目录

[**安装gulp** 3](#_Toc441567457)

[**安装gulp外挂** 3](#_Toc441567458)

[**载入外挂** 4](#_Toc441567459)

[**建立任务** 4](#_Toc441567460)

[**编译Sass，Autoprefix及缩小化** 4](#_Toc441567461)

[**JSHint，拼接及缩小化JavaScript** 5](#_Toc441567462)

[**图片压缩** 6](#_Toc441567463)

[**收拾乾淨!** 7](#_Toc441567464)

[**预设任务** 7](#_Toc441567465)

[**看守** 7](#_Toc441567466)

[**即时重整(LiveReload)** 8](#_Toc441567467)

[**全部在一起** 8](#_Toc441567468)

[BUG 修复 11](#_Toc441567469)

**Grunt靠边，全新的建构工具来了。**Gulp**的code-over-configuration不只让撰写任务(tasks)更加容易，也更好阅读及维护。**

Glup使用node.js串流(streams)让建构更快速，不须写出资料到硬盘的暂存档案/目录。如果你想了解更多有关串流–虽然不是必须的–你可以阅读[这篇文章](https://github.com/substack/stream-handbook)。Gulp利用来源档案当作输入，串流到一群外挂(plugins)，最后取得输出的结果，并非配置每一个外挂的输入与输出–就像Grunt。让我们来看个范例，分别在Gulp及Grunt建构Sass：

Grunt:

sass: {

dist: {

options: {

style: 'expanded'

},

files: {

'dist/assets/css/main.css': 'src/styles/main.scss',

}

}

},

autoprefixer: {

dist: {

options: {

browsers: [

'last 2 version', 'safari 5', 'ie 8', 'ie 9', 'opera 12.1', 'ios 6', 'android 4'

]

},

src: 'dist/assets/css/main.css',

dest: 'dist/assets/css/main.css'

}

},

grunt.registerTask('styles', ['sass', 'autoprefixer']);

Grunt需要各别配置外挂，指定其来源与目的路径。例如，我们将一个档案作为外挂Sass的输入，并储存输出结果。在设置Autoprefixer时，需要将Sass的输出结果作为输入，产生出一个新档案。来看看在Gulp中同样的配置：

Gulp:

gulp.task('sass', function() {

return gulp.src('src/styles/main.scss')

.pipe(sass({ style: 'compressed' }))

.pipe(autoprefixer('last 2 version', 'safari 5', 'ie 8', 'ie 9', 'opera 12.1', 'ios 6', 'android 4'))

.pipe(gulp.dest('dist/assets/css'))

});

在Gulp中我们只需要输入一个档案即可。经过外挂Sass处理，再传到外挂Autoprefixer，最终取得一个档案。这样的流程加快建构过程，省去读取及写出不必要的档案，只需要最终的一个档案。

所以，有趣了，现在要？让我们开始安装gulp并建立一个基本的gulpfile，包含几个核心任务来作为入门吧。

**安装gulp**

深入设置任务之前，需先安装gulp：

$ npm install gulp -g

这会将gulp安装到全域环境下，让你可以存取gulp的CLI。接著，需要在本地端的专案进行安装。cd到你的专案根目录，执行下列指令(请先确定你有package.json档案)：

package.json 文件：

{

"name": "gulp-build", //项目名，无重复

"version": "0.0.1",

"description": "The build process of my website with Gulp.js",

"private": true

}

$ npm install --save-dev require-dir@0.3.0

$ npm install gulp --save-dev

上述指令将gulp安装到本地端的专案内，并纪录于package.json内的devDependencies物件。

**安装gulp外挂**

接著安装一些外挂，完成下列任务：

* 编译Sass ([gulp-ruby-sass](https://github.com/sindresorhus/gulp-ruby-sass))
* Autoprefixer ([gulp-autoprefixer](https://github.com/Metrime/gulp-autoprefixer))
* 缩小化(minify)CSS ([gulp-minify-css](https://github.com/jonathanepollack/gulp-minify-css))
* JSHint ([gulp-jshint](https://github.com/wearefractal/gulp-jshint))
* 拼接 ([gulp-concat](https://github.com/wearefractal/gulp-concat))
* 丑化(Uglify) ([gulp-uglify](https://github.com/terinjokes/gulp-uglify))
* 图片压缩 ([gulp-imagemin](https://github.com/sindresorhus/gulp-imagemin))
* 即时重整(LiveReload) ([gulp-livereload](https://github.com/vohof/gulp-livereload))
* 清理档案 ([gulp-clean](https://github.com/peter-vilja/gulp-clean))
* 图片快取，只有更改过得图片会进行压缩 ([gulp-cache](https://github.com/jgable/gulp-cache/))
* 更动通知 ([gulp-notify](https://github.com/mikaelbr/gulp-notify))

