/vue-loader/lib/index.js?!. /src/App.vue?vue&type=script&lang=js&] ./node_modules/vue-loader/lib??vue-loader-options!./src/App.vue?vue&type=script&lang=js& 352 bytes {main} [built]  
[./node_modules/vue-loader/lib/loaders/templateLoader.js?!. /node_modules/vue-loader/lib/index.js?!. /src/App.vue?vue&type=template&id=7ba5bd90&] ./node_modules/vue-loader/lib/loaders/templateLoader.js??vue-loader-options!./node_modules/vue-loader/lib??vue-loader-options!./src/App.vue?vue&type=template&id=7ba5bd90& 349 bytes {main} [built]  
[./node_modules/webpack/buildin/global.js] (webpack)/buildin/global.js 472 bytes {main} [built]  
[./src/App.vue] 1.03 KiB {main} [built]  
[./src/App.vue?vue&type=script&lang=js&] 248 bytes {main} [built]  
[./src/App.vue?vue&type=template&id=7ba5bd90&] 195 bytes {main} [built]  
[./src/main.js] 109 bytes {main} [built]  
+ 5 hidden modules  
Child HtmlWebpackCompiler:  
  1 asset  
    Entrypoint HtmlWebpackPlugin_0 = __child-HtmlWebpackPlugin_0  
      [./node_modules/html-webpack-plugin/lib/loader.js!./public/index.html] 616 bytes {HtmlWebpackPlugin_0} [built]
```

搭建vue环境就到这里结束了，后面的文章会继续完善相关的东西，还有一些css和图片什么的没做处理，我们后面继续处理。。。博主暂时学到这里

搭建本地服务器

在我们日常敲代码中，基本都是保存完，网页就更新了，用过vuecli都知道吧，挺方便的，接下来我们开始实习这个

附上官网链接：

<https://vue-loader.vuejs.org/zh/guide/hot-reload.html>

用法

当使用脚手架工具 `vue-cli` 时，热重载是开箱即用的。

当手动设置你的工程时，热重载会在你启动 `webpack-dev-server --hot` 服务时自动开启。

高阶用户可能希望移步 `vue-loader` 内部使用的 `vue-hot-reload-api` 继续查阅。

vue官网提到的，使用`webpack-dev-server --hot`

我们去webpack官网搜索，附上我找到的链接，还请阅读5分钟

<https://webpack.docschina.org/guides/development/#using-webpack-dev-server>

开发环境

本指南继续沿用 [管理输出](#) 指南中的代码示例。

如果你一直跟随之前的指南，应该对一些 webpack 基础知识有着很扎实的理解。在我们继续之前，先来看看如何设置一个开发环境，使我们的开发体验变得更轻松一些。

本指南中的工具仅用于开发环境，请不要在生产环境中使用它们！

使用 webpack-dev-server

webpack-dev-server 为你提供了一个简单的 web server，并且具有 live reloading(实时重新加载)功能。设置如下：

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

修改配置文件，告知 dev server，从什么位置查找文件：

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

开始运行这条命令吧

```
管理员: npm
Child HtmlWebpackCompiler:
  1 asset
  Entrypoint HtmlWebpackPlugin_0 = __child-HtmlWebpackPlugin_0
  [0] ./node_modules/html-webpack-plugin/lib/loader.js!./public/index.html 680 bytes {0} [built]
F:\我的项目\webpack3>npm install --save-dev webpack-dev-server
npm WARN deprecated chokidar@2.1.8: Chokidar 2 will break on node v14+. Upgrade to chokidar 3 with 15x less dependencies.
npm WARN deprecated fsevents@1.2.13: fsevents 1 will break on node v14+ and could be using insecure binaries. Upgrade to fsevents 2.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\webpack-dev-server\node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN vue-loader@15.9.3 requires a peer of css-loader* but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.3 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules\watchpack-chokidar2\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
+ webpack-dev-server@3.11.0
added 149 packages from 107 contributors and audited 650 packages in 29.759s
26 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
F:\我的项目\webpack3>
```

老规矩，编写我们的配置文件

```
{
  "name": "webpack3",
  "version": "1.0.0",
  "description": "",
  "private": true,
  "scripts": {
    "webpackDev": "webpack --config webpack.development.config.js",
```

```

    "webpackPro": "webpack --config webpack.production.config.js",
    "serve": "webpack-dev-server"
  },
  "keywords": [],
  "author": "博丽灵梦",
  "license": "ISC",
  "devDependencies": {
    "clean-webpack-plugin": "^3.0.0",
    "html-webpack-plugin": "^4.5.0",
    "vue-loader": "^15.9.3",
    "vue-template-compiler": "^2.6.12",
    "webpack": "^4.44.2",
    "webpack-cli": "^3.3.12",
    "webpack-dev-server": "^3.11.0",
    "webpack-merge": "^5.1.4"
  },
  "dependencies": {
    "vue": "^2.6.12"
  }
}

```

就简单的加个指令，来试试效果，运行命令

```
npm run serve
```

运行成功

```

i [wds]: Project is running at http://localhost:8080/
i [wds]: webpack output is served from /
i [wds]: Content not from webpack is served from F:\我的项目\webpack3
i [wdm]: Hash: 401bf09db881d492605d
Version: webpack 4.44.2
Time: 750ms
Built at: 2020/10/04 下午8:44:05
    Asset      Size  Chunks             Chunk Names
index.html  384 bytes          0 [emitted]
main.js    618 KiB          0 [emitted] main
Entrypoint main = main.js
[0] multi (webpack)-dev-server/client?http://localhost:8080 ./src/main.js 40 bytes {main} [built]
[./node_modules/ansi-html/index.js] 4.16 KiB {main} [built]
[./node_modules/html-entities/lib/index.js] 449 bytes {main} [built]
[./node_modules/vue/dist/vue.runtime.esm.js] 222 KiB {main} [built]
[./node_modules/webpack-dev-server/client/index.js?http://localhost:8080] 4.29 KiB {main} [built]
[./node_modules/webpack-dev-server/client/overlay.js] (webpack)-dev-server/client/overlay.js 3.51 KiB {main} [built]
[./node_modules/webpack-dev-server/client/socket.js] (webpack)-dev-server/client/socket.js 1.53 KiB {main} [built]
[./node_modules/webpack-dev-server/client/utils/createSocketUrl.js] (webpack)-dev-server/client/utils/createSocketUr
l.js 2.91 KiB {main} [built]
[./node_modules/webpack-dev-server/client/utils/log.js] (webpack)-dev-server/client/utils/log.js 964 bytes {main} [b
uilt]
[./node_modules/webpack-dev-server/client/utils/reloadApp.js] (webpack)-dev-server/client/utils/reloadApp.js 1.59 Ki
B {main} [built]
[./node_modules/webpack-dev-server/client/utils/sendMessage.js] (webpack)-dev-server/client/utils/sendMessage.js 402
bytes {main} [built]
[./node_modules/webpack-dev-server/node_modules/strip-ansi/index.js] (webpack)-dev-server/node_modules/strip-ansi/in
dex.js 161 bytes {main} [built]
[./node_modules/webpack-hot-sync /\.\/log$] (webpack)/hot sync nonrecursive /\.\/log$ 170 bytes {main} [built]
[./src/App.vue] 1.03 KiB {main} [built]
[./src/main.js] 109 bytes {main} [built]
+ 28 hidden modules

```

打开浏览器，把localhost:8080输入进去

成功看到我们的网页。

但是好像有点麻烦，好像别人的命令都是直接打开浏览器的？

我们也要这样的效果

我们去官网找链接，前面我们已经给过链接了，这里再放一次

<https://webpack.docschina.org/configuration/dev-server/#devserver>

devServer.openPage

string [string]

指定打开浏览器时要浏览的页面。

webpack.config.js

