1. webpack
   1. 概念

webpack是前端资源模块化管理和打包工具，可以按照设定的依赖规则将模块打包成前端部署资源。

之前的前端任务工具通常都是将标记、样式、JS文件分离管理。而webpack则从整体角度考虑，使用js来管理它们之间的依赖关系。可以按需引入模块、解决如重复引入模块等功能。

webpack中，可以使用如es6模块的import语句、commonJS的require语句、AMD的require/define语句、css/less/sass的@import语句等来表达模块之间的依赖。

webpack通过loader支持各种语言和预处理器编写模块。通过loader转换，ES6模块、AMD模块、图片、LESS等都被视为模块。

webpack从入口点开始，递归地构建一个依赖图表。这个依赖图表包括应用所需的每个模块，包括非代码资源（例如图像或web字体）。然后将所有模块打包为少量的包（bundle），被可由浏览器加载。

* 1. 基本使用
     1. 入口

入口是webpack创建所有模块、代码的依赖关系图表的起点。配置项有：

**1.context**

entry和loader的参考目录，默认当前目录，为绝对路径。通常没见用到。



**2.entry**

（1）单入口

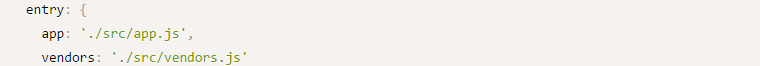
可以为1个字符串，也可以为1个数组，但都是单入口。如果是数组，最终还是会打包到1个出口文件中。



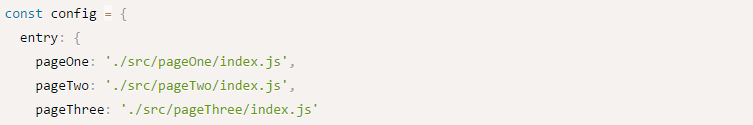
（2）多入口

对1个对象，将根据这些入口文件创建不同的依赖路径包。

分离第3方代码：



多页应用：



* + 1. 输出

指定在哪里输出打包好的应用程序，为1个对象，至少应该有filename和path两个配置项。



常用配置：

**1.path和publicPath**

（1）最终的路径有publicPath和path一起决定。最终打包的路径是publicPath+path+配置的路径。所以，没有publicPath的时候，就只由path决定。

（2）path必须是一个绝对路径。publicPath默认值为空字符串。

（3）publicPath的意义：如，在localhost环境下，css文件中可以使用“./test.png”来加载本地图片。此时，不写publicPath。但在生产模式下，“test.png”文件可能会定位到CDN上。这就意味着在生产环境需要更新所有文件里的url为CDN的路径。显然，手动更新特别麻烦，这时就可以添加publicPath配置，再结合一些插件自动更新这些url。

**2.filename**

具体为什么名称，可以使用name、id、hash、chunkhash等。其中，name就是entry的文件名（单文件）或key（多文件）。









* + 1. 使用加载器解析

**1.使用方式**

有3种方式使用方式：

（1）通过webpack.config.js

写在module.rules下，可以写多个，如下：



（2）通过require

在require语句（或define等）中指定loader。使用!将资源中的loader分开。分开的每个部分都相对于当前目录解析。会覆盖配置文件中的loader。?后面为参数，格式为?key=value&foo=bar。



应尽量减少使用。

（3）CLI



**2.rules及rule的组成**

（1）rules为1个规则数组。按照其中的匹配规则，能够对模块应用加载器。

（2）每个rule可以分为3部分：条件、结果、嵌套规则。

（3）rule中属性可以为：

* test、include、exclude等值，如下：



* use：指定loader，支持链式调用。可以使用1个options，其值可以传递到loader中，即loader选项。



* enforce：用于指定loader种类。值有："pre"|"post"。没有值表示是普通loader。
* 其他：如loaders，use的别名，最好不要使用。options/query，是Rule.use:[{options}]的简写，也最好不要使用。

**3.加载器自身特性**

所谓Loader，只是导出了一个函数的js模块。Compiler会调用这个函数，然后把之前Loader产生的结果或者资源文件传入进去，返回处理后的资源文件。loader通常被命名为XXX-loader，其中XXX是上下文的名称，例如json-loader。加载器有如下特征：

