2017.12.04-20180112

模块化构建工具Webpack

学信网 - 设计部 黄卉
huangh@chsi.com.cn

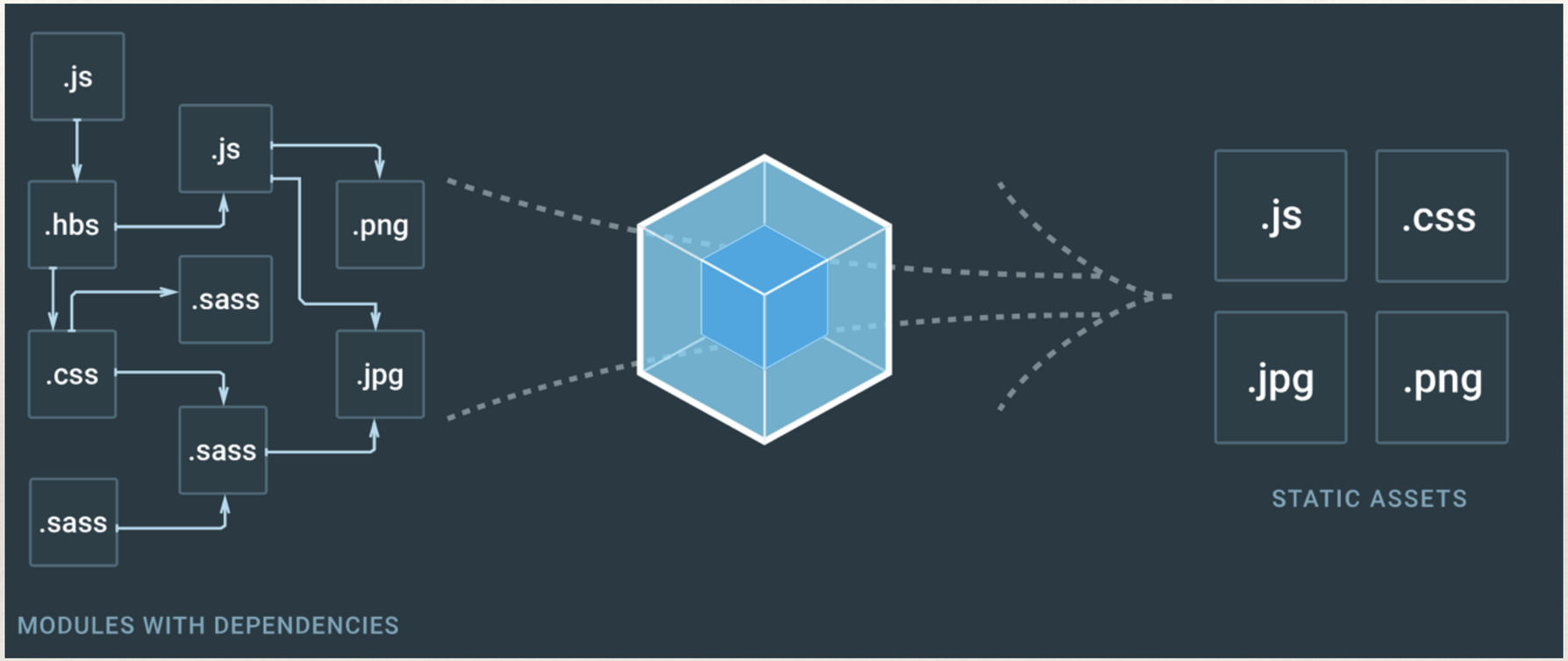
模块化构建工具Webpack

- 一、webpack简介
- 二、webpack基本用法
- 三、webpack参数配置
- 四、webpack实际应用

一、webpack简介

- 1.1 webpack是什么
- 1.2 webpack与gulp的关系
- 1.3 webpack的应用场景
- 1.4 webpack的特性

1.1 webpack是什么

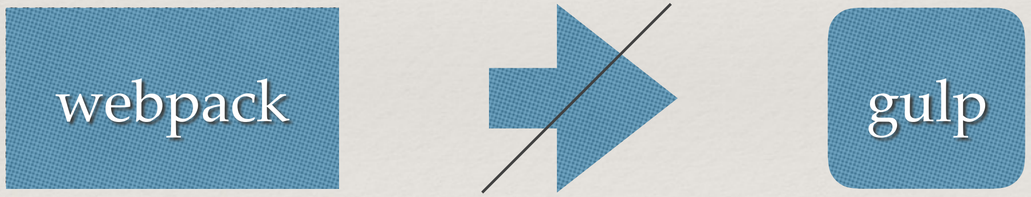


- webpack 是一个现代 JavaScript 应用程序的模块打包器。当 webpack 处理应用程序时，它会递归地构建一个依赖关系图，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 模块。

1.2 webpack与gulp、browserify的关系

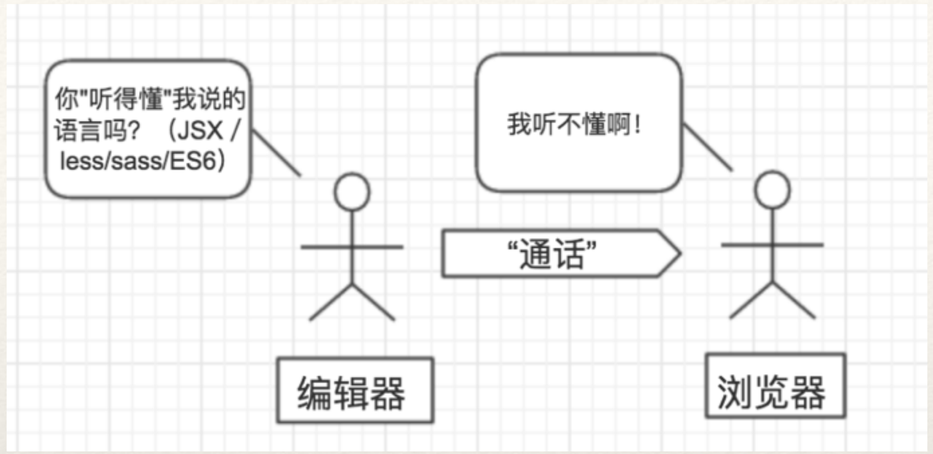
（1）本质不同：

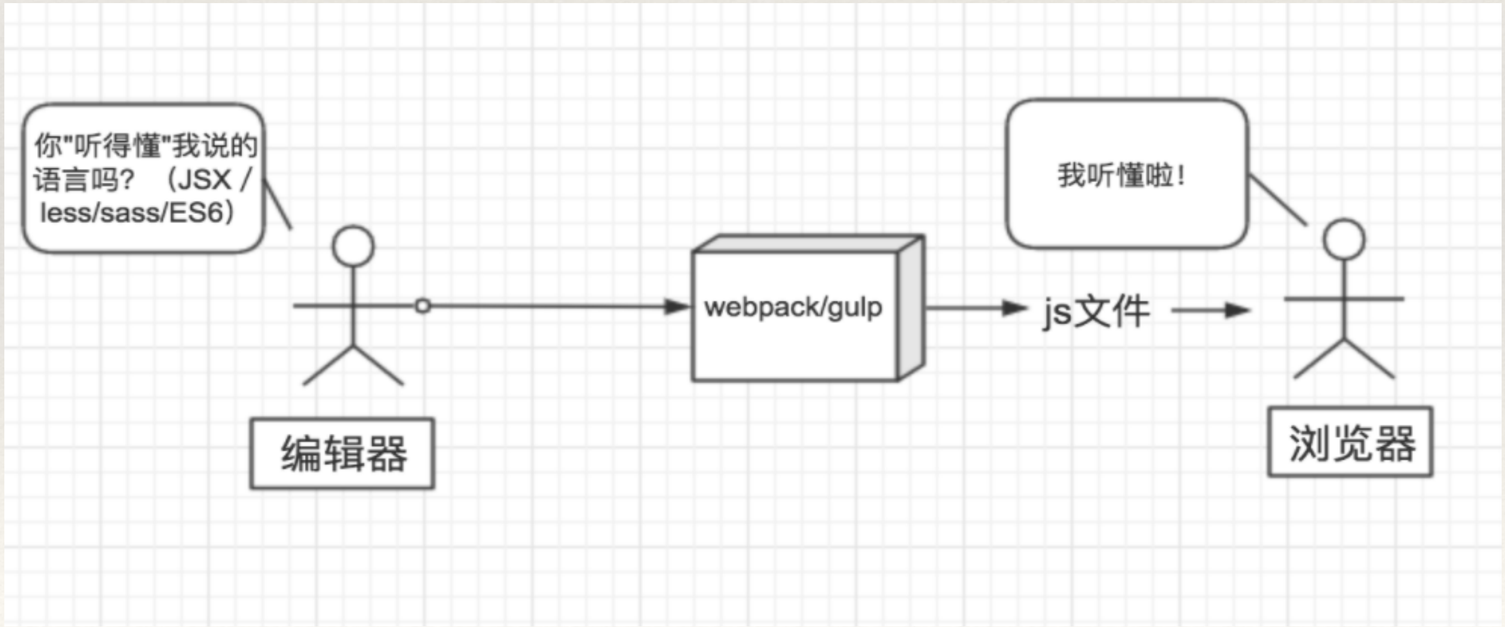
- webpack：一个模块化工具（a module bundle）
- gulp：一个任务运行器（a task runner）