执行下列指令来安装这些外挂:

$ npm install gulp-ruby-sass gulp-autoprefixer gulp-minify-css gulp-jshint gulp-concat gulp-uglify gulp-imagemin gulp-clean gulp-notify gulp-rename gulp-livereload gulp-cache --save-dev

指令将会安装必要的外挂，并纪录于package.json内的devDependencies物件。完整的gulp外挂清单可以在[这裡](http://gratimax.github.io/search-gulp-plugins/)找到。

**载入外挂**

接下来，我们需要建立一个gulpfile.js档案，并且载入这些外挂：

var gulp = require('gulp'),

sass = require('gulp-ruby-sass'),

autoprefixer = require('gulp-autoprefixer'),

minifycss = require('gulp-minify-css'),

jshint = require('gulp-jshint'),

uglify = require('gulp-uglify'),

imagemin = require('gulp-imagemin'),

rename = require('gulp-rename'),

clean = require('gulp-clean'),

concat = require('gulp-concat'),

notify = require('gulp-notify'),

cache = require('gulp-cache'),

livereload = require('gulp-livereload');

呼！看起来比Grunt有更多的事要做，对吧？Gulp外挂跟Grunt外挂有些许差异–它被设计成做一件事并且做好一件事。例如；Grunt的 imagemin利用快取来避免重複压缩已经压缩好的图片。在Gulp中，这必须透过一个快取外挂来达成，当然，快取外挂也可以拿来快取其他东西。这让建 构过程中增加了额外的弹性层面。蛮酷的，哼？

我们也可以像Grunt一样[自动载入](https://github.com/jackfranklin/gulp-load-tasks)所有已安装的外挂，但这不在此文章目的，所以我们将维持在手动的方式。

**建立任务**

**编译Sass，Autoprefix及缩小化**

首先，我们设置编译Sass。我们将编译Sass、接著通过Autoprefixer，最后储存结果到我们的目的地。接著产生一个缩小化的.min版本、自动重新整理页面及通知任务已经完成：

gulp.task('styles', function() {

return gulp.src('src/styles/main.scss')

.pipe(sass({ style: 'expanded' }))

.pipe(autoprefixer('last 2 version', 'safari 5', 'ie 8', 'ie 9', 'opera 12.1', 'ios 6', 'android 4'))

.pipe(gulp.dest('dist/assets/css'))

.pipe(rename({suffix: '.min'}))

.pipe(minifycss())

.pipe(gulp.dest('dist/assets/css'))

.pipe(notify({ message: 'Styles task complete' }));

});

继续下去之前，一个小小的说明。

gulp.task('styles', function() { ... )};

这个gulp.taskAPI用来建立任务。可以透过终端机输入$ gulp styles指令来执行上述任务。

return gulp.src('src/styles/main.scss')

这个gulp.srcAPI用来定义一个或多个来源档案。允许使用glob样式，例如/\*\*/\*.scss比对多个符合的档案。传回的串流(stream)让它成为非同步机制，所以在我们收到完成通知之前，确保该任务已经全部完成。

.pipe(sass({ style: 'expanded' }))

使用pipe()来串流来源档案到某个外挂。外挂的选项通常在它们各自的Github页面中可以找到。上面列表中我有留下各个外挂的连结，让你方便使用。

.pipe(gulp.dest('dist/assets/css'));

这个gulp.dest()API是用来设定目的路径。一个任务可以有多个目的地，一个用来输出扩展的版本，一个用来输出缩小化的版本。这个在上述的styles任务中已经有展示。

我建议阅读gulp的[API文件](https://github.com/gulpjs/gulp/blob/master/docs/API.md)，以了解这些函式方法。它们并不像看起来的那样可怕！

**JSHint，拼接及缩小化JavaScript**

希望你现在对于如何建立一个新的gulp任务有好想法。接下来，我们将设定脚本任务，包括lint、拼接及丑化:

gulp.task('scripts', function() {

return gulp.src('src/scripts/\*\*/\*.js')

.pipe(jshint('.jshintrc'))

.pipe(jshint.reporter('default'))

.pipe(concat('main.js'))

.pipe(gulp.dest('dist/assets/js'))

.pipe(rename({suffix: '.min'}))

.pipe(uglify())

.pipe(gulp.dest('dist/assets/js'))

.pipe(notify({ message: 'Scripts task complete' }));

});

一件事提醒，我们需要指定JSHint一个reporter。这裡我使用预设的reporter，适用于大多数人。更多有关此设定，你可以在[JSHint网站](http://www.jshint.com/docs/reporters/)取得。

**图片压缩**

接著，我们将设定图片压缩:

gulp.task('images', function() {

return gulp.src('src/images/\*\*/\*')

