

[Webpack](#) 是一个前端资源加载/打包工具，只需要相对简单的配置就可以提供前端工程化需要的各种功能，并且如果有需要它还可以被整合到其他比如 Grunt / Gulp 的工作流

### 一键傻瓜式安装

`create-react-app my-app --scripts-version custom-react-scripts`

1. 新建目录react-webpack
2. 运行`cnpm init -y`，生成package.json
3. 运行`cnpm i webpack webpack-cli -g`

`cnpm i webpack webpack-cli -D` (两个都要安装) 开发依赖

4. 新建src文件夹，src下新建index.js（入口文件），运行webpack, 会生成dist目录，dist下有main.js(出口文件)（现在是production模式）
5. 转换成开发模式，运行`webpack --mode development`  
再转换成生产模式，运行`webpack --mode production`
6. 打开package.json, 在scripts内添加

```
"scripts": {
  "test": "echo \\\"Error: no test specified\\\" && exit 1",
  "dev": "webpack --mode development",
  "build": "webpack --mode production"
},
```

将模式参数加入到webpack中

7. 运行`npm run dev`，然后查看./dist/main.js，文件是**没有**压缩的代码文件；
8. 运行`npm run build`，然后查看./dist/main.js，文件是**已经**压缩的代码文件；
9. 如果要修改webpack默认的入口文件和出口文件，同样的打开package.json, 在scripts内修改

```
"scripts": {
  "test": "echo \\\"Error: no test specified\\\" && exit 1",
  "dev": "webpack --mode development ./foo/src/index --output ./foo/main.js",
  "build": "webpack --mode production ./foo/src/index --output ./foo/main.js"
},
```

10. webpack，**利用Babel**将ES6语法转换成ES5

`cnpm i babel-core babel-loader babel-preset-env -D`

11. 新建.babelrc文件

```
index.js  .babelrc x
.babelrc > ...
1  {
2    "presets": ["env"]
3  }
```

12. 新建webpack.config.js文件

```

1  module.exports = {
2    module: {
3      rules: [
4        {
5          test: /\.js$/,
6          exclude: /node_modules/,
7          use: {
8            loader: "babel-loader"
9          }
10       }
11     ]
12   }
13 }

```

13. 打开package.json, 在scripts内还原入口文件和出口文件

```

"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "dev": "webpack --mode development",
  "build": "webpack --mode production"
},

```

运行rm -rf foo移除foo文件

14. 打开src下的index.js

```

1  const arr = [1,2,3];
2  const iAmJavascriptEs6 = () => console.log(...arr); //arrow function 箭头函数 spread 展开式参数传递
3  window.iAmJavascriptEs6 = iAmJavascriptEs6;

```

运行npm run bulid 报错 (版本冲突)

15. 查看所有版本信息

```

bash-3.2# npm info babel-loader

```

安装指定版本

```

bash-3.2# cnpm i babel-loader@7.1.5 -D

```

重新运行npm run bulid, 就转换成ES5了, 也可以运行npm run dev更方便查看代码

以下是react 的转换

1. 安装react开发模块

```

# cnpm i react react-dom -S

```

2. 安装react代码语法解析

```

cnpm i babel-preset-react -D

```

3. 打开.babelrc配置文件, 添加react

```

1  {
2    "presets": ["env", "react"]
3  }

```

4. 打开webpack.config



```

babelrc  webpack.config.js x
webpack.config.js > <unknown>
1  module.exports = {
2    module: {
3      rules: [
4        {
5          test: /\.jsx$/,
6          exclude: /node_modules/,
7          use: {
8            loader: "babel-loader"
9          }
10         }
11       ]
12     }
13   }

```

5. 在src下新建index.js（入口文件）App.js（主组件）

在入口文件中分别引入react、react-dom

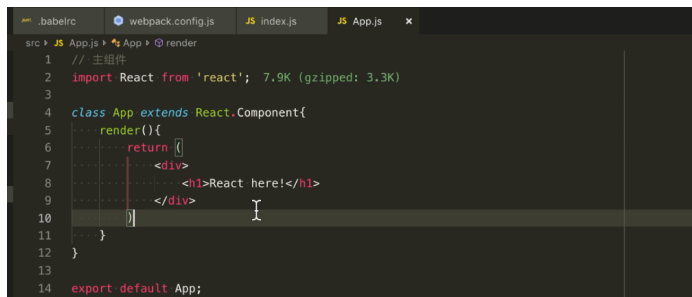


```

babelrc  webpack.config.js  JS index.js x  JS App.js x
src > JS index.js
1  // 入口文件
2  import React from 'react'; 7.9K (gzipped: 3.3K)
3  import ReactDOM from 'react-dom'
4

```

在主组件内引入react, 创建组件(class), 接口暴露(export default)

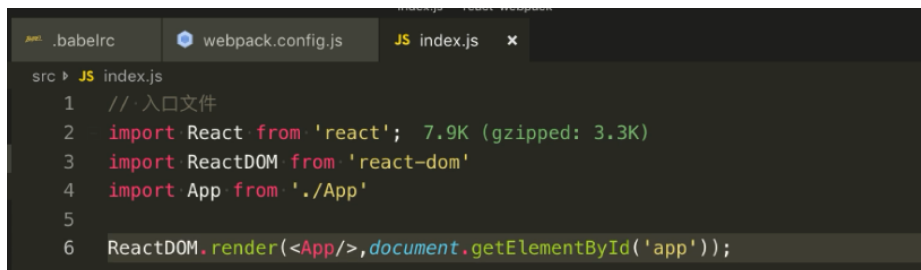


```

babelrc  webpack.config.js  JS index.js  JS App.js x
src > JS App.js > App > render
1  // 主组件
2  import React from 'react'; 7.9K (gzipped: 3.3K)
3
4  class App extends React.Component {
5    render() {
6      return (
7        <div>
8          <h1>React here!</h1>
9        </div>
10      );
11    }
12  }
13
14  export default App;

