gulp

引入：

项目做好以后，在上线之前还有一些工作需要去做：

- 压缩css
- 压缩js
- 压缩图片
- 编译sass
- 合并文件
- 。。。

等等，在前端工作流出现之前，这些工作都由人力完成，而这些工作往往比写业务本身更加费时，效率非常之低且还容易出错，于是自动化的处理工具也就必然出现了。

前端的构建工具常见的有Grunt、Gulp、Webpack三种，Grunt比较老旧，功能少，更新少，插件少。

概念：

gulp是一个自动化构建工具，主要用来设定程序自动处理静态资源的工作。简单的说，gulp就是用来打包项目的。

官网：https://gulpjs.com/

中文官网：https://www.gulpjs.com.cn/docs/

搜索包的使用方法：http://www.npmjs.com

安装：

安装gulp工具：

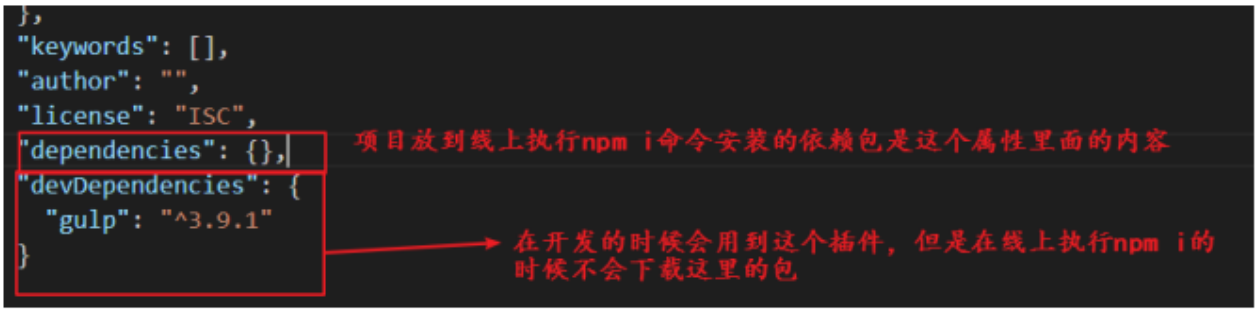
```
npm i gulp@3.9.1 -g
gulp -v # 测试是否安装成功
```

全局安装表示在当前电脑中可以使用gulp工具了

安装gulp依赖包

```
npm i gulp@3.9.1 --save-dev # 因为在上线后是不需要这个包的，所以将这个项目安装在开发依赖
```

安装在开发环境



局部安装表示在当前项目要使用的gulp

局部安装gulp要和全局安装的gulp版本保持一致

gulp是一个基于任务的工具，也就是说，gulp规定，不管做什么功能，都用统一的接口管理，必须去注册一个任务，然后去执行这个任务，在任务代码中，去做想想做的功能。这是gulp的特点之一：任务化。

gulp的每个功能都是一个任务，压缩css的任务、合并文件的任务。。。gulp规定任务要写在一个叫做glupfile.js的文件中，在这个文件中用来配置所有任务。

首先，gulp和node中的其他模块一样，使用的时候需要引入：

```
var glup = require("gulp");
```

这个gulp是一个对象，gulp提供了很多接口，都是这个对象的方法。

gulp提供的接口：

注册任务

```
gulp.task(name[, deps], fn)
```

参数：

1. name是任务名称，执行任务时，使用这个名称
2. fn是一个回掉函数，代表这个任务要做的事情

例：

```
gulp.task("print",function(){
    console.log("打印123");
})
```

执行任务：在命令行使用gulp命令，后面跟任务名称：

```
gulp print
```

如果任务比较多的话，一个一个来执行，效率会很低，所以gulp提供了一个默认任务，可以将要执行的所有任务放在一个数组中，这样只需要执行这个默认任务就能执行数组中的所有任务：

```
gulp.task("print1",function(){
    console.log("打印123");
})
gulp.task("print3",function(){
    console.log("打印321");
})
gulp.task("default",["print1","print3"]);
```

执行默认任务：不用写任务名

```
gulp
```

gulp自己有内存，当我们使用gulp进行项目构建的时候，gulp会将本地文件数据读取到gulp内存中，接下来的操作都在内存中进行，操作完成以后，再从gulp的内存中输出到本地，比如说当我们要合并两个文件的时候，先将这两个文件中的内容读取到内存中，然后在内存中进行合并，最后将合并后的内容从内存中输出到本地的文件中。

