

第六节 Glup

1.介绍

2. 准备

2.1 安装gulp命令行工具

2.2 安装gulp

2.3 初始化项目

2.4 创建gulpfile.js

2.5 执行

3.创建任务

3.1 导出任务

3.2 组合任务

4.异步执行

4.1 callback

4.2 promise

4.3 stream

4.4 child process

4.5 event emitter

4.6 使用async/await

5.处理文件

6.特殊字符

6.1 分隔符

6.2 *一个星号

6.3 **两个星号

6.4 !取反

7.gulp的核心原理

8.一个完整的例子

8.1 sass的编译

8.2 处理js

8.3 模板编译

8.4 字体和图片转换

8.5 编译清理和处理其他静态文件

8.6 自动加载插件

8.7 开发服务器

8.8 修改src代码后自动编译

8.9 useref文件引用处理

8.9 按需导出任务

9.封装自动化 workflow

1. 介绍

- 基于 node 强大的流(stream)能力，gulp 在构建过程中并不把文件立即写入磁盘，从而提高了构建速度。
- 代码优于配置、node 最佳实践、精简的 API 集，gulp 让工作前所未有的简单。
- 遵循严格的准则，确保我们的插件结构简单、运行结果可控。

2. 准备

2.1 安装gulp命令行工具

```
1 npm install --global gulp-cli
```

2.2 安装gulp

```
1 npm install --save-dev gulp
```

2.3 初始化项目

```
1 yarn init -y
```

2.4 创建gulpfile.js

一般都是导出函数的形式

因为gulp新版的任务都是异步的所以要执行cb,标识任务执行完成

```

1 function defaultTask(cb) {
2   // place code for your default task here
3   // 因为gulp新版的任务都是异步的所以要执行cb,标识任务执行完成
4   cb();
5 }
6
7 exports.default = defaultTask

```

2.5 执行

```
1 gulp 函数名(default可以省略)
```

如需运行多个任务 (task) , 可以执行 `gulp <task> <othertask>`。

3.创建任务

- 公开任务 (Public tasks) 从 gulpfile 中被导出 (export) , 可以通过 `gulp` 命令直接调用。
- 私有任务 (Private tasks) 被设计为在内部使用, 通常作为 `series()` 或 `parallel()` 组合的组成部分。

3.1 导出任务

如需将一个任务 (task) 注册为公开 (public) 类型的, 只需从 gulpfile 中导出 (export) 即可。

```

1 const { series } = require('gulp');
2
3 // `clean` 函数并未被导出 (export) , 因此被认为是私有任务 (private task) 。
4 // 它仍然可以被用在 `series()` 组合中。
5 function clean(cb) {
6   // body omitted
7   cb();
8 }
9
10 // `build` 函数被导出 (export) 了, 因此它是一个公开任务 (public task) , 并且可
    以被 `gulp` 命令直接调用。
11 // 它也仍然可以被用在 `series()` 组合中。
12 function build(cb) {

```

```

13 // body omitted
14 cb();
15 }
16
17 exports.build = build;
18 exports.default = series(clean, build);

```

3.2 组合任务

- 如果需要让任务（task）按顺序执行，请使用 `series()` 方法。

```

1 const { series } = require('gulp');
2
3 function transpile(cb) {
4   // body omitted
5   cb();
6 }
7
8 function bundle(cb) {
9   // body omitted
10  cb();
11 }
12
13 exports.build = series(transpile, bundle);

```

- 对于希望以最大并发来运行的任务（tasks），可以使用 `parallel()` 方法将它们组合起来。

```

1 const { parallel } = require('gulp');
2
3 function javascript(cb) {
4   // body omitted
5   cb();
6 }
7
8 function css(cb) {
9   // body omitted
10  cb();

```

```
11 }  
12  
13 exports.build = parallel(javascript, css);
```

4.异步执行

- 任务完成通知

当从任务（task）中返回 stream、promise、event emitter、child process 或 observable 时,如果任务（task）出错, gulp 将立即结束执行并显示该错误。

- 当使用 `series()` 组合多个任务（task）时, 任何一个任务（task）的错误将导致整个任务组合结束, 并且不会进一步执行其他任务
- 当使用 `parallel()` 组合多个任务（task）时, 一个任务的错误将结束整个任务组合的结束, 但是其他并行的任务（task）可能会执行完, 也可能没有执行完。

4.1 callback

如果任务（task）不返回任何内容, 则必须使用 callback 来指示任务已完成。

callback 将作为唯一一个名为 `cb()` 的参数传递给你的任务（task）。

```
1 function callbackTask(cb) {  
2   // `cb()` should be called by some async work  
3   cb();  
4 }  
5  
6 exports.default = callbackTask;
```

4.2 promise

```
1 exports.callback_error = done => {  
2   console.log('callback task')  
3   done(new Error('task failed'))  
4 }  
5 exports.promise = () => {  
6   console.log('promise task')  
7   return Promise.resolve()  
8 }  
9 exports.promise_error = () => {  
10  console.log('promise_error task')
```

```

11   return Promise.reject(new Error('task failed'))
12 }
13 exports.timeout = time => {
14   return new Promise((resolve, reject) => {
15     setTimeout(resolve, time)
16   })
17 }

```

4.3 stream

```

1 exports.stream = () => {
2   const read = fs.createReadStream('yarn.lock')
3   const write = fs.createWriteStream('a.txt')
4   read.pipe(write)
5   return read
6 }
7 exports.stream2 = done => {
8   const read = fs.createReadStream('yarn.lock')
9   const write = fs.createWriteStream('a.txt')
10  read.pipe(write)
11  read.on('end', () => {
12    done()
13  })
14 }