```
module.exports = {  
  //...  
  devServer: {  
    openPage: '/different/page'  
  }  
};
```

通过命令行使用

```
webpack-dev-server --open-page "/different/page"
```

如果希望指定多个页面在浏览器中打开。

webpack.config.js

```
module.exports = {  
  //...  
  devServer: {  
    openPage: ['/different/page1', '/different/page2']  
  }  
};
```

通过命令行使用

```
webpack-dev-server --open-page "/different/page1,/different/page2"
```

```
{
  "name": "webpack3",
  "version": "1.0.0",
  "description": "",
  "private": true,
  "scripts": {
    "webpackDev": "webpack --config webpack.development.config.js",
    "webpackPro": "webpack --config webpack.production.config.js",
    "serve": "webpack-dev-server --open"
  },
  "keywords": [],
  "author": "博丽灵梦",
  "license": "ISC",
  "devDependencies": {
    "clean-webpack-plugin": "^3.0.0",
    "html-webpack-plugin": "^4.5.0",
    "vue-loader": "^15.9.3",
    "vue-template-compiler": "^2.6.12",
    "webpack": "^4.44.2",
    "webpack-cli": "^3.3.12",
    "webpack-dev-server": "^3.11.0",
    "webpack-merge": "^5.1.4"
  },
  "dependencies": {
    "vue": "^2.6.12"
  }
}
```

我们在serve后面加个 --open命令

```
"serve": "webpack-dev-server --open"
```

再次运行命令行，发现已经可以自动帮我们打开浏览器

或者在config文件里面写

```
package.json × App.vue × main.js × webpack.development.config.js × webpack.production.config.js × webpack.c
const path = require("path");
module.exports = {
  entry: {
    main: "./src/main.js"
  },
  output: {
    filename: "[name].js"
  },
  plugins: [
    new VueLoaderPlugin(),
    new HtmlWebpackPlugin( options: {
      filename: 'index.html',
      template: path.resolve(__dirname, './public/index.html'),
      title: 'blog'
    }),
    new CleanWebpackPlugin()
  ],
  module: {
    rules: [
      {
        test: /\.vue$/,
        exclude: /node_modules/,
        loader: "vue-loader"
      }
    ]
  },
  devServer: {
    open: true
  }
}
```

因为vue-loader是默认开启热加载的，所以我们只需要安装devServe就可以直接有了。

CSS样式和CSS预编译

写网站肯定少不了样式，先在我们的vue文件写一点css样式，试试效果，看看报什么错


```
npm
✖ [wdm]: Hash: f937281d8b031d567059
Version: webpack 4.44.2
Time: 44ms
Built at: 2020/10/04 下午9:07:30
    Asset      Size  Chunks             Chunk Names
  main.js    621 KiB       0  [emitted]  main
Entrypoint main = main.js
[./node_modules/vue-loader/lib/index.js?!.src/App.vue?vue&type=script&lang=js&] ./node_modules/vue-loader/lib??vue-loader-options!./src/App.vue?vue&type=script&lang=js& 367 bytes {main} [built]
[./node_modules/vue-loader/lib/index.js?!.src/App.vue?vue&type=style&index=0&lang=css&] ./node_modules/vue-loader/lib??vue-loader-options!./src/App.vue?vue&type=style&index=0&lang=css& 286 bytes {main} [built] [failed] [1 error]
[./node_modules/vue-loader/lib/loaders/templateLoader.js?!.src/App.vue?vue&type=template&id=7ba5bd90&] ./node_modules/vue-loader/lib/loaders/templateLoader.js??vue-loader-options!./src/App.vue?vue&type=template&id=7ba5bd90& 349 bytes {main} [built]
[./src/App.vue] 1.09 KiB {main} [built]
[./src/App.vue?vue&type=style&index=0&lang=css&] 264 bytes {main} [built]
+ 40 hidden modules

ERROR in ./src/App.vue?vue&type=style&index=0&lang=css& (.node_modules/vue-loader/lib??vue-loader-options!./src/App
..vue?vue&type=style&index=0&lang=css&) 27:0
Module parse failed: Unexpected character '#' (27:0)
File was processed with these loaders:
 * ./node_modules/vue-loader/lib/index.js
You may need an additional loader to handle the result of these loaders.

> #app{
  font-size: 20px;
  color: blue;
}
@ ./src/App.vue?vue&type=style&index=0&lang=css& 1:0-123 1:139-142 1:144-264 1:144-264
@ ./src/App.vue
@ ./src/main.js
i [wdm]: Failed to compile.
```

说我们少了loader，无法识别

那我们就来安装

附上链接<https://www.npmjs.com/package/css-loader>

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

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

```
const VueLoaderPlugin = require("vue-loader/lib/plugin");
const HtmlWebpackPlugin = require("html-webpack-plugin");
const { CleanWebpackPlugin } = require("clean-webpack-plugin");
const path = require("path");
module.exports = {
  entry: {
    main: "./src/main.js"
  },
  output: {
    filename: "[name].js"
  },
  module: {
    rules: [
      {
        test: /\.vue$/,
        exclude: /node_modules/,
        loader: "vue-loader"
      },
      {
        test: /\.css$/,
        use: [
          'vue-style-loader',
          {
            loader: 'css-loader',
            options: {
```