模块化，按需加载　　自动化开发流程工具

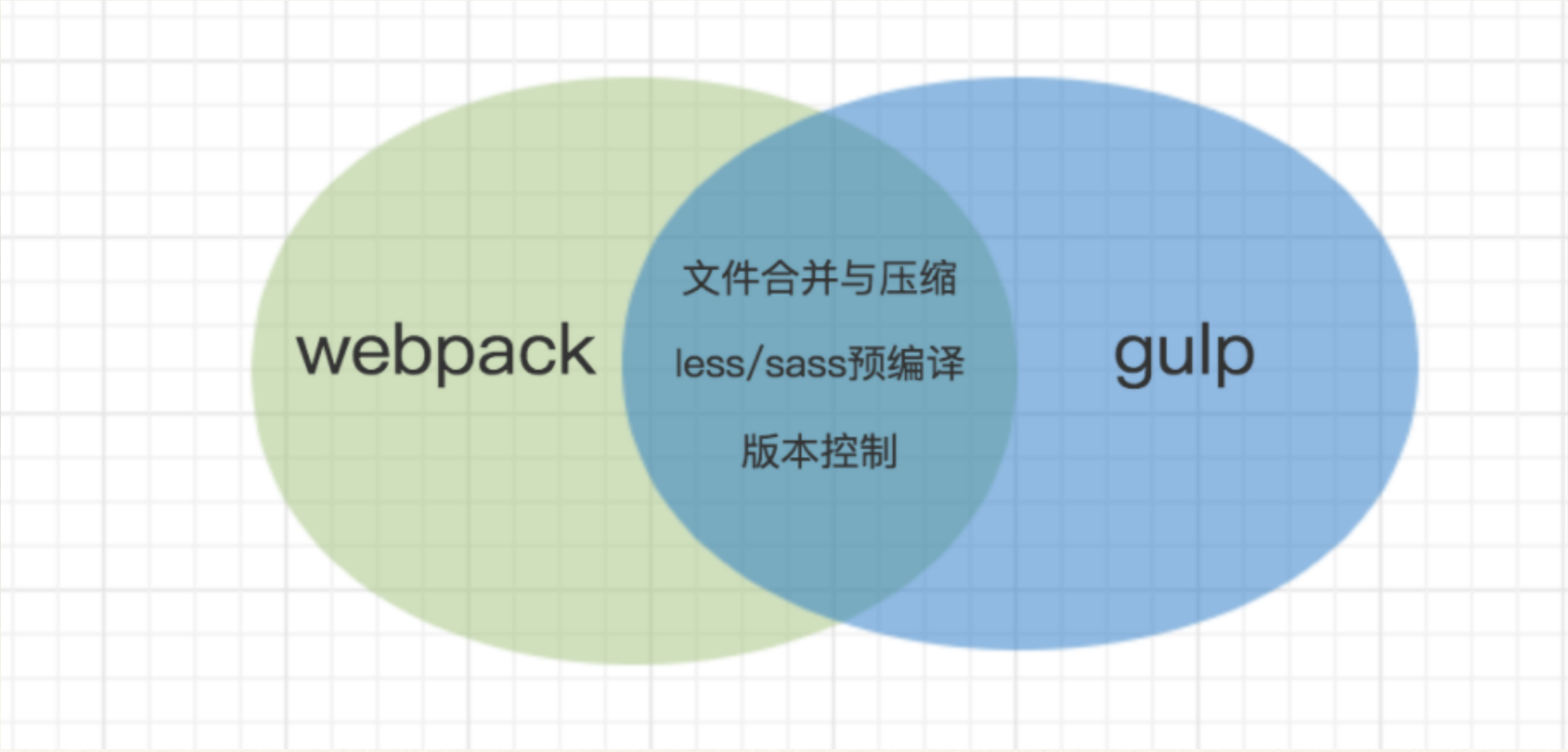
（2）webpack与gulp相同点：让各类语言变成浏览器可识别的语言。





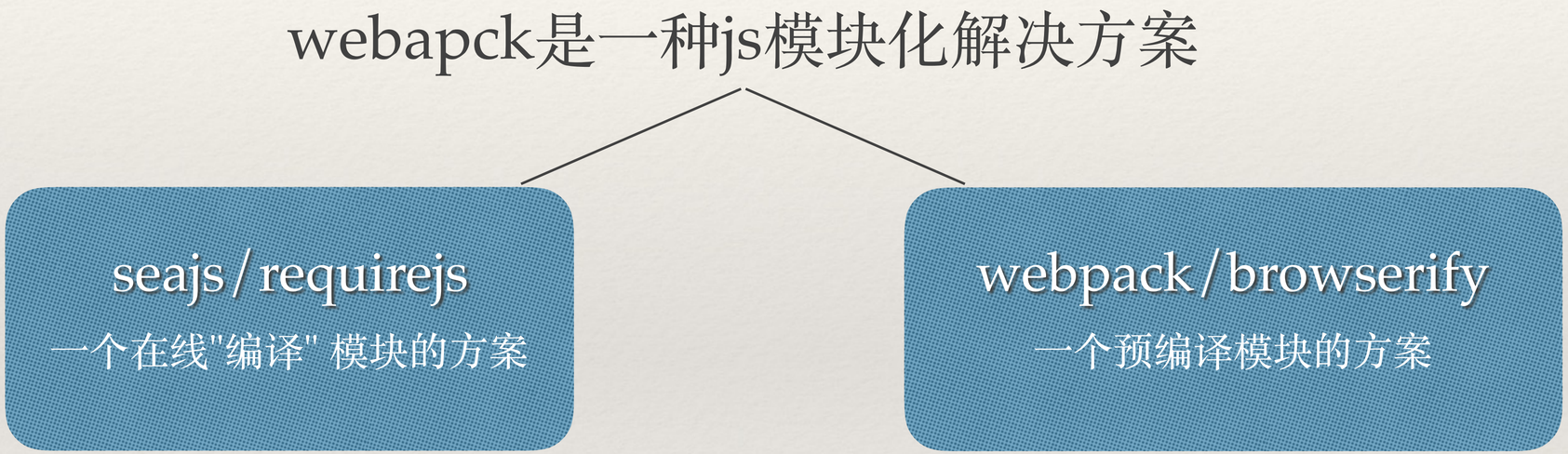
（3）webpack与gulp区别总结

- 无可比性，不过webpack的优点使得webpack可以替代gulp/grunt类的工具。
- 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。



（4）webpack的定位

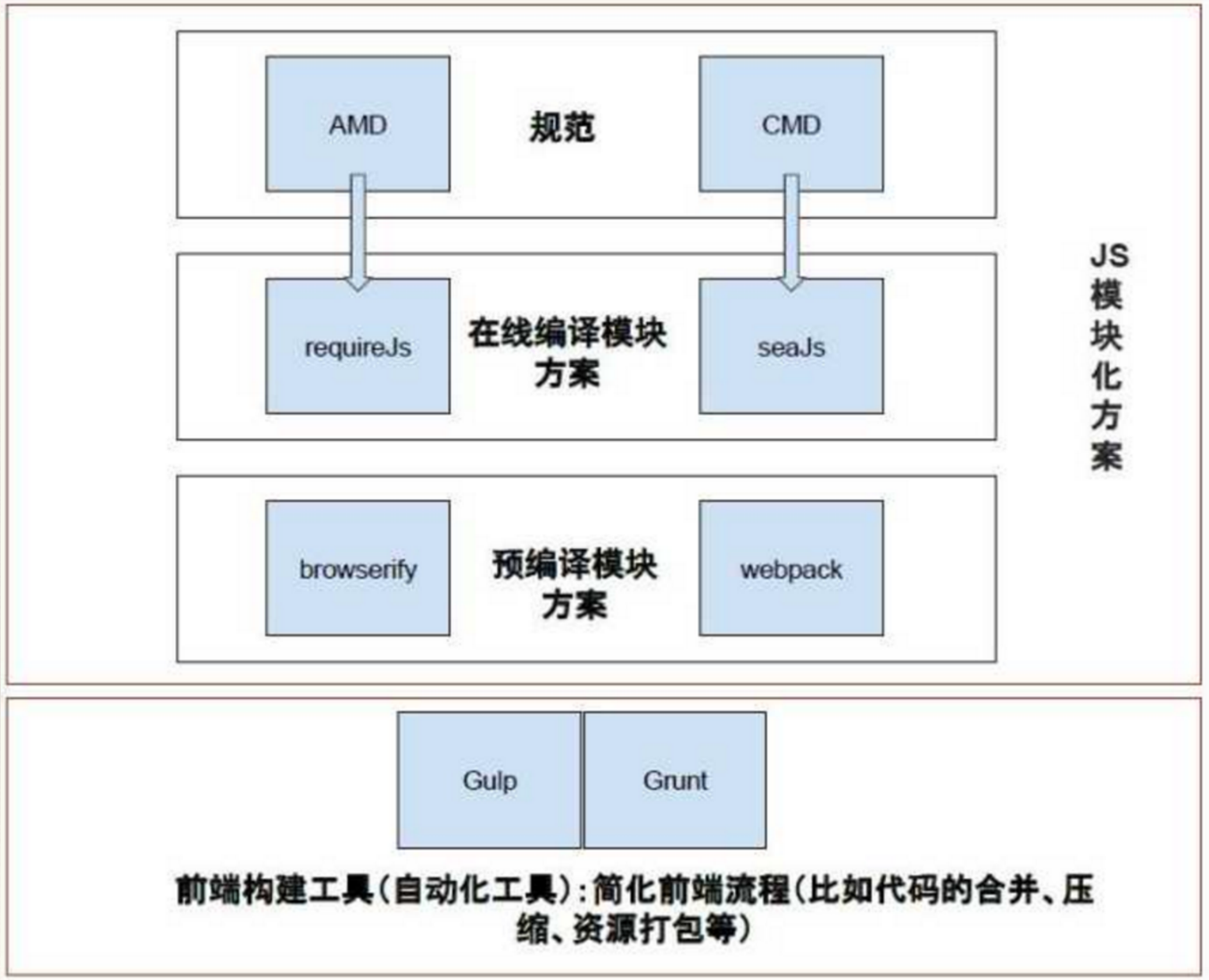
- webpack是模块化方案，更加强调模块化开发。
- gulp是一个工具，旨在规范前端开发流程。
- gulp也可以配置seajs、requirejs甚至webpack的插件



