第六节 Glup

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

```
function defaultTask(cb) {
  // place code for your default task here
  // 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
  cb();
}

exports.default = defaultTask
```

2.5 执行

```
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。

```
const { series } = require('gulp');

// `clean` 函数并未被导出（export），因此被认为是私有任务（private task）。
// 它仍然可以被用在 `series()` 组合中。
function clean(cb) {
  // body omitted
  cb();
}

// `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。
// 它也仍然可以被用在 `series()` 组合中。
function build(cb) {
```

```
  // body omitted
  cb();
}

exports.build = build;
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。

```
const { series } = require('gulp');

function transpile(cb) {
  // body omitted
  cb();
}

function bundle(cb) {
  // body omitted
  cb();
}

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。

```
const { parallel } = require('gulp');

function javascript(cb) {
  // body omitted
  cb();
}

function css(cb) {
  // body omitted
  cb();
```

```
}

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）。

```
function callbackTask(cb) {
  // `cb()` should be called by some async work
  cb();
}

exports.default = callbackTask;
```

4.2 promise

```
exports.callback_error = done => {
  console.log('callback task')
  done(new Error('task failed'))
}
exports.promise = () => {
  console.log('promise task')
  return Promise.resolve()
}
exports.promise_error = () => {
  console.log('promise_error task')
```

```
  return Promise.reject(new Error('task failed'))
}
exports.timeout = time => {
  return new Promise((resolve,reject) => {
    setTimeout(resolve,time)
  })
}
```

4.3 stream

```
exports.stream = () => {
  const read = fs.createReadStream('yarn.lock')
  const write = fs.createWriteStream('a.txt')
  read.pipe(write)
  return read
}
exports.stream2 = done => {
  const read = fs.createReadStream('yarn.lock')
  const write = fs.createWriteStream('a.txt')
  read.pipe(write)
  read.on('end', () => {
    done()
  })
}
```

4.4 child process

```
const { exec } = require('child_process');

function childProcessTask() {
  return exec('date');
}

exports.default = childProcessTask;
```

4.5 event emitter

```
const { EventEmitter } = require('events');

function eventEmitterTask() {
  const emitter = new EventEmitter();
  // Emit has to happen async otherwise gulp isn't listening yet
  setTimeout(() => emitter.emit('finish'), 250);
  return emitter;
}

exports.default = eventEmitterTask
```

4.6 使用async/await

```
const timeout = time => {
  return new Promise(resolve => {
    setTimeout(resolve, time)
  })
}

exports.async = async () => {
  await timeout(1000)
  console.log('async task')
}
```

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:

```
const {src,dest} = require('gulp')
const cleanCss = require('gulp-clean-css')
const rename = require('gulp-rename')

exports.default = () => {
  return src('src/*.css')
    .pipe(cleanCss())
    .pipe(rename({extname: '.min.css'}))
    .pipe(dest('dist'))
}
```

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。

```
'*.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。

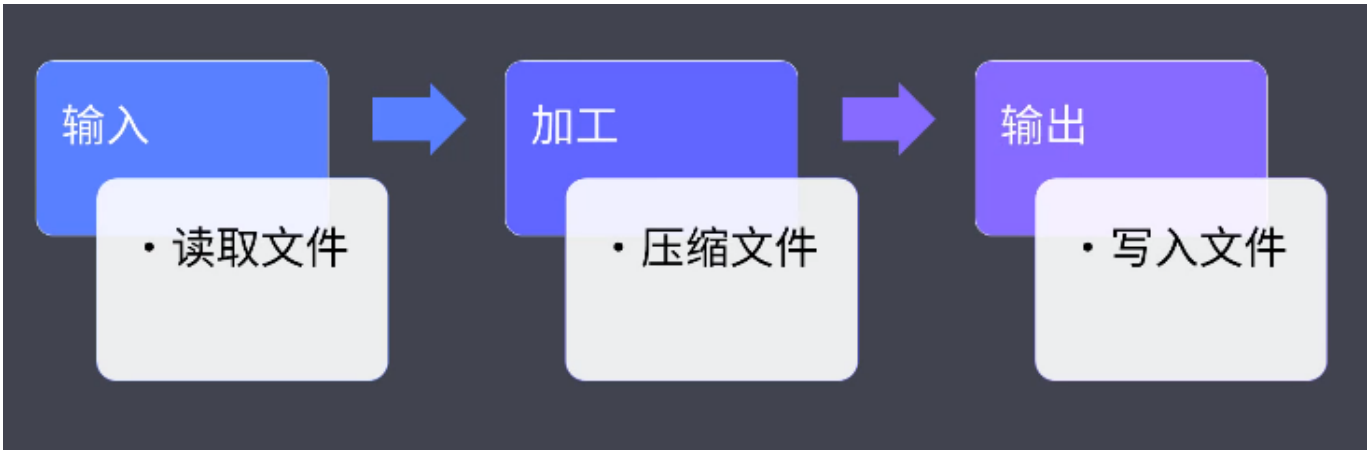
```
'scripts/**/*.js' // 匹配scripts目录下,任何层级的js 文件
```

6.4 !取反

