# 包管理工具-npm

## npm是什么？

官网（<https://www.npmjs.com/>）对npm的描述是："npm makes it easy for JavaScript developers to share and reuse(重用) code, and it makes it easy to update the code that you’re sharing."

npm就是一个包管理工具，它是随node.js一起安装的，能解决node.js代码部署上的很多问题，常见的使用场景有以下几种：

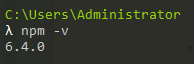
①允许用户从NPM服务器下载别人编写的第三方包到本地使用；

②允许用户从NPM服务器下载并安装别人编写的命令行程序到本地使用；

③允许用户将自己编写的包或命令行程序上传到NPM服务器供别人使用。

## 安装npm

npm是随着node.js安装的，所以先去<https://nodejs.org/en/>下载安装node.js，使用npm -v可以看到当前安装的npm版本：



## 更新npm

npm更新的速度远高于node.js，所以如果需要安装最新版可以执行以下命令：npm install npm@latest -g

"npm@latest" 其实是 <packageName>@<version> 的格式，在下载其他模块

时也是这个格式，当然也可以指定安装npm的版本

"-g" 代表全局安装

## package.json

package.json，定义了模块中所需各种模板及项目中的配置信息。一般放在npm下载的模块的根目录下面。npm install命令根据这个配置文件，自动下载所需的模块，也就是配置项目中所需的运行和开发环境。

package.json文件其实就是一个json对象，该对象的每个成员就是当前项目的每项设置，因此package.json满足json对象的限制条件，例如不能写注释

### package.json创建

使用 npm init 即可在当前目录创建一个 package.json 文件，输入 npm init 后会弹出一堆问题，可以输入对应内容，也可以使用默认值。在回答一堆问题后输入 yes，就生成按照你设置内容的package.json文件。也可以直接输入 npm init --yes 跳过回答问题步骤，直接生成默认值的 package.json 文件

### package.json配置项

package.json的完整配置项如下：

1、name：项目名称----------------必须

2、description：应用项目描述

3、version：版本号----------------必须

4、config：应用的配置项

5、author：作者

6、respository：资源仓库地址

7、licenses：授权方式

8、directories：目录

9、main：应用入口文件

10、bin：命令行文件

11、dependencries：项目应用运行依赖模块

12、devDependencies：项目应用开发环境依赖

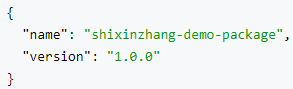
13、engines：运行引擎，指明node.js运行所需要的版本

14、script：脚本

在以上配置中，name和version的配置是必须存在的，它唯一标识一个模块：

1、name：要求全部小写，没有空格，可以使用下划线和横线

2、version：x.x.x的格式，满足“语义化版本规则”



### package.json搭建项目

#### 初始化项目

如果工程还没有package.json，使用npm init初始化工程

如果工程已经有package.json，使用npm install安装工程

(npm install 默认会安装 package.json 中 dependencies 和

devDependencies 里的所有模块)

#### 添加项目依赖

若想添加一个依赖(即别人已经编译打包好的模块)，使用npm install命令

使用 npm 安装 模块 有两种方式：本地（当前项目路径）安装 或者 全局安装

**1、本地安装**

执行命令：npm install <package\_name> -D

npm会自动寻找并下载模块放到当前项目的 node\_modules 文件夹下，并改变的package.json中dependencies 或devDependencies

如果需要安装指定版本的模块，

执行命令：npm install <package\_name>@版本号，其中latest表示最新，

例如:npm install jQuery@latest

npm install jQuery@2.3.1

执行npm install时，可以添加额外的参数：--save 或 --save -dev

不过好像默认安装的时候，不加它也会自动加到package.json文件中，

例如：

①npm install <package\_name> --save

表示将这个包名及对应的版本添加到 package.json的 dependencies

②npm install <package\_name> --save-dev

表示将这个包名及对应的版本添加到 package.json的 devDependencies

**2、全局安装**

全局安装比本地安装多了个 -g：npm install -g <package-name>

#### 更新项目依赖

若想知道模块是否有新版本，可以使用 npm outdated 查看，如果发现有的包有新版本，就可以使用 npm update <package-name> 更新它，或者直接 npm update 更新所有。