相当于在页面上加载一个 CMD/AMD 解释器，
让浏览器识别 define、exports、module
这些东西，也就实现了模块化。

这个方案更加智能。预编译，本地直接写js。
不管是 AMD / CMD / ES6 风格的模块化，它都
能认识，并且编译成浏览器认识的JS。

（5）webpack与browserify、gulp三者之间的关系图



1.3 webpack的应用场景

（1）疑问

- webpack只适合前后端分离的项目结构吗?
- webpack适合多页面的项目构建吗?

（2）解答

- webpack跟前后端分离没关系。只要配置得到，可以实现前后端分离。
- 与单页面或者多页面无关。

（3）应用

- 常用于vue和react等项目中，需要了解;
- 可用处插件开发，进行模块处理。

1.4 webpack的特性

（1）webpack能更好的落实到业务，提高开发效率。

（2）一切皆模块，即完成代码分割：

- 正如js文件可以是一个“模块（module）”一样，其他的（如css、image或html）文件也可视作模 块。因此，你可以require('myJSfile.js')亦可以require('myCSSfile.css')。这意味着我们可以将事物（业务）分割成更小的易于管理的片段，从而达到重复利用等的目的。

（3）按需加载，懒加载：

- 传统的模块打包工具（module bundlers）最终将所有的模块编译生成一个庞大的bundle.js文件。但是在真实的app里边，“bundle.js”文件可能有10M到15M之大可能会导致应用一直处于加载中状态。因此Webpack使用许多特性来分割代码然后生成多个“bundle”文件，而且异步加载部分代码以实现按需加载。

（4）同时可以用到nodejs中的require和export等语法，快速便捷。

二、webpack基本用法

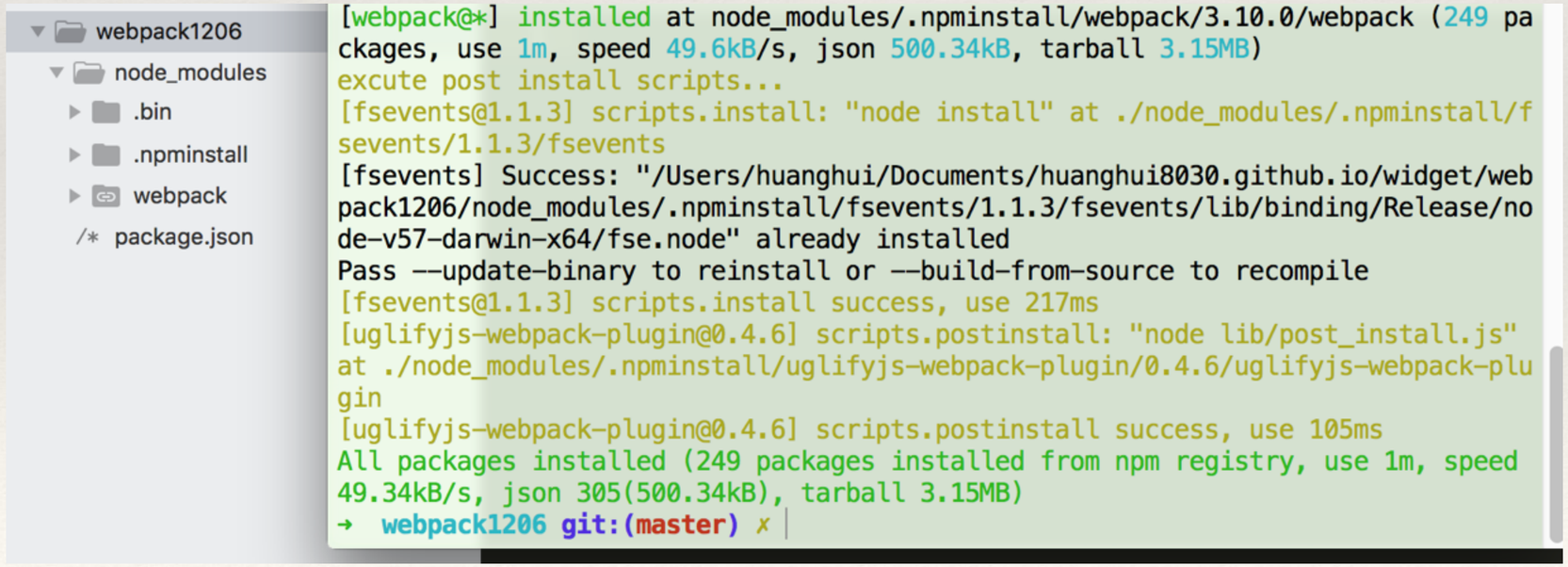
- 2.1 安装webpack
- 2.2 webpack的基本demo
- 2.3 webpack命令行

2.1安装webpack

- 全局安装：cnpm install webpack -g

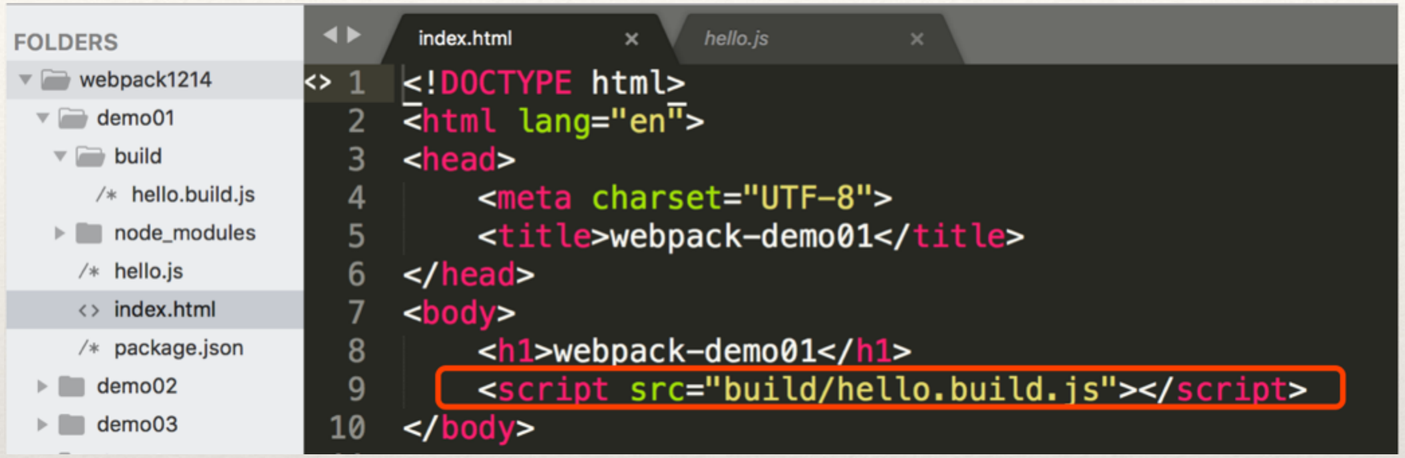
```
[→  demo04 git:(master) ✗ webpack -v
3.4.0
```

- 项目依赖：cnpm install webpack --save-dev
- 项目初始化：cnpm init



2.2 webpack的基本demo

- 页面：index.html

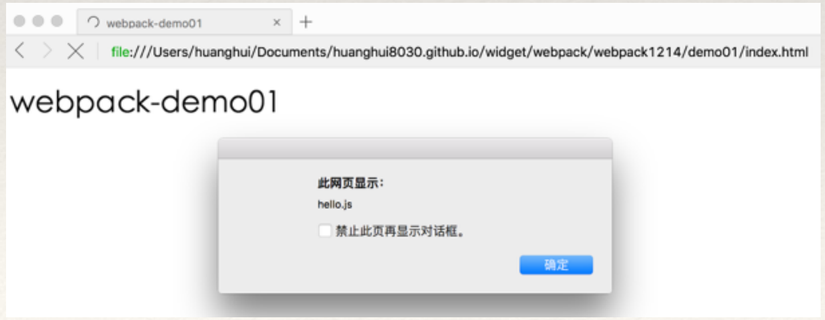


- 脚本：hello.js

- 依赖：cnpm install webpack --save-dev
- 执行命令行：
  - webpack hello.js build/hello.build.js

```
→  demo01 git:(master) ✗ webpack hello.js build/hello.build.js

Hash: d1b054bf01b12ba31164
Version: webpack 3.10.0
Time: 59ms
          Asset     Size  Chunks             Chunk Names
hello.build.js  2.61 kB       0  [emitted]  main
   [0] ./hello.js 140 bytes {0} [built]
```