```

        esModule: false,
      }
    }
  ],
}
},
devServer: {
  open: true
},
plugins: [
  new VueLoaderPlugin(),
  new HtmlWebpackPlugin({
    filename: 'index.html',
    template: path.resolve(__dirname, './public/index.html'),
    title: 'blog'
  }),
  new CleanWebpackPlugin()
]
}

```

大家可能发现了，为什么多了个esModule: false

这是因为，最新版本的cssloader它默认是true，如果不改，他的css就不会显示了，坑了我一小时。。

运行服务器命令npm run serve

成功执行

```

C:\管理: npm
[./node_modules/html-entities/lib/index.js] 449 bytes {main} [built]
[./node_modules/vue/dist/vue.runtime.esm.js] 222 KiB {main} [built]
[./node_modules/webpack-dev-server/client/index.js?http://localhost:8080] (webpack)-dev-server/client?http://localhost:8080 4.29 KiB {main} [built]
[./node_modules/webpack-dev-server/client/overlay.js] (webpack)-dev-server/client/overlay.js 3.51 KiB {main} [built]
[./node_modules/webpack-dev-server/client/socket.js] (webpack)-dev-server/client/socket.js 1.53 KiB {main} [built]
[./node_modules/webpack-dev-server/client/utils/createSocketUrl.js] (webpack)-dev-server/client/utils/createSocketUrl.js 2.91 KiB {main} [built]
[./node_modules/webpack-dev-server/client/utils/log.js] (webpack)-dev-server/client/utils/log.js 964 bytes {main} [built]
[./node_modules/webpack-dev-server/client/utils/reloadApp.js] (webpack)-dev-server/client/utils/reloadApp.js 1.59 KiB {main} [built]
[./node_modules/webpack-dev-server/client/utils/sendMessage.js] (webpack)-dev-server/client/utils/sendMessage.js 402 bytes {main} [built]
[./node_modules/webpack-dev-server/node_modules/strip-ansi/index.js] (webpack)-dev-server/node_modules/strip-ansi/index.js 161 bytes {main} [built]
[./node_modules/webpack/hot sync ^\\.\\.\\/log$] (webpack)/hot sync nonrecursive ^\\.\\.\\/log$ 170 bytes {main} [built]
[./src/App.vue] 1.09 KiB {main} [built]
[./src/main.js] 109 bytes {main} [built]
+ 41 hidden modules
Child HtmlWebpackCompiler:
  1 asset
Entrypoint HtmlWebpackPlugin_0 = __child-HtmlWebpackPlugin_0
[./node_modules/html-webpack-plugin/lib/loader.js!./public/index.html] 680 bytes {HtmlWebpackPlugin_0} [built]
[webpack]: Compiled successfully.
终止批处理操作吗(Y/N)?
C
F:\我的项目\webpack3>npm install --save-dev css-loader

```

```
Elements Console vue Sources Network >>
<meta name="viewport" content="width=device-width, user-scalable=no,
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>blog</title>
... <style type="text/css">div{
    font-size: 20px;
    color: chocolate;
}</style> == $0
<style type="text/css">
</style>
▶ <style type="text/css">...</style>
</head>
▼ <body>
▶ <div id="app">...</div>
<script src="main.js"></script>
```

已经可以正常显示了

但是貌似并没有打包，我们看看有没有打包的方案，记住，我们优先看vue官网给的推荐，最后在选择webpack给的推荐，这样子才能保证我们的vue项目能够正常运行。不然满满坑。。。好了，花5分钟点开链接快速过下文档吧

附上链接：

<https://vue-loader.vuejs.org/zh/guide/extract-css.html#webpack-4>

<https://github.com/webpack-contrib/mini-css-extract-plugin>

<https://github.com/NMFR/optimize-css-assets-webpack-plugin>

<https://webpack.js.org/plugins/mini-css-extract-plugin/#minimizing-for-production>

webpack 4

```
npm install -D mini-css-extract-plugin
```

sh

```
// webpack.config.js
var MiniCssExtractPlugin = require('mini-css-extract-plugin')

module.exports = {
  // 其它选项...
  module: {
    rules: [
      // ... 忽略其它规则
      {
        test: /\.css$/,
        use: [
          process.env.NODE_ENV !== 'production'
            ? 'vue-style-loader'
            : MiniCssExtractPlugin.loader,
          'css-loader'
        ]
      }
    ],
  },
  plugins: [
    // ... 忽略 vue-loader 插件
    new MiniCssExtractPlugin({
      filename: 'style.css'
    })
  ]
}
```

js

你还可以查阅 [mini-css-extract-plugin 文档](#)。

减少生产

要缩小输出，可以使用像optimize-css-assets-webpack-plugin这样的插件。设置优化。minimizer覆盖了webpack提供的默认值，所以请确保指定一个JS minimizer:

webpack.config.js

```
const TerserJSPlugin = require('terser-webpack-plugin');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const OptimizeCSSAssetsPlugin = require('optimize-css-assets-webpack-plugin');

module.exports = {
  optimization: {
    minimizer: [new TerserJSPlugin({}), new OptimizeCSSAssetsPlugin({})],
  },
  plugins: [
    new MiniCssExtractPlugin({
      filename: '[name].css',
      chunkFilename: '[id].css',
    }),
  ],
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [MiniCssExtractPlugin.loader, 'css-loader'],
      },
    ],
  },
};
```

☒ 原图模式 原文 检测到英文

```
npm install -D mini-css-extract-plugin
```

```
npm install --save-dev optimize-css-assets-webpack-plugin
```

安装这两个东西

首先说明下，为什么安装这两个

mini-css-extract-plugin：这个是用来提取css的

optimize-css-assets-webpack-plugin：这个是用来压缩css的

只能用于生产环境!!! 切记

不支持热加载

现在我重新贴下我的代码，不然有人可能看到这里看懂了。。。

```
// webpack.development.config.js
const { merge } = require("webpack-merge");
const common = require("./webpack.config.js");
module.exports = merge(common, {
  mode: "development",
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          "vue-style-loader",
          {
            loader: 'css-loader',
            options: {
              esModule: false,
            }
          }
        ]
      }
    ]
  }
})
```

```
// webpack.production.config.js
const common = require("./webpack.config.js");
const { merge } = require("webpack-merge");
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const OptimizeCssAssetsPlugin = require("optimize-css-assets-webpack-plugin");
module.exports = merge(common, {
  mode: "production",
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          MiniCssExtractPlugin.loader,
          {
            loader: 'css-loader',
            options: {
              esModule: false
            }
          }
        ]
      }
    ]
  }
})
```

```

    }
  ]
},
plugins: [
  new MiniCssExtractPlugin({
    filename: "style.css"
  }),
  new OptimizeCssAssetsPlugin()
]
})

```

