

webpack：从入门到真实项目配置

盒子爱上猫

新晋铲屎官一枚

关注她

124 人赞同了该文章

本文原载于掘金，作者夕阳（饥人谷学员），转载已获作者授权。

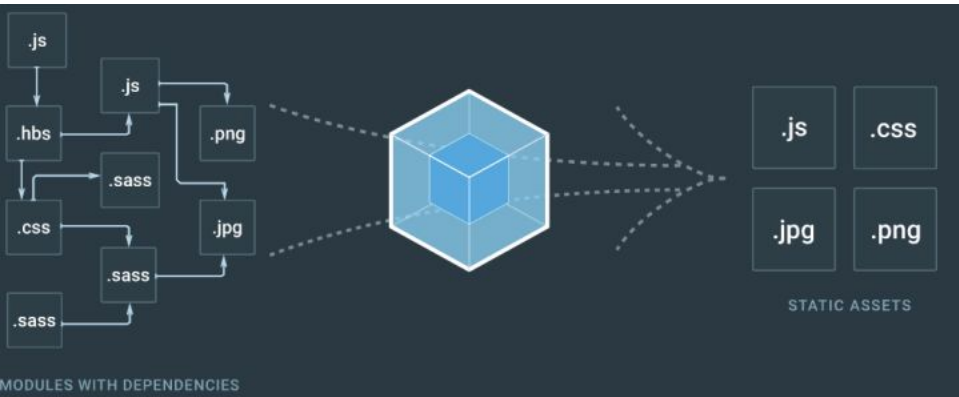
该文使用的 Webpack 版本为 3.6.0，本文分两部分。第一步是简单的使用 webpack，第二部分通过一个真实项目来配置 webpack，没有使用任何的 CLI，都是一步步配置直到完成生产代码的打包。这是本项目对应的仓库，每个小节基本都对应了一次 commit。

这是本文的大纲，如果觉得有兴趣你就可以往下看了



Webpack 到底是什么

自从出现模块化以后，大家可以将原本一坨代码分离到个个模块中，但是由此引发了一个问题。每个 JS 文件都需要从服务器去拿，由此会导致加载速度变慢。Webpack 最主要的目的就是为了解决这个问题，将所有小文件打包成一个或多个大文件，官网的图片很好的诠释了这个事情，除此之外，Webpack 也是一个能让你使用各种前端新技术的工具。



简单使用

安装

在命令行中依次输入

```
mkdir webpack-demo
cd webpack-demo
// 创建 package.;
```

赞同 124

13 条评论

分享

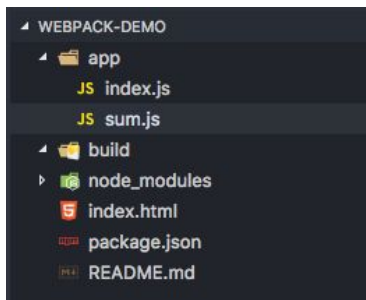
收藏

...



```
npm init
// 推荐这个安装方式，当然你也安装在全局环境下
// 这种安装方式会将 webpack 放入 devDependencies 依赖中
npm install --save-dev webpack
```

然后按照下图创建文件



在以下文件写入代码

```
// sum.js
// 这个模块化写法是 node 环境独有的，浏览器原生不支持使用
module.exports = function(a, b) {
  return a + b
}
// index.js
var sum = require('./sum')
console.log(sum(1, 2))
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  <div id="app"></div>
  <script src="./build/bundle.js"></script>
</body>
</html>
```

现在我们将开始配置最简单的 webpack，首先创建 webpack.config.js 文件，然后写入如下代码

```
// 自带的库
const path = require('path')
module.exports = {
  entry: './app/index.js', // 入口文件
  output: {
    path: path.resolve(__dirname, 'build'), // 必须使用绝对地址，输出文件夹
    filename: "bundle.js" // 打包后输出文件的文件名
  }
}
```

