**1.快速入门**

Grunt和 Grunt 插件是通过 [npm](https://npmjs.org/) 安装并管理的，npm是 [Node.js](http://nodejs.org/) 的包管理器。

*Grunt 0.4.x 必须配合Node.js >= 0.8.0版本使用。；奇数版本号的 Node.js 被认为是不稳定的开发版。*

在安装 Grunt 前，请确保当前环境中所安装的 [npm](https://npmjs.org/) 已经是最新版本，执行 npm update -g npm 指令进行升级（在某些系统中可能需要 sudo 指令）。

如果你已经安装了 Grunt，现在需要参考一些文档手册，那就请看一看 [Gruntfile 实例](http://www.gruntjs.net/sample-gruntfile) 和如何 [配置任务](http://www.gruntjs.net/configuring-tasks)吧。

**[安装 CLI](http://www.gruntjs.net/getting-started" \l "cli)**

**还在使用 Grunt 0.3 版本吗？请查看**[**Grunt 0.3 注意事项**](http://www.gruntjs.net/upgrading-from-0.3-to-0.4#grunt-0.3-notes)

在继续学习前，你需要先将Grunt命令行（CLI）安装到全局环境中。安装时可能需要使用sudo（针对OSX、\*nix、BSD等系统中）权限或者作为管理员（对于Windows环境）来执行以下命令。

npm install -g grunt-cli

上述命令执行完后，grunt 命令就被加入到你的系统路径中了，以后就可以在任何目录下执行此命令了。

注意，安装grunt-cli并不等于安装了 Grunt！Grunt CLI的任务很简单：调用与Gruntfile在同一目录中 Grunt。这样带来的好处是，允许你在同一个系统上同时安装多个版本的 Grunt。

这样就能让多个版本的 Grunt 同时安装在同一台机器上。

[**CLI 是如何工作的**](http://www.gruntjs.net/getting-started#cli)

每次运行grunt 时，他就利用node提供的require()系统查找本地安装的 Grunt。正是由于这一机制，你可以在项目的任意子目录中运行grunt 。

如果找到一份本地安装的 Grunt，CLI就将其加载，并传递Gruntfile中的配置信息，然后执行你所指定的任务。为了更好的理解 Grunt CLI的执行原理，请[阅读源码](https://github.com/gruntjs/grunt-cli/blob/master/bin/grunt)。

**[拿一份现有的 Grunt 项目练手](http://www.gruntjs.net/getting-started" \l "grunt)**

假定Grunt CLI已经正确安装，并且已经有一份配置好package.json 和 Gruntfile 文件的项目了，接下来就很容易拿Grunt练手了：

1. 将命令行的当前目录转到项目的根目录下。
2. 执行npm install命令安装项目依赖的库。
3. 执行 grunt 命令。

OK，就是这么简单。还可以通过grunt --help 命令列出所有已安装的Grunt任务（task），但是一般更建议去查看项目的文档以获取帮助信息。

[**准备一份新的 Grunt 项目**](http://www.gruntjs.net/getting-started#grunt)

一般需要在你的项目中添加两份文件：package.json 和 Gruntfile。

**package.json**: 此文件被[npm](https://npmjs.org/)用于存储项目的元数据，以便将此项目发布为npm模块。你可以在此文件中列出项目依赖的grunt和Grunt插件，放置于[devDependencies](https://docs.npmjs.com/files/package.json#devdependencies)配置段内。

**Gruntfile**: 此文件被命名为 Gruntfile.js 或 Gruntfile.coffee，用来配置或定义任务（task）并加载Grunt插件的。 **此文档中提到的 Gruntfile 其实说的是一个文件，文件名是 Gruntfile.js 或Gruntfile.coffee**。

**[package.json](http://www.gruntjs.net/getting-started" \l "package.json)**

package.json应当放置于项目的根目录中，与Gruntfile在同一目录中，并且应该与项目的源代码一起被提交。在上述目录(package.json所在目录)中运行npm install将依据package.json文件中所列出的每个依赖来自动安装适当版本的依赖。

下面列出了几种为你的项目创建package.json文件的方式：

* 大部分 [grunt-init](http://www.gruntjs.net/project-scaffolding) 模版都会自动创建特定于项目的package.json文件。
* [npm init](https://docs.npmjs.com/cli/init)命令会创建一个基本的package.json文件。
* 复制下面的案例，并根据需要做扩充，参考此[说明](https://docs.npmjs.com/files/package.json).

{

"name": "my-project-name",

"version": "0.1.0",

"devDependencies": {

"grunt": "~0.4.5",

"grunt-contrib-jshint": "~0.10.0",

"grunt-contrib-nodeunit": "~0.4.1",

"grunt-contrib-uglify": "~0.5.0"

}

}

**[安装Grunt 和 grunt插件](http://www.gruntjs.net/getting-started" \l "grunt-grunt)**

向已经存在的package.json 文件中添加Grunt和grunt插件的最简单方式是通过npm install <module> --save-dev命令。此命令不光安装了<module>，还会自动将其添加到[devDependencies](https://docs.npmjs.com/files/package.json#devdependencies) 配置段中，遵循[tilde version range](https://npmjs.org/doc/misc/semver.html#Ranges)格式。

例如，下面这条命令将安装Grunt最新版本到项目目录中，并将其添加到devDependencies内：

npm install grunt --save-dev

同样，grunt插件和其它node模块都可以按相同的方式安装。下面展示的实例就是安装 JSHint 任务模块：

npm install grunt-contrib-jshint --save-dev

在 [Grunt 插件](http://www.gruntjs.net/plugins) 页面可以看到当前可用的 Grunt 插件，他们可以直接在项目中安装并使用。

安装插件之后，请务必确保将更新之后的 package.json 文件提交到项目仓库中。

**[Gruntfile](http://www.gruntjs.net/getting-started" \l "gruntfile)**

Gruntfile.js 或 Gruntfile.coffee 文件是有效的 JavaScript 或 CoffeeScript 文件，应当放在你的项目根目录中，和package.json文件在同一目录层级，并和项目源码一起加入源码管理器。

Gruntfile由以下几部分构成：

* "wrapper" 函数
* 项目与任务配置
* 加载grunt插件和任务
* 自定义任务

[**Gruntfile文件案例**](http://www.gruntjs.net/getting-started#gruntfile)

在下面列出的这个 Gruntfile 中，package.json文件中的项目元数据（metadata）被导入到 Grunt 配置中， [grunt-contrib-uglify](http://github.com/gruntjs/grunt-contrib-uglify) 插件中的uglify 任务（task）被配置为压缩（minify）源码文件并依据上述元数据动态生成一个文件头注释。当在命令行中执行 grunt 命令时，uglify 任务将被默认执行。

module.exports = function(grunt) {

// Project configuration.

grunt.initConfig({

pkg: grunt.file.readJSON('package.json'),

uglify: {

options: {

banner: '/\*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> \*/\n'

},

build: {

src: 'src/<%= pkg.name %>.js',

dest: 'build/<%= pkg.name %>.min.js'

}

}

});

// 加载包含 "uglify" 任务的插件。

grunt.loadNpmTasks('grunt-contrib-uglify');

// 默认被执行的任务列表。

grunt.registerTask('default', ['uglify']);

};

前面已经向你展示了整个 Gruntfile，接下来将详细解释其中的每一部分。

**["wrapper" 函数](http://www.gruntjs.net/getting-started" \l "wrapper)**

每一份 Gruntfile （和grunt插件）都遵循同样的格式，你所书写的Grunt代码必须放在此函数内：

module.exports = function(grunt) {

// Do grunt-related things in here

};

