

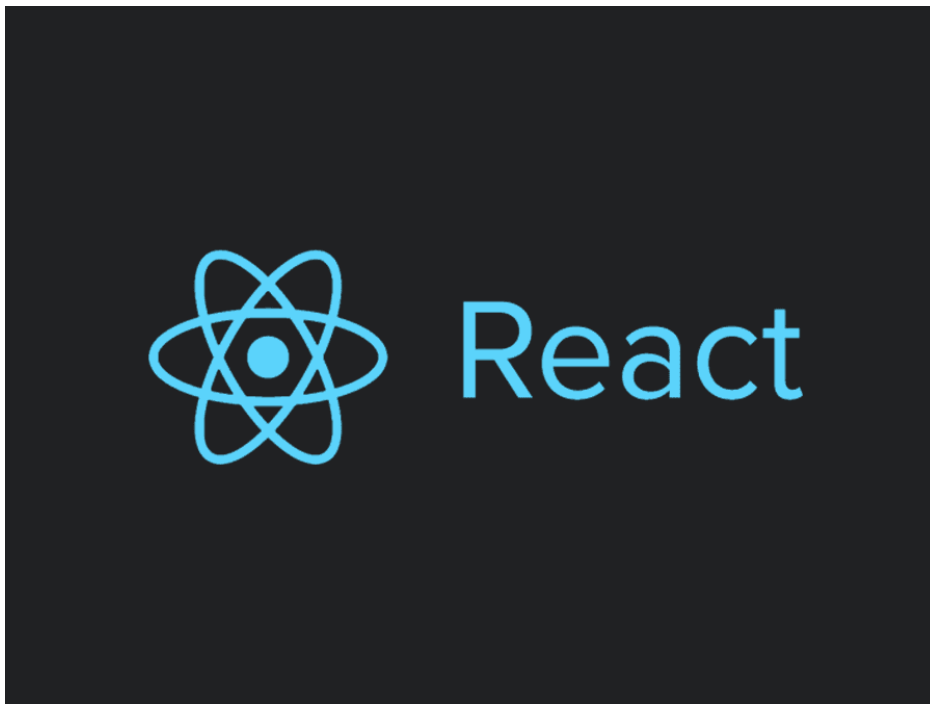
# 从零开始，教你用Webpack构建React基础工程



作者 waka (/u/d184d3ab9060) [+ 关注](#)

2016.10.18 09:19\* 字数 3890 阅读 15583 评论 46 喜欢 44 赞赏 2

(/u/d184d3ab9060)



## 20170315更新

由于webpack已更新至2.0版本，许多配置项都已改变，所以可能本文有些项已经过时了

推荐大家使用facebook官方构建工具facebookincubator/create-react-app  
(<https://github.com/facebookincubator/create-react-app>)来创建React基础工程

因为webpack的配置太复杂了，我们没有必要全部了解，它就是个工具，说不定哪天又变成别的工具了。真正需要明白的是它的原理，剩余的东西，对着文档来或者用别人做好的轮子就行，把时间花在真正值得花的地方。

## 前言

随着前端代码越来越多，越来越复杂，整个工程变得越来越难以管理。所以，前端工程化已是必然的趋势，已经是2016年了，还在用手动添加依赖吗？快来学习用构建工具来管理web项目吧。

就像是Android工程使用gradle来进行项目构建一样，在前端世界里，现在最流行的工具就是webpack。

而现在最流行的前端框架，分别是Google的Angular.js，Facebook的React.js，还有我们国人自己的尤雨溪大神的Vue.js。因为最近Angular和Vue发布了2.0版本，区别较大，所以推荐学习更为稳定的React.js。

自己是阅读了以下两篇文章：



入门Webpack，看这篇就够了 (<http://www.jianshu.com/p/42e11515c10f>)  
React+Webpack快速上手指南 (<http://www.jianshu.com/p/418e48e0cef1>)

两篇文章写的非常好，但中间还是有些地方不是很清楚，需要结合起来看，所以写下这篇文章，总结自己实际开发中遇到的坑。乐于分享，收获更多。

## 开发环境

推荐使用JetBrain的Webstorm，有强大的代码提示，支持JSX和ES6语法；可以试用一个月，相对于开发效率的巨大提升，一年300块真的不算太贵。

**我们将会使用npm来下载和构建依赖，需要提前安装node.js，如果没有安装可以参考这篇文章安装node.js和npm (<http://www.jianshu.com/p/0299e8f36976>)**

**使用npm时需要将终端定位到项目的根目录下**

React作为Facebook全家桶的老大哥，今后肯定会使用Facebook新出的开源依赖工具Yarn (<https://github.com/yarnpkg/yarn>)，推荐好奇的同学们去试（cai）试（keng）。我们还是先使用稳定的npm，毕竟依赖工具只是辅助，以后可以迁移到yarn上，重要的是代码。

源码地址：[https://github.com/BadWaka/ReactDemo\\_webpack](https://github.com/BadWaka/ReactDemo_webpack)

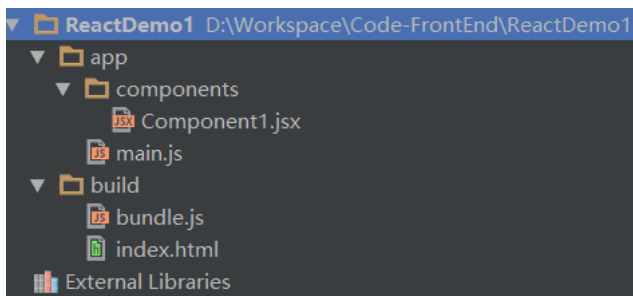
([https://github.com/BadWaka/ReactDemo\\_webpack](https://github.com/BadWaka/ReactDemo_webpack))

