

React+Webpack快速上手指南



作者 Chester723 (/u/3549de9e5a55) [+ 关注](#)

2015.09.15 16:33 字数 2099 阅读 49444 评论 34 喜欢 46 赞赏 1

(/u/3549de9e5a55)

前言

这篇文章不是有关React和Webpack的教程，只是一篇能够让你快速上手使用目前这两种热门技术的前端指南，并假设你对两者有一个基本的认识。如果你想先行了解下React，可以放肆的移步至 React官方教程 (<http://reactjs.cn/react/docs/getting-started.html>)，如果你已经使用了其他的模块加载与资源打包技术，不妨也来看看Webpack (<https://webpack.github.io/>) 提供的思路。

webstorm+react+webpack

强烈推荐使用webstorm！（当然你完全可以使用诸如atom，Sublime之类的编辑器，但之所以选择webstorm是因为它默认支持对react JSX (<http://reactjs.cn/react/docs/displaying-data.html#jsx->) 的语法高亮以及可以手动开启Emmet对jsx的支持，棒棒哒~）

首先请这么组织你的项目结构：

```
--your project
|--app
|  |--components
|  |  |--productBox.jsx
|  |--main.js
|--build
|  |--index.html
|  |--bundle.js (该文件是webpack打包后生成的)
```

用npm安装react、webpack

默认已经安装了nodejs，并输入：`npm init` 根据提示输入内容并创建package.json文件然后依次输入：

```
npm install react --save-dev
```

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

安装最新版本的React与Webpack并将它们保存至package.json内的开发依赖项目中。

创建webpack.config.js配置文件

新建一个名为 `webpack.config.js` 的文件，它应该长这个样子：



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

module.exports = {
  entry: path.resolve(__dirname, './app/main.js'),
  output: {
    path: path.resolve(__dirname, './build'),
    filename: 'bundle.js',
  }
};
```

其中entry指定了webpack的入口程序，好比c++和java中的main程序一样，我们把最终要插入到页面指定位置的react模板写入main.js中：

app/main.js

```
var React = require('react');
var AppComponent = require('./components/productBox.js');
React.render(<AppComponent />, document.getElementById('content'));
```

以及引入的自定义react组件：

app/components/productBox.js

```
var React = require('react');
var ProductBox = React.createClass({
  render: function () {
    return (
      <div className="productBox">
        Hello World!
      </div>
    );
  }
});

module.exports = ProductBox;
```

而output则指定了webpack打包成功之后文件名称以及文件的存放位置。依照之前指定的项目结构，我们可以在 index.html 中直接引入打包生成的 bundle.js ，像这样：

build/index.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>React Test</title>
</head>
<body>
  <div id="content"></div>
  <script src="bundle.js"></script>
</body>
</html>
```

安装并启用webpack-dev-server

webpack-dev-server允许我们可以把本地项目跑在像nginx那样的web服务器上，**更重要的是**我们可以在package.json文件内定义scripts，同时修改webpack的配置文件来达到类似BrowserSync（即文件修改能被监听，并自动刷新浏览器）的效果！

我们先打开package.json，并找到 scripts 代码块。在没引入webpack-dev-server之前，我们运行这个项目的姿势是这样的：

修改scripts为

```
"scripts": {
  "build": "webpack"
}
```

并且执行：



```
npm run build
```

项目就跑起来啦，但是每次修改程序我们都要手动输入 `npm run build` 来跑项目，这无疑是一件非常蛋疼的事情。但有了webpack-dev-server光环，我们的姿势应该是这样的：为scripts添加

```
"scripts": {  
  "build": "webpack",  
  "dev": "webpack-dev-server --devtool eval --progress --colors --content-base build"  
}
```

ps：dev里各属性值的意思是：

1. webpack-dev-server：在 localhost:8080 建立一个 Web 服务器
2. --devtool eval：为你的代码创建源地址。当有任何报错的时候可以让你更加精确地定位到文件和行号
3. --progress：显示合并代码进度
4. --colors：在命令行中显示颜色
5. --content-base build：指向设置的输出目录

并且在index.html里加入：

build/index.html

```
<!DOCTYPE html>  
<html>  
  <head lang="en">  
    <meta charset="UTF-8">  
    <title>React Test</title>  
  </head>  
  <body>  
    <div id="content"></div>  
    <script src="http://localhost:8080/webpack-dev-server.js"></script>  
    <script src="bundle.js"></script>  
  </body>  
</html>
```

在webpack.config.js的入口处加入：

```
var path = require('path');  
  
module.exports = {  
  entry: ['webpack/hot/dev-server', path.resolve(__dirname, './app/main.js')],  
  output: {  
    path: path.resolve(__dirname, './build'),  
    filename: 'bundle.js',  
  },  
};
```

以便在进行改动时浏览器可以自动刷新。

最后执行：

```
npm run dev
```

你只要在浏览器打开这个地址：`http://localhost:8080`，webpack-dev-server会为你准备好一切，你只要敲一敲键盘，save一下，所见即所得。

Webpack Loader



讲到这里，我们基本上就可以迅速搭建一个简单的web项目，但不得不提的是webpack loader。它是我个人认为相比于其他模块加载更牛X的地方，将它用于react的开发，结合react与生俱来的优越性能，两者天衣无缝的配合简直就是**黄金组合**。

总的来说webpack loader可以实现：

- 可以将React JSX语法转为js语句
- React开发中支持ES6语法
- 支持通过import来直接引入css、less、sass甚至是图片
- 支持css中引用的图片大小在某一大小范围之内直接转为BASE64格式
。。。。等等等

