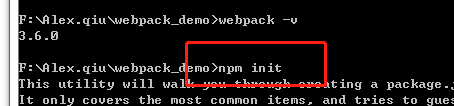
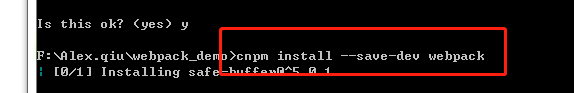


全局安装

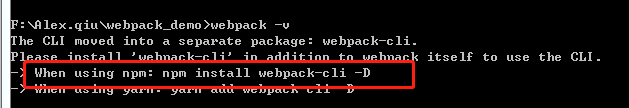


生成package.json配置文件



本地安装webpack

（官方不推荐全局安装 在项目中需要什么版本自行安装即可）



本地项目安装好webpack后 执行 webpack –v如果如图显示的话

意思是CLI被移动到了一个专门的包 webpack-cli里了。

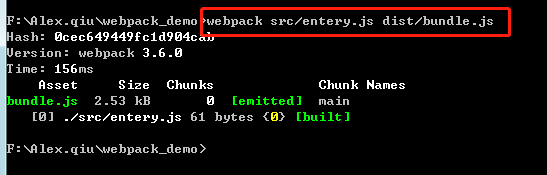
请安装webpack-cli 的除了webpack本身使用cli

当用npm时，使用npm install webpack-cli -D安装。

当用yarn时，使用yarn add webpack-cli -D安装。

解决办法：

npm install webpack-cli -D就能提示出版本号了。



目录结构一般都有dist(生产文件) src(项目文件)

Webpack src/xxx.js(入口文件) dist/yyy.js（出口文件）



配置webpack.config.js

1. 单入口单出口

Const path = require(“path”);

Module.exports={

Entry:{

// 这个entry的名字是随便取得

Entry:’./src/entry.js’

},

Output:{

//output有2个参数 一个是path(输出的路径) 这里配置用的是node写法意思是项目文件dist的绝对路径 ，filename就是输出的文件名

Path:path.resolve(\_\_dirname,’dist’),

Filename:’bundle.js’

}

}

1. 多入口多出口

Const path = require(“path”);

Module.exports={

Entry:{

Entry1:’./src/entry1.js’,

Entry2:’./src/entry2.js’

},

Output:{

Path:path.resolve(\_\_dirname,’dist’)，

Filename:’[name].js’

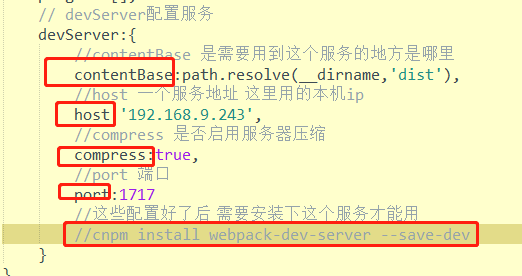
}

}

服务和热更新

--registry=https://registry.npm.taobao.org

Cnpm有的时候安装可能不全 用上面这个npm的可以



devServer是配置服务的 有下面几个参数

contentBase:哪里需要用到这个服务 对应一个绝对路径

host:’192.168.9.243’//配置一个服务地址

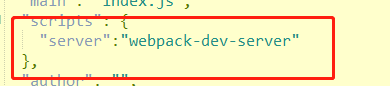
compress:true // 是否启用服务器压缩 知道就行

port:1717 // 端口

配置完以上的参数后 就启动服务 需要先安装下服务

Cnpm install webpack-dev-server –save-dev

安装好了之后直接在终端运行webpack-dev-server会报错的，因为服务安装的是在项目的node\_modules里面并不是环境变量里面，所以不能直接调用，我们需要在package.json里面配置下



这样 直 npm run server 就可以运行了 服务地址就是192.168.9.243:1717

如果报错为 can’t found module ‘xxxxx’ 那么你就去npm install 缺的那个xxx 要是还是不好使的话 就把node\_module删了 从新安装下 可以用

