

之前是每次更新完都需要在命令行中输入webpack才可以实现代码的更新，
为了实现我们修改完代码就可以进行更新，可以安装下webpack-dev-server这个工具
使用webpack-dev-server 这个工具，实现自动打包编译的功能

1: 运行 `cnpm i webpack-dev-server -D`

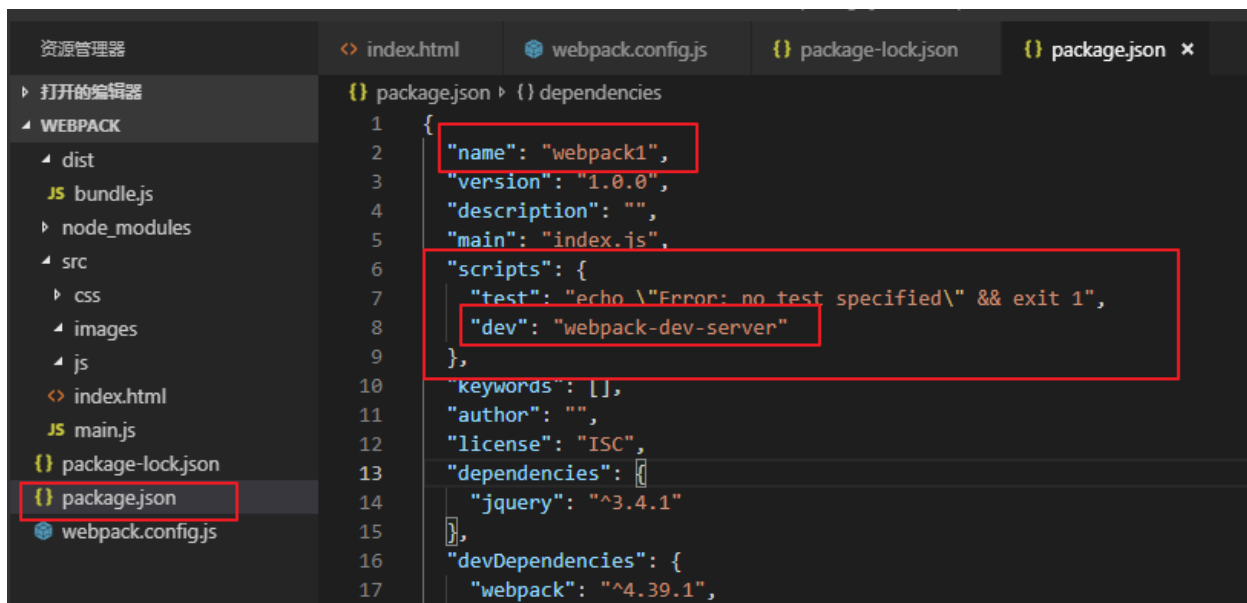
-D的意思（项目中/本地安装的）

把这个工具安装到项目的本地开发依赖

2: 用法和 `webpack` 命令用法一致

3: 由于是在项目中安装的，不是全局安装（-g）的，所以无法当作 脚本命令，
在powershell终端中直接运行

4:注意: `webpack-dev-server` 这个工具，如果想正常运行，
要求在本地项目中 必须安装 `webpack` ， 全局安装的webpack的不可以
安装完之后按照下图配置一下：



```
1 {
2   "name": "webpack1",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\"Error: no test specified\\" && exit 1",
8     "dev": "webpack-dev-server"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "dependencies": {
14    "jquery": "^3.4.1"
15  },
16  "devDependencies": {
17    "webpack": "^4.39.1",
```

