

入门Webpack , 看这篇就够了



zhangwang (/u/7091a52ac9e5) [+ 关注](#)

2016.08.05 11:21* 字数 7229 阅读 655797 评论 787 喜欢 2540 赞赏 98
(/u/7091a52ac9e5)

2017年12月7日更新, 添加了 `clean-webpack-plugin`, `babel-env-preset`, 添加本文涉及到的所有代码的示例, 如果你在学习过程中出错了, 可点击此处参考 (<https://link.jianshu.com?t=https://github.com/zhangwang1990/blogs/tree/master/sources/webpackTest>)

写在前面的话

阅读本文之前, 先看下面这个webpack的配置文件, 如果每一项你都懂, 那本文能带给你的收获也许就比较有限, 你可以快速浏览或直接跳过; 如果你和十天前的我一样, 对很多选项存在着疑惑, 那花一段时间慢慢阅读本文, 你的疑惑一定一个一个都会消失; 如果你以前没怎么接触过Webpack, 而你又你对webpack感兴趣, 那么动手跟着本文中那个贯穿始终的例子写一次, 写完以后你会发现你已明明白白的走进了Webpack的大门。

```
// 一个常见的`webpack`配置文件
const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const ExtractTextPlugin = require('extract-text-webpack-plugin');

module.exports = {
  entry: __dirname + "/app/main.js", //已多次提及的唯一入口文件
  output: {
    path: __dirname + "/build",
    filename: "bundle-[hash].js"
  },
  devtool: 'none',
  devServer: {
    contentBase: "./public", //本地服务器所加载的页面所在的目录
    historyApiFallback: true, //不跳转
    inline: true,
    hot: true
  },
  module: {
    rules: [{
      test: /\.jsx?$/,
      use: {
        loader: "babel-loader"
      },
      exclude: /node_modules/
    }, {
      test: /\.css$/,
      use: ExtractTextPlugin.extract({
        fallback: "style-loader",
        use: [{
          loader: "css-loader",
          options: {
            modules: true,
            localIdentName: '[name]__[local]--[hash:base64:5]'
          }
        }, {
          loader: "postcss-loader"
        }
      ]
    }
  ]
},
  plugins: [
    new webpack.BannerPlugin('版权所有，翻版必究'),
    new HtmlWebpackPlugin({
      template: __dirname + "/app/index.tmpl.html" //new 一个这个插件的实例，并传入模板文件
    }),
    new webpack.optimize.OccurrenceOrderPlugin(),
    new webpack.optimize.UglifyJsPlugin(),
    new ExtractTextPlugin("style.css")
  ]
};
```

什么是WebPack，为什么要使用它？

为什么要使用WebPack

现今的很多网页其实可以看做是功能丰富的应用，它们拥有着复杂的JavaScript代码和一大堆依赖包。为了简化开发的复杂度，前端社区涌现出了很多好的实践方法

- **模块化**，让我们可以把复杂的程序细化为小的文件；
- 类似于TypeScript这种在JavaScript基础上拓展的开发语言：使我们能够实现目前版本的JavaScript不能直接使用的特性，并且之后还能转换为JavaScript文件使浏览器可以识别；
- Scss，less等CSS预处理器
- ...

这些改进确实大大的提高了我们的开发效率，但是利用它们开发的文件往往需要进行额外的处理才能让浏览器识别,而手动处理又是非常繁琐的，这就为WebPack类的工具的出现提供了需求。

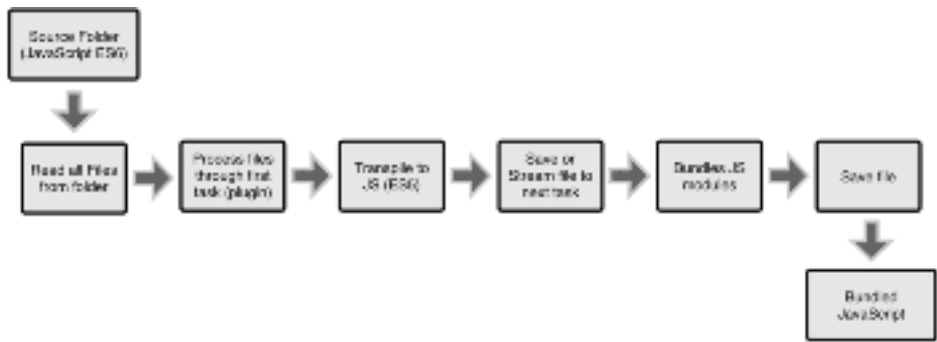
什么是Webpack

WebPack可以看做是**模块打包机**：它做的事情是，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和Grunt以及Gulp相比有什么特性

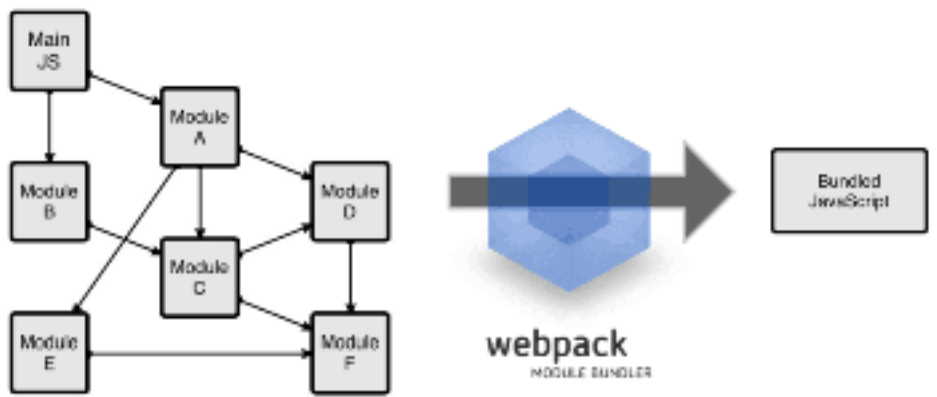
其实Webpack和另外两个并没有太多的可比性，Gulp/Grunt是一种能够优化前端的开发流程的工具，而WebPack是一种模块化的解决方案，不过Webpack的优点使得Webpack在很多场景下可以替代Gulp/Grunt类的工具。

Grunt和Gulp的工作方式是：在一个配置文件中，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，工具之后可以自动替你完成这些任务。



Grunt和Gulp的工作流程

Webpack的工作方式是：把你的项目当做一个整体，通过一个给定的主文件（如：index.js），Webpack将从这个文件开始找到你的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。



Webpack工作方式

如果实在要把二者进行比较，Webpack的处理速度更快更直接，能打包更多不同类型的文件。

开始使用Webpack

初步了解了Webpack工作方式后，我们一步步的开始学习使用Webpack。

安装

Webpack可以使用npm安装，新建一个空的练习文件夹（此处命名为webpack sample project），在终端中转到该文件夹后执行下述指令就可以完成安装。

```
//全局安装
npm install -g webpack
//安装到你的项目目录
npm install --save-dev webpack
```

正式使用Webpack前的准备

1. 在上述练习文件夹中创建一个package.json文件，这是一个标准的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等。在终端中使用 `npm init` 命令可以自动创建这个package.json文件

```
npm init
```

输入这个命令后，终端会问你一系列诸如项目名称，项目描述，作者等信息，不过不用担心，如果你不准备在npm中发布你的模块，这些问题的答案都不重要，回车默认即可。

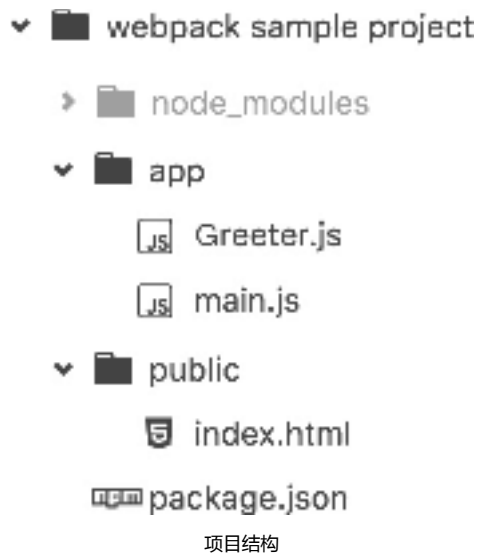
2. package.json文件已经就绪，我们在本项目中安装Webpack作为依赖包

```
// 安装Webpack
npm install --save-dev webpack
```

