gulp 中文文档

- 入门指南 - 如何开始使用 gulp
- API 文档 - 学习 gulp 的输入和输出方式
- CLI 文档 - 学习如何执行任务（task）以及如何使用一些编译工具
- 编写插件 - 所以，你已经在写一个 gulp 插件了么？ 去这儿看一些基本文档，并了解下什么样的事情不应该做

常见问题

常见问题，请查看 FAQ。

秘籍

社区中对于常用的一些 gulp 应用场景已经有了一些现成的秘籍

还是有问题？

到 StackOverflow 发一个带有 #gulp 标签的问题，或者在 Freenode 上的 #gulpjs IRC 频道寻求帮助。

书籍

- Developing a gulp Edge

文章（英文）

- Tagtree intro to gulp video
- Introduction to node.js streams
- Video introduction to node.js streams
- Getting started with gulp (by @markgdyr)
- A cheatsheet for gulp
- Why you shouldn't create a gulp plugin (or, how to stop worrying and learn to love existing node packages)
- Inspiration (slides) about why gulp was made
- Building With Gulp
- Gulp - The Basics (screencast)
- Get started with gulp (video series)
- Optimize your web code with gulp

例子

- Web Starter Kit gulpfile

License

gulp API 文档

gulp.src(globs[, options])

输出（Emits）符合所提供的匹配模式（glob）或者匹配模式的数组（array of globs）的文件。 将返回一个 Vinyl files 的 stream 它可以被 piped 到别的插件中。

```
gulp.src('client/templates/*.jade')

  .pipe(jade())

  .pipe(minify())

  .pipe(gulp.dest('build/minified_templates'));
```

`glob` 请参考 node-glob 语法 或者，你也可以直接写文件的路径。

globs

类型： `String` 或 `Array`

所要读取的 glob 或者包含 globs 的数组。

options

类型： `Object`

通过 glob-stream 所传递给 node-glob 的参数。

除了 node-glob 和 glob-stream 所支持的参数外，gulp 增加了一些额外的选项参数：

options.buffer

类型： `Boolean` 默认值： `true`

如果该项被设置为 `false`，那么将会以 stream 方式返回 `file.contents` 而不是文件 buffer 的形式。这在处理一些大文件的时候将会很有用。**注意：**插件可能并不会实现对 stream 的支持。

options.read

类型： `Boolean` 默认值： `true`

如果该项被设置为 `false`， 那么 `file.contents` 会返回空值（null），也就是并不会去读取文件。

options.base

类型： `String` 默认值： 将会加在 glob 之前 (请看 glob2base)

如, 请想像一下在一个路径为 `client/js/somedir` 的目录中，有一个文件叫 `somefile.js`：

```
gulp.src('client/js/**/*.js') // 匹配 'client/js/somedir/somefile.js' 并且将 `base` 解析为 `client/js/`

  .pipe(minify())

  .pipe(gulp.dest('build')); // 写入 'build/somedir/somefile.js'
```

```
gulp.src('client/js/**/*.js', { base: 'client' })

  .pipe(minify())

  .pipe(gulp.dest('build')); // 写入 'build/js/somedir/somefile.js'
```

gulp.dest(path[, options])

能被 pipe 进来，并且将会写文件。并且重新输出（emits）所有数据，因此你可以将它 pipe 到多个文件夹。如果某文件夹不存在，将会自动创建它。

```
gulp.src('./client/templates/*.jade')
```

```
  .pipe(jade())

  .pipe(gulp.dest('./build/templates'))

  .pipe(minify())

  .pipe(gulp.dest('./build/minified_templates'));
```