Npm install --registry=https://registry.npm.taobao.org

就可以了 webpack3.6以上的 启动服务后都支持热更新

打包css文件

Webpack在生产环境中有一个重要的作用就是减少http的请求数，就是把多个文件打包到一个js里，这样请求数就可以减少好多。这节课我们就学习一个重要的知识，把我们的CSS文件打包。在学习CSS打包之前，需要先对webpack.config.js里的Loaders配置项进行了解。

Loaders

Loaders是Webpack最重要的功能之一，他也是Webpack如此盛行的原因。通过使用不同的Loader，Webpack可以的脚本和工具，从而对不同的文件格式进行特定处理。

可以把SASS文件的写法转换成CSS，而不在使用其他转换工具。

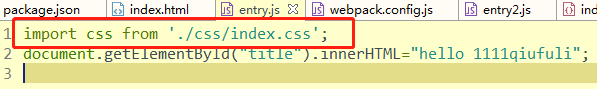
可以把ES6或者ES7的代码，转换成大多浏览器兼容的JS代码。

可以把React中的JSX转换成JavaScript代码。

注意：所有的Loaders都需要在npm中单独进行安装，并在webpack.config.js里进行配置

1. 先创建一个css文件,配置好样式后 在js中引用

Import css from ‘./css/index.css’



1. 引用好了之后 我们就用loader来解析css文件啦！





需要安装style-loader 和 css-loader

Styler-loader：它是用来处理文件中的url（）

Css-loader：它是用来将css插入页面的style标签的

这两个都下载好了之后我们就去配置webpack.config.js的loaders啦！

3.webpack.config.js中配置



还有其他的写法

{

Text:/\.css$/,

Loader:[‘style-loader’,’css-loader’]

} or

{

Text:/\.css$/,

use:[{

loader:”style-loader”

},{

Loader:’css-loader’

}]

}

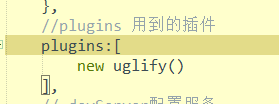
参数说明：

* test：用于匹配处理文件的扩展名的表达式，这个选项是必须进行配置的；
* use：loader名称，就是你要使用模块的名称，这个选项也必须进行配置，否则报错；
* include/exclude:手动添加必须处理的文件（文件夹）或屏蔽不需要处理的文件（文件夹）（可选）；
* query：为loaders提供额外的设置选项（可选）。

Js代码压缩

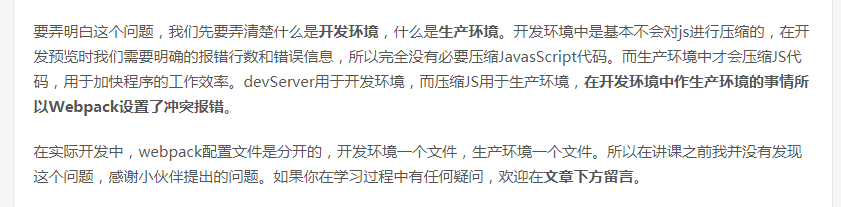
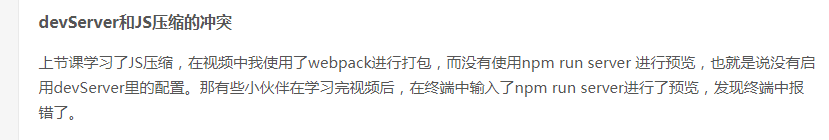
现在你写的JS代码，在上线之前，都是需要进行压缩的，在没有webpack和gulp这些工具前，你可能需要找一个压缩软件或者在线进行压缩，在Webpack中可以很轻松的实现JS代码的压缩，它是通过插件的方式实现的，这里我们就先来引入一个**uglifyjs-webpack-plugin**(JS压缩插件，简称uglify)。

const uglify = require('uglifyjs-webpack-plugin');