npm update 的工作过程是这样的：

①先到远程仓库查询最新版本

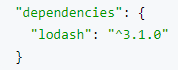
②然后对比本地版本，如果本地版本不存在，或者远程版本较新

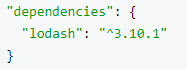
③查看 package.json 中对应的语义版本规则

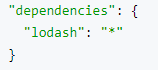
④如果当前新版本符合语义规则，就更新，否则不更新

这样会导致，即使有项目依赖是最新版的，但是由于package.json中该依赖当前语义版本满足要求，npm就不会去更新最新版依赖，例如：

当前package.json 的依赖是这样的：

这表示只接受小版本或者补丁版本的更新，因此在执行了一次 npm update 后它变成了这样：

第二位升到了最高，但是无法更新第一位，因此无法更新到最新的 4.17.4，若想安装最新版需要把它修改成：

表示任何版本的更新都接受，然后再执行 npm update，就发现更新成功了

#### 卸载项目依赖

卸载本地依赖(模块) 很简单，执行：npm uninstall <package-name> 即可：

# 项目编译构建-webpack

webPack是基于Node.js的，所以需要先下载安装好Node.js:

<https://nodejs.org/en/download/>安装完Node.js后，就使用npm命令

## webpack基本使用

### nrm-切换镜像地址

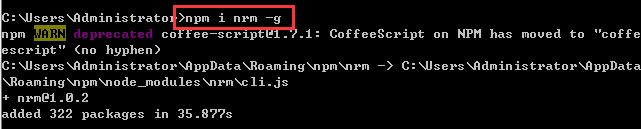
作用：提供了一些最常用的npm包镜像地址，能够快速地切换安装包时候的服务器地址。**PS：**nrm只是提供几个镜像地址，真正装包时都是用npm

1、运行 npm i nrm -g 全局安装nrm包

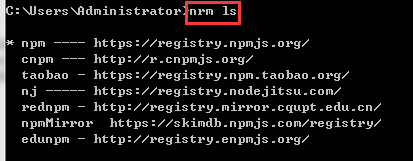
2、使用 nrm ls 查看当前所有可用的镜像源地址以及当前所使用的镜像源地址

3、使用 nrm use npm 或 nrm use taobao 切换不同的镜像源地址

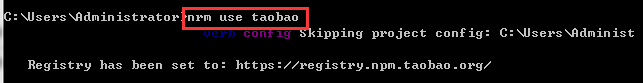
先安装：



再查看所有镜像地址：



最后切换当前使用的地址：



### cnpm-国内npm

npm的资源一般是在国外，所以使用淘宝的镜像根据cnpm。安装cnpm之前需要先安装好node.js并配置好环境变量，执行下面的命令：

npm install -g cnpm --registry=https://registry.npm.taobao.org

### 安装webpack

webpack有两种安装方式：

①全局安装webpack： npm i webpack -g

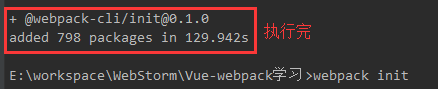
全局安装指定版本： npm install webpack@<版本号> -g

②在安装到项目： npm install --save-dev webpack

### 入门程序

1、安装完webpack后，就可以初始化项目：npm -init -y

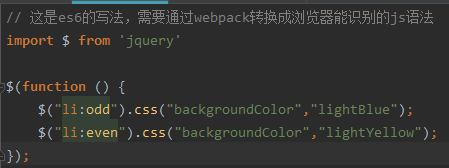
这条命令执行完，会在项目中增加一个package.json文件



2、加载需要的组件，比如我需要加载jquery组件：npm i jquery -S



3、在项目新建一个main.js，内容如下：

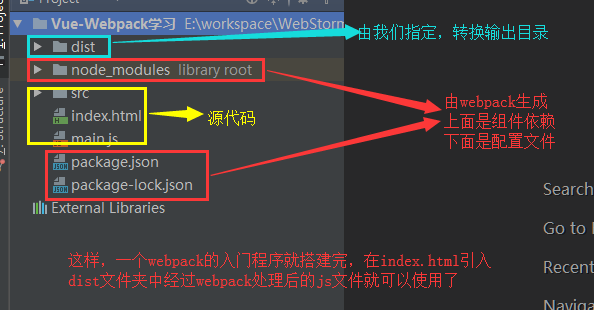


然后执行：webpack .\main.js .\dist\bundle.js

webpack会将main.js处理成浏览器能识别的js文件--bundle.js

( 存放位置和文件名由命令指定... )

