# 一 webpack简介

## 1 简介

网页引入过多的静态资源会造成网页加载速度变慢（多次请求），且需要处理复杂的依赖关系。一般采用合并、压缩、精灵图等方式来解决上述问题，比如require.js、sea.js。但是ES6已经原生支持模块化了，使用原生开发，然后编译、打包更能体验ES6带来的便捷。

ES6、Less却不能直接被浏览器支持！WebPack可以看做是模块打包机，可以分析你的项目结构，找到JavaScript模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript等），并将其转换和打包为合适的格式供浏览器使用。

webpack的优点：

1. 对 CommonJS 、 AMD 、ES6的语法规范都做了兼容

2. 对js、css、图片等资源文件都支持打包

3. 串联式模块加载器以及插件机制，使用更加灵活，扩展性更强

4. 有独立的配置文件webpack.config.js

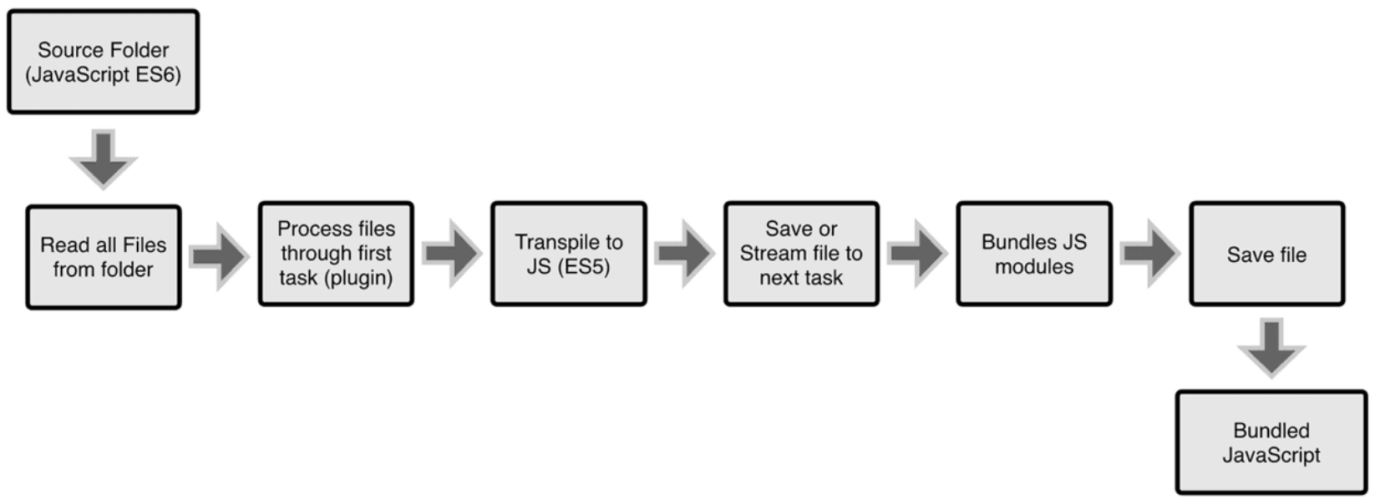
5. 可以将代码切割成不同的chunk，实现按需加载，降低了初始化时间

6. 支持 SourceUrls 和 SourceMaps，易于调试

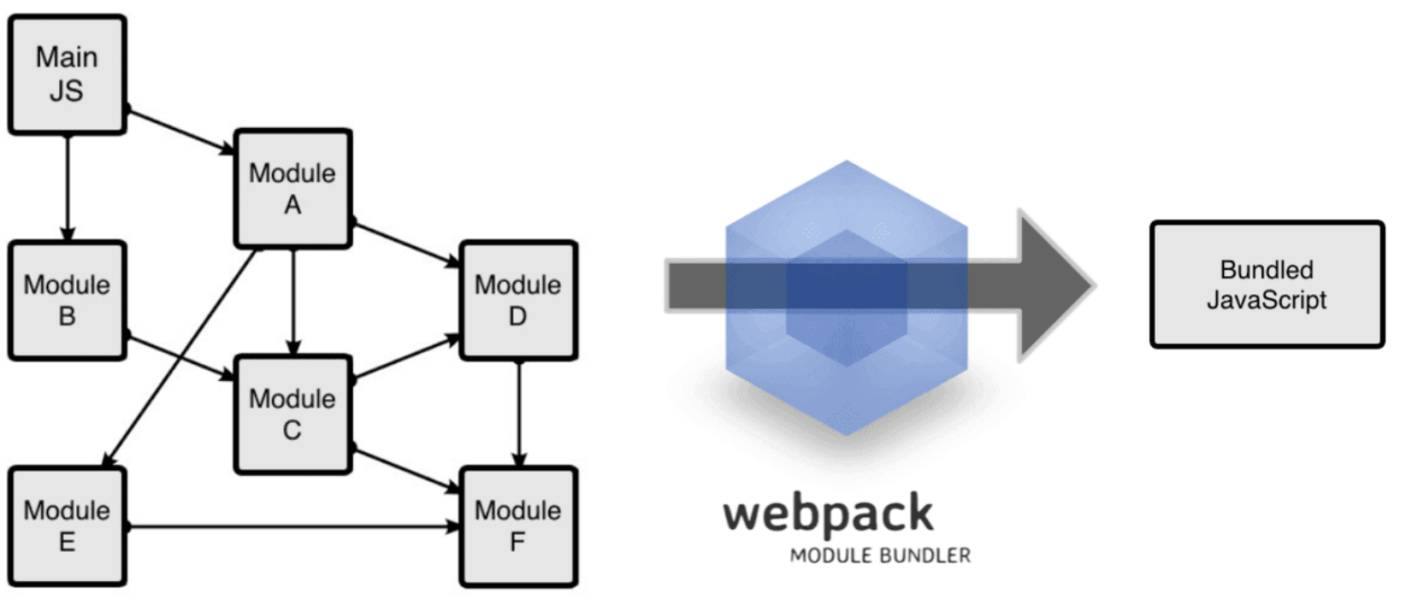
7. webpack使用异步 IO 并具有多级缓存。这使得 webpack 很快且在增量编译上更快。

## 2 webpack与gulp区别

Gulp 的定位是 Task Runner, 用来跑一个一个任务，但是没有解决 js module 的问题。其工作方式是：指明对某些文件进行类似编译、组合、压缩等任务的具体步骤，之后gulp工具可以自动替你完成这些任务。



Webpack工作方式：把项目当做一个整体，通过一个给定的主文件（如index.js），Webpack将从这个文件开始找到项目的所有依赖文件，使用loaders处理它们，最后打包为一个（或多个）浏览器可识别的JavaScript文件。



完美解决方案：gulp制作工作流，webpack进行项目构建（打包、压缩、合并、混淆）

## 2 安装

首先需要： npm init

全局安装：npm install webpack -g

本地安装：npm install webpack -D #一些插件需要用到本地webpack

注意事项：

如果安装webpack3，需要附带版本 npm install webpack@3

如果安装最新的webpack4，则额外需要安装npm install webpack-cli -D

在webpack4中，cli命令从webpack中分离了出来。

## 3 项目结构



## 4 打包

输入命令： webpack #此时将会在dist目录中生成一个打包后的文件 main.js