这样，对应着两个操作，一个是输入，一个输出，也就是I/O操作。这是gulp的又一个特点之一：基于流。

读取文件

将本地文件读取到gulp内存中

```
gulp.src(globs[, options])
```

参数：

src方法主要是用来读取目标源文件，所以参数就是一个目标源文件的路径

输出到文件

将内存中数据输出到本地文件中

```
gulp.dest(path[, options])
```

参数：

dest方法主要用来将数据输出到文件中，所以参数就是目标文件路径。

监视文件变化

用来监视某个或某些文件发生变化，可以在变化的时候，执行一个回掉函数，以保证文件中的代码和效果一致

```
gulp.watch(被监视的文件路径或被监视的文件路径组成的数据，要执行的任务组成的数组)
```

gulp插件

我们要处理文件的合并、压缩等操作，接口中没有提供，都放在了插件中。

插件下载：

```
npm install 插件名 --save-dev
```

- gulp-concat : 合并文件(js/css)
- gulp-uglify : 压缩js文件
- gulp-rename : 文件重命名
- gulp-less : 编译less
- gulp-sass：编译sass
- gulp-clean-css : 压缩css
- gulp-livereload : 实时自动编译刷新
- gulp-htmlmin：压缩html文件
- gulp-connect：热加载，配置一个服务器
- gulp-load-plugins：打包插件（里面包含了其他所有插件）

案例

文件目录结构：

```
|- dist # 存放目标文件
|- src # 存放源文件
  |- js
  |- css
  |- less
|- index.html
|- gulpfile.js-----gulp配置文件
```

首先在js文件夹下新建test1.js和test2.js，并写入内容

下载多个插件：

```
npm install gulp-concat gulp-uglify gulp-rename --save-dev
```

合并压缩js

合并这两个test文件，并放到dist下的js文件夹下，起名叫test.js

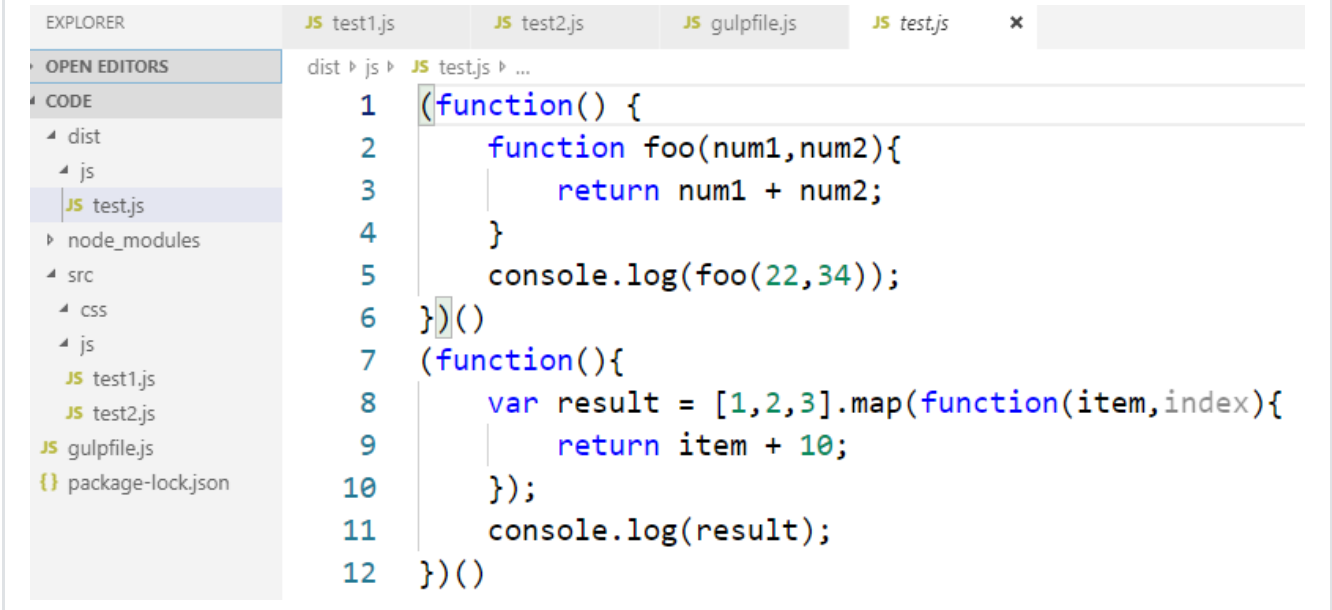
```
// 引入gulp
var gulp = require("gulp");
// 引入下载好的插件，插件返回的都是方法
var concat = require("gulp-concat");
var uglify = require("gulp-uglify");
var rename = require("gulp-rename");
// 注册任务：合并压缩js的
gulp.task("handleJs",function(){
    return gulp.src("./src/js/*.js")  // 找到目标源文件，将数据读取到gulp的内存中
    .pipe(concat("test.js")) // pipe是管道的意思，表示将上一步的操作流向下一步，concat在调用的时候指定合并后的文件名
    // 如果要找到js文件夹下的所有js文件和子目录下的js文件，应该写成 gulp.src("./src/js/**/*.js");
    .pipe(gulp.dest("./dist/js/")); // 输出内容到本地
});
```

