**Gulp新手入门教程**

[Gulp](http://gulpjs.com/) 是一个自动化工具，前端开发者可以使用它来处理常见任务：

* 搭建web服务器
* 文件保存时自动重载浏览器
* 使用预处理器如Sass、LESS
* 优化资源，比如压缩CSS、JavaScript、压缩图片

当然Gulp能做的远不止这些。如果你够疯狂，你甚至可以使用它搭建一个静态页面生成器。Gulp真的足够强大，但你必须学会驾驭它。

这是这篇文章的主要目的。帮助你了解Gulp的基础用法，助你早日完成一统天下的大业。

在我们深入了解之前，我们先来说说为什么是Gulp。

**为什么是Gulp?**

类似Gulp的工具，我们通常称之为构建工具。如今最流行的两个构建工具是Gulp和Grunt。

已经有非常多的文章论证它们之前的异同，你随便搜一下就知道了。

主要区别是你如何使用他们构建自动化工作流。与Grunt相比，Gulp更加简洁，执行效率更高。

让我们继续学习Gulp并搭建一个简单的工作流。

**我们将要做的**

这篇文章的最后，你会拥有简单的一个工作流：

* 搭建本地web服务器
* 编译Sass
* 每当保存更改，自动刷新浏览器
* 优化&压缩资源

除此之外，你还将学会使用简单命令链式调用多个任务。

**安装Gulp**

安装Gulp之前你需要先安装Node.js。

如果你还没安装Node，你可以在 [这里](http://nodejs.cn/) 下载。

安装完Node.js，使用Terminal(终端，win下是cmd)用下面命令安装Gulp

$ sudo npm install gulp -g

只有mac用户才需要sudo命令，并且不要把$符号也复制进去，这不是你的菜。

npm install 是指定从Node Package Manager(npm 你怕毛)安装的命令。

-g 表示全局安装，这样你在电脑上任何位置都能只用gulp 命令。

Mac 用户需要管理员权限才能全局安装，所以需要sudo。

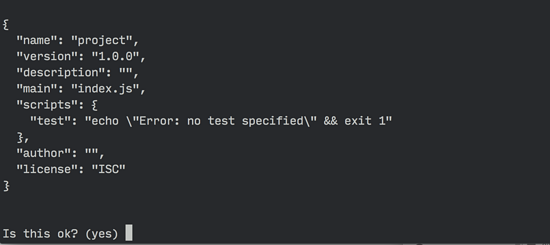
接下来使用Gulp创建项目。

**创建Gulp项目**

首先，我们新建一个project文件夹，并在该目录下执行 npm init 命令：

$ npm init

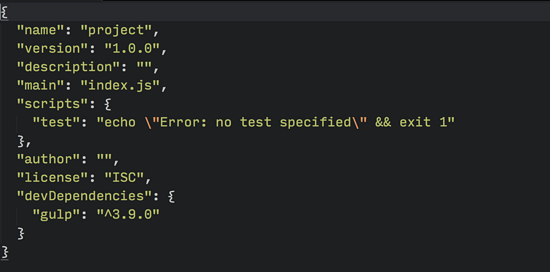
npm init命令会为你创建一个package.json文件，这个文件保存着这个项目相关信息。比如你用到的各种依赖（这里主要是插件）（终端会自动出现下面内容，这里先随便填就行）