**项目和任务配置**

大部分的Grunt任务都依赖某些配置数据，这些数据被定义在一个object内，并传递给[grunt.initConfig](http://www.gruntjs.net/grunt#grunt.initconfig) 方法。

在下面的案例中，grunt.file.readJSON('package.json') 将存储在package.json文件中的JSON元数据引入到grunt config中。 由于<% %>模板字符串可以引用任意的配置属性，因此可以通过这种方式来指定诸如文件路径和文件列表类型的配置数据，从而减少一些重复的工作。

你可以在这个配置对象中(传递给initConfig()方法的对象)存储任意的数据，只要它不与你任务配置所需的属性冲突，否则会被忽略。此外，由于这本身就是JavaScript，你不仅限于使用JSON；你可以在这里使用任意的有效的JS代码。如果有必要，你甚至可以以编程的方式生成配置。

与大多数task一样，[grunt-contrib-uglify](http://github.com/gruntjs/grunt-contrib-uglify) 插件中的uglify 任务要求它的配置被指定在一个同名属性中。在这里有一个例子, 我们指定了一个banner选项(用于在文件顶部生成一个注释)，紧接着是一个单一的名为build的uglify目标，用于将一个js文件压缩为一个目标文件。

// Project configuration.

grunt.initConfig({

pkg: grunt.file.readJSON('package.json'),

uglify: {

options: {

banner: '/\*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> \*/\n'

},

build: {

src: 'src/<%= pkg.name %>.js',

dest: 'build/<%= pkg.name %>.min.js'

}

}

});

[**加载 Grunt 插件和任务**](http://www.gruntjs.net/getting-started#grunt)

像 [concatenation](https://github.com/gruntjs/grunt-contrib-concat)、[minification]、[grunt-contrib-uglify](http://github.com/gruntjs/grunt-contrib-uglify) 和 [linting](https://github.com/gruntjs/grunt-contrib-jshint)这些常用的任务（task）都已经以[grunt插件](https://github.com/gruntjs)的形式被开发出来了。只要在 package.json 文件中被列为dependency（依赖）的包，并通过npm install安装之后，都可以在Gruntfile中以简单命令的形式使用：

// 加载能够提供"uglify"任务的插件。

grunt.loadNpmTasks('grunt-contrib-uglify');

**注意：** grunt --help 命令将列出所有可用的任务。

**自定义任务**

通过定义 default 任务，可以让Grunt默认执行一个或多个任务。在下面的这个案例中，执行 grunt 命令时如果不指定一个任务的话，将会执行uglify任务。这和执行grunt uglify 或者 grunt default的效果一样。default任务列表数组中可以指定任意数目的任务（可以带参数）。

// Default task(s).

grunt.registerTask('default', ['uglify']);

如果Grunt插件中的任务（task）不能满足你的项目需求，你还可以在Gruntfile中自定义任务（task）。例如，在下面的 Gruntfile 中自定义了一个default 任务，并且他甚至不依赖任务配置：

module.exports = function(grunt) {

// A very basic default task.

grunt.registerTask('default', 'Log some stuff.', function() {

grunt.log.write('Logging some stuff...').ok();

});

};

特定于项目的任务不必在 Gruntfile 中定义。他们可以定义在外部.js 文件中，并通过[grunt.loadTasks](http://www.gruntjs.net/grunt/#grunt.loadtasks)方法加载。

**扩展阅读**

* The [Installing grunt](http://www.gruntjs.net/installing-grunt/) guide has detailed information about installing specific, production or in-development, versions of Grunt and grunt-cli.
* The [Configuring Tasks](http://www.gruntjs.net/configuring-tasks/) guide has an in-depth explanation on how to configure tasks, targets, options and files inside the Gruntfile, along with an explanation of templates, globbing patterns and importing external data.
* The [Creating Tasks](http://www.gruntjs.net/creating-tasks/) guide lists the differences between the types of Grunt tasks and shows a number of sample tasks and configurations.
* For more information about writing custom tasks or Grunt plugins, check out the [developer documentation](http://www.gruntjs.net/grunt).

# 2.配置任务

这个指南解释了如何使用 Gruntfile 来为你的项目配置task。如果你还不知道 Gruntfile 是什么，请先阅读 [快速入门](http://www.gruntjs.net/getting-started) 指南并看看这个[Gruntfile 实例](http://www.gruntjs.net/sample-gruntfile)。

## [Grunt配置](http://www.gruntjs.net/configuring-tasks#grunt)

Grunt的task配置都是在 Gruntfile 中的grunt.initConfig方法中指定的。此配置主要是以任务名称命名的属性，也可以包含其他任意数据。一旦这些代表任意数据的属性与任务所需要的属性相冲突，就将被忽略。

此外，由于这本身就是JavaScript，因此你不仅限于使用JSON；你可以在这里使用任何有效的JavaScript。必要的情况下，你甚至可以以编程的方式生成配置。

grunt.initConfig({

concat: {

// 这里是concat任务的配置信息。

},

uglify: {

// 这里是uglify任务的配置信息

},

// 任意数据。

my\_property: 'whatever',

my\_src\_files: ['foo/\*.js', 'bar/\*.js'],

});

## 任务配置和目标

当运行一个任务时，Grunt会自动查找配置对象中的同名属性。多任务（multi-task）可以通过任意命名的“目标（target）”来定义多个配置。在下面的案例中，concat任务有名为foo和bar两个目标，而uglify任务仅仅只有一个名为bar目标。

grunt.initConfig({

concat: {

foo: {

// concat task "foo" target options and files go here.

},

bar: {

// concat task "bar" target options and files go here.

},

},

uglify: {

bar: {

// uglify task "bar" target options and files go here.

},

},

});

同时指定任务（task）和目标（target），例如grunt concat:foo或者grunt concat:bar，将只会处理指定目标（target）的配置，而运行grunt concat将遍历所有目标（target）并依次处理。注意，如果一个任务使用[grunt.task.renameTask](http://www.gruntjs.net/grunt.task#grunt.task.renametask)重命名过，Grunt将在配置对象中查找以新的任务名命名的属性。

## [options属性](http://www.gruntjs.net/configuring-tasks" \l "options)

在一个任务配置中，options属性可以用来指定覆盖内置属性的默认值。此外，每一个目标（target）中还可以拥有一个专门针对此目标（target）的options属性。目标（target）级的平options将会覆盖任务级的options。

options对象是可选的，如果不需要，可以忽略。

grunt.initConfig({

concat: {

options: {

// 这里是任务级的Options，覆盖默认值

},

foo: {

options: {

// "foo" target options may go here, overriding task-level options.

},

},

bar: {

// No options specified; this target will use task-level options.