```
['script/**/*.js', '!scripts/vendor/'] // !不匹配这些目录或者文件
['**/*.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

```
/*! normalize.css v8.0.1 | MIT License | github.com/necolas/normalize.css */

/* Document
   ========== */

/**
 * 1. Correct the line height in all browsers.
 * 2. Prevent adjustments of font size after orientation changes in iOS.
 */

html {
  line-height: 1.15; /* 1 */
  -webkit-text-size-adjust: 100%; /* 2 */
}

```

```
/* Sections
   ========================================================================== */

/**
 * Remove the margin in all browsers.
 */

body {
  margin: 0;
}

/**
 * Render the `main` element consistently in IE.
 */

main {
  display: block;
}

/**
 * Correct the font size and margin on `h1` elements within `section` and
 * `article` contexts in Chrome, Firefox, and Safari.
 */

h1 {
  font-size: 2em;
  margin: 0.67em 0;
}

/* Grouping content
   ========================================================================== */

/**
 * 1. Add the correct box sizing in Firefox.
 * 2. Show the overflow in Edge and IE.
 */

```

```
hr {
  box-sizing: content-box; /* 1 */
  height: 0; /* 1 */
  overflow: visible; /* 2 */
}

/**
 * 1. Correct the inheritance and scaling of font size in all browsers.
 * 2. Correct the odd `em` font sizing in all browsers.
 */

pre {
  font-family: monospace, monospace; /* 1 */
  font-size: 1em; /* 2 */
}

/* Text-level semantics
   ========================================================== */

/**
 * Remove the gray background on active links in IE 10.
 */

a {
  background-color: transparent;
}

/**
 * 1. Remove the bottom border in Chrome 57-
 * 2. Add the correct text decoration in Chrome, Edge, IE, Opera, and Safari.
 */

abbr[title] {
  border-bottom: none; /* 1 */
  text-decoration: underline; /* 2 */
  text-decoration: underline dotted; /* 2 */
}
```

```
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%;
```

```
  line-height: 0;
  position: relative;
  vertical-align: baseline;
}

sub {
  bottom: -0.25em;
}

sup {
  top: -0.5em;
}

/* Embedded content
   ========================================================================== */

/**
 * Remove the border on images inside links in IE 10.
 */

img {
  border-style: none;
}

/* Forms
   ========================================================================== */

/**
 * 1. Change the font styles in all browsers.
 * 2. Remove the margin in Firefox and Safari.
 */

button,
input,
optgroup,
select,
textarea {
  font-family: inherit; /* 1 */
```

```
  font-size: 100%; /* 1 */
  line-height: 1.15; /* 1 */
  margin: 0; /* 2 */
}

/**
 * Show the overflow in IE.
 * 1. Show the overflow in Edge.
 */

button,
input { /* 1 */
  overflow: visible;
}

/**