```

4.4 child process

```

1 const { exec } = require('child_process');
2
3 function childProcessTask() {
4   return exec('date');
5 }
6
7 exports.default = childProcessTask;

```

4.5 event emitter

```

1 const { EventEmitter } = require('events');
2
3 function eventEmitterTask() {
4   const emitter = new EventEmitter();
5   // Emit has to happen async otherwise gulp isn't listening yet
6   setTimeout(() => emitter.emit('finish'), 250);
7   return emitter;
8 }
9
10 exports.default = eventEmitterTask

```

4.6 使用async/await

```

1 const timeout = time => {
2   return new Promise(resolve => {
3     setTimeout(resolve, time)
4   })
5 }
6
7 exports.async = async () => {
8   await timeout(1000)
9   console.log('async task')
10 }

```

5.处理文件

- gulp 暴露了 `src()` 和 `dest()` 方法用于处理计算机上存放的文件
- `src()` 接受 文件路径字符串，并从文件系统中读取文件然后生成一个 Node 流 (stream)
- 它将所有匹配的文件读取到内存中并通过流 (stream) 进行处理。
- 流 (stream) 所提供的主要的 API 是 `.pipe()` 方法，用于连接转换流 (Transform streams) 或可写流 (Writable streams)
- `dest()` 接受一个输出目录作为参数，并且它还会产生一个 Node 流 (stream)，通常作为终止流 (terminator stream)。当它接收到通过管道 (pipeline) 传输的文件时，它会将文件内容及文件属性写入到指定的目录中。

一个案例:

```

1 const {src,dest} = require('gulp')
2 const cleanCss = require('gulp-clean-css')
3 const rename = require('gulp-rename')
4
5 exports.default = () => {
6   return src('src/*.css')
7     .pipe(cleanCss())
8     .pipe(rename({extname: '.min.css'}))
9     .pipe(dest('dist'))
10 }

```

6.特殊字符

6.1 分隔符

建议采用\作为分隔符,不区分操作系统

6.2 *一个星号

在一个字符串片段中匹配任意数量的字符，包括零个匹配。对于匹配单级目录下的文件很有用。

```
1 '*.js' // 匹配当前目录下所有的js 文件
```

6.3 **两个星号

在多个字符串片段中匹配任意数量的字符，包括零个匹配。对于匹配嵌套目录下的文件很有用。

```
1 'scripts/**/*.js' // 匹配scripts目录下,任何层级的js 文件
```

6.4 !取反

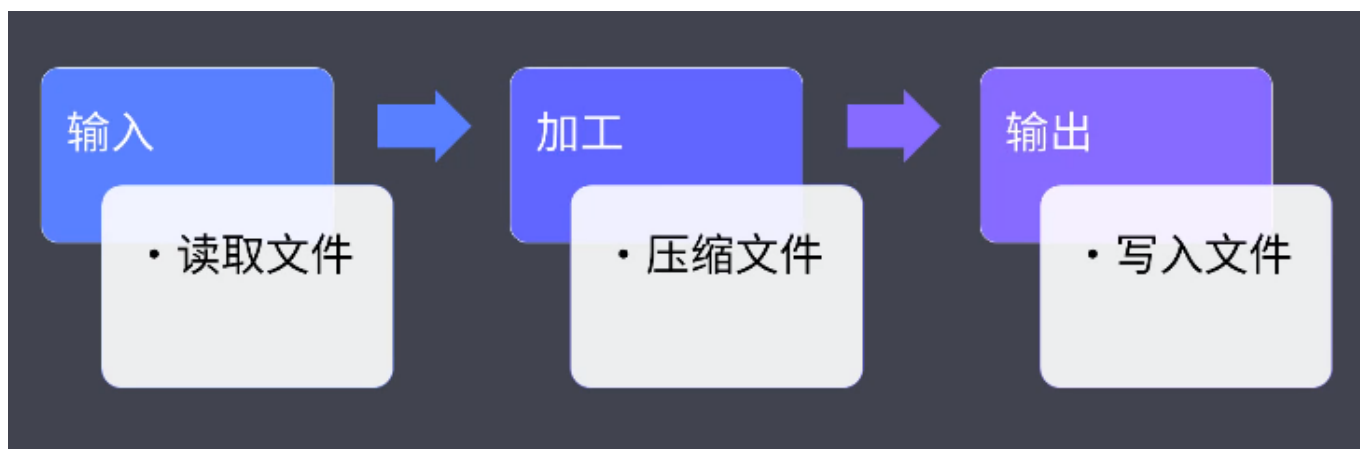
```

1 ['script/**/*.js', '!scripts/vendor/'] // !不匹配这些目录或者文件
2 ['/**/*.js', '!node_modules/'] // 不匹配node_modules下的文件

```


7.gulp的核心原理

构建过程就是将文件读取出来,然后经过一些转换,最后写入到另外一个位置



- 读取流: 通过读取流读取需要转换的文件
- 转换流: 经过转换流的转换逻辑,转换成我们想要的结果
- 写入流: 在通过写入流写入指定的位置
- 构建过程实现了一个构建管道的概念,扩展插件的时候有一个统一的方式