现在我们可以开始使用 webpack 了，在命令行中输入

```
node_modules/.bin/webpack
```

没问题的话你应该可以看到类似的样子



```

→ webpack-demo (master) X node_modules/.bin/webpack
Hash: 096a49efe8f8f10b9734 这次打包的哈希值
Version: webpack 3.5.6 版本号
Time: 156ms 花费的时间 为什么打包出来的变大那么多?
  Asset      Size  Chunks             Chunk Names
bundle.js    2.66 kB      0  [emitted]  main
  [0] ./app/index.js 49 bytes {0} [built]
  [1] ./app/sum.js 52 bytes {0} [built]

```

可以发现原本两个 JS 文件只有 100B，但是打包后却增长到 2.66KB，这之中 webpack 肯定做了什么事情，我们去 bundle.js 文件中看看。

把代码简化以后，核心思路是这样的

```

var array = [(function () {
    var sum = array[1]
    console.log(sum(1, 2))
}),
(function (a,b) {
    return a + b
})]
array[0]() // -> 3

```

因为 module.export 浏览器是不支持的，所以 webpack 将代码改成浏览器能识别的样子。现在将 index.html 文件在浏览器中打开，应该也可以看到正确的 log。

我们之前是在文件夹中安装的 webpack，每次要输入 node_modules/.bin/webpack 过于繁琐，可以在 package.json 如下修改

```

"scripts": {
  "start": "webpack"
},

```

然后再次执行 npm run start，可以发现和之前的效果是相同的。简单的使用到此为止，接下来我们来探索 webpack 更多的功能。

Loader

Loader 是 webpack 一个很强大功能，这个功能可以让你使用很多新的技术。

Babel

Babel 可以让你使用 ES2015/16/17 写代码而不用顾忌浏览器的问题，Babel 可以帮你转换代码。首先安装必要的几个 Babel 库

```
npm i --save-dev babel-loader babel-core babel-preset-env
```

先介绍下我们安装的两个库

- babel-loader 用于让 webpack 知道如何运行 babel
- babel-core 可以看做编译器，这个库知道如何解析代码
- babel-preset-env 这个库可以根据环境的不同转换代码

接下来更改 webpack-config.js 中的代码

```

module.exports = {
  // .....
  module: {

```

▲ 赞同 124 ▼ 13 条评论 分享 收藏 ...



```
rules: [
  {
    // js 文件才使用 babel
    test: /\.js$/,
    // 使用哪个 loader
    use: 'babel-loader',
    // 不包括路径
    exclude: /node_modules/
  }
]
```

配置 Babel 有很多方式，这里推荐使用 .babelrc 文件管理。

```
// ..babelrc
{
  "presets": ["babel-preset-env"]
}
```

现在将之前 JS 的代码改成 ES6 的写法

```
// sum.js
export default (a, b) => {
  return a + b
}
// index.js
import sum from './sum'
console.log(sum(1, 2))
```

执行 npm run start，再观察 bundle.js 中的代码，可以发现代码被转换过了，并且同样可以正常输出3。

当然 Babel 远不止这些功能，有兴趣的可以前往官网自己探索。

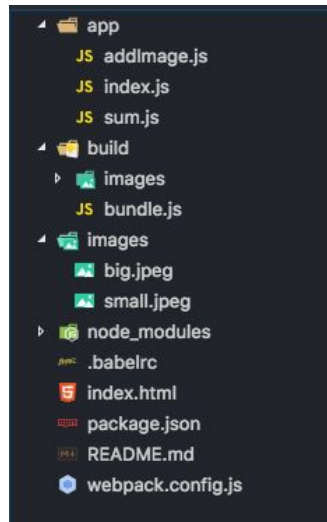
处理图片

这一小节我们将使用 url-loader 和 file-loader，这两个库不仅可以处理图片，还有其他的功能，有兴趣的可以自行学习。

先安装库

```
npm i --save-dev url-loader file-loader
```

创建一个 images 文件夹，放入两张图片，并且在 app 文件夹下创建一个 js 文件处理图片，目前的文件夹结构如图



```
// addImage.js
let smallImg = document.createElement('img')
// 必须 require 进来
smallImg.src = require('../images/small.jpeg')
document.body.appendChild(smallImg)

let bigImg = document.createElement('img')
bigImg.src = require('../images/big.jpeg')
document.body.appendChild(bigImg)
```