},

},

});

## 文件

由于大多的任务都是执行文件操作，Grunt有一个强大的抽象层用于声明任务应该操作哪些文件。这里有好几种定义**src-dest**(源文件-目标文件)文件映射的方式，均提供了不同程度的描述和控制操作方式。任何一种多任务（multi-task）都能理解下面的格式，所以你只需要选择满足你需求的格式就行。

所有的文件格式都支持src和dest属性，此外"Compact"[简洁]和"Files Array"[文件数组]格式还支持以下一些额外的属性：

* filter 它通过接受任意一个有效的[fs.Stats方法名](http://nodejs.org/docs/latest/api/fs.html#fs_class_fs_stats)或者一个函数来匹配src文件路径并根据匹配结果返回true或者false。
* nonull 如果被设置为 true，未匹配的模式也将执行。结合Grunt的--verbore标志, 这个选项可以帮助用来调试文件路径的问题。
* dot 它允许模式模式匹配句点开头的文件名，即使模式并不明确文件名开头部分是否有句点。
* matchBase如果设置这个属性，缺少斜线的模式(意味着模式中不能使用斜线进行文件路径的匹配)将不会匹配包含在斜线中的文件名。 例如，a?b将匹配/xyz/123/acb但不匹配/xyz/acb/123。
* expand 处理动态的src-dest文件映射，更多的信息请查看[动态构建文件对象](http://www.gruntjs.net/configuring-tasks#building-the-files-object-dynamically)。
* 其他的属性将作为匹配项传递给底层的库。 请查看[node-glob](https://github.com/isaacs/node-glob) 和[minimatch](https://github.com/isaacs/minimatch) 文档以获取更多信息。

### 简洁格式

这种形式允许每个目标对应一个**src-dest**文件映射。通常情况下它用于只读任务，比如[grunt-contrib-jshint](https://github.com/gruntjs/grunt-contrib-jshint)，它就只需要一个单一的src属性，而不需要关联的dest选项. 这种格式还支给每个src-dest文件映射指定额外的属性。

grunt.initConfig({

jshint: {

foo: {

src: ['src/aa.js', 'src/aaa.js']

},

},

concat: {

bar: {

src: ['src/bb.js', 'src/bbb.js'],

dest: 'dest/b.js',

},

},

});

### 文件对象格式

这种形式支持每个目标对应多个src-dest形式的文件映射，属性名就是目标文件，源文件就是它的值(源文件列表则使用数组格式声明)。可以使用这种方式指定数个src-dest文件映射， 但是不能够给每个映射指定附加的属性。

grunt.initConfig({

concat: {

foo: {

files: {

'dest/a.js': ['src/aa.js', 'src/aaa.js'],

'dest/a1.js': ['src/aa1.js', 'src/aaa1.js'],

},

},

bar: {

files: {

'dest/b.js': ['src/bb.js', 'src/bbb.js'],

'dest/b1.js': ['src/bb1.js', 'src/bbb1.js'],

},

},

},

});

### 文件数组格式

这种形式支持每个目标对应多个src-dest文件映射，同时也允许每个映射拥有额外属性：

grunt.initConfig({

concat: {

foo: {

files: [

{src: ['src/aa.js', 'src/aaa.js'], dest: 'dest/a.js'},

{src: ['src/aa1.js', 'src/aaa1.js'], dest: 'dest/a1.js'},

],

},

bar: {

files: [

{src: ['src/bb.js', 'src/bbb.js'], dest: 'dest/b/', nonull: true},

{src: ['src/bb1.js', 'src/bbb1.js'], dest: 'dest/b1/', filter: 'isFile'},

],

},

},

});

### 较老的格式

**dest-as-target**文件格式在多任务和目标出现之前是一个过渡形式，目标文件路径实际上就是目标名称。遗憾的是, 由于目标名称是文件路径，那么运行grunt task:target可能不合适。此外，你也不能指定一个目标级的options或者给每个src-dest文件映射指定额外属性。

此种格式已经不赞成使用，请尽量不要使用。

grunt.initConfig({

concat: {

'dest/a.js': ['src/aa.js', 'src/aaa.js'],

'dest/b.js': ['src/bb.js', 'src/bbb.js'],

},

});

### 自定义过滤函数

filter属性可以给你的目标文件提供一个更高级的详细帮助信息。只需要使用一个有效的[fs.Stats 方法名](http://nodejs.org/docs/latest/api/fs.html#fs_class_fs_stats)。下面的配置仅仅清理一个与模式匹配的真实的文件：

grunt.initConfig({

clean: {

foo: {

src: ['tmp/\*\*/\*'],

filter: 'isFile',

},

},

});

或者创建你自己的filter函数，根据文件是否匹配来返回true或者false。下面的例子将仅仅清理一个空目录：

grunt.initConfig({

clean: {

foo: {

src: ['tmp/\*\*/\*'],

filter: function(filepath) {

return (grunt.file.isDir(filepath) && require('fs').readdirSync(filepath).length === 0);

},

},

},

});

### 通配符模式

通常分别指定所有源文件路径是不切实际的，因此Grunt通过内置支持[node-glob](https://github.com/isaacs/node-glob) 和 [minimatch](https://github.com/isaacs/minimatch) 库来匹配文件名(又叫作globbing)。

然这并不是一个综合的匹配模式方面的教程，你只需要知道如何在文件路径匹配过程中使用它们即可：

* \* 匹配任意数量的字符，但不匹配 /
* ? 匹配单个字符，但不匹配 /
* \*\* 匹配任意数量的字符，包括 /，只要它是路径中唯一的一部分
* {} 允许使用一个逗号分割的“或”表达式列表
* ! 在模式的开头用于排除一个匹配模式所匹配的任何文件

每个人都需要知道的是：foo/\*.js将匹配位于foo/目录下的所有的.js结尾的文件；而foo/\*\*/\*js将匹配foo/目录以及其子目录中所有以.js结尾的文件。

此外, 为了简化原本复杂的通配符模式，Grunt允许指定一个数组形式的文件路径或者一个通配符模式。所有模式按顺序处理，模式处理的过程中，带有!前缀的模式所匹配的文件将不包含在结果集中。 而且其结果集中的每一项也是唯一的。

例如：

// 指定单个文件：

{src: 'foo/this.js', dest: ...}

// 指定一个文件数组：

{src: ['foo/this.js', 'foo/that.js', 'foo/the-other.js'], dest: ...}

// 使用一个匹配模式：

{src: 'foo/th\*.js', dest: ...}

// 一个独立的node-glob模式：

{src: 'foo/{a,b}\*.js', dest: ...}

// 也可以这样编写：

{src: ['foo/a\*.js', 'foo/b\*.js'], dest: ...}

// foo目录中所有的.js文件，按字母顺序排序：

{src: ['foo/\*.js'], dest: ...}

// 首先是bar.js，接着是剩下的.js文件，并按字母顺序排序：

{src: ['foo/bar.js', 'foo/\*.js'], dest: ...}

// 除bar.js之外的所有的.js文件，按字母顺序排序：

{src: ['foo/\*.js', '!foo/bar.js'], dest: ...}

// 按字母顺序排序的所有.js文件，但是bar.js在最后。

{src: ['foo/\*.js', '!foo/bar.js', 'foo/bar.js'], dest: ...}

// 模板也可以用于文件路径或者匹配模式中：

{src: ['src/<%= basename %>.js'], dest: 'build/<%= basename %>.min.js'}

// 它们也可以引用在配置中定义的其他文件列表：

{src: ['foo/\*.js', '<%= jshint.all.src %>'], dest: ...}

更多关于通配符模式的语法，请查看[node-glob](https://github.com/isaacs/node-glob) 和 [minimatch](https://github.com/isaacs/minimatch) 的文档。