3. 回到之前的空文件夹，并在里面创建两个文件夹,app文件夹和public文件夹，app文件夹用来存放原始数据和我们将写的JavaScript模块，public文件夹用来存放之后供浏览器读取的文件（包括使用webpack打包生成的js文件以及一个 `index.html` 文件）。接下来我们再创建三个文件:

- `index.html` --放在public文件夹中;
- `Greeter.js` -- 放在app文件夹中;
- `main.js` -- 放在app文件夹中;

此时项目结构如下图所示



我们在`index.html`文件中写入最基础的html代码，它在这里目的在于引入打包后的js文件（这里我们先把之后打包后的js文件命名为 `bundle.js`，之后我们还会详细讲述）。

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Webpack Sample Project</title>
  </head>
  <body>
    <div id='root'>
    </div>
    <script src="bundle.js"></script>
  </body>
</html>
```

我们在 Greeter.js 中定义一个返回包含问候信息的 html 元素的函数,并依据CommonJS规范导出这个函数为一个模块：

```
// Greeter.js
module.exports = function() {
  var greet = document.createElement('div');
  greet.textContent = "Hi there and greetings!";
  return greet;
};
```

main.js 文件中我们写入下述代码，用以把 Greeter模块 返回的节点插入页面。

```
//main.js
const greeter = require('./Greeter.js');
document.querySelector("#root").appendChild(greeter());
```

正式使用Webpack

webpack可以在终端中使用，在基本的使用方法如下：

```
# {entry file}出填写入口文件的路径，本文中就是上述main.js的路径，
# {destination for bundled file}处填写打包文件的存放路径
# 填写路径的时候不用添加{}
webpack {entry file} {destination for bundled file}
```

指定入口文件后，webpack将自动识别项目所依赖的其它文件，不过需要注意的是如果你的webpack不是全局安装的，那么当你在终端中使用此命令时，需要额外指定其在node_modules中的地址，继续上面的例子，在终端中输入如下命令

```
# webpack非全局安装的情况
node_modules/.bin/webpack app/main.js public/bundle.js
```

结果如下

```
node_modules/.bin/webpack app/main.js public/bundle.js
Hash: e617000114aae964fd73
Version: webpack 3.5.3
Time: 73ms

   Asset      Size  Chunks             Chunk Names
bundle.js  2.79 kB       0  [emitted]  main
   [0] ./app/main.js 96 bytes {0} [built]
   [1] ./app/Greeter.js 148 bytes {0} [built]
```

使用命令行打包

可以看出 webpack 同时编译了 main.js 和 Greeter.js ,现在打开 index.html ,可以看到如下结果



htmlResult1

有没有很激动，已经成功的使用 Webpack 打包了一个文件了。不过在终端中进行复杂的操作，其实是不太方便且容易出错的，接下来看看Webpack的另一种更常见的使用方法。

通过配置文件来使用 Webpack

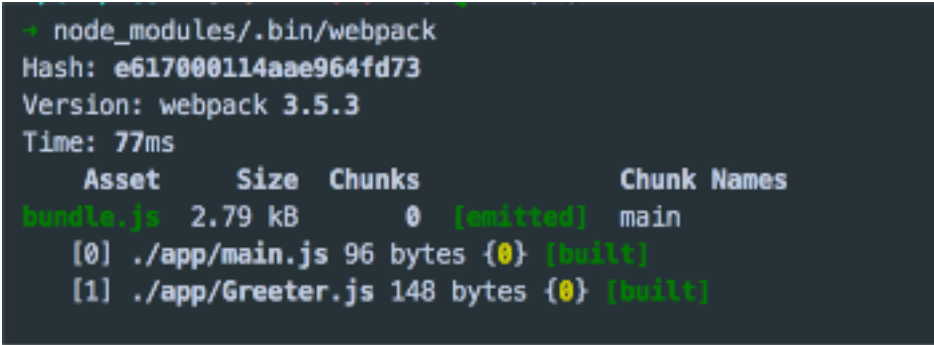
Webpack拥有很多其它的比较高级的功能（比如说本文后面会介绍的 loaders 和 plugins ），这些功能其实都可以通过命令行模式实现，但是正如前面提到的，这样不太方便且容易出错的，更好的办法是定义一个配置文件，这个配置文件其实也是一个简单的JavaScript模块，我们可以把所有的与打包相关的信息放在里面。

继续上面的例子来说明如何写这个配置文件，在当前练习文件夹的根目录下新建一个名为 webpack.config.js 的文件，我们在其中写入如下所示的简单配置代码，目前的配置主要涉及到的内容是入口文件路径和打包后文件的存放路径。

```
module.exports = {
  entry: __dirname + "/app/main.js", //已多次提及的唯一入口文件
  output: {
    path: __dirname + "/public", //打包后的文件存放的地方
    filename: "bundle.js" //打包后输出文件的文件名
  }
}
```

注：“__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。

有了这个配置之后，再打包文件，只需在终端里运行 webpack (非全局安装需使用 node_modules/.bin/webpack) 命令就可以了，这条命令会自动引用 webpack.config.js 文件中的配置选项，示例如下：



配合配置文件进行打包

又学会了一种使用 Webpack 的方法，这种方法不用管那烦人的命令行参数，有没有感觉很爽。如果我们可以连 webpack(非全局安装需使用node_modules/.bin/webpack) 这条命令都可以不用，那种感觉会不会更爽~，继续看下文。

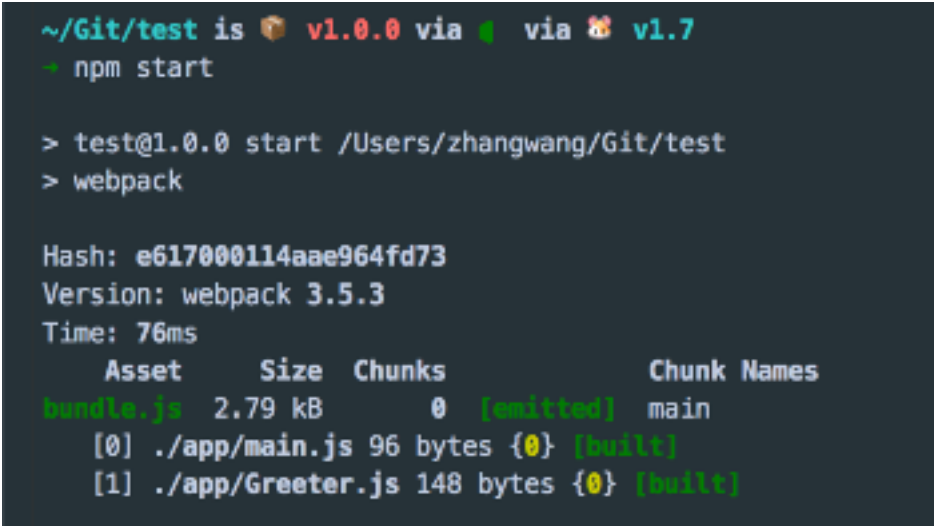
更快捷的执行打包任务

在命令行中输入命令需要代码类似于 node_modules/.bin/webpack 这样的路径其实是比较烦人的，不过值得庆幸的是 npm 可以引导任务执行，对 npm 进行配置后可以在命令行中使用简单的 npm start 命令来替代上面略微繁琐的命令。在 package.json 中对 scripts 对象进行相关设置即可，设置方法如下。