（1）运行在Node.js中。

（2）支持链式传递，按照先后顺序进行编译。前一个loader返回值给下一个loader。

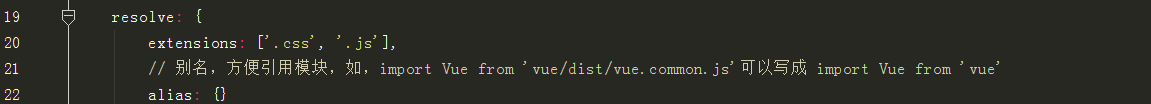
* + 1. 忽略解析匹配文件

使用Module的noParse属性。



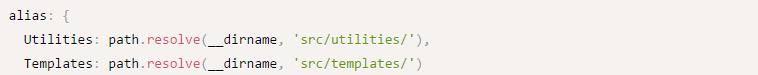
* + 1. 如何解析（别名、后缀）

用于控制如何加载并解析模块。



（1）alias

创建import或require时的别名，如下，将1个路径作为1个别名，在之后import时，就可以使用这个别名+后面的路径，可以更简洁一点。





（2）extensions

自动解析确定的扩展，默认值为[".js",".json"]。

* 1. 代码分离

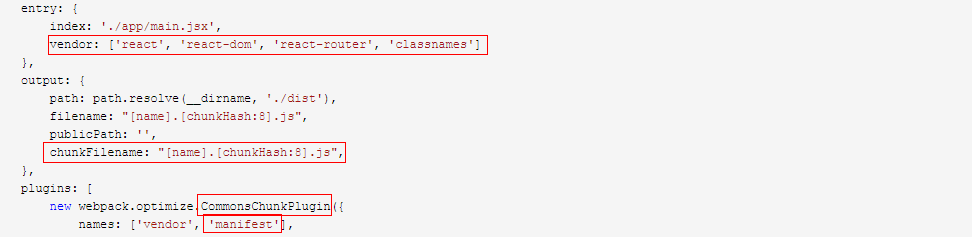
有3中常用分离方法：（1）在entry中手动分离。（2）使用CommonsChunkPlugin分离js代码。（3）模块的内联函数调用来分离代码（动态导入）。

* + 1. 分离第三方js

**1.为什么要分离？**

当多入口时，如果不分离，会将第三方代码打包到每个出口文件bundle中。

**2.配置**



说明：

（1）入口哪里新增加1个入口，没有使用具体路径，那就会从node\_modules中加载。

（2）chunkFilename用于指定第三方代码文件（非入口chunk文件）名称。

（3）特别要注意CommonsChunkPlugin配置中有个manifest。如果不加这个，每次运行webpack，都会再次编译vendor文件，即使它没有改变。如上，chunkHash会改变，导致浏览器会重新加载文件。

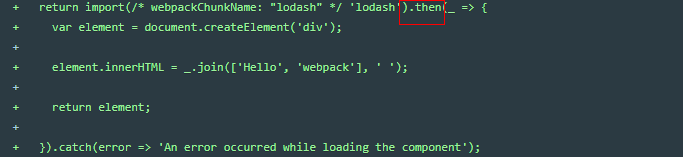
出现的原因是每次构建时，webpack会生成webpack runtime代码，用来帮助webpack完成其工作。当只有1个bundle时，runtime代码驻留在其中。当生成多个bundle的时候，runtime代码被提取到公共模块中，这里就是vendor文件。因此，需要将运行时代码提取到一个单独的manifest文件中。

* + 1. 动态导入

（1）配置chunkFilename。



（2）使用import导入模块。注意import返回的是1个promise。



此时，该chunk将分离处理，成为1个单独的文件。



* + 1. 懒加载

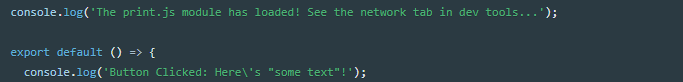
（1）即按需加载。原理位：在代码中的一些逻辑断点处分割代码，当完成某些操作后，如果需要引用另外一些新的代码块，才加载这些代码。有助于加快应用的初始加载速度，减轻总体体积，因为某些代码块可能永远不会被加载。

（2）懒加载针对的是模块，所以首先需要代码分离（动态导入）配置。

（3）如下，有2个文件：index和print。

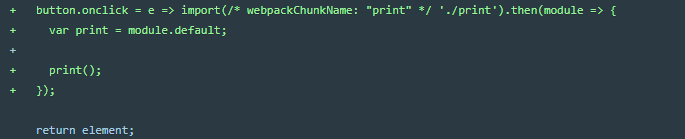


print.js文件为：



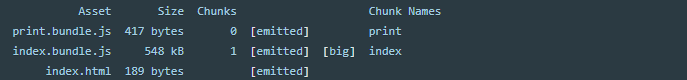
index.js文件为：







（4）打包后有：



（5）vue中有自己的实现。

* + 1. 将css从js文件分离出来

如果通过webpack打包CSS，需要像任何模块一样将CSS导入JavaScript代码，并使用css-loader，以便使CSS作为JS模块输出。可以使用ExtractTextWebpackPlugin提取打包的CSS并分离输出CSS文件。