使用命令执行：

```
gulp handleJs
```

最终结果：

合并两个文件



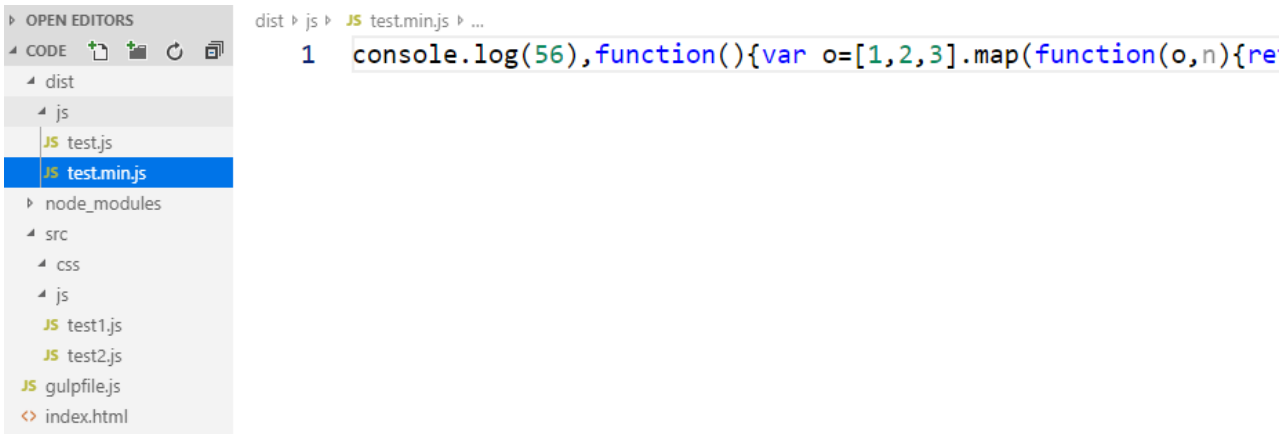
继续压缩这个文件：

```
// 引入gulp
var gulp = require("gulp");
// 引入下载好的插件，插件返回的都是方法
var concat = require("gulp-concat");
var uglify = require("gulp-uglify");
var rename = require("gulp-rename");
// 注册任务：合并压缩js的
gulp.task("handleJs",function(){
    return gulp.src("./src/js/*.js")
    .pipe(concat("test.js"))
    .pipe(gulp.dest("./dist/js/")) // 输出文件到本地
    .pipe(uglify()) // 压缩文件
    .pipe(rename("test.min.js")) // 将压缩后的文件重命名
    // 如果要逼格高一点，使用rename中的对象  rename({suffix:'.min'})
    .pipe(gulp.dest("./dist/js")); // 将压缩后的文件再放到这个目录下
});
```

重命名和压缩的顺序可以交换

执行，最终效果：

合并并压缩重命名



注意：当我们在源文件中写了没有调用的函数的时候，压缩的时候，系统默认觉得这个函数没有用，就不会被压缩

注意：压缩js，不能压缩es6的语法

解析sass语法

在sass文件夹下新建test3.scss文件，写入sass语法。下载gulp-sass插件

```
var gulp = require("gulp");
var sass = require("gulp-sass");
// 解析sass的任务
gulp.task("handleScss",function(){
    return gulp.src("./src/sass/*.scss")
    .pipe(sass()) // 编译scss文件为css文件，默认编译后的文件名和原文件名一样
    .pipe(gulp.dest("./src/css/"))
});
```

执行这个任务：

```
gulp handleScss
```

最终效果;