 * Remove the inheritance of text transform in Edge, Firefox, and IE.
 * 1. Remove the inheritance of text transform in Firefox.
 */

button,
select { /* 1 */
  text-transform: none;
}

/**
 * Correct the inability to style clickable types in iOS and Safari.
 */

button,
[type="button"],
[type="reset"],
[type="submit"] {
  -webkit-appearance: button;
}

/**
 * Remove the inner border and padding in Firefox.
```

```
 */

button::-moz-focus-inner,
[type="button"]::-moz-focus-inner,
[type="reset"]::-moz-focus-inner,
[type="submit"]::-moz-focus-inner {
  border-style: none;
  padding: 0;
}

/**
 * Restore the focus styles unset by the previous rule.
 */

button:-moz-focusring,
[type="button"]:-moz-focusring,
[type="reset"]:-moz-focusring,
[type="submit"]:-moz-focusring {
  outline: 1px dotted ButtonText;
}

/**
 * Correct the padding in Firefox.
 */

fieldset {
  padding: 0.35em 0.75em 0.625em;
}

/**
 * 1. Correct the text wrapping in Edge and IE.
 * 2. Correct the color inheritance from `fieldset` elements in IE.
 * 3. Remove the padding so developers are not caught out when they zero out
 *    `fieldset` elements in all browsers.
 */

legend {
  box-sizing: border-box; /* 1 */
  color: inherit; /* 2 */
```

```
  display: table; /* 1 */
  max-width: 100%; /* 1 */
  padding: 0; /* 3 */
  white-space: normal; /* 1 */
}

/**
 * Add the correct vertical alignment in Chrome, Firefox, and Opera.
 */

progress {
  vertical-align: baseline;
}

/**
 * Remove the default vertical scrollbar in IE 10+.
 */

textarea {
  overflow: auto;
}

/**
 * 1. Add the correct box sizing in IE 10.
 * 2. Remove the padding in IE 10.
 */

[type="checkbox"],
[type="radio"] {
  box-sizing: border-box; /* 1 */
  padding: 0; /* 2 */
}

/**
 * Correct the cursor style of increment and decrement buttons in Chrome.
 */

[type="number"]::-webkit-inner-spin-button,
```

```
[type="number"]::-webkit-outer-spin-button {
  height: auto;
}

/**
 * 1. Correct the odd appearance in Chrome and Safari.
 * 2. Correct the outline style in Safari.
 */

[type="search"] {
  -webkit-appearance: textfield; /* 1 */
  outline-offset: -2px; /* 2 */
}

/**
 * Remove the inner padding in Chrome and Safari on macOS.
 */

[type="search"]::-webkit-search-decoration {
  -webkit-appearance: none;
}

/**
 * 1. Correct the inability to style clickable types in iOS and Saf
ari.
 * 2. Change font properties to `inherit` in Safari.
 */

::-webkit-file-upload-button {
  -webkit-appearance: button; /* 1 */
  font: inherit; /* 2 */
}

/* Interactive
   ============================================================
========== */

/*
 * Add the correct display in Edge, IE 10+, and Firefox.
 */
```

```
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 }
```