```
{
  "name": "webpack-sample-project",
  "version": "1.0.0",
  "description": "Sample webpack project",
  "scripts": {
    "start": "webpack" // 修改的是这里，JSON文件不支持注释，引用时请清除
  },
  "author": "zhang",
  "license": "ISC",
  "devDependencies": {
    "webpack": "3.10.0"
  }
}
```

注：package.json 中的 script 会安装一定顺序寻找命令对应位置，本地的 node_modules/.bin 路径就在这个寻找清单中，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 命令是一个特殊的脚本名称，其特殊性表现在，在命令行中使用 npm start 就可以执行其对于的命令，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，我们在命令行中输入 npm start 试试，输出结果如下：



使用npm start 打包代码

现在只需要使用 npm start 就可以打包文件了，有没有觉得 webpack 也不过如此嘛，不过不要太小瞧 webpack，要充分发挥其强大的功能我们需要修改配置文件的其它选项，一项项来看。

Webpack的强大功能

生成Source Maps (使调试更容易)

开发总是离不开调试，方便的调试能极大的提高开发效率，不过有时候通过打包后的文件，你是不容易找到出错了的地方，对应的你写的代码的位置的，Source Maps 就是来帮我们解决这个问题的。

通过简单的配置，webpack 就可以在打包时为我们生成的 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；
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；

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，较快的打包速度的后果就是对打包后的文件的执行有一定影响。

对小到中型的项目中，eval-source-map 是一个很好的选项，再次强调你只应该开发阶段使用它，我们继续对上文新建的 webpack.config.js，进行如下配置：

```
module.exports = {
  devtool: 'eval-source-map',
  entry: __dirname + "/app/main.js",
  output: {
    path: __dirname + "/public",
    filename: "bundle.js"
  }
}
```

cheap-module-eval-source-map 方法构建速度更快，但是不利于调试，推荐在大型项目考虑时间成本时使用。

使用webpack构建本地服务器

想不想让你的浏览器监听你的代码的修改，并自动刷新显示修改后的结果，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

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

devserver作为webpack配置选项中的一项，以下是它的一些配置选项，更多配置可参考这里 (<https://link.jianshu.com?t=https://webpack.js.org/configuration/dev-server/>)



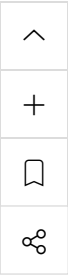
devserver的配置选项	功能描述
contentBase	默认webpack-dev-server会为根文件夹提供本地服务器，如果想为另外一个目录下的文件提供本地服务器，应该在这里设置其所在目录（本例设置到"public"目录）
port	设置默认监听端口，如果省略，默认为"8080"
inline	设置为 true ，当源文件改变时会自动刷新页面
historyApiFallback	在开发单页应用时非常有用，它依赖于HTML5 history API，如果设置为 true ，所有的跳转将指向index.html

把这些命令加到webpack的配置文件中，现在的配置文件 webpack.config.js 如下所示

```
module.exports = {
  devtool: 'eval-source-map',

  entry: __dirname + "/app/main.js",
  output: {
    path: __dirname + "/public",
    filename: "bundle.js"
  },

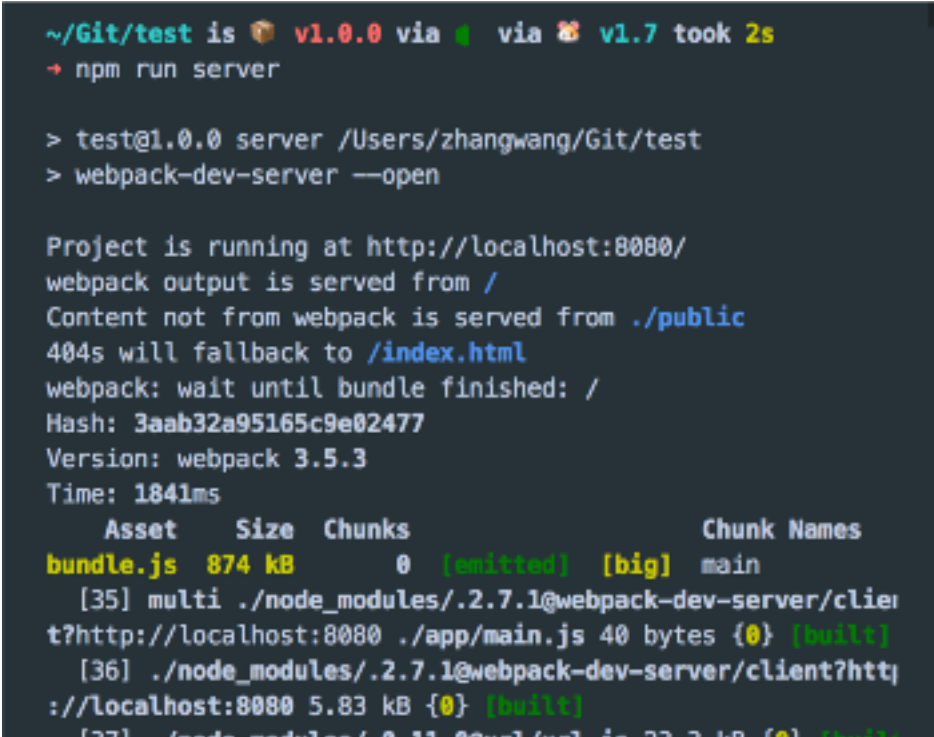
  devServer: {
    contentBase: "./public", //本地服务器所加载的页面所在的目录
    historyApiFallback: true, //不跳转
    inline: true //实时刷新
  }
}
```



在 package.json 中的 scripts 对象中添加如下命令，用以开启本地服务器：

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "webpack",
  "server": "webpack-dev-server --open"
},
```

在终端中输入 npm run server 即可在本地的 8080 端口查看结果



开启本地服务器

Loaders

鼎鼎大名的Loaders登场了！

Loaders 是 webpack 提供的最激动人心的功能之一了。通过使用不同的 loader，webpack 有能力调用外部的脚本或工具，实现对不同格式的文件的处理，比如说分析转换scss为css，或者把下一代的JS文件（ES6，ES7）转换为现代浏览器兼容的JS文件，对React的开发而言，合适的Loaders可以把React的中用到的JSX文件转换为JS文件。

Loaders需要单独安装并且需要在 webpack.config.js 中的 modules 关键字下进行配置，Loaders的配置包括以下几方面：

- test：一个用以匹配loaders所处理文件的拓展名的正则表达式（必须）
- loader：loader的名称（必须）
- 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）

不过在配置loader之前，我们把 Greeter.js 里的问候消息放在一个单独的JSON文件里.并通过合适的配置使 Greeter.js 可以读取该JSON文件的值，各文件修改后的代码如下：

在app文件夹中创建带有问候信息的JSON文件(命名为 config.json)

```
{
  "greetText": "Hi there and greetings from JSON!"
}
```

更新后的Greeter.js

```
var config = require('./config.json');

module.exports = function() {
  var greet = document.createElement('div');
  greet.textContent = config.greetText;
  return greet;
};
```

注 由于 webpack3.* / webpack2.* 已经内置可处理JSON文件，这里我们无需再添加 webpack1.* 需要的 json-loader。在看如何具体使用loader之前我们先看看Babel是什么？

Babel

Babel其实是一个编译JavaScript的平台，它可以编译代码帮你达到以下目的：

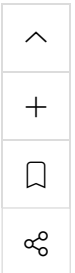
- 让你能使用最新的JavaScript代码（ES6，ES7...），而不用管新标准是否被当前使用的浏览器完全支持；
- 让你能使用基于JavaScript进行了拓展的语言，比如React的JSX；

Babel的安装与配置

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，webpack可以把其不同的包整合在一起使用，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-env-preset 包和解析JSX的 babel-preset-react 包）。

我们先来一次性安装这些依赖包

```
// npm一次性安装多个依赖模块，模块之间用空格隔开
npm install --save-dev babel-core babel-loader babel-preset-env babel-preset-react
```



在 webpack 中配置Babel的方法如下:

```
module.exports = {
  entry: __dirname + "/app/main.js", //已多次提及的唯一入口文件
  output: {
    path: __dirname + "/public", //打包后的文件存放的地方
    filename: "bundle.js" //打包后输出文件的文件名
  },
  devtool: 'eval-source-map',
  devServer: {
    contentBase: "./public", //本地服务器所加载的页面所在的目录
    historyApiFallback: true, //不跳转
    inline: true //实时刷新
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        use: {
          loader: "babel-loader",
          options: {
            presets: [
              "env", "react"
            ]
          }
        },
        exclude: /node_modules/
      }
    ]
  }
};
```

