**Webpack总结**

# 一、安装webpack

## 全局安装webpack指令: cnpm/npm install -g webpack

## 安装项目目录

### 1、建立一个文件夹（英文，除webpack）

### 2、cmd进入文件夹初始化 npm init

### 3、安装项目目录指令：cnpm/npm install webpack --save-dev

### 4、进入根目录后建立两个文件夹

* a、src文件夹 用来存放我们编写的javascript代码，可以简单的理解为用JavaScript编写的模块。

• index.html写好后，在src文件夹建立entery.js(入口)文件，用来编写JavaScript代码，也是入口文件

• entery.js代码

•

•

* b、dist文件夹 用来存放供浏览器读取的文件，这个是webpack打包后(代码压缩后)的文件。

• 在dist文件下建立index.html，

• 并写入代码

•

• 这里引入了一个JavaScript的bundle.js文件。这个文件现在还没有，这是用webpack执行打包命令后生产的文件

## 查看webpack版本指令： webpack -v

## 查看node\_modules版本指令：node\_modules/.bin/webpack -v

### 会提示需要安装 webpack-cli，按照提示使用 cnpm install webpack-cli –D 此时输入 node\_modules/.bin/webpack –v 会出现 webpack 的版本号，代表安 装成功

## 卸载webpack版本指令：cnpm/npm uninstall webpack -g

# 二、打包webpack文件

## 打包文件，执行命令： webpack src/entery.js dist/build.js

### src/entery.js 是入口文件的路径

### dist/build.js 是打包后存放的路径

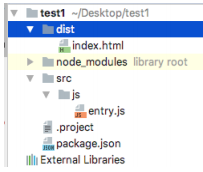
## 执行打包命令的结果

### 



## 新建项目结构进行打包

### 



* 将 src 中的 entry.js 文件打包到 dist 中

# 三、配置文件

## 为了快捷启动打包，通过对 package.json 进行配置

### 配置分成两个模式，npm run dev 进行打包的是开发者模式(打包 JS 未压缩)，build 是生产模式

### 



### 运行：npm run dev {入口文件路径} 直接默认打包到 dist 中 main.js

## 配置webpack.config.js文件，这个文件建立在项目根目录下

## 建立后写入代码

### 



* 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。

•

• 报错找不到path文件，在webpack.config.js的头部引入path
var path = require('path');

* 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，后期我们会详细讲解。

## 为了使用快捷命令打包，可以对 webpack.config.js 文件进行入口和出口的配置

### 



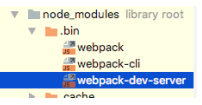
## 配置成功之后 直接运行 npm run dev 或者 npm run build 直接打包，会生成对应 文件

# 四、配置文件，服务和热更新

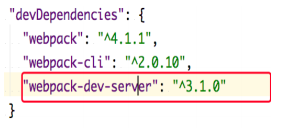
## 安装webpack-dev-server指令 cnpm install webpack-dev-server --save-dev

## 安装成功之后会出现

### 



### 



### 代表下载成功，然后进行配置

## webpack.config.js配置

### 



## 配置完成之后 本地启动热更新服务输入 node\_modules/.bin/webpack-dev-server --mode development 但是命令太长，可以再 package.json 中进行配置快速启动 项，在 script 中添加 如下

### 



## 此时可以使用 npm run server 进行启动热更新服务。 (PS：注意在 dist 文件夹中的 index.html 中引入正确的 JS 文件)然后更改入口文件 的 JS 内容，查看是否实现热更新！！！ 它是有一种监控机制的（也叫 watch）。它可以监控到我们修改源码，并立即在浏 览器里给我们更新。

# 五、CSS 打包和分离

## 1、loader 作用

### a) 可以将 sass 文件转化成 css

### b) 将 ES6/ES7 的代码转化成大多数浏览器支持的代码

### c) 将 React 中的 JSX 转化成 JavaScript 代码

## 2、 loader 需要单独安装 里面有配置参数

### a) test:正则匹配需要处理的文件拓展名，必须配置

### b) use:loader 名称，就是你需要使用的 loader,必须进行配置

### c) include/exclude:手动添加必须处理的文件夹或者排除的文件夹

### d) query:为 loader 提供额外的设置选项(可选)

## 3、 在 src 中的 css 文件夹中新建需要打包的 index.css 文件,并且将 css 文件引入 到 entry.js 中 import css from “../css/index.css”

## 4、 下载打包 css 所需要的必须的插件

### a) cnpm install css-loader --save-dev 将 css 文件插入到页面的 style 标签中

### b) cnpm install style-loader --save-dev 用来处理 css 文件中的 url 问题

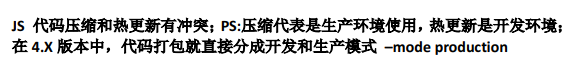
## 5、 在webpack.config.js配置 module 中的 loader