一个压缩css文件的案例

```
1  /*! normalize.css v8.0.1 | MIT License | github.com/necolas/normali
   ze.css */
2
3  /* Document
4     =====
   ===== */
5
6  /**
7   * 1. Correct the line height in all browsers.
8   * 2. Prevent adjustments of font size after orientation changes in
   iOS.
9   */
10
11 html {
12     line-height: 1.15; /* 1 */
13     -webkit-text-size-adjust: 100%; /* 2 */
14 }
15
```

```

16 /* Sections
17  =====
18  ===== */
19 /**
20  * Remove the margin in all browsers.
21  */
22
23 body {
24     margin: 0;
25 }
26
27 /**
28  * Render the `main` element consistently in IE.
29  */
30
31 main {
32     display: block;
33 }
34
35 /**
36  * Correct the font size and margin on `h1` elements within `sectio
37  n` and
38  * `article` contexts in Chrome, Firefox, and Safari.
39  */
40 h1 {
41     font-size: 2em;
42     margin: 0.67em 0;
43 }
44
45 /* Grouping content
46  =====
47  ===== */
48
49 /**
50  * 1. Add the correct box sizing in Firefox.
51  * 2. Show the overflow in Edge and IE.
52  */

```

```

53 hr {
54     box-sizing: content-box; /* 1 */
55     height: 0; /* 1 */
56     overflow: visible; /* 2 */
57 }
58
59 /**
60  * 1. Correct the inheritance and scaling of font size in all brows
61     ers.
62  * 2. Correct the odd `em` font sizing in all browsers.
63  */
64 pre {
65     font-family: monospace, monospace; /* 1 */
66     font-size: 1em; /* 2 */
67 }
68
69 /* Text-level semantics
70     =====
71     ===== */
72 /**
73  * Remove the gray background on active links in IE 10.
74  */
75
76 a {
77     background-color: transparent;
78 }
79
80 /**
81  * 1. Remove the bottom border in Chrome 57-
82  * 2. Add the correct text decoration in Chrome, Edge, IE, Opera, a
83     nd Safari.
84  */
85 abbr[title] {
86     border-bottom: none; /* 1 */
87     text-decoration: underline; /* 2 */
88     text-decoration: underline dotted; /* 2 */
89 }

```

```

90
91 /**
92  * Add the correct font weight in Chrome, Edge, and Safari.
93  */
94
95 b,
96 strong {
97     font-weight: bolder;
98 }
99
100 /**
101  * 1. Correct the inheritance and scaling of font size in all brows
    ers.
102  * 2. Correct the odd `em` font sizing in all browsers.
103  */
104
105 code,
106 kbd,
107 samp {
108     font-family: monospace, monospace; /* 1 */
109     font-size: 1em; /* 2 */
110 }
111
112 /**
113  * Add the correct font size in all browsers.
114  */
115
116 small {
117     font-size: 80%;
118 }
119
120 /**
121  * Prevent `sub` and `sup` elements from affecting the line height
    in
122  * all browsers.
123  */
124
125 sub,
126 sup {
127     font-size: 75%;

```

```

128     line-height: 0;
129     position: relative;
130     vertical-align: baseline;
131 }
132
133 sub {
134     bottom: -0.25em;
135 }
136
137 sup {
138     top: -0.5em;
139 }
140
141 /* Embedded content
142     =====
143     ===== */
144 /**
145  * Remove the border on images inside links in IE 10.
146  */
147
148 img {
149     border-style: none;
150 }
151
152 /* Forms
153     =====
154     ===== */
155 /**
156  * 1. Change the font styles in all browsers.
157  * 2. Remove the margin in Firefox and Safari.
158  */
159
160 button,
161 input,
162 optgroup,
163 select,
164 textarea {
165     font-family: inherit; /* 1 */

```

```

166     font-size: 100%; /* 1 */
167     line-height: 1.15; /* 1 */
168     margin: 0; /* 2 */
169 }
170
171 /**
172  * Show the overflow in IE.
173  * 1. Show the overflow in Edge.
174  */
175
176 button,
177 input { /* 1 */
178     overflow: visible;
179 }
180
181 /**
182  * Remove the inheritance of text transform in Edge, Firefox, and I
183     E.
184  * 1. Remove the inheritance of text transform in Firefox.
185  */
186
187 button,
188 select { /* 1 */
189     text-transform: none;
190 }
191
192 /**
193  * Correct the inability to style clickable types in iOS and Safari
194     i.
195  */
196
197 button,
198 [type="button"],
199 [type="reset"],
200 [type="submit"] {
201     -webkit-appearance: button;
202 }
203
204 /**
205  * Remove the inner border and padding in Firefox.

```

```

204 */
205
206 button::-moz-focus-inner,
207 [type="button"]::-moz-focus-inner,
208 [type="reset"]::-moz-focus-inner,
209 [type="submit"]::-moz-focus-inner {
210     border-style: none;
211     padding: 0;
212 }
213
214 /**
215  * Restore the focus styles unset by the previous rule.
216  */
217
218 button:-moz-focusring,
219 [type="button"]:-moz-focusring,
220 [type="reset"]:-moz-focusring,
221 [type="submit"]:-moz-focusring {
222     outline: 1px dotted ButtonText;
223 }
224
225 /**
226  * Correct the padding in Firefox.
227  */
228
229 fieldset {
230     padding: 0.35em 0.75em 0.625em;
231 }
232
233 /**
234  * 1. Correct the text wrapping in Edge and IE.
235  * 2. Correct the color inheritance from `fieldset` elements in IE.
236  * 3. Remove the padding so developers are not caught out when they
237    zero out
238    `fieldset` elements in all browsers.
239  */
240 legend {
241     box-sizing: border-box; /* 1 */
242     color: inherit; /* 2 */