### 动态构建文件对象

当你希望处理大量的单个文件时，这里有一些附加的属性可以用来动态的构建一个文件列表。这些属性都可以用于Compact和Files Array文件映射格式。

expand 设置为true用于启用下面的选项：

* cwd 所有src指定的匹配都将相对于此处指定的路径（但不包括此路径） 。
* src 相对于cwd路径的匹配模式。
* dest 目标文件路径前缀。
* ext 对于生成的dest路径中所有实际存在文件，均使用这个属性值替换扩展名。
* extDot 用于指定标记扩展名的英文点号的所在位置。可以赋值 'first' （扩展名从文件名中的第一个英文点号开始） 或 'last' （扩展名从最后一个英文点号开始），默认值为 'first' [添加于 0.4.3 版本]
* flatten 从生成的dest路径中移除所有的路径部分。
* rename 对每个匹配的src文件调用这个函数(在重命名后缀和移除路径之后)。dest和匹配的src路径将被作为参数传入，此函数应该返回一个新的dest值。 如果相同的dest返回不止一次，那么，每个返回此值的src来源都将被添加到一个数组中作为源列表。

在下面的例子中，uglify 任务中的static\_mappings和dynamic\_mappings两个目标具有相同的src-dest文件映射列表, 这是因为任务运行时Grunt会自动展开dynamic\_mappings文件对象为4个单独的静态src-dest文件映射--假设这4个文件能够找到。

可以指定任意静态src-dest和动态的src-dest文件映射相互结合。

grunt.initConfig({

uglify: {

static\_mappings: {

// Because these src-dest file mappings are manually specified, every

// time a new file is added or removed, the Gruntfile has to be updated.

files: [

{src: 'lib/a.js', dest: 'build/a.min.js'},

{src: 'lib/b.js', dest: 'build/b.min.js'},

{src: 'lib/subdir/c.js', dest: 'build/subdir/c.min.js'},

{src: 'lib/subdir/d.js', dest: 'build/subdir/d.min.js'},

],

},

dynamic\_mappings: {

// Grunt will search for "\*\*/\*.js" under "lib/" when the "uglify" task

// runs and build the appropriate src-dest file mappings then, so you

// don't need to update the Gruntfile when files are added or removed.

files: [

{

expand: true, // Enable dynamic expansion.

cwd: 'lib/', // Src matches are relative to this path.

src: ['\*\*/\*.js'], // Actual pattern(s) to match.

dest: 'build/', // Destination path prefix.

ext: '.min.js', // Dest filepaths will have this extension.

extDot: 'first' // Extensions in filenames begin after the first dot

},

],

},

},

});

## 模板

使用<% %>分隔符指定的模板会在任务从它们的配置中读取相应的数据时将自动扩展扫描。模板会被递归的展开，直到配置中不再存在遗留的模板相关的信息(与模板匹配的)。

整个配置对象决定了属性上下文(模板中的属性)。此外，在模板中使用grunt以及它的方法都是有效的，例如： <%= grunt.template.today('yyyy-mm-dd') %>。

* <%= prop.subprop %> 将会自动展开配置信息中的prop.subprop的值，不管是什么类型。像这样的模板不仅可以用来引用字符串值，还可以引用数组或者其他对象类型的值。
* <% %> 执行任意内联的JavaScript代码。对于控制流或者循环来说是非常有用的。

下面以concat任务配置为例，运行grunt concat:sample时将通过banner中的/\* abcde \*/连同foo/\*.js+bar/\*.js+bar/\*.js匹配的所有文件来生成一个名为build/abcde.js的文件。

grunt.initConfig({

concat: {

sample: {

options: {

banner: '/\* <%= baz %> \*/\n', // '/\* abcde \*/\n'

},

src: ['<%= qux %>', 'baz/\*.js'], // [['foo/\*.js', 'bar/\*.js'], 'baz/\*.js']

dest: 'build/<%= baz %>.js', // 'build/abcde.js'

},

},

//用于任务配置模板的任意属性

foo: 'c',

bar: 'b<%= foo %>d', // 'bcd'

baz: 'a<%= bar %>e', // 'abcde'

qux: ['foo/\*.js', 'bar/\*.js'],

});

## 导入外部数据

在下面的Gruntfile中，项目的元数据是从package.json文件中导入到Grunt配置中的，并且[grunt-contrib-uglify 插件](http://github.com/gruntjs/grunt-contrib-uglify)中的 uglify 任务被配置用于压缩一个源文件以及使用该元数据动态的生成一个banner注释。

Grunt有grunt.file.readJSON和grunt.file.readYAML两个方法分别用于引入JSON和YAML数据。

grunt.initConfig({

pkg: grunt.file.readJSON('package.json'),

uglify: {

options: {

banner: '/\*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> \*/\n'

},

dist: {

src: 'src/<%= pkg.name %>.js',

dest: 'dist/<%= pkg.name %>.min.js'

}

}

});

Found an error in the documentation? [File an issue](https://github.com/gruntjs/grunt-docs/issues).

# 3.[Gruntfile 实例](http://www.gruntjs.net/sample-gruntfile#gruntfile)

下面就针对一个 Gruntfile 案例做简单分析，也可以作为一个实例使用：

module.exports = function(grunt) {

grunt.initConfig({

jshint: {

files: ['Gruntfile.js', 'src/\*\*/\*.js', 'test/\*\*/\*.js'],

options: {

globals: {

jQuery: true

}

}

},

watch: {

files: ['<%= jshint.files %>'],

tasks: ['jshint']

}

});

grunt.loadNpmTasks('grunt-contrib-jshint');

grunt.loadNpmTasks('grunt-contrib-watch');

grunt.registerTask('default', ['jshint']);

};

在页面底部是这个 Gruntfile 实例的完整内容，如果你按顺序阅读本文的话，可以跟随我们一步步分析这个文件中的每一部分。我们会用到以下5个 Grunt 插件：

* [grunt-contrib-uglify](https://github.com/gruntjs/grunt-contrib-uglify)
* [grunt-contrib-qunit](https://github.com/gruntjs/grunt-contrib-qunit)
* [grunt-contrib-concat](https://github.com/gruntjs/grunt-contrib-concat)
* [grunt-contrib-jshint](https://github.com/gruntjs/grunt-contrib-jshint)
* [grunt-contrib-watch](https://github.com/gruntjs/grunt-contrib-watch)

第一部分是"wrapper" 函数，它包含了整个Grunt配置信息。

module.exports = function(grunt) {

}

在这个函数中，我们可以初始化 configuration 对象：

grunt.initConfig({

});

接下来可以从package.json 文件读入项目配置信息，并存入pkg 属性内。这样就可以让我们访问到package.json文件中列出的属性了，如下：

pkg: grunt.file.readJSON('package.json')

到目前为止我们就可以看到如下配置：

module.exports = function(grunt) {

grunt.initConfig({

pkg: grunt.file.readJSON('package.json')

});

};

现在我们就可以为我们的每个任务来定义相应的配置(逐个定义我们为项目定义的任务配置)，然后每个任务的配置对象作为Grunt配置对象(即grunt.initConfig({})所接受的配置对象)的属性，并且这个属性名称与任务名相同。因此"concat"任务就是我们的配置对象中的"concat"键(属性)。下面便是我的"concat"任务的配置对象。

concat: {

options: {

// 定义一个用于插入合并输出文件之间的字符

separator: ';'

},

dist: {

// 将要被合并的文件

src: ['src/\*\*/\*.js'],

// 合并后的JS文件的存放位置

dest: 'dist/<%= pkg.name %>.js'

}

}

注意我是如何引用JSON文件(也就是我们在配置对象顶部引入的配置文件)中的name属性的。这里我使用pkg.name来访问我们刚才引入并存储在pkg属性中的package.json文件信息，它会被解析为一个JavaScript对象。Grunt自带的有一个简单的模板引擎用于输出配置对象(这里是指package.json中的配置对象)属性值，这里我让concat任务将所有存在于src/目录下以.js结尾的文件合并起来，然后存储在dist目录中，并以项目名来命名。