现在你的webpack的配置已经允许你使用ES6以及JSX的语法了。继续用上面的例子进行测试，不过这次我们会使用React，记得先安装 React 和 React-DOM

```
npm install --save react react-dom
```

接下来我们使用ES6的语法，更新 Greeter.js 并返回一个React组件

```
//Greeter.js
import React, {Component} from 'react'
import config from './config.json';

class Greeter extends Component{
  render() {
    return (
      <div>
        {config.greetText}
      </div>
    );
  }
}

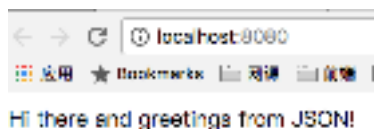
export default Greeter
```

修改 main.js 如下，使用ES6的模块定义和渲染Greeter模块

```
// main.js
import React from 'react';
import {render} from 'react-dom';
import Greeter from './Greeter';

render(<Greeter />, document.getElementById('root'));
```

重新使用 npm start 打包，如果之前打开的本地服务器没有关闭，你应该可以在 localhost:8080 下看到与之前一样的内容，这说明 react 和 es6 被正常打包了。



Babel的配置

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。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：

```
module.exports = {
  entry: __dirname + "/app/main.js", //已多次提及的唯一入口文件
  output: {
    path: __dirname + "/public", //打包后的文件存放的地方
    filename: "bundle.js" //打包后输出文件的文件名
  },
  devtool: 'eval-source-map',
  devServer: {
    contentBase: "./public", //本地服务器所加载的页面所在的目录
    historyApiFallback: true, //不跳转
    inline: true //实时刷新
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        use: {
          loader: "babel-loader"
        },
        exclude: /node_modules/
      }
    ]
  }
};
```

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

到目前为止，我们已经知道了，对于模块，Webpack能提供非常强大的处理功能，而那些是模块呢。

一切皆模块

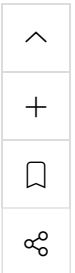
Webpack有一个不可不说的优点，它把所有的文件都当做模块处理，JavaScript代码，CSS和fonts以及图片等等通过合适的loader都可以被处理。

CSS

webpack提供两个工具处理样式表，`css-loader` 和 `style-loader`，二者处理的任务不同，`css-loader` 使你能够使用类似 `@import` 和 `url(...)` 的方法实现 `require()` 的功能，`style-loader` 将所有的计算后的样式加入页面中，二者组合在一起使你能够把样式表嵌入webpack打包后的JS文件中。

继续上面的例子

```
//安装
npm install --save-dev style-loader css-loader
```



```
//使用
module.exports = {
  ...
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        use: {
          loader: "babel-loader"
        },
        exclude: /node_modules/
      },
      {
        test: /\.css$/,
        use: [
          {
            loader: "style-loader"
          }, {
            loader: "css-loader"
          }
        ]
      }
    ]
  }
};
```

请注意这里对同一个文件引入多个loader的方法。

接下来，在app文件夹里创建一个名字为"main.css"的文件，对一些元素设置样式

```
/* main.css */
html {
  box-sizing: border-box;
  -ms-text-size-adjust: 100%;
  -webkit-text-size-adjust: 100%;
}

*, *:before, *:after {
  box-sizing: inherit;
}

body {
  margin: 0;
  font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;
}

h1, h2, h3, h4, h5, h6, p, ul {
  margin: 0;
  padding: 0;
}
```

我们这里例子中用到的 webpack 只有单一的入口，其它的模块需要通过 import, require, url 等与入口文件建立其关联，为了让webpack能找到"main.css"文件，我们把它导入"main.js"中，如下

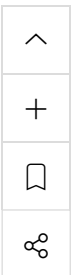
```
//main.js
import React from 'react';
import {render} from 'react-dom';
import Greeter from './Greeter';

import './main.css';//使用require导入css文件

render(<Greeter />, document.getElementById('root'));
```

通常情况下，css会和js打包到同一个文件中，并不会打包为一个单独的css文件，不过通过合适的配置webpack也可以把css打包为单独的文件的。

上面的代码说明webpack是怎么把css当做模块看待的，咱们继续看一个更加真实的css模块实践。



CSS module

在过去的一些年里，JavaScript通过一些新的语言特性，更好的工具以及更好的实践方法（比如说模块化）发展得非常迅速。模块使得开发者把复杂的代码转化为小的，干净的，依赖声明明确的单元，配合优化工具，依赖管理和加载管理可以自动完成。

不过前端的另外一部分，CSS发展就相对慢一些，大多样式表却依旧巨大且充满了全局类名，维护和修改都非常困难。

被称为 CSS modules 的技术意在把JS的模块化思想带入CSS中来，通过CSS模块，所有的类名，动画名默认都只作用于当前模块。Webpack对CSS模块化提供了非常好的支持，只需要在CSS loader中进行简单配置即可，然后就可以直接把CSS的类名传递到组件的代码中，这样做有效避免了全局污染。具体的代码如下

```
module.exports = {  
  
  ...  
  
  module: {  
    rules: [  
      {  
        test: /\.jsx?$/,  
        use: {  
          loader: "babel-loader"  
        },  
        exclude: /node_modules/  
      },  
      {  
        test: /\.css$/,  
        use: [  
          {  
            loader: "style-loader"  
          }, {  
            loader: "css-loader",  
            options: {  
              modules: true, // 指定启用css modules  
              localIdentName: '[name]__[local]--[hash:base64:5]' // 指定c  
            }  
          }  
        ]  
      }  
    ]  
  }  
};
```

我们在app文件夹下创建一个 Greeter.css 文件来进行一下测试

```
/* Greeter.css */  
.root {  
  background-color: #eee;  
  padding: 10px;  
  border: 3px solid #ccc;  
}
```

导入 .root 到Greeter.js中

```
import React, {Component} from 'react';  
import config from './config.json';  
import styles from './Greeter.css';//导入  
  
class Greeter extends Component{  
  render() {  
    return (  
      <div className={styles.root}> //使用cssModule添加类名的方法  
        {config.greetText}  
      </div>  
    );  
  }  
}  
  
export default Greeter
```

放心使用把，相同的类名也不会造成不同组件之间的污染。

```
<body>
  <div id="root">
    <div class="Greeter__root--16xre">Hi there and
      greetings from JSON!!!</div>
    </div>
  <script type="text/javascript" src="bundle.js">

```

应用了css module后的样式

CSS modules 也是一个很大的主题，有兴趣的话可以去其官方文档 (<https://link.jianshu.com?t=https://github.com/css-modules/css-modules>)了解更多。

CSS预处理器

Sass 和 Less 之类的预处理器是对原生CSS的拓展，它们允许你使用类似于 variables , nesting , mixins , inheritance 等不存在于CSS中的特性来写CSS，CSS预处理器可以这些特殊类型的语句转化为浏览器可识别的CSS语句，

你现在可能都已经熟悉了，在webpack里使用相关loaders进行配置就可以使用了，以下是常用的CSS 处理 loaders：

- Less Loader
- Sass Loader
- Stylus Loader

不过其实也存在一个CSS的处理平台 -PostCSS，它可以帮助你的CSS实现更多的功能，在其官方文档 (<https://link.jianshu.com?t=https://github.com/postcss/postcss>)可了解更多相关知识。

举例来说如何使用PostCSS，我们使用PostCSS来为CSS代码自动添加适应不同浏览器的CSS前缀。

首先安装 postcss-loader 和 autoprefixer（自动添加前缀的插件）

```
npm install --save-dev postcss-loader autoprefixer
```

接下来，在webpack配置文件中添加 postcss-loader，在根目录新建 postcss.config.js,并添加如下代码之后，重新使用 npm start 打包时，你写的css会自动根据Can i use里的数据添加不同前缀了。