```

```

243     display: table; /* 1 */
244     max-width: 100%; /* 1 */
245     padding: 0; /* 3 */
246     white-space: normal; /* 1 */
247 }
248
249 /**
250  * Add the correct vertical alignment in Chrome, Firefox, and Oper
    a.
251  */
252
253 progress {
254     vertical-align: baseline;
255 }
256
257 /**
258  * Remove the default vertical scrollbar in IE 10+.
259  */
260
261 textarea {
262     overflow: auto;
263 }
264
265 /**
266  * 1. Add the correct box sizing in IE 10.
267  * 2. Remove the padding in IE 10.
268  */
269
270 [type="checkbox"],
271 [type="radio"] {
272     box-sizing: border-box; /* 1 */
273     padding: 0; /* 2 */
274 }
275
276 /**
277  * Correct the cursor style of increment and decrement buttons in C
    hrome.
278  */
279
280 [type="number"]::-webkit-inner-spin-button,

```



```

281 [type="number"]::-webkit-outer-spin-button {
282     height: auto;
283 }
284
285 /**
286  * 1. Correct the odd appearance in Chrome and Safari.
287  * 2. Correct the outline style in Safari.
288  */
289
290 [type="search"] {
291     -webkit-appearance: textfield; /* 1 */
292     outline-offset: -2px; /* 2 */
293 }
294
295 /**
296  * Remove the inner padding in Chrome and Safari on macOS.
297  */
298
299 [type="search"]::-webkit-search-decoration {
300     -webkit-appearance: none;
301 }
302
303 /**
304  * 1. Correct the inability to style clickable types in iOS and Safari.
305  * 2. Change font properties to `inherit` in Safari.
306  */
307
308 ::-webkit-file-upload-button {
309     -webkit-appearance: button; /* 1 */
310     font: inherit; /* 2 */
311 }
312
313 /* Interactive
314     =====
315     ===== */
316
317 /*
318  * Add the correct display in Edge, IE 10+, and Firefox.
319  */

```

```

319
320 details {
321     display: block;
322 }
323
324 /*
325  * Add the correct display in all browsers.
326  */
327
328 summary {
329     display: list-item;
330 }
331
332 /* Misc
333     =====
334     ===== */
335 /**
336  * Add the correct display in IE 10+.
337  */
338
339 template {
340     display: none;
341 }
342
343 /**
344  * Add the correct display in IE 10.
345  */
346
347 [hidden] {
348     display: none;
349 }

```

gulpfile.js

```

1 const fs = require('fs')
2 const { Transform } = require('stream')
3
4 exports.default = () => {

```

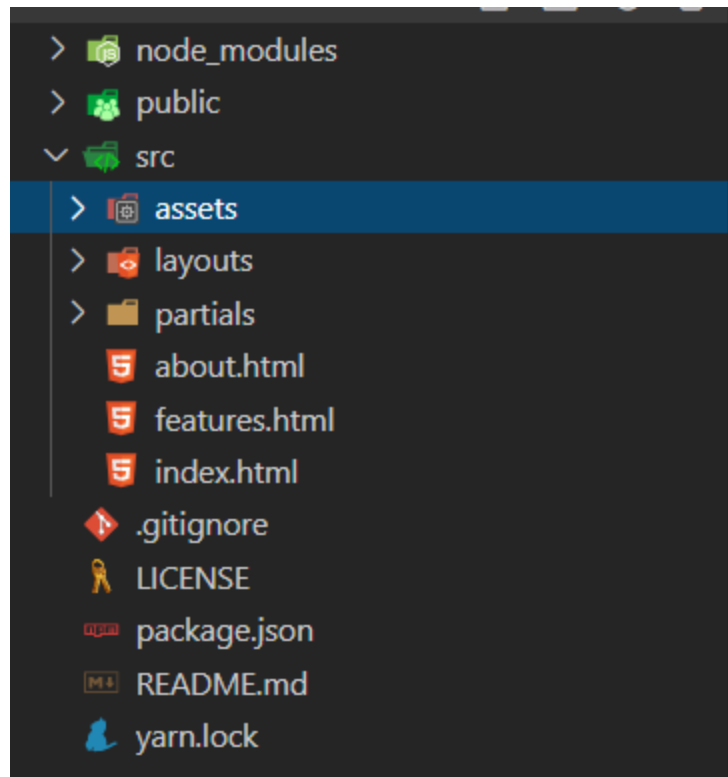
```

5  // 文件读取流
6  const readStream = fs.createReadStream('normalize.css')
7
8  // 文件写入流
9  const writeStream = fs.createWriteStream('normalize.min.css')
10
11 // 文件转换流
12 const transformStream = new Transform({
13   // 核心转换过程
14   transform: (chunk, encoding, callback) => {
15     // chunk 是一个二进制Buffer对象,需要转换成字符串
16     const input = chunk.toString()
17     const output = input.replace(/\s+/g, '').replace(/\/\*\.+?\*\/\n/g, '')
18     // callback是一个错误优先的函数,第一个参数就是接受error的,如果为空则为null
19     // 第二个参数是往后传递的留
20     callback(null, output)
21   }
22 })
23
24 return readStream
25   .pipe(transformStream) // 转换
26   .pipe(writeStream) // 写入
27 }

```

8.一个完整的例子

基础项目结构



8.1 sass的编译

1.安装gulp

```
1 yarn add gulp -D
```

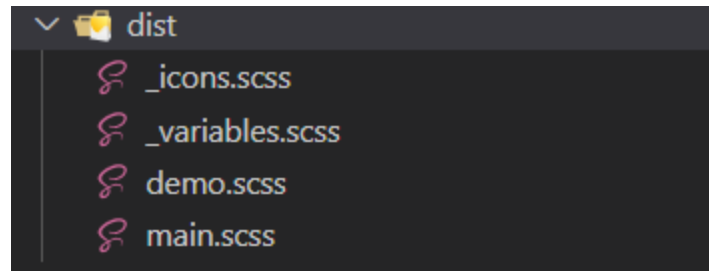
2.项目根目录创建gulpfile.js

3.gulpfile.js中导入src,dest

```
1 const {src ,dest} = require('gulp')
2
3 const style = () => {
4   return src('src/assets/styles/*.scss')
5     .pipe(dest('dist'))
6 }
7
8 module.exports = {
9   style
10 }
```