完整命令： webpack index.js main.js #将src下的index打包为dist下的main

webpack4踩坑注意：

1 输入webpack提示No such file，因为本地安装webpack-cli，仍然会去全局查找命令。

解决办法一：不使用本地安装webpack-cli 使用全局安装，不推荐；

解决办法二：使用npm脚本，第二章2；

2 webpack4会有警告提示，需要设置mode为produciont或者development。

解决办法：使用命令行参数：webpack --mode production

# 二 webpack使用

## 1 webpack命令

webpack命令会默认将 src目录下的 index.js打包到dist目录下，生成main.js。如果此时我们在dist目录下有个html引入了该main.js文件，那么就等于直接使用了。

webpack带参数命令： webpack 源文件地址 目的文件地址

常用命令参数：

--open 打包后自动打开浏览器

--port 设置端口

--contentBase 打开目的文件目录

--hot 浏览器异步更新 主要针对样式的更改

--config a.js 指定配置文件，默认指定根目录下的 webpack.config.js

## 2 npm脚本

命令参数在webpack命令中直接使用很麻烦，在package.json的script中添加脚本：

"dev": "webpack --open --port 3000 --contentBase src --hot"

使用： npm run dev 即可快速完成上述webpack命令任务。

注意： webpack4会出现mod警告，命令额外加入： --mod development即可

## 3 webpack配置文件

使用npm脚本仍然会需要在脚本中输入大量的配置参数，造成脚本臃肿，我们可以直接在脚本中指定一个配置文件，将命令参数书写在配置文件中，这样便简单明了了。

"dev": "webpack --config webpack.config.js --mod development"

注意：即使脚本中没有书写config，这时候如果存在webpack.config.js，该配置也会生效，因为webpack会默认查找该名字的配置文件。

在根目录下创建webpack.config.js文件，内容如下：

const path = require('path');

module.exports = {

entry:path.resolve(\_\_dirname,'src/index.js'),  *//入口*

output: {

path: path.resolve(\_\_dirname, 'dist'),      *//输出文件夹*

filename: 'bundle.js'           *//输出文件名*

}

};

