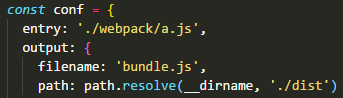
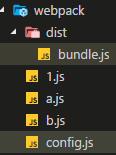
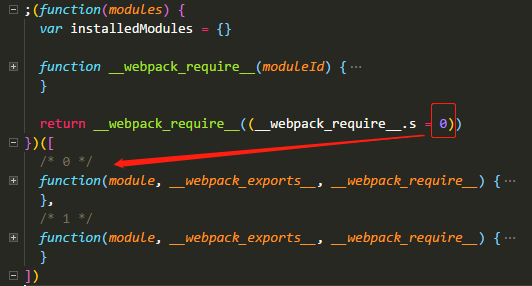
1. webpack工作机制

**1.最简单的例子**



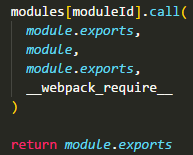


编译后的结构如下：

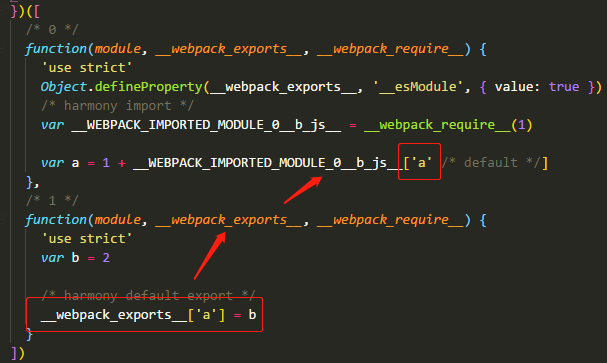


整体是1个立即执行函数。每个模块被封装到1个函数中，从而模拟1个模块。所有模块函数组成入参数组传入立即执行函数中。立即执行函数体内，会将入口文件模块传入\_\_webpack\_require\_\_中执行。\_\_webpack\_require\_\_.s就是是启动模块对应的index。

\_\_webpack\_require\_\_用于根据moduleId去加载模块。由此可见每个模块函数入参结构为固定格式。



\_\_webpack\_require\_\_返回的就是传入模块中的exports。



**2.打包流程**

（1）初始化参数

从配置文件和Shell语句中读取、合并参数。

（2）开始编译

使用上一步得到的参数，初始化Compiler对象；**“加载”**配置参数中所有的插件；执行对象的run方法开始执行编译。

（3）确定入口

根据配置中的entry找出所有的入口文件。

（4）编译模块

从入口文件出发，**“递归”**调用所有配置的Loader对模块进行编译，直到所有依赖文件均被编译。

（5）完成模块编译

编译完所有模块后，即得到“每个模块、模块编译后的chunk，以及它们之间的依赖关系”。

（6）输出资源

根据入口和模块之间的依赖关系，组装成一个个包含多个模块的Chunk，再把每个Chunk转换成一个单独的文件加入到输出列表。

（7）输出完成

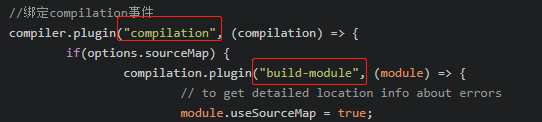
确定好输出内容后，根据配置确定输出的路径和文件名，把文件内容写入到文件系统。

**3.插件机制**

插件是一个具有apply方法的js对象。apply方法会被webpack的compiler对象调用。compiler对象可在整个编译生命周期内访问。

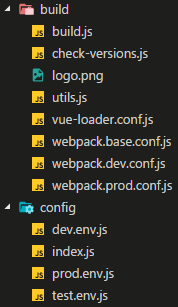
插件通过挂载到webpack事件钩子上，在不同生命周期实现功能。



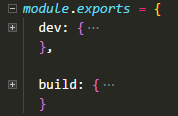


需要说明的是：执行插件程序并不是串行的，而是可以在插件代码中定义是否串行、并行以及依赖。插件之间还可以传递参数。

1. 配置文件结构



* build文件夹保存的webpack的配置文件，而config文件夹中的文件只是webpack配置文件用到的不同环境下的配置。
* config文件只有index.js放主要配置，dev.env.js、prod.env.js、test.env.js都只有1个NODE\_ENV配置项。