如果觉得有帮助，点个Star呗

让我们开始吧

## 项目结构

新建一个新项目，命名为ReactDemo，将项目目录组织如下：



- 其中app是开发目录
  - components用来放React的组件
    - Component1.jsx是React特有的jsx文件
  - main.js是入口文件，用来将React组件放在真正的Html中
- build是输出目录
  - index.html是最终要呈现的页面文件，代码如下



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>ReactDemo1</title>
</head>
<body>
<!-- 要插入React组件的位置-->
<div id="content"></div>
<!-- 引入bundle.js-->
<script src="bundle.js"></script>
</body>
</html>
```

- bundle.js是app文件夹下的文件经webpack打包后最终生成的供浏览器解析的js文件，这个不需要手动进行配置，甚至你不需要新建它，由webpack自动生成。

## 创建package.json文件

package.json是一个标准的npm说明文件，里面蕴含了丰富的信息，包括当前项目的依赖模块，自定义的脚本任务等；

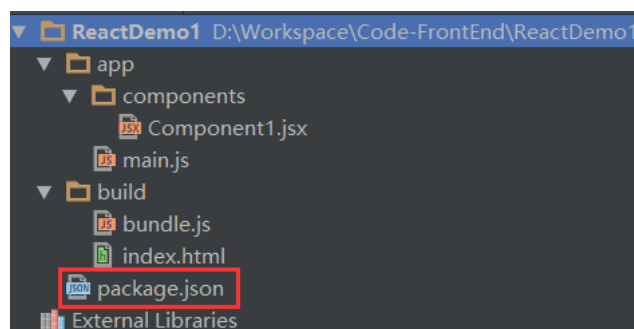
在工程根目录打开命令行（win10中使用ctrl+shift+右键可以在菜单中看到"在此处打开命令窗口"的选项），输入 `npm init` 可以自动创建package.json文件；

输入命令后，终端会问用户一系列的问题，如果不需要在npm中发布你的模块，一路回车默认即可（注意：name需要小写）；



```
npm
Press ^C at any time to quit.
name: (ReactDemo1) react_demo1
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author: waka
license: (ISC)
About to write to D:\Workspace\Code-FrontEnd\ReactDemo1\package.json:
{
  "name": "react_demo1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo `Error: no test specified` && exit 1"
  },
  "author": "waka",
  "license": "ISC"
}
Is this ok? (yes) _
```

确认后输入yes即可看到package.json文件创建完成



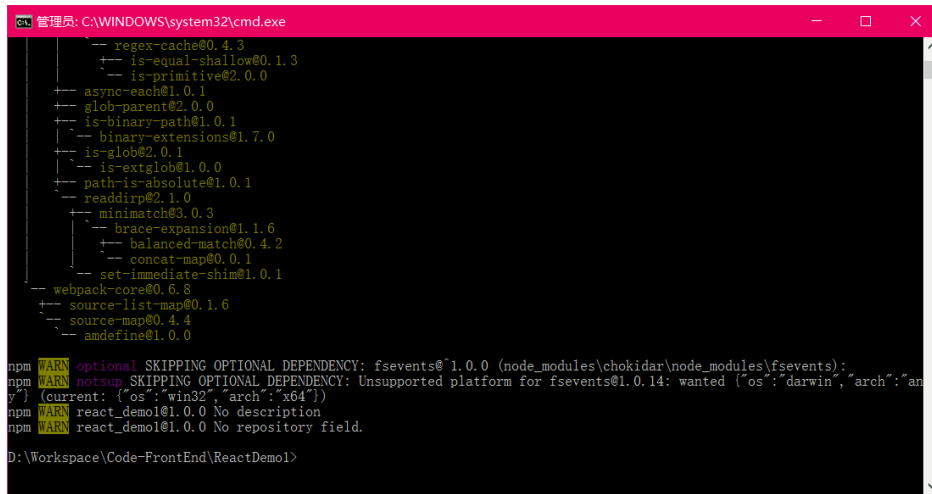
## 安装webpack



```
//全局安装webpack，优点是打包时可以直接输webpack命令
npm install -g webpack

//在本项目中安装webpack，--save-dev的意思是将依赖写入项目的package.json文件
npm install --save-dev webpack
```

在命令行中输入 `npm install --save-dev webpack`，在本项目中安装webpack作为依赖包



```
管理员: C:\WINDOWS\system32\cmd.exe
-- regex-cache@0.4.3
  +- is-equal-shallow@0.1.3
  -- is-primitive@2.0.0
+- async-each@1.0.1
+- glob-parent@2.0.0
+- is-binary-path@1.0.1
  -- binary-extensions@1.7.0
+- is-glob@2.0.1
  +- is-extglob@1.0.0
  +- path-is-absolute@1.0.1
+- readdirp@2.1.0
  +- minimatch@3.0.3
    +- brace-expansion@1.1.6
      +- balanced-match@0.4.2
      -- concat-map@0.0.1
    -- set-immediate-shim@1.0.1
-- webpack-core@0.6.8
  +- source-list-map@0.1.6
  -- source-map@0.4.4
  -- amdefine@1.0.0
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.0.14: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN react_demo1@1.0.0 No description
npm WARN react_demo1@1.0.0 No repository field.
D:\Workspace\Code-FrontEnd\ReactDemo1>
```

如上图所示则代表安装成功

## 创建webpack.config.js

文 / zhangwang (简书作者)  
原文链接：<http://www.jianshu.com/p/42e11515c10f>  
(<http://www.jianshu.com/p/42e11515c10f>)

Webpack拥有很多其它的比较高级的功能（比如说本文后面会介绍的loaders和plugins），这些功能其实都可以通过命令行模式实现，但是正如已经提到的，这样不太方便且容易出错的，一个更好的办法是定义一个配置文件，这个配置文件其实也是一个简单的JavaScript模块，可以把所有的与构建相关的信息放在里面。

webpack.config.js是webpack的配置文件

在项目的根目录下新建一个webpack.config.js的文件，代码如下：