将单独输出styles.css（因为不需要将css写入html文件，因此不需要style-loader）。

* + 1. 不打包从外部获取的包

**1.基本使用**

如，希望通过CDN方式加载jquery，不需要将jquery打包到输出文件中，但代码中还能用jquery。可以使用externals配置。





在编译后的文件中，如下：



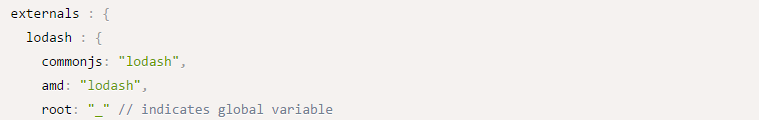
**2.支持的模块上下文**

（1）global：默认设置，外部library能够作为全局变量使用。用户可以通过在script标签中引入来实现。

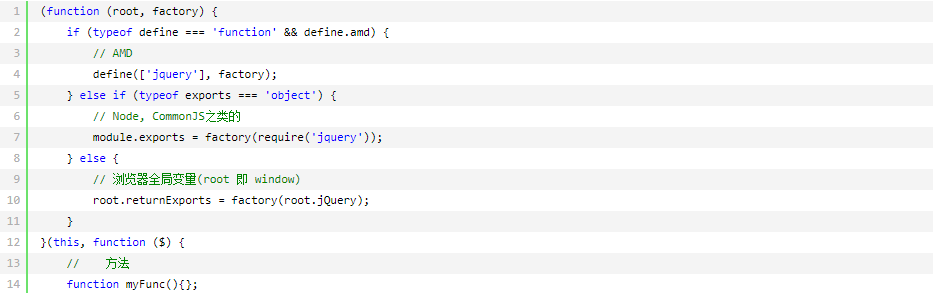
（2）commonjs：应用程序使用CommonJS模块系统，因此外部library应该使用CommonJS模块系统，并且应该是一个CommonJS模块。

（3）commonjs2：同上，但导出的是module.exports.default。

（4）amd：同上，但使用AMD模块系统。



上面的意思就是应用程序的模块化系统：



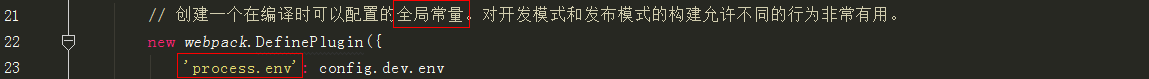
* 1. 常用操作
     1. 创建全局变量

如，每个文件都会引入jquery，可以ProvidePlugin创建1个jquery全局变量。在每次遇到全局标识符$的时候，在webpack中预先设置var $=require(“jquery”)。这样就不用每个文件都写一遍。而且，只有在需要使用此变量的时候，这个模块才会被require进来。



* + 1. 开发环境

（1）使用DefinePlugin。



代码中如果有process.env这个常量，将会被config.dev.env的值代替。

（2）命令行中使用-p选项。

执行webpack -p，也会调用DefinePlugin。在导入的代码中,任何出现process.env.NODE\_ENV的地方都会被替换为"production"。

* + 1. SourceMaps

使用如下配置：

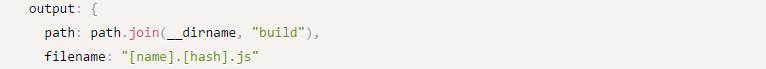


* + 1. 缓存

（1）问题：如果更新版本时没有更改打包资源的文件名，浏览器会认为文件没有被更新，就会使用浏览器自己的缓存版本。

（2）之前的解决方法：添加1个内部版本号作为参数。

（3）webpack的解决方法：通过每次webpack构建时都会生成1个唯一的哈希值用来构成文件名，如下：



（4）上面方法的问题：任何文件更新之后构建都会更新所有文件名，而不是只更新变化的文件。解决方法：使用[chunkhash]替换[hash]即可。但不要在开发环境下使用[chunkhash]，因为会增加编译时间。

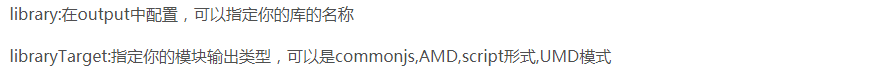
（5）上面的处理还存在另外1个问题，那就是即使只有1个文件修改，webpack仍然会编译所有文件。原因是webpack在编译期间使用标识符（不是模块名称）来标识模块。为此，webpack内部有1个标识符和文件名的映射表，名为chunk manifest的js对象。该对象默认会保存到每个输出文件中。因此，可以将它提取出来，即前面的提取公共模块。