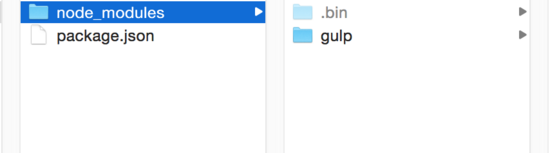
创建完之后，我们执行下面的命令：

$ npm install gulp --save-dev

这一次，我们局部安装Gulp。使用—save-dev，将通知计算机在package.json中添加gulp依赖。



执行完之后，gulp将创建node\_modules文件夹，里面有个gulp文件夹。

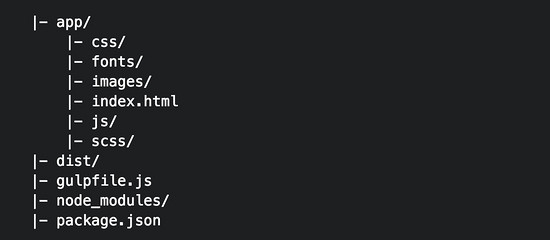


在正式开始之前，我们再来明确下项目的目录结构。

**目录结构**

Gulp非常之灵活，理解它的内部工作，你就能在项目中得心应手。

这篇文章，我们使用通用的webapp目录结构：



在这个结构中，我们使用app文件夹作为开发目录（所有的源文件都放在这下面），dist文件夹用来存放生产环境下的内容。

这些文件名，你想怎么起都行，但请务必记住你的目录结构。现在我们来创建gulpfile.js。

**第一个Gulp任务**

（你需要先在根目录下创建一个gulpfile.js文件）。

var gulp = require('gulp');

这行命令告知Node去node\_modules中查找gulp包，先局部查找，找不到就去全局环境中查找。找到之后就会赋值给gulp变量，然后我们就可以使用它了。

简单的任务如下所示：

gulp.task('task-name', function() {

// Stuff here

});

task-name 是给你的任务起的名字，稍后在命令行中执行gulp task-name，将运行该任务。

写个HelloWorld，是这样的：

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

console.log('Hello World!');

});

命令行中执行：

$ gulp hello

那么将会输出Hello World!。 够简单吧？

Gulp任务通常都会比这难一丁点，就一丁点。通常会包含两个特定的Gulp方法和一些列Gulp插件。

大概这样：

gulp.task('task-name', function () {

return gulp.src('source-files') // Get source files with gulp.src

.pipe(aGulpPlugin()) // Sends it through a gulp plugin

.pipe(gulp.dest('destination')) // Outputs the file in the destination folder

})

正如你所见，两个Gulp方法，src,dest，一进一出[捂脸.jpg]。下面用编译Sass来举栗子。

**Gulp执行预处理**

我们使用gulp-sass插件来编译Sass。安装插件的步骤是这样的：

* 使用npm install 命令安装 $ npm install gulp-sass --save-dev
* 在gulpfile中引入插件，用变量保存 var gulp = require('gulp');
* // Requires the gulp-sass plugin
* var sass = require('gulp-sass');
* 在任务中使用 gulp.task('sass', function(){
* return gulp.src('source-files')
* .pipe(sass()) // Using gulp-sass
* .pipe(gulp.dest('destination'))
* });

我们需要给sass任务提供源文件和输出位置。所以我们先在项目中创建app/scss文件夹，里面有个styles.scss文件。这个文件将在gulp.src中用到。

sass处理之后，我们希望它生成css文件并产出到app/css目录下，可以这样写：

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

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

.pipe(sass()) // Converts Sass to CSS with gulp-sass

.pipe(gulp.dest('app/css'))

});

测试一下：

// styles.scss

.testing {

width: percentage(5/7);

}

使用Terminal执行gulp sass,你将看到app/css/styles.css文件下会有下面的代码：

/\* styles.css \*/

.testing {

width: 71.42857%;

}

styles.css是gulp智动为我们生成的。percentage 是Sass的方法。

使用Sass就这么简单。但是通常我们不止有一个scss文件。这时候可以使用Node通配符。

**Node中的通配符**

通配符是一种匹配模式，允许你匹配到多个文件。不止是Node，很多平台都有，有点像正则表达式。

使用通配符，计算机检查文件名和路径进行匹配。

大部分时候，我们只需要用到下面4种匹配模式：

1. \*.scss ： \* 号匹配当前目录任意文件，所以这里 \*.scss 匹配当前目录下所有scss文件
2. \*\*/\*.scss ：匹配当前目录及其子目录下的所有scss文件。
3. !not-me.scss ：！号移除匹配的文件，这里将移除not-me.scss
4. \*.+(scss|sass) ：+号后面会跟着圆括号，里面的元素用|分割，匹配多个选项。这里将匹配scss和sass文件。

那么还是上面的栗子，改造一下：

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

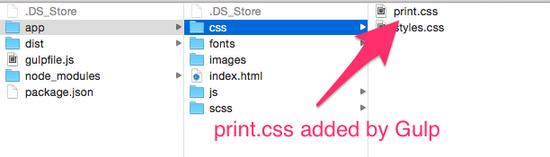
return gulp.src('app/scss/\*\*/\*.scss') // Gets all files ending with .scss in app/scss and children dirs