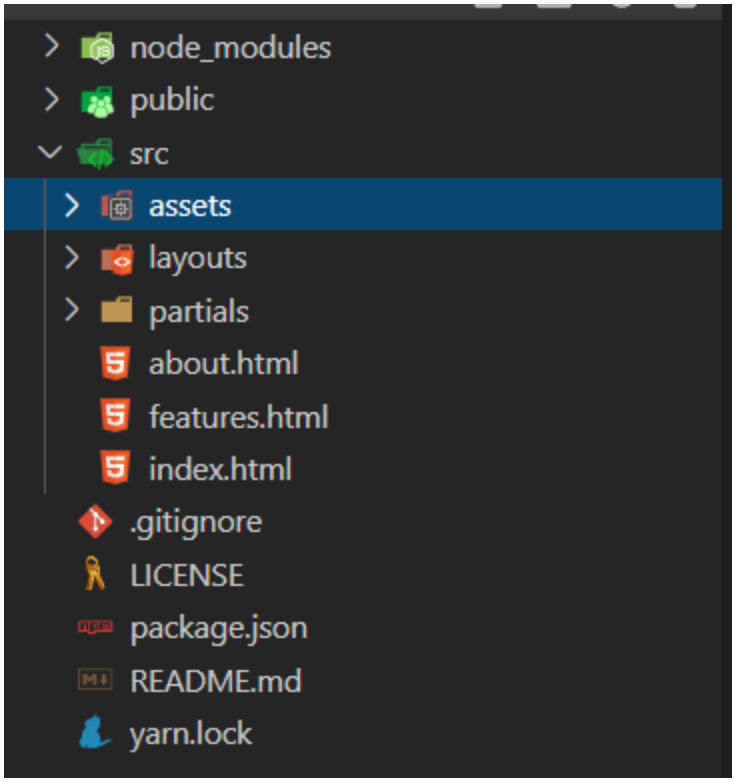
gulpfile.js

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

```
  // 文件读取流
  const readStream = fs.createReadStream('normalize.css')

  // 文件写入流
  const writeStream = fs.createWriteStream('normalize.min.css')

  // 文件转换流
  const transformStream = new Transform({
    // 核心转换过程
    transform: (chunk, encoding, callback) => {
      // chunk 是一个二进制Buffer对象,需要转转换成字符串
      const input = chunk.toString()
      const output = input.replace(/\s+/g, '').replace(/\/\*.+?\*\//g, '')
      // 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
      // 第二个参数是往后传递的留
      callback(null, output)
    }
  })

  return readStream
    .pipe(transformStream) // 转换
    .pipe(writeStream) // 写入
}
```

8.一个完整的例子

基础项目结构



8.1 sass的编译

1.安装gulp

```
yarn add gulp -D
```

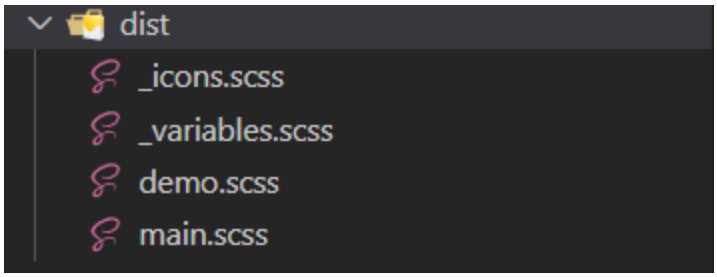
2.项目根目录创建gulpfile.js

3.gulpfile.js中导入src,dest

```
const {src ,dest} = require('gulp')

const style = () => {
    return src('src/assets/styles/*.scss')
        .pipe(dest('dist'))
}

module.exports = {
    style
}
```

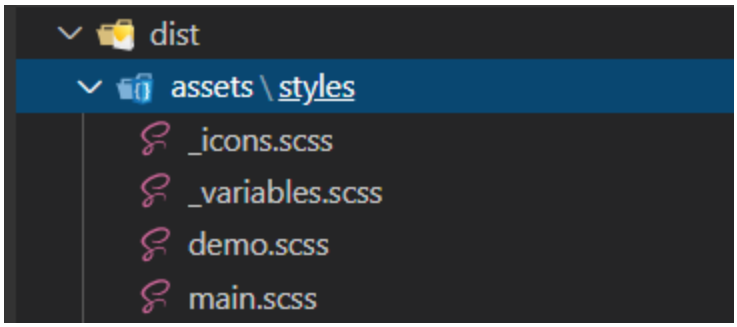
4.运行命令yarn glup style

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

```
const {src ,dest} = require('gulp')

const style = () => {
    return src('src/assets/styles/*.scss',{base: 'src'})
        .pipe(dest('dist'))
}

module.exports = {
    style
}
```



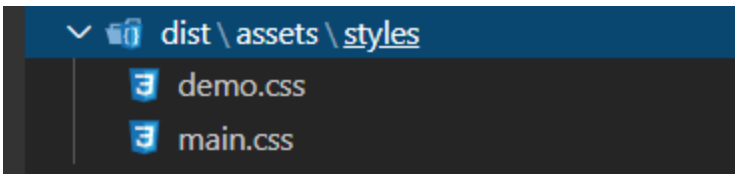
6.安装转换scss的插件

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

7,添加转换插件

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

```

```
const style = () => {
    return src('src/assets/styles/*.scss',{base: 'src'})
        .pipe(sass())
        .pipe(dest('dist'))
}

module.exports = {
    style
}
```



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

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

const style = () => {
    return src('src/assets/styles/*.scss', { base: 'src' })
        .pipe(sass({ outputStyle: 'expanded' })) // 编译之后括号展开
        .pipe(dest('dist'));
}

module.exports = {
    style
}
```