* + 1. 去掉无用代码

依赖于ES6模块import/export模块系统的静态结构。webpack2内置支持对ES6模块（别名harmonymodules）以及未使用的模块的导出检测。当运行生产环境构建，node\_modules/.bin/webpack --optimize-minimize main.js dist.min.js时，自动去掉没有使用的模块。

* + 1. 自定义库

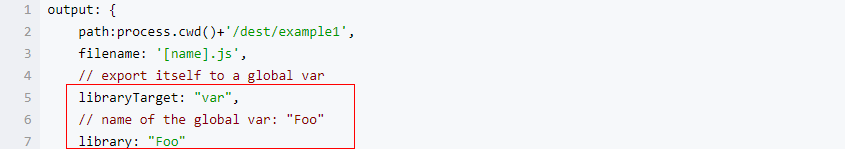
（1）使用library和libraryTarget，含义如下：



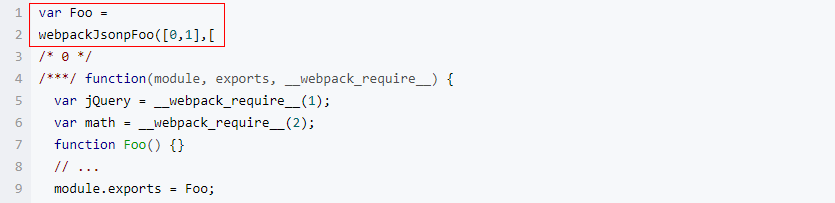
（2）libraryTarget指定的导出格式可以为：

* var：导出为一个变量：var Library=xxx（默认）。
* this：导出为this的一个属性：this[Library]=xxx。
* commonjs：导出为exports的一个属性：exports[Library]=xxx。
* commonjs2：通过module.exports：module.exports=xxx导出。
* amd：导出为AMD（可选命名：通过library选项设置名称）。
* umd：导出为AMD，CommonJS2或者导出为root的属性。

（3）使用：如将自己的包打包为名为Foo的文件。



打包之后的代码为：



* 1. 热更新
     1. DevServer

为1个对象。如果通过Node.jsAPI来使用dev-server，配置项将作为第2个参数传入：new WebpackDevServer(compiler,{...})。

（1）publicPath

此路径下的打包文件可在浏览器中访问（1个安全相关的设置）。确保publicPath总是以斜杠(/)开头和结尾。

（2）contentBase

规定服务器获取前端静态文件的地址。但devServer.publicPath优先级更高。推荐使用绝对路径。



也可以从多个目录提供内容：



如果值为false则禁用。

（3）filename

默认在惰性模式，每个请求结果都会产生全新的编译。



使用filename，可以只在某个文件时被请求时编译。

（4）compress

都启用gzip压缩。

（5）lazy

启用时，dev-server只有在请求时才编译包。

（6）watchContentBase

监视通过devServer.contentBase选项提供的文件。监视文件的改动将触发整个页面重新加载。默认禁用。

（7）inline

默认启用内联模式，即实时重载的脚本会被插入到bundle中，并且构建消息会出现在浏览器控制台。也可以使用iframe模式。

（8）clientLogLevel

使用内联模式(inline-mode)时，在开发工具(DevTools)的控制台(console)将显示消息。值有none、error、warning、info（默认值）。

（9）quiet

启用后，除了初始启动信息之外的任何内容都不会被打印到控制台，即，来自webpack的错误或警告在控制台不可见。

（10）noInfo

启用时， bundle的消息将被隐藏。错误和警告仍然会显示。

（11）stats

用于精确控制bundle信息展示。

（12）host

指定使用一个host。默认是localhost。只用在命令行工具使用。

（13）port

指定要监听请求的端口号，只用在命令行工具。

（14）headers

在所有请求中添加首部。

（15）https

默认情况为HTTP，也可以选择HTTPS。可以使用自签名证书，也可以自定义。

（16）hot

是否启用webpack的模块热替换特性。

* + 1. webpack-dev-middleware

**1.简介**

webpack-dev-middleware用于生成一个与webpack的compiler绑定的中间件，然后可以在express启动的服务app中调用这个中间件。主要作用为：

（1）通过watch mode，监听资源的变更，然后自动打包。

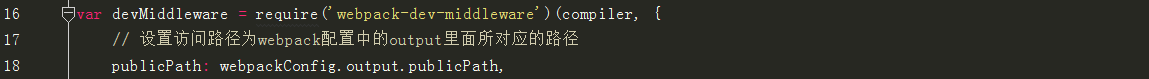
（2）快速编译，走内存。

（3）返回中间件，支持express的use格式。

**2.配置**

（1）配置publicPath

因为采用内存的方式，内存中的文件存储路径就是此处的publicPath。



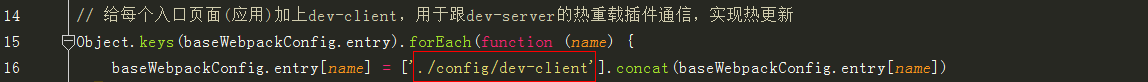
（2）将中间件引入到express server中。



* + 1. webpack-hot-middleware

webpack-hot-middleware是一个结合webpack-dev-middleware使用的middleware，它可以实现浏览器的无刷新更新（hot reload）。配置步骤如下：