.pipe(sass())

.pipe(gulp.dest('app/css'))

})

任何app下的scss文件，在执行命令之后将生成对应的css文件存放到相应路径。(智动…)



现在我们能处理多个文件了，但是不想每次都要执行命令，怎么办？Gulp就是为懒人而生的，我们可以使用watch命令，自动检测并执行。

**监听Sass文件**

Gulp提供watch方法给我们，语法如下：

// Gulp watch syntax

gulp.watch('files-to-watch', ['tasks', 'to', 'run']);

将上面的栗子再改下：

// Gulp watch syntax

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

通常我们监听的还不只是一个文件，把它变成一个任务：

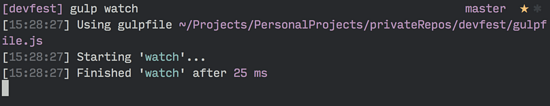
gulp.task('watch', function(){

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

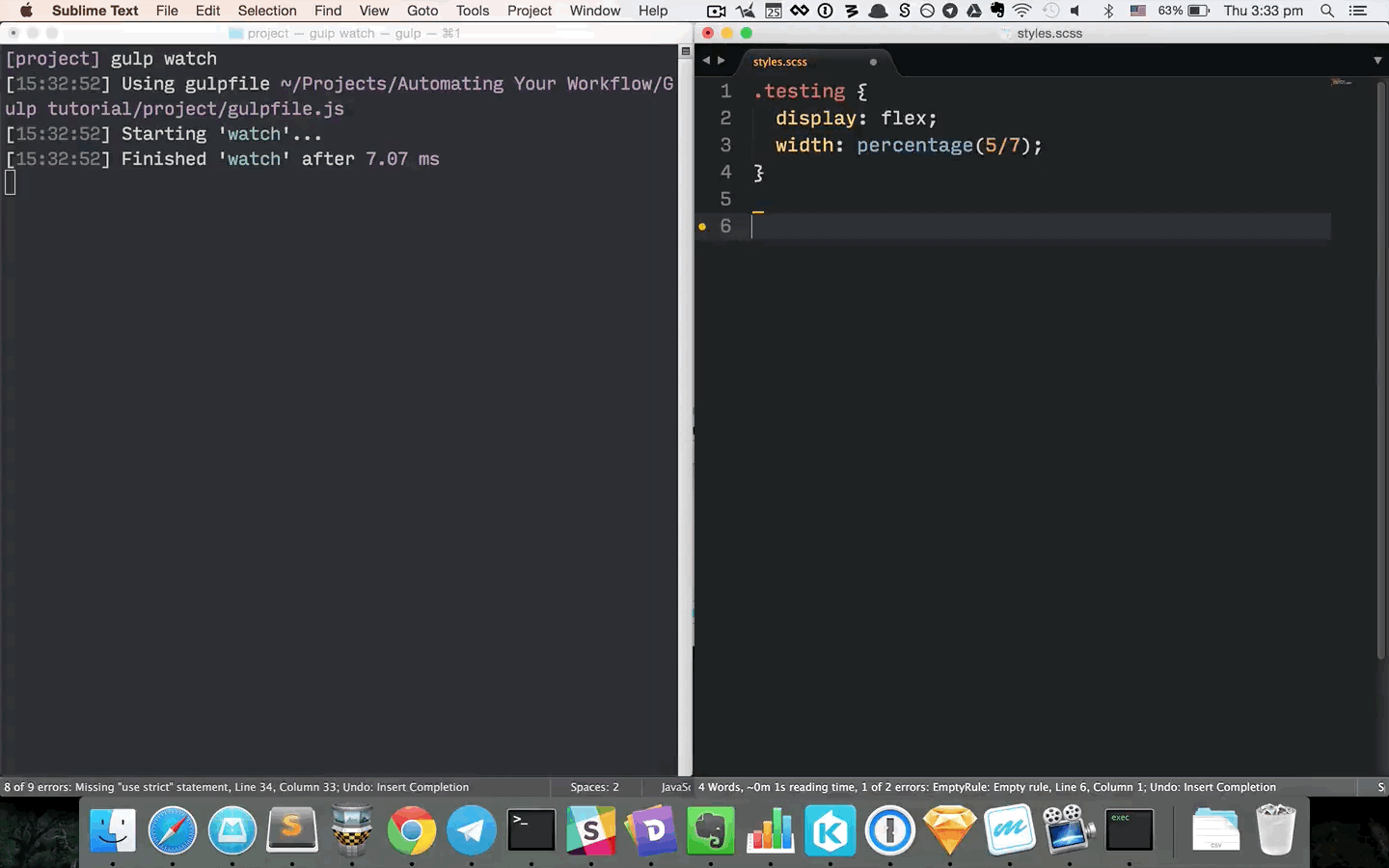
// Other watchers

})

执行gulp watch命令。



有了监听，每次修改文件，Gulp都将自动为我们执行任务。编译前若是.scss文件在文件夹内，编译后还会输出到输出目录的相应文件夹，若没有此文件夹则会新建。



还不够，修改完直接帮我刷新浏览器行吗，我不想每次都要手动按Command + R;

**使用Browser Sync自动刷新**

Browser Sync 帮助我们搭建简单的本地服务器并能实时刷新浏览器，它还能 [同时刷新多个设备](http://addyosmani.com/blog/browser-sync/)

新插件？记住！安装，引入，使用。