文件被写入的路径是以所给的相对路径根据所给的目标目录计算而来。类似的，相对路径也可以根据所给的 base 来计算。 请查看上述的 `gulp.src` 来了解更多信息。

path

类型： `String` or `Function`

文件将被写入的路径（输出目录）。也可以传入一个函数，在函数中返回相应路径，这个函数也可以由 vinyl 文件实例 来提供。

options

类型： `Object`

options.cwd

类型： `String` 默认值： `process.cwd()`

输出目录的 `cwd` 参数，只在所给的输出目录是相对路径时候有效。

options.mode

类型： `String` 默认值： `0777`

八进制权限字符，用以定义所有在输出目录中所创建的目录的权限。

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

定义一个使用 Orchestrator 实现的任务（task）。

```
gulp.task('somename', function() {

  // 做一些事
```

```
});
```

name

任务的名字，如果你需要在命令行中运行你的某些任务，那么，请不要在名字中使用空格。

deps

类型： `Array`

一个包含任务列表的数组，这些任务会在你当前任务运行之前完成。

```
gulp.task('mytask', ['array', 'of', 'task', 'names'], function() {
  // 做一些事
});
```

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。

fn

该函数定义任务所要执行的一些操作。通常来说，它会是这种形式：`gulp.src().pipe(someplugin())`。

异步任务支持

任务可以异步执行，如果 `fn` 能做到以下其中一点：

接受一个 callback

```
// 在 shell 中执行一个命令
var exec = require('child_process').exec;

gulp.task('jekyll', function(cb) {
```

```
  // 编译 Jekyll
  exec('jekyll build', function(err) {
    if (err) return cb(err); // 返回 error
    cb(); // 完成 task
  });
});
```

返回一个 stream

```
gulp.task('somename', function() {
  var stream = gulp.src('client/**/*.js')
    .pipe(minify())
    .pipe(gulp.dest('build'));
  return stream;
});
```

返回一个 promise

```
var Q = require('q');

gulp.task('somename', function() {
  var deferred = Q.defer();

  // 执行异步的操作
  setTimeout(function() {
```

```
    deferred.resolve();

  }, 1);


  return deferred.promise;

});
```

注意： 默认的，task 将以最大的并发数执行，也就是说，gulp 会一次性运行所有的 task 并且不做任何等待。如果你想要创建一个序列化的 task 队列，并以特定的顺序执行，你需要做两件事：

- 给出一个提示，来告知 task 什么时候执行完毕，
- 并且再给出一个提示，来告知一个 task 依赖另一个 task 的完成。

对于这个例子，让我们先假定你有两个 task，"one" 和 "two"，并且你希望它们按照这个顺序执行：

1. 在 "one" 中，你加入一个提示，来告知什么时候它会完成：可以再完成时候返回一个 callback，或者返回一个 promise 或 stream，这样系统会去等待它完成。
2. 在 "two" 中，你需要添加一个提示来告诉系统它需要依赖第一个 task 完成。

因此，这个例子的实际代码将会是这样：

```
var gulp = require('gulp');


// 返回一个 callback，因此系统可以知道它什么时候完成

gulp.task('one', function(cb) {

    // 做一些事 -- 异步的或者其他的
```

```
    cb(err); // 如果 err 不是 null 或 undefined，则会停止执行，且注意，这样代表执行失败了
});

// 定义一个所依赖的 task 必须在这个 task 执行之前完成
gulp.task('two', ['one'], function() {
    // 'one' 完成后
});

gulp.task('default', ['one', 'two']);
```

gulp.watch(glob [, opts], tasks) 或 gulp.watch(glob [, opts, cb])

监视文件，并且可以在文件发生改动时候做一些事情。它总会返回一个 EventEmitter 来发射（emit） `change` 事件。

gulp.watch(glob[, opts], tasks)

glob

类型： `String` or `Array`

一个 glob 字符串，或者一个包含多个 glob 字符串的数组，用来指定具体监控哪些文件的变动。

opts

类型： `Object`

传给 `gaze` 的参数。

tasks

类型： `Array`

需要在文件变动后执行的一个或者多个通过 `gulp.task()` 创建的 task 的名字，

```
var watcher = gulp.watch('js/**/*.js', ['uglify','reload']);

watcher.on('change', function(event) {

  console.log('File ' + event.path + ' was ' + event.type + ',
 running tasks...');

});
```

gulp.watch(glob[, opts, cb])

glob

类型： `String` or `Array`

一个 glob 字符串，或者一个包含多个 glob 字符串的数组，用来指定具体监控哪些文件的变动。

opts

类型： `Object`

传给 `gaze` 的参数。

cb(event)