```

// webpack.config.js
const VueLoaderPlugin = require("vue-loader/lib/plugin");
const HtmlWebpackPlugin = require("html-webpack-plugin");
const { CleanWebpackPlugin } = require("clean-webpack-plugin");
const path = require("path");
module.exports = {
  entry: {
    main: "./src/main.js"
  },
  output: {
    filename: "[name].js"
  },
  module: {
    rules: [
      {
        test: /\.vue$/,
        exclude: /node_modules/,
        loader: "vue-loader"
      }
    ],
  },
  devServer: {
    open: true
  },
  plugins: [
    new VueLoaderPlugin(),
    new HtmlWebpackPlugin({
      filename: 'index.html',
      template: path.resolve(__dirname, './public/index.html'),
      title: 'blog'
    }),
    new CleanWebpackPlugin()
  ]
}

```

运行我们的3个命令，看看都有什么区别

```

npm run serve
npm run webpackDev
npm run webpackPro

```

仔细的话，你会发现，生产环境下的是引入的style.css，

开发环境则是用的style标签写的css

接下来，我们开始使用预处理器，我们这里用less或者postcss

首先，先安装less，一步一步来

附上链接: <https://vue-loader.vuejs.org/zh/guide/pre-processors.html#less>

Less

```
npm install -D less less-loader
```

sh

```
// webpack.config.js -> module.rules
{
  test: /\.less$/,
  use: [
    'vue-style-loader',
    'css-loader',
    'less-loader'
  ]
}
```

js

```
npm install -D less less-loader
```

```
// webpack.production.config.js
module: {
  rules: [
    {
      test: /\..(css|less)$/,
      use: [
        MiniCssExtractPlugin.loader,
        {
          loader: 'css-loader',
          options: {
            esModule: false
          }
        },
        "less-loader"
      ]
    }
  ]
},
```

```
// webpack.development.config.js
module: {
  rules: [
    {
      test: /\..(css|less)$/,
      use: [
        "vue-style-loader",
        {
          loader: 'css-loader',
          options: {
            esModule: false,
          }
        },
      ],
    }
  ]
},
```

```

    "less-loader"
  ]
}
]
}

```

运行3个命令

```

npm run serve
npm run webpackDev
npm run webpackPro

```

分别看各自的区别

就是支持less语法了



现在来进行postcss

附上链接:

<https://vue-loader.vuejs.org/zh/guide/pre-processors.html#postcss>

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

中文文档: <https://github.com/postcss/postcss/blob/master/docs/README-cn.md>

```

npm install -D postcss-loader
npm install -D autoprefixer 如果版本太新, 可以回退版本

```


PostCSS

TIP

Vue Loader v15 不再默认应用 PostCSS 变换。你需要通过 `postcss-loader` 使用 PostCSS。

```
npm install -D postcss-loader
```

```
// webpack.config.js -> module.rules
{
  test: /\.css$/,
  use: [
    'vue-style-loader',
    {
      loader: 'css-loader',
      options: { importLoaders: 1 }
    },
    'postcss-loader'
  ]
}
```

PostCSS 的配置可以通过 `postcss.config.js` 或 `postcss-loader` 选项来完成。其更多细节请查阅 [postcss-loader 文档](#)。

`postcss-loader` 也可以和上述其它预处理器结合使用。

提前使用先进的 CSS 特性

- `autoprefixer` 添加了 vendor 浏览器前缀，它使用 Can I Use 上面的数据。
- `postcss-preset-env` 允许你使用未来的 CSS 特性。

啥都别说了，先安装再说。。。博主有点累了。。

```

rules: [
  {
    test: /\.css$/,
    use: [
      MiniCssExtractPlugin.loader,
      {
        loader: 'css-loader',
        options: {
          esModule: false
        }
      },
      "less-loader",
      "postcss-loader"
    ]
  }
],
plugins: [

```

基本的是没变的，只需要在生产环境下加就行了，开发环境不需要这个，因为比较耗时间，如果代码量大了，会影响你的编译。。。

```

// webpack.production.config.js
const common = require("../webpack.config.js");
const autoprefixer = require('autoprefixer');
const { merge } = require("webpack-merge");
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const OptimizeCssAssetsPlugin = require("optimize-css-assets-webpack-plugin");
module.exports = merge(common, {
  mode: "production",
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          MiniCssExtractPlugin.loader,
          {
            loader: 'css-loader',
            options: {
              esModule: false
            }
          },
          "less-loader",
          "postcss-loader"
        ]
      }
    ]
  },
  plugins: [
    new MiniCssExtractPlugin({
      filename: "[name].css"
    }),

```

```
    new OptimizeCssAssetsPlugin()
  ]
})
```

写完postcss，接下来就是autoprefixer，这个是给css标签加前缀的

为此，我们要在创建一个配置文件postcss.config.js