接下来修改 webpack.config.js 代码

```
module.exports = {
  // ...
  module: {
    rules: [
      // ...
      {
        // 图片格式正则
        test: /\.?(png|jpe?g|gif|svg)(\?.*)?$/,
        use: [
          {
            loader: 'url-loader',
            // 配置 url-loader 的可选项
            options: {
              // 限制 图片大小 10000B, 小于限制会将图片转换为 base64格式
              limit: 10000,
              // 超出限制, 创建的文件格式
              // build/images/[图片名].[hash].[图片格式]
              name: 'images/[name].[hash].[ext]'}
            }
          ]
        }
      ]
    }
  }
}
```

运行 npm run start , 打包成功如下图



```

Hash: afd5697d5e9012bd4bf5
Version: webpack 3.5.6
Time: 4139ms

      Asset      Size  Chunks
  Chunk Names
images/big.d364620e5312f4427a916c11c6065311.jpeg  303 kB      [emitted] [big
]
      bundle.js  13.5 kB      0 [emitted]

  main
  [0] ./app/index.js 247 bytes {0} [built]
  [1] ./app/sum.js 139 bytes {0} [built]
  [2] ./app/addImage.js 269 bytes {0} [built]
  [3] ./images/small.jpeg 10 kB {0} [built]
  [4] ./images/big.jpeg 94 bytes {0} [built]

```

可以发现大的图片被单独提取了出来，小的图片打包进了 bundle.js 中。

在浏览器中打开 HTML 文件，发现小图确实显示出来了，但是却没有看到大图，打开开发者工具栏，可以发现我们大图的图片路径是有问题的，所以我们要修改 webpack.config.js 代码了。

```

module.exports = {
  entry: './app/index.js', // 入口文件
  output: {
    path: path.resolve(__dirname, 'build'), // 必须使用绝对地址，输出文件夹
    filename: "bundle.js", // 打包后输出文件的文件名
    publicPath: 'build/' // 知道如何寻找资源
  }
  // ...
}

```

最后运行下 npm run start，编译成功了，再次刷新下页面，可以发现这次大图被正确的显示了。下一小节我们将介绍如何处理 CSS 文件。

处理 CSS 文件

添加 styles 文件夹，新增 addImage.css 文件，然后在该文件中新增代码

```

img {
  border: 5px black solid;
}
.test {border: 5px black solid;}

```

这一小节我们先使用 css-loader 和 style-loader 库。前者可以让 CSS 文件也支持 impost，并且会解析 CSS 文件，后者可以将解析出来的 CSS 通过标签的形式插入到 HTML 中，所以后面依赖前者。

```
npm i --save-dev css-loader style-loader
```

首先修改 addImage.js 文件

```

import '../styles/addImage.css'

let smallImg = document.createElement('img')
smallImg.src = require('../images/small.jpeg')
document.body.appendChild(smallImg)

// let bigImg = document.createElement('img')
// bigImg.src = require('../images/big.jpeg')
// document.body.appendChild(bigImg)

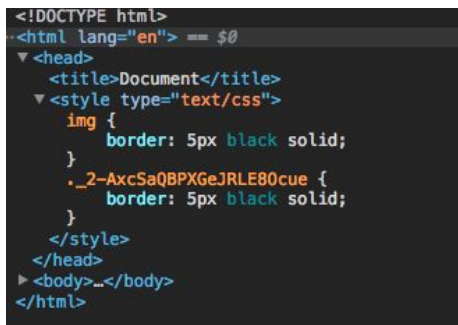
```

然后修改 webpack.config.js 代码



```
module.exports = {
  // ...
  module: {
    rules: [
      {
        test: /\.css$/,
        use: ['style-loader',
          {
            loader: 'css-loader',
            options: {
              modules: true
            }
          }
        ],
      },
    ],
  },
}
```

运行下 `npm run start`，然后刷新页面，可以发现图片被正确的加上了边框，现在我们来看一下 HTML 的文件结构



```
<!DOCTYPE html>
<html lang="en" == $0
  <head>
    <title>Document</title>
    <style type="text/css">
      img {
        border: 5px black solid;
      }
      ._2-AxcSaQBPXGeJRLE80cue {
        border: 5px black solid;
      }
    </style>
  </head>
  <body>...</body>
</html>
```