```

回到index.js 文件, 引入App, 渲染App



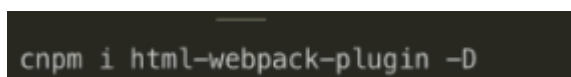
```

babelrc  webpack.config.js  JS index.js x
src > JS index.js
1  // 入口文件
2  import React from 'react'; 7.9K (gzipped: 3.3K)
3  import ReactDOM from 'react-dom'
4  import App from './App'
5
6  ReactDOM.render(<App />, document.getElementById('app'));

```

6. 运行npm run dev命令, 会发现main.js 文件非常大, 可以再运行npm run bulid进行压缩丑化（处理js文件）

7. 安装html插件（github）



```

cnpm i html-webpack-plugin -D

```

在webpack.config内引入插件



```

babelrc  webpack.config.js x
webpack.config.js
1  const HtmlWebpackPlugin = require('html-webpack-plugin') Calculating...
2

```

调用插件

```

babelrc  webpack.config.js x
webpack.config.js > |< unknown>
8  ...exclude: /node_modules/,
9  ...use: {
10 ...loader: "babel-loader"
11 ...}
12 ...}
13 ...}
14 ...},
15 ...plugins: [
16 ...new HtmlWebpackPlugin({
17 ...template: './src/index.html',
18 ...filename: './index.html'
19 ...})
20 ...]
21 }

```

然后在src下新建index.html

```

babelrc  webpack.config.js  index.html x
src > index.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  ...<meta charset="UTF-8">
5  ...<meta name="viewport" content="width=device-width, initial-scale=1.0">
6  ...<meta http-equiv="X-UA-Compatible" content="ie=edge">
7  ...<title>React</title>
8  </head>
9  <body>
10 ...<div id="app"></div>
11 </body>
12 </html>

```

这时候运行npm run dev就会在dist下面自动创建一个index.html，点击index.html就可以直接运行了

## 8. 安装html-loader (github)

```

bash-3.2# cnpm i html-loader -D

```

在webpack.config添加一条rules

```

{
  test: /\.html$/,
  use: [{
    loader: 'html-loader',
    options: {
      minimize: true
    }
  }]
}

```

运行npm run dev, 打开dist下的index.html文件，会看到html也被压缩了 (html文件的自动生成以及代码压缩)

## 9. 安装css-loader (github)

```

bash-3.2# cnpm i css-loader -D

```

在src下建立一个main.css文件，设置一个样式，在index.js内引入css

在webpack.config添加一条css的rules

## 10. 安装css插件

```

# cnpm i mini-css-extract-plugin -D

```

在webpack.config内引入插件，在下面的plugins内调用插件，在css的rules里面添加上css插件的加载器

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

运行npm run dev时dist目录下会自动生成一个main.css文件，同时会在index.html内自动链接css, 再打开index.html时css样式就会生效了(css的解析以及css文件的抽离)

备注：如果是sass, 需要再安装，修改webpack.config内的css的rules (sass/less的支持)

```
问题 输出 调试控制台 终端
bash-3.2# cnpm i node-sass sass-loader -D
```

```
{
  test: /\.scss$/,
  use: [MiniCssExtractPlugin.loader, 'css-loader', 'sass-loader']
}
```

11. 安装开发服务 (本地服务的支持)

```
# cnpm i webpack-dev-server -D
```

12. 打开package.json, 修改dev

```
"dev": "webpack-dev-server --mode development --open",
```

运行npm run dev 就会自动打开localhost (--open实现自动打开)

13. react新语法的支持

```
cnpm i babel-present-stage-0 -D
```

修改.babelrc加上stage-0

14.

```
1 // 主组件
2 import React from 'react';
3 // 函数式组件（无状态组件），只有属性，没有状态，没有钩子函数
4 const Child=(props)=>{
5   ...console.log(props);
6   ...return(
7     ...<div>
8     ...<h1>React Child component</h1>
9     ...{props.name}
10    ...</div>
11    ...)
12  }
13 // 普通的函数式组件，有属性，状态，钩子函数
14 class App extends React.Component{
15   // 设置默认属性
16   ...static defaultProps={
17     ...url: 'http://localhost:3000/posts'
18     ...}
19   // 设置状态，第二种方法一定要加super
20   .../*state={
21     ...result: 'This is a state'
22     ...}*/
23   ...constructor(props){
24     ...super(props);
25     ...console.log(props);
26     ...this.state={
27       ...result: "This is constructor state"
28     ...}
29     ...}
30   // 事件绑定用箭头函数
31   ...handlerClick=()=>{
32     ...console.log(this);
33     ...}
34   ...render(){
35     ...return(
36     ...<div>
37     ...<h1>React here!</h1>
38     ...<p>{this.props.url}</p>
39     ...<span>{this.state.result}</span>
40     ...<button onClick={this.handlerClick}>click evrnt</button>
41     ...<Child name="vane" />
42     ...</div>
43     ...)
44     ...}
45   }
46
47   export default App;
```