4、整个项目结构如下：

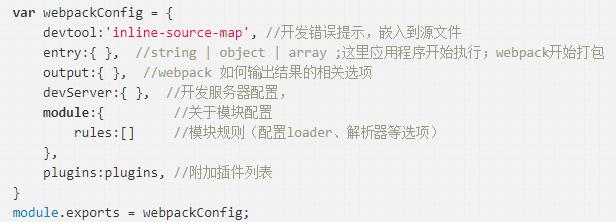


### webpack.config.js介绍

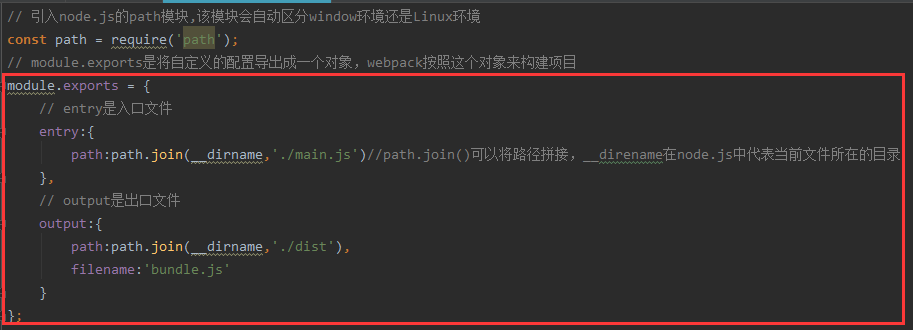
webpack.config.js是webpack的配置文件，webpack在执行时，除在命令行传入参数，还可以通过指定的配置文件来执行，一般会将它放到工程的跟目录，

它本身也是一个标准的Commonjs规范的模块。

它的配置架构如下：



最基本的webpack.config.js配置文件，只需指定入口和出口文件即可：



## webpack-dev-server插件

使用webpack-dev-server可以实现自动打包编译的功能

### 安装

①在项目根目录中安装webpack-dev-server：

执行命令: npm i webpack-dev-server -D

②webpack-dev-server强制要求在本地工程中安装webpack，所以还需要在

项目中本地安装webpack→执行命令：npm i webpack -D

③由于webpack-dev-server只是安装到本地工程中，因此无法直接在DOS窗

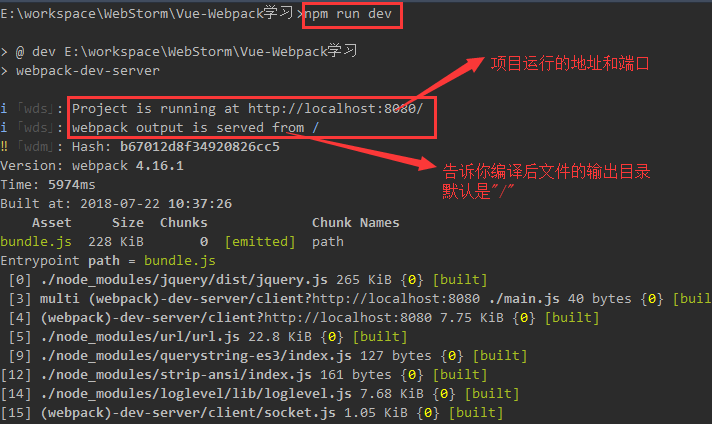
口使用，需要在package.json文件中配置：将webpack-dev-server配置

到scripts属性中，配置命令"dev"可自选。

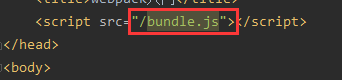


④当执行完以上3步后，webpack-dev-server就可以使用：

执行命令：npm run dev （dev按照package.json中的scripts属性配置）



⑤webpack-dev-server 打包生成的bundle.js文件，并没有存放到工程的物理磁盘上，而是直接保存到电脑的内存中。在webpack.config.js配置文件的output属性指定的输出目录中，是看不到打包好的bundle.js文件(这个文件也是通过webpack.config.js的output属性配置的)，所以，需要将引入bundle.js的html文件修改它的引入地址

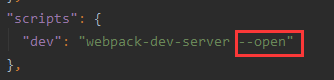


这样子，一旦修改源码并保存后，webpack-dev-server监听到会自动编译部署，我们才会在页面看到效果

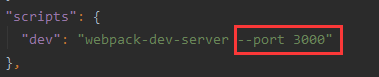
### 常用命令参数

webpack-dev-server的命令参数配置在packaeg.json的scripts属性

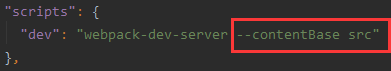
①项目部署后自动打开默认浏览器展示：--open



②指定端口打开： --port <端口号>



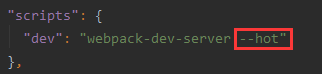
③指定项目打开的默认目录： --contentBase <目录>



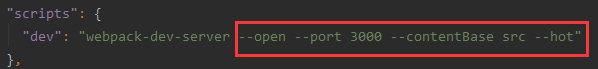
④热部署，修改样式后不需要刷新页面就可以生效，每次修改代码后，不会重新

编译打包整个项目，只是将修改的地方打成补丁就行。如果没有热部署，每次

修改都会重新打包整个项目： --hot



⑤这些命令可以一起存在，用空格隔开即可：



## html-webpack-plugin插件

html-webpack-plugin插件会根据配置自动映射物理硬盘的html 文件，将它存放在内存中并且引用相关的 assets(资源) 文件(如 css, js)。