.pipe(imagemin({ optimizationLevel: 3, progressive: true, interlaced: true }))

.pipe(gulp.dest('dist/assets/img'))

.pipe(notify({ message: 'Images task complete' }));

});

这会将对所有来源图片进行imagemin处理。我们可以稍微更进一步，利用快取保存已经压缩过的图片，以便每次进行此任务时不需要再重新压缩。这裡只需要[gulp-cache](https://github.com/jgable/gulp-cache)外挂–稍早已经安装。我们需要额外设置才能使用这个外挂，因此修改这段程式码:

.pipe(imagemin({ optimizationLevel: 3, progressive: true, interlaced: true }))

成为这段:

.pipe(cache(imagemin({ optimizationLevel: 5, progressive: true, interlaced: true })))

现在只有新的或更动的图片会被压缩。乾淨俐落!

**收拾乾淨!**

在我们进行佈署之前，清除目的地目录并重建档案是一个好主意–以防万一任何已经被删除的来源档案遗留在目的地目录之中:

gulp.task('clean', function() {

return gulp.src(['dist/assets/css', 'dist/assets/js', 'dist/assets/img'], {read: false})

.pipe(clean());

});

我们可以传入一个目录(或档案)阵列到gulp.src。因为我们不想要读取已经被删除的档案，我们可以加入read: false选项来防止gulp读取档案内容–让它快一些。

**预设任务**

我们可以建立一个预设任务，当只输入$ gulp指令时执行的任务，这裡执行三个我们所建立的任务:

gulp.task('default', ['clean'], function() {

gulp.start('styles', 'scripts', 'images');

});

注意额外传入gulp.task的阵列。这裡我们可以定义任务相依(task dependencies)。在这个范例中，gulp.start开始任务前会先执行清理(clean)任务。Gulp中所有的任务都是并行(concurrently)执行，并没有先后顺序哪个任务会先完成，所以我们需要确保clean任务在其他任务开始前完成。

**注意**: 透过相依任务阵列来执行clean而非gulp.start是经过考虑的，在这个情境来看是最好的选择，以确保清理任务全部完成。

**看守**

为了能够看守档案，并在更动发生后执行相关任务，首先需要建立一个新的任务，使用gulp.watchAPI来看守档案:

gulp.task('watch', function() {

// 看守所有.scss档

gulp.watch('src/styles/\*\*/\*.scss', ['styles']);

// 看守所有.js档

gulp.watch('src/scripts/\*\*/\*.js', ['scripts']);

// 看守所有图片档

gulp.watch('src/images/\*\*/\*', ['images']);

});

透过gulp.watch指定想要看守的档案，并且透过相依任务阵列定义任务。执行$ gulp watch来开始看守档案，任何.scss、.js或图片档案一旦有了更动，便会执行相对应的任务。

**即时重整(LiveReload)**

Gulp也可以处理档案更动后，自动重新整理页面。我们需要修改watch任务，设置即时重整伺服器。

gulp.task('watch', function() {

// 建立即时重整伺服器

var server = livereload();

// 看守所有位在 dist/ 目录下的档案，一旦有更动，便进行重整

gulp.watch(['dist/\*\*']).on('change', function(file) {

server.changed(file.path);

});

});

为了让这个功能有效，除了伺服器之外，还需要安装并启用LiveReload的浏览器外挂。或者你也可以手动加上这个[片段程式码](http://feedback.livereload.com/knowledgebase/articles/86180-how-do-i-add-the-script-tag-manually-)。

**全部在一起**

这裡是完整的gulpfile:

// 载入外挂

var gulp = require('gulp'),

sass = require('gulp-ruby-sass'),

autoprefixer = require('gulp-autoprefixer'),

minifycss = require('gulp-minify-css'),

jshint = require('gulp-jshint'),

uglify = require('gulp-uglify'),

imagemin = require('gulp-imagemin'),

rename = require('gulp-rename'),

clean = require('gulp-clean'),

concat = require('gulp-concat'),

notify = require('gulp-notify'),

cache = require('gulp-cache'),

livereload = require('gulp-livereload');

// 样式

gulp.task('styles', function() {

return gulp.src('src/styles/main.scss')

.pipe(sass({ style: 'expanded', }))

.pipe(autoprefixer('last 2 version', 'safari 5', 'ie 8', 'ie 9', 'opera 12.1', 'ios 6', 'android 4'))

.pipe(gulp.dest('dist/styles'))

.pipe(rename({ suffix: '.min' }))

.pipe(minifycss())

.pipe(gulp.dest('dist/styles'))

.pipe(notify({ message: 'Styles task complete' }));

});

// 脚本

gulp.task('scripts', function() {

return gulp.src('src/scripts/\*\*/\*.js')

.pipe(jshint('.jshintrc'))