```
//postcss.config.js
module.exports = {
  plugins: {
    // 兼容浏览器，添加前缀
    'autoprefixer': {
      overrideBrowserslist: [
        "last 10 versions",
      ],
      grid: true
    }
  }
}
```

不会写？怕写错？没关系！咱们找文档

🔗 提前使用先进的 CSS 特性

`autoprefixer` 添加了 vendor 浏览器前缀，它使用 Can I Use 上面的数据。

- `postcss-preset-env` 允许你使用未来的 CSS 特性。

点进去就是文档了，文档就是权威，不要看别人博客就复制粘贴，不然等某些东西升级之后，一堆bug够你受的，所以找文档才是正确的

autoprefixer文档: <https://github.com/postcss/autoprefixer>

配置文档: <https://github.com/browserslist/browserslist#queries>

您可以通过查询（不区分大小写）指定浏览器和Node.js版本：

- defaults : Browserslist的默认浏览器 (> 0.5%, last 2 versions, Firefox ESR, not dead) 。
- > 5% : 通过全局使用情况统计信息选择的浏览器版本。 >= , < 和 <= 工作过。
 - > 5% in US : 使用美国使用情况统计信息。它接受两个字母的国家/地区代码。
 - > 5% in alt-AS : 使用亚洲地区使用情况统计信息。有关所有区域代码的列表, 请参见 [caniuse-lite/data/regions](#) 。
 - > 5% in my stats : 使用自定义用法数据。
 - > 5% in browserslist-config-mycompany stats : 使用 来自的自定义使用情况数据 browserslist-config-mycompany/browserslist-stats.json 。
 - cover 99.5% : 提供覆盖率的最受欢迎的浏览器。
 - cover 99.5% in US : 与上述相同, 但国家/地区代码由两个字母组成。
 - cover 99.5% in my stats : 使用自定义用法数据。
- dead : 24个月内没有官方支持或更新的浏览器。现在是 IE 10 , IE_Mob 11 , BlackBerry 10 , BlackBerry 7 , Samsung 4 和 OperaMobile 12.1 。
- last 2 versions : 每个浏览器的最后2个版本。
 - last 2 Chrome versions : 最近2个版本的Chrome浏览器。
 - last 2 major versions 或 last 2 iOS major versions : 最近2个主要版本的所有次要/补丁版本。
- node 10 和 node 10.4 : 选择最新的Node.js 10.x.x 或 10.4.x 版本。
 - current node : Browserslist现在使用的Node.js版本。
 - maintained node versions : 所有Node.js版本, 仍由 Node.js Foundation维护。
- iOS 7 : 直接使用iOS浏览器版本7。
 - Firefox > 20 : Firefox的版本高于20 >= , < 并且 <= 也可以使用。它也可以与Node.js一起使用。
 - ie 6-8 : 选择一个包含范围的版本。
 - Firefox ESR : 最新的[Firefox ESR]版本。
 - PhantomJS 2.1 和 PhantomJS 1.9 : 选择类似于PhantomJS运行时的Safari版本。
- extends browserslist-config-mycompany : 从 browserslist-config-mycompany npm包中查询。
- supports es6-module : 支持特定功能的浏览器。 es6-module 这是“我可以使用的”页面 feat 的URL上的参数。有关所有可用功能的列表, 请参见 [caniuse-lite/data/features](#) 。
- since 2015 或 last 2 years : 自2015年以来发布的所有版本 (since 2015-03 以及 since 2015-03-10) 。
- unreleased versions 或 unreleased Chrome versions : Alpha和Beta版本。
- not ie <= 8 : 排除先前查询选择的浏览器。

They can RI RI RIRI RIRI I YEAH

博主用的是红色圈起来那个，浏览器最后10个版本

Babel 是一个 JavaScript 编译器

链接: <https://www.babeljs.cn/docs/babel-preset-env>

英文看不懂，就用谷歌的翻译，鼠标右键

```
npm install --save-dev @babel/preset-env
```

在这之前，相信你也看过其他博主的文档，装一个babel，还有其他的东西要装，其实也是相当的繁琐，所以，这里一定要看官网，给出结果，看看是否真的需要这么多东西

首先，我们只需要ES6+语法转换成ES5，所以去官网看看，到底需要啥，找到官网一个链接

<https://www.babeljs.cn/docs/usage>

```
npm install --save-dev @babel/core @babel/cli @babel/preset-env
npm install --save @babel/polyfill
```

总览

本指南将向您展示如何将使用ES2015 +语法的JavaScript应用程序代码编译为可在当前浏览器中使用的代码。这将涉及转换新语法和填充缺少的功能。

整个设置过程包括：

1. 运行以下命令以安装软件包：

Shell

Copy

```
npm install --save-dev @babel/core @babel/cli @babel/preset-env
npm install --save @babel/polyfill
```

在安装之前，我们先了解下每个东西都是啥，避免不必要的安装

@babel/core babel的核心模块

@babel/cli 官方说明:是允许您从终端使用babel的工具,个人感觉不太必要，待会我们不要他

@babel/preset-env 预设命令，可以看到下图，简单来说，就是代码中如果遇到更高级的语法，可以自由配置，然后进行转换

这是一个好的开始！但是我们的代码中还有其他ES2015 +功能需要转换。我们可以使用一个“预设”，它只是一组预定的插件，而不是——添加所有我们想要的插件。

就像插件一样，您也可以创建自己的预设来共享所需的插件的任意组合。对于我们这里的用例，有一个很好的预设名为 `env`。

Shell

复制

```
npm install --save-dev @babel/preset-env

./node_modules/.bin/babel src --out-dir lib --presets=@babel/env
```

没有任何配置，此预设将包括所有插件以支持现代JavaScript（ES2015，ES2016等）。但是预设也可以选择。而不是从终端传递cli和预设选项，让我们看一下传递选项的另一种方式：配置文件。

@babel/polyfill 官网balabala...反正就是用来体验ES6+新特性的代码，兼容浏览器，

安装ES6+，ES6+的兼容性很差，需要安装ES6+的兼容性，可以安装[@babel/polyfill](#)。

由于我们正在构建应用程序，因此可以直接安装 @babel/polyfill：

Shell

复制

```
npm install --save @babel/polyfill
```

请注意该 `--save` 选项，而不是 `--save-dev` 因为它是需要在源代码之前运行的polyfill。

现在对我们来说幸运的是，我们正在使用 `env` 预设，该预设具有一个 `"useBuiltIns"` 选项，当设置 `"usage"` 为 `时`，它将实际上应用上面提到的最后一个优化，其中您仅包含所需的polyfills。使用此新选项，配置将如下更改：

那现在开始安装吧

最后，在安装一个**babel-loader**

```
npm i babel-loader -D
```

博主现在所安装的清单

```

{
  "name": "webpack3",
  "version": "1.0.0",
  "description": "",
  "private": true,
  "scripts": {
    "webpackDev": "webpack --config webpack.development.config.js",
    "webpackPro": "webpack --config webpack.production.config.js",
    "serve": "webpack-dev-server --config webpack.development.config.js"
  },
  "keywords": [],
  "author": "博丽灵梦",
  "license": "ISC",
  "devDependencies": {
    "@babel/core": "^7.11.6",
    "@babel/preset-env": "^7.11.5",
    "autoprefixer": "^9.8.6",
    "babel-loader": "^8.1.0",
    "clean-webpack-plugin": "^3.0.0",
    "css-loader": "^4.3.0",
    "html-webpack-plugin": "^4.5.0",
    "less": "^3.12.2",
    "less-loader": "^7.0.1",
    "mini-css-extract-plugin": "^0.11.3",
    "optimize-css-assets-webpack-plugin": "^5.0.4",
    "postcss-loader": "^4.0.3",
    "vue-loader": "^15.9.3",
    "vue-style-loader": "^4.1.2",
    "vue-template-compiler": "^2.6.12",
    "webpack": "^4.44.2",
    "webpack-cli": "^3.3.12",
    "webpack-dev-server": "^3.11.0",
    "webpack-merge": "^5.1.4"
  },
  "dependencies": {
    "@babel/polyfill": "^7.11.5",
    "vue": "^2.6.12"
  }
}

```

安装完之后，开始编写我们的配置文件吧

```

// webpack.config.js
const VueLoaderPlugin = require("vue-loader/lib/plugin");
const HtmlWebpackPlugin = require("html-webpack-plugin");

const { CleanWebpackPlugin } = require("clean-webpack-plugin");
const path = require("path");
module.exports = {
  entry: {
    main: ["@babel/polyfill", "./src/main.js"]
  },
  output: {
    filename: "[name].js"
  },
  module: {
    rules: [

```