```
//__dirname是node.js中的一个全局变量，它指向当前执行脚本所在的目录
module.exports = { //注意这里是exports不是export
  entry: __dirname + "/app/main.js", //唯一入口文件，就像Java中的main方法
  output: { //输出目录
    path: __dirname + "/build", //打包后的js文件存放的地方
    filename: "bundle.js" //打包后的js文件名
  }
};
```

这时候已经可以使用webpack进行打包了

在命令行下输入 `webpack`(非全局安装需使用`node_modules/.bin/webpack`)



```
管理员: C:\WINDOWS\system32\cmd.exe
D:\Workspace\Code-FrontEnd\ReactDemo1>webpack
Hash: 3a128f5c00b01ab6723a
Version: webpack 1.13.2
Time: 156ms
   Asset      Size  Chunks             Chunk Names
bundle.js  1.44 kB      0  [emitted]  main
    [0] ./app/main.js 46 bytes {0} [built]
D:\Workspace\Code-FrontEnd\ReactDemo1>
```

看到以上提示说明打包成功了，可以看到build目录下的bundle.js里多了很多自动生成的代码，但预览index.html却什么都没有看到，这是因为还没有往页面中写入内容。

## 更方便的执行打包任务

文 / zhangwang (简书作者)

原文链接: <http://www.jianshu.com/p/42e11515c10f>

(<http://www.jianshu.com/p/42e11515c10f>)

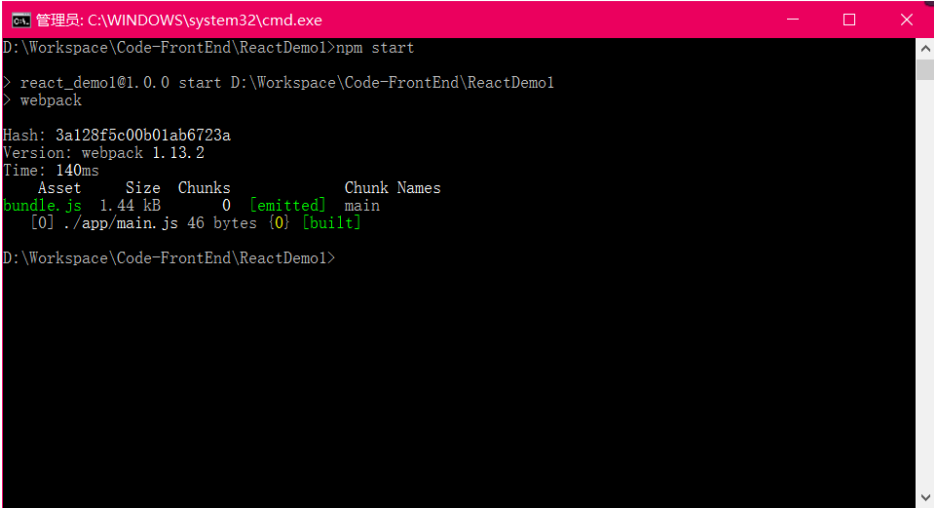
执行类似于node\_modules/.bin/webpack这样的命令其实是比较烦人且容易出错的，不过值得庆幸的是npm可以引导任务执行，对其进行配置后可以使用简单的npm start命令来代替这些繁琐的命令。在package.json中对npm的脚本部分进行相关设置即可，设置方法如下。

```
//package.json
{
  "name": "react_demo1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": { //默认test直接删除即可
    "start": "webpack" //配置的地方就是这里，相当于把npm的start命令指向webpack命令；注意：实际代码
  },
  "author": "waka",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^1.13.2"
  }
}
```

**注：**package.json中的脚本部分已经默认在命令前添加了node\_modules/.bin路径，所以无论是全局还是局部安装的Webpack，你都不需要写前面那指明详细的路径了。

npm的start是一个特殊的脚本名称，它的特殊性表现在，在命令行中使用 npm start 就可以执行相关命令，如果对应的此脚本名称不是 start，想要在命令行中运行时，需要这样用 npm run {script name} 如 npm run build，以下是执行 npm start 后命令行的输出显示：





以后直接输入 `npm start` 就可以进行打包操作了

## 使用Source Maps，使调试更容易

文 / zhangwang（简书作者）  
原文链接：<http://www.jianshu.com/p/42e11515c10f>  
(<http://www.jianshu.com/p/42e11515c10f>)

开发总是离不开调试，如果可以更加方便的调试当然就能提高开发效率，不过打包后的文件有时候你是不容易找到出错了的地方对应的源代码的位置的，Source Maps就是来帮我们解决这个问题。

通过简单的配置后，Webpack在打包时可以为我们的生成的source maps，这为我们提供了一种对应编译文件和源文件的方法，使得编译后的代码可读性更高，也更容易调试。

在webpack的配置文件中配置source maps，需要配置devtool，它有以下四种不同的配置选项，各具优缺点，描述如下：

devtool 选项	配置结果
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。这个文件具有最好的source map，但是它会减慢打包文件的构建速度
cheap-module-source-map	在一个单独的文件中生成一个不带列映射的map，不带列映射提高项目构建速度，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
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。不过在开发阶段这是一个非常好的选项，但是在生产阶段一定不要用这个选项
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



文 / zhangwang ( 简书作者 )

原文链接 : <http://www.jianshu.com/p/42e11515c10f>  
(<http://www.jianshu.com/p/42e11515c10f>)

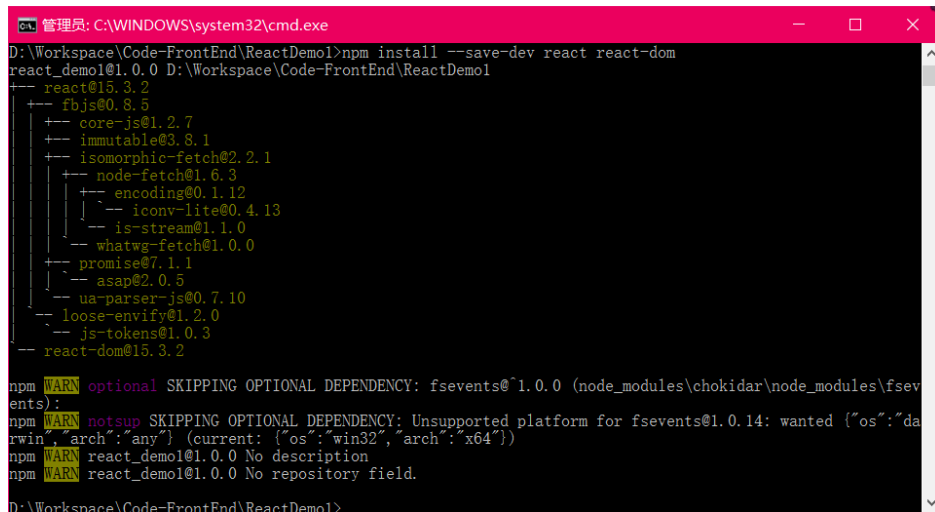
正如上表所述，上述选项由上到下打包速度越来越快，不过同时也具有越来越多的负面作用，较快的构建速度的后果就是对打包后的文件的执行有一定影响。

在学习阶段以及在小到中性的项目上，eval-source-map是一个很好的选项，不过记得只在开发阶段使用它，继续上面的例子，进行如下配置

