

GULP

- `gulp` 是一个项目开发的 自动化打包构建工具
- 基于 `node` 环境来运行的

什么是自动化打包构建工具

- 比如
 - 我们在开发的过程中，会写到 `js` 文件，`css` 文件，等等
 - 我们的项目如果想上线，那么一定要体积小一点，文件大小越小越好
 - 而我们在写 `js` 文件的时候，会有很多 换行/空格 之类的东西
 - 这些 换行/空格 都是占文件体积的一部分
 - 那么我们在上线之前就要把这些 换行/空格 尽可能的删除掉
 - 我们又不能一个文件一个文件的去删除
 - 就要用到一个自动化工具来帮助我们把这些多余的东西干掉
- 这个就是自动化工具的意义
- 常见的自动化打包构建工具
 - `gulp`
 - `webpack`

安装 GULP

- `gulp` 是一个依赖于 `node` 的环境工具
- 所以我们需要先安装一个 全局 `gulp` 依赖
- 直接使用 `npm` 去安装就可以了

1. # 使用 `npm` 安装全局依赖 `gulp`
2. # 我们这里安装一个 3.9.1 版本的就好了
3. \$ `npm install --global gulp@3.9.1`

- 等待安装完毕就好了
- 这个全局环境一个电脑安装一次就好了
- 还是照例检查一下是否安装成功

1. \$ `gulp --version`

使用 GULP

- 安装完毕以后，我们就可以使用 GULP 对我们的项目进行自动化构建了
- 首先我们要有一个项目

1. - gulp_demo 项目文件夹
2. - src 项目源码
3. + css css 文件夹
4. + js js 文件夹
5. + pages html 文件夹
6. + sass sass 文件夹
7. + lib 第三方文件夹
8. + static 静态资源文件夹

- 目录结构不一定都是这个样子
- 但是最好是一个便于管理的文件夹目录结构
- 因为是一个项目了，最好使用 `npm` 来帮我们管理一下
 - 这样可以记录我们的下载使用了那些依赖
- 所以在项目文件夹 `gulp-demo` 里面执行一个 `npm` 初始化

1. \$ cd gulp_demo
2. \$ npm init -y

- 执行完毕之后，再来看一下我们的项目目录

1. - gulp_demo
2. + src
3. + package.json

项目 GULP 配置

- 我们之前已经安装过 `gulp` 全局依赖了
- 但是每一个项目都要在安装一次 `gulp` 的项目依赖
- 因为每一个项目的打包构建规则都不一样，所以不能全都配置成一个
- 所以我们要在项目里面再次进行 `gulp` 安装

1. `$ cd gulp_demo`
2. `$ npm install -D gulp@3.9.1`

- 项目中的 `gulp` 依赖要和全局 `gulp` 环境保持版本一致

- 接下来就是对这个项目进行打包构建的配置
- `gulp` 的使用，要在项目目录下新建一个 `gulpfile.js`
- 在这个 `gulpfile.js` 文件里面进行配置
- 然后使用 `gulp` 进行构建的时候就会按照 `gulpfile.js` 文件里面的规则进行打包构建
- 再来看一下我们的目录结构

1. `- gulp_demo`
2. `+ node_modules` 依赖包目录
3. `+ src` 项目源码
4. `+ gulpfile.js` gulp 配置文件
5. `+ package-lock.json` 依赖下载版本 json 文件
6. `+ package.json` 项目管理 json 文件

- 接下来我们就是在 `gulpfile.js` 文件里面进行配置，让我们的打包构建可以生效

打包 CSS 文件

- 我们从简单的内容开始，先来打包构建 `css` 文件
- 其实就是在 `gulpfile.js` 里面进行一些配置
- 第一个事情就是引入 `gulp`

1. `// 我是 gulpfile.js 文件`
- 2.
3. `// 1. 引入 gulp`
4. `const gulp = require('gulp')`

- `gulp` 是基于任务来完成构建的
- 所以我们要创建一个打包 `css` 的任务

```
```javascript
// 我是 gulpfile.js 文件

// 1. 引入 gulp
.
```

```
const gulp = require('gulp')

// 2. 创建一个 css 的任务
// gulp.task() 是用来创建任务的
// 参数一： 任务名称
// 参数二： 一个函数（这个任务要做什么事情）
gulp.task('css' , function () {

 1.
 2. - 有了任务以后，我们就要在 `css` 这个任务里面写一些这个任务要做的事情了
 3.
 4. - 我们要把 `./src/css/所有css文件` 都进行压缩处理
 5.
 6. - 这个时候我们自己完成不了，就需要借助一个第三方依赖
 7.
 8. - `npm i -D gulp-cssmin`
 9.
 10. - 下载完毕以后，去文件中进行配置
 11.
 12. ```javascript
 13. // 我是 gulpfile.js 文件
 14.
 15. // 1. 引入 gulp
 16. const gulp = require('gulp')
 17.
 18. // 2. 引入 gulp-cssmin
 19. const cssmin = require('gulp-cssmin')
 20.
 21.
 22. // 2. 创建一个 css 的任务
 23. gulp.task('css', function () {
 24. return gulp
 25. .src('./src/css/**') // 对哪些文件进行操作
 26. .pipe(cssmin()) // 都做什么，这里做的就是进行 css 压缩
 27. .pipe(gulp.dest('./dist/css')) // 把压缩完毕的文件放在 dist 文件夹下的 css 文件夹
 28. })
 • gulp.src() 是指找到那些文件对其操作
 • gulp.pipe() 是指要做什么
 • gulp.dest() 是指输出到哪个目录下，如果没有这个目录存在会自动创建这个目录
 • 所以上面哪个 css 任务的意思就是
```

- 把 `./src/css/` 目录下的所有文件
- 进行压缩
- 压缩完毕后放在 `./dist/` 下的 `css/` 文件夹下
- 接下来，就是执行一下这个叫做 `css` 的任务就行了
- 直接在控制台使用指令
  1. `#` 表示运行 `gulpfile.js` 配置文件中的 `css` 任务
  2. `$ gulp css`
    - 执行完毕以后，就会在 `gulp_demo` 目录下生成一个 `dist/` 文件夹
    - 里面就有我们压缩好的 `css` 文件

## 自动添加前缀

- 我们希望在 `css` 压缩之前，能帮我自动把需要前缀的属性 自动把前缀添加上 就好了
- 这个是可以做到的
- 我们又需要一个依赖了
  - `npm i -D gulp-autoprefixer`
- 安装完还是去到文件中进行配置