```

    {
      test: /\.vue$/,
      exclude: /node_modules/,
      loader: "vue-loader"
    },
    {
      test: /\.js$/,
      include: path.resolve(__dirname, "src"),
      use: {
        loader: "babel-loader",
        options: {
          presets: ['@babel/preset-env']
        }
      }
    }
  ]
},
devServer: {
  open: true
},
plugins: [
  new VueLoaderPlugin(),
  new HtmlWebpackPlugin({
    filename: 'index.html',
    template: path.resolve(__dirname, './public/index.html'),
    title: 'blog'
  }),
  new CleanWebpackPlugin()
]
}

```

入口开始，转化我们的代码

`main: ["@babel/polyfill", "./src/main.js"]` 这个看实际应用，若有需要可以按需引入，避免js包体积变大，原版打包后，只有70kb，加入这个会变成150kb，emmm

引入loader

```

{
  test: /\.js$/,
  include: path.resolve(__dirname, "src"),
  use: {
    loader: "babel-loader",
    options: {
      presets: ['@babel/preset-env']
    }
  }
}

```

写完之后，在我们的app.vue编写es6的代码

然后开始打包

在我们的打包文件查找我们的代码，看看有没有被转换

正常都是应该被转化了。

babel就到这里结束，将来如果在写博客的时候，遇到了问题，我会第一时间更新

eslint

这玩意，大家都不陌生了

万恶的bug

。 。 。

<https://vue-loader.vuejs.org/zh/guide/linting.html#eslint>

另一个选项是使用 [eslint-loader](#) 那么你的 `*.vue` 文件在开发过程中每次保存的时候就会自动进行代码校验：

```
npm install -D eslint eslint-loader
```

请确保它是作为一个 pre-loader 运用的：

```
// webpack.config.js
module.exports = {
  // ... 其它选项
  module: {
    rules: [
      {
        enforce: 'pre',
        test: /\.js|vue$/,
        loader: 'eslint-loader',
        exclude: /node_modules/
      }
    ]
  }
}
```

```
npm install -D eslint eslint-webpack-plugin
```

安装完之后，在命令行输入 `npx eslint --init`

根据提示初始化 `eslint`

安装 `eslint-plugin-vue`

然后得到进行配置编写

我先把代码贴了，大家跟上步伐

```
//package.json
{
  "name": "webpack3",
  "version": "1.0.0",
  "description": "",
  "private": true,
  "scripts": {
    "webpackDev": "webpack --config webpack.development.config.js",
    "webpackPro": "webpack --report --config webpack.production.config.js",
    "serve": "webpack-dev-server --config webpack.development.config.js"
  },
  "keywords": [],
```



```

"author": "博丽灵梦",
"license": "ISC",
"devDependencies": {
  "@babel/core": "^7.11.6",
  "@babel/preset-env": "^7.11.5",
  "autoprefixer": "^9.8.6",
  "babel-loader": "^8.1.0",
  "clean-webpack-plugin": "^3.0.0",
  "css-loader": "^4.3.0",
  "eslint": "^7.10.0",
  "eslint-plugin-vue": "^7.0.1",
  "eslint-webpack-plugin": "^2.1.0",
  "html-webpack-plugin": "^4.5.0",
  "less": "^3.12.2",
  "less-loader": "^7.0.1",
  "mini-css-extract-plugin": "^0.11.3",
  "optimize-css-assets-webpack-plugin": "^5.0.4",
  "postcss-loader": "^4.0.3",
  "vue-loader": "^15.9.3",
  "vue-style-loader": "^4.1.2",
  "vue-template-compiler": "^2.6.12",
  "webpack": "^4.44.2",
  "webpack-cli": "^3.3.12",
  "webpack-dev-server": "^3.11.0",
  "webpack-merge": "^5.1.4"
},
"dependencies": {
  "@babel/polyfill": "^7.11.5",
  "vue": "^2.6.12"
}
}

```

```

// webpack.config.js
const VueLoaderPlugin = require("vue-loader/lib/plugin");
const HtmlWebpackPlugin = require("html-webpack-plugin");
const { CleanWebpackPlugin } = require("clean-webpack-plugin");
const path = require("path");
module.exports = {
  entry: {
    main: ["@babel/polyfill", "./src/main.js"]
  },
  output: {
    filename: "[name].js"
  },
  module: {
    rules: [
      {
        test: /\.vue$/,
        exclude: /node_modules/,
        loader: "vue-loader"
      },
      {
        test: /\.js$/,
        include: path.resolve(__dirname, "src"),
        exclude: /node_modules/,
        use: {

```

```

        loader: "babel-loader",
        options: {
          presets: ['@babel/preset-env']
        }
      }
    ]
  },
  devServer: {
    open: true,
    overlay: true
  },
  plugins: [
    new VueLoaderPlugin(),
    new HtmlWebpackPlugin({
      filename: 'index.html',
      template: path.resolve(__dirname, './public/index.html'),
      title: 'blog'
    }),
    new CleanWebpackPlugin()
  ]
}

```

```

// webpack.development.config.js
const { merge } = require("webpack-merge");
const common = require("./webpack.config.js");
const ESLintPlugin = require('eslint-webpack-plugin');
const path = require("path")
module.exports = merge(common,{
  mode: "development",
  module: {
    rules: [
      {
        test: /\.css|less$/,
        use: [
          "vue-style-loader",
          {
            loader: 'css-loader',
            options: {
              esModule: false
            }
          },
          "less-loader"
        ]
      }
    ]
  },
  plugins: [
    new ESLintPlugin({
      context : path.resolve(__dirname,"./src"),
      extensions: ["js","vue"]
    })
  ]
})

```

```
// webpack.production.config.js
const common = require("../webpack.config.js");
const { merge } = require("webpack-merge");
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const OptimizeCssAssetsPlugin = require("optimize-css-assets-webpack-plugin");
module.exports = merge(common, {
  mode: "production",
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          MiniCssExtractPlugin.loader,
          {
            loader: 'css-loader',
            options: {
              esModule: false
            }
          },
          "less-loader",
          "postcss-loader"
        ]
      }
    ]
  },
  plugins: [
    new MiniCssExtractPlugin({
      filename: "[name].css"
    }),
    new OptimizeCssAssetsPlugin()
  ]
})
```