为了能够让以上功能奏效，我们要先安装对应的：

babel-loader

```
npm install babel-loader --save-dev
```

css-loader

```
npm install css-loader --save-dev
```

less-loader

```
npm install less-loader --save-dev
```

style-loader

```
npm install style-loader --save-dev
```

url-loader

```
npm install url-loader --save-dev
```

而具体的实现，我们只要在webpack的配置文件中加入module属性里的loaders：

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

module.exports = {
  entry: ['webpack/hot/dev-server', path.resolve(__dirname, './app/main.js')],
  output: {
    path: path.resolve(__dirname, './build'),
    filename: 'bundle.js',
  },
  module: {
    loaders: [{
      test: /\.jsx?$/,
      loader: 'babel'
    }, {
      test: /\.css$/,
      loader: 'style!css'
    }, {
      test: /\.less$/,
      loader: 'style!css!less'
    }, {
      test: /\.?(png|jpg)$/,
      loader: 'url?limit=25000'
    }
  ]
}
```



通过指定文件后缀来执行对应的loader操作，我们甚至可以轻松的引入其他类型的文件，最后所有被引入进来的文件都会统统打包到bundle.js中去！

开发环境下的优化

在开发React组件的代码中，在webpack的帮助下我们都是通过require('react')的方式引入ReactJS，默认的require()方法会在webpack打包的时候去遍历ReactJS及其依赖。试想下，如果我们还引入了其他的库，例如jQuery、react-router、moment等，require都会以同样的方式去遍历，这会大大增加打包的时间。这显然会影响到开发的效率。

为了能够尽可能的加速打包过程，我们应该重写它的行为，鼓励在开发中使用依赖的压缩版本

假如我们只引入了react，我们可以这样修改webpack.config.js:

```
var path = require('path');
var node_modules = path.resolve(__dirname, 'node_modules');
var pathToReact = path.resolve(node_modules, 'react/dist/react.min.js');

var config = {
  entry: ['webpack/hot/dev-server', path.resolve(__dirname, './app/main.js')],
  resolve: {
    alias: {
      'react': pathToReact
    }
  },
  output: {
    path: path.resolve(__dirname, './build'),
    filename: 'bundle.js'
  },
  module: {
    noParse: [pathToReact],
    loaders: [{
      test: /\.jsx?$/,
      loader: 'babel'
    }, {
      test: /\.css$/,
      loader: 'style!css'
    }, {
      test: /\.less$/,
      loader: 'style!css!less'
    }, {
      test: /\.?(png|jpg)$/,
      loader: 'url?limit=25000'
    }]
  }
};

module.exports = config;
```

resolve 属性里的 alias 用来告诉webpack，当引入react时，试图去匹配压缩过的react；而 module 中的 noParse 则是告诉当webpack尝试去解析压缩文件时，这种行为是不允许的。

当然啦，细心的你可能会发现当引入更多的库时，这样写多少会显得有点冗余，为此下面有个更加优雅的写法：



```
var path = require('path');
var node_modules_dir = path.join(__dirname, 'node_modules');
var deps = [
  'react/dist/react.min.js',
  'react-router/dist/react-router.min.js',
  'moment/min/moment.min.js'
];
var config = {
  entry: ['webpack/hot/dev-server', path.resolve(__dirname, './app/main.js')],
  resolve: {
    alias: {}
  },
  output: {
    path: path.resolve(__dirname, './build'),
    filename: 'bundle.js'
  },
  module: {
    noParse: [],
    loaders: [{
      test: /\.jsx?$/,
      loader: 'babel'
    }, {
      test: /\.css$/,
      loader: 'style!css'
    }, {
      test: /\.less$/,
      loader: 'style!css!less'
    }, {
      test: /\.?(png|jpg)$/,
      loader: 'url?limit=25000'
    }]
  }
};

deps.forEach(function (dep) {
  var depPath = path.resolve(node_modules_dir, dep);
  config.resolve.alias[dep.split(path.sep)[0]] = depPath;
  config.module.noParse.push(depPath);
});

module.exports = config;
```

瞬间biger起来了，待加以修饰之后，让我们更加便捷高效的享受React与Webpack带来的乐趣吧。

相关的代码请狠狠点击[这里查看](https://coding.net/u/chester723/p/react-webpack-simple-example/git) (https://coding.net/u/chester723/p/react-webpack-simple-example/git) ,




随笔 (/nb/1301570)

举报文章 © 著作权归作者所有

 **Chester723** (/u/3549de9e5a55)
写了 15189 字, 被 101 人关注, 获得了 105 个喜欢
(/u/3549de9e5a55)

+ 关注

喜欢 (/sign_in) | 46

   更多分享





(http://cwb.assets.jianshu.io/notes/images/2113564

被以下专题收入，发现更多相似内容

-  技术干货 (/c/38d96caffb2f?utm_source=desktop&utm_medium=notes-included-collection)
-  笔戈 Web ... (/c/2a433580d4c?utm_source=desktop&utm_medium=notes-included-collection)
-  React.js (/c/4dcf98759530?utm_source=desktop&utm_medium=notes-



- included-collection)

 iOS自学之路 (/c/4de557e5bd96?utm_source=desktop&utm_medium=notes-included-collection)
-  前端学习 (/c/92a2e5ab3d86?utm_source=desktop&utm_medium=notes-included-collection)
-  node (/c/a70687de1afb?utm_source=desktop&utm_medium=notes-included-collection)
-  React (/c/79180afbe622?utm_source=desktop&utm_medium=notes-included-collection)
- 展开更多

▼