```
//webpack.config.js
module.exports = {
  devtool: 'eval-source-map',//生成Source Maps,这里选择eval-source-map
  entry: __dirname + '/app/main.js',//唯一入口文件,__dirname是node.js中的一个全局变量，它指向
  output: { //输出目录
    path: __dirname + '/build',//打包后的js文件存放的地方
    filename: 'bundle.js' //打包后输出的js的文件名
  }
};
```

## 安装React

在终端输入 `npm install --save-dev react react-dom` 同时安装React和React-DOM



```
C:\WINDOWS\system32\cmd.exe
D:\Workspace\Code-FrontEnd\ReactDemol>npm install --save-dev react react-dom
react_demol@1.0.0 D:\Workspace\Code-FrontEnd\ReactDemol
+-- react@15.3.2
+-- fbjs@0.8.5
+-- core-js@1.2.7
+-- immutable@3.8.1
+-- isomorphic-fetch@2.2.1
+-- node-fetch@1.6.3
+-- encoding@0.1.12
+-- iconv-lite@0.4.13
+-- is-stream@1.1.0
+-- whatwg-fetch@1.0.0
+-- promise@7.1.1
+-- asap@2.0.5
+-- ua-parser-js@0.7.10
+-- loose-envify@1.2.0
+-- js-tokens@1.0.3
+-- react-dom@15.3.2

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.0.14: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN react_demol@1.0.0 No description
npm WARN react_demol@1.0.0 No repository field.

D:\Workspace\Code-FrontEnd\ReactDemol>
```

标准的React组件后缀名为.jsx，而.jsx是默认不被浏览器所支持的；所以需要有一个转换器，将不被浏览器识别的.jsx文件转换成浏览器能够正常运行的文件，这个转换器就是

**Babel**

## 安装和配置Babel



文 / zhangwang ( 简书作者 )

原文链接 : <http://www.jianshu.com/p/42e11515c10f>

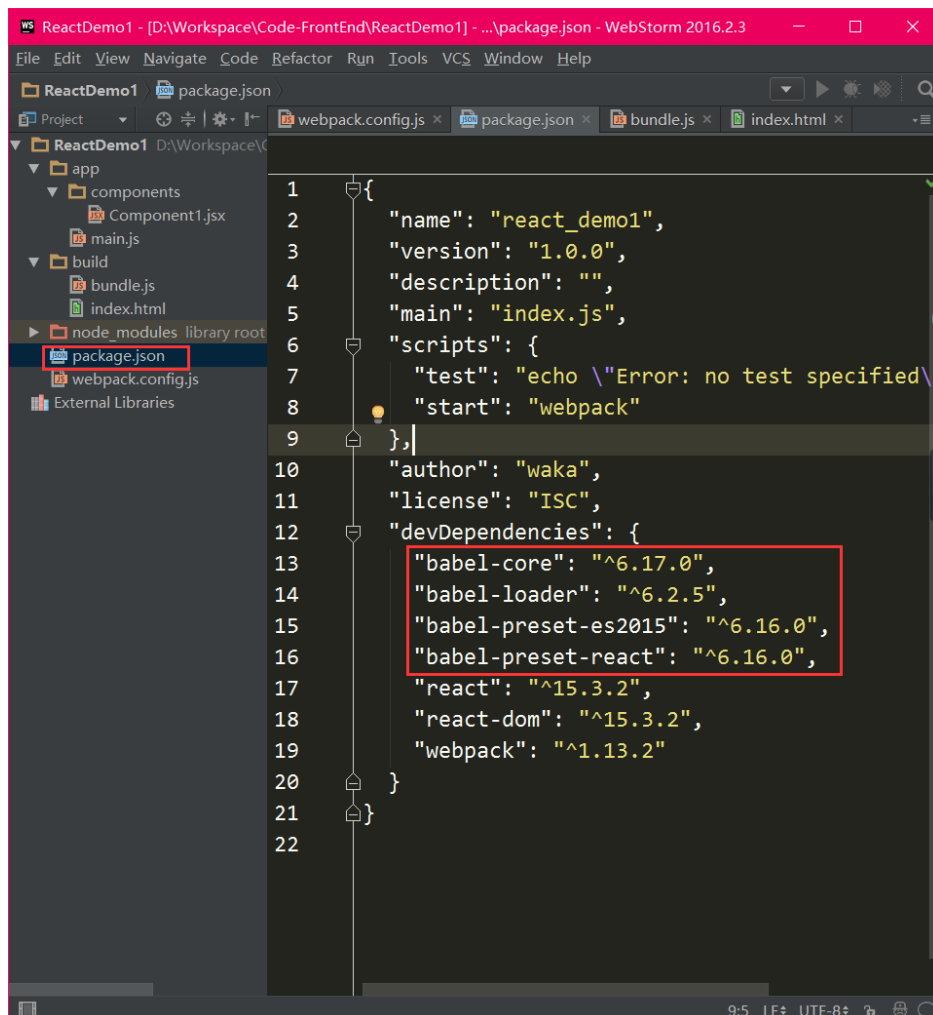
(<http://www.jianshu.com/p/42e11515c10f>)

Babel其实是一个编译JavaScript的平台，它的强大之处表现在可以通过编译帮你达到以下目的：

- 下一代的JavaScript标准（ES6，ES7），这些标准目前并未被当前的浏览器完全的支持；
- 使用基于JavaScript进行了拓展的语言，比如React的JSX

Babel其实是几个模块化的包，其核心功能位于称为babel-core的npm包中，不过webpack把它们整合在一起使用，但是对于每一个你需要的功能或拓展，你都需要安装单独的包（用得最多的是解析Es6的babel-preset-es2015包和解析JSX的babel-preset-react包）。

用npm一次性安装这些依赖包 `npm install --save-dev babel-core babel-loader babel-preset-es2015 babel-preset-react`