- 浏览器访问页面



- 打包后的脚本：build/hello.build.js

```
index.html | hello.js | hello.build.js
/******/             function getDefault() { return module['default']; } :
/******/             function getModuleExports() { return module; };
/******/         __webpack_require__.d(getter, 'a', getter);
/******/         return getter;
/******/     };
/******/
/******/     // Object.prototype.hasOwnProperty.call
/******/     __webpack_require__.o = function(object, property) { return Ob
/******/
/******/     // __webpack_public_path__
/******/     __webpack_require__.p = "";
/******/
/******/     // Load entry module and return exports
/******/     return __webpack_require__(__webpack_require__.s = 0);
/******/ })
/************************************************************************/
/******/ ([
/* 0 */
/***/ (function(module, exports) {

/**
 * 只通过命令行来压缩js
 * 依赖: cnpm install webpack
 * 执行:  webpack hello.js build/hello.build.js
 */

a();

function a(){
    alert('hello.js');
}

/***/ })
```


详见demo01

2.3 webpack命令行

- --watch：热更新
- -p：打包文件压缩

```
→  demo02 git:(master) ✗ webpack src/test.js bulid/index.js -p
Hash: 413acdbf15b76e0f53a1
Version: webpack 3.10.0
Time: 919ms
   Asset     Size  Chunks             Chunk Names
index.js  6.37 kB       0  [emitted]  main
   [0] ./src/test.js 479 bytes {0} [built]
   [1] ./node_modules/.npminstall/style-loader/0.19.0/style-loader!./node_modules/
.npminstall/css-loader/0.28.7/css-loader!./src/test.css 1.13 kB {0} [built]
   [2] ./node_modules/.npminstall/css-loader/0.28.7/css-loader!./src/test.css 263
bytes {0} [built]
   [6] ./src/a.js 45 bytes {0} [built]
    + 3 hidden modules
```

index.js

```
!function(t){function e(r){if(n[r])return n[r].
```


详见demo02

- --progress：打包过程，中间有进度。

```
→  demo02 git:(master) ✗ webpack src/test.js bulid/index.js -p --progress
 10% building modules 3/4 modules 1 active ...bpack/webpack1214/demo02/src/test.cs
s
```

- --display-modules：打包中的所有文件

```
→  demo02 git:(master) ✗ webpack src/test.js bulid/index.js --display-modules
Hash: 020bc54e2866be2f8cee
Version: webpack 3.10.0
Time: 250ms
   Asset     Size  Chunks             Chunk Names
index.js  19.3 kB       0  [emitted]  main
   [0] ./src/test.js 479 bytes {0} [built]
   [1] ./node_modules/.npminstall/style-loader/0.19.0/style-loader!./node_modules/
.npminstall/css-loader/0.28.7/css-loader!./src/test.css 1.13 kB {0} [built]
   [2] ./node_modules/.npminstall/css-loader/0.28.7/css-loader!./src/test.css 299
bytes {0} [built]
   [3] ./node_modules/.npminstall/css-loader/0.28.7/css-loader/lib/css-base.js 2.2
6 kB {0} [built]
   [4] ./node_modules/.npminstall/style-loader/0.19.0/style-loader/lib/addStyles.j
s 9.41 kB {0} [built]
   [5] ./node_modules/.npminstall/style-loader/0.19.0/style-loader/lib/urls.js 3.0
1 kB {0} [built]
   [6] ./src/a.js 45 bytes {0} [built]
```

- --display-reasons：打包原因

```
→  demo02 git:(master) ✗ webpack src/test.js bulid/index.js --module-bind 'css=style-loa
der!css-loader' --display-modules --display-reasons
Hash: 83e633f3e7b5df1be919
Version: webpack 3.10.0
Time: 237ms
   Asset     Size  Chunks             Chunk Names
index.js  19.3 kB       0  [emitted]  main
   [0] ./src/test.js 395 bytes {0} [built]
   [1] ./src/test.css 1.13 kB {0} [built]
       cjs require ./test.css [0] ./src/test.js 9:0-21
   [2] ./node_modules/.npminstall/css-loader/0.28.7/css-loader!./src/test.css 299 bytes
{0} [built]
       cjs require !!../node_modules/.npminstall/css-loader/0.28.7/css-loader/index.js!.
/test.css [1] ./src/test.css 4:14-103
   [3] ./node_modules/.npminstall/css-loader/0.28.7/css-loader/lib/css-base.js 2.26 kB {
0} [built]
       cjs require ../node_modules/.npminstall/css-loader/0.28.7/css-loader/lib/css-base
.js [2] ./node_modules/.npminstall/css-loader/0.28.7/css-loader!./src/test.css 1:27-110
   [4] ./node_modules/.npminstall/style-loader/0.19.0/style-loader/lib/addStyles.js 9.41
 kB {0} [built]
       cjs require !../node_modules/.npminstall/style-loader/0.19.0/style-loader/lib/add
Styles.js [1] ./src/test.css 12:13-102
   [5] ./node_modules/.npminstall/style-loader/0.19.0/style-loader/lib/urls.js 3.01 kB {
0} [built]
       cjs require ./urls [4] ./node_modules/.npminstall/style-loader/0.19.0/style-loade
r/lib/addStyles.js 54:14-31
   [6] ./src/a.js 45 bytes {0} [built]
       cjs require ./a.js [0] ./src/test.js 10:0-17
```

- -d： 提供source map，方便调式代码

- --module-bin：css引入时，不用加载loader，则命令行需要修改
  - 如果把test.css中的文件引入修改一下，require('style-loader!css-loader!./test.css'); 改成 require('./test.css')
  - 则，命令行执行时，需要改成：webpack src/test.js bulid/index.js --module-bind 'css=style-loader!css-loader'

```
*/
require('style-loader!css-loader!./test.css');
require('./a.js');

b();

function b(){
    console.log('test.js1111');
}
```

```
require('./test.css');
require('./a.js');

b();

function b(){
    console.log('test.js1111');
}
```

```
→  demo02 git:(master) ✗ webpack src/test.js bulid/index.js
Hash: 004bae553ac33a1fbc38
Version: webpack 3.10.0
Time: 304ms
   Asset     Size  Chunks             Chunk Names
index.js  19.2 kB       0  [emitted]  main
   [0] ./src/test.js 370 bytes {0} [built]
   [1] ./node_modules/.npminstall/style-loader/0.19.0/style-loader!./node_modules
.npminstall/css-loader/0.28.7/css-loader!./src/test.css 1.13 kB {0} [built]
   [2] ./node_modules/.npminstall/css-loader/0.28.7/css-loader!./src/test.css 299
bytes {0} [built]
   [6] ./src/a.js 45 bytes {0} [built]
    + 3 hidden modules
```

```
→  demo02 git:(master) ✗ webpack src/test.js bulid/index.js --module-bind 'css=sty
le-loader!css-loader'
Hash: 37d3c873d10463d0bf4f
Version: webpack 3.10.0
Time: 281ms
   Asset     Size  Chunks             Chunk Names
index.js  19.3 kB       0  [emitted]  main
   [0] ./src/test.js 479 bytes {0} [built]
   [1] ./src/test.css 1.13 kB {0} [built]
   [2] ./node_modules/.npminstall/css-loader/0.28.7/css-loader!./src/test.css 299
bytes {0} [built]
   [6] ./src/a.js 45 bytes {0} [built]
    + 3 hidden modules
```


详见demo02

- --config：指定打包文件
  - webpack --config webpack.config.js，需要新建配置文件
  - 解决什么？　　解决了开发和发布不同的环境的问题
  - 在哪配置呢？　package.jspn文件中的scripts项配置
  - 如何启动呢？　其在终端执行的方式有些不同，分两种情况：
    - start命令：cnpm start
    - 其他name命令：cnpm run name

```
"scripts": {
  "start": "webpack --config webpack.config.js -p",
  "detail": "webpack --config webpack.config.js --progress --disp
  "c": "webpack --config webpack.config.css.js --p",
  "l": "webpack --config webpack.config.less.js --p",
  "u": "webpack --config webpack.config.url.js --p",
  "e": "webpack --config webpack.config.extract.js -p",
  "o": "webpack --config webpack.config.optimize.js --p"
},
```

```
→  demo03 git:(master) ✗ cnpm start

> webpack1206@1.0.0 start /Users/huanghui/Documents/hua
nghui8030.github.io/widget/webpack/webpack1214/demo03
> webpack --config webpack.config.js -p
```

```
→  demo03 git:(master) ✗ cnpm c
npm ERR! Usage:
npm ERR! npm config set <key> <value>
npm ERR! npm config get [<key>]
npm ERR! npm config delete <key>
npm ERR! npm config list
npm ERR! npm config edit
npm ERR! npm set <key> <value>
npm ERR! npm get [<key>]
npm ERR!
npm ERR! alias: c
→  demo03 git:(master) ✗ cnpm run c

> webpack1206@1.0.0 c /Users/huanghui/Documents/huanghu
i8030.github.io/widget/webpack/webpack1214/demo03
> webpack --config webpack.config.css.js --p
```