```
// .eslintrc.js
module.exports = {
  "env": {
    "browser": true,
    "es2021": true
  },
  "extends": [
    "eslint:recommended",
    "plugin:vue/essential"
  ],
  "parserOptions": {
    "ecmaVersion": 12,
    "sourceType": "module"
  },
  "plugins": [
    "vue"
  ],
  "rules": {
    "no-undef": "off",
    "no-alert": "error" //这是测试用的，写上这个，只要你写alert，就会报错
  }
};
```

写到这里，不知道大家还记得配置怎么写吗，下面的链接，是他的配置清单，不知道为什么这么写的，可以参考下面的链接，比如上面的代码

```
new ESLintPlugin({
  context : path.resolve(__dirname, "./src"),
  extensions: ["js", "vue"]
})
```

这个配置应该怎么写呢？参考<https://webpack.js.org/plugins/eslint-webpack-plugin/>

下面也给了链接，里面的options，官网有给详细的作用，如果看不懂英文，可以用谷歌翻译

比如context就是文件目录

extensions就是指定检查文件类型

至于`/.eslintrc.js`这里面的，只需要初始化就行了，根据提示来弄，然后自己只需要写规则，也就是上面提到的`npx eslint --init`

写完之后，自己在vue里面测试，只要报错就说明安装成功了

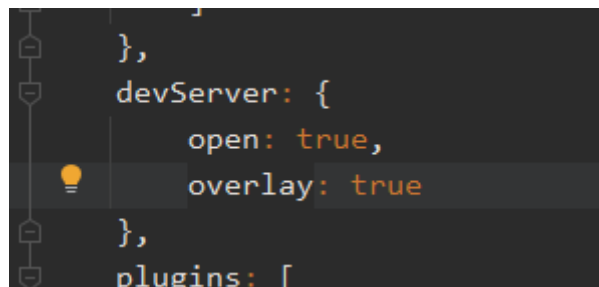
基础： <https://eslint.org/docs/rules/>

Vue： <https://eslint.vuejs.org/rules/>

插件： <https://webpack.js.org/plugins/eslint-webpack-plugin/>

仔细地话，你们会发现vue自带的那个全屏报错

我们只需要在devServer里面增加代码即可



```
},
devServer: {
  open: true,
  overlay: true
},
plugins: [
```

再次附上链接，前面有发过的： <https://webpack.docschina.org/configuration/dev-server/#devserver>

区分环境

用法

只需在配置对象中提供 `mode` 选项：