### 安装配置

①在本地目录执行命令：npm i html-webpack-plugin -D

②在webpack.config.js中配置：

第一步：先引入html-webpack-plugin插件  
 

第二步：在plugins属性中配置第一步引入的对象：

所有webpack的插件都是配置在webpack.config.js文件中的

plugins属性(它是一个数组)，其中htmlWebpackPlugin就是第一

步引入的对象，它有个配置对象，指定要映射的html文件：



第三步：由于html-webpack-plugin插件会自动引入资源js文件，所以可以将物理硬盘上项目的html文件引入注释掉：

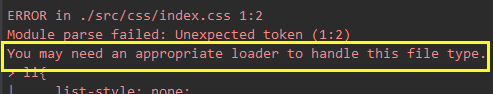


直接npm run dev就可以看到效果了....

## 第三方加载器(loader)

### css-loader

webpack默认也只能构建打包JS文件，遇到其他类型的文件需要先安装对应的加载器才能构建。例如让原生webpack加载css文件，会报错：



要让webpack能够构建打包css，需要用到"style-loader"和"css-loader"

①第一步：安装css加载器，执行命令：npm i style-loader css-loader -D

②在webpack.config.js配置加载器—配置加载器用到module属性：

module与entry、output是同级属性，它是一个对象，module里面有个rules属性(它是数组)，rules属性是设置匹配规则的，每条规则为一个对象，

每个对象包含"test"属性和"use"属性，如下图所示：



③第三步，在入口文件引入css即可生效：

webpack引入css文件，使用import+css文件路径 即可：



### less-loader

Less 是一门 CSS 预处理语言，它扩充了 CSS 语言，增加了诸如变量、混合（mixin）、函数等功能，让 CSS 更易维护、方便制作主题、扩充。Less 可以运行在 Node 或浏览器端。webpack加载.less文件需要less加载器：

①第一步，安装less-loader，而less-loader内部需要load，所以

需要执行：npm i less-loader -D 和 npm i less -D

②第二步，在webpack.config.js配置less加载器



③第三步，在入口文件，引入less即可：



### sass-loader

Sass 是一个 CSS 的扩展，它在 CSS 语法的基础上，允许使用变量, 嵌套规则, 混合, 导入等功能。sass 有两种后缀名文件：一种后缀名为 sass，不使用大括号和分号；另一种就是我们这里使用的 scss 文件，这种和我们平时写的 css 文件格式差不多，使用大括号和分号，建议用后缀名为 scss 的文件。

webpack加载sass文件需要sass-loader插件：

①第一步，安装sass-loader，而sass-loader内部依赖于node-sass，建议

使用cnpm安装node-css，npm经常性出错：

执行命令：cnpm i node-sass -D 和npm i sass-loader -D

②第二步，在webpack.config.js中配置sass加载器



③第三步，在入口文件，引入sass文件即可：



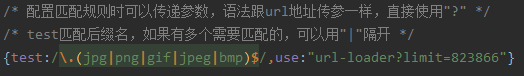
### url-loader

url-loader处理.css中对图片(.jpg、.jpeg、.png、.bmp、.gif等)的url引用，例如， background: url("../images/葫芦娃.jpg")，原生webpack没办法解析url地址，需要使用到第三方加载器：url-loader：

①安装url-loader，而url-loader内部依赖于file-loader，所以要将这两

个加载器一起安装，执行：cnpm i url-loader file-loader -D

②在webpack.config.js中配置url-loader：



③在css中引入图片地址：



注意，url-loader有几个新知识点：

①当loader匹配规则需要匹配多个文件类型时，可以用"|"隔开

加括号只是为了更容易看而已，不加也可以的



②指定loader处理特定类型文件可以传递参数，语法跟浏览器的url地址一样

直接在对应的"\*-loader"后加"?"：



url-loader的limit参数规定: 图片容量小于它，会以base64编码展示

图片容量大于等于它，以url形式展示

它的单位是字节(B)，以上图为例，当图片容量小于823866字节时，会以base64编码展示在页面上：



但是当图片容量大小大于或等于823866字节时，就以url的形式展示：



③url-loader并没有直接显示图片的原名称，而是以hash值的形式引用

为什么？webpack会把外部依赖的js、css、img等资源加载到内存中，

在项目启动的时候放在根路径中，如果图片名字一样，后加载的图片会把前加

载的图片覆盖掉。不过，通过name参数，让图片hash值和图片原名称一起

显示，这样既不会导致重命名也可以让图片辨识清楚



[hash:8]--意思是去该图片hash值的前8位

[name]--意思是图片原来的名称

[ext]--意思是图片原来的后缀

这样配置以后，图片的样式就变为：



④url-loader不仅可以处理图片，也可以处理字体，像bootstrap中就带有

很多字体，如果需要在node\_modules引入依赖的文件，可以这样写：

webpakc默认在node\_modules查找...