### 



## 6、 此时可以运行 npm run dev 进行打包，或者启动 npm run server 进行热更新 发现 css 已经正常引入到文件中；

### 



# 六、HTML发布

## 一般项目中生产环境代码会由打包生成，也就是说 dist 文件夹中的 index.html 也 是需要生成的，将 index.html 文件打包到 dist 中

### 1、 下载依赖包

* a) cnpm install –save-dev html-webpack-plugin

• I –save 指的是将依赖包名称和版本放置到 package.json 的清单中

• II –dev 是代表这个依赖包是属于开发环境

• III –production 使用 npm instal -production 只下载生产环境的包

* b) 在 webpack.config.js 进行引入依赖包

• const htmlPlugin=require(“html-webpack-plugin”)

* c) 对 congfig.js 进行配置

•

• minigy:是 html 文件压缩，里面参数是去除 html 中的双引号

• hash:为了开发中 js 有缓存效果，加入 hash 有效避免缓存

• template:要打包的 html 模板路径和文件名称

### 2、 下面就能直接使用npm run dev打包命令进行查看

# 七、CSS 中图片处理

## 对于图片路径处理，在 src 目录下新建 img 文件夹，内部放置图片，给 index.css 中的 div 添加一个背景图，我们直接打包会报错 意思是图片打包之后路径找不到

### 



## 需要安装 file-loader 和 url-loader 插件 安装指令：cnpm install -save-dev file-loader url-loader

### 1、 file-loader:解决引用路径问题，可以解析项目中的 url 路径问题(不仅限于 css 文件)，能根据我们配置将图片拷贝到相应路径，再根据我们打包路径指向正 确文件路径

### 2、 url-loader:如果图片较多，会有很多 http 请求，降低页面性能，通过 url-loader 会引入图片编码，生成 dataURL,相当于把图片翻译成一串字符，将字符打包 到页面中，如果图片比较大，编码会消耗性能，因此其提供了一个 limit 参数， 小于 limit 字节的文件就被转为 dataURL，大于 limit 会使用 file-loader 进行复制

### 3、 两者关系，为何只需要引入 url-loader

* a) url-loader 封装内置了 file-loader,url-loader 不依赖于 file-loader,fileloader方便以后使用
* b) 文件小于 limit 参数，url-loader 会把文件转化成 dataURL（base64 格式）

### 4、在webpack.config.js配置 module 中的 loader

• a) test；匹配图片文件后缀名

• b) use：是指定使用 loader 和 loader 的配置参数

• c) limit：是把小于 500000B 的文件打包成 base64 的格式，写入 JS 中，大
于就会调用 file-loader,参数也会传给 file-loader

## 单独分离 CSS 文件

### 有时候项目需要将 css 文件单独分离出来方便维护，就需要使用插件

* a) 安装：cnpm i -D extract-text-webpack-plugin@next 注:此版本和 3.X 有
  区别
* b) 安装之后需要引入到 webpack.config.js 中

•

• 配置webpack.config.js中的module

•

• 下一步配置webpack.config.js中的plugin内容 和之前版本差别较大

•

• 此时可使用 npm run server 查看效果，实现 CSS 效果即可达到目的，此
时运行 npm run dev 即可发现在 dist 目录中存在 css 文件夹存在 index.css 文
件。

• 但是如果在 css 中存在图片，打包之后会出现路径问题

• a) 于是在 webpack.config.js 上方声明一个对象叫做 website

•

• b) 打包上线问题测试时候可填写本地地址和端口号（如开启热更新），如
果上线需要填写域名和端口号(如：使用 npm run build 进行打包上线)，
再次修改 output 中的部分，设置成绝对路径，就不会出现路径找不到
的问题，修改如下