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

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

1.修改gulpfile.js文件

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

2.运行yarn gulp script,但是没有编译成功

3.bable()需要出入参数

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

```
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',
```

```
                },
                {
                    name: 'divider',
                },
                {
                    name: 'About',
                    link: 'https://github.com/zce',
                },
            ],
        },
    ],
    pkg: require('./package.json'),
    date: new Date(),
};
const page = () => {
    return src('src/**/*.html', {base: 'src'})
        .pipe(swig({data})) // 指定数据
        .pipe(dest('dist'))
}
```

4.组合任务,并行执行page,script,style

```
const {src ,dest, parallel } = require('gulp')
const compile = parallel(style,script,page)
module.exports = {
    compile
}
```

8.4 字体和图片转换

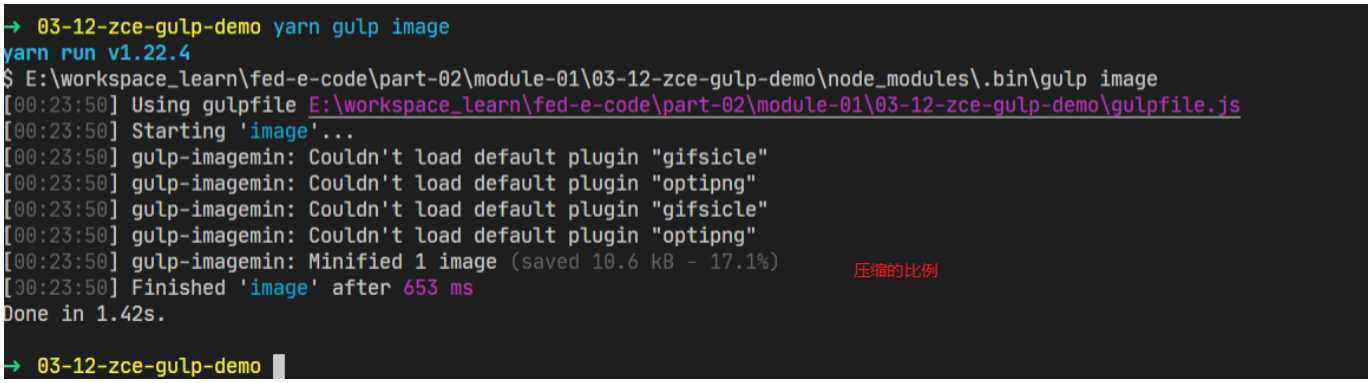
1.安装插件

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

2.修改gulpfile.js

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

```
    return src('src/assets/images/**',{base: 'src'})
        .pipe(imagemin())
        .pipe(dest('dist'))
}
const font = () => {
    return src('src/assets/fonts/**',{base: 'src'})
        .pipe(imagemin()) // 这里的压缩是指字体文件中的svg
        .pipe(dest('dist'))
}
```



3.合并任务

```
const compile = parallel(style,script,page,image,font)
module.exports = {
    compile,
};
```