```
1 {
2   "name": "react_demo1",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\"",
8     "start": "webpack"
9   },
10  "author": "waka",
11  "license": "ISC",
12  "devDependencies": {
13    "babel-core": "^6.17.0",
14    "babel-loader": "^6.2.5",
15    "babel-preset-es2015": "^6.16.0",
16    "babel-preset-react": "^6.16.0",
17    "react": "^15.3.2",
18    "react-dom": "^15.3.2",
19    "webpack": "^1.13.2"
20  }
21 }
```





Babel其实可以完全在webpack.config.js中进行配置

但是考虑到babel具有非常多的配置选项，在单一的webpack.config.js文件中进行配置往往使得这个文件显得太复杂，因此一些开发者支持把babel的配置选项放在一个单独的名为 ".babelrc" 的配置文件中的。

我们现在的babel的配置并不算复杂，不过之后我们会再加一些东西，因此现在我们就提取出相关部分，分两个配置文件进行配置（webpack会自动调用.babelrc里的babel配置选项），如下：

在webpack.config.js中

```
//webpack.config.js
module.exports = {
  devtool: 'eval-source-map',//生成Source Maps,这里选择eval-source-map
  entry: __dirname + "/app/main.js",//唯一入口文件
  output: { //输出目录
    path: __dirname + "/build",//打包后的js文件存放的地方
    filename: "bundle.js"//打包后输出的js的文件名
  },

  module: {
    //loaders加载器
    loaders: [
      {
        test: /\.jsx$/,//一个匹配loaders所处理的文件的拓展名的正则表达式，这里用来匹配
        exclude: /node_modules/,//屏蔽不需要处理的文件（文件夹）（可选）
        loader: 'babel'//loader的名称（必须）
      }
    ]
  }
};
```

在项目根目录下新建 .babelrc 文件，没错你没看错，就是只有后缀名的文件，添加如下代码：

```
//.babelrc
{
  "presets": [
    "react",
    "es2015"
  ]
}
```

这时已将babel配置完成，可以使用JSX和ES6的语法了。

## 使用ES6书写React组件

在app->components目录下新建一个Component.jsx文件（注意首字母一定要大写），使用ES6语法返回一个React组件，代码如下：

```
//Component1.jsx
import React from 'react';

class Component1 extends React.Component {
  render() {
    return (
      <div>Hello World!</div>
    )
  }
}

//导出组件
export default Component1;
```

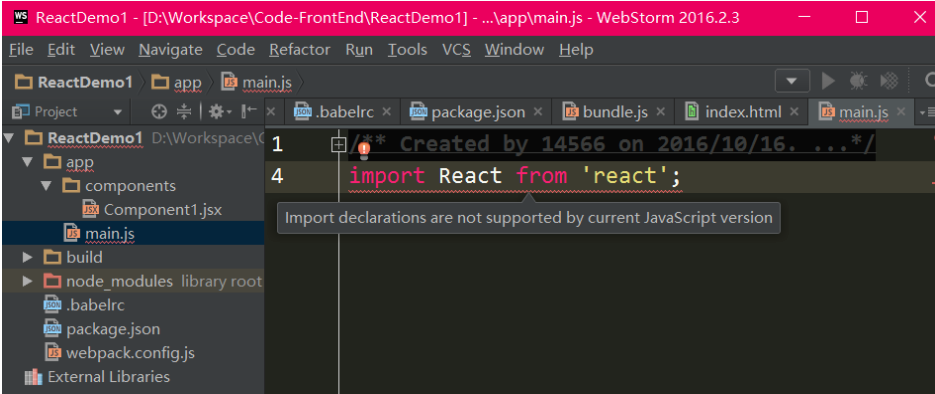


在main.js中使用ES6的语法，定义和渲染Component1模块，将React组件渲染至html的标签中：

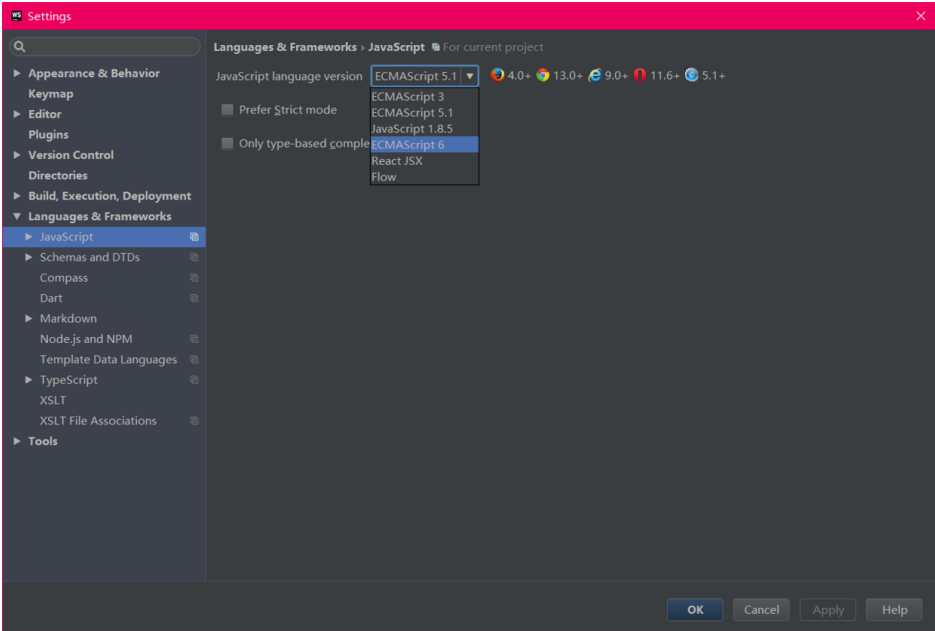
```
//main.js
import React from 'react';
import ReactDOM from 'react-dom';
import Component1 from './components/Component1.jsx';