5: 一般情况下，也需要在项目中安装webpack-cli模块，命令如下：

`cnpm i -D webpack-cli`

6: 安装完之后，在命令行中输入 `npm run dev`

```
问题 输出 调试控制台 终端
FS C:\Users\GouQian\Desktop\vscode学习\webpack> npm run dev
> webpack1@1.0.0 dev C:\Users\GouQian\Desktop\vscode学习\webpack
> webpack-dev-server

i [wds]: Project is running at http://localhost:8081/
i [wds]: webpack output is served from /
i [wds]: Content not from webpack is served from C:\Users\GouQian\Desktop\vscode学习\webpack
i [wdm]: Hash: ccaa47ba020e3bd3f2f9
Version: webpack 4.39.1
Time: 1722ms
Built at: 2019-08-08 12:25:45 PM
    Asset      Size  Chunks             Chunk Names
bundle.js  670 KiB       0  [emitted]  main
Entrypoint main = bundle.js
[0] multi (webpack)-dev-server/client?http://localhost ./src/main.js
[./node_modules/ansi-html/index.js] 4.16 KiB {main} [built]
[./node_modules/ansi-regex/index.js] 135 bytes {main} [built]
[./node_modules/html-entities/index.js] 231 bytes {main} [built]
[./node_modules/jquery/dist/jquery.js] 274 KiB {main} [built]
[./node_modules/strip-ansi/index.js] 161 bytes {main} [built]
[./node_modules/webpack-dev-server/client/index.js?http://localhost] 1.5 KiB {main} [built]
[./node_modules/webpack-dev-server/client/overlay.js] (webpack)-dev-server/client/overlay.js [built]
[./node_modules/webpack-dev-server/client/socket.js] (webpack)-dev-server/client/socket.js [built]
[./node_modules/webpack-dev-server/client/utils/createSocketUrl.js] (webpack)-dev-server/client/utils/createSocketUrl.js [built]
[./node_modules/webpack-dev-server/client/utils/log.js] (webpack)-dev-server/client/utils/log.js [built]
[./node_modules/webpack-dev-server/client/utils/reloadApp.js] (webpack)-dev-server/client/utils/reloadApp.js [built]
[./node_modules/webpack-dev-server/client/utils/sendMessage.js] (webpack)-dev-server/client/utils/sendMessage.js [built]
[./node_modules/webpack/hot sync ^\.\/log$] (webpack)/hot sync nonrecursive
[./src/main.js] 180 bytes {main} [built]
+ 19 hidden modules
i [wdm]: Compiled successfully.
```

以后我们编辑代码都会被实时监控，不用我们手动打包

5:

```
// 5. webpack-dev-server 帮我们打包生成的 bundle.js 文件，并没有存放到 实际的 物理磁盘上；而是，直接托管到了 电脑的内存中，所以，我们在 项目根目录中，根本找不到 这个打包好的 bundle.js；
```

6:

```
// 6. 我们可以认为， webpack-dev-server 把打包好的 文件，以一种虚拟的形式，托管到了 咱们项目的 根目录中，虽然我们看不到它，但是，可以认为，和 dist src node_modules 同级，有一个看不见的文件，叫做 bundle.js
```

7:打包完之后自动打开浏览器

```
index.html  webpack.config.js  package-lock.json  package.json x
} package.json > {} scripts > abc dev
2   "name": "webpack1",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1",
8     "dev": "webpack-dev-server --open"
9   },
```

8: 修改端口号: --port 端口号

```
{ package.json > {} scripts > abc dev
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1",
8     "dev": "webpack-dev-server --open --port 3000 --contentBase src --hot"
9   },
10
```

9: 直接打开某个文件 --contentBase 文件目录

10: --hot 异步刷新界面样式, 和 以增量的形式打包界面

11: 配置dev-server的第二种方式（推荐使用第一种）

一共有3步: 依次如下

```
index.html  package.json  JS main.js  webpack.config.js x
3  // 这个配置文件, 起始就是一个 JS 文件, 通过 Node 中的模块操作, 向外暴露了一个 配置对象
4  module.exports = {
5    // 大家已经学会了举一反三, 大家觉得, 在配置文件中, 需要手动指定 入口 和 出口
6    entry: path.join(__dirname, './src/main.js'), // 入口, 表示, 要使用 webpack 打包哪个文件
7    output: { // 输出文件相关的配置
8      path: path.join(__dirname, './dist'), // 指定 打包好的文件, 输出到哪个目录中去
9      filename: 'bundle.js' // 这是指定 输出的文件的名称
10   },
11   devServer: { // 这是配置 dev-server 命令参数的第二种形式, 相对来说, 这种方式麻烦一些
12     // --open --port 3000 --contentBase src --hot
13     open: true, // 自动打开浏览器
14     port: 3000, // 设置启动时候的运行端口
15     contentBase: 'src', // 指定托管的根目录
16     hot: true // 启用热更新
17   }
18 }
```

```
index.html JS main.js webpack.config.js x
1 const path = require('path')
2 // 启用热更新的第 2 步
3 const webpack = require('webpack')
4
```

```
},
plugins: [ // 配置插件的节点
  new webpack.HotModuleReplacementPlugin() // new 一个热更新的 模块对象，这是 启用热更新
    的第 3 步
]
}
```