然后 终端 执行 webpack 就好了

（压缩的文件是生产环境需要的 ，开发环境的js100%不能压缩）



Html文件打包

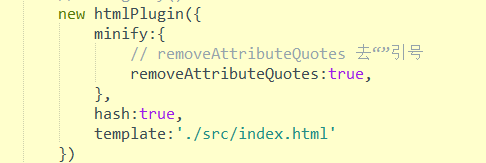
const htmlPlugin= require('html-webpack-plugin');

html打包和js压缩不一样 需要安装一下

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

最后在webpack.config.js里的plugins里进行插件配置，配置代码如下。

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



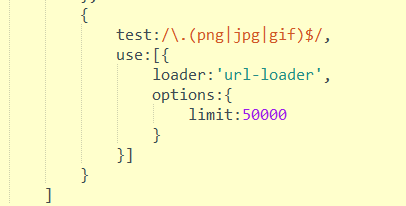
CSS中引用图片

1. 首先需要建立一个images文件夹 放入一张图片 在index.html中加入<div id=”tupian ”></div>在css把图片引入好

这个时候要是直接webpack打包会报错

需要 cnpm install –save-dev filer-loader url-loader

安装好后配置



**url-loader：**如果图片较多，会发很多http请求，会降低页面性能。这个问题可以通过url-loader解决。url-loader会将引入的图片编码，生成dataURl。相当于把图片数据翻译成一串字符。再把这串字符打包到文件中，最终只需要引入这个文件就能访问图片了。当然，如果图片较大，编码会消耗性能。因此url-loader提供了一个limit参数，小于limit字节的文件会被转为DataURl，大于limit的还会使用file-loader进行copy。

**file-loader：**解决引用路径的问题，拿background样式用url引入背景图来说，我们都知道，webpack最终会将各个模块打包成一个文件，因此我们样式中的url路径是相对入口html页面的，而不是相对于原始css文件所在的路径的。这就会导致图片引入失败。这个问题是用file-loader解决的，file-loader可以解析项目中的url引入（不仅限于css），根据我们的配置，将图片拷贝到相应的路径，再根据我们的配置，修改打包后文件引用路径，使之指向正确的文件。

**为什么只使用了url-loader**

有的小伙伴会发现我们并没有在webpack.config.js中使用file-loader，但是依然打包成功了。我们需要了解file-loader和url-loader的关系。url-loader和file-loader是什么关系呢？简答地说，url-loader封装了file-loader。url-loader不依赖于file-loader，即使用url-loader时，只需要安装url-loader即可，不需要安装file-loader，因为url-loader内置了file-loader。通过上面的介绍，我们可以看到，url-loader工作分两种情况：

1.文件大小**小于**limit参数，url-loader将会把文件转为DataURL（Base64格式）；

2.文件大小**大于**limit，url-loader会调用file-loader进行处理，参数也会直接传给file-loader。

也就是说，其实我们只安装一个url-loader就可以了。但是为了以后的操作方便，我们这里就顺便安装上file-loader。

这时候再webpack就好了 重复的webpack需要把dist目录清空 然后npm run server 就可以了

CSS分离与图片路径处理

### CSS分离:extract-text-webpack-plugin

### 

### 

### 这个时候 执行webpack如果报错为

### 

### 解决方案：

### npm install extract-text-webpack-plugin@next

### 图片路径问题：

### 利用extract-text-webpack-plugin插件很轻松的就把CSS文件分离了出来，但是CSS路径并不正确，很多小伙伴就在这里搞个几天还是没有头绪，网上也给出了很多的解决方案，我觉的最好的解决方案是使用publicPath解决，我也一直在用。