（1）给每个entry后都增加一个hotMiddlewareScript。



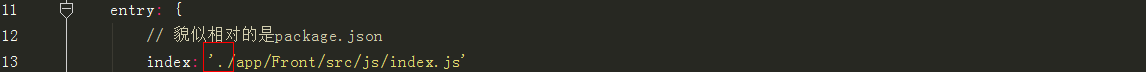
（2）加载中间件。

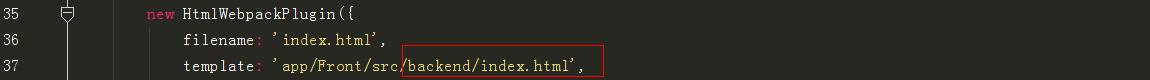




（3）注意，这里加的dev-client是相对于package.json的，包括entry、HtmlWebpackPlugin都是。







* 1. 其他

**1.安装**

全局安装webpack。安装后，可直接使用webpack命令（编译文件），也可使用webpack--help查看帮助信息。



在本项目下安装，前提是先有package.json文件。



也可以不全局安装，直接使用node\_modules/.bin/webpack执行命令。安装后，可以在package.json的scripts中写入命令配置。



**2.调整编辑器**



**3.生产环境构建**

运行(等价于运行)，会执行：1）使用UglifyJsPlugin压缩JS文件，2）运行LoaderOptionsPlugin，3）设置Node环境变量。

webpack自带了UglifyJsPlugin，支持所有的UglifyJS选项。可以使用cheap-module-source-map选项生成SourceMaps。