$ npm install browser-sync --save-dev

这里没有gulp-前缀，因为browser-sync支持Gulp，所以没有人专门去搞一个给Gulp用。

var browserSync = require('browser-sync');

我们创建一个broswerSync任务，我们需要告知它，根目录在哪里。

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

browserSync({

server: {

baseDir: 'app'

},

})

})

我们稍微修改一下之前的代码，让每次css文件更改都刷新一下浏览器：

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

return gulp.src('app/scss/\*\*/\*.scss') // Gets all files ending with .scss in app/scss

.pipe(sass())

.pipe(gulp.dest('app/css'))

.pipe(browserSync.reload({

stream: true

}))

});

现在我们配置好Broswer Sync了，我们需要运行这两个命令。

我们可以在watch任务之前告知Gulp，先把browserSync和Sass任务执行了再说。

语法如下：

gulp.task('watch', ['array', 'of', 'tasks', 'to', 'complete','before', 'watch'], function (){

// ...

})

应用下来是这样：

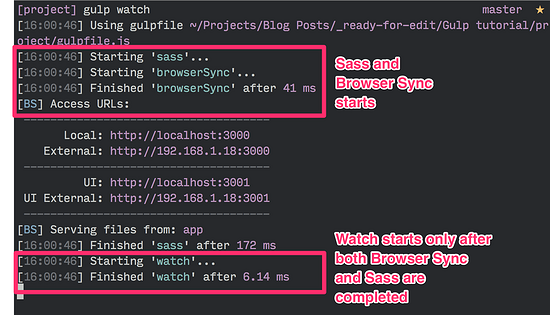
gulp.task('watch', ['browserSync', 'sass'], function (){