```
```javascript
// 我是 gulpfile.js 文件

// 1. 引入 gulp
const gulp = require( 'gulp' )

// 2. 引入 gulp-cssmin
const cssmin = require( 'gulp-cssmin' )

// 2-1. 引入 gulp-autoprefixer
const autoPrefixer = require( 'gulp-autoprefixer' )

// 2. 创建一个 css 的任务
gulp.task( 'css' , function () {
  return gulp
    .src( './src/css/**' )
    .pipe(autoPrefixer({
      browsers: [ 'last 2 versions' ]
    }))
    .pipe(cssmin())
    .pipe(gulp.dest('./dist/css'))
})
```
```

```
.pipe(cssmin())
.pipe(gulp.dest('./dist/css'))
})
```

```
1.
2. - 因为添加前缀需要在压缩之前添加
3. - 所以我们就直接在任务里面多做一个事情就行了
4.
5.
6.
7. ### 打包 SASS 文件
8.
9. - 接下来我们解决一下 `sass` 的问题
10.
11. - 因为有的时候我们开发要使用 `sass` 所以我们要解决一下 `sass` 的问题
12.
13. - 其实打包 `sass` 和 `css` 差不多，只不过先要把 `sass` 解析成 `css`
14.
15. 1. 把 `sass` 解析成 `css`
16. 2. 自动添加前缀
17. 3. 压缩一下
18. 4. 放到对应的文件夹中
19.
20. - 使用 `gulp` 解析 `sass` 文件需要用到一个依赖
21.
22. - `npm i -D gulp-sass`
23.
24. - 然后我们就去到配置文件里面进行配置就好了
25.
26. ```javascript
27. // 我是 gulpfile.js 文件
28.
29. // 1. 引入 gulp
30. const gulp = require('gulp')
31.
32. // 2. 引入 gulp-cssmin
33. const cssmin = require('gulp-cssmin')
34.
35. // 2-1. 引入 gulp-autoprefixer
36. const autoPrefixer = require('gulp-autoprefixer')
37.
38. // 3. 引入 gulp-sass
```

```
38. // 2-2. 引入 gulp-sass
39. const sass = require('gulp-sass')
40.
41.
42. // 2-1. 创建一个 css 的任务
43. gulp.task('css', function () {
44. return gulp
45. .src('./src/css/**')
46. .pipe(autoPrefixer({
47. browsers: ['last 2 versions']
48. }))
49. .pipe(cssmin())
50. .pipe(gulp.dest('./dist/css'))
51. })
52.
53. // 2-2. 创建一个 sass 任务
54. gulp.task('sass', function () {
55. return gulp
56. .src('./src/sass/**')
57. .pipe(sass())
58. .pipe(autoPrefixer())
59. .pipe(cssmin())
60. .pipe(gulp.dest('./dist/css'))
61. })
```

- 我们的 `sass` 文件编译完以后也是 `css` 文件，所以就也放在 `css` 文件夹下就好了
- 就是要注意一下别和本来 `css` 文件夹下的文件重名就好了
- 然后就可以去命令行执行 `sass` 这个任务了

1. # 执行 sass 任务
2. \$ gulp sass

## 打包 JS 文件

- 接下来就是打包一下 `js` 文件了
- 其实套路是一样的
- 先来做最简单的操作
- 压缩一下 `js` 文件
- 需要用到一个依赖

- `npm i -D gulp-uglify`

- 然后依旧是去到配置文件中配置

```
```javascript
// 我是 gulpfile.js 文件

// 1. 引入 gulp
const gulp = require( 'gulp' )

// 2-1. 引入 gulp-cssmin
const cssmin = require( 'gulp-cssmin' )

// 2-2. 引入 gulp-autoprefixer
const autoPrefixer = require( 'gulp-autoprefixer' )

// 2-3. 引入 gulp-sass
const sass = require( 'gulp-sass' )

// 3-1. 引入 gulp-uglify
const uglify = require( 'gulp-uglify' )

// 2-1. 创建一个 css 的任务
gulp.task( 'css' , function () {
return gulp
.src( './src/css/**' )
.pipe(autoPrefixer({
browsers: [ 'last 2 versions' ]
}))
.pipe(cssmin())
.pipe(gulp.dest( './dist/css' ))
})

// 2-2. 创建一个 sass 任务
gulp.task( 'sass' , function () {
return gulp
.src( './src/sass/**' )
.pipe(sass())
.pipe(autoPrefixer())
.pipe(cssmin())
.pipe(gulp.dest( './dist/css' ))
})
```
```



```
// 3. 创建一个 js 任务
gulp.task('js' , function () {
return gulp
.src('./src/js/**')
.pipe(uglify())
.pipe(gulp.dest('./dist/js'))
})
```