解析sass文件

合并压缩css

gulp-clean-css插件在使用的时候，有一个属性叫做compatibility，值为ie8，表示让这个css兼容到ie8

```
var gulp = require("gulp");
// 引入下载好的插件，插件返回的都是方法
var concat = require("gulp-concat");
```

```
var rename = require("gulp-rename");
var cleanCss = require("gulp-clean-css");
// 合并压缩css任务
gulp.task("handleCss",function(){
    return gulp.src("./src/css/*.css")
    .pipe(concat("test.css"))
    .pipe(gulp.dest("./dist/css/"))
    .pipe(cleanCss({
        compatibility:"ie8" // 让css兼容到ie8
    }))
    .pipe(rename({
        suffix:".min"
    }))
    .pipe(gulp.dest("./dist/css/"))
});
```

执行这个任务，最终结果;

合并压缩css

注意：合并css用的插件是gulp-clean-css，而不是前面使用过的uglify

每次都是手动启动很多任务，我们可以将多个任务都放在默认任务中，只需要启动默认任务就可运行多个任务

异步任务

将多个任务放在默认任务中执行，

```
gulp.task('default',['handleJs','handleSass','handleCss']);
```

然后只要使用gulp命令不用参数就可以运行。**注意：这样的写法在gulp4的版本中格式不支持的，在gulp4中需要用另外的写法**

执行过程图

从上图中可以看出，这几个任务并没有按照设定的顺序去执行，也就是说，这几个任务都是异步执行的。

接下来，将每个任务中的return删掉：

```
// 引入gulp
var gulp = require("gulp");
// 引入下载好的插件，插件返回的都是方法
var concat = require("gulp-concat");
var uglify = require("gulp-uglify");
var rename = require("gulp-rename");
var sass = require("gulp-sass");
var cleanCss = require("gulp-clean-css");
// 注册任务
gulp.task("handleJs",function(){
    gulp.src("./src/js/*.js")  // 找到目标源文件，将数据读取到gulp的内存中
    .pipe(concat("test.js")) // pipe是管道的意思，表示将上一步的操作流向下一步，concat在调用的时候指定合并后的文件名
    // 如果要找到js文件夹下的所有js文件和子目录下的js文件，应该写成 gulp.src("./src/js/**/*.js");
    .pipe(gulp.dest("./dist/js/")) // 输出文件到本地
    .pipe(uglify()) // 压缩文件
    .pipe(rename("test.min.js")) // 将压缩后的文件重命名
    // 如果要逼格高一点，使用rename中的对象  rename({suffix:'.min'})
    .pipe(gulp.dest("./dist/js")); // 将压缩后的文件再放到这个目录下
});
// 解析sass的任务
gulp.task("handleSass",function(){
    gulp.src("./src/less/*.scss")
    .pipe(less()) // 编译less文件为css文件，默认编译后的文件名和原文件名一样
    .pipe(gulp.dest("./src/css/"))
});
// 合并压缩css任务
gulp.task("handleCss",function(){
    gulp.src("./src/css/*.css")
    .pipe(concat("test.css"))
    .pipe(gulp.dest("./dist/css/"))
    .pipe(cleanCss({
        compatibility:"ie8" // 让css兼容到ie8
    }))
    .pipe(rename({
        suffix:".min"
    }))
    .pipe(gulp.dest("./dist/css/"))
});
gulp.task('default',['handleJs','handleSass','handleCss']);
```

然后继续调用执行默认任务

删掉return的执行过程

将多个任务代码中的return删掉以后，这多个任务就是同步的。

起关键性作用是任务代码中的return。return的作用是可以保证任务是异步执行，还可以保证任务执行完成之后，将gulp内存中数据清除掉。

也就是说，gulp任务可以是同步的也可以是异步的。这是gulp的第三个特点：可同步可异步

任务依赖

上述案例中，css任务明显依赖于sass任务，但是如果异步执行的话，很可能在合并css的时候，sass还没有结束，那么sass任务的执行完全没有了意义，此时，需要在css任务的task方法中，添加第二个参数，是一个数组，放的是依赖的任务名称。

```
// 合并压缩css任务
gulp.task("handleCss",["handleSass"],function(){
    return gulp.src("./src/css/*.css")
    .pipe(concat("test.css"))
    .pipe(gulp.dest("./dist/css/"))
    .pipe(cleanCss({
        compatibility:"ie8" // 让css兼容到ie8
    }))
    .pipe(rename({
        suffix:".min"
    }))
    .pipe(gulp.dest("./dist/css/"))
});
```