类型： `Function`

每次变动需要执行的 callback。

```
gulp.watch('js/**/*.js', function(event) {
```

```
  console.log('File ' + event.path + ' was ' + event.type + ',
 running tasks...');
```

```
});
```

callback 会被传入一个名为 `event` 的对象。这个对象描述了所监控到的变动：

event.type

类型： `String`

发生的变动的类型：`added`, `changed` 或者 `deleted`。

event.path

类型： `String`

触发了该事件的文件的路径。

gulp 命令行（CLI）文档

参数标记

gulp 只有你需要熟知的参数标记，其他所有的参数标记只在一些任务需要的时候使用。

- `-v` 或 `--version` 会显示全局和项目本地所安装的 gulp 版本号
- `--require <module path>` 将会在执行之前 reqiure 一个模块。这对于一些语言编译器或者需要其他应用的情况来说来说很有用。你可以使用多个`--require`
- `--gulpfile <gulpfile path>` 手动指定一个 gulpfile 的路径，这在你有很多个 gulpfile 的时候很有用。这也会将 CWD 设置到该 gulpfile 所在目录

- `--cwd <dir path>` 手动指定 CWD。定义 gulpfile 查找的位置，此外，所有的相应的依赖（require）会从这里开始计算相对路径
- `-T` 或 `--tasks` 会显示所指定 gulpfile 的 task 依赖树
- `--tasks-simple` 会以纯文本的方式显示所载入的 gulpfile 中的 task 列表
- `--color` 强制 gulp 和 gulp 插件显示颜色，即便没有颜色支持
- `--no-color` 强制不显示颜色，即便检测到有颜色支持
- `--silent` 禁止所有的 gulp 日志

命令行会在 process.env.INIT_CW 中记录它是从哪里被运行的。

Task 特定的参数标记

请参考 StackOverflow 了解如何增加任务特定的参数标记。

Tasks

Task 可以通过 `gulp <task> <othertask>` 方式来执行。如果只运行 `gulp` 命令，则会执行所注册的名为 `default` 的 task，如果没有这个 task，那么 gulp 会报错。

编译器

你可以在 interpret 找到所支持的语言列表。如果你想要增加一个语言的支持，请在这里提交一个 pull request 或者 issue。

编写插件

如果你打算自己写一个 Gulp 插件，为了节约你的时间，你可以先完整地阅读下这个文档。

- 导览 (必读)
- 使用 buffer
- 使用 stream 来处理
- 测试

它要做什么？

流式处理文件对象（Streaming file objects）

gulp 插件总是返回一个 object mode 形式的 stream 来做这些事情：

1. 接收 vinyl File 对象
2. 输出 vinyl File 对象

这通常被叫做 transform streams (有时候也叫做 through streams)。transform streams 是可读又可写的，它会对传给它的对象做一些转换的操作。

修改文内容

Vinyl 文件可以通过三种不同形式来访问文件内容：

- Streams
- Buffers
- 空 (null) - 对于删除, 清理, 等操作来说，会很有用，因为这时候内容是不需要处理的。

有用的资源

- File object
- PluginError
- event-stream
- BufferStream
- gulp-util

插件范例

- sindresorhus' gulp plugins
- Fractal's gulp plugins
- gulp-replace

关于 stream

如果你不熟悉 stream，你可以阅读这些来

- https://github.com/substack/stream-handbook (必读)
- http://nodejs.org/api/stream.html

其他的一些为 gulp 创建的和使用的，但又并非通过 stream 去处理的库，在 npm 上都会被打上 gulpfriendly 标签。