```
module.exports = {  
  mode: 'development'  
};
```

或者从 CLI 参数中传递：

```
webpack --mode=development
```

支持以下字符串值：

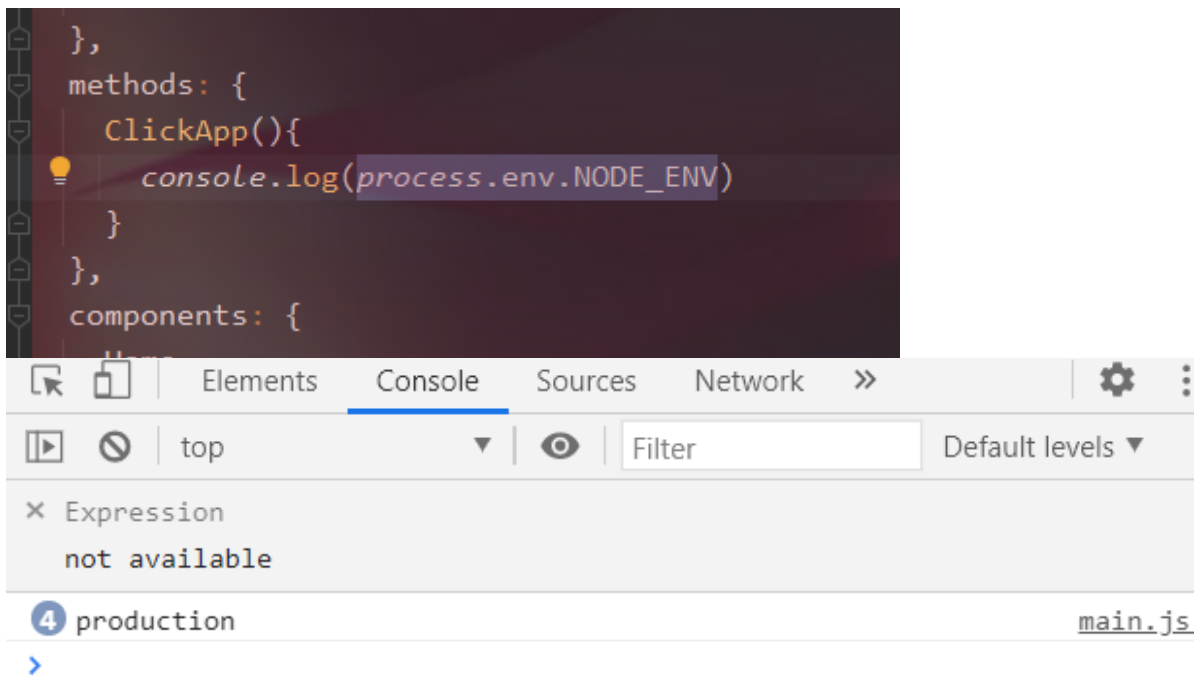
选项	描述
<code>development</code>	会将 <code>DefinePlugin</code> 中 <code>process.env.NODE_ENV</code> 的值设置为 <code>development</code> 。启用 <code>NamedChunksPlugin</code> 和 <code>NamedModulesPlugin</code> 。
<code>production</code>	会将 <code>DefinePlugin</code> 中 <code>process.env.NODE_ENV</code> 的值设置为 <code>production</code> 。启用 <code>FlagDependencyUsagePlugin</code> ， <code>FlagIncludedChunksPlugin</code> ， <code>ModuleConcatenationPlugin</code> ， <code>NoEmitOnErrorsPlugin</code> ， <code>OccurrenceOrderPlugin</code> ， <code>SideEffectsFlagPlugin</code> 和 <code>TerserPlugin</code> 。
<code>none</code>	不使用任何默认优化选项

如果没有设置，webpack 会给 `mode` 的默认值设置为 `production`。

链接：<https://webpack.docschina.org/configuration/mode/#usage>

看到有颜色的字体了吗？

这个你可以在代码中获取

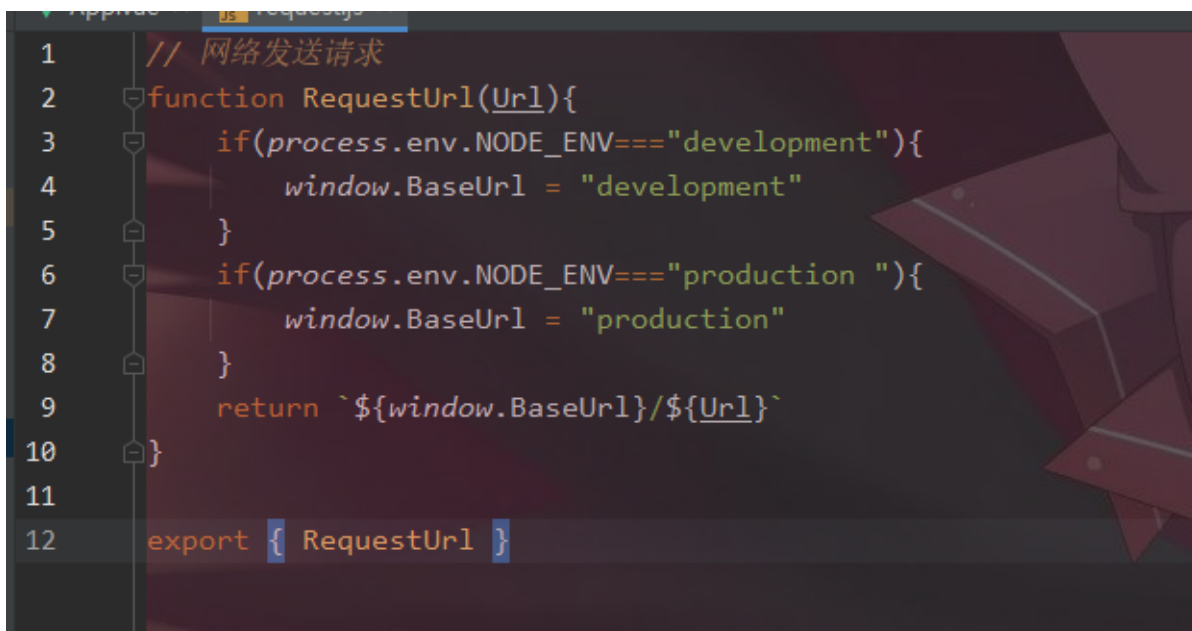


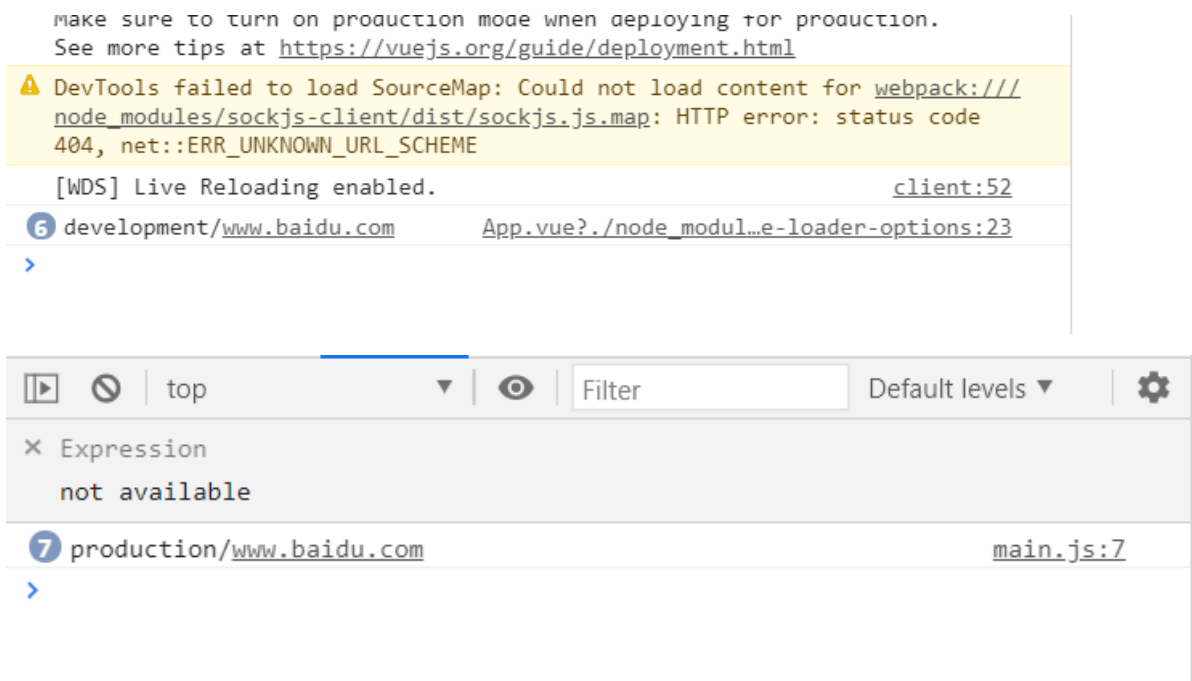
也就是说，你可以在打包后，知道你现在处于什么状态，让代码在不同的环境下有不同的用法

最常用的，可能是api的调用

比如你在开发环境中，用的是本地3000端口，但是在线上，用的是www.xxx.com这样的接口

那么，就可以通过识别环境，通过变量进行灵活的调用





两个环境下，输出的都是不一样，当然啦，我这个只是举个例子，正常肯定是用axios或者fetch这些请求，全局的话用vuex这种，大家知道有这么个东西就行了

后期我如果在工作用到，会一起发文章分享出来的。

代理

先附上链接：<https://webpack.docschina.org/configuration/dev-server/#devserverproxy>

啥是代理呢，就是前端访问后端的时候，因为不在一个服务器，所以拒绝访问，也就是不同源，webpack可以解决这个问题

webpack有个devServer.proxy的方法，也就是上面链接的方法

简单说下这个代码

```
module.exports = {
  //...
  devServer: {
    proxy: {
      '/api': {
        target: 'http://localhost:3000', //需要访问的地址
        //备注：你的访问地址需要改成你本地的地址，比如你访问的是3000/API/XXX，那么你访问的时候就要改成8080/API/XXX，让他识别
        pathRewrite: {'^/api' : ''}, //改写的api，当他匹配到这个api的时候，会自己匹配代理，具体可以根据实际更改，不一定是这个api这个词，或者自己用变量接住啥的，
      }
    }
  }
};
```

最后

写到这里，其实我想写其他东西，但是不太好写，主要这两天准备研究小程序，其实你们大概知道其他东西应该怎么去用了，但是这只是初级篇，如果想要深入，还需要看一篇文档以及在工作中根据需求去配置，还需要扎实的nodejs的基础，这玩意我是不太行，所以这只是初级篇，后期等博主学到了，就会继续发文章，一起学习！

这个配置，我也会一直用下去，如果遇到BUG，我会发文章的，因为这只是搭建vue的环境，还没进行实际项目编写，后面我会用这个写自己的博客