详见demo04

三、webpack参数配置

- 3.1 配置文件webpack.config.js
- 3.2 entry和output，入口/出口配置
- 3.3 loaders 加载器配置
- 3.4 plugins 插件配置
- 3.5 resolve 其他配置
- 3.6 小结

3.1 配置文件webpack.config.js

- 新建配置文件：webpack.config.js或者是webpack.config.othername.js

webpack.config.js

```
module.exports = {
    entry: "./static/js/optimize.js",
    output: {
        path: __dirname + "/build/demo07/",
        filename: "js/[name]-[hash:8].js"
    },
    module: {
        rules: [
            {
                test: /\.css$/,
                use: ExtractTextPlugin.extract({
                    fallback: "style-loader",
                    use: ["css-loader",'postcss-loader']
                })
            }
        ]
    },
    plugins: [
        new webpack.BannerPlugin('这是我的注释! ')
    ]
};
```

（1）名称

- 新增加配置文件：每个项目下都必须配置有一个 webpack.config.js ， 它的作用如同常规的 gulpfile.js。就是一个配置项，告诉 webpack 它需要做什么。

（2）主要配置项：

- entry 是页面入口文件配置。
- output 是对应输出项配置。
- module.rules中的loaders 是最关键的一块配置。它告知 webpack 每一种文件都需要使用什么加载器来处理。
- plugins 是插件项。

（3）其他配置

- resolve 辅助配置。
- devtool 开发时使用，能够较快识别源文件。
- devServer 启动本地服务

3.2 entry和output

- 最主要的两个配置（有这两个就可以直接使用webpack了）
  - entry有三种数据类型：String、Array、Object。
  - output，有各种配置项，可实现多目标文件以及静态资源替换等效果。

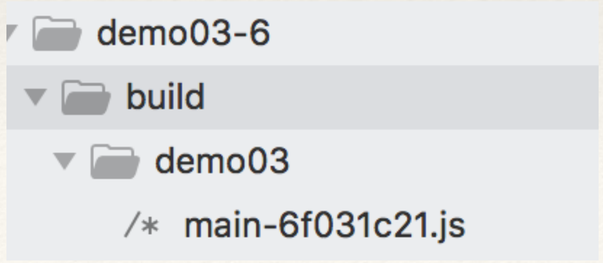
```
{
    entry: [String | Array | Object], // 入口模块
    output: {
        path: String,       // 输出路径
        filename: String    // 输出名称或名称
        publicPath: String // 指定静态资源的位置
        ...                 // 其他配置
    }
}
```

（1）entry： String类型，只能单一入口，无别名。

webpack.config.js

```
/**...
var webpack = require('webpack');
module.exports = {
    entry:"./src/js/a.js",
    output: {
        path: __dirname + "/build/demo03/",
        filename: "[name]-[hash:8].js"
    }
};
```

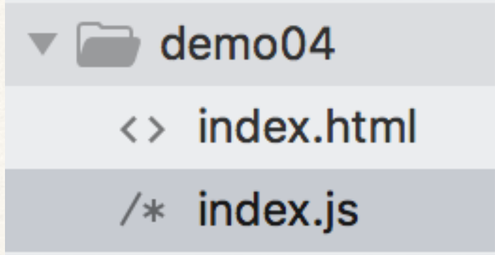
```
➜  demo03-6 git:(master) ✗ cnpm start

> demo02@1.0.0 start /Users/huanghui/Documents/huanghui8030.
github.io/widget/webpack/webpack1214/demo03-6
> webpack -p

Hash: 6f031c210b7dbd85ffb5
Version: webpack 3.10.0
Time: 92ms
          Asset       Size  Chunks                    Chunk Names
main-6f031c21.js  514 bytes       0  [emitted]         main
   [0] ./src/js/a.js 46 bytes {0} [built]
```



详见demo03

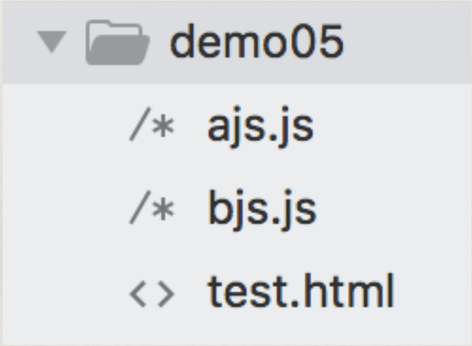
（2）entry： Array类型，多文件入口，无别名。

```
//第二种Array类型
entry: ["./src/js/a.js","./src/js/b.js"],
output: {
    path: __dirname + "/build/demo04/",
    filename: "index.js?v=[hash:8]"
},
```



（3）entry： Obejct类型，多文件入口，有别名。

```
//第二种Object类型
entry: {
    ajs : "./src/js/a.js",
    bjs : "./src/js/b.js",
},
output: {
    path: __dirname + "/build/demo05/",
    filename: "[name].js"
},
```



- String只能是单目标文件，Array和Object的entry可实现多目标文件。

详见demo04/demo05

（4）output：实现多目标文件。通过output.filename 进行参数配置。

- [name]：entry 对应的名称，默认为main。
- [hash]：webpack 命令执行结果显示的 Hash 值，默认20个字符长度。其中可以约定hash的长度，例如：[hash:8]。

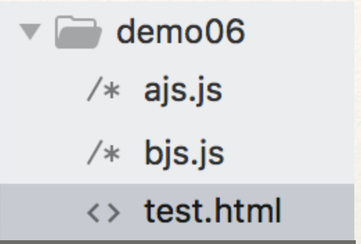
```
> webpack --config webpack.config.d6.js -p

Hash: 56501484f4555fcc2bd3
Version: webpack 3.10.0
Time: 439ms
```

- [chunkhash]：chunk 的 hash。为了让编译的结果名称是唯一的，可以利用 hash 。

详见demo05/demo06

（5）output： 实现静态资源替换。

- 通过output.publicPath 来替换output.path
- 同时结合html模板插件：html-webpack-plugin




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

module.exports = {
    entry: {
        ajs : "./src/js/a.js",
        bjs : "./src/js/b.js",
    },
    output: {
        path: __dirname + "/build/demo06/",
        filename: "[name].js",
        publicPath:"https://t1.chei.com.cn/"   //静态路径替换
    },
    plugins:[
        new HtmlWebpackPlugin({
            title:'测试webpack',
            filename: 'test.html',
            inject: 'header',
            hash:'true'    //自动添加参数去，缓存。
        })  //默认加载一个index.html页面
    ]
};
```

test.html × webpack.config.d6.js ×

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>测试webpack</title>
  </head>
  <body>
  <script type="text/javascript" src="https://
  t1.chei.com.cn/bjs.js?56501484f4555fcc2bd3"></script>
  <script type="text/javascript" src="https://
  t1.chei.com.cn/ajs.js?56501484f4555fcc2bd3"></script>
  </body>
</html>
```


详见demo06

3.3 loaders

（1）loaders作用

- 它告知 webpack 每一种文件都需要使用什么加载器来处理。
- 将各种文件类型转为webpack可识别的js文件类型。

（2）loaders安装

- 所有的加载器都需要通过 npm 来加载
- 例如：cnpm install less-loader —save-dev

（3）loaders 功能

- loaders 管道：在同一种类型的源文件上，可以同时执行多个 loaders，loaders 的执行方式可以类似管道的方式，管道执行的方式是从右到左的方式。
- loaders 可以支持同步和异步。
- loaders 可以接收配置参数。
- loaders 可以通过正则表达式或者文件后缀指定特定类型的源文件。
- loaders 除了做文件转换以外，还可以创建额外的文件。
- 插件可以提供给 loaders 更多功能。

（4）loaders配置：（webpack版本不同loader的用法方式也不同）

- webpack2.0

```
module: {
        //加载器配置
        loaders: [
            //.css 文件使用 style-loader 和 css-loader 来处理
            { test: /\.css$/, loader: 'style-loader!css-loader' },
            //.js 文件使用 jsx-loader 来编译处理
            { test: /\.js$/, loader: 'jsx-loader?harmony' },
            //.scss 文件使用 style-loader、css-loader 和 less-loader 来编译处
            { test: /\.les$/, loader: 'style!css!less?sourceMap'},
            //图片文件使用 url-loader 来处理，小于8kb的直接转为base64
            { test: /\.(png|jpg)$/, loader: 'url-loader?limit=8192'}
        ]
    }
```

- webpack3.0

```
module: {
       rules: [{
         test: /\.(png|gif|svg)$/,
         use: [{
           loader: 'url-loader',//转成base64格式图片
           options: {limit: 2000}
       }]},{
          test: /\.(jpg)$/,
          use: [{
            loader: 'file-loader',
            options:{name: 'images/[name].[ext]?[hash:8]'}
       }]}
]}
```