输入构建命令：npm run dev

#会在dist目录中生成一个名为bundle.js的打包后文件，证明配置文件生效了（因为没走配置文件会生成默认的main.js）

## 2 webpack-dev-server

使用原生的webpack命令，每次修改完代码都要重新打包，需要服务器来实现自动化：

npm i webpack-dev-server -D

注意：该工具和webpack的用法完全一样，由于只有全局安装的工具才能在终端使用，该命令本地安装后不能直接使用，需要在package.json文件中配置一个命令：

"dev": "webpack-dev-server --open --port 3000 --contentBase src --hot"

命令与参数解释：

webpack-dev-server 在 localhost:8080 建立一个 Web 服务器

--devtool eval 为代码创建源地址，报错时精确地定位到文件和行号

--progress 显示合并代码进度

--colors 命令行中显示颜色！

--content-base 默认webpack-dev-server会为根文件夹提供本地服务器，设置了该目 录，则从该目录提供本地服务

--open 自动打开浏览器

--inline 设置为true，当源文件改变时会自动刷新页面

--historyApiFallback依赖于HL5 history API，设置为true，在开发单页应用时所有的跳转 将指向index.html

注意： 打包生成的bundle.js文件位于内存中，可以认为位置是和dist src平级！

所以html的引入方式也要相应修改为 /bundle.js

脚本+配置文件方式实现上述命令：

"dev": "webpack-dev-server --mode development"

配置文件webpack.config.js内容：

const webpack = require('webpack');

const path = require('path');

module.exports = {

entry: path.resolve(\_\_dirname, 'src/index.js'),

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'bundle.js'

},

devServer:{

hot: true,

open: true,

port: 3000,

inline: true

}

}

*//设置hot为ture后才能开启热更新插件*

new webpack.HotModuleReplacementPlugin();

## 3 插件 html-webpack-plugin案例

插件的使用步骤：npm安装插件--配置文件引入插件--配置文件的plugins添加参数

由于使用 --contentBase 指令的过程比较繁琐，需要指定启动的目录，同时还需要修改index.html中script标签的src属性，所以推荐使用html-webpack-plugin插件配置启动页面。

运行： npm i html-webpack-plugin -D

注意：此时可以将dist中index.html中script标签注释掉，因为该插件会自动把bundle.js注入到index.html页面中！

配置文件如下：

const webpack = require('webpack');

const path = require('path');

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

module.exports = {

entry: path.resolve(\_\_dirname, 'src/index.js'),

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'bundle.js'