**4.命令行接口**

普通用法



指定配置文件进行构建。



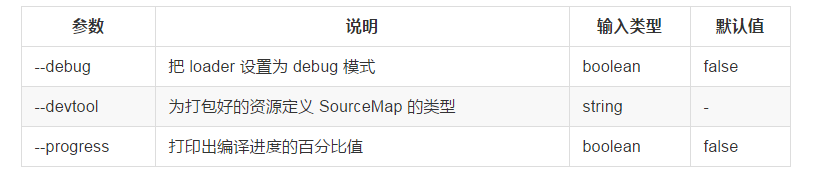
传入Webpack配置文件中用到的环境变量



以JSON格式输出Webpack的运行结果



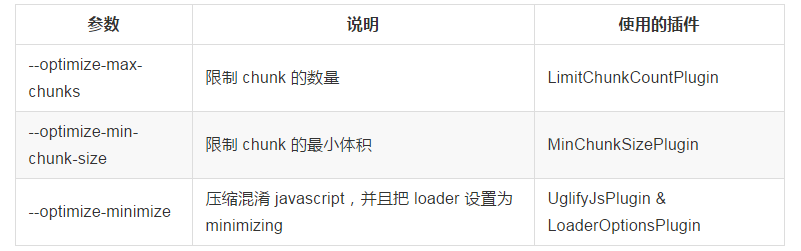
Debug配置



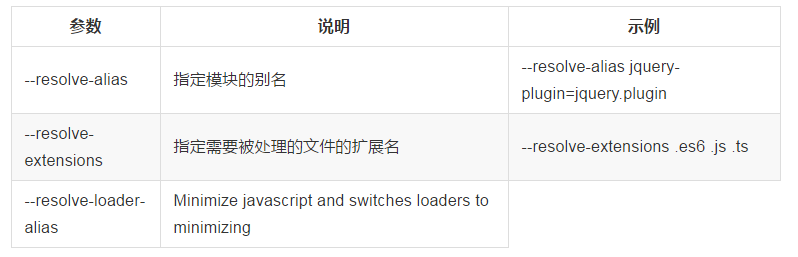
Watch配置



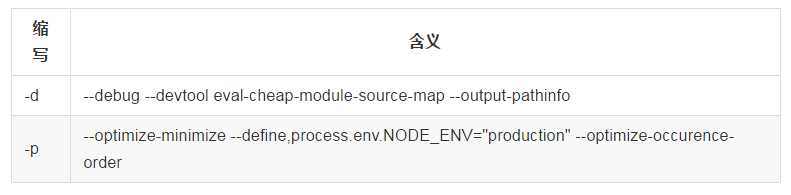
性能优化配置



Resolve配置



缩写



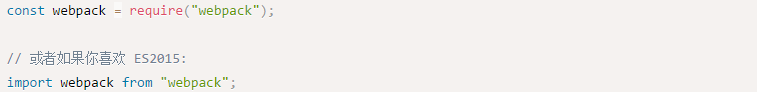
编译的性能数据



**5.Node.js API**

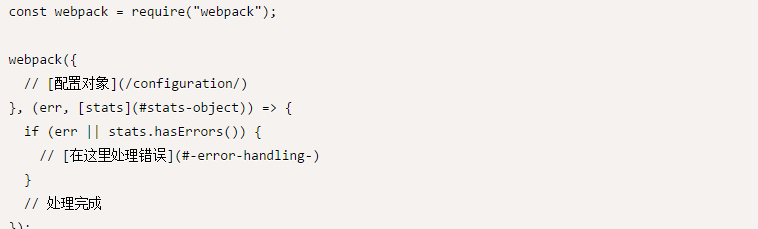
webpack提供了Node.js API，可以在Node.js运行时下直接使用。

（1）引入



（2）webpack函数

导入的webpack函数需要传入一个webpack配置对象。



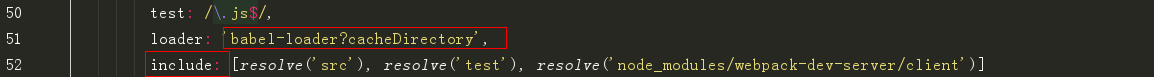
（3）Compiler实例

如果你不传入回调函数到webpack函数中，执行webpack就会得到一个webpackCompiler实例。可以通过它手动触发webpack执行器，或者是让它执行构建并监听变更。

* 1. 加载器
     1. babel-loader

（1）使用/\.js$/匹配。

（2）可以使用很多选项，但都不常用。其中，可以使用exclude跳过node\_modules目录或其他不需要的源代码。。



cacheDirectory用于缓存loader的执行结果。之后的webpack构建，将会尝试读取缓存，来避免在每次执行时，可能产生的、高性能消耗的Babel重新编译过程。默认值为false。

* + 1. style-loader/css-loader

使用import、require等引入css文件，就需要安装css-loader。此外，css-loader还会解析@import和url等。比较重要的选项有：（1）minimize是否压缩。（2）modules是否启用CSS模块规范，可以使用:local、:global等标识，还会使用base64代替类名。

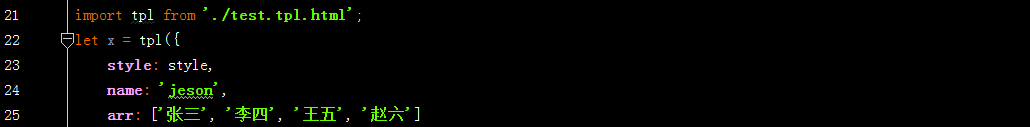
style-loader是将css文件以style的方式嵌入。

* + 1. ejs-loader

**1.配置**



**2.传入数据**



在模板中即可按EJS使用变量。对于图片，可以使用require加载，可以转为base64，前提是安装了url-loader。





ejs-loader和html-loader貌似不可以混用，否则图片没法正常显示。

在模板html文件中，最好不要使用es6语法，因为很难转码。

* + 1. file-loader/url-loader

**1.****file-loader**

（1）作用

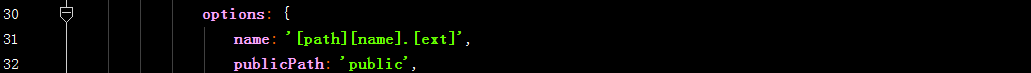
获取文件名，并能将文件复制到output。如下：



将会输出txt变量：。另外，在output处，也会将该文件复制过去，文件名也会相应修改：。

（2）文件名模板

可以使用文件名模板占位符等参数，如下：



占位符有：

[ext]：扩展名

[name]：基本名称

[path]：相对于context查询参数或者配置的路径

[hash]：内容的哈希值，默认为十六进制编码的md5

[<hashType>:hash:<digestType>:<length>]：可选配置，hashType值可以为sha1、md5、sha256、sha512、hex、base26、base32等；length为输出文件名长度。

（2）url-loader

对file-loader的封装，可以使用参数limit，当低于指定的限制时，可以返回一个data URL。

* 1. 插件

很多webpack中的对象都继承了Tapable类，暴露plugin方法。插件可以使用plugin方法注入自定义的构建步骤。基本上，每个插件的调用都在构建流程中绑定了回调来触发特定的步骤。

每个插件会在webpack启动时被安装一次。webpack通过调用插件的apply方法来安装它们，并且传递一个webpack compiler对象的引用。可以调用compiler.plugin来访问资源的编译和它们独立的构建步骤。