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

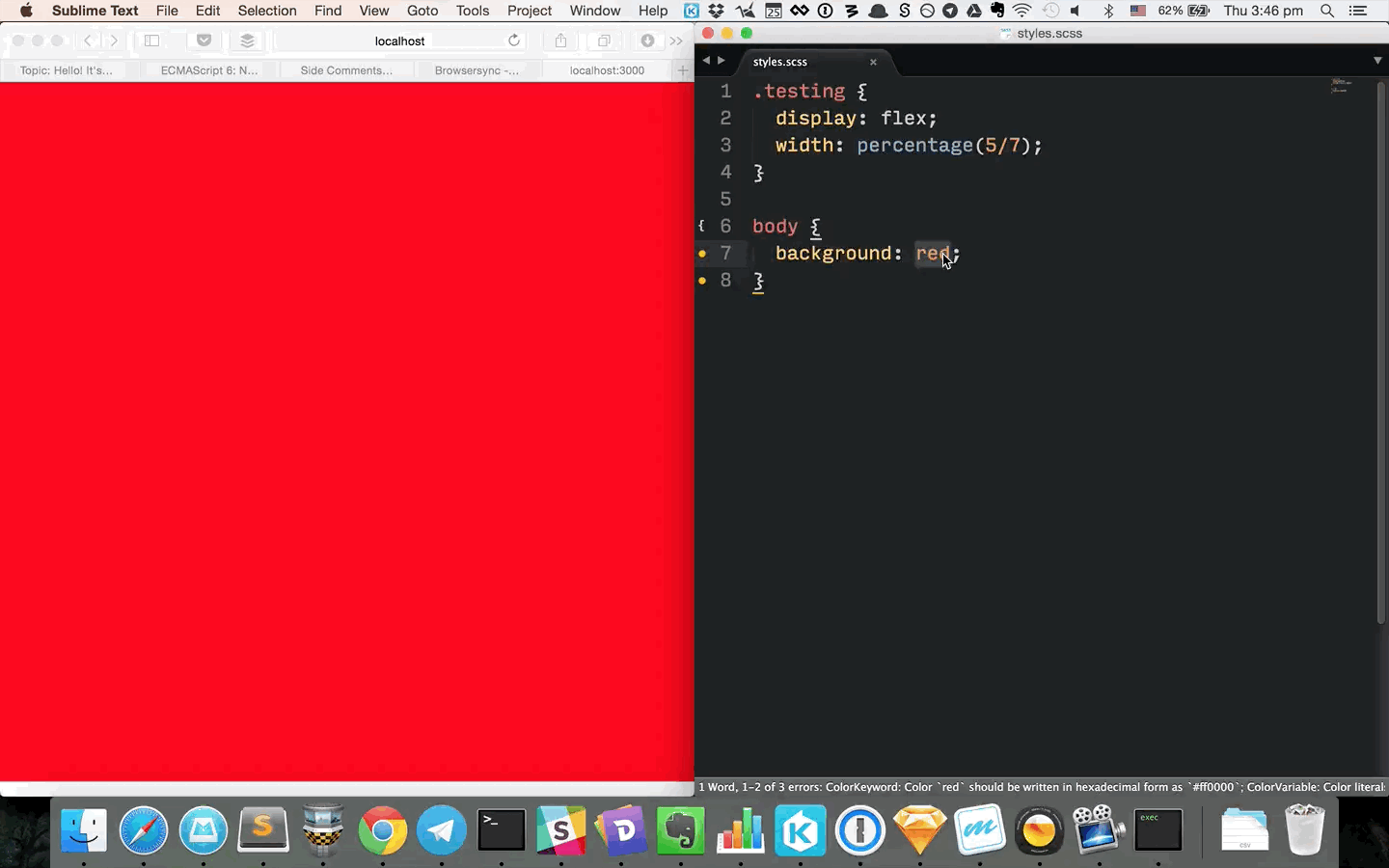
// Other watchers

})

现在你执行gulp watch命令，在执行完browserSync和Sass，才会开始监听。



并且现在浏览器的显示的页面为app/index.html。你修改了styles.scss之后，浏览器将自动属性页面。



不止是scss修改的时候需要刷新浏览器吧？再改改：

gulp.task('watch', ['browserSync', 'sass'], function (){

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

// Reloads the browser whenever HTML or JS files change

gulp.watch('app/\*.html', browserSync.reload);

gulp.watch('app/js/\*\*/\*.js', browserSync.reload);

});

到目前为止，我们做了下面三件事：

* 可运转的web开发服务
* 使用Sass预处理器
* 自动刷新浏览器

接下来说说优化方面的技巧

**优化CSS和JavaScript文件**

说到优化的时候，我们需要想到：压缩，拼接。也就是减少体积和HTTP次数。

开发者面临的主要问题是很难按照正确的顺序合并文件。

<body>

<!-- other stuff -->

<script src="js/lib/a-library.js"></script>

<script src="js/lib/another-library.js"></script>

<script src="js/main.js"></script>

</body>

由于文件路径的的不同，使用 [https://www.npmjs.com/package/gulp-concat](http://w3ctrain.com/2015/12/22/gulp-for-beginners/gulp-concatenate) 等插件非常困难。

庆幸的是， [gulp-useref](https://www.npmjs.com/package/gulp-useref) 解决了这个问题。

gulp-useref会将多个文件拼接成单一文件，并输出到相应目录。

<!-- build:<type> <path> -->

... HTML Markup, list of script / link tags.