},

devServer:{

hot: true,

open: true,

port: 3000,

progress: true

},

plugins: [

new htmlWebpackPlugin({

*//模板路径*

template: path.resolve(\_\_dirname,'src/index.html'),

filename: 'index.html' *//生成文件名*

}),

*//热更新插件*

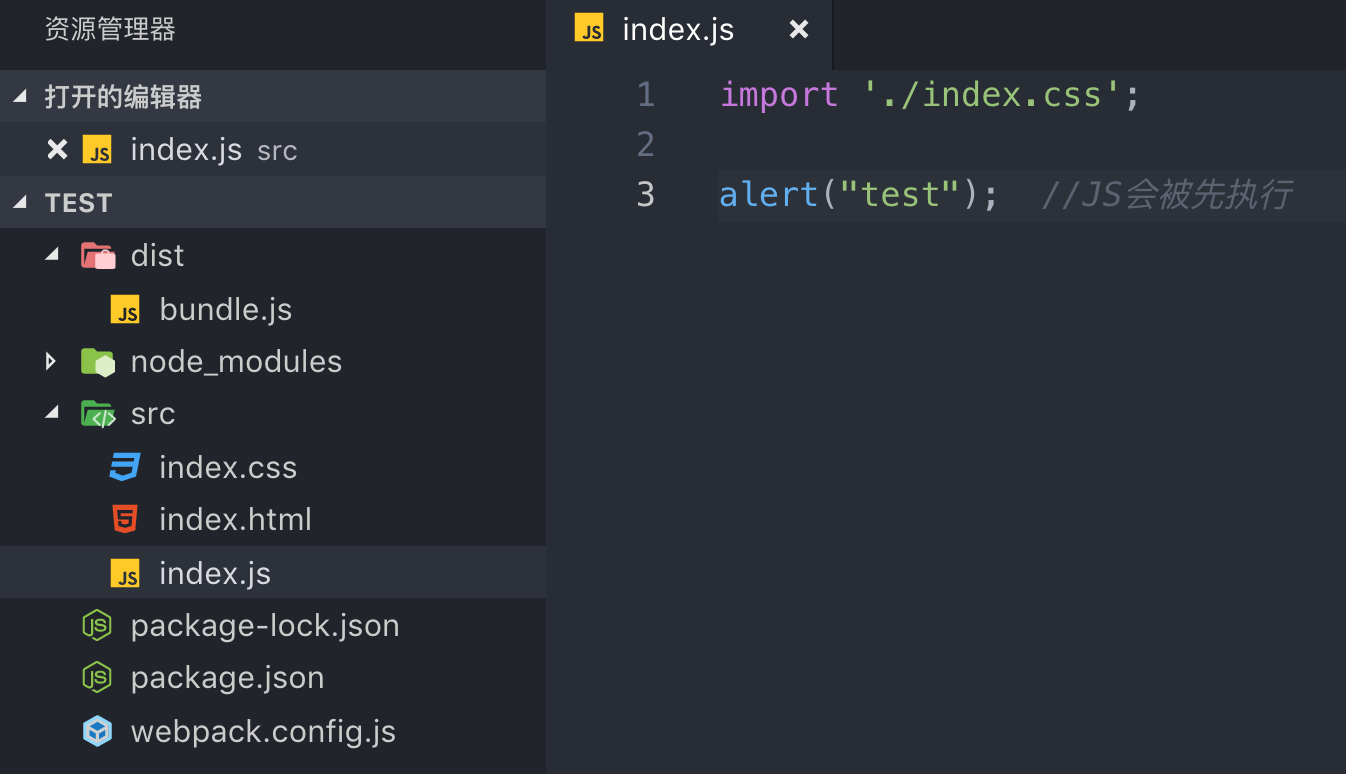
new webpack.HotModuleReplacementPlugin()

]

}

## 4 加载器 css-loader案例

CSS也可以像模块一样被引入，而不是像以前需要使用link：



但是webpack只能处理JS文件，处理CSS文件需要第三方加载器：

npm i style-loader css-loader -D

添加配置：

module:{

rules: [

{ test: /\.css$/, use: ['style-loader', 'css-loader'] }

]

}

use 表示使用哪些模块来处理test所匹配到的文件，按顺序从前到后调用；

css-loader会遍历 CSS 文件，然后找到 url() 表达式然后处理他们，style-loader 会把原来的 CSS 代码插入页面中的一个 style 标签中。

## 4 全局变量声明插件 jquery案例

webpack可以将 jquery中的$ 声明为全局变量，使用步骤如下：

安装jquery：npm install jquery -S

配置$:

new webpack.ProvidePlugin({

$: 'jquery',

jQuery: 'jquery'

})

在index.js中使用jquery：

$('#box').css("background-color","red");

## 5 跨域代理

由于跨域问题，本地文件无法发送ajax，webpack可以将本地文件的ajax进行转发：



## 6 Source Maps 调试

打包后的文件很难找到出错的位置，利用source maps可以解决。

devtool的配置如下：

devtool: 'eval-source-map',

可选项如下：

source-map：

在一个单独的文件中产生一个完整且功能完全的文件，会减慢打包速度；

cheap-module-source-map：

在一个单独的文件中生成一个不带列映射的map，提高了打包速度，但是浏览器开发 者工具只能对应到具体的行，不能对应到具体的列（符号）

eval-source-map：

使用eval打包源文件模块，在同一个文件中生成干净的完整的source map。这个选项 可以在不影响构建速度的前提下生成完整的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 定不要启用这个选项；

cheap-module-eval-source-map：

这是在打包文件时最快的生成source map的方法，生成的Source Map 会和打包后的 JavaScript文件同行显示，没有列映射，和eval-source-map选项具有相似的缺点；

官方文档中常用关键字的解释：

eval： 使用eval包裹模块代码

source-map： 产生.map文件

cheap： 不包含列信息，也不包含loader的sourcemap

module： 包含loader的sourcemap（比如jsx to js ，babel的sourcemap）

inline： 将.map作为DataURI嵌入，不单独生成.map文件（这个配置项比较少见）

eval模式是使用eval将webpack中每个模块包裹，然后在模块末尾添加模块来源//# souceURL， 依靠souceURL找到原始代码的位置。

“在cheap-source-map模式下sourcemap不包含列信息，也不包含loaders的sourcemap”这里的“column-mappings”就是代码列数的意思。

# 三 webpack与框架

## 1 webpack与babel

### 3.1.1 babel-loader基础使用

注意：webpack本身只是一个打包工具，不具备任何功能。各类打包、编译工具都是由加载器或者第三方工具完成的，比如编译ES6,react都是由babel这个第三方工具完成的，webpack可以使用babel-loader加载这个第三方工具。