然后配置，字体匹配规则即可：



### babel-loader

webpack默认只能转换一部分的ES6新语法，对于更高级的ES6甚至ES7语法，webpack无法处理。需要安装[babel](https://www.babeljs.cn/)加载器，让babel解析完所有语法后将结果交于webpack打包编译执行...在webpack中安装配置babel可以分为如下步骤：

①安装babel基本包和插件包,执行命令：

cnpm i babel-core babel-loader babel-plugin-transform-runtime -D

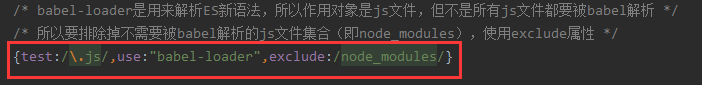
再安装babel语法解析包，执行命令：

cnpm i babel-preset-env babel-preset-stage-0 -D

②在webpack.config.js中配置babel-loader

由于babel-loader用来解析ES6或ES7新语法，所以它的作用文件的.js

文件，但又不是所有的.js都要让babel解析，所以需要如下配置：

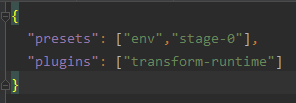


③在项目根目录下创建babel的配置文件".babelrc"(注意有个".")，该文件

属于JSON格式的文档，需要配置第①步安装的babel插件包和语法解析包，

第①步安装哪几个包，就在这里配置那几个...(注意前缀可以不用写)

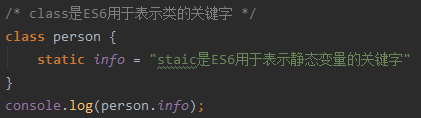
presets表示"babel语法解析",plugins表示"babel插件"



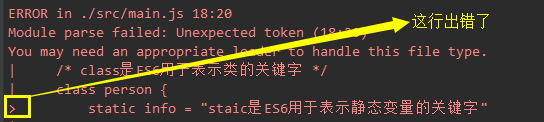
至此，babel从下载到配置都完成了.....

**测试：**

在main.js中写入ES6的语法：



如果没有安装或者配置好babel，webpack会抛出如下错误：

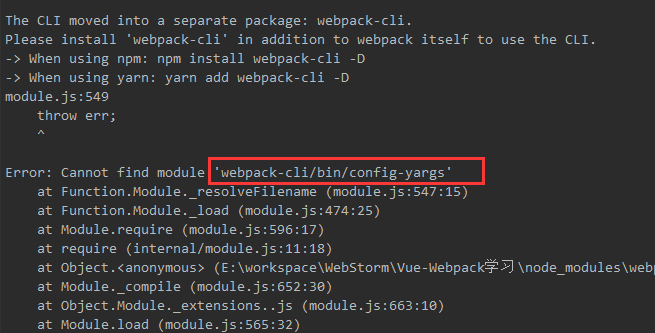


如果既安装好babel也配置好了，打开console口，可以看到可以正常运行打印了...

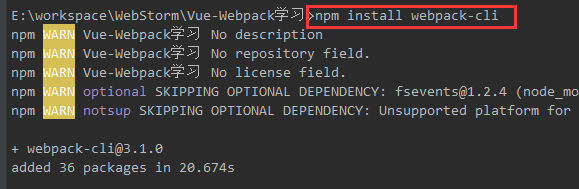
## 小结

### 运行webpack-dev-server出错

当安装好webpack和webpack-dev-server，准备运行时，报错：



原因是没有安装好webpack-cli，执行：npm i webpack-cli即可



### webpack执行加载器过程

1、webpack发现要处理的文件不是js，就去配置文件(webpack.config.js)

查询是否有对应的 第三方 loader 加载规则

2、如果能找到相应的规则，就直接调用 loader 处理这种类型的文件

3、调用 loader 的时候，是按照 use 定义的顺序 从左往右 调用的，直至

到最后一个loader，将处理结果交给webpack打包到bundle.js中