现在我们来配置uglify插件，它的作用是压缩（minify）JavaScript文件：

uglify: {

options: {

// 此处定义的banner注释将插入到输出文件的顶部

banner: '/\*! <%= pkg.name %> <%= grunt.template.today("dd-mm-yyyy") %> \*/\n'

},

dist: {

files: {

'dist/<%= pkg.name %>.min.js': ['<%= concat.dist.dest %>']

}

}

}

这里我们让uglify在dist/目录中创建了一个包含压缩结果的JavaScript文件。注意这里我使用了<%= concat.dist.dest>，因此uglify会自动压缩concat任务中生成的文件。

QUnit插件的设置非常简单。 你只需要给它提供用于测试运行的文件的位置，注意这里的QUnit是运行在HTML文件上的。

qunit: {

files: ['test/\*\*/\*.html']

},

JSHint插件的配置也很简单：

jshint: {

// define the files to lint

files: ['gruntfile.js', 'src/\*\*/\*.js', 'test/\*\*/\*.js'],

// configure JSHint (documented at http://www.jshint.com/docs/)

options: {

// more options here if you want to override JSHint defaults

globals: {

jQuery: true,

console: true,

module: true

}

}

}

JSHint只需要一个文件数组(也就是你需要检测的文件数组)， 然后是一个options对象(这个对象用于重写JSHint提供的默认检测规则)。你可以到[JSHint官方文档](http://www.jshint.com/docs/)站点中查看完整的文档。如果你乐于使用JSHint提供的默认配置，那么在Gruntfile中就不需要重新定义它们了.

最后，我们来看看watch插件：

watch: {

files: ['<%= jshint.files %>'],

tasks: ['jshint', 'qunit']

}

你可以在命令行使用grunt watch来运行这个任务。当它检测到任何你所指定的文件(在这里我使用了JSHint任务中需要检测的相同的文件)发生变化时，它就会按照你所指定的顺序执行指定的任务(在这里我指定了jshint和qunit任务)。

最后, 我们还要加载所需要的Grunt插件。 它们应该已经全部通过npm安装好了。

grunt.loadNpmTasks('grunt-contrib-uglify');

grunt.loadNpmTasks('grunt-contrib-jshint');

grunt.loadNpmTasks('grunt-contrib-qunit');

grunt.loadNpmTasks('grunt-contrib-watch');

grunt.loadNpmTasks('grunt-contrib-concat');

最后设置了一些task。最重要的是default任务：

// 在命令行上输入"grunt test"，test task就会被执行。

grunt.registerTask('test', ['jshint', 'qunit']);

// 只需在命令行上输入"grunt"，就会执行default task

grunt.registerTask('default', ['jshint', 'qunit', 'concat', 'uglify']);

下面便是最终完成的 Gruntfile 文件：

module.exports = function(grunt) {

grunt.initConfig({

pkg: grunt.file.readJSON('package.json'),

concat: {

options: {

separator: ';'

},

dist: {

src: ['src/\*\*/\*.js'],

dest: 'dist/<%= pkg.name %>.js'

}

},

uglify: {

options: {

banner: '/\*! <%= pkg.name %> <%= grunt.template.today("dd-mm-yyyy") %> \*/\n'

},

dist: {

files: {

'dist/<%= pkg.name %>.min.js': ['<%= concat.dist.dest %>']

}

}

},

qunit: {

files: ['test/\*\*/\*.html']

},

jshint: {

files: ['Gruntfile.js', 'src/\*\*/\*.js', 'test/\*\*/\*.js'],

options: {

//这里是覆盖JSHint默认配置的选项

globals: {

jQuery: true,

console: true,

module: true,

document: true

}

}

},

watch: {

files: ['<%= jshint.files %>'],

tasks: ['jshint', 'qunit']

}

});

grunt.loadNpmTasks('grunt-contrib-uglify');

grunt.loadNpmTasks('grunt-contrib-jshint');

grunt.loadNpmTasks('grunt-contrib-qunit');

grunt.loadNpmTasks('grunt-contrib-watch');

grunt.loadNpmTasks('grunt-contrib-concat');

grunt.registerTask('test', ['jshint', 'qunit']);

grunt.registerTask('default', ['jshint', 'qunit', 'concat', 'uglify']);

};

Found an error in the documentation? [File an issue](https://github.com/gruntjs/grunt-docs/issues).

# 4.创建任务

任务是Grunt的面包和奶油。就像你常用的工具，如： jshint 或 nodeunit。每当运行Grunt时, 你可以为其指定一个或多个任务, 这些任务用于告诉Grunt你想要它做什么事情。

如果你没有指定一个任务，并且你已经定义一个名为 "default" 的任务，那么该任务将会默认被执行（不用诧异，总要做点儿什么啊！）。

## 任务别名

如果指定了一个任务列表，**新任务将是这一个或多个指定任务的别名**。当运行此 "任务别名" 时，在taskList 中指定的每个任务都会按照其出现的顺序依次执行。taskList参数必须时一个任务数组。

grunt.registerTask(taskName, [description, ] taskList)

下面的任务别名案例中定义了一个 'default' 任务，如果运行Grunt时没有指定任何任务，它将自动执行'jshint'、'qunit'、'concat' 和 'uglify' 任务。

grunt.registerTask('default', ['jshint', 'qunit', 'concat', 'uglify']);

还可以给任务指定参数。在下面的案例中，别名 "dist" 将执行 "concat" 和 "uglify" 两个任务，并且它们都带有一个 "dist" 参数：

grunt.registerTask('dist', ['concat:dist', 'uglify:dist']);

## 多任务

当运行一个多任务时，Grunt会自动从项目的配置对象中查找同名属性。多任务可以有多个配置，并且可以使用任意命名的'targets'。

同时指定像grunt concat:foo或者grunt concat:bar这样的任务和目标，在运行时Grunt只会处理指定目标的配置；然而如果运行grunt concat，将会遍历所有的目标, 并按任务指定的顺序处理每个目标。注意，如果一个任务已经使用[grunt.task.renameTask](http://www.gruntjs.net/grunt.task#grunt.task.renametask)重命名过，Grunt将会自动在配置对象中查找新任务名称属性。

大部分的contrib任务(主要是指官方提供的任务)，包括[grunt-contrib-jshint插件的jshint任务](https://github.com/gruntjs/grunt-contrib-jshint#jshint-task)，以及[grunt-contrib-concat插件的concat任务](https://github.com/gruntjs/grunt-contrib-concat#concat-task)都是多任务形式的。

grunt.registerMultiTask(taskName, [description, ] taskFunction)

对于指定的配置，这里有一个案例演示了如果通过grunt log:foo运行Grunt，它会输出foo: 1,2,3；如果通过grunt log:bar来运行Grunt， 它会输出bar: hello world。然而如果通过grunt log运行Grunt, 它会输出foo: 1,2,3，然后是bar: hello world，最后是baz: false(任务目标会按照指定的顺序进行处理)。

grunt.initConfig({

log: {

foo: [1, 2, 3],

bar: 'hello world',

baz: false

}

});

grunt.registerMultiTask('log', 'Log stuff.', function() {

grunt.log.writeln(this.target + ': ' + this.data);

});