从上图可以看到，我们在 `addImage.css` 文件中写的代码被加入到了 `style` 标签中，并且因为我们开启了 CSS 模块化的选项，所以 `.test` 被转成了唯一的哈希值，这样就解决了 CSS 的变量名重复问题。

但是将 CSS 代码整合进 JS 文件也是有弊端的，大量的 CSS 代码会造成 JS 文件的大小变大，操作 DOM 也会造成性能上的问题，所以接下来我们将使用 `extract-text-webpack-plugin` 插件将 CSS 文件打包为一个单独文件

首先安装 `npm i --save-dev extract-text-webpack-plugin`

然后修改 webpack.config.js 代码

```
const ExtractTextPlugin = require("extract-text-webpack-plugin")

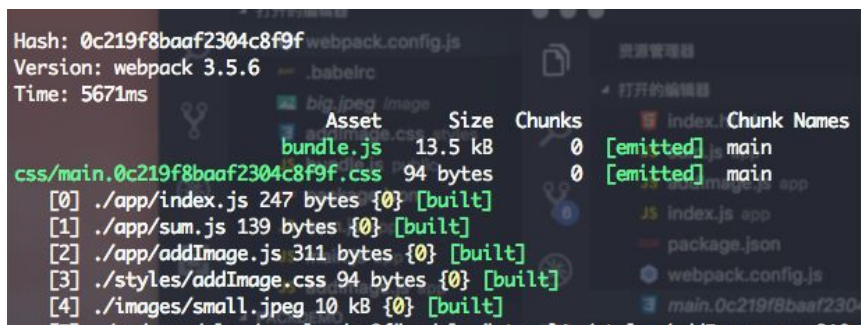
module.exports = {
  // ....
  module: {
    rules: [
      {
        test: /\.css$/,
        // 写法和之前基本一致
        loader: ExtractTextPlugin.extract({
          // 必须这样写，否则会报错
          fallback: 'style-loader',
        })
      },
    ],
  },
}
```

▲ 赞同 124 ▼ ● 13 条评论 ➤ 分享 ★ 收藏 ...



```
    options: {
      modules: true
    }
  }
}
})
}
}
]
},
// 插件列表
plugins: [
  // 输出的文件路径
  new ExtractTextPlugin("css/[name].[hash].css")
]
}
```

运行下 `npm run start`，可以发现 CSS 文件被单独打包出来了



但是这时候刷新页面会发现图片的边框消失了，那是因为我们的 HTML 文件没有引用新的 CSS 文件，所以这里需要我们手动引入下，在下面的章节我们会通过插件的方式自动引入新的文件。

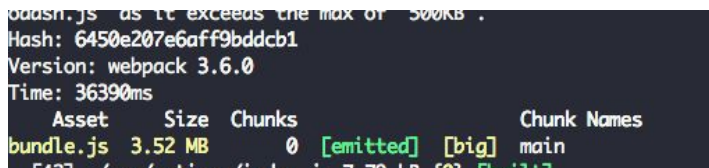
接下来，会用一个项目来继续我们的 webpack 学习，在这之前，先 clone 一下项目。该项目原地址是 [这里](#)，因为使用的 webpack 版本太低，并且依赖的库也有点问题，故我将项目拷贝了过来并修改了几个库的版本号。

请依次按照以下代码操作

```
git clone https://github.com/KieSun/webpack-demo.git
cd webpack-demo
// 切换到 0.1 标签上并创建一个新分支
git checkout -b demo 0.1
// 查看分支是否为 demo，没问题的话就可以进行下一步
gst
```