（5）样式相关loader配置：

- style-loader：将css样式以style的方式加载到脚本文件中，样式起作用。
- css-loader：css文件可以直接作为模块加载到其他脚本文件中。
  - options.modules：true，将css文件作为局部变量。

```
module: {
    rules: [
        {test: /\.css$/,
            use: [
                {loader: "style-loader"},
                {loader: "css-loader",
                    options: {
                        modules: true  //css是否局部
                    }
                },
                {loader: "postcss-loader"}  //厂商前缀
            ]
        },{test: /\.less$/,
            use: [
            {loader: "style-loader" },
            {loader: "css-loader" },
            {loader: "less-loader"},
            {loader: "postcss-loader"}]  //厂商前缀
        }
    ]
},
```

- css-loader：局部设置。options.modules：true

```
var Common = require('./common.js');

var Styele = require('../styles/css/atest.css');

test();

function test(){
    console.log('执行b方法！  ');

    Common.commonTest2();

    var rootB = document.getElementById('root');
    rootB.className = Styele.cname;
    rootB.innerHTML = "哈哈哈test";

}
```

css.tmpl.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>1、Webpack: css-loader配置</title>
  </head>
  <body>
    <h1>Webpack: css-loader配置</h1>
    <div id='root'>css简单配置
    </div>
  </body>
</html>
```

```
<html lang="en">
▼<head> == $0
    <meta charset="utf-8">
    <title>1、Webpack: css-loader配置</title>
  ▼<style type="text/css">
      #orUwMIPYjofl_-LceA6xX,._3-jOUH9r_62AQoKiN1-9Kl{display:-webkit-
      box;display:-webkit-flex;display:-ms-
      flexbox;display:flex;width:200px;height:200px;background:#ddf2e8;borde
      r-radius:5px}
    </style>
  </head>
▼<body>
    <div id="root" class="_3-jOUH9r_62AQoKiN1-9Kl">哈哈哈test</div>
    <script type="text/javascript" src="main-01029c7e.js"></script>
  </body>
</html>
```


详见demo08

- postcss-loader：兼容性，加厂商前缀。
  - 需要新建一个配置文件，postcss.config.js

postcss.config.js

```
module.exports = {
  plugins: [
    require('autoprefixer')({                    //加厂商前缀
        browsers: ['Android 2.3','Android >= 4','iOS >= 6',
          'Explorer >= 6','Chrome >= 20','Firefox >= 24','Opera >= 12'],
        cascade: true,                           //是否美化属性值 默认：true
        remove:true                              //是否去掉不必要的前缀 默认：true
    })
  ]
}
```

atest.css

```
#root,.cname{
    display: flex;
    width:200px;
    height:200px;
    background:#DDF2E8;
    border-radius: 5px;
}
```

```
#orUwMIPYjofl_-LceA6xX, ._3-jOUH9r_62AQoKiN1-
9Kl {
  display: -webkit-box;
  display: -webkit-flex;
  display: -ms-flexbox;
  display: flex;
  width: 200px;
  height: 200px;
  background: #ddf2e8;
  border-radius: 5px;
}
```


详见demo08

- less-loader：将less/scss文件转为css文件
  - strictMath，是否严格匹配
  - noIeCompat，是否不兼容ie
  - 注意顺序，从后往前
  - 其他文件类型类似，例如：sass-loader、json-loader、svg-online-loader、mocha-loader、babel-loader

```
},{
    test: /\.less$/,
    use: [
        {loader: "style-loader" },
        {loader: "css-loader" },
        {loader: "less-loader",
            options: {
                strictMath: true,
                noIeCompat: true
            }
        },
        {loader: "postcss-loader"} //厂商前缀
    ]
},{
```

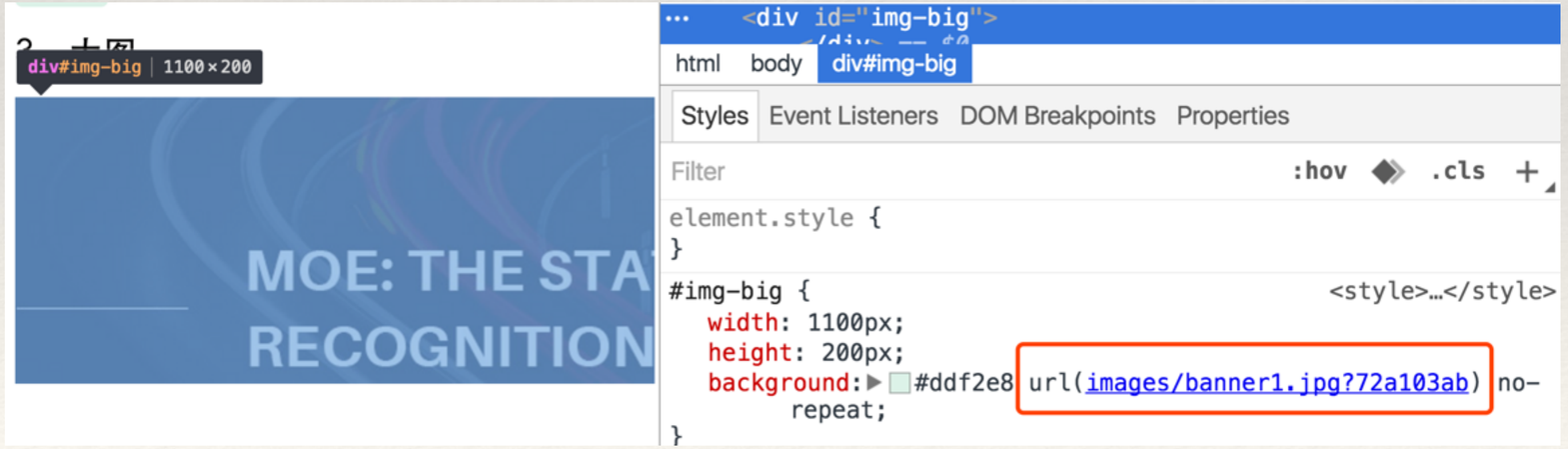

详见demo09

（6）图片相关配置，file-loader和url-loader

- file-loader：文件加载器

```
},{
    test: /\.(jpg)$/,
    use: [
        {
            loader: 'file-loader',
            options:{
                name: 'images/[name].[ext]?[hash:8]'
            }
        }
    ]
}
```

```
#img-big{
    width:1100px;
    height:200px;
    background:@colorbg url(../../images/banner1.jpg) no-repea
}
```



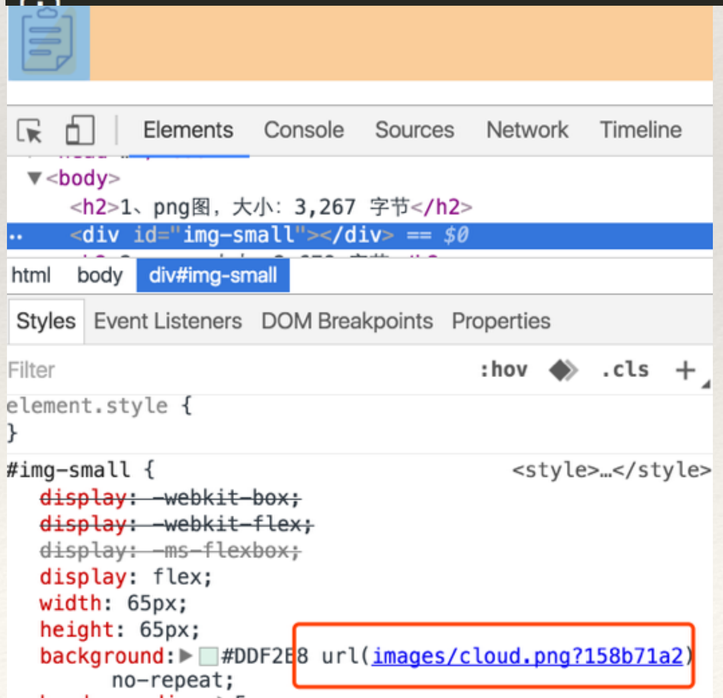
详见demo10

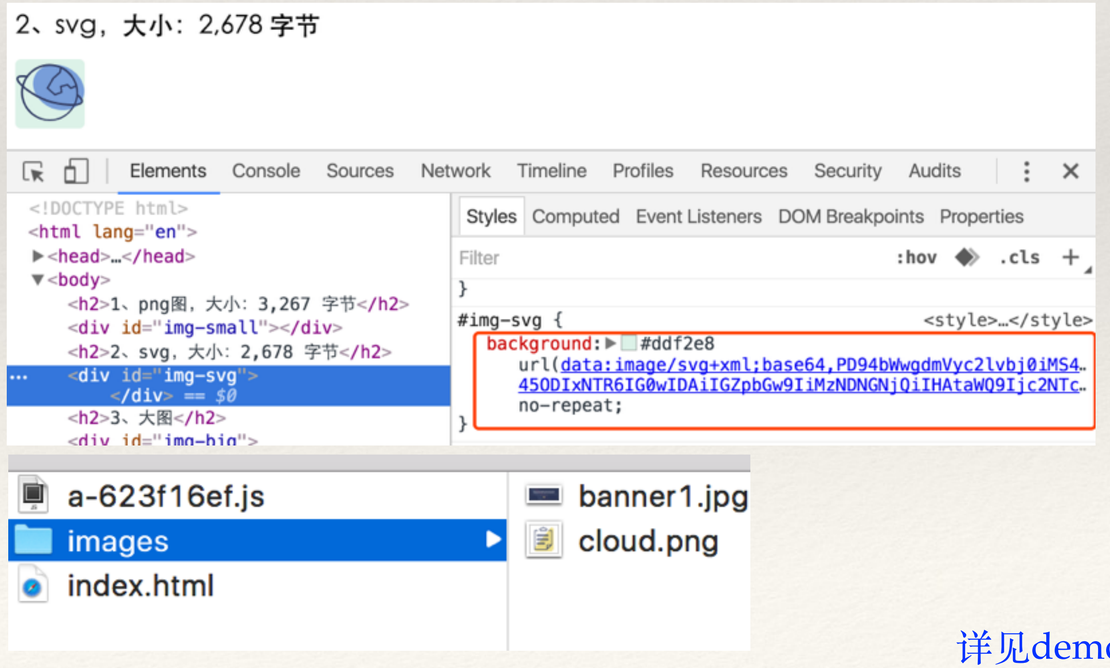
（6）图片相关配置，file-loader和url-loader

- url-loader：将指定格式的文件，转为base64格式图片