### 假如把limit改小了 dist生成的文件就会有base64的图片 那么引入的路径就是不对的是直接dist>xxx.png

### 

### 

### 

### 这样就有了一个公共路径也是绝对路径 在webpack就可以了

# 处理HTML中的图片

Css既然可以分离 那么images也可以打包成单独的文件夹



outputPath

### 这回你再执行打包就可以把图片打包到images文件夹里了

### html-withimg-loader

### html-withimg-loader就是我们今天的重点了，这个插件并不是很火，也是我个人喜欢的一个小loader。解决的问题就是在hmtl文件中引入<img>标签的问题。

### npm install html-withimg-loader –save

### 

打包和分离LESS







这样就能运行了 不过less会被打包到entry.js中

### 把Lees文件分离。





这样less会被放到extractTextPlugin所配置的文件中去 跟css一样

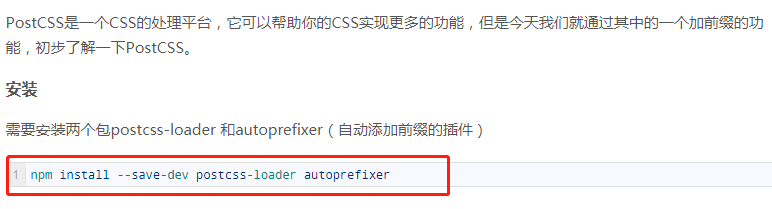
打包和分离SASS

跟less的步骤是一样的 就是安装的包不一样





postcss自动添加CSS属性前缀



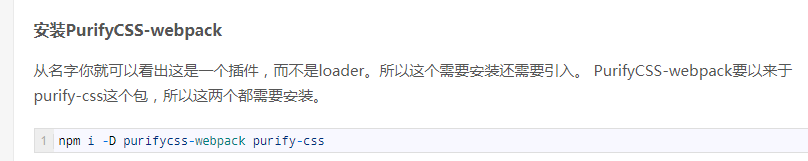






消除无用的CSS

比如用bootstrap框架一样 或者自己写的冗余代码



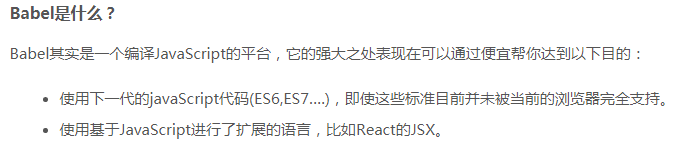


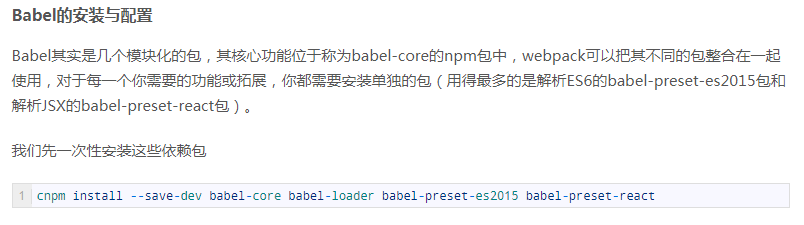


这里配置了一个paths，主要是需找html模板，purifycss根据这个配置会遍历你的文件，查找哪些css被使用了。

**注意**：使用这个插件必须配合extract-text-webpack-plugin这个插件，这个插件在前边的课程已经讲解过了。如果你还不会请自学一下。

Babel转换ES6





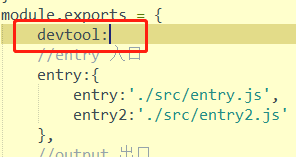


上面的代码使用了ES6的let声明方法。如果你不使用Babel来进行转换，你会发现打包出来的js代码没有作兼容处理，使用了Babel转换的代码是进行处理过的。





打包后的代码调试