如何在项目中使用 webpack

项目中已经配置了很简单的 babel 和 webpack，直接运行 `npm run start` 即可



这时候你会发现这个 bundle.js 居然有这么大，这肯定是不能接受的，所以接下来章节的主要目的就是单个文件拆分为多个文件，优化项目。



分离代码

先让我们考虑下缓存机制。对于代码中依赖的库很少会去主动升级版本，但是我们自己的代码却每时每刻都在变更，所以我们可以考虑将依赖的库和自己的代码分割开来，这样用户在下一次使用应用时就可以尽量避免重复下载没有变更的代码，那么既然要将依赖代码提取出来，我们需要变更下入口和出口的部分代码。

```
// 这是 package.json 中 dependencies 下的
const VENDORS = [
  "lodash",
  "react",
  "react-dom",
  "react-input-range",
  "react-redux",
  "redux",
  "redux-form",
  "redux-thunk"
]

module.exports = {
  // 之前我们都是使用了单文件入口
  // entry 同时也支持多文件入口，现在我们有两个入口
  // 一个是我们自己的代码，一个是依赖库的代码
  entry: {
    // bundle 和 vendor 都是自己随便取名的，会映射到 [name] 中
    bundle: './src/index.js',
    vendor: VENDORS
  },
  output: {
    path: path.join(__dirname, 'dist'),
    filename: '[name].js'
  },
  // ...
}
```

现在我们 build 一下，看看是否有惊喜出现

```
Version: webpack 3.6.0
Time: 27253ms
   Asset      Size  Chunks             Chunk Names
bundle.js  3.52 MB          0  [emitted] [big] bundle
vendor.js   3.08 MB          1  [emitted] [big] vendor
   [37] (webpack)/buildin/module.js 521 bytes {0} {1} [built]
   [52] (webpack)/buildin/global.js 823 bytes {0} {1} [built]
  [1543] ./src/index.js 1.14 kB {0} [built]
  [1544] ./src/reducers/index.js 988 bytes {0} [built]
```



真的有惊喜。。为什么 bundle 文件大小压根没变。这是因为 bundle 中也引入了依赖库的代码，刚才的步骤并没有抽取

赞同 124

13 条评论

分享

收藏

...



抽取共同代码

在这小节我们使用 webpack 自带的插件 CommonsChunkPlugin。

```
module.exports = {
  //...
  output: {
    path: path.join(__dirname, 'dist'),
    // 既然我们希望缓存生效，就应该每次在更改代码以后修改文件名
    // [chunkhash]会自动根据文件是否更改而更换哈希
    filename: '[name].[chunkhash].js'
  },
  plugins: [
    new webpack.optimize.CommonsChunkPlugin({
      // vendor 的意义和之前相同
      // manifest文件是将每次打包都会更改的东西单独提取出来，保证没有更改的代码无需重新打包，这
      names: ['vendor', 'manifest'],
      // 配合 manifest 文件使用
      minChunks: Infinity
    })
  ]
};
```

当我们重新 build 以后，会发现 bundle 文件很明显的减小了体积

```
Hash: 4ef74c7d579acbc1453
Version: webpack 3.6.0
Time: 33417ms

   Asset      Size  Chunks             Chunk Names
vendor.278c46d3f4d3a3379c95.js  3.08 MB      0  [emitted] [big] vendor
bundle.cfde6df7b4d6dd1a90f0.js   442 kB      1  [emitted] [big] bundle
manifest.4ff456f60e59e7a0c6c9.js  5.85 kB      2  [emitted]          manifest
```

但是我们使用哈希来保证缓存的同时会发现每次 build 都会生成不一样的文件，这时候我们引入另一个插件来帮助我们删除不需要的文件。

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

然后修改配置文件

```
module.exports = {
  //...
  plugins: [
    // 只删除 dist 文件夹下的 bundle 和 manifest 文件
    new CleanWebpackPlugin(['dist/bundle.*.js', 'dist/manifest.*.js'], {
      // 打印 log
      verbose: true,
      // 删除文件
      dry: false
    })
  ]
};
```

然后 build 的时候会发现以上文件被删除了。

因为我们现在将文件已经打包成三个 JS 了，以后也许会更多，每次新增 JS 文件我们都需要手动在 HTML 中新增标签，现在我们可以通过一个插件来自动完成这个功能。

```
npm install html
```

▲ 赞同 124 ▼

● 13 条评论

➦ 分享

★ 收藏

...