```
1.
2. - 然后我们去命令行执行 `js` 这个任务
3.
4. ```shell
5. # 执行 js 任务
6. $ gulp js
```

## 编译 ES6 语法

- 之前我们学习 `es6` 的时候就说过
- 很多浏览器不是很认识 `es6` 的语法
- 我们要把他编译成 `es5` 的语法
- 所以我们在打包 `js` 文件的时候，就要转换一下
- 我们依旧是使用依赖来完成，只不过 `es6` 转 `es5` 需要三个依赖
  - `npm i -D gulp-babel@7.0.1`
  - `npm i -D babel-core`
  - `npm i -D babel-preset-es2015`
- 然后我们就去配置文件里面进行配置就可以了

```
```javascript
// 我是 gulpfile.js 文件

// 1. 引入 gulp
const gulp = require( 'gulp' )

// 2-1. 引入 gulp-cssmin
const cssmin = require( 'gulp-cssmin' )

// 2-2. 引入 gulp-autoprefixer
const autoPrefixer = require( 'gulp-autoprefixer' )
```

```
const autoprefixer = require( 'gulp-autoprefixer' );

// 2-3. 引入 gulp-sass
const sass = require( 'gulp-sass' )

// 3-1. 引入 gulp-uglify
const uglify = require( 'gulp-uglify' )

// 3-2. 引入 gulp-babel
// es6 转 es5 虽然需要下载三个依赖，但是只需要引入一个 gulp-babel 就可以了
const babel = require( 'gulp-babel' )

// 2-1. 创建一个 css 的任务
gulp.task( 'css' , function () {
return gulp
.src( './src/css/**' )
.pipe(autoprefixer({
browsers: [ 'last 2 versions' ]
}))
.pipe(cssmin())
.pipe(gulp.dest( './dist/css' ))
})

// 2-2. 创建一个 sass 任务
gulp.task( 'sass' , function () {
return gulp
.src( './src/sass/**' )
.pipe(sass())
.pipe(autoprefixer())
.pipe(cssmin())
.pipe(gulp.dest( './dist/css' ))
})

// 3. 创建一个 js 任务
gulp.task( 'js' , function () {
return gulp
.src( './src/js/**' )
.pipe(babel({
presets: [ 'es2015' ]
}))
.pipe(uglify())
.pipe(gulp.dest( './dist/js' ))
})
```

```
1.
2.   - 引入以后，只要在压缩之前执行一下转码就可以了
3.   - 注意要传递一个参数
4.
5.   - 然后再次执行 `js` 任务就可以了
6.
7.   ``shell
8.   # 执行 js 任务
9.   $ gulp js
```

打包 HTML 文件

- 接下来就要把 `html` 文件解决一下了
- 还是一个套路
- 压缩 `html` 文件需要一个依赖
 - `npm i -D gulp-htmlmin`
- 下载好以后去到配置文件中配置

```
````javascript
// 我是 gulpfile.js 文件

// 1. 引入 gulp
const gulp = require('gulp')

// 2-1. 引入 gulp-cssmin
const cssmin = require('gulp-cssmin')

// 2-2. 引入 gulp-autoprefixer
const autoPrefixer = require('gulp-autoprefixer')

// 2-3. 引入 gulp-sass
const sass = require('gulp-sass')

// 3-1. 引入 gulp-uglify
const uglify = require('gulp-uglify')

// 3-2. 引入 gulp-babel
// es6 转 es5 虽然需要下载三个依赖，但是只需要引入一个 gulp-babel 就可以了
const babel = require('gulp-babel')
```

```
// 4. 引入 gulp-htmlmin
const htmlmin = require('gulp-htmlmin')

// 2-1. 创建一个 css 的任务
gulp.task('css' , function () {
return gulp
.src('./src/css/**')
.pipe(autoPrefixer({
browsers: ['last 2 versions']
}))
.pipe(cssmin())
.pipe(gulp.dest('./dist/css'))
})

// 2-2. 创建一个 sass 任务
gulp.task('sass' , function () {
return gulp
.src('./src/sass/**')
.pipe(sass())
.pipe(autoPrefixer())
.pipe(cssmin())
.pipe(gulp.dest('./dist/css'))
})

// 3. 创建一个 js 任务
gulp.task('js' , function () {
return gulp
.src('./src/js/**')
.pipe(babel({
presets: ['es2015']
}))
.pipe(uglify())
.pipe(gulp.dest('./dist/js'))
})

// 4. 创建一个 html 任务
gulp.task('html' , function () {
return gulp
.src('./src/pages/**')
.pipe(htmlmin({
removeEmptyAttributes: true, // 移出所有空属性
```

```
collapseWhitespace: true // 压缩 html
}))
.pipe(gulp.dest('./dist/pages'))
})
```

- 1.
2. - 这样就配置完毕了
- 3.
4. - 接下来就是去命令行运行一下 `html` 任务
- 5.
6. ```shell
7. # 运行 html 任务
8. \$ gulp html

## 处理 LIB 和 STATIC 文件

- 接下来我们就是处理 `lib` 和 `static` 里面的文件了
- 因为这些都是第三方的文件和一些图片之类的信息
- 不需要压缩处理，只要给我转到 `dist` 文件夹下就可以了
- 所以不需要依赖，直接处理就行