如果有多个依赖任务，都放到第二个参数数组中。这样可以保证在执行当前任务之前，先执行依赖任务。

压缩html

在压缩html的时候，gulp-htmlmin方法中有一个属性叫collapseWhitespace，值为true，表示压缩掉html中的空格

```
var gulp = require("gulp");
var htmlmin = require("gulp-htmlmin");
// 压缩html
gulp.task("handleHtml",function(){
    return gulp.src("./index.html")
    .pipe(htmlmin({
        collapseWhitespace:true
    }))
    .pipe(gulp.dest("./dist/"))
});
gulp.task('default',['handleJs','handleSass','handleCss',"handleHtml"]);
```

执行任务

压缩html

注意：压缩后的html放在另外的文件夹，引入的css和js文件会不生效，所以在开发的时候，一定要注意这个路径的问题。

此时，我们每次对文件做修改后，都需要手动启动任务，重新进行压缩、合并等操作，然后手动刷新页面才能看到改变，为了方便，我们可以启动一个监视任务，每当文件发生变化， 自动启动启动任务，重新进行压缩、合并等操作

半自动化构建项目

下载插件：

```
npm install gulp-livereload --save-dev
```

启动监视任务的时候，应该保证默认任务已经启动才能进行监视，所以，在watch任务中应该传入依赖任务参数default，任务代码中分两部分内容，首先开启监听，然后确认监听目标文件并绑定相应的任务。这样就已经ok了，但是为了安全起见，我们遵循官网的写法，在每个被监听的任务重，添加实时刷新的管道

```
// 监视任务
gulp.task("watch",["default"],function(){
    // 开启监听
    livereload.listen();
    // 确认监听的目标文件以及绑定相应的任务
    gulp.watch("./src/js/*.js",["handleJs"]);
    gulp.watch(["./src/css/*.css","./src/less/*.sass"],["handleCss"]); // 监听多种文件放在数组中
});
```

执行的时候，只要执行watch任务就可以，因为watch任务不会退出，会阻塞在这里进行监听，且watch有依赖任务default，会自动帮我们启动默认任务。启动watch任务后，每当我们修改被监听的任务，系统都会帮我们重新进行压缩、合并等我们绑定的任务操作。

此时，我们还需要手动刷新页面，所以，只能算作半自动

全自动化构建项目

下载插件：

```
npm i gulp-connect --save-dev
```

根据插件启动一个服务器，帮我们自动刷新页面

```
// 注册全自动监视任务
gulp.task("server",["default"],function(){
    // 配置服务器选项
    connect.server({
        root:'./dist/', // 配置服务器根目录
        livereload:true, // 实时刷新
        port:5000 // 配置服务器端口
    });
    // 确认监听的目标文件以及绑定相应的任务
    gulp.watch("./src/js/*.js",["handleJs"]);
    gulp.watch(["./src/css/*.css","./src/less/*.scss"],["handleCss"]); // 监听多种文件放在数组中
});
```

还需要在每个任务中添加实时刷新设置：

```
.pipe(connect.reload()) // 将更改后的内容实时填充到页面中
```

启动这个任务后，给我们提供了一个访问项目的服务器，只要我们对内容做了修改，就会自动执行任务，且页面会自动刷新：

全自动任务启动过程

如果我们想再省事一点，不用手动打开网页，而让系统自动帮我们打开网页的话，使用open插件，自动打开链接

下载插件：

```
npm i open --save-dev
```

在全自动任务下，设置自动打开服务器链接

```
var open = require("open");
// 注册全自动监视任务
gulp.task("server",["default"],function(){
    // 配置服务器选项
    connect.server({
        root:'./dist/', // 配置服务器根目录
        livereload:true, // 实时刷新
        port:5000 // 配置服务器端口
    });
    // 自动打开指定连接
    open("http://localhost:5000");
    // 确认监听的目标文件以及绑定相应的任务
    gulp.watch("./src/js/*.js",["handleJs"]);
```

```
    gulp.watch(["./src/css/*.css","./src/less/*.scss"],["handleCss"]); // 监听多种文件放在数组中
});
```

这时候只要我们启动这个任务，就会自动帮我们打开网页。

扩展

所有包

gulp提供了一个插件，可以代替所有插件。也就是说只要下载这一个插件就可以使用其他所有插件了

下载插件：

```
npm install gulp-load-plugins --save-dev
```

使用方式：引入以后，是个函数，这个函数有打包其他插件的代码，所以一定要调用这个函数，得到对象，这个对象中就包含了其他插件的方法，不用再引入其他插件了

```
var $ = require("gulp-load-plugins")();
```