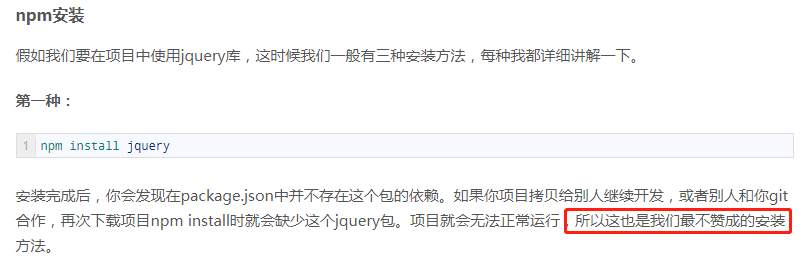


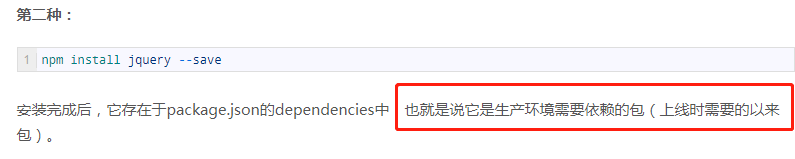
有4总模式：

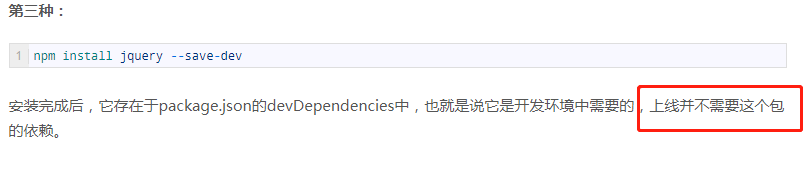
* source-map:在一个单独文件中产生一个完整且功能完全的文件。这个文件具有最好的source map,但是它会减慢打包速度；（即包括行和列）
* cheap-module-source-map:在一个单独的文件中产生一个不带列映射的map，不带列映射提高了打包速度，但是也使得浏览器开发者工具只能对应到具体的行，不能对应到具体的列（符号）,会对调试造成不便。（不包括列）
* eval-source-map:使用eval打包源文件模块，在同一个文件中生产干净的完整版的sourcemap，但是对打包后输出的JS文件的执行具有性能和安全的隐患。在开发阶段这是一个非常好的选项，在生产阶段则一定要不开启这个选项。
* cheap-module-eval-source-map:这是在打包文件时最快的生产source map的方法，生产的 Source map 会和打包后的JavaScript文件同行显示，没有影射列，和eval-source-map选项具有相似的缺点。

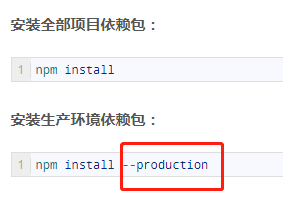
开发和生产环境并行

* 开发依赖：只在开发中用来帮助你进行开发，简化代码或者生成兼容设置的以来包。你可以打开package.json来查看，devDependencies的下面的这些包为开发使用的包。这些包在生产环境中并没有用处。
* 生产依赖：就是比如我们的js使用了jquery，jquery的程序要在浏览器端起作用，也就是说我们最终的程序也需要这个包，这就是生产依赖。这些包在dependencies中。









### 配置生产和开发并行

我们在以前的配置中设置了一个变量**website**，用于静态资源正确找到路径。那如果生产环境和开发环境不一样，而且我们需要来回切换，这时候我们需要更好的设置方法。







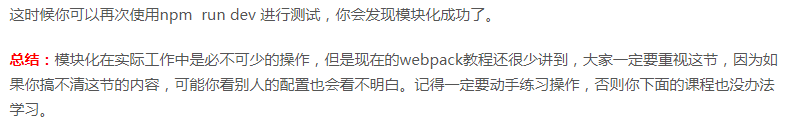


模块化配置

Webpack的模块化跟es6的语法差不多







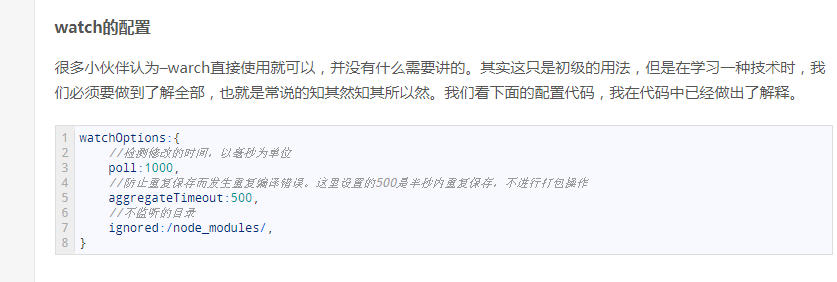
优雅打包第三方类库

Jquery,vue,react….