•

• c) 此时就完全实现 css 分离和 css 中图片路径问题；

• 接着进行处理 HTML 文件中 img 标签的图片路径问题，使用自带的 htmlloader
即可。如果不行就直接下载 （推荐使用第一种方式）

• a)在 webpack.config.js 中配置

•

• b) 也可以使用 html-withimg-loader 插件实现处理 html 文件中 img 的路径
问题，

• I 安装：cnpm install html-withimg-loader –save-dev

• II 在 webpack.config.js 中的loader配置

•

• 即可直接打包，推荐使用第一种方法

# 八、css进阶

## 1、自动添加css3前缀

### 通过使用 postcss-loader 给属性自动添加前缀，为了浏览器的兼容 性，有时候我们必须加入-webkit,-ms,-o,-moz 这些前缀。目的就是让 我们写的页面在每个浏览器中都可以顺利运行

### 安装 cnpm install --save-dev post-loader autoprefixer

### 建立一个postcss.config.js文件，和webpack.config.js同级

### 修改wepack.config.js文件

### npm run dev 打包查看css中有没有添加前缀

## 2、消除未使用的css

### 安装 cnpm i –D purifycss-webpack purify-css

* –D 代表的是 --save-dev 缩写

### 在css文件中直接添加一些不使用的css样式

### 配置webpack

* var glob=require('glob')
* var PurifyCSSPlugin=require('purifycss-webpack')
* 对plugins配置

•

# 九、给 webpack 添加 babel 支持

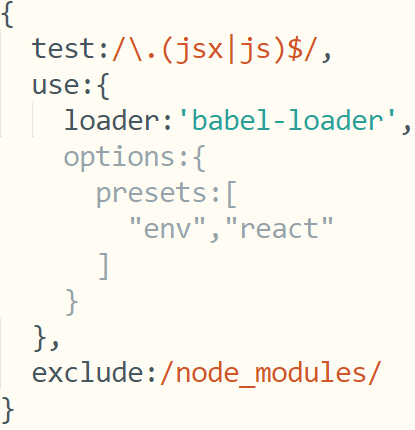
## babel

### 对es6和es7进行转码，能使用基于javascript进行拓展的语言，例如React的JSX

## 安装 cnpm install --save-dev babel-core babel-loader babel-preset-env babel-preset-react

## 配置webpack.config.js

### 



## 将入口文件的语法更改为es6语法，使用npm run dev查看是否被更改

## 打包调试

# 十、在项目中使用jquery

## 开发和生产环境都需要使用安装时候

### cnpm install jquery -save 保存到生产环境

### 在webpack.config.js头部引入 var webpack=require('webpack')

### 配置plugins

## 抽离jquery

### 修改webpack中的入口文件

### 引入optimize插件

• minChunks:固定配置，必须写，不写会打包失败

• filename可以省略，直接打包到dist文件夹下

## 多个第三方类库抽离(以引入vue为例)

### npm install vue --save

* 没有-dev 因为这个类库再生产环境中也要使用

### 在入口中配置vue和jquery

### 修改CommonsChunkPlugin配置

# 十一、静态资源集中输出

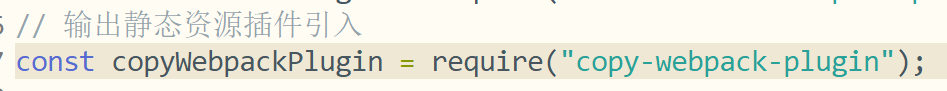
## copy-webpack-plugin 是专门做静态资源转移的插件

## 安装copy-webpack-plugin

### cnpm install --save-dev copy-webpack-plugin

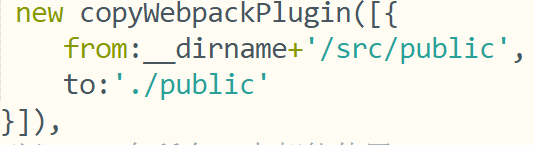
## 引入

### 



## 配置plugins

### 



* from 要打包的静态资源目录地址
* 要打包到的文件夹路径，跟随output配置中的，剥

# 十二、静态资源集中输出

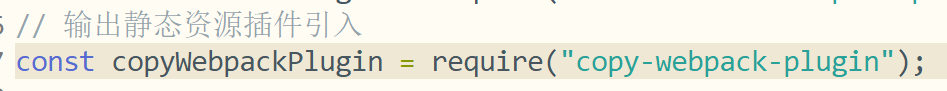
## copy-webpack-plugin 是专门做静态资源转移的插件

## 安装copy-webpack-plugin

### cnpm install --save-dev copy-webpack-plugin

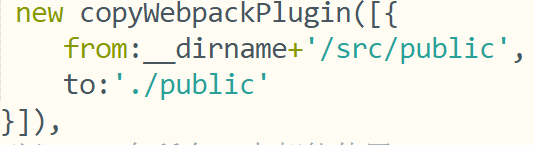
## 引入

### 



## 配置plugins

### 



* from 要打包的静态资源目录地址
* 要打包到的文件夹路径，跟随output配置中的，剥

# 十三、Json配置文件使用



# --save指的是将依赖包名称和

# -dev是代表这个依赖包是属于开发环境

# -production 使用npm install -production 只下载生产环境的包