安装babel相关工具： 提前注意事项：babel与 babel-loader的版本一定要一致！！！

npm install -D babel-core@6 babel-loader@7 babel-preset-env@1

配置webpack：在rules数组中添加新的一条规则：

{

test: /(\.jsx|\.js)$/,

use: {

loader: "babel-loader",

options: {

presets: ["env"]

}

},

exclude: /node\_modules/

}

exclude代表不编译哪些文件，presets表示babel编译规则，我们通常可以将该规则写在babel自己的配置文件中，注释掉webpack中的options选项，新建.babelrc,书写如下配置：

{

"presets": ["env"]

}

注意：babel-preset-env是个泉新新的babel语法插件，包含了所有ES\*的语法，如果我们不需要更多的新特性，可以使用 babel-preset-es2015这些语法插件代替。

尤其注意：如果不按照上述的版本安装，会出现版本错误，因为webpack4的babel-loader已经不再使用 babel-core，新版的babel-loader依赖的组件是：@babel/core。所以：

安装的babel相关包为：@babel/core @babel/preset-env @babel/preset-react，

babel的配置也为：{ "presets": ["@babel/env", "@babel/react"]}

### 3.1.2 babel转换函数

在上述的配置中，babel只是将一些es6，es7的语法转换成符合目标的js代码，但是如果我们使用一些JS特性或方法，比如Generator, Set，babel并不能转换为低版本浏览器识别的代码。

解决方案一 babel-polifill：

安装：npm install --S babel-polifill ##这是生产依赖

在应用入口顶部引入：import 'babel-polifill'; //也支持commenJS规范

在webpack中加入入口数组：entry: ["babel-polifill", "./app.js"]

注意：polifill用于应用开发中，会添加相应变量到全局，所以会污染全局变量。

解决方案二 runtime transform：

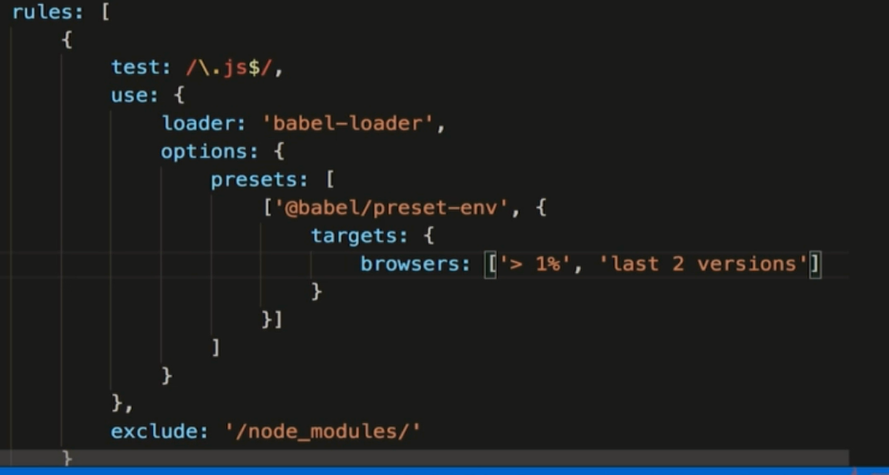
runtime transform也是一个插件，它与polifill有些类似，但不污染全局变量，所以经常用于框架开发。

npm install -D babel-plugin-transform-runtime

npm install -S babel-runtime

babel配置中添加："plugins": ["transform-runtime"]

### 3.1.2 babel针对不同浏览器编译

不同的浏览器，编译出不同的代码，比如let const这样在谷歌浏览器本身支持较好，无需编译：

## 2 配置react

安装react：npm install -S react react-dom

安装babel：和配置babel一样，注意版本，额外安装react相关语法插件即可

npm install -D babel-core@6 babel-loader@7 babel-preset-env@1 babel-preset-react@6

配置webpack：和配置babel一样

{

test: /(\.jsx|\.js)$/,

use: {

loader: "babel-loader"

},

exclude: /node\_modules/

}

同样babel配置需要写在.babelrc中：

{

"presets": ["env", "react"]

}

再次强调：

尤其注意：如果不按照上述的版本安装，会出现版本错误，因为webpack4的babel-loader已经不再使用 babel-core，新版的babel-loader依赖的组件是：@babel/core。所以：

安装的babel相关包为：@babel/core @babel/preset-env @babel/preset-react，

babel的配置也为：{ "presets": ["@babel/env", "@babel/react"]}

## 3 配置vue