* webpack最终用到的配置文件是build目录下的webpack.dev.conf.js和build.js。其他文件是为这两个文件所用。





其中，base文件为两者通用配置。prod文件为生产环境配置，但build.js引入prod文件作为webpack的配置项，同时还有删除dist目录、提示编译成功、失败等功能。

check-versions用于在build.js中检查编译环境，如果没有按照node、npm等会报错，并终止编译。

vue-loader.conf为vue-loader所用到的配置文件。

1. 基本配置

webpack是前端资源模块化管理和打包工具。它将所有前端资源都视为1个个模块，如js模块、css模块、图片模块等。然后将这些模块按照配置文件设置的规则进行管理。

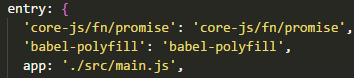
webpack配置文件的一些基本概念：

**1.context、入口、出口**

* context设置工作根目录。如下，将项目根目录设为根目录，之后的entry还是后面插件，如果使用相对路径，都是基于context。

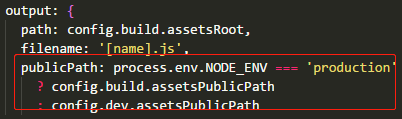


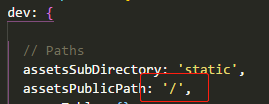
* 有多少入口，就有多少出口。

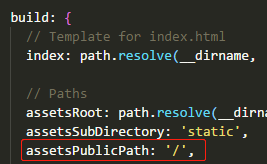




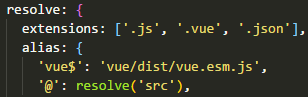
* 出口的最终路径由publicPath和path一起决定。没有publicPath，那就只由path决定。
* publicPath的意义：如，在localhost环境下，css文件中可以使用“./test.png”来加载本地图片。此时，不写publicPath。但在生产模式下，“test.png”文件可能会定位到CDN上。这就意味着在生产环境需要更新所有文件里的url为CDN的路径。显然，手动更新特别麻烦，这时就可以添加publicPath配置，再结合一些插件自动更新这些url。







**2.resolve**

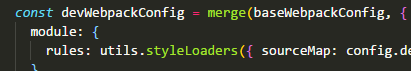


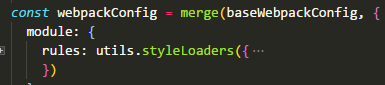
extensions用于省略后缀，alias用于设置别名。其中vue$后面用到。

**3.module**

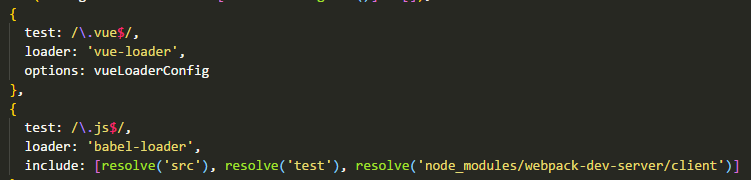
正如前面所说，项目用到的所有资源都被webpack视为了模块。module就用于设置如何处理这些模块。在vue的webpack配置中，只涉及到了rules这1个子配置项。





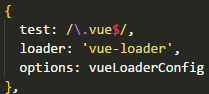
rules为1个rule规则数组。每个rule可以分为3部分：条件、结果、嵌套规则。按照rule的匹配规则，能够对模块应用加载器。



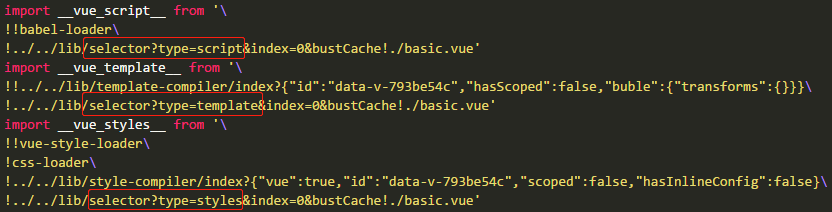
1. vue-loader

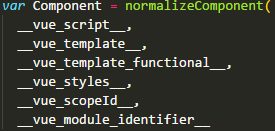
**1.大概流程**

Vue Loader的作用是将.vue文件的Vue组件编译成普通的JavaScript模块。



如下，.basic.vue文件会编译为1个vue组件。



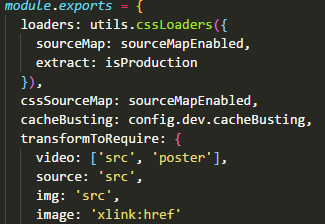


selector是vue-loader的lib下的loader，用于读取.vue中对应的模块。然后分别交给后续loader处理。

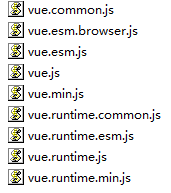
正是因为js部分被babel-loader处理了，所以.vue文件默认支持ES2015语法。