^

+

🔖

🔗

```
//webpack.config.js
module.exports = {
  ...
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        use: {
          loader: "babel-loader"
        },
        exclude: /node_modules/
      },
      {
        test: /\.css$/,
        use: [
          {
            loader: "style-loader"
          }, {
            loader: "css-loader",
            options: {
              modules: true
            }
          }, {
            loader: "postcss-loader"
          }
        ]
      }
    ]
  }
}
```

```
// postcss.config.js
module.exports = {
  plugins: [
    require('autoprefixer')
  ]
}
```

至此，本文已经谈论了处理JS的Babel和处理CSS的PostCSS的基本用法，它们其实也是两个单独的平台，配合 webpack 可以很好的发挥它们的作用。接下来介绍Webpack中另一个非常重要的功能-Plugins

插件 (Plugins)

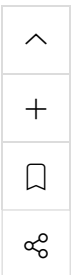
插件 (Plugins) 是用来拓展Webpack功能的，它们会在整个构建过程中生效，执行相关的任务。

Loaders和Plugins常常被弄混，但是他们其实是完全不同的东西，可以这么说，loaders是在打包构建过程中用来处理源文件的（JSX，Scss，Less..），一次处理一个，插件并不直接操作单个文件，它直接对整个构建过程起作用。

Webpack有很多内置插件，同时也有很多第三方插件，可以让我们完成更加丰富的功能。

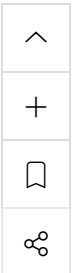
使用插件的方法

要使用某个插件，我们需要通过 npm 安装它，然后要做的就是在webpack配置中的plugins关键字部分添加该插件的一个实例（plugins是一个数组）继续上面的例子，我们添加了一个给打包后代码添加版权声明的插件 (<https://link.jianshu.com?t=https://webpack.js.org/plugins/banner-plugin/>)。

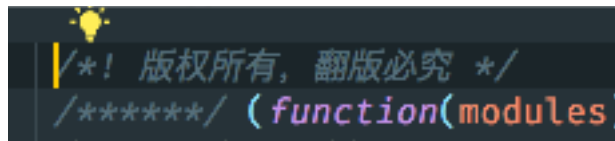



```
const webpack = require('webpack');

module.exports = {
  ...
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        use: {
          loader: "babel-loader"
        },
        exclude: /node_modules/
      },
      {
        test: /\.css$/,
        use: [
          {
            loader: "style-loader"
          }, {
            loader: "css-loader",
            options: {
              modules: true
            }
          }, {
            loader: "postcss-loader"
          }
        ]
      }
    ]
  },
  plugins: [
    new webpack.BannerPlugin('版权所有，翻版必究')
  ],
};
```



通过这个插件，打包后的JS文件显示如下



版权所有，翻版必究

这就是webpack插件的基础用法了，下面给大家推荐几个常用的插件

HtmlWebpackPlugin

这个插件的作用是依据一个简单的 `index.html` 模板，生成一个自动引用你打包后的JS文件的新 `index.html`。这在每次生成的js文件名称不同时非常有用（比如添加了 hash 值）。

安装

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

这个插件自动完成了我们之前手动做的一些事情，在正式使用之前需要对一直以来的项目结构做一些更改：

1. 移除public文件夹，利用此插件，`index.html` 文件会自动生成，此外CSS已经通过前面的操作打包到JS中了。
2. 在app目录下，创建一个 `index.tpl.html` 文件模板，这个模板包含 `title` 等必须元素，在编译过程中，插件会依据此模板生成最终的html页面，会自动添加所依赖的 `css`, `js`, `favicon`等文件，`index.tpl.html` 中的模板源代码如下：

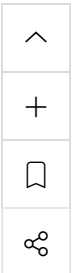
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Webpack Sample Project</title>
  </head>
  <body>
    <div id='root'>
    </div>
  </body>
</html>
```

3.更新 webpack 的配置文件,方法同上,新建一个 build 文件夹用来存放最终的输出文件

```
const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: __dirname + "/app/main.js",//已多次提及的唯一入口文件
  output: {
    path: __dirname + "/build",
    filename: "bundle.js"
  },
  devtool: 'eval-source-map',
  devServer: {
    contentBase: "./public",//本地服务器所加载的页面所在的目录
    historyApiFallback: true,//不跳转
    inline: true//实时刷新
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        use: {
          loader: "babel-loader"
        },
        exclude: /node_modules/
      },
      {
        test: /\.css$/,
        use: [
          {
            loader: "style-loader"
          }, {
            loader: "css-loader",
            options: {
              modules: true
            }
          }, {
            loader: "postcss-loader"
          }
        ]
      }
    ]
  },
  plugins: [
    new webpack.BannerPlugin('版权所有,翻版必究'),
    new HtmlWebpackPlugin({
      template: __dirname + "/app/index.tmpl.html"//new 一个这个插件的实例,并传入相
    })
  ],
};
```

再次执行 `npm start` 你会发现, build 文件夹下面生成了 `bundle.js` 和 `index.html`。





build文件夹

Hot Module Replacement

Hot Module Replacement (HMR) 也是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；

还是继续上例来实际看看如何配置

^

+

🔖

🔗

```
const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: __dirname + "/app/main.js", //已多次提及的唯一入口文件
  output: {
    path: __dirname + "/build",
    filename: "bundle.js"
  },
  devtool: 'eval-source-map',
  devServer: {
    contentBase: "./public", //本地服务器所加载的页面所在的目录
    historyApiFallback: true, //不跳转
    inline: true,
    hot: true
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        use: {
          loader: "babel-loader"
        },
        exclude: /node_modules/
      },
      {
        test: /\.css$/,
        use: [
          {
            loader: "style-loader"
          }, {
            loader: "css-loader",
            options: {
              modules: true
            }
          }, {
            loader: "postcss-loader"
          }
        ]
      }
    ]
  },
  plugins: [
    new webpack.BannerPlugin('版权所有，翻版必究'),
    new HtmlWebpackPlugin({
      template: __dirname + "/app/index.tpl.html" //new 一个这个插件的实例，并传入相
    }),
    new webpack.HotModuleReplacementPlugin() //热加载插件
  ],
};
```

安装 react-transform-hmr

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

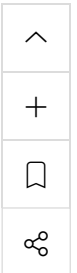
配置Babel

```
// .babelrc
{
  "presets": ["react", "env"],
  "env": {
    "development": {
      "plugins": [["react-transform", {
        "transforms": [{
          "transform": "react-transform-hmr",

          "imports": ["react"],

          "locals": ["module"]
        }
      ]]]
    }
  }
}
```

现在当你使用React时，可以热加载模块了,每次保存就能在浏览器上看到更新内容。



产品阶段的构建

目前为止，我们已经使用webpack构建了一个完整的开发环境。但是在产品阶段，可能还需要对打包的文件进行额外的处理，比如说优化，压缩，缓存以及分离CSS和JS。

对于复杂的项目来说，需要复杂的配置，这时候分解配置文件为多个小的文件可以使得事情井井有条，上面的例子来说，我们创建一个 `webpack.production.config.js` 的文件，在里面加上基本的配置,它和原始的`webpack.config.js`很像，如下

```
// webpack.production.config.js
const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: __dirname + "/app/main.js", //已多次提及的唯一入口文件
  output: {
    path: __dirname + "/build",
    filename: "bundle.js"
  },
  devtool: 'null', //注意修改了这里，这能大大压缩我们的打包代码
  devServer: {
    contentBase: "./public", //本地服务器所加载的页面所在的目录
    historyApiFallback: true, //不跳转
    inline: true,
    hot: true
  },
  module: {
    rules: [{
      test: /\.jsx?$/,
      use: {
        loader: "babel-loader"
      },
      exclude: /node_modules/
    }, {
      test: /\.css$/,
      use: ExtractTextPlugin.extract({
        fallback: "style-loader",
        use: [{
          loader: "css-loader",
          options: {
            modules: true
          }
        }, {
          loader: "postcss-loader"
        }
      ]
    })
  ]
},
  plugins: [
    new webpack.BannerPlugin('版权所有，翻版必究'),
    new HtmlWebpackPlugin({
      template: __dirname + "/app/index.tmpl.html" //new 一个这个插件的实例，并传入模板文件
    }),
    new webpack.HotModuleReplacementPlugin() //热加载插件
  ],
};
```