```
```javascript
// 我是 gulpfile.js 文件

// 1. 引入 gulp
const gulp = require( 'gulp' )

// 2-1. 引入 gulp-cssmin
const cssmin = require( 'gulp-cssmin' )

// 2-2. 引入 gulp-autoprefixer
const autoPrefixer = require( 'gulp-autoprefixer' )

// 2-3. 引入 gulp-sass
const sass = require( 'gulp-sass' )

// 3-1. 引入 gulp-uglify
const uglify = require( 'gulp-uglify' )

// 3-2. 引入 gulp-babel
// es6 转 es5 虽然需要下载三个依赖，但是只需要引入一个 gulp-babel 就可以了
```

```
const babel = require( 'gulp-babel' )

// 4. 引入 gulp-htmlmin
const htmlmin = require( 'gulp-htmlmin' )

// 2-1. 创建一个 css 的任务
gulp.task( 'css' , function () {
return gulp
.src( './src/css/**' )
.pipe(autoPrefixer({
browsers: [ 'last 2 versions' ]
}))
.pipe(cssmin())
.pipe(gulp.dest( './dist/css' ))
})

// 2-2. 创建一个 sass 任务
gulp.task( 'sass' , function () {
return gulp
.src( './src/sass/**' )
.pipe(sass())
.pipe(autoPrefixer())
.pipe(cssmin())
.pipe(gulp.dest( './dist/css' ))
})

// 3. 创建一个 js 任务
gulp.task( 'js' , function () {
return gulp
.src( './src/js/**' )
.pipe(babel({
presets: [ 'es2015' ]
}))
.pipe(uglify())
.pipe(gulp.dest( './dist/js' ))
})

// 4. 创建一个 html 任务
gulp.task( 'html' , function () {
return gulp
.src( './src/pages/**' )
.pipe(htmlmin({
```

```
removeEmptyAttributes: true, // 移出所有空属性
collapseWhitespace: true // 压缩 html
}))
.pipe(gulp.dest( './dist/pages' ))
})
```

// 5. 创建一个 lib 任务

```
gulp.task( 'lib' , function () {
return gulp
.src( './src/lib/**' )
.pipe(gulp.dest( './dist/lib' ))
})
```

// 6. 创建一个 static 任务

```
gulp.task( 'static' , function () {
return gulp
.src( './src/static/**' )
.pipe(gulp.dest( './dist/static' ))
})
```

```
1.
2. - 接下来就是去命令行执行 `lib` 任务和 `static` 任务
3.
4. ```shell
5. # 执行 lib 任务
6. $ gulp lib
7.
8. # 执行 static 任务
9. $gulp static
```

批量执行任务

- 我们的几个和文件相关的任务都配置完毕了
- 但是我们不能在开发过程中一个任务一个任务的去执行
- 不方便也不是很只能
- 所以我们要统一执行任务
- 这个时候就需要进行一个配置，让几个任务一起执行一下
- 这个不需要依赖，我们只需要配置一个叫做 `default` 的任务

- `gulp` 在运行的时候会默认执行 `default` 任务

```
```javascript
// 我是 gulpfile.js 文件

// 1. 引入 gulp
const gulp = require('gulp')

// 2-1. 引入 gulp-cssmin
const cssmin = require('gulp-cssmin')

// 2-2. 引入 gulp-autoprefixer
const autoPrefixer = require('gulp-autoprefixer')

// 2-3. 引入 gulp-sass
const sass = require('gulp-sass')

// 3-1. 引入 gulp-uglify
const uglify = require('gulp-uglify')

// 3-2. 引入 gulp-babel
// es6 转 es5 虽然需要下载三个依赖，但是只需要引入一个 gulp-babel 就可以了
const babel = require('gulp-babel')

// 4. 引入 gulp-htmlmin
const htmlmin = require('gulp-htmlmin')

// 2-1. 创建一个 css 的任务
gulp.task('css' , function () {
return gulp
.src('./src/css/**')
.pipe(autoPrefixer({
browsers: ['last 2 versions']
}))
.pipe(cssmin())
.pipe(gulp.dest('./dist/css'))
})

// 2-2. 创建一个 sass 任务
gulp.task('sass' , function () {
return gulp
.src('./src/sass/**')
.pipe(sass())
```



```

 .pipe(autoPrefixer())
 .pipe(cssmin())
 .pipe(gulp.dest('./dist/css'))
 })

// 3. 创建一个 js 任务
gulp.task('js' , function () {
 return gulp
 .src('./src/js/**')
 .pipe(babel({
 presets: ['es2015']
 }))
 .pipe(uglify())
 .pipe(gulp.dest('./dist/js'))
 })

// 4. 创建一个 html 任务
gulp.task('html' , function () {
 return gulp
 .src('./src/pages/**')
 .pipe(htmlmin({
 removeEmptyAttributes: true, // 移出所有空属性
 collapseWhitespace: true // 压缩 html
 }))
 .pipe(gulp.dest('./dist/pages'))
 })

// 5. 创建一个 lib 任务
gulp.task('lib' , function () {
 return gulp
 .src('./src/lib/**')
 .pipe(gulp.dest('./dist/lib'))
 })

// 6. 创建一个 static 任务
gulp.task('static' , function () {
 return gulp
 .src('./src/static/**')
 .pipe(gulp.dest('./dist/static'))
 })

// 7. 配置一个默认任务
```

```
gulp.task('default', ['css', 'sass', 'js', 'html', 'lib', 'static'])
```

- 1.
2. - 这样配置完毕以后，当你运行 `gulp` 的时候，会自动执行 `default` 任务
3. - 然后 `default` 任务就会把后面数组中写的几个任务并行执行了
- 4.
5. - 接下来就是去命令行里面运行 `gulp`
- 6.
7. ```shell
8. # 运行 gulp，会默认执行 default 任务
9. \$ gulp

## 清除 DIST 文件夹

- 当你在编译的时候，如果第一次编译有一个叫做 `a.css` 的文件，他会帮你编译
- 后来你把 `a.css` 文件改名字了，改叫 `b.css`，再次进行编译的时候
- 会给你新生成一个 `b.css` 文件在 `dist` 文件夹中
- 之前的 `a.css` 文件也不会消失
- 那么这样就不是很好了
- 我们比较好的做法是
  - 在执行所有的编译之前
  - 先把 `dist` 文件夹删除掉
  - 然后再执行所有的编译操作
- 这样就不会用多余的文件留下了
- 这个时候我们就要使用一个依赖
  - `npm i -D gulp-clean`
- 然后去配置文件里面配置一个任务