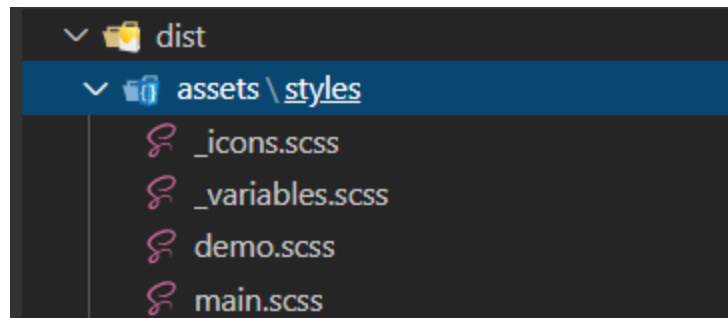
4.运行命令yarn glup style

发现编程成功了但是,目录层级不对



5.修改gulpfile.js文件,src添加base,转换时保存基本路径

```
1 const {src ,dest} = require('gulp')
2
3 const style = () => {
4     return src('src/assets/styles/*.scss',{base: 'src'})
5         .pipe(dest('dist'))
6 }
7
8 module.exports = {
9     style
10 }
```



6.安装转换scss的插件

```
1 yarn add gulp-sass -D
```

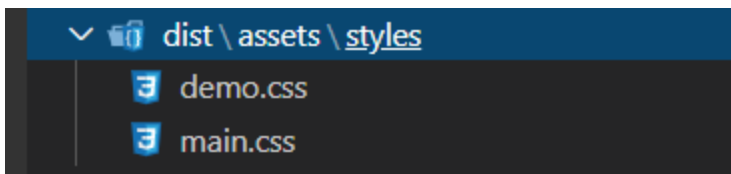
7,添加转换插件

```
1 const {src ,dest} = require('gulp')
2 const sass = require('gulp-sass')
3
```

```

4 const style = () => {
5     return src('src/assets/styles/*.scss',{base: 'src'})
6         .pipe(sass())
7         .pipe(dest('dist'))
8 }
9
10 module.exports = {
11     style
12 }

```



8.问题来了,以下划线开头的文件不会被转换
sass模块还要很多参数,详情请看npm官网

```

1 const {src ,dest} = require('gulp')
2 const sass = require('gulp-sass')
3
4 const style = () => {
5     return src('src/assets/styles/*.scss', { base: 'src' })
6         .pipe(sass({ outputStyle: 'expanded' })) // 编译之后括号展开
7         .pipe(dest('dist'));
8 }
9
10 module.exports = {
11     style
12 }

```

添加参数之前



添加参数之后

```
.icon-aperture:before {
  content: '\e900';
}
```

8.2 处理js

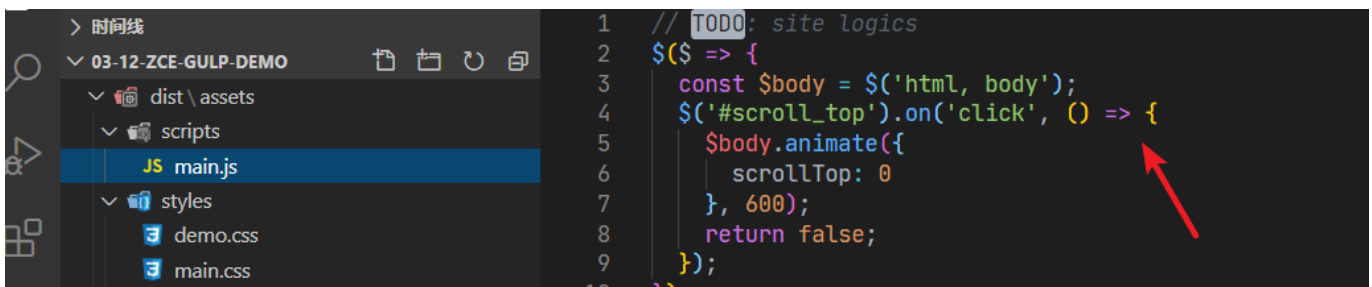
安装依赖

```
1  "gulp-babel": "^8.0.0", // 转换babel插件
2  "@babel/core": "^7.5.5", // babel核心包
3  "@babel/preset-env": "^7.5.5", // babel es6+新语法
4  "gulp-uglify": "^3.0.2", // 代码压缩混淆
```

1.修改gulpfile.js文件

```
1 const babel = require('gulp-babel')
2 const script = () => {
3   return src('src/assets/scripts/*.js', {base: 'src'})
4     .pipe(babel())
5     .pipe(dest('dist'));
6 }
```

2.运行yarn gulp script,但是没有编译成功



3.bable()需要出入参数

```
1 const script = () => {
2   return src('src/assets/scripts/*.js', {base: 'src'})
3     .pipe(babel({
4       presets: ['@babel/preset-env']
5     }))
```

```
6     .pipe(dest('dist'));
7 }
```