```
//package.json
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "webpack",
    "server": "webpack-dev-server --open",
    "build": "NODE_ENV=production webpack --config ./webpack.production.config.js --progress",
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    ...
  },
  "dependencies": {
    "react": "^15.6.1",
    "react-dom": "^15.6.1"
  }
}
```

注意:如果是window电脑，build 需要配置为 "build": "set NODE_ENV=production && webpack --config ./webpack.production.config.js --progress" .谢谢评论区简友提醒。

优化插件

webpack提供了一些在发布阶段非常有用的优化插件，它们大多来自于webpack社区，可以通过npm安装，通过以下插件可以完成产品发布阶段所需的功能

- OccurenceOrderPlugin :为组件分配ID，通过这个插件webpack可以分析和优先考虑使用最多的模块，并为它们分配最小的ID
- UglifyJsPlugin : 压缩JS代码；
- ExtractTextPlugin : 分离CSS和JS文件

我们继续用例子来看看如何添加它们，OccurenceOrder 和 UglifyJS plugins 都是内置插件，你需要做的只是安装其它非内置插件

```
npm install --save-dev extract-text-webpack-plugin
```


在配置文件的plugins后引用它们



```
// webpack.production.config.js
const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const ExtractTextPlugin = require('extract-text-webpack-plugin');

module.exports = {
  entry: __dirname + "/app/main.js", //已多次提及的唯一入口文件
  output: {
    path: __dirname + "/build",
    filename: "bundle.js"
  },
  devtool: 'none',
  devServer: {
    contentBase: "./public", //本地服务器所加载的页面所在的目录
    historyApiFallback: true, //不跳转
    inline: true,
    hot: true
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        use: {
          loader: "babel-loader"
        },
        exclude: /node_modules/
      },
      {
        test: /\.css$/,
        use: [
          {
            loader: "style-loader"
          }, {
            loader: "css-loader",
            options: {
              modules: true
            }
          }, {
            loader: "postcss-loader"
          }
        ]
      }
    ]
  },
  plugins: [
    new webpack.BannerPlugin('版权所有, 翻版必究'),
    new HtmlWebpackPlugin({
      template: __dirname + "/app/index.html"
    }),
    new webpack.optimize.OccurrenceOrderPlugin(),
    new webpack.optimize.UglifyJsPlugin(),
    new ExtractTextPlugin("style.css")
  ],
};
```

此时执行 `npm run build` 可以看见代码是被压缩后的



压缩后的代码

缓存

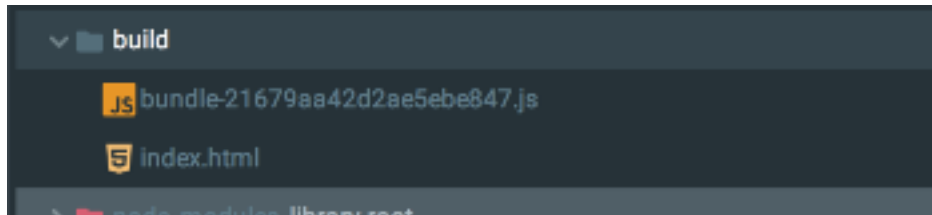
缓存无处不在, 使用缓存的最好方法是保证你的文件名和文件内容是匹配的 (内容改变, 名称相应改变)

webpack可以把一个哈希值添加到打包的文件名中,使用方法如下,添加特殊的字符串混合体 ([name], [id] and [hash]) 到输出文件名前

```
const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const ExtractTextPlugin = require('extract-text-webpack-plugin');

module.exports = {
  ..
  output: {
    path: __dirname + "/build",
    filename: "bundle-[hash].js"
  },
  ...
};
```

现在用户会有合理的缓存了。



带hash值的js名

去除 build 文件中的残余文件

添加了 hash 之后,会导致改变文件内容后重新打包时,文件名不同而内容越来越多,因此这里介绍另外一个很好用的插件 clean-webpack-plugin。

安装：

```
cnpm install clean-webpack-plugin --save-dev
```

使用：

引入 clean-webpack-plugin 插件后在配置文件的 plugins 中做相应配置即可：

```
const CleanWebpackPlugin = require("clean-webpack-plugin");
plugins: [
  ...// 这里是之前配置的其它各种插件
  new CleanWebpackPlugin('build/*.*', {
    root: __dirname,
    verbose: true,
    dry: false
  })
]
```

关于 clean-webpack-plugin 的详细使用可参考这里 (<https://link.jianshu.com?t=https://github.com/johnagan/clean-webpack-plugin>)

总结

其实这是一年前的文章了,趁周末重新运行和修改了一下,现在所有的代码都可以正常运行,所用webpack基于最新的 webpack3.5.3。希望依旧能对你有帮助。

这是一篇好长的文章,谢谢你的耐心,能仔细看到了这里,大概半个月我第一次自己一步步配置项目所需的Webpack后就一直想写一篇笔记做总结,几次动笔都不能让自己满意,总觉得写不清楚。其实关于Webpack本文讲述得仍不完全,不过相信你看完后已经进入Webpack的大门,能够更好的探索其它的关于Webpack的知识了。

欢迎大家在文后发表自己的观点讨论。

更新说明

2017-12-11更新，修改 `css module` 部分代码及示例图片，`css module` 真的非常好用，希望大家都能用上。

2017年9月18日更新，添加了一个使用 `webpack` 配置多页应用的demo,可以点击此处查看 (<https://link.jianshu.com?t=https://github.com/zhangwang1990/blogs/tree/master/sources/MultiPageWebpackDemos>)

2017年8月13日更新，本文依据 `webpack3.5.3` 将文章涉及代码完全重写，所有代码都在Mac上正常运行过。希望依旧对你学习 `webpack` 有帮助。

2017年8月16号更新：

最近在Gitchat上将发起了一场关于webpack的分享，目的在于一起花最短的时间理解和学会webpack，感兴趣的童鞋可以微信扫描注册哈。



webpack从入门到工程实践

^

+

🔖

🔗

小礼物走一走，来简书关注我

赞赏支持



(/u/524edf)

📖 前端笔记 (/nb/3227908)

举报文章 © 著作权归作者所有



zhangwang (/u/7091a52ac9e5)
写了 197035 字，被 3014 人关注，获得了 3190 个喜欢
(/u/7091a52ac9e5)

+ 关注

前端摄影阅读好奇。

喜欢 2540



更多分享

(<http://cwb.assets.jianshu.io/notes/images/5124994/v>)



写下你的评论...

787条评论

只看作者

按喜欢排序 按时间正序 按时间倒序



叶隐_faf1 (/u/e2f11fd62dd4)

453楼 · 2018.01.19 11:35

(/u/e2f11fd62dd4)

写的灰常好啊,初学者的福音

赞 回复



彼岸13686687410 (/u/a97059eefe45)

452楼 · 2018.01.17 09:06

(/u/a97059eefe45)

为什么npm run server 就这样了

Invalid configuration object. Webpack has been initialised using a configuration object that does not match the API schema.

- configuration.module.rules[0] has an unknown property 'text'. These properties are valid: object { enforce?, exclude?, include?, issuer?, loader?, loaders?, oneOf?, options?, parser?, query?, resource?, resourceQuery?, compiler?, rules?, test?, use? }

-> A rule

ode ELIFECYCLE

npm ERR! ermo 1

npm ERR! my-webpack@1.0.0 server: `webpack-dev-server --open`

npm ERR! Exit status 1

npm ERR!

npm ERR! Failed at the my-webpack@1.0.0 server script.

npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:

npm ERR! C:\Users\wt910\AppData\Roaming\npm-cache_logs\2018-01-17T00_59_03_973Z-debug.log 是什么原因呢。

赞 回复



听说_bc29 (/u/0c0d31651d82)

451楼 · 2018.01.16 17:06

(/u/0c0d31651d82)

非常感谢作者的整理,写得非常好。这里我有一个小问题,在用这个插件的时候一直报 cannot find module debug,我很懵。

赞 回复



不找了、 (/u/37240763b9c2)

450楼 · 2018.01.15 11:36

(/u/37240763b9c2)

很赞,博主辛苦了

赞 回复



萱苏m (/u/4940993c9046)

449楼 · 2018.01.12 18:47

(/u/4940993c9046)

太赞了,终于全部跑完了,中间各种出问题,总算——解决,有的问题也不知道是为

啥,折腾折腾就解决,有的问题的解决就了解到了新知识,谢谢楼主,跟着楼主的文章做完一遍,瞬间觉得webpack以及好多人家的项目不那么陌生了,后面的路还长,继续努力

赞 回复





董苏m (/u/4940993c9046)

448楼 · 2018.01.12 17:00

(/u/4940993c9046)

两个问题，1、react热启动那块是不是少写了个安装react-hot-loader；

2、请问一下，为什么我的webpack-dev-server在按ctrl+c退出后并没有退出进程，需要去任务管理器结束node.exe进程才可以。

赞 回复



y_2897 (/u/ab3855e9d135)

447楼 · 2018.01.11 16:50

(/u/ab3855e9d135)

在优化插件这里 npm run build 就报错.....

npm run build