```
```javascript
// 我是 gulpfile.js 文件

// 1. 引入 gulp
const gulp = require( 'gulp' )

// 2. 配置一个任务
```

```
// 2-1. 引入 gulp-cssmin
const cssmin = require( 'gulp-cssmin' )

// 2-2. 引入 gulp-autoprefixer
const autoPrefixer = require( 'gulp-autoprefixer' )

// 2-3. 引入 gulp-sass
const sass = require( 'gulp-sass' )

// 3-1. 引入 gulp-uglify
const uglify = require( 'gulp-uglify' )

// 3-2. 引入 gulp-babel
// es6 转 es5 虽然需要下载三个依赖，但是只需要引入一个 gulp-babel 就可以了
const babel = require( 'gulp-babel' )

// 4. 引入 gulp-htmlmin
const htmlmin = require( 'gulp-htmlmin' )

// 5. 引入 gulp-clean
const clean = require( 'gulp-clean' )

// 2-1. 创建一个 css 的任务
gulp.task( 'css' , function () {
return gulp
.src( './src/css/**' )
.pipe(autoPrefixer({
browsers: [ 'last 2 versions' ]
}))
.pipe(cssmin())
.pipe(gulp.dest( './dist/css' ))
})

// 2-2. 创建一个 sass 任务
gulp.task( 'sass' , function () {
return gulp
.src( './src/sass/**' )
.pipe(sass())
.pipe(autoPrefixer())
.pipe(cssmin())
.pipe(gulp.dest( './dist/css' ))
})

// 3 创建一个 js 任务
```

```
// 3. 创建一个 js 任务
gulp.task( 'js' , function () {
  return gulp
    .src( './src/js/**' )
    .pipe(babel({
      presets: [ 'es2015' ]
    }))
    .pipe(uglify())
    .pipe(gulp.dest( './dist/js' ))
})

// 4. 创建一个 html 任务
gulp.task( 'html' , function () {
  return gulp
    .src( './src/pages/**' )
    .pipe(htmlmin({
      removeEmptyAttributes: true, // 移出所有空属性
      collapseWhitespace: true // 压缩 html
    }))
    .pipe(gulp.dest( './dist/pages' ))
})

// 5. 创建一个 lib 任务
gulp.task( 'lib' , function () {
  return gulp
    .src( './src/lib/**' )
    .pipe(gulp.dest( './dist/lib' ))
})

// 6. 创建一个 static 任务
gulp.task( 'static' , function () {
  return gulp
    .src( './src/static/**' )
    .pipe(gulp.dest( './dist/static' ))
})

// 8. 创建一个 clean 任务
gulp.task( 'clean' , function () {
  return gulp
    .src( './dist' )
    .pipe(clean())
})
```

// 7. 配置一个默认任务

```
gulp.task('default', ['css', 'sass', 'js', 'html', 'lib', 'static'])
```

- 1.
2. - 接下来我们执行任务的时候就应该先执行 `clean` 任务，再执行 `default` 任务
- 3.
4. ````shell`
5. `# 执行 clean 任务`
6. `$ gulp clean`
- 7.
8. `# 执行 default 任务`
9. `$ gulp`

按顺序执行任务

- 因为每次打包都要执行一个 `clean` 任务
- 那么我们就可以把 `clean` 任务也加到 `default` 任务的队列里面执行
 - `gulp.task('default', ['clean', 'css', 'sass', 'js', 'html', 'lib', 'static'])`
- 这样我们运行的时候，每次就都会帮我们执行 `clean` 任务了
- 但是会出现一个问题
 - 因为后面队列里面的任务是并行的
 - 那么有的时间长有的时间短
 - 那么就会出现有一些文件已经压缩完毕放在 `dist/` 文件夹里面了
 - 然后 `clean` 任务执行完毕的时候又给直接清除掉了
 - 这样不是很好
- 我们就需要让这些任务出现先后顺序
 - 先执行一下 `clean` 任务
 - `clean` 执行完毕以后，剩下的几个压缩文件的任务可以并行执行
- 这个时候我们就需要用到一个依赖
 - `npm i -D run-sequence`
- 这个依赖是逐步执行任务的一个包
- 这个时候我们修改一下 `default` 任务就好了

```
```javascript
// 我是 gulpfile.js 文件

// 1. 引入 gulp
const gulp = require('gulp')

// 2-1. 引入 gulp-cssmin
const cssmin = require('gulp-cssmin')

// 2-2. 引入 gulp-autoprefixer
const autoPrefixer = require('gulp-autoprefixer')

// 2-3. 引入 gulp-sass
const sass = require('gulp-sass')

// 3-1. 引入 gulp-uglify
const uglify = require('gulp-uglify')

// 3-2. 引入 gulp-babel
// es6 转 es5 虽然需要下载三个依赖，但是只需要引入一个 gulp-babel 就可以了
const babel = require('gulp-babel')

// 4. 引入 gulp-htmlmin
const htmlmin = require('gulp-htmlmin')

// 5. 引入 gulp-clean
const clean = require('gulp-clean')

// 6. 引入 run-sequence
const runSequence = require('run-sequence')

// 2-1. 创建一个 css 的任务
gulp.task('css' , function () {
 return gulp
 .src('./src/css/**')
 .pipe(autoPrefixer({
 browsers: ['last 2 versions']
 }))
 .pipe(cssmin())
 .pipe(gulp.dest('./dist/css'))
})