* + 1. html-webpack-plugin

https://github.com/jantimon/html-webpack-plugin#configuration

http://www.cnblogs.com/wonyun/p/6030090.html

**1.作用**

（1）生成html入口文件，可以是单个或多个。

（2）引入如script、link等外部资源时，动态添加每次compile后的hash，解决引用缓存。

**2.使用**

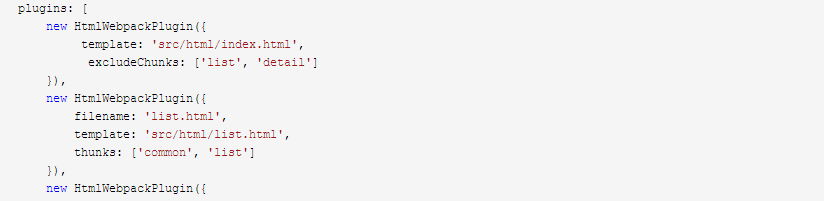




如果不配置任何选项，默认将webpack中的entry配置所有入口thunk和extract-text-webpack-plugin抽取的css样式都插入到文件指定的位置。



配置多个html页面



**3.配置项**

title：生成html文档title，但并不会替换指定模板文件中的title，除非html模板文件中使用了模板引擎语法，如下ejs模板语法形式：



filename：输出文件名称，默认为index.html，还指定目录位置，如html/index.html。配置的html文件目录是路径相对于webpackConfig.output.path，而link、script等则是相对于生成目录下。

template：本地模板文件的位置，支持如ejs、undersore、html等加载器。如果没有给模板文件指定任何loader的话，默认使用ejs-loader。

templateContent：指定模板的内容（html字符串），不能与template共存。

inject：向template或者templateContent中注入静态资源。值有：1）true或者body：插入到body元素的底部；2）head：插入到head元素中；3）false：所有静态资源不会注入到模板文件中。

favicon：favicon路径。

hash：true/false，是否为所有注入的静态资源添加webpack每次编译产生的唯一hash值。

chunks：允许插入到模板中的chunk。不配置则所有thunk都注入到模板中。在配置多个页面时，每个页面注入的thunk是不相同的，每个页面是相同的，因此需要设置进行区分。



excludeChunks：与chunks配置项相反。

chunksSortMode：thunk在插入到html前排序，可以为none就是不排序，function指定具体排序规则，auto基于thunk的id进行排序。默认auto。

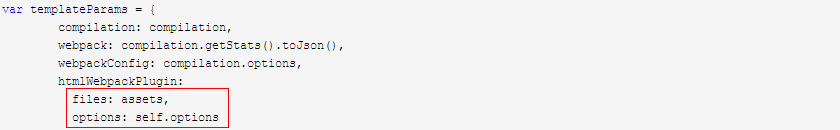
xhtml：是否渲染link为自闭合的标签。

showErrors：是否将错误信息输出到html页面中，默认true。

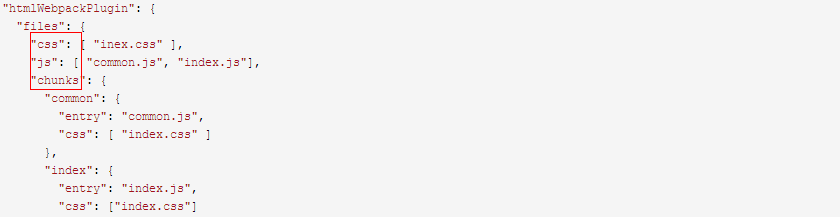
minify：是否压缩，使用html-minifier的选项。

**4.模板数据**

可根据这些数据，结合模板语法，动态生成html文件的内容。查看其源码，模板能够访问的到数据如下：



其中，compilation为所有webpack插件都可以访问到的内部编译对象；webpack为webpack的stats对象；webpackConfig可以获取webpack的相关配置项；htmlWebpackPlugin内的files为此次html-webpack-plugin插件配置的chunk和抽取的css样式，生成的数据格式如下：