.pipe(jshint.reporter('default'))

.pipe(concat('main.js'))

.pipe(gulp.dest('dist/scripts'))

.pipe(rename({ suffix: '.min' }))

.pipe(uglify())

.pipe(gulp.dest('dist/scripts'))

.pipe(notify({ message: 'Scripts task complete' }));

});

// 图片

gulp.task('images', function() {

return gulp.src('src/images/\*\*/\*')

.pipe(cache(imagemin({ optimizationLevel: 3, progressive: true, interlaced: true })))

.pipe(gulp.dest('dist/images'))

.pipe(notify({ message: 'Images task complete' }));

});

// 清理

gulp.task('clean', function() {

return gulp.src(['dist/styles', 'dist/scripts', 'dist/images'], {read: false})

.pipe(clean());

});

// 预设任务

gulp.task('default', ['clean'], function() {

gulp.start('styles', 'scripts', 'images');

});

// 看手

gulp.task('watch', function() {

// 看守所有.scss档

gulp.watch('src/styles/\*\*/\*.scss', ['styles']);

// 看守所有.js档

gulp.watch('src/scripts/\*\*/\*.js', ['scripts']);

// 看守所有图片档

gulp.watch('src/images/\*\*/\*', ['images']);

// 建立即时重整伺服器

var server = livereload();

// 看守所有位在 dist/ 目录下的档案，一旦有更动，便进行重整

gulp.watch(['dist/\*\*']).on('change', function(file) {

server.changed(file.path);

});

});

你也可以在[gist](https://gist.github.com/markgoodyear/8497946#file-gulpfile-js)看整个gulpfile。我也将达到相同任务的Gruntfile放在同一个[gist](https://gist.github.com/markgoodyear/8497946#file-gruntfile-js)，方便做比较。

如果你有任何疑问或议题，请在文章下方留下评论或者可以在[Twitter](https://twitter.com/markgdyr)找到我。

文本由[markgdyr](https://twitter.com/markgdyr)提供翻译。英文地址: [Getting started with gulp](http://markgoodyear.com/2014/01/getting-started-with-gulp/)

# 雪碧图 gulp-css-spriter

gulp.task('sprite',function() {

var timestamp = +new Date();

//需要自动合并雪碧图的样式文件

return gulp.src('./src/sprite/\*.css')

.pipe(spriter({

// 生成的spriter的位置

'spriteSheet': './dist/assets/images/sprite'+timestamp+'.png',

// 生成样式文件图片引用地址的路径

// 如下将生产：backgound:url(../images/sprite20324232.png)

'pathToSpriteSheetFromCSS': '../images/sprite'+timestamp+'.png'

}))

.pipe(gulp.dest('dist/assets/css'))

.pipe(rename({suffix: '.min'}))

.pipe(minifycss())

//产出路径

.pipe(gulp.dest('./dist/assets/css'));

});

技巧 ：

gulp-css-spriter默认会对样式文件里，所有的background/background-image的图片合并，

但实际项目中，我们不是所有的图片都需要合并。

background-image: url(../slice/p1-3.png?\_\_spriter);//有?\_\_spriter后缀的合并

background-image: url(../slice/p-cao1.png); //不合并

修改下面文件可以按需合并。

node\_modules\gulp-css-spriter\lib\map-over-styles-and-transform-background-image-declarations.js

48行开始的if-else if代码块中，替换为下面代码：

// background-image always has a url 且判断url是否有?\_\_spriter后缀

if(transformedDeclaration.property === 'background-image' && /\?\_\_spriter/i.test(transformedDeclaration.value)) {

transformedDeclaration.value = transformedDeclaration.value.replace('?\_\_spriter','');

return cb(transformedDeclaration, declarationIndex, declarations);

}

// Background is a shorthand property so make sure `url()` is in there 且判断url是否有?\_\_spriter后缀

else if(transformedDeclaration.property === 'background' && /\?\_\_spriter/i.test(transformedDeclaration.value)) {

transformedDeclaration.value = transformedDeclaration.value.replace('?\_\_spriter','');

var hasImageValue = spriterUtil.backgroundURLRegex.test(transformedDeclaration.value);

if(hasImageValue) {

return cb(transformedDeclaration, declarationIndex, declarations);

}

}

# BUG 修复

# [npm报错lodash版本太低](http://segmentfault.com/q/1010000004094439)

npm install -g lodash

npm uninstall lodash

npm install lodash@latest.

之后虽然报错但还是能用

# Gulp 默认执行顺序错误

**注意：** 你的任务是否在这些前置依赖的任务完成之前运行了？请一定要确保你所依赖的任务列表中的任务都使用了正确的异步执行方式：使用一个 callback，或者返回一个 promise 或 stream。

即：每个【】中的任务都必须有return