// 2-2. 创建一个 sass 任务
gulp.task('sass' , function () {
```

```
return gulp
.src('./src/sass/**')
.pipe(sass())
.pipe(autoPrefixer())
.pipe(cssmin())
.pipe(gulp.dest('./dist/css'))
})

// 3. 创建一个 js 任务
gulp.task('js' , function () {
return gulp
.src('./src/js/**')
.pipe(babel({
presets: ['es2015']
}))
.pipe(uglify())
.pipe(gulp.dest('./dist/js'))
})

// 4. 创建一个 html 任务
gulp.task('html' , function () {
return gulp
.src('./src/pages/**')
.pipe(htmlmin({
removeEmptyAttributes: true, // 移出所有空属性
collapseWhitespace: true // 压缩 html
}))
.pipe(gulp.dest('./dist/pages'))
})

// 5. 创建一个 lib 任务
gulp.task('lib' , function () {
return gulp
.src('./src/lib/**')
.pipe(gulp.dest('./dist/lib'))
})

// 6. 创建一个 static 任务
gulp.task('static' , function () {
return gulp
.src('./src/static/**')
.pipe(gulp.dest('./dist/static'))
})
```

```
// 8. 创建一个 clean 任务
gulp.task('clean', function () {
 return gulp
 .src('./dist')
 .pipe(clean())
})

// 7. 改写 default 任务
gulp.task('default', function () {
 // 里面的每一个参数都可以是一个任务或者一个任务队列
 // 再执行任务的时候，会把前一个任务队列完成的情况下再执行下一个任务队列
 runSequence('clean', ['css', 'sass', 'js', 'html', 'lib', 'static'])
})

1.
2. - 这样依赖，每次都是先执行 `clean` 任务
3. - 执行完毕之后再并行执行那些压缩的任务就不会出现问题了
4.
5. - 然后我们就去命令行中运行 `gulp` 执行一下 `default` 任务就可以了
6.
7. ```shell
8. # 运行 gulp
9. $ gulp
```

## 自动打开浏览器

- 我们的打包工作已经完成了
- 接下来要是能再打包完毕自动帮我把浏览器打开就好了
- 省的我自己去开了
- 这个时候是可以做到的
- 需要一个依赖
  - `npm i -D gulp-webserver`
- 然后进行配置

```
```javascript
```



```
// 我是 gulpfile.js 文件

// 1. 引入 gulp
const gulp = require( 'gulp' )

// 2-1. 引入 gulp-cssmin
const cssmin = require( 'gulp-cssmin' )

// 2-2. 引入 gulp-autoprefixer
const autoPrefixer = require( 'gulp-autoprefixer' )

// 2-3. 引入 gulp-sass
const sass = require( 'gulp-sass' )

// 3-1. 引入 gulp-uglify
const uglify = require( 'gulp-uglify' )

// 3-2. 引入 gulp-babel
// es6 转 es5 虽然需要下载三个依赖，但是只需要引入一个 gulp-babel 就可以了
const babel = require( 'gulp-babel' )

// 4. 引入 gulp-htmlmin
const htmlmin = require( 'gulp-htmlmin' )

// 5. 引入 gulp-clean
const clean = require( 'gulp-clean' )

// 6. 引入 run-sequence
const runSequence = require( 'run-sequence' )

// 7. 引入 gulp-webserver
const webserver = require( 'gulp-webserver' )

// 2-1. 创建一个 css 的任务
gulp.task( 'css' , function () {
return gulp
.src( './src/css/**' )
.pipe(autoPrefixer({
browsers: [ 'last 2 versions' ]
}))
.pipe(cssmin())
.pipe(gulp.dest( './dist/css' ))
})
```

// 2-2. 创建一个 sass 任务

```
gulp.task( 'sass' , function () {  
  return gulp  
    .src( './src/sass/**' )  
    .pipe(sass())  
    .pipe(autoPrefixer())  
    .pipe(cssmin())  
    .pipe(gulp.dest( './dist/css' ))  
  })
```

// 3. 创建一个 js 任务

```
gulp.task( 'js' , function () {  
  return gulp  
    .src( './src/js/**' )  
    .pipe(babel({  
      presets: [ 'es2015' ]  
    })))  
    .pipe(uglify())  
    .pipe(gulp.dest( './dist/js' ))  
  })
```

// 4. 创建一个 html 任务

```
gulp.task( 'html' , function () {  
  return gulp  
    .src( './src/pages/**' )  
    .pipe(htmlmin({  
      removeEmptyAttributes: true, // 移出所有空属性  
      collapseWhitespace: true // 压缩 html  
    })))  
    .pipe(gulp.dest( './dist/pages' ))  
  })
```

// 5. 创建一个 lib 任务

```
gulp.task( 'lib' , function () {  
  return gulp  
    .src( './src/lib/**' )  
    .pipe(gulp.dest( './dist/lib' ))  
  })
```

// 6. 创建一个 static 任务

```
gulp.task( 'static' , function () {  
  return gulp
```

```
.src( './src/static/**' )
.pipe(gulp.dest( './dist/static' ))
})
```

// 8. 创建一个 clean 任务

```
gulp.task( 'clean' , function () {
return gulp
.src( './dist' )
.pipe(clean())
})
```

// 9. 创建一个 webserver 任务

```
gulp.task( 'webserver' , function () {
return gulp
.src( './dist' )
.pipe(webserver({
host: 'localhost' , // 配置打开浏览器的域名
port: 3000, // 配置打开浏览器的端口号
livereload: true, // 自动刷新浏览器
open: './pages/index.html' // 默认打开 dist 文件夹下的哪个文件
}))
})
```

// 7. 改写 default 任务

```
gulp.task( 'default' , function () {
// 里面的每一个参数都可以是一个任务或者一个任务队列
// 再执行任务的时候, 会把前一个任务队列完成的情况下再执行下一个任务队列
runSequence(
  'clean' ,
  [ 'css' , 'sass' , 'js' , 'html' , 'lib' , 'static' ],
  'webserver' )
})
```

- 1.
2. - 因为我们打开浏览器应该再所有压缩任务都完成以后
3. - 再把浏览器打开
4. - 所以我们把他排在任务的第三个队列上
- 5.
6. - 这个 `webserver` 会自动帮我们启动一个服务器
- 7.
8. - 是一个 `node` 的服务器
- 9.