<!-- endbuild -->

可以是js,css,或者remove。如果你设为remove,Gulp将不会生成文件。

指定产出路径。

我们想最终产出main.min.js。可以这样写：

<!--build:js js/main.min.js -->

<script src="js/lib/a-library.js"></script>

<script src="js/lib/another-library.js"></script>

<script src="js/main.js"></script>

<!-- endbuild -->

我们来安装gulp-useref。

$ npm install gulp-useref --save-dev

引用

var useref = require('gulp-useref');

使用起来非常简单：

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

return gulp.src('app/\*.html')

.pipe(useref())

.pipe(gulp.dest('dist'));

});

（新版的gulp-useref已经不需要写多余的useref.assets了）

执行useref命令，Gulp将合并三个script标签成一个文件，并保存到dist/js/main.min.js。

合并完之后，我们再来压缩。使用gulp-uglify插件。

安装

$ npm install gulp-uglify --save-dev

使用

// Other requires...

var uglify = require('gulp-uglify');

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

return gulp.src('app/\*.html')

.pipe(uglify()) // Uglifies Javascript files

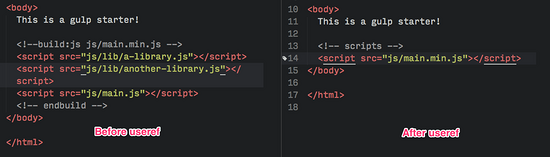
.pipe(useref())

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

});

搞定！

注意：执行完useref后，html中的script路径将只剩下main.min.js。



王祖蓝：完美~

gulp-useref同样可以用在css上。除了压缩，需要区分，其它内容同js一样。所以我们使用gulp-if来做不同处理。

使用gulp-minify-css压缩css。

$ npm install gulp-if gulp-minify-css --save-dev

应用

var gulpIf = require('gulp-if');

var minifyCSS = require('gulp-minify-css');

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

return gulp.src('app/\*.html')

// Minifies only if it's a CSS file

.pipe(gulpIf('\*.css', minifyCSS()))

// Uglifies only if it's a Javascript file

.pipe(gulpIf('\*.js', uglify()))

.pipe(useref())

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

});

搞定！

再说说如何压缩图片。同样easy。

**优化图片**

使用gulp-imagemin插件。

$ npm install gulp-imagemin --save-dev

引入，使用

var imagemin = require('gulp-imagemin');

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

return gulp.src('app/images/\*\*/\*.+(png|jpg|gif|svg)')

.pipe(imagemin())

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

});

（所有的gulp插件都是有相关参数可以配置，如果需要，请自行查看。）

压缩图片可能会占用较长时间，使用 [gulp-cache](https://www.npmjs.com/package/gulp-cache) 插件可以减少重复压缩。

$ npm install gulp-cache --save-dev

引入、使用

var cache = require('gulp-cache');

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

return gulp.src('app/images/\*\*/\*.+(png|jpg|jpeg|gif|svg)')

// Caching images that ran through imagemin

.pipe(cache(imagemin({

interlaced: true

})))

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

});

接下来，我们说说发布流程。

**清理生成文件**

由于我们是自动生成文件，我们不想旧文件掺杂进来。

使用 [del](https://www.npmjs.com/package/del)

npm install del --save-dev

引入、使用

var del = require('del');

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

del('dist');

});

但是我们又不想图片被删除（图片改动的几率不大）,启用新的任务。

gulp.task('clean:dist', function(callback){

del(['dist/\*\*/\*', '!dist/images', '!dist/images/\*\*/\*'], callback)

});

这个任务会删除，除了images/文件夹，dist下的任意文件。为了知道clean:dist任务什么时候完成，我们需要提供callback参数。

在某些时候我们还是需要清除图片，所以clean任务我们还需要保留。

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

del('dist');

return cache.clearAll(callback);

})