然后修改配置文件



```
module.exports = {
  //...
  plugins: [
    // 我们这里将之前的 HTML 文件当做模板
    // 注意在之前 HTML 文件中请务必删除之前引入的 JS 文件
    new HtmlWebpackPlugin({
      template: 'index.html'
    })
  ]
};
```

执行 build 操作会发现同时生成了 HTML 文件，并且已经自动引入了 JS 文件

```
Version: webpack 3.6.0
Time: 31498ms

Asset      Size  Chunks
vendor.278c46d3f4d3a3379c95.js  3.08 MB    0  [emitted] [big]
bundle.cfde6df7b4d6dd1a90f0.js    442 kB    1  [emitted] [big]
manifest.4ff456f60e59e7a0c6c9.js  5.85 kB    2  [emitted]
index.html  387 bytes  [emitted]
```

按需加载代码

在这一小节我们将学习如何按需加载代码，在这之前的 vendor 入口我发现忘记加入 router 这个库了，大家可以加入这个库并且重新 build 下，会发现 bundle 只有不到 300KB 了。

现在我们的 bundle 文件包含了我们全部的自己代码。但是当用户访问我们的首页时，其实我们根本无需让用户加载除了首页以外的代码，这个优化我们可以通过路由的异步加载来完成。

现在修改 src/router.js

```
// 注意在最新版的 V4路由版本中，更改了按需加载的方式，如果安装了 V4版，可以自行前往官网学习
import React from 'react';
import { Router, Route, IndexRoute, hashHistory } from 'react-router';

import Home from './components/Home';
import ArtistMain from './components/artists/ArtistMain';

const rootRoute = {
  component: Home,
  path: '/',
  indexRoute: { component: ArtistMain },
  childRoutes: [
    {
      path: 'artists/new',
      getComponent(location, cb) {
        System.import('./components/artists/ArtistCreate')
          .then(module => cb(null, module.default))
      }
    },
    {
      path: 'artists/:id/edit',
      getComponent(location, cb) {
        System.import('./components/artists/ArtistEdit')
          .then(module => cb(null, module.default))
      }
    }
  ],
  {
    path: 'art
```

赞同 124

13 条评论

分享

收藏

...



```
getComponent(location, cb) {
  System.import('./components/artists/ArtistDetail')
    .then(module => cb(null, module.default))
}
}
]
}

const Routes = () => {
  return (
    <Router history={hashHistory} routes={rootRoute} />
  );
};

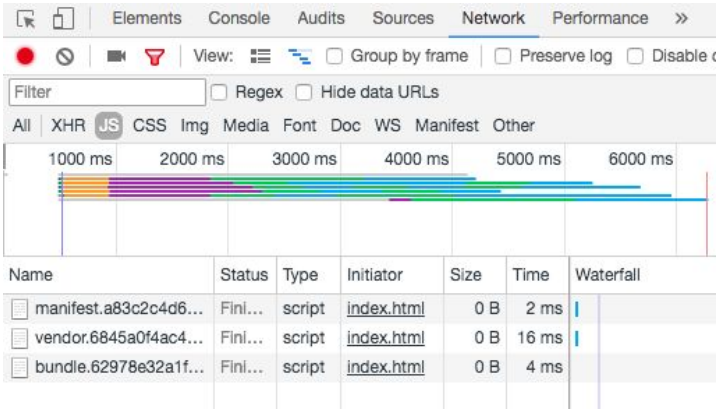
export default Routes;
```

然后执行 build 命令，可以发现我们的 bundle 文件又瘦身了，并且新增了几个文件

Asset	Size	Chunks	Chunk Name
0.d64feb4d6d919c11fff.js	5.87 kB	0	[emitted]
1.df1278b76ec2dc44b276.js	9.11 kB	1	[emitted]
2.28e005e73ca86a2cc607.js	4.54 kB	2	[emitted]
vendor.6845a0f4ac43e135b384.js	3.26 MB	3	[emitted] [big] vendor
bundle.62978e32a1f5f30cafde.js	241 kB	4	[emitted] bundle
manifest.a83c2c4d665c3ce642be.js	5.93 kB	5	[emitted] manifest
index.html	387 bytes		[emitted]
[48] (webpack)/buildin/module.js	521 bytes	{3}	[built]

将 HTML 文件在浏览器中打开，当点击路由跳转时，可以在开发者工具中的 Network 一栏中看到加载了一个 JS 文件。

首页



点击右上角 Random Artist 以后





自动刷新

每次更新代码都需要执行依次 build，并且还要等上一会很麻烦，这一小节介绍如何使用自动刷新的功能。

首先安装插件

```
npm i --save-dev webpack-dev-server
```

然后修改 packet.json 文件

```
"scripts": {  
  "build": "webpack",  
  "dev": "webpack-dev-server --open"  
},
```