## "基本" 任务

当一个基本任务执行时，Grunt并不会检查配置和环境 -- 它仅仅执行指定的任务函数，并传递任何使用冒号分割的参数作为函数的参数。

grunt.registerTask(taskName, [description, ] taskFunction)

下面的案例中，如果执行 grunt foo:testing:123，将输出日志 foo, testing 123。 如果执行这个任务时不传递参数，只是执行 grunt foo，那么将输出日志 foo, no args。

grunt.registerTask('foo', 'A sample task that logs stuff.', function(arg1, arg2) {

if (arguments.length === 0) {

grunt.log.writeln(this.name + ", no args");

} else {

grunt.log.writeln(this.name + ", " + arg1 + " " + arg2);

}

});

## 自定义任务

你可以和任务一起疯狂。如果你的任务并没有遵循 "多任务" 结构，那就使用自定义任务。

grunt.registerTask('default', 'My "default" task description.', function() {

grunt.log.writeln('Currently running the "default" task.');

});

在一个任务内部，你可以执行其他的任务。

grunt.registerTask('foo', 'My "foo" task.', function() {

// Enqueue "bar" and "baz" tasks, to run after "foo" finishes, in-order.

grunt.task.run('bar', 'baz');

// Or:

grunt.task.run(['bar', 'baz']);

});

任务也可以是**异步的**。

grunt.registerTask('asyncfoo', 'My "asyncfoo" task.', function() {

// Force task into async mode and grab a handle to the "done" function.

var done = this.async();

// Run some sync stuff.

grunt.log.writeln('Processing task...');

// And some async stuff.

setTimeout(function() {

grunt.log.writeln('All done!');

done();

}, 1000);

});

任务也可以访问它们自身名称和参数。

grunt.registerTask('foo', 'My "foo" task.', function(a, b) {

grunt.log.writeln(this.name, a, b);

});

// 用法：

// grunt foo foo:bar

// logs: "foo", undefined, undefined

// logs: "foo", "bar", undefined

// grunt foo:bar:baz

// logs: "foo", "bar", "baz"

如果记录到任何错误，那么任务就会失败。

grunt.registerTask('foo', 'My "foo" task.', function() {

if (failureOfSomeKind) {

grunt.log.error('This is an error message.');

}

// 如果这个任务出现错误则返回false

if (ifErrors) { return false; }

grunt.log.writeln('This is the success message');

});

当任务失败时，所有后续任务都将终止，除非指定 --force 。

grunt.registerTask('foo', 'My "foo" task.', function() {

// Fail synchronously.

return false;

});

grunt.registerTask('bar', 'My "bar" task.', function() {

var done = this.async();

setTimeout(function() {

// Fail asynchronously.

done(false);

}, 1000);

});

任务还可以依赖于其他任务的成功执行。注意 grunt.task.requires 并不会真正的运行其他任务，它仅仅检查其它任务是否已经执行，并且没有失败。

grunt.registerTask('foo', 'My "foo" task.', function() {

return false;

});

grunt.registerTask('bar', 'My "bar" task.', function() {

// 如果"foo"任务运行失败或者没有运行则任务失败。

grunt.task.requires('foo');

// 如果"foo"任务运行成功则执行这里的代码。

grunt.log.writeln('Hello, world.');

});

// 用法：

// grunt foo bar

// 没有输出，因为"foo"失败。

// grunt bar

// 没有输出，因为"foo"从未运行。

如果任务需要的配置属性不存在，其也可能失败。

grunt.registerTask('foo', 'My "foo" task.', function() {

// Fail task if "meta.name" config prop is missing

// Format 1: String

grunt.config.requires('meta.name');

// or Format 2: Array

grunt.config.requires(['meta', 'name']);

// Log... conditionally.

grunt.log.writeln('This will only log if meta.name is defined in the config.');

});

任务还可以访问配置属性。

grunt.registerTask('foo', 'My "foo" task.', function() {

// 记录属性值，如果属性未定义（undefined）则返回null。

grunt.log.writeln('The meta.name property is: ' + grunt.config('meta.name'));

// 同样的记录属性值，如果属性未定义（undefined）则返回null。

grunt.log.writeln('The meta.name property is: ' + grunt.config(['meta', 'name']));

});

在 [contrib tasks](https://github.com/gruntjs/) 中可以查看更多案例。

## [CLI 参数 / 环境](http://www.gruntjs.net/creating-tasks#cli)

通过 process.env 来访问[环境变量](http://en.wikipedia.org/wiki/Environment_variable)。

请参考 [使用命令行工具](http://using-the-cli/)章节，查看完整的的命令行选项列表。

## [为什么我的异步task没有完成？](http://www.gruntjs.net/creating-tasks" \l "task)

如果发生这种情况，可能是由于你忘记调用 [this.async](http://www.gruntjs.net/inside-tasks#this.async) 方法来告诉Grunt你的任务是异步的。为了简单起见，Grunt使用同步的编码风格，可以在task体中通过调用 this.async() 将其转换为异步的。

注意，传递 false 给 done() 函数就会告诉Grunt你的任务已经失败。

例如：

grunt.registerTask('asyncme', 'My asynchronous task.', function() {

var done = this.async();

doSomethingAsync(done);

});

## 额外参考资料

如果你需要更多参考资料来创建自己的 task ，请参考 [API](http://www.gruntjs.net/api) 文档 if you need extra reference to create your tasks.

**2.** [**grunt.config**](http://www.gruntjs.net/api/grunt.config#grunt.config)

从 Gruntfile 中获取针对当前项目的配置数据。

注意，任何标记为 ☃ (unicode snowman) 的方法也是可以直接通过 grunt 对象访问的；任何标记为 ☆ (white star) 的方法都可以在task内部通过 this 对象访问的。请知晓。

**初始化配置数据**

*注意，下面列出的方法也可以通过 grunt 对象访问，访问形式为 grunt.initConfig。*

**[grunt.config.init ☃](http://www.gruntjs.net/api/grunt.config" \l "grunt.config.init)**

为当前项目初始化一个配置对象。其中传入的 configObject 参数可以用在后续的task中，可以通过grunt.config 方法访问。几乎每个项目的 Gruntfile 都会调用此方法。

grunt.config.init(configObject)

注意，任何 <% %> 模板字符串只会在取到配置数据后才被处理。

下面的案例展示了针对 [grunt-contrib-jshint插件](https://github.com/gruntjs/grunt-contrib-jshint) 中的 jshint task的配置数据：

grunt.config.init({

jshint: {

all: ['lib/\*.js', 'test/\*.js', 'Gruntfile.js']

}

});

查看 [Getting started](http://www.gruntjs.net/getting-started) 指南可以获取更多的配置案例。

*此方法还可以以 grunt.initConfig 的形式访问。*

**获取配置数据**

The following methods allow Grunt configuration data to be accessed either via dot-delimited string like'pkg.author.name' or via array of property name parts like ['pkg', 'author', 'name'].

Note that if a specified property name contains a . dot, it must be escaped with a literal backslash, eg.'concat.dist/built\\.js'. If an array of parts is specified, Grunt will handle the escaping internally with the grunt.config.escape method.