watch正确使用方法

在初级开发阶段，使用webpack-dev-server就可以充当服务器和完成打包任务，但时随着你项目的进一步完成，可能需要前后台联调或者两个前端合并代码时，就需要一个公共的服务器了。这时候我们每次保存后手动打包显然效率太低，我们希望的场景是代码发生变化后，只要保存，webpack自动为我们进行打包。这个工具就是watch，这节课我们把wacht完全学会，你会发现在开发中更加的得心应手。

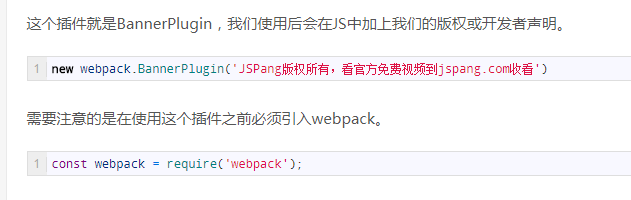


执行

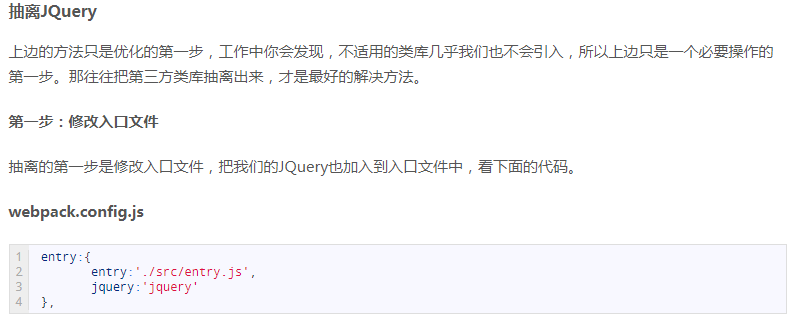
Webpack –watch 要是自己在config中配置了 直接把watch加在后面就行

BannerPlugin插件

为的就是在发生问题时可以找到当时写代码的人。有时候也用于版权声明。

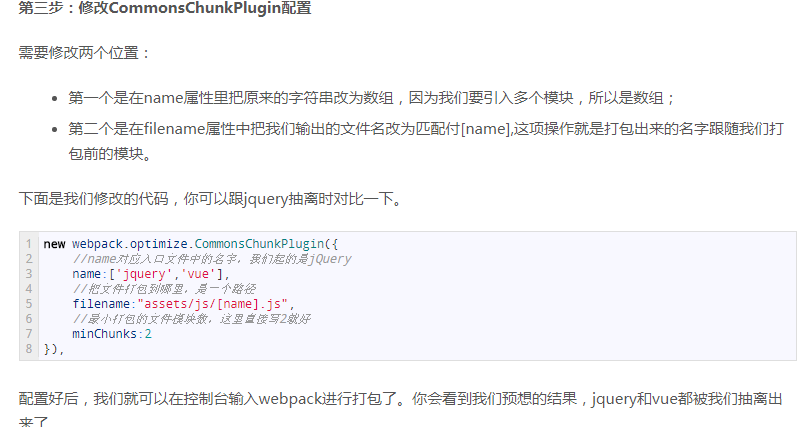


实战技巧：webpack优化黑技能









**总结：**在项目开发中，我们很使用很多第三方类库，那好的做法就是把第三方这些类库全部抽离处理，这样在项目维护和性能上都是不错的选择。希望学会这个技巧后，你也能在工作中使用上。