ReactDOM.render(
  <Component1 />,
  document.getElementById('content')
);
```

如果你使用的是webstorm，可能会报这个错 Import declarations are not supported by current JavaScript version

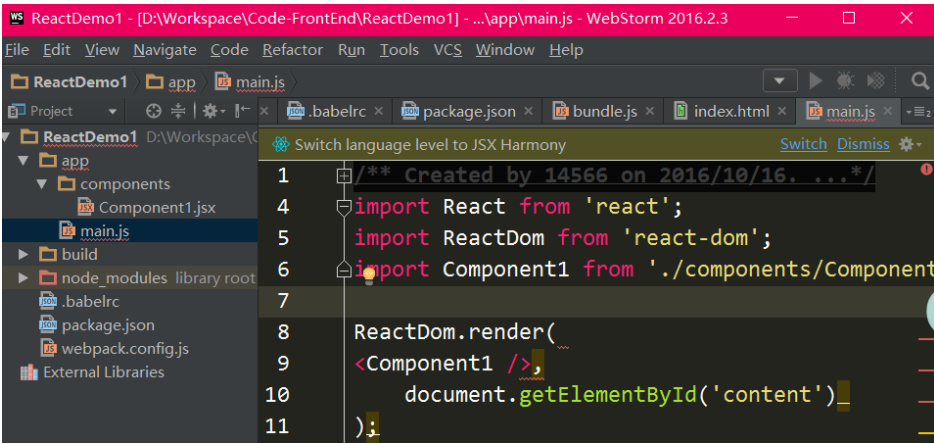


解决的方法是 File->Settings->Languages&Frameworks->JavaScript，将 JavaScript language version 改为 ECMAScript6



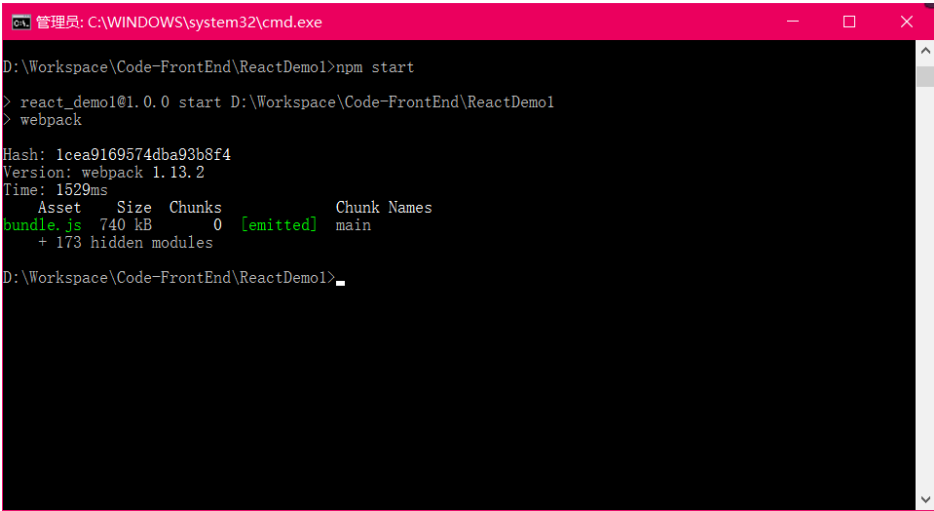
也可能会报这个错 Switch language level to JSX Harmony





在弹出栏，点击switch即可

这样，我们已经将React组件渲染至Html之中了，使用npm start命令重新打包，打包完成后在webstorm中预览index.html文件，即可看到Hello World！



每次修改完代码，都需要重新打包才能看到变化吗？  
Facebook的前端大神们怎么可能会容忍每次都手动打包呢？

我们可以使用webpack-dev-server来搭建本地开发服务器，修改代码后，立即可以看到变化；所见即所得，大大增加开发效率。

## 安装并启用webpack-dev-server



文 / zhangwang ( 简书作者 )

原文链接：<http://www.jianshu.com/p/42e11515c10f>  
(<http://www.jianshu.com/p/42e11515c10f>)

想不想让你的浏览器监测你都代码的修改，并自动刷新修改后的结果，其实Webpack提供一个可选的本地开发服务器，这个本地服务器基于node.js构建，可以实现你想要的这些功能，不过它是一个单独的组件，在webpack中进行配置之前需要单独安装它作为项目依赖

在终端输入，安装webpack-dev-server