```
> webpack@1.0.0 build E:\sirbin\1\webpack
> set NODE_ENV=production && webpack --progress --colors --config
webpack.production.config.js
```

E:\sirbin\1\webpack\webpack.production.config.js:27

use: ExtractTextPlugin.extract({

^

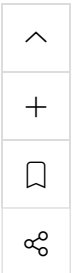
```
ReferenceError: ExtractTextPlugin is not defined
at Object.<anonymous> (E:\sirbin\1\webpack\webpack.production.config.js:27:18)
at Module._compile (module.js:635:30)
at Object.Module._extensions..js (module.js:646:10)
at Module.load (module.js:554:32)
at tryModuleLoad (module.js:497:12)
at Function.Module._load (module.js:489:3)
at Module.require (module.js:579:17)
at require (internal/module.js:11:18)
at requireConfig
(E:\sirbin\1\webpack\node_modules\_webpack@3.10.0@webpack\bin\convert-
argv.js:97:18)
at E:\sirbin\1\webpack\node_modules\_webpack@3.10.0@webpack\bin\convert-
argv.js:104:17
at Array.forEach (<anonymous>)
at module.exports
(E:\sirbin\1\webpack\node_modules\_webpack@3.10.0@webpack\bin\convert-
argv.js:102:15)
at yargs.parse
(E:\sirbin\1\webpack\node_modules\_webpack@3.10.0@webpack\bin\webpack.js:171:4
1)
at Object.Yargs.self.parse
(E:\sirbin\1\webpack\node_modules\_yargs@8.0.2@yargs\yargs.js:533:18)
at Object.<anonymous>
(E:\sirbin\1\webpack\node_modules\_webpack@3.10.0@webpack\bin\webpack.js:152:7)
at Module._compile (module.js:635:30)
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! webpack@1.0.0 build: `set NODE_ENV=production && webpack --progress --
colors --config webpack.production.config.js`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the webpack@1.0.0 build script.
npm ERR! This is probably not a problem with npm. There is likely additional logging
output above.
```

npm ERR! A complete log of this run can be found in:

npm ERR! C:\Users\YH\AppData\Roaming\npm-cache_logs\2018-01-11T08_49_04_666Z-debug.log

这是怎么回事?

赞 回复





y_2897 (/u/ab3855e9d135)

446楼 · 2018.01.11 14:52

(/u/ab3855e9d135)

在引入插件的时候,怎么就报错了

ndex.html 244 bytes [emitted]

[0] ./app/main.js 258 bytes {0} [built] [failed] [1 error]

ERROR in ./app/main.js

Module build failed: SyntaxError: E:/sirbin/1/webpack/app/main.js: Unexpected token (6:7)

4 | import Greeter from './Greeter';

5 | import './main.css';

> 6 | render(<Greeter />, document.getElementById('root'));

| ^

Child html-webpack-plugin for "index.html":

1 asset

[0] ./node_modules/_html-webpack-plugin@2.30.1@html-webpack-plugin/lib/loader.js!./app/index.html 581 bytes {0} [built]

[2] (webpack)/buildin/global.js 509 bytes {0} [built]

[3] (webpack)/buildin/module.js 517 bytes {0} [built]

+ 1 hidden module

npm ERR! code ELIFECYCLE

npm ERR! errno 2

npm ERR! webpack@1.0.0 start: `webpack`

npm ERR! Exit status 2

npm ERR!

npm ERR! Failed at the webpack@1.0.0 start script.

npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:

npm ERR! C:\Users\YH\AppData\Roaming\npm-cache_logs\2018-01-

11T06_51_02_576Z-debug.log

赞 回复



董苏m (/u/4940993c9046)

445楼 · 2018.01.11 14:20

(/u/4940993c9046)

\$ npm run server

> webpack@1.0.0 server D:\web\webpack

> webpack-dev-server --open

events.js:182

throw er; // Unhandled 'error' event

^

Error: listen EADDRINUSE 127.0.0.1:8080

at Object.exports._errnoException (util.js:1022:11)

at exports._exceptionWithHostPort (util.js:1045:20)

at Server.setupListenHandle [as _listen2] (net.js:1315:14)

at listenInCluster (net.js:1363:12)

at GetAddrInfoReqWrap.doListen [as callback] (net.js:1489:7)

at GetAddrInfoReqWrap.onlookup [as oncomplete] (dns.js:96:10)

npm ERR! code ELIFECYCLE

npm ERR! errno 1

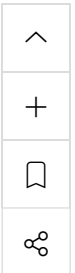
npm ERR! webpack@1.0.0 server: `webpack-dev-server --open`

npm ERR! Exit status 1

npm ERR!

npm ERR! Failed at the webpack@1.0.0 server script.

npm ERR! This is probably not a problem with npm. There is likely additional logging output above.



npm ERR! A complete log of this run can be found in:
npm ERR! C:\Users\mlamp\AppData\Roaming\npm-cache_logs\2018-01-11T06_19_04_249Z-debug.log
这个是为什么啊, npm start可以启动, 就是npm run server报错

赞 回复



null_d430 (/u/ccd93eea92c3)

444楼 · 2018.01.11 13:58

(/u/ccd93eea92c3)
npm run build 报错了

G:\webpack\webpack-one>npm run build

```
> webpack-one@1.0.0 build G:\webpack\webpack-one
> set NODE_ENV=production && webpack --config ./webpack.production.config.js --progress
```