原来使用其他插件的时候，都是函数形式，使用打包插件的话， 就是将原来的函数换成现在这个对象的方法，一下是修改后的全部代码：

```
var $ = require("gulp-load-plugins")();
var gulp = require("gulp");
gulp.task("handleJs",function(){
    return gulp.src("./src/js/*.js")  // 找到目标源文件，将数据读取到gulp的内存中
    .pipe($.concat("test.js"))
    .pipe(gulp.dest("./dist/js/")) // 输出文件到本地
    .pipe($.uglify()) // 压缩文件
    .pipe($.rename("test.min.js"))
    .pipe(gulp.dest("./dist/js"))
    .pipe($.connect.reload()) // 将更改后的内容实时填充到页面中
});
// 解析less的任务
gulp.task("handleSass",function(){
    return gulp.src("./src/less/*.less")
    .pipe($.less())
    .pipe(gulp.dest("./src/css/"))
    .pipe($.connect.reload()) // 将更改后的内容实时填充到页面中
});
// 合并压缩css任务
gulp.task("handleCss",["handleSass"],function(){
    return gulp.src("./src/css/*.css")
    .pipe($.concat("test.css"))
    .pipe(gulp.dest("./dist/css/"))
    .pipe($.cleanCss({
        compatibility:"ie8" // 让css兼容到ie8
    }))
    .pipe($.rename({
        suffix:".min"
    }))
```

```
    .pipe(gulp.dest("./dist/css/"))
    .pipe($.connect.reload()) // 将更改后的内容实时填充到页面中
});
// 压缩html
gulp.task("handleHtml",function(){
    return gulp.src("./index.html")
    .pipe($.htmlmin({
        collapseWhitespace:true
    }))
    .pipe(gulp.dest("./dist/"))
    .pipe($.connect.reload()) // 将更改后的内容实时填充到页面中
});
var open = require("open");
// 注册全自动监视任务
gulp.task("server",["default"],function(){
    $.connect.server({
        root:'./dist/', // 配置服务器根目录
        livereload:true, // 实时刷新
        port:5000 // 配置服务器端口
    });
    open("http://localhost:5000");
    gulp.watch("./src/js/*.js",["handleJs"]);
    gulp.watch(["./src/css/*.css","./src/less/*.less"],["handleCss"]);
});
gulp.task('default',['handleJs','handleSass','handleCss',"handleHtml"]);
```

css自动添加前缀

目的：将一些不兼容的css属性添加前缀让各个浏览器兼容

依赖插件：gulp-autoprefixer

任务代码：

```
var gulp = require("gulp");
var prefix = require("gulp-autoprefixer");
gulp.task("css",function(){
    gulp.src("./src/css**")
        .pipe(prefix({
                browsers:["last 5 version","iOS > 3","Firefox > 2","Google > 30"]
        }))
        .pipe(gulp.dest("./dist/css"));
});
```

会出现一个提示，希望将这个配置写在package.json中：

```
"browsersList":[
    "last 2 version",
    "iOS > 7",
    "Fixefox > 20"
]
```

es6转es5

为了让更多浏览器兼容项目，需要将项目中的es6的语法转为es5的语法。

依赖包：

```
gulp-babel@7.0.1
babel-core
babel-preset-es2015
```

导入的时候只要导入一个即可：

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

任务代码：

```
gulp.task('js', function () {
    return gulp
        .src('./src/js/**')
        .pipe(babel({
                presets: ['es2015'] // 必须要有这个参数，否则会报错
        }))
        .pipe(uglify())
        .pipe(gulp.dest('./dist/js'))
})
```

压缩图片

将切好的图片进行压缩，使图片更小，让项目运行起来更快

插件：gulp-imagemin

任务代码：

```
const imagemin = require("gulp-imagemin");
// 压缩图片：gulp-imagemin
gulp.task("image",function(){
    gulp.src("images/*.png")
    .pipe(imagemin())
    .pipe(gulp.dest("./imagemin/"))
});
```

清除目标文件夹

如果每次打包的时候起不一样的名字，会造成有些文件没有用，但是还占据空间。所以每次在打包之前应该先将之前的文件夹情空，然后再打包。

插件：gulp-clean

任务代码：

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

Gulp官方插件网站找寻插件。（ gulp-sass-china ）