8.3 模板编译

安装插件

```
1     "gulp-swig": "^0.9.1",
```

修改gulpfile.js

```
1 const data = {
2     menus: [
3         {
4             name: 'Home',
5             icon: 'aperture',
6             link: 'index.html',
7         },
8         {
9             name: 'Features',
10            link: 'features.html',
11        },
12        {
13            name: 'About',
14            link: 'about.html',
15        },
16        {
17            name: 'Contact',
18            link: '#',
19            children: [
20                {
21                    name: 'Twitter',
22                    link: 'https://twitter.com/w_zce',
23                },
24                {
25                    name: 'About',
26                    link: 'https://weibo.com/zceme',
```



```

27         },
28         {
29             name: 'divider',
30         },
31         {
32             name: 'About',
33             link: 'https://github.com/zce',
34         },
35     ],
36 },
37 ],
38 pkg: require('./package.json'),
39 date: new Date(),
40 };
41 const page = () => {
42     return src('src/**/*.html', {base: 'src'})
43         .pipe(swig({data})) // 指定数据
44         .pipe(dest('dist'))
45 }

```

4.组合任务,并行执行page,script,style

```

1 const {src ,dest, parallel } = require('gulp')
2 const compile = parallel(style,script,page)
3 module.exports = {
4     compile
5 }

```

8.4 字体和图片转换

1.安装插件

```

1     "gulp-imagemin": "^6.1.0",

```

2.修改gulpfile.js

```

1 const imagemin = require('gulp-imagemin');
2 const image = () => {

```

```

3     return src('src/assets/images/**',{base: 'src'})
4     .pipe(imagemin())
5     .pipe(dest('dist'))
6 }
7 const font = () => {
8     return src('src/assets/fonts/**',{base: 'src'})
9     .pipe(imagemin()) // 这里的压缩是指字体文件中的svg
10    .pipe(dest('dist'))
11 }

```

```

→ 03-12-zce-gulp-demo yarn gulp image
yarn run v1.22.4
$ E:\workspace_learn\fed-e-code\part-02\module-01\03-12-zce-gulp-demo\node_modules\.bin\gulp image
[00:23:50] Using gulpfile E:\workspace_learn\fed-e-code\part-02\module-01\03-12-zce-gulp-demo\gulpfile.js
[00:23:50] Starting 'image'...
[00:23:50] gulp-imagemin: Couldn't load default plugin "gifsicle"
[00:23:50] gulp-imagemin: Couldn't load default plugin "optipng"
[00:23:50] gulp-imagemin: Couldn't load default plugin "gifsicle"
[00:23:50] gulp-imagemin: Couldn't load default plugin "optipng"
[00:23:50] gulp-imagemin: Minified 1 image (saved 10.6 kB - 17.1%) 压缩的比例
[00:23:50] Finished 'image' after 653 ms
Done in 1.42s.
→ 03-12-zce-gulp-demo

```

3.合并任务

```

1 const compile = parallel(style,script,page,image,font)
2 module.exports = {
3     compile,
4 };

```

8.5 编译清理和处理其他静态文件

1.移动public文件下的文件到dist

```

1 const extra = () => {
2     return src('public/**', { base: 'public' }).pipe(dest('dist'));
3 }

```

2.编译之前清理文件, 安装插件

```
1 "del": "^5.1.0",
```

```
1 const del = require('del')
2 const clean = () => {
3   return del(['dist'])
4 }
```

3. 需要在编译之前删除文件,所以需要串行执行

```
1 const {src, dest, parallel, series} = require('gulp')
2
3 const build = series(clean, parallel(compile, extra));
```

8.6 自动加载插件

手动加载的插件越来越多,需要解决,帮我们自动加载全部需要的插件

1.安装插件

```
1 "gulp-load-plugins": "^2.0.1",
```

2.使用插件,并删掉关于gulp-xx的插件

```
1 const loadPlugins = require('gulp-load-plugins')
2 const plugins = loadPlugins()
```

8.7 开发服务器

修改文件自动刷新浏览器,热更新功能

1.安装插件

```
1 "browser-sync": "^2.26.7",
```

2.使用插件

```
1 const browserSync = require('browser-sync')
2 // 创建服务器
3 const bs = browserSync.create()
4 const serve = () => {
5   bs.init({
6     server: {
7       baseDir: 'dist', // 网站根目录
8     }
9   })
10 }
11 module.exports = {
12   serve,
13 };
```

3.运行yarn gulp serve

但是html中的css依赖没有修改,所以界面没有样式

```
<!-- build:css assets/styles/vendor.css -->
<link rel="stylesheet" href=
"/node_modules/bootstrap/dist/css/bootstrap.css">
<!-- endbuild -->
<!-- build:css assets/styles/main.css -->
<link rel="stylesheet" href="assets/styles/main.css">
```

- [Home \(current\)](#)
- [Features](#)
- [About](#)
- [Contact](#)
- [Twitter About](#)
- [About](#)

ZCE-GULP-DEMO

[Getting started](#)

4.暂时解决办法

给服务器添加一个路由

```
1 const serve = () => {
2   bs.init({
3     server: {
4       baseDir: 'dist', // 网站根目录
5       routes: {
6         '/node_modules': 'node_modules'
7         // 如果/node_modules请求不到,就会请求根目录的node_modules
8       }
9     }
10  })
11 }
```

5.去掉网站右上角的连接提示

```
1 const serve = () => {
2   bs.init({
3     notify: false, // 去掉提示
4     server: {
5       baseDir: 'dist', // 网站根目录
6       routes: {
7         '/node_modules': 'node_modules'
8         // 如果/node_modules请求不到,就会请求根目录的node_modules
9       }
10     }
11  })
12 }
```

```

9         }
10    }
11  })
12 }

```

6. 设置服务器端口, 自动打开浏览器

```

1  const serve = () => {
2    bs.init({
3      notify: false,
4      port: 3000, //设置端口
5      open: true,
6      server: {
7        baseDir: 'dist', // 网站根目录
8        routes: {
9          '/node_modules': 'node_modules'
10         // 如果/node_modules请求不到,就会请求根目录的node_modules
11       }
12     }
13   })
14 }