```
npm install --save-dev webpack-dev-server
```

在webpack.config.js中配置webpack-dev-server，在这里需要修改下entry的路径，给它加一个 webpack/hot/dev-server，这里会用到Hot Module Replacement（热替换）插件，所以需要增加这个前缀，后文会提到，代码如下：

```
//webpack.config.js
module.exports = {
  devtool: 'eval-source-map', //生成Source Maps,这里选择eval-source-map
  entry: ['webpack/hot/dev-server', __dirname + '/app/main.js'], //唯一入口文件,__dirname是
  output: { //输出目录
    path: __dirname + '/build', //打包后的js文件存放的地方
    filename: 'bundle.js' //打包后输出的js的文件名
  },

  module: {
    //loaders加载器
    loaders: [
      {
        test: /\.js$/, //一个匹配loaders所处理的文件的拓展名的正则表达式，这里用来匹
        exclude: /node_modules/, //屏蔽不需要处理的文件（文件夹）（可选）
        loader: 'babel' //loader的名称（必须）
      }
    ]
  },

  //webpack-dev-server配置
  devServer: {
    contentBase: './build', //默认webpack-dev-server会为根文件夹提供本地服务器，如果想为另外
    colors: true, //在cmd终端中输出彩色日志
    historyApiFallback: true, //在开发单页应用时非常有用，它依赖于HTML5 history API，如果设置
    inline: true, //设置为true，当源文件改变时会自动刷新页面
    port: 8080, //设置默认监听端口，如果省略，默认为"8080"
    process: true, //显示合并代码进度
  }
};
```

在package.json中配置启动webpack-dev-server的命令

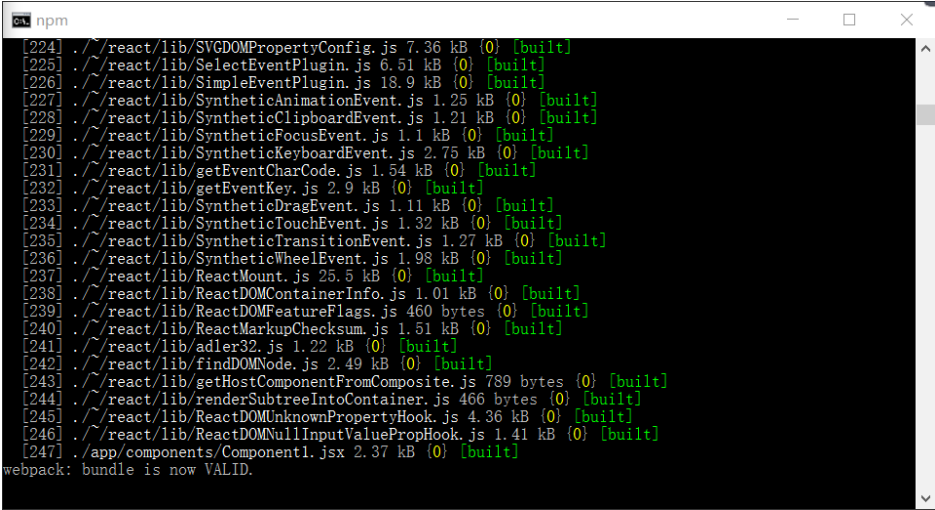
```
//package.json, 注意：在实际使用过程中package.json不能有任何注释
{
  ...
  "scripts": {
    "start": "webpack",
    "dev": "webpack-dev-server"
  },
  ...
}
```

这时，我们可以启动webpack-dev-server服务器，看看是否能够实时预览  
在终端里输入

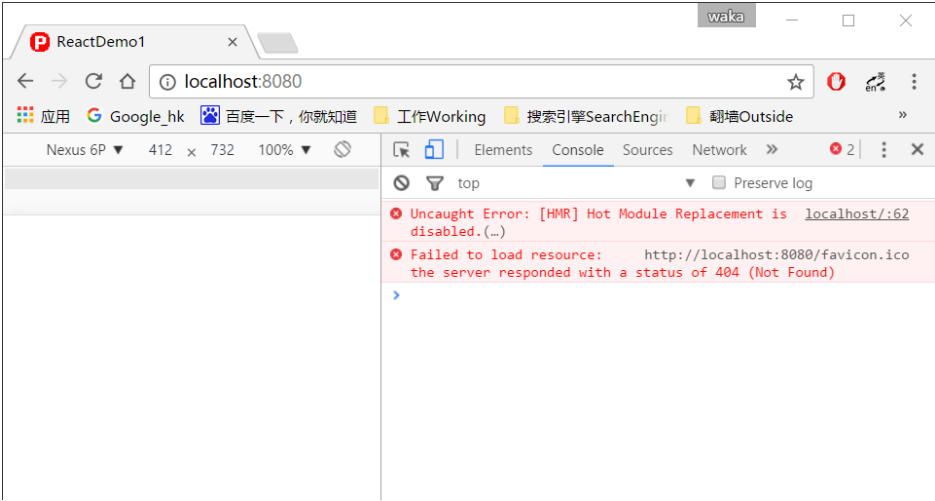
```
npm run dev
```



如果终端输出一大堆日志后，提示 `webpack: bundle is now VALID` 则代表本地服务器启动成功，如果需要停止服务，在终端按 `ctrl+c`，按提示输入 `y` 即可（如果不行多按几次...）



在浏览器中输入 `http://localhost:8080`，可以看到，打开了一个空页面，并没有显示Hello World；按F12打开控制台，可以清除地看到错误提示：Uncaught Error: [HMR] Hot Module Replacement is disabled



所以我们来安装和配置这个Hot Module Replacement。

## Hot Module Replacement



文 / zhangwang ( 简书作者 )

原文链接 : <http://www.jianshu.com/p/42e11515c10f>

(<http://www.jianshu.com/p/42e11515c10f>)

Hot Module Replacement ( HMR ) 是webpack里很有用的一个插件，它允许你在修改组件代码后，自动刷新实时预览修改后的效果。

在webpack中实现HMR也很简单，只需要做两项配置

1. 在webpack配置文件中添加HMR插件；
2. 在Webpack Dev Server中添加“hot”参数；

不过配置完这些后，JS模块其实还是不能自动热加载的，还需要在你的JS模块中执行一个Webpack提供的API才能实现热加载，虽然这个API不难使用，但是如果是React模块，使用我们已经熟悉的Babel可以更方便的实现功能热加载。

整理下我们的思路，具体实现方法如下

- Babel和webpack是独立的工具
- 二者可以一起工作
- 二者都可以通过插件拓展功能
- HMR是一个webpack插件，它让你能浏览器中实时观察模块修改后的效果，但是如果你想让它工作，需要对模块进行额外的配额；
- Babel有一个叫做react-transform-hrm的插件，可以在不对React模块进行额外的配置的前提下让HMR正常工作；

在webpack.config.js中配置如下：

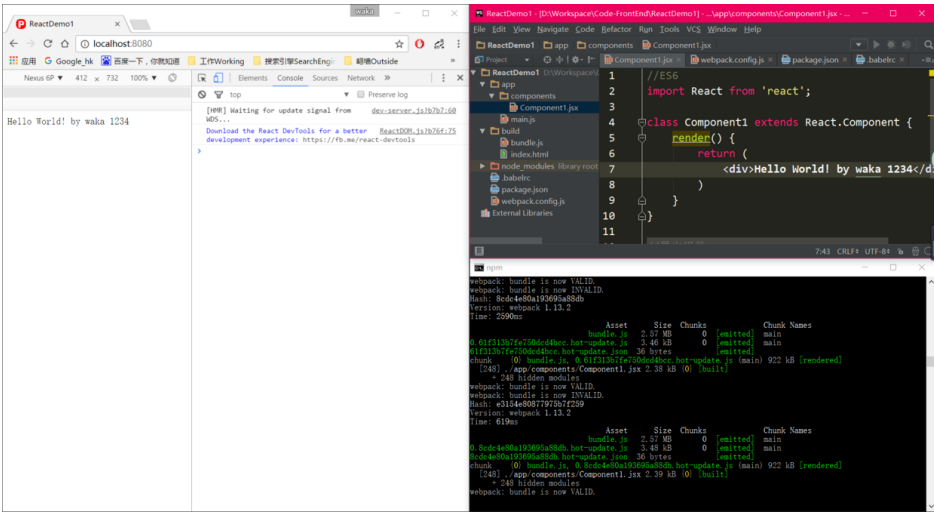
```
//webpack.config.js
var webpack = require('webpack');//引入Webpack模块供我们调用，这里只能使用ES5语法，使用ES6语法会