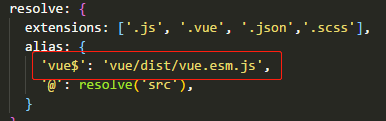
**2.配置文件**

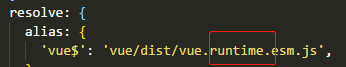
如下，loaders为css部分loaders的配置（后续loader的配置）。transformToRequire用于指示Vue Loader将那些标签的何种属性视为模块加载。

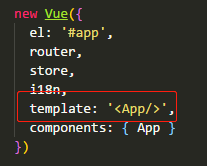
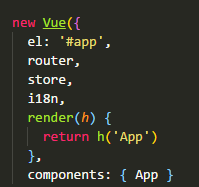


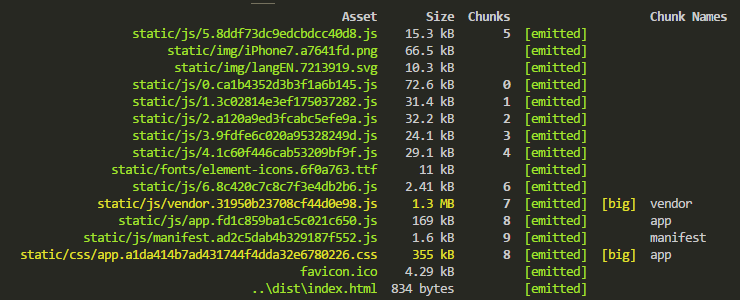
**3.vue的不同版本**

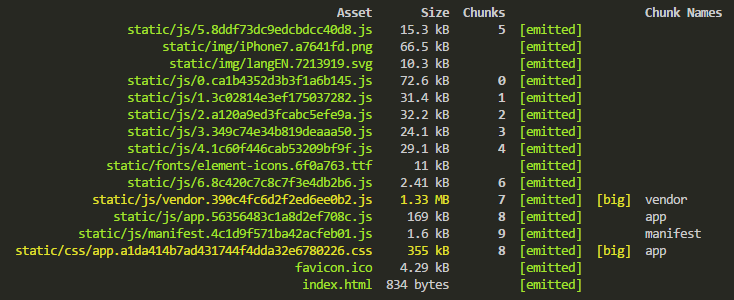




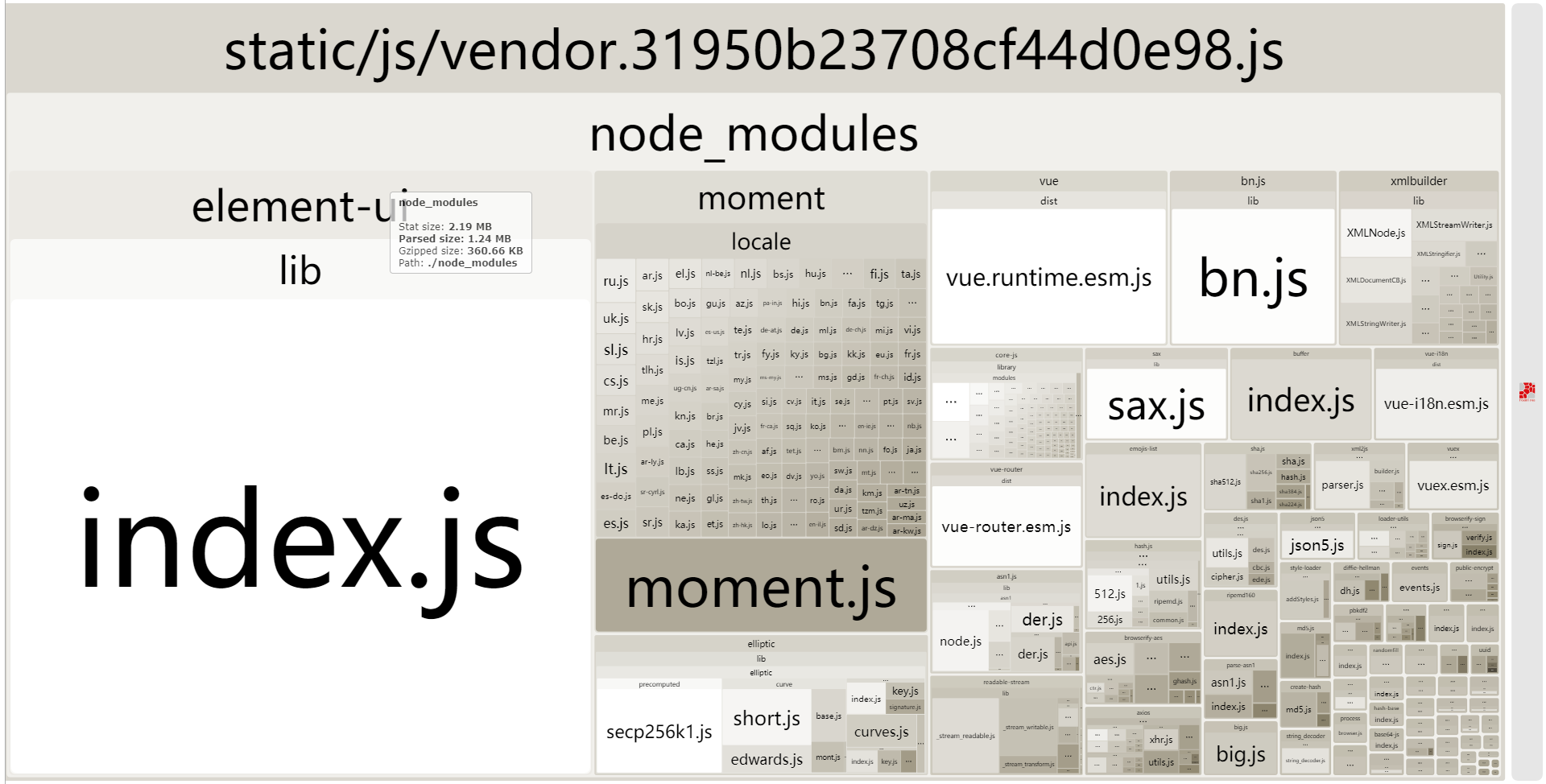


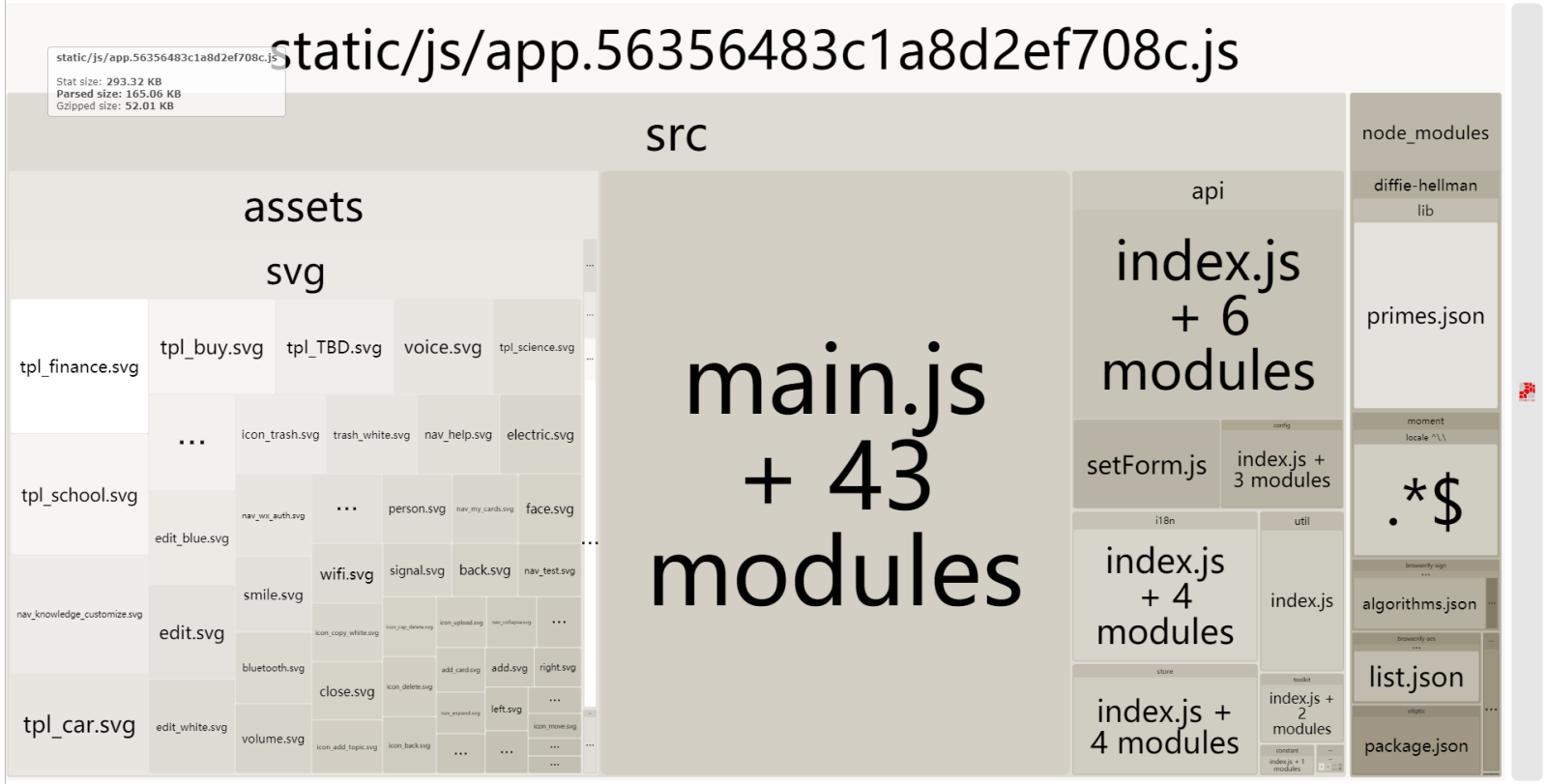












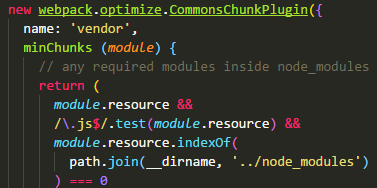
1. 一些细节