```

7. 监听那些文件自动刷新浏览器

```

1  const serve = () => {
2    bs.init({
3      notify: false,
4      port: 3000, //设置端口
5      open: true,
6      files: 'dist/**',
7      server: {
8        baseDir: 'dist', // 网站根目录
9        routes: {
10         '/node_modules': 'node_modules'
11        // 如果/node_modules请求不到,就会请求根目录的node_modules
12      }
13    }
14  })

```

8.8 修改src代码后自动编译

1.添加watch那些文件

```

1  const {src ,dest, parallel,series,watch } = require('gulp')
2  const serve = () => {
3      // 监听某些文件变化,然后执行某个编译任务
4      watch('src/assets/styles/*.scss',style);
5      watch('src/assets/scripts/*.js',script);
6      watch('src/**/*.html',page);
7      watch('src/assets/images/**', image);
8      watch('src/assets/fonts/**', font);
9      watch('public/**', extra);
10     bs.init({
11         notify: false,
12         port: 3000, //设置端口
13         open: true,
14         files: 'dist/**',
15         server: {
16             baseDir: 'dist', // 网站根目录
17             routes: {
18                 '/node_modules': 'node_modules'
19                 // 如果/node_modules请求不到,就会请求根目录的node_modules
20             }
21         }
22     })
23 }
```

2.修改模板引擎不会发生变化,因为swig模板引擎的缓存机制导致页面不会变化

此时需要额外将swig选项中cache设置为false

```

1  const page = () => {
2      return src('src/**/*.html', {base: 'src'})
3          .pipe(plugins.swig({
4              data,
5              cache: false // 缓存设置为false
6          })
7      )
8  }
```

```

6      ))) // 指定数据
7      .pipe(dest('dist'))
8  }

```

3.对静态资源文件不需要监视,因为不是静态资源不是实时改变的
只需要变化的时候重新刷新浏览器,重新请求资源就好了

```

1  const serve = () => {
2      // 监听某些文件变化,然后执行某个编译任务
3      watch('src/assets/styles/*.scss',style);
4      watch('src/assets/scripts/*.js',script);
5      watch('src/**/*.html',page);
6      // watch('src/assets/images/**', image);
7      // watch('src/assets/fonts/**', font);
8      // watch('public/**', extra);
9      // 优化
10     // 只有这些静态资源变化之后,重新请求就可以了,不用运行任务
11     watch([
12         'src/assets/images/**',
13         'src/assets/fonts/**',
14         'public/**'
15     ],bs.reload);
16
17     bs.init({
18         notify: false,
19         port: 3000, //设置端口
20         open: true,
21         files: 'dist/**',
22         server: {
23             // 网站根目录,静态资源优化
24             // 静态资源只有在打包的时候才需要把它打包进dist
25             // 开发阶段就让他放在原位置
26             // 这时候的web服务器会依次按照数组的属性找资源文件
27             baseDir: ['dist', 'src', 'public'], // 指定一个数组
28             routes: {
29                 '/node_modules': 'node_modules',
30                 // 如果/node_modules请求不到,就会请求根目录的node_modules
31             },
32         },

```



```
33     });  
34 }
```

4.设置开发阶段串行任务

```
1 // 只是编译文件,不转移静态资源  
2 const compile = parallel(style,script,page)  
3 // 编译构建的时候先清空,然后在编译,转移静态资源  
4 const build = series(clean, parallel(compile, image, font, extra));  
5 // 开发任务,先编译,然后在启动服务  
6 const develop = series(compile, serve);  
7 module.exports = {  
8     compile,  
9     build,  
10    develop,  
11 };
```

5.其他更新方式

可以在watch后面加上bs.reload

```
1 watch('src/assets/styles/*.scss',style,bs.reload);
```

可以在运行编译后面执行bs.reload

```
1 const style = () => {  
2     return src('src/assets/styles/*.scss', { base: 'src' })  
3         .pipe(plugins.sass({ outputStyle: 'expanded' })) // 编译之后括号  
    展开  
4         .pipe(dest('dist'))  
5         .pipe(bs.reload(stream: true))  
6     };
```

8.9 useref文件引用处理

根据上述所做,已经完成了所有的编译,但是静态资源的引用还没有解决,还是原来的

```

<!-- build:css assets/styles/vendor.css -->
<link rel="stylesheet" href=
"/node_modules/bootstrap/dist/css/bootstrap.css">
<!-- endbuild -->
<!-- build:css assets/styles/main.css -->
<link rel="stylesheet" href="assets/styles/main.css">

```

虽然开发环境做了路由映射,但是真实打包线上是不能使用的

之前构建的时候有构建注释,

使用useref会把引用的资源构建到一个资源文件中,也就是注释的路径

1.安装插件

```
1 "gulp-useref": "^3.1.6"
```

2.使用plugins.if判断是哪个类型文件,然后对其进行针对压缩

```
1 "gulp-if": "^3.0.0",
```

```

1 //      "gulp-useref": "^3.1.6" 已经自动安装了
2 const useref = () => {
3   return (
4     src('dist/*.html', { base: 'dist' })
5     // 构建一个转换流,对刚刚的构建注释,做一个对应的转换
6     // 对文件的合并首先要找到这些文件
7     // 如果这些文件已经在dist了
8     // 对已有些文件还在/node_modules,则设置 .
9     // searchPath就是制定搜索路径的
10    .pipe(plugings.useref({ searchPath: ['dist', '.'] })))
11    // 对引用的三方资源进行压缩打包
12    .pipe(plugings.if(/\.js$/, plugings.uglify()))
13    .pipe(plugings.if(/\.css$/, plugings.cleanCss()))
14    .pipe(
15      plugings.if(
16        /\.html$/,
17        // htmlmin之后压缩属性中的空白字符,其他换行和空格没有压
        // 缩,需要制定参数collapseWhitespace