噢，我真的是废话太多了。把我们学到的组合到一块吧！

**组合Gulp任务**

废话了这么多，我们主要有两条线路。

第一条是开发过程，我们便以Sass，监听文件，刷新浏览器。

第二条是优化，我们优化CSS,JavaScript,压缩图片，并把资源从app移动到dist。

开发任务我们上面的watch已经组装好了。

gulp.task('watch', ['browserSync', 'sass'], function (){

// ... watchers

})

我们也做一个来执行第二条线路。

gulp.task('build', [`clean`, `sass`, `useref`, `images`, `fonts`], function (){

console.log('Building files');

})

但是这样Gulp会同时触发 [] 的事件。我们要让clean在其他任务之前完成。

这里要用到 [RunSequence](https://www.npmjs.com/package/run-sequence) 。

$ npm install run-sequence --save-dev

用法如下：

var runSequence = require('run-sequence');

gulp.task('task-name', function(callback) {

runSequence('task-one', 'task-two', 'task-three', callback);

});

执行task-name时，Gulp会按照顺序执行task-one,task-two,task-thre。

RunSequence也允许你同时执行多个任务。

gulp.task('task-name', function(callback) {

runSequence('task-one', ['tasks','two','run','in','parallel'], 'task-three', callback);

});

改造我们的代码：

gulp.task('build', function (callback) {

runSequence('clean:dist',

['sass', 'useref', 'images', 'fonts'],

callback

)

})

开发任务我们也用runSequence:

gulp.task('default', function (callback) {

runSequence(['sass','browserSync', 'watch'],

callback

)

})

default? 如果你的任务名字叫做default，那么只需要输入gulp命令即可执行。

这里是我们最终的 [代码仓库](https://github.com/zellwk/gulp-starter-csstricks) 。

**总结**

上面的的内容搭建了一个基本的Gulp工作流。还有更精彩的内容等着你去开发。这里提供些插件：

开发过程：

* 使用 [Autoprefixer](https://www.npmjs.com/package/gulp-autoprefixer) ，你不再需要写CSS浏览器内核前缀
* 增加 [Sourcemaps](https://www.npmjs.com/package/gulp-sourcemaps) ，让你更方便的调试Sass,coffeescript
* 使用 [sprity](https://www.npmjs.com/package/sprity) 创建精灵图
* [gulp-changed](https://www.npmjs.com/package/gulp-changed) 只允许通过修改的文件
* [Babel](https://www.npmjs.com/package/gulp-babel) 或 [Traceur](https://www.npmjs.com/package/gulp-traceur) 写ES6
* [Browserify](http://browserify.org/) , [webpack](https://github.com/webpack/webpack) , [jspm](https://github.com/jspm) 模块化JavaScript
* [Handlebars](https://www.npmjs.com/package/gulp-handlebars) , [Swing](https://www.npmjs.com/package/gulp-swig) 模块化Html
* [require-dir](https://www.npmjs.com/package/require-directory) 分割gulpfile成多个文件
* [gulp-moderinizr](https://www.npmjs.com/package/gulp-modernizr) 自动生成Modernizr脚本

优化：

* [unCSS](https://www.npmjs.com/package/gulp-uncss) 移除多余的CSS
* [CSSO](https://www.npmjs.com/package/gulp-csso) 更深入地优化CSS
* [Critical](https://www.npmjs.com/package/critical) 生成行内CSS

除了开发和优化过程，你可以使用gulp-jasmine写JavaScript单元测试，甚至使用gulp-rync直接部署dist文件到生产环境。

**译者的话**

花了点时间去学习Gulp，这篇文章是我找到比较全面的，非常适合新手入门。

Gulp其实非常简单，多玩一下，你就能完全掌握它了，有能力的话，可以尝试开发Gulp插件。

我在前端收藏夹 以及 [使用gulp生成sprites图片和样式表](http://www.w3ctrain.com/2015/12/09/generating-sprites-with-gulp/) 都有实践过。不过正如作者 [Zell Liew](http://www.zell-weekeat.com/) 所说，Gulp的强大之处远不止这些，所以他出了本电子书 [automating your workflow](http://zell-weekeat.com/automate-your-workflow/) ，可以帮助你搭建属于自己的工作流，剩下时间去喝咖啡，有兴趣的可以去购买，支持作者。