```
},{
    test: /\.(png|gif|svg)$/,
    use: [
        {
            loader: 'url-loader',//
            options: {
                limit: 3000, //小于转成base64格式图片
                name: 'images/[name].[ext]?[hash:8]'
            }
        }
    ]
},{
```

```
#img-small{
    display: flex;
    width:@width;
    height:@width;
    background:@colorbg url(../../images/cloud.png) no-repeat;
    border-radius: 5px;
}

#img-svg{
    display: flex;
    width:@width;
    height:@width;
    background:@colorbg url(../../images/global.svg) no-repeat;
    border-radius: 5px;
}
```





详见demo10

3.4 plugins

（1）plugins是什么

- webpack 提供插件机制，可以对每次 build 的结果进行处理。

（2）plugins与loaders的区别

- loaders和plugins常常被弄混，但是他们其实是完全不同的东西：
  - loaders是在打包构建过程中用来处理源文件的（jsx，scss，less..），一个loader处理一类文件；
  - plugins插件直接对整个构建过程其作用，针对全部文件。

（3）plugins分类

- 内置插件：webpack内置方法，不需要安装依赖包，直接通过webpack.BannerPlugin直接使用。
  - BannerPlugin
  - HotModuleReplacementPlugin
- 第三方插件：需要安装npm依赖包，cnpm install html-webpack-plugin —save-dev。
  - html-webpack-plugin
  - extract-text-webpack-plugin
  - clean-webpack-plugin

（4）plugins使用方法

- 配置 plugins 的方法为在 webpack.config.js 中添加
- webpack 提供插件机制，可以对每次 build 的结果进行处理。
- plugins可以携带参数/选项，向plugins属性传入new实例。常见配置如下：

```
plugins: [
    new webpack.BannerPlugin('将css单独分离出来，去重复！'),//压缩文件，注释
    new HtmlWebpackPlugin({
        template: __dirname + "/static/html/optimize.tmpl.html"//new 一个这个插件的实例
    }),
    new webpack.HotModuleReplacementPlugin(),//热加载插件
    new CleanWebpackPlugin('./dist/build-optimize/*', {//清除dist目录
        root: __dirname,
        verbose: true,
        dry: false
    }),
    new ExtractTextPlugin("custom.css"),//合并css
    new OptimizeCssAssetsPlugin({   //去掉重复的css
      assetNameRegExp: /\.optimize\.css$/g,
      cssProcessor: require('cssnano'),
      cssProcessorOptions: { discardComments: {removeAll: true } },
      canPrint: true
    })
],
```

（5）插件详解：添加版权注释

- webpack插件内置方法
- 直接在plugins中使用，直接new一个对象，通过webpack直接调用。
- new webpack.BannerPlugin(‘我是一行注释！')

```
plugins: [
    new webpack.BannerPlugin('这是我的注释! '),
```

- 生成的文件：

a-d4a4d871.js

```
/*! 这是我的注释!  */
!function(e){function n(e){delete installedChunks[e]
```


详见demo7

（6）插件详解：html模板

- 安装npm包： html-webpack-plugin
- 在配置文件webpack.config.js的plugin中new一个对象：new HtmlWebpackPlugin({options})。
- 参数详解：
  - title：标题
  - filename：生成的html文件
  - inject：”true丨body丨header”，js引入的位置。默认true，并且true和body一样，都放到底部。
  - hash：添加hash值，去缓存
  - template：添加自定义模板

（6）插件详解：html模板

- 默认不加配置时，直接生成一个html页面，只引用output中的js。
- webpack.config.js的plugins中增加：new HtmlWebpackPlugin()

index.html × webpack.config.d4.js ×

```
/**...
var webpack = require('webpack'),
    HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
    //第二种Array类型
    entry: ["./src/js/a.js","./src/js/b.js"],
    output: {
        path: __dirname + "/build/demo04/",
        filename: "index.js?v=[hash:8]"
    },
    plugins:[
        new HtmlWebpackPlugin()  //默认加载一个index.html页面
    ]
};
```

- 生成的html模板页面：index.html

index.html ×

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Webpack App</title>
  </head>
  <body>
  <script type="text/javascript" src="index.js?v=5951a698"></script></body>
</html>
```


详见demo4

（6）插件详解：html模板

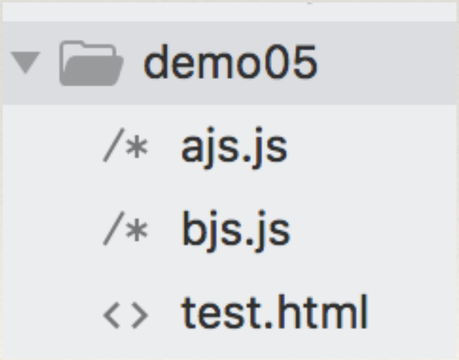
- 通过参数配置html模板，具体配置如下：

```
plugins:[
    new HtmlWebpackPlugin({
        title:'测试webpack',
        filename: 'test.html',
        inject: 'header',
        hash:'true'    //自动添加参数去，缓存。
    })  //默认加载一个index.html页面
]
```

- 生成的html模板页面：test.html

index.html × webpack.config.d5.js × test.html ×

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="UTF-8">
   <title>测试webpack</title>
 </head>
 <body>
 <script type="text/javascript' src="bjs.js?e39da9cc63c24787b006"></
 script><script type="text/javascript" src="
 ajs.js?e39da9cc63c24787b006"></script></body>
</html>
```



详见demo5

（6）插件详解：html模板

- 自定义配置html模板

```
plugins:[
    new HtmlWebpackPlugin({
        template: __dirname + "/src/html/test.tmpl.html"
    })
]
```

- 编写test.tmpl.html模板

index.html × test.tmpl.html ×

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>自定义模板</title>
  </head>
  <body>
    <h1>自定义模板</h1>
    <div id='root'>自定义模板</div>
  </body>
</html>
```

- 生成index.html

index.html ×

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>自定义模板</title>
  </head>
  <body>
    <h1>自定义模板</h1>
    <div id='root'>自定义模板</div>
  <script type="text/javascript" src="https://t1.chei.com.cn/bjs.js">
  </script><script type="text/javascript" src="https://t1.chei.com.cn
  /ajs.js"></script></body>
</html>
```


详见demo6

（7）插件详解，单独提取css

- 安装npm包：extract-text-webpack-plugin
- 在配置文件webpack.config.js中设置
  - 将插件require进来

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

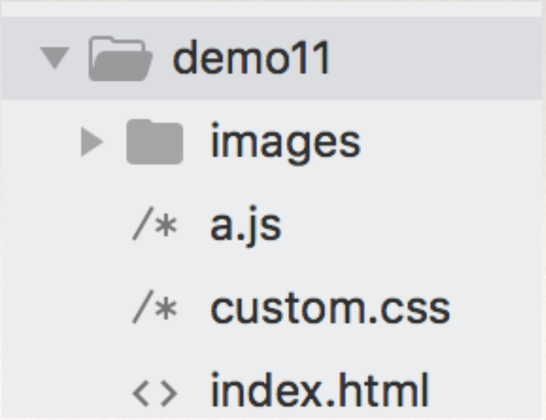
  - module.rules中引入

```
rules: [
    {
        test: /\.css$/,
        use: ExtractTextPlugin.extract({
            use: ["css-loader",'postcss-loader'],
            fallback: "style-loader"
        })
    },{
```

  - plugins中new一个对象，定义提取css的名称

```
plugins: [
    new ExtractTextPlugin("custom.css"),
],
```

- 生成的单独css



详见demo11

（8）插件详解，清除生成的文件

- 安装npm包：clean-webpack-plugin
- 在配置文件webpack.config.js中设置
  - 将插件require进来

```
var CleanWebpackPlugin = require("clean-webpack-plugin");
```

  - 在plugins中new一个对象：

```
plugins: [
    new CleanWebpackPlugin('./build/demo12/*', {//清除dist目录
        root: __dirname,
        verbose: true,
        dry: false
    }),
```

  - 其中路径一般为output中的path