10. - 所以我们的页面也相当于是服务器上打开的

14. **### 修改内容自动刷新**

16. - 我们刚才再配置 `webserver` 的时候有一个自动刷新的功能被我们开启了

18. - 但是我们修改一些文件的时候发现并没有自动刷新

20. - 这个是因为，我们只是开启了自动刷新，但是你修改文件以后并没有自动帮我们重新编译

22. - 那么 `dist/` 文件夹下的内容就不会更改，那么刷新就没有意义

24. - 所以我们应该再制作一个任务，当文件夹下的内容修改的时候，自动帮我们重新编译

26. - 这个不需要任何依赖，我们只需要配置一个监控文件改变的任务就行了

```
28. ```javascript
```

```
29. // 我是 gulpfile.js 文件
```

```
31. // 1. 引入 gulp
```

```
32. const gulp = require('gulp')
```

```
34. // 2-1. 引入 gulp-cssmin
```

```
35. const cssmin = require('gulp-cssmin')
```

```
37. // 2-2. 引入 gulp-autoprefixer
```

```
38. const autoPrefixer = require('gulp-autoprefixer')
```

```
40. // 2-3. 引入 gulp-sass
```

```
41. const sass = require('gulp-sass')
```

```
43. // 3-1. 引入 gulp-uglify
```

```
44. const uglify = require('gulp-uglify')
```

```
46. // 3-2. 引入 gulp-babel
```

```
47. // es6 转 es5 虽然需要下载三个依赖，但是只需要引入一个 gulp-babel 就可以了
```

```
48. const babel = require('gulp-babel')
```

```
50. // 4. 引入 gulp-htmlmin
```

```

51.   const htmlmin = require('gulp-htmlmin')
52.
53.   // 5. 引入 gulp-clean
54.   const clean = require('gulp-clean')
55.
56.   // 6. 引入 run-sequence
57.   const runSequence = require('run-sequence')
58.
59.   // 7. 引入 gulp-webserver
60.   const webserver = require('gulp-webserver')
61.
62.
63.   // 2-1. 创建一个 css 的任务
64.   gulp.task('css', function () {
65.     return gulp
66.       .src('./src/css/**')
67.       .pipe(autoPrefixer({
68.         browsers: ['last 2 versions']
69.       }))
70.       .pipe(cssmin())
71.       .pipe(gulp.dest('./dist/css'))
72.   })
73.
74.   // 2-2. 创建一个 sass 任务
75.   gulp.task('sass', function () {
76.     return gulp
77.       .src('./src/sass/**')
78.       .pipe(sass())
79.       .pipe(autoPrefixer())
80.       .pipe(cssmin())
81.       .pipe(gulp.dest('./dist/css'))
82.   })
83.
84.   // 3. 创建一个 js 任务
85.   gulp.task('js', function () {
86.     return gulp
87.       .src('./src/js/**')
88.       .pipe(babel({
89.         presets: ['es2015']
90.       }))
91.   })

```

```
91.     .pipe(uglify())
92.     .pipe(gulp.dest('./dist/js'))
93.   })
94.
95.   // 4. 创建一个 html 任务
96.   gulp.task('html', function () {
97.     return gulp
98.       .src('./src/pages/**')
99.       .pipe(htmlmin({
100.         removeEmptyAttributes: true, // 移出所有空属性
101.         collapseWhitespace: true // 压缩 html
102.       }))
103.       .pipe(gulp.dest('./dist/pages'))
104.   })
105.
106.   // 5. 创建一个 lib 任务
107.   gulp.task('lib', function () {
108.     return gulp
109.       .src('./src/lib/**')
110.       .pipe(gulp.dest('./dist/lib'))
111.   })
112.
113.   // 6. 创建一个 static 任务
114.   gulp.task('static', function () {
115.     return gulp
116.       .src('./src/static/**')
117.       .pipe(gulp.dest('./dist/static'))
118.   })
119.
120.   // 8. 创建一个 clean 任务
121.   gulp.task('clean', function () {
122.     return gulp
123.       .src('./dist')
124.       .pipe(clean())
125.   })
126.
127.   // 9. 创建一个 webserver 任务
128.   gulp.task('webserver', function () {
129.     return gulp
130.       .src('./dist')
131.       .pipe(webserver({
```

```
132.   host: 'localhost', // 配置打开浏览器的域名
133.   port: 3000, // 配置打开浏览器的端口号
134.   livereload: true, // 自动刷新浏览器
135.   open: './pages/index.html' // 默认打开 dist 文件夹下的哪个文件
136.   )))
137. })
138.
139. // 10. 创建一个 watch 任务
140. gulp.task('watch', function () {
141.   gulp.watch('./src/css/**', ['css'])
142.   gulp.watch('./src/sass/**', ['sass'])
143.   gulp.watch('./src/js/**', ['js'])
144.   gulp.watch('./src/pages/**', ['html'])
145.   gulp.watch('./src/lib/**', ['lib'])
146.   gulp.watch('./src/static/**', ['static'])
147. })
148.
149. // 7. 改写 default 任务
150. gulp.task('default', function () {
151.   // 里面的每一个参数都可以是一个任务或者一个任务队列
152.   // 再执行任务的时候，会把前一个任务队列完成的情况下再执行下一个任务队列
153.   runSequence(
154.     'clean',
155.     ['css', 'sass', 'js', 'html', 'lib', 'static'],
156.     ['webserver', 'watch'])
157. })
```