现在直接执行 npm run dev 可以发现浏览器自动打开了一个空的页面，并且在命令行中也多了新的输出

```
> webpack-dev-server --open  
  
clean-webpack-plugin: /Users/yuchengkai/webpack-demo/project/dist/*.js has been removed.  
Project is running at http://localhost:8081/  
webpack output is served from /  
webpack: wait until bundle finished: /
```

等待编译完成以后，修改 JS 或者 CSS 文件，可以发现 webpack 自动帮我们完成了编译，并且只更新了需要更新的代码

```
Asset      Size  Chunks  Chunk Names  
bundle.7bd3728f45158240366f.js  241 kB    4  [emitted]  bundle  
manifest.28eebf0d8017c73813d8.js  5.93 kB    5  [emitted]  manifest  
index.html  387 bytes    [emitted]  
+ 4 hidden assets  
[287] (webpack)/hot nonrecursive ^\.\/log$ 170 bytes {3} [built]  
[291] ./src/index.js 1.14 kB {4} [built]  
+ 1687 hidden modules  
Child html-webpack-plugin for "index.html":  
  1 asset  
  5 modules  
webpack: Compiled successfully.
```

但是每次重新刷新页面对于 debug 来说很不友好，这时候就需要用到模块热替换了。但是因为项目中使用了 React，并且 Vue 或者其他框架都有自己的一套 hot-loader，所以这里就略过了，有兴趣的可以自己学习下。

生成生产环境代码

现在我们可以将之前所学和一些新加的插件整合在一起，build 生产环境代码。

```
npm i --save-dev url-loader optimize-css-assets-webpack-plugin file-loader extract-text-webpack-plugin
```

修改 webpack 配置

```
var webpack = require('webpack')  
var path = require('path')
```

▲ 赞同 124 ▼

● 13 条评论

➤ 分享

★ 收藏

...



```

var HtmlWebpackPlugin = require('html-webpack-plugin')
var CleanWebpackPlugin = require('clean-webpack-plugin')
var ExtractTextPlugin = require('extract-text-webpack-plugin')
var OptimizeCSSPlugin = require('optimize-css-assets-webpack-plugin')

const VENOR = ["faker",
  "lodash",
  "react",
  "react-dom",
  "react-input-range",
  "react-redux",
  "redux",
  "redux-form",
  "redux-thunk",
  "react-router"
]

module.exports = {
  entry: {
    bundle: './src/index.js',
    vendor: VENOR
  },
  // 如果想修改 webpack-dev-server 配置, 在这个对象里面修改
  devServer: {
    port: 8081
  },
  output: {
    path: path.join(__dirname, 'dist'),
    filename: '[name].[chunkhash].js'
  },
  module: {
    rules: [{
      test: /\.js$/,
      use: 'babel-loader'
    },
    {
      test: /\.?(png|jpe?g|gif|svg)(\?.*)?$/,
      use: [{
        loader: 'url-loader',
        options: {
          limit: 10000,
          name: 'images/[name].[hash:7].[ext]'
        }
      }]
    },
    {
      test: /\.css$/,
      loader: ExtractTextPlugin.extract({
        fallback: 'style-loader',
        use: [{
          // 这边其实还可以使用 postcss 先处理下 CSS 代码
          loader: 'css-loader'
        }]
      })
    }
  ],
  plugins: [
    new webpack.optimize.CommonsChunkPlugin({
      name: ['vendor', 'manifest'],
      minChunks: Infinity
    }),
    new CleanWebpackPlugin(['dist/*.js'], {
      verbose: true,
      dry: false
    })
  ]
}

```



```

    }),
    new HtmlWebpackPlugin({
      template: 'index.html'
    }),
    // 生成全局变量
    new webpack.DefinePlugin({
      "process.env.NODE_ENV": JSON.stringify("process.env.NODE_ENV")
    }),
    // 分离 CSS 代码
    new ExtractTextPlugin("css/[name].[contenthash].css"),
    // 压缩提取出的 CSS，并解决ExtractTextPlugin分离出的 JS 重复问题
    new OptimizeCSSPlugin({
      cssProcessorOptions: {
        safe: true
      }
    }),
    // 压缩 JS 代码
    new webpack.optimize.UglifyJsPlugin({
      compress: {
        warnings: false
      }
    })
  ]
};