**[grunt.config](http://www.gruntjs.net/api/grunt.config" \l "grunt.config)**

从项目的 Grunt 配置中获取或者设置一个值。这个方法作为其他方法的别名；如果传递两个参数，grunt.config.set被调用，另一方面grunt.config.get也被调用。Get or set a value from the project's grunt configuration. This method serves as an alias to other methods; if two arguments are passed, grunt.config.set is called, otherwise grunt.config.get is called.

grunt.config([prop [, value]])

**[grunt.config.get](http://www.gruntjs.net/api/grunt.config" \l "grunt.config.get)**

Get a value from the project's Grunt configuration. If prop is specified, that property's value is returned, ornull if that property is not defined. If prop isn't specified, a copy of the entire config object is returned. Templates strings will be recursively processed using the grunt.config.process method.

grunt.config.get([prop])

**[grunt.config.process](http://www.gruntjs.net/api/grunt.config" \l "grunt.config.process)**

Process a value, recursively expanding <% %> templates (via the grunt.template.process method) in the context of the Grunt config, as they are encountered. this method is called automatically bygrunt.config.get but *not* by grunt.config.getRaw.

grunt.config.process(value)

If any retrieved value is entirely a single '<%= foo %>' or '<%= foo.bar %>' template string, and the specified foo or foo.bar property is a non-string (and not null or undefined) value, it will be expanded to the *actual* value. That, combined with grunt's task system automatically flattening arrays, can be extremely useful.

**[grunt.config.getRaw](http://www.gruntjs.net/api/grunt.config" \l "grunt.config.getraw)**

Get a raw value from the project's Grunt configuration, without processing <% %> template strings. If propis specified, that property's value is returned, or null if that property is not defined. If prop isn't specified, a copy of the entire config object is returned.

grunt.config.getRaw([prop])

**[grunt.config.set](http://www.gruntjs.net/api/grunt.config" \l "grunt.config.set)**

给当前项目的 Grunt 配置中的某个属性设置一个值。

grunt.config.set(prop, value)

注意，任何 <% %> 模板字符串只会在取到配置数据后才被处理。

**[grunt.config.escape](http://www.gruntjs.net/api/grunt.config" \l "grunt.config.escape)**

忽略给定的propString中的.点号。这应该用于包含点号的属性名。Escape . dots in the givenpropString. This should be used for property names that contain dots.

grunt.config.escape(propString)

**[grunt.config.merge](http://www.gruntjs.net/api/grunt.config" \l "grunt.config.merge)**

*Added in 0.4.5*

Recursively merges properties of the specified configObject into the current project configuration.

grunt.config.merge(configObject)

You can use this method to append configuration options, targets, etc., to already defined tasks, for example:

grunt.config.merge({

watch: {

files: ["path/to/files"],

tasks: ["task"]

}

});

**[Requiring Config Data](http://www.gruntjs.net/api/grunt.config" \l "requiring-config-data)**

*注意，下面列出的方法都可以在task内部通过 this 对象访问，访问形式为 this.requiresConfig。*

**[grunt.config.requires ☆](http://www.gruntjs.net/api/grunt.config" \l "grunt.config.requires)**

如果需要的配置属性有一个或多个不存在、值为null 或 undefined，当前task将失败。此方法可以指定一个或多个字符串、配置属性数组作为参数。

grunt.config.requires(prop [, prop [, ...]])

*此方法在task内部以 this.requiresConfig 形式调用。*

# 3.[grunt.event](http://www.gruntjs.net/api/grunt.event#grunt.event)

在这个页面中仅仅列出了最重要的方法，完整的 [EventEmitter2 API](https://github.com/hij1nx/EventEmitter2) 在 grunt.event 对象中都有定义。事件命名空间可以使用 .(英文句号)做分隔，并且可以使用命名空间通配符。

注意：Grunt目前并不发出任何事件，但是在你自己的task中仍然是有用的。

### [grunt.event.on](http://www.gruntjs.net/api/grunt.event" \l "grunt.event.on)

为指定的事件添加一个监听器，并将此监听器放倒监听器数组的尾部。

grunt.event.on(event, listener)

### [grunt.event.once](http://www.gruntjs.net/api/grunt.event" \l "grunt.event.once)

为指定的事件添加一个执行 **一次** 的监听器。次监听器只在事件头一次被触发后才会被调用，并且执行完之后被移除。

grunt.event.once(event, listener)

### [grunt.event.many](http://www.gruntjs.net/api/grunt.event" \l "grunt.event.many)

为指定的事件添加一个监听器，并在被移除之前执行 **n 次** 。

grunt.event.many(event, timesToListen, listener)

### [grunt.event.off](http://www.gruntjs.net/api/grunt.event" \l "grunt.event.off)

从监听器数组中移除所有监听某个指定事件的监听器。

grunt.event.off(event, listener)

### [grunt.event.removeAllListeners](http://www.gruntjs.net/api/grunt.event" \l "grunt.event.removealllisteners)

移除所有的监听器，或者某个指定事件的所有监听器。

grunt.event.removeAllListeners([event])

### [grunt.event.emit](http://www.gruntjs.net/api/grunt.event" \l "grunt.event.emit)

依次执行每一个可能监听此事件名的监听器，并将参数列表传给每个事件监听器。

grunt.event.emit(event, [arg1], [arg2], [...])

Found an error in the documentation? [File an issue](https://github.com/gruntjs/grunt-docs/issues).

# 4.[grunt.fail](http://www.gruntjs.net/api/grunt.fail#grunt.fail)

用于发生致命的错误时。

参见 [fail lib source](https://github.com/gruntjs/grunt/blob/master/lib/grunt/fail.js) 以获取更多信息。

## [The fail API](http://www.gruntjs.net/api/grunt.fail" \l "the-fail-api)

如果任务内部某些代码已经（或即将）崩溃，可能导致Grunt强行中止。参见 [退出码文档](http://www.gruntjs.net/exit-codes) 以获取完整的退出码（exit code）列表。

注意，任何带有 ☃ (unicode snowman) 标记的方法都可以直接通过 grunt 对象访问。参见 [API首页](http://www.gruntjs.net/grunt) 以获取更多信息。

### [grunt.fail.warn ☃](http://www.gruntjs.net/api/grunt.fail" \l "grunt.fail.warn)

显示一条警告信息然后立即退出Grunt。如果在命令行中指定了--force 选项，Grunt将继续执行后续任务。error 参数可以是字符串信息或error对象。

grunt.fail.warn(error [, errorcode])

如果在命令行中指定 --stack 参数，并且给定error对象，stack trace 将会被记入日志。

此方法还可以通过*grunt.warn*访问。

### [grunt.fail.fatal ☃](http://www.gruntjs.net/api/grunt.fail" \l "grunt.fail.fatal)

显示一条警告信息然后立即退出Grunt。error 参数可以是字符串信息或error对象。

grunt.fail.fatal(error [, errorcode])

如果在命令行中指定 --stack 参数，并且给定error对象，stack trace 将会被记入日志。

出现致命（fatal）错误时会发出一声蜂鸣声，除非指定了 --no-color 参数。

此方法还可以通过*grunt.fatal*访问。

Found an error in the documentation? [File an issue](https://github.com/gruntjs/grunt-docs/issues).