```
module.js:557
throw err;
^
```

```
Error: Cannot find module 'webpack'
at Function.Module._resolveFilename (module.js:555:15)
at Function.Module._load (module.js:482:25)
at Module.require (module.js:604:17)
at require (internal/module.js:11:18)
at Object.<anonymous> (G:\webpack\webpack-one\webpack.production.config.js:1:79)
at Module._compile (module.js:660:30)
at Object.Module._extensions.js (module.js:671:10)
at Module.load (module.js:573:32)
at tryModuleLoad (module.js:513:12)
at Function.Module._load (module.js:505:3)
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! webpack-one@1.0.0 build: `set NODE_ENV=production && webpack --config ./webpack.production.config.js --progress`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the webpack-one@1.0.0 build script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.
```

npm ERR! A complete log of this run can be found in:
npm ERR! C:\Users\Administrator\AppData\Roaming\npm-cache_logs\2018-01-11T05_55_14_539Z-debug.log

赞 回复



y_2897 (/u/ab3855e9d135)

443楼 · 2018.01.10 19:15

(/u/ab3855e9d135)
老铁,你的文件夹命名 webpack sample project ,我初学的直接就报错.....windows下的,直接就报了个npm ERR! Callback called more than once.

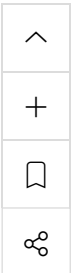
赞 回复



写优雅 (/u/532f5a10d5e6)


442楼 · 2018.01.10 15:41

(/u/532f5a10d5e6)
产品阶段的构建 标题下的第一个源码 webpack.production.config.js 的配置, 没有声明
const ExtractTextPlugin = require('extract-text-webpack-plugin');
虽然后面已经补上了, 但是我是一边看, 一边粘贴源码调试的时候发现的, 如果加上就



完美了 ☐

赞 回复



恋怵止迟 (/u/0220fda35a37)

441楼 · 2018.01.10 12:00

(/u/0220fda35a37)

我怎么给css添加前缀失败，但打包成功了

赞 回复




MartinQ (/u/f1ceb3d109b1)

440楼 · 2018.01.10 10:00

(/u/f1ceb3d109b1)

感谢您的文章！所有用中文写的技术好文章都应该极力点赞的，中国的技术进步需要你们！😁

赞 回复



null_d430 (/u/ccd93eea92c3)

439楼 · 2018.01.09 16:51

(/u/ccd93eea92c3)

我按照你的写，我输入npm run server 怎么报错了？

赞 回复

上一页

1

2

3

4

5

下一页

^

+

🔖

🔗

被以下专题收入，发现更多相似内容

- + 收入我的专题
- 

前端之旅 (/c/bfb5f14ca7fe?utm_source=desktop&utm_medium=notes-included-collection)
- 

首页投稿 (/c/bDHhpK?utm_source=desktop&utm_medium=notes-included-collection)
- 

前端开发那些事 (/c/0020d95b7928?utm_source=desktop&utm_medium=notes-included-collection)
- 

程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)
- 

MobDevG... (/c/1f37d896b5de?utm_source=desktop&utm_medium=notes-included-collection)
- 

全栈开发者 (/c/794aaea3cf01?utm_source=desktop&utm_medium=notes-included-collection)
- 

Web前段开发记录 (/c/baced03d8a91?utm_source=desktop&utm_medium=notes-included-collection)

展开更多 ▾

推荐阅读 更多精彩内容 > (/)

(React启蒙)认识React nodes (/p/3cadf6faa7a8?utm_campaign=males...
React nodes 本章是翻译的React启蒙系列的第四章，主要将讲述如何使用纯JavaScript语句创建React节点，本章内容依旧非常基础，通过阅读本章内容你将了解React nodes的定义，React.createElement()所需的各...

zhangwang (/u/7091a52ac9e5?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

我的2016总结和2017目标 (/p/513b5faa661b?utm_campaign=maleskine...

总结篇 2016年是我的转折年，这一年经历了很多事情，世界观和价值观其实都可以说有了多次的改变，记录以下事情17年底总结时用以对比： 实践转行：放下学习七年的专业，奔向一个未知的领域，是一个比较难...

zhangwang (/u/7091a52ac9e5?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

第一次请姑娘吃大餐，如何才能博得她的好感？ (/p/70...

(/p/7024de254fae?
utm_campaign=maleskine&utm_content=note&utm_r

文 | 义琳 从周三开始，前台的姑娘发现失恋了大半年的Sum哥又焕发了第二春。 万年不变的格子衬衫换成了精梳棉水洗蓝色的衬衫，趴在头顶的一头鸡窝发...

义琳 (/u/0d59bca0b480?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

《前任3》：没看到爱情，只看到了少女的油腻 (/p/b1c...

(/p/b1c162d31f71?
utm_campaign=maleskine&utm_content=note&utm_r

趁着元旦放假，跑到电影院遛了一圈，怀着想看喜剧的心情，选了《前任3》， 结果在笑声不断的影院，我被剧情震惊到，陷入深思。 故事很简单，总结起...

Miss余点 (/u/f351af6d6c85?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

免费分享课：O基础写作，如何一年入账O元？ (/p/26...

(/p/26eb0504d548?
utm_campaign=maleskine&utm_content=note&utm_r

我先说三点： 其一，这是一次免费课程。既然一年入账O元，我也确实没脸收 费。 其二，这不是绝对的原创。有前辈的经验之谈，也有我自己写作十年的...

尹洁城 (/u/f1b13122a132?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

翻译 | 上手 Webpack？这篇就够了！ (/p/ca76ae7652e7?utm_campaign...

作者：小 boy（沪江前端开发工程师） 本文原创，转载请注明作者及出处。原文地址：
https://www.smashingmagazine.com/2017/02/a-detailed-introduction-to-webpack JavaScript 模块化打包已...

iKcamp (/u/8d80133b96e5?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/b83a251d53db?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

你一定喜欢看的 Webpack 2.x 入门实战 (/p/b83a251d53db?utm_campai...

最近在学习 Webpack,网上大多数入门教程都是基于 Webpack 1.x 版本的,我学习 Webpack 的时候是看了 zhangwang 的 <<入门 Webpack, 看这篇就够了>> 写的非常好,不过是基于 Webpack 1.x 版本的,语法上...

My_Oh_My (/u/666067560d68?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/5b69a7e61fe4?




utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

webpack教程 (/p/5b69a7e61fe4?utm_campaign=maleskine&utm_cont...


无意中看到zhangwnag大佬分享的webpack教程感觉受益匪浅，特此分享以备自己日后查看，也希望更多的

人看到这篇好的文章：http://www.jianshu.com/p/42e11515c10f 写在前面的话阅读本文之前，先看下面这...

 小小字符 (/u/1689862fc5a0?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


webpack 从入门到工程实践 (/p/9349c30a6b3e?utm_campaign=maleski...

GitChat技术杂谈 前言 本文较长，为了节省你的阅读时间，在文前列写作思路如下：什么是 webpack，它要解决的是什么问题？对webpack的主要配置项进行分析，虽然不会涉及太多细节，但是期待在本节能让我...

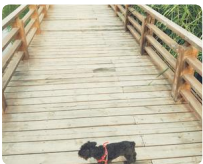
 玄辞 (/u/654b04d75afb?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

无标题文章 (/p/d074301c1bfa?utm_campaign=maleskine&utm_conten...

从V1迁移到V2由于使用的是webpack版本是2.2.1，所以针对原文做了一些修改。针对webpack2的修改部分和添加的部分在最底部，文中已经改过来了。写在前面的话阅读本文之前，先看下面这个webpack的配置...

 yzc123446 (/u/304734d3ba47?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


(/p/aacd587ae852?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)


夫妻关系成长第19天---爱是舍得花时间 (/p/aacd587ae852?utm_campaig...

夫妻关系成长作业（8月13日）：爱是舍得花时间 ♥特意取消你平日常做的某件事，花时间和你的伴侣好好欢聚。做些他喜欢的事，或是开始某个他很想进行的计划，总之要在一起。如果他不愿意一起做，请尝试...

 感恩有你李瑞 (/u/13e0d53ed96f?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


狗 (/p/8d13197a2f90?utm_campaign=maleskine&utm_content=note&...

婆婆嘴

 ymm201314 (/u/cc430be9bc98?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


《念情伤》 (/p/521604bc9466?utm_campaign=maleskine&utm_conten...

梦断长安殇别离。而今思情明月时，梅落回首心戚戚。作品导读：此诗配古琴曲《长门怨》，当年长安，金屋藏娇，到头来，只余长门赋，世事变迁，难得长久，思情浮现，知己曾有期盼终身之人，而今却因病...

 林端木 (/u/03697c683ed6?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

依赖 (/p/c76433765481?utm_campaign=maleskine&utm_content=not...

最好的爱情，依赖而不纠缠。或许每个人都应该具备这么一个能力，一个局内局外切换的能力。当自己身处局内，一切便方寸大乱，没了自我。咱们的感情或许是时候升级了，升级到一个新的阶段，升级到一个更...

 田老九 (/u/be139214c52b?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/964585cc49f1?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

叶 (/p/964585cc49f1?utm_campaign=maleskine&utm_content=note&...

十分喜欢开着鲜花的植物，或锦绣繁荣或清雅美丽，但也很喜欢看到那些散种在土中的小小的、形态各异的


^

+

🔖

🔗

花花草草，充满了山野之趣。越是一小株越是惹人怜爱。小而完整，有的甚至还开着小小的花。一两片叶...

 夏栀、 (/u/a2bad928e034?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