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

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

2.编译之前清理文件,

安装插件

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

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

3. 需要在编译之前删除文件,所以需要串行执行

```
const {src ,dest, parallel,series } = require('gulp')

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

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

2.使用插件,并删掉关于gulp-xx的插件

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

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

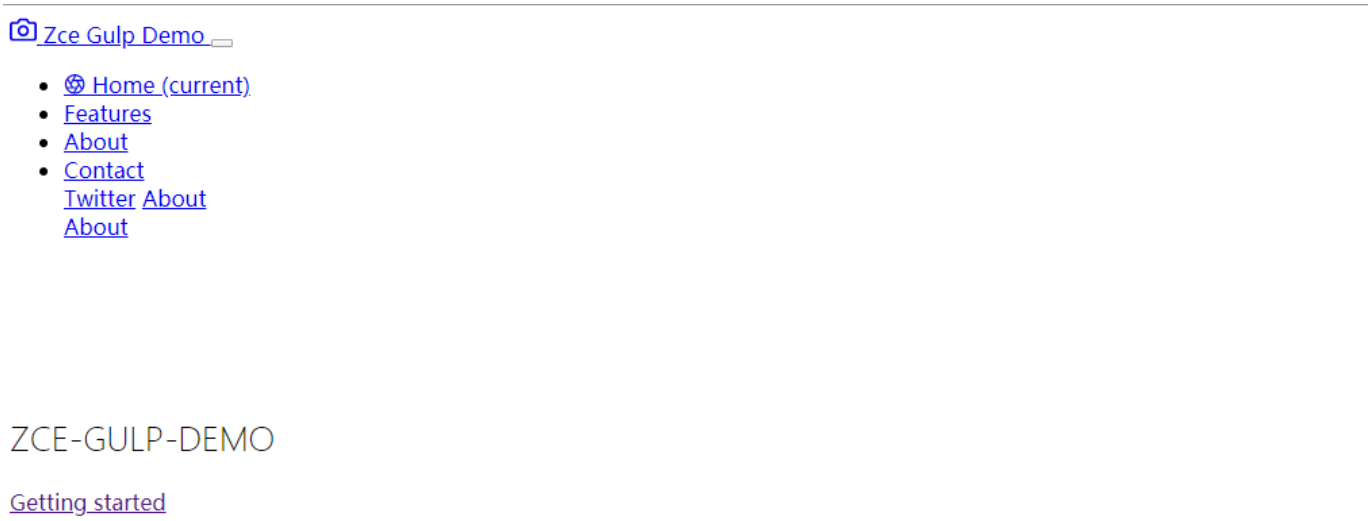
2.使用插件

```
const browserSync = require('browser-sync')
// 创建服务器
const bs = browserSync.create()
const serve = () => {
    bs.init({
        server: {
            baseDir: 'dist', // 网站根目录
        }
    })
}
module.exports = {
    serve,
};
```

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">
```



4.暂时解决办法

给服务器添加一个路由

```
const serve = () => {
    bs.init({
        server: {
            baseDir: 'dist', // 网站根目录
            routes: {
                '/node_modules': 'node_modules'
                // 如果/node_modules请求不到,就会请求根目录的node_modules
            }
        }
    })
}
```

5.去掉网站右上角的连接提示

```
const serve = () => {
    bs.init({
        notify: false, // 去掉提示
        server: {
            baseDir: 'dist', // 网站根目录
            routes: {
                '/node_modules': 'node_modules'
                // 如果/node_modules请求不到,就会请求根目录的node_modules
```

```
            }
        }
    })
}
```

6.设置服务器端口,自动打开浏览器

```
const serve = () => {
    bs.init({
        notify: false,
        port: 3000, //设置端口
        open: true,
        server: {
            baseDir: 'dist', // 网站根目录
            routes: {
                '/node_modules': 'node_modules'
                // 如果/node_modules请求不到,就会请求根目录的node_modules
            }
        }
    })
}
```

7.监听那些文件自动刷新浏览器

```
const serve = () => {
    bs.init({
        notify: false,
        port: 3000, //设置端口
        open: true,
        files: 'dist/**',
        server: {
            baseDir: 'dist', // 网站根目录
            routes: {
                '/node_modules': 'node_modules'
                // 如果/node_modules请求不到,就会请求根目录的node_modules
            }
        }
    })
```

```
}
```

8.8 修改src代码后自动编译

1.添加watch那些文件

```
const {src ,dest, parallel,series,watch } = require('gulp')
const serve = () => {
    // 监听某些文件变化,然后执行某个编译任务
    watch('src/assets/styles/*.scss',style);
    watch('src/assets/scripts/*.js',script);
    watch('src/**/*.html',page);
    watch('src/assets/images/**', image);
    watch('src/assets/fonts/**', font);
    watch('public/**', extra);
    bs.init({
        notify: false,
        port: 3000, //设置端口
        open: true,
        files: 'dist/**',
        server: {
            baseDir: 'dist', // 网站根目录
            routes: {
                '/node_modules': 'node_modules'
                // 如果/node_modules请求不到,就会请求根目录的node_modules
            }
        }
    })
}
```

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

```
const page = () => {
    return src('src/**/*.html', {base: 'src'})
        .pipe(plugins.swig({
            data,
            cache: false // 缓存设置为false
```

```
        })) // 指定数据
        .pipe(dest('dist'))
}
```