因此，可以是用如下模板引擎来动态输出script脚本：



htmlWebpackPlugin内的options为传递给插件的配置项，因此可以传递数据进去。



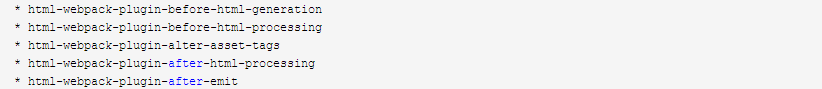


在引入页面内部，可以使用如下方式获取数据：



**5.插件事件**

（1）Async



（2）



**6.第三方扩展插件**

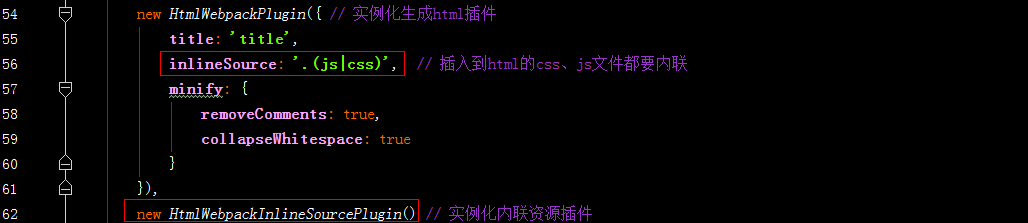
（1）html-webpack-inline-source-plugin

通过添加inlineSource属性，将js、css等代码内联进html中。

1）安装



2）使用



* + 1. CommonsChunkPlugin

用于多个入口提取公共模块，最终合成的文件能够在最开始的时候加载1次。配置项有：

（1）name或names（数组）：chunk 的名称，如果本选项被忽略，且async或children被设置，将选择所有chunk。

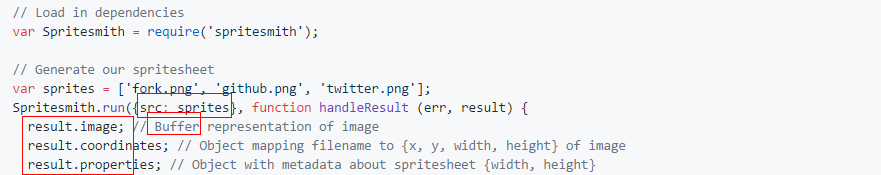
（2）filename：文件名模板。

（3）children：所有公共chunk 的子模块都会被选择。

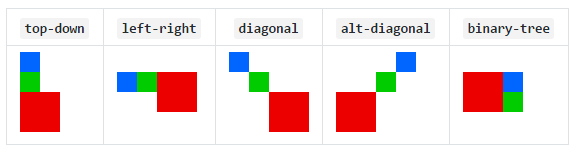
* + 1. webpack-spritesmith

**1.spritesmith**

webpack-spritesmith内部使用的是spritesmith库。通过这个库，可以直接组装图片精灵。这个库貌似现在使用buffer。

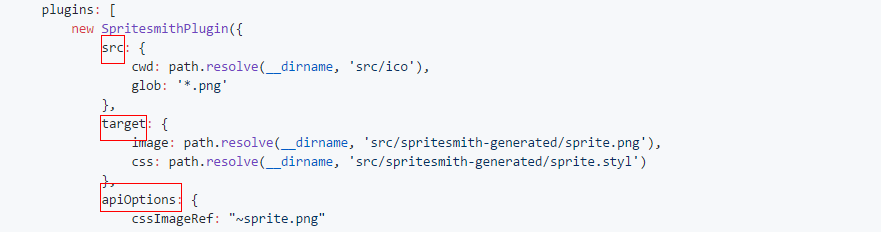


可以自己设置每一个图片部分的坐标、大小，也可以使用Algorithms选项。





**2.webpack-spritesmith**





配置选项有：

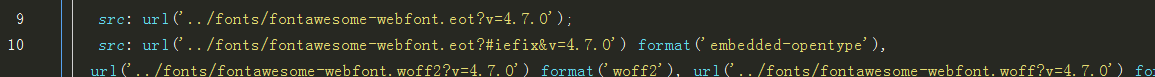
（1）apiOptions-cssImageRef：在API中引用生成的图像的路径。

（2）spritesmithOptions：spritesmith的选项。

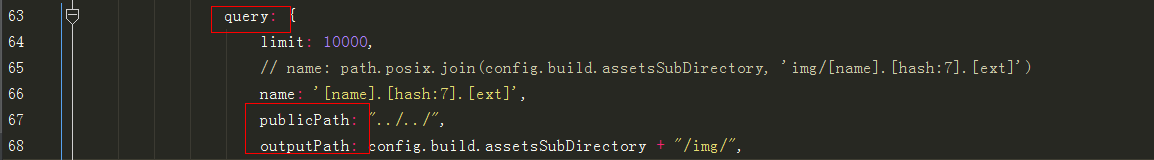
* 1. 使用问题解决

**1.打包css中引用的字体和图片**

如font-awosome，css中应用如下字体：



但貌似打包的时候，css路径转换（publicPath）和url-loader字体路径转换除了问题。可以使用如下方式解决：



2.如果js和css同名，那么必须加后缀，否则报错。



使用gulp-autoprefixer并不能转化opacity，需要手动写。