**1.分离第三方js代码**

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。

一共抽离了3个：

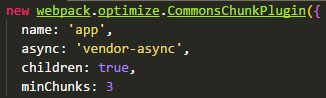
* 如果是node\_modules中的js模块，返回true，抽离到vendor中。



* 因每次构建时，webpack会生成webpack runtime代码，用来帮助webpack完成其工作。所以需要将这些runtime代码抽离为manifest。runtime及manifest具体内容为模块在加载和解析中的映射关系。

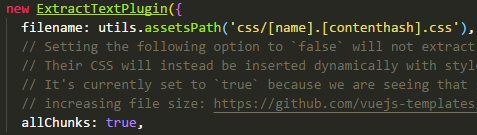


* 因多个chunk会有公共依赖，因此，需要将分离出来的chunk的公共依赖抽离为1个单独的chunk。



**2.将.vue文件外的css抽离出来**

否则，将作为js代码的一部分。



**3.异步组件与懒加载**

（1）Vue异步组件

和懒加载不是同一个事，仅指在需要该组件的时候才会实例化。好处：如tab、弹窗等，有的用户可能并不一定会点击，因此可以提高初次渲染效率。



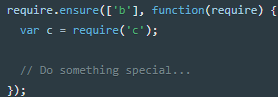
（2）模块加载



ES2015 loader规范定义了import()方法，用于在运行时动态地加载ES2015模块。

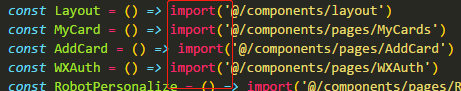


require.ensure()是webpack特有的，已经被import()取代。



（3）懒加载

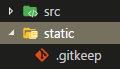
webpack将import()和require.ensure()视为代码中的逻辑断点。不需要额外的webpack配置，自动将需要引入的模块单独切割为1个chunk。



**4.静态文件**

对于在css或.vue中的图片等，如果使用相对路径，会被解析为依赖模块，会被webpack输出配置所自动生成的URL所代替。

如果使用绝对路径，则不会被webpack处理。此时的静态文件应该放在与src同级的static目录下。



在webpack配置文件有如下配置，用于将该文件夹下的静态文件复制到config所配置的目录下。