```

```

18             plugins.htmlmin({
19                 collapseWhitespace: true,
20                 minifyCSS: true, // 压缩行内样式
21                 minifyjs: true, // 压缩行内js
22             })
23         )
24     )
25     .pipe(dest('dist'))
26 );
27 };

```

3.存在一个问题,就是,一边在读,一边在写,而且是同一个dist目录写,同一个文件这时候就会产生文件空白

解决办法:

可以把useref之后的输出的文件存储在别的目录

虽然可以把文件存储到release中,但是这是不合理的,因为静态资源都在dist目录下因为useref是转换之后最终的代码,所以应该吧第一次转换的代码放到临时目录中,然后经过useref之后才转换到dist目录中

修改clean任务,添加清除temp目录

```

1 const clean = () => {
2     return del(['dist', 'temp']);
3 };

```

修改page,style,script,编译后放到临时目录temp

```

1 const style = () => {
2     return src('src/assets/styles/*.scss', { base: 'src' })
3         .pipe(plugins.sass({ outputStyle: 'expanded' })) // 编译之后括号展开
4         .pipe(dest('temp'));
5 };
6 const script = () => {
7     return src('src/assets/scripts/*.js', { base: 'src' })
8         .pipe(
9             plugins.babel({
10                 presets: ['@babel/preset-env'],

```

```

11         })
12     )
13     .pipe(dest('temp'));
14 };
15 const page = () => {
16     return src('src/**/*.html', { base: 'src' })
17         .pipe(
18             plugins.swig({
19                 data,
20                 cache: false,
21             })
22         ) // 指定数据
23     .pipe(dest('temp'));
24 };

```

静态不需要放到临时目录,因为它们只需要转换过一次,所以转换一次放到dist目录即可
修改serve任务的baseDir,添加temp目录

```

1  const serve = () => {
2      // 监听某些文件变化,然后执行某个编译任务
3      watch('src/assets/styles/*.scss', style);
4      watch('src/assets/scripts/*.js', script);
5      watch('src/**/*.html', page);
6      // watch('src/assets/images/**', image);
7      // watch('src/assets/fonts/**', font);
8      // watch('public/**', extra);
9      // 优化
10     // 只有这些静态资源变化之后,重新请求就可以了,不用运行任务
11     watch(['src/assets/images/**', 'src/assets/fonts/**', 'public/*
12         *'], bs.reload);
13     bs.init({
14         notify: false,
15         port: 3000, //设置端口
16         open: true,
17         files: 'dist/**',
18         server: {
19             // 网站根目录,静态资源优化
20             // 静态资源只有在打包的时候才需要把它打包进dist

```

```

21         // 开发阶段就让他放在原位置
22         // 这时候的web服务器会依次按照数组的属性找资源文件
23         baseDir: ['temp', 'src', 'public'], // 指定一个数组
24         routes: {
25             '/node_modules': 'node_modules',
26             // 如果/node_modules请求不到,就会请求根目录的node_modules
27         },
28     },
29 });
30 };

```

修改useref,从temp中取文件进行最后的转换

```

1 //      "gulp-useref": "^3.1.6" 已经自动安装了
2 const useref = () => {
3     return (
4         src('temp/*.html', { base: 'temp' })
5         // 构建一个转换流,对刚刚的构建注释,做一个对应的转换
6         // 对文件的合并首先要找到这些文件
7         // 如果这些文件已经在dist了
8         // 对已有些文件还在/node_modules,则设置 .
9         // searchPath就是制定搜索路径的
10        .pipe(plUGINS.useref({ searchPath: ['temp', '.'] })))
11        // 对引用的三方资源进行压缩打包
12        .pipe(plUGINS.if(/\.js$/, PLUGINS.uglify()))
13        .pipe(plUGINS.if(/\.css$/, PLUGINS.cleanCSS()))
14        .pipe(
15            PLUGINS.if(
16                /\.html$/,
17                // htmlmin之后压缩属性中的空白字符,其他换行和空格没有压
18                // 缩,需要制定参数collapseWhitespace
19                PLUGINS.htmlmin({
20                    collapseWhitespace: true,
21                    minifyCSS: true, // 压缩行内样式
22                    minifyjs: true, // 压缩行内js
23                })
24            )
25        .pipe(dest('dist'))

```

```
26     );  
27 };
```

修改build任务,先进行ompile在进行useref

```
1  
2 // 只是编译文件,不转移静态资源  
3 const compile = parallel(style, script, page);  
4 // 编译构建的时候先清空,然后在编译,转移静态资源  
5 const build = series(clean, parallel(series(compile,useref), image,  
    font, extra));  
6 // 开发任务,先编译,然后在启动服务  
7 const develop = series(compile, serve);  
8 module.exports = {  
9     compile,  
10    build,  
11    develop,  
12    useref,  
13 };
```

8.9 按需导出任务

1.gulpfile导出任务

```
1 module.exports = {  
2     build, // 编译发布  
3     develop, // 开发环境  
4     clean, // 清除所有编译文件和临时目录  
5 };
```

2.修改package.json

```
1 "scripts": {  
2     "clean": "gulp clean",  
3     "build": "gulp build",  
4     "develop": "gulp develop"  
5 },
```

9.封装自动化 workflow