- 因为再文件压缩完毕以后，我们就可以开始监控了
- 所以把这个 `watch` 任务和 `webserver` 任务并行就可以了
- 接下来我们就可以再命令行运行 `gulp` 了

1. `# 运行 gulp`
2. `$ gulp`

- 这个时候，我们只要进行代码书写就可以了
- 随着我们写完，会自动帮我们打包编译，并且自动进行浏览器刷新的

使用 GULP 配置代理

- 我们的 `gulp` 自动打包构建已经完成了

- 我们的正常开发已经可以进行下去了
- 我们就剩最后一个功能没有实现了
- 那就是跨域请求的问题
- 因为我们的项目中肯定有一些数据是向后台请求过来的
- 那么我们就要发送请求
- 我们又不能保重 服务端接口 和我们是同源的
- 所以我们就要配置一个代理
- `gulp` 给我们启动的这个服务器是 `node` 的服务器
- 所以也可以直接配置代理
- 不需要任何依赖，只要再 `webserver` 任务中多加一个配置就可以了

```
1.  gulp.task('webserver', function () {
2.     return gulp
3.       .src('./dist')
4.       .pipe(webserver({
5.         host: 'localhost',
6.         port: 3000,
7.         livereload: true,
8.         open: './pages/index.html',
9.         proxies: [ // 配置所有代理
10.            { // 其中一个代理配置
11.              source: '/login', // 使用的关键字
12.              target: 'http://localhost:80/login.php' // 代理的路径
13.            }
14.          ]
15.        }))
16.    })
```

- 这个时候你就可以正常发送请求了
- 当你要请求代理地址的时候，只要再这里配置一下，使用代理关键字进行请求就可以了

GULP完整配置文件（简单版）

- 所有需要用到的依赖

- i. `gulp@3.9.1`
- ii. `gulp-cssmin`
- iii. `gulp-autoprefixer`
- iv. `gulp-sass`
- v. `gulp-uglify`
- vi. `gulp-babel@7.0.1`
- vii. `babel-core`
- viii. `babel-preset-es2015`
- ix. `gulp-htmlmin`
- x. `gulp-clean`
- xi. `run-sequence`
- xii. `gulp-webserver`

- `gulpfile.js` 文件内容

```
1. // 我是 gulpfile.js 文件
2. const gulp = require('gulp')
3. const cssmin = require('gulp-cssmin')
4. const autoPrefixer = require('gulp-autoprefixer')
5. const sass = require('gulp-sass')
6. const uglify = require('gulp-uglify')
7. const babel = require('gulp-babel')
8. const htmlmin = require('gulp-htmlmin')
9. const clean = require('gulp-clean')
10. const runSequence = require('run-sequence')
11. const webserver = require('gulp-webserver')
12.
13. gulp.task('css', function () {
14.   return gulp
15.     .src('./src/css/**')
16.     .pipe(autoPrefixer({
17.       browsers: ['last 2 versions']
18.     }))
19.     .pipe(cssmin())
20.     .pipe(gulp.dest('./dist/css'))
21.   })
22.
23. gulp.task('sass', function () {
24.   return gulp
25.     .src('./src/sass/**')
26.     .pipe(sass())
```

```
26.   .pipe(sass())
27.   .pipe(autoPrefixer())
28.   .pipe(cssmin())
29.   .pipe(gulp.dest('./dist/css'))
30. })
31.
32. gulp.task('js', function () {
33.   return gulp
34.     .src('./src/js/**')
35.     .pipe(babel({
36.       presets: ['es2015']
37.     }))
38.     .pipe(uglify())
39.     .pipe(gulp.dest('./dist/js'))
40. })
41.
42. gulp.task('html', function () {
43.   return gulp
44.     .src('./src/pages/**')
45.     .pipe(htmlmin({
46.       removeEmptyAttributes: true,
47.       collapseWhitespace: true
48.     }))
49.     .pipe(gulp.dest('./dist/pages'))
50. })
51.
52. gulp.task('lib', function () {
53.   return gulp
54.     .src('./src/lib/**')
55.     .pipe(gulp.dest('./dist/lib'))
56. })
57.
58. gulp.task('static', function () {
59.   return gulp
60.     .src('./src/static/**')
61.     .pipe(gulp.dest('./dist/static'))
62. })
63.
64. gulp.task('clean', function () {
65.   return gulp
66.     .src('./dist')
```

```
67.   .pipe(clean())
68. })
69.
70. gulp.task('webserver', function () {
71.   return gulp
72.     .src('./dist')
73.     .pipe(webserver({
74.       host: 'localhost',
75.       port: 3000,
76.       livereload: true,
77.       open: './pages/index.html',
78.       proxies: [
79.         {
80.           source: '/login',
81.           target: 'http://localhost:80/login.php'
82.         }
83.       ]
84.     })))
85. })
86.
87. gulp.task('watch', function () {
88.   gulp.watch('./src/css/**', ['css'])
89.   gulp.watch('./src/sass/**', ['sass'])
90.   gulp.watch('./src/js/**', ['js'])
91.   gulp.watch('./src/pages/**', ['html'])
92.   gulp.watch('./src/lib/**', ['lib'])
93.   gulp.watch('./src/static/**', ['static'])
94. })
95.
96. gulp.task('default', function () {
97.   runSequence(
98.     'clean',
99.     ['css', 'sass', 'js', 'html', 'lib', 'static'],
100.    ['webserver', 'watch'])
101. })
```