```

修改 packet.json 文件

```

"scripts": {
  "build": "NODE_ENV=production webpack -p",
  "dev": "webpack-dev-server --open"
}

```

执行 npm run build

Time: 69446ms

Asset	Size	Chunks	Chunk Names
0.bebf658f3631fc43d17f.js	3.07 kB	0 [emitted]	
1.2e5d37a9dddf87c37.js	4.11 kB	1 [emitted]	
2.e48d3c391d2dba71f0a9.js	2.42 kB	2 [emitted]	
vendor.4a4c9fbbba4e9c6c427b.js	1.57 MB	3 [emitted] [big]	vendor
bundle.408dec6137f468b9d3d9.js	27.1 kB	4 [emitted]	bundle
manifest.4cb5241a04440e0cdebc.js	1.49 kB	5 [emitted]	manifest
css/bundle.f5df50a19a2e8d5db32778930c2398d6.css	115 kB	4 [emitted]	bundle
index.html	465 bytes	[emitted]	

[47] (webpack)/building/module.js 521 bytes [3] [built]

可以看到我们在经历了这么多步以后，将 bundle 缩小到了只有 27.1KB，像 vendor 这种常用的库我们一般可以使用 CDN 的方式外链进来。

补充

webpack 配置上有些实用的小点在上文没有提到，统一在这里提一下。

```

module.exports = {
  resolve: {
    // 文件扩展名，写明以后就不需要每个文件写后缀
    extensions: ['.js', '.css', '.json'],
    // 路径别名，比如这里可以使用 css 指向 static/css 路径
    alias: {
      '@': resolve('src'),
      'css': resolve('static/css')
    }
  },
  // 生成 source-

```

▲ 赞同 124 ▼

● 13 条评论

➦ 分享

★ 收藏

...


```
devtool: '#cheap-module-eval-source-map',
}
```



后记

如果你是跟着本文一个个步骤敲下来的，那么大部分的 webpack 配置你应该都是可以看懂了，并且自己应该也知道如何去配置。谢谢大家看到这里，这是本项目对应的仓库，每个小节基本都对应了一次 commit。

文章较长，有错误也难免，如果你发现了任何问题或者我有任何表述的不明白的地方，都可以留言给我。

—————end—————

加饥人谷官方微信号: hungervalley ，暗号：来自知乎

每日一题，每周资源推荐，精彩博客推荐，工作、笔试、面试经验交流解答，免费直播课，群友轻分享... 数不尽的福利免费送

编辑于 2017-12-15

前端开发 webpack 前端工程师

文章被以下专栏收录



饥人谷前端学习指南
前端小课：xiedaimala.com，前端培训：jirengu.com

已关注

推荐阅读



章子怡不愿签的婚前协议，为什么奶茶妹妹却签了？

小麦心理X



扒一扒刘强东和前女友：第一桶金有多干净？

笨虎



彻底解决Webpack打包性能问题

Stark... 发表于Stark...



Webpack Splitting

ly你个c

13 条评论

⇌ 切换为时间排序

写下你的评论...



姜先生

mark

👍 赞

1 年前

▲ 赞同 124 ▼ 13 条评论 ➦ 分享 ★ 收藏 ...



- luckybai

1 年前
- m
- 赞

总督察

1 年前

瞬间不想继续往下看了

1

龙之母风暴降生

1 年前

m

赞

幽忧偶心

1 年前

感谢~

赞

出去骑车出去吃

1 年前

webpack是个好东西。但是让我再配置一遍，我选择死亡。

5

40哥哥 回复 出去骑车出去吃

1 年前

还有各大框架，都有cli

赞

二胖

1 年前

学习一下

赞

王朝

1 年前

写的很好，赞👍

赞

李老酸

1 年前

6666学习了

赞

叶茂

1 年前

mark，写的很详细。

赞

那天你远去

1 年前

npm start不需要run

赞

glory

9 个月前

66666，写的真不错

赞