```
output: {
    path: __dirname + "/build/demo12",
    filename: "js/[name]-[hash:8].js"
},
```


详见demo12

（8）插件详解，清除生成的文件

- clean-webpack-plugin参数说明
  - root，默认__dirname，一个根的绝对路径。
  - verbose，默认：true，将log写到 console。
  - dry，默认：false。删除文件，为true则不删除，主要用于测试。
  - watch，默认：false。为true则只删除重新编译的文件。
  - exclude，排除不删除的目录，主要用于避免删除公用的文件。
  - allowExternal，默认：false-不允许在webpack根目录之外的干净的文件夹。

3.5 其他配置

（1）resolve 配置：

- root，绝对路径。
- extensions，省略文件后缀名。
- alias，模块别名。

```
resolve: {
        //查找module的话从这里开始查找
        root: 'E:/github/flux-example/src', //绝对路径
        //自动扩展文件后缀名，意味着我们require模块可以省略不写后缀名
        extensions: ['', '.js', '.json', '.scss'],
        //模块别名定义，方便后续直接引用别名，无须多写长长的地址
        alias: {
            AppStore : 'js/stores/AppStores.js',//后续直接 require('AppStore') 即可
            ActionType : 'js/actions/ActionType.js',
            AppAction : 'js/actions/AppAction.js'
        }
    }
```

（2）devtool：eval-source-map | source-map

- 开发时使用，便于调试。
- 其中source-map是最详细的，但构建速度慢。

（3）devServer，启动本地服务

- proxy，端口号等设置
- compress，是否进行gzip压缩
- historyApiFallback，404页面是否显示在页面。
- hot，是否热更新，
- noInfo，只有errors 或者是warns时才重新刷新。

3.6 webpack参数配置小结

- 配置文件webpack.config.js
- entry和output，入口/出口配置
  - entry，三种情况String、Array、Object。
  - output，可多个目标输出，也可进行资源替换。
- loaders 加载器配置
  - loaders作用、安装、使用，以及功能。
  - 具体实例，样式和图片相关实例
- plugins 插件配置
  - plugins概率，与loaders区别，分类，以及使用方法。
  - 具体实例：添加注释、html模板、单独提取css、清楚冗余文件。
- resolve 其他配置
  - resolve、devtool、devServer等配置项

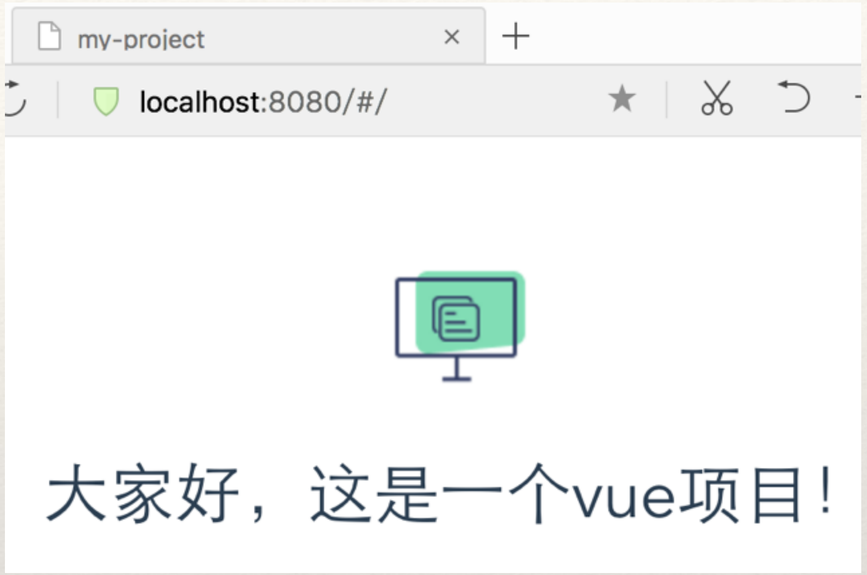
四、webpack实际应用

- 4.1 webpack+vue
- 4.2 webpack+react
- 4.3 webpack+gulp

4.1 webpack+vue

- Vue是一套用于构建用户界面的渐进式框架
- Vue.js的特性：轻量级的框架、双向数据绑定、指令、插件化。
- 有自己的webpack脚手架（建议用，不用自己设置）：vue-cli。

- 用vue-cli来搭建一个项目
  - npm install -g vue-cli
  - vue init webpack my-project
  - cd my-project
  - npm install
  - npm run dev



- 启动服务后，在浏览器中输入：http://localhost:8080
- 在项目下面的src/App.vue和src/main.js中输入自己的代码即可。

App.vue

```vue
<template>
  <div id="app">
    <img src="./assets/online.png">
    <router-view/>
  </div>
</template>

<script>
export default {
  name: 'app'
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

main.js

```js
// The Vue build version to load
// (runtime-only or standalone) h
import Vue from 'vue'
import App from './App'
import router from './router'

Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  template: '<App/>',
  components: { App }
})
```


详见demo13

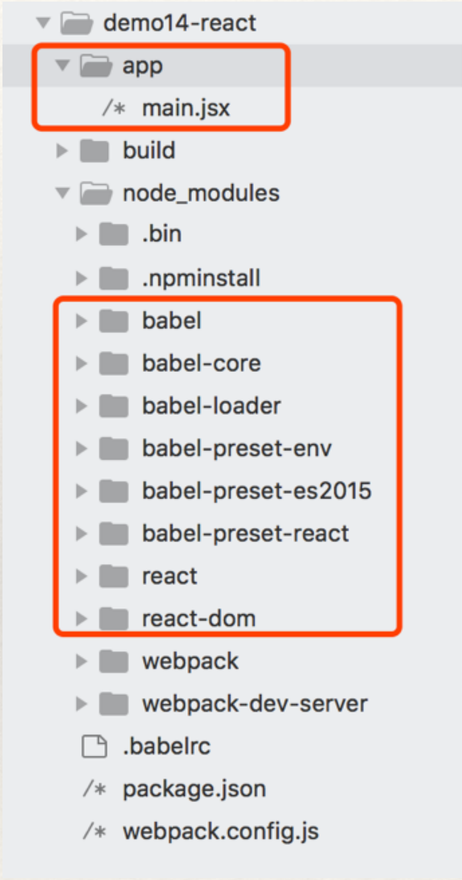
4.2 webpack+react

- React是目前非常热门的一个JavascriptMVC框架。
- 主要以虚拟DOM而闻名。
- 组件化、易复用解耦、数据控制视图 、没有太多api，知道对应的写作模式就可以上手。
- 与webpack结合是标配。

- webpack+react实例
  - 需要安装的npm包，以及路径。
  - 新建main.jsx文件

main.jsx

```jsx
import React,{Component} from 'react';
import ReactDOM from 'react-dom';

class Hello extends Component{
    render(){
        return (
            <div>
                <h1>Hello world</h1>
                <p>这是一个react的demo</p>
            </div>
        );
    }
}

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



- webpack.config.js配置，如下：

main.jsx | webpack.config.js | package.json

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

module.exports = {
    entry:  __dirname + '/app/main.jsx',
    output: {
        path: path.resolve(__dirname, './build'),
        filename: 'bundle.js',
    },
    devtool: 'eval-source-map',
    module: {
        rules:[{
                test: /\.(js|jsx)$/,
                use: [
                    {
                        loader: 'babel-loader'
                    }
                ],
                exclude: /(node_modules|bower_components)/,
            }]
    },
```

- 生成页面：

webpack.config.js | index.html | packa

```
<!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>
```

Hello world

这是一个react的demo

详见demo14

4.3 webpack + gulp

```
//引入js 模块化工具gulp-webpack,
var webpack = require('gulp-webpack');
var jsFiles = [
    './src/scripts/app.js'
];
gulp.task('packjs',function () {
    gulp.src(jsFiles)
    .pipe(webpack({
        output:{
            filename:'[name].js'
        },
        module:{
            loaders:[
                {
                    test:/\.js$/,
                    loader:'imports?define=>false'
                }
            ]
        }
    }))
    .pipe(gulp.dest('./build/prd/scripts/'))
})
```

- 两者一起合作：前端工程化；更好的管理前端代码；同时符合现有项目配置。
  - gulp：处理html压缩/预处理/条件编译，图片压缩，图片自动合并等任务；
  - webpack：管理模块化，构建js/css。
  - 插件：gulp-webpack，将webpack结合到gulp中。
  - webpack原生方法

```
// 生成js/css
gulp.task('webpack', ['clean:webpack'], function(callback) {
    webpack(require('../webpack.config.js')(), function(err, stats) {
        compileLogger(err, stats);
        callback();
    });
});
```

总结

- webpack简介
  - 基本概念、与gulp的详细区别、应用场景，以及特性。
- webpack安装和执行
  - 如何安装、简单demo，以及命令行用法。
- webpack参数配置
  - 主要参数配置：entry、output、loaders、plugins
  - 其他参数配置：resolve、devtool、devServer等配置项
- webpack应用
  - 与vue、webpack等框架的应用实例
  - 与gulp相互结合使用的应用实例

参考资料

- webpack官方网址
- 入门Webpack，看这篇就够了
- 彻底解决Webpack打包性能问题
- 前端工程与模块化框架
- gulp+webpakc构建多页面前端项目

The end， Thanks!

—— 设计部 黄卉 *20180116*

huangh@chsi.com.cn