module.exports = {
  devtool: 'eval-source-map',
  entry: ['webpack/hot/dev-server', __dirname + '/app/main.js'],
  output: {
    path: __dirname + '/build',
    filename: 'bundle.js'
  },
  module: {
    loaders: [
      {
        test: /\.?(js|jsx)$/,
        exclude: /node_modules/,
        loader: 'babel'
      }
    ]
  },
  plugins: [
    new webpack.HotModuleReplacementPlugin()//热模块替换插件
  ],
  devServer: {
    contentBase: './build',
    colors: true,
    historyApiFallback: true,
    inline: true,
    port: 8080,
    process: true
  }
};
```



其实，这时候已经可以正常工作了；在终端输入 `npm run dev`，待命令行提示 `webpack: bundle is now VALID` 后，在浏览器中输入 `http://localhost:8080`，可以看到，正常显示Hello World

在webstorm中更改React返回的内容，`ctrl+s` 即可看到浏览器页面同步刷新，所见即所得



如果觉得这样不放心，让我们谨遵医嘱，进行如下额外配置

安装react-transform-hmr，在不对React模块进行额外的配置的前提下让HMR正常工作

```
npm install --save-dev babel-plugin-react-transform react-transform-hmr
```

在 `.babelrc` 文件里配置babel，注意这里有一堆括号，别写错了

```

//.babelrc
{
  "presets": [
    "react",
    "es2015"
  ],
  "env": {
    "development": {
      "plugins": [
        [
          "react-transform",
          {
            "transforms": [
              {
                "transform": "react-transform-hmr",
                "imports": [
                  "react"
                ],
                "locals": [
                  "module"
                ]
              }
            ]
          }
        ]
      ]
    }
  }
}

```

至此，已经用webpack构建好了React项目的基础依赖，可以愉快的开发React程序了。

Thank you.





waka (/u/d184d3ab9060)

写了 24310 字，被 78 人关注，获得了 88 个喜欢

(/u/d184d3ab9060)

+ 关注

♡ 喜欢 (/sign\_in)

| 44









更多分享

(http://cwb.assets.jianshu.io/notes/images/636502c


被以下专题收入，发现更多相似内容




Web前端之路 (/c/684ef36cb2df?utm\_source=desktop&utm\_medium=notes-included-collection)




React.js (/c/4dcf98759530?utm\_source=desktop&utm\_medium=notes-included-collection)




webpack... (/c/3097fa4f0b3c?utm\_source=desktop&utm\_medium=notes-included-collection)



react (/c/e1f44f68e742?utm\_source=desktop&utm\_medium=notes-included-collection)



web前端 (/c/f274fb303f26?utm\_source=desktop&utm\_medium=notes-included-collection)



webpack (/c/31a0fc722431?utm\_source=desktop&utm\_medium=notes-included-collection)