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

```
const serve = () => {
    // 监听某些文件变化,然后执行某个编译任务
    watch('src/assets/styles/*.scss',style);
    watch('src/assets/scripts/*.js',script);
    watch('src/**/*.html',page);
    // watch('src/assets/images/**', image);
    // watch('src/assets/fonts/**', font);
    // watch('public/**', extra);
    // 优化
    // 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
    watch([
        'src/assets/images/**',
        'src/assets/fonts/**',
        'public/**'
    ],bs.reload);

    bs.init({
        notify: false,
        port: 3000, //设置端口
        open: true,
        files: 'dist/**',
        server: {
            // 网站根目录,静态资源优化
            // 静态资源只有在打包的时候才需要把它打包进dist
            // 开发阶段就让他放在原位置
            // 这时候的web服务器会依次按照数组的属性找资源文件
            baseDir: ['dist', 'src', 'public'], // 指定一个数组
            routes: {
                '/node_modules': 'node_modules',
                // 如果/node_modules请求不到,就会请求根目录的node_modules
            },
        },
```

```
    });
}
```

4.设置开发阶段串行任务

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

5.其他更新方式
可以在watch后面加上bs.reload

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

可以在运行编译后面执行bs.reload

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

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(plugins.useref({ searchPath: ['dist', '.'] }))
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
   缩,需要制定参数collapseWhitespace
```

```
                    plugins.htmlmin({
                        collapseWhitespace: true,
                        minifyCSS: true, // 压缩行内样式
                        minifyjs: true, // 压缩行内js
                    })
                )
            )
            .pipe(dest('dist'))
    );
};
```

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

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

修改page,style,script,编译后放到临时目录temp

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

```
            })
        )
        .pipe(dest('temp'));
};
const page = () => {
    return src('src/**/*.html', { base: 'src' })
        .pipe(
            plugins.swig({
                data,
                cache: false,
            })
        ) // 指定数据
        .pipe(dest('temp'));
};
```

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

```
const serve = () => {
    // 监听某些文件变化,然后执行某个编译任务
    watch('src/assets/styles/*.scss', style);
    watch('src/assets/scripts/*.js', script);
    watch('src/**/*.html', page);
    // watch('src/assets/images/**', image);
    // watch('src/assets/fonts/**', font);
    // watch('public/**', extra);
    // 优化
    // 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
    watch(['src/assets/images/**', 'src/assets/fonts/**', 'public/*
*'], bs.reload);

    bs.init({
        notify: false,
        port: 3000, //设置端口
        open: true,
        files: 'dist/**',
        server: {
            // 网站根目录,静态资源优化
            // 静态资源只有在打包的时候才需要把它打包进dist
```

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

修改useref,从temp中取文件进行最后的转换

```
//     "gulp-useref": "^3.1.6" 已经自动安装了
const useref = () => {
    return (
        src('temp/*.html', { base: 'temp' })
            // 构建一个转换流,对刚刚的构建注释,做一个对应的转换
            // 对文件的合并首先要找到这些文件
            // 如果这些文件已经在dist了
            // 对已有些文件还在/node_modules,则设置 .
            // searchPath就是制定搜索路径的
            .pipe(plugins.useref({ searchPath: ['temp', '.'] }))
            // 对引用的三方资源进行压缩打包
            .pipe(plugins.if(/\.js$/, plugins.uglify()))
            .pipe(plugins.if(/\.css$/, plugins.cleanCss()))
            .pipe(
                plugins.if(
                    /\.html$/,
                    // 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
                    plugins.htmlmin({
                        collapseWhitespace: true,
                        minifyCSS: true, // 压缩行内样式
                        minifyjs: true, // 压缩行内js
                    })
                )
            )
            .pipe(dest('dist'))
```

```
    );
};
```

修改build任务,先进行ompile在进行useref

```

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

8.9 按需导出任务

1.gulpfile导出任务

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

2.修改package.json

```
  "scripts": {
    "clean": "gulp clean",
    "build": "gulp build",
    "develop": "gulp develop"
  },
```

9.封装自动化工作流