**5.**[**grunt.file**](http://www.gruntjs.net/api/grunt.file#grunt.file)

这里提供了很多用于读写文件、遍历文件系统和通过模式匹配查找文件的方法。其中很多方法都是Node.js中的文件操作函数的封装，但是提供了额外的错误处理、日志记录和字符编码转换。

*注意：所有的文件路径都是参照 Gruntfile 文件的相对路径，除非通过 grunt.file.setBase 函数或在命令行中指定 --base 参数改变当前工作目录。*

**字符编码**

**[grunt.file.defaultEncoding](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.defaultencoding)**

设置此属性可以改变所有 grunt.file 方法的默认编码。默认是 'utf8'。如果必须改变这个值，建议你在Gruntfile文件中尽可能早改变。

grunt.file.defaultEncoding = 'utf8';

**[grunt.file.preserveBOM](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.preservebom)**

*添加于 0.4.2 版本*

是否在 file.read 时保留字节顺序标记（BOM）。

grunt.file.preserveBOM = false;

**读写文件**

**[grunt.file.read](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.read)**

读取并返回文件的内容。返回值为一个字符串，如果 options.encoding 为 null ，则返回一个 [Buffer](http://nodejs.org/docs/latest/api/buffer.html)。

grunt.file.read(filepath [, options])

options 对象可以设置以下属性：

var options = {

// If an encoding is not specified, default to grunt.file.defaultEncoding.

// If specified as null, returns a non-decoded Buffer instead of a string.

encoding: encodingName

};

**[grunt.file.readJSON](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.readjson)**

读取一个文件的内容，将其按照JSON格式解析，返回结果。参见 grunt.file.read 获取其所支持的参数列表。

grunt.file.readJSON(filepath [, options])

**[grunt.file.readYAML](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.readyaml)**

读取一个文件的内容，将其按照YAML格式解析，返回结果。参见 grunt.file.read 获取其所支持的参数列表。

grunt.file.readYAML(filepath [, options])

**[grunt.file.write](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.write)**

将指定的内容写入文件中，如果需要，将创建文件路径中所有不存在的目录。字符串将按照指定的字符编码进行编码，[Buffers](http://nodejs.org/docs/latest/api/buffer.html) 将会按照指定的方式写入磁盘。

*如果指定了 --no-write 命令行参数，将不会真正写入文件。*

grunt.file.write(filepath, contents [, options])

options 对象可设置以下属性：

var options = {

// If an encoding is not specified, default to grunt.file.defaultEncoding.

// If `contents` is a Buffer, encoding is ignored.

encoding: encodingName

};

**[grunt.file.copy](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.copy)**

将原文件拷贝到指定路径，如果需要，将创建文件路径中所有不存在的目录

*如果指定了 --no-write 命令行参数，将不会真正写入文件。*

grunt.file.copy(srcpath, destpath [, options])

options 对象可设置以下属性：

var options = {

// If an encoding is not specified, default to grunt.file.defaultEncoding.

// If null, the `process` function will receive a Buffer instead of String.

encoding: encodingName,

// The source file contents, source file path, and destination file path

// are passed into this function, whose return value will be used as the

// destination file's contents. If this function returns `false`, the file

// copy will be aborted.

process: processFunction,

// These optional globbing patterns will be matched against the filepath

// (not the filename) using grunt.file.isMatch. If any specified globbing

// pattern matches, the file won't be processed via the `process` function.

// If `true` is specified, processing will be prevented.

noProcess: globbingPatterns

};

**[grunt.file.delete](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.delete)**

删除指定的文件。文件和目录会被依次递归删除。

*Will not delete the current working directory or files outside the current working directory unless the--force command-line option is specified.*

*如果指定了 --no-write 命令行参数，那么，文件路径将不会真的被删除。*

grunt.file.delete(filepath [, options])

options 对象只可以设置以下属性：

var options = {

// Enable deleting outside the current working directory. This option may

// be overridden by the --force command-line option.

force: true

};

**目录操作**

**[grunt.file.mkdir](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.mkdir)**

工作方式类似 mkdir -p。创建一个目录和所有的中间目录。如果没有指定 mode ，默认是0777 & (~process.umask()).

*如果没有 --no-write 命令行参数，目录不会被真正创建。*

grunt.file.mkdir(dirpath [, mode])

**[grunt.file.recurse](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.recurse)**

递归遍历整个目录，对每个文件都执行 callback 函数。

grunt.file.recurse(rootdir, callback)

callback 函数接收以下参数：

function callback(abspath, rootdir, subdir, filename) {

// The full path to the current file, which is nothing more than

// the rootdir + subdir + filename arguments, joined.

abspath

// The root director, as originally specified.

rootdir

// The current file's directory, relative to rootdir.

subdir

// The filename of the current file, without any directory parts.

filename

}

**模式匹配**

有时单独指定所有原始文件路径是不现实的，因此，Grunt通过内置的[node-glob](https://github.com/isaacs/node-glob) 库支持文件名 expansion (或者叫做 globbing) 。

参见 [配置任务](http://www.gruntjs.net/configuring-tasks) 指南中的 "Globbing patterns" 章节以获取 globbing pattern 实例。

**[grunt.file.expand](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.expand)**

返回包含匹配给定通配符模式的文件或者目录路径的特殊数组。这个方法接收一个逗号分割的匹配模式或者一个匹配模式数组。如果路径匹配模式以!开头，它会从返回的数组排除所匹配的项。模式是按指定的顺序进行处理的， 因此包含和排除文件的顺序是很重要的。

grunt.file.expand([options, ] patterns)

文件路径都是参照 Gruntfile 文件的相对路径，除非通过 grunt.file.setBase 或 --base 命令行参数修改了当前工作目录。

options 对象支持所有 [minimatch library](https://github.com/isaacs/minimatch) 的参数，也支持额外的一些，如下：

* filter E接受一个有效的 [fs.Stats 方法名](http://nodejs.org/docs/latest/api/fs.html#fs_class_fs_stats) 或者一个已经通过了src文件路径匹配的函数，这个函数会返回true或false。
* nonull 会保留src匹配模式，即使文件匹配失败。结合Grunt的-verbose标志，这个选项有助于文件路径问题的调试。
* matchBase 不带斜线的模式只会匹配基本的名称部分。例如，这会使\*.js就像\*\*/\*.js一样。
* cwd 会让模式相对于当前路径进行模式匹配，所有返回的文件路径也是相对于当前路径的。

**[grunt.file.expandMapping](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.expandmapping)**

返回一个src-dest文件映射对象的数组。通过所指定的模式来匹配每一个源文件，然后将匹配的文件路径加入指定的dest中(dest存放匹配的文件路径)。这个文件路径会按照指定的选项加工或者重命名过。 查看grunt.file.expand方法文档可以了解如何指定patterns和options

grunt.file.expandMapping(patterns, dest [, options])

*注意：这个方法可以用于以编程的方式针对多任务的情况生成一个files数组，它会优先使用在*[*配置任务*](http://configuring-tasks/)*指南中"动态构建文件对象"一节所描述的语法。*

除了支持那些grunt.file.expand方法之外，options对象还支持下面这些属性:

var options = {

// The directory from which patterns are matched. Any string specified as

// cwd is effectively stripped from the beginning of all matched paths.

cwd: String,

// Remove the path component from all matched src files. The src file path

// is still joined to the specified dest.

flatten: Boolean,

// Remove anything after (and including) either the first or last "." in the

// destination path (indicated by options.extDot), then append this value.

ext: String,

// \*Added in 0.4.3\*

// Indicates where the period demarcating the extension is located. Can take:

// - 'first' (extension begins after the first period in the file name)

// - 'last' (extension begins after the last period)

// Default: 'first'

extDot: String,

// If specified, this function will be responsible for returning the final

// dest filepath. By default, it joins dest and matchedSrcPath like so:

rename: function(dest, matchedSrcPath, options) {

return path.join(dest, matchedSrcPath);

}

};

**[grunt.file.match](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.match)**

针对一个或者多个文件路径来匹配一个或者多个匹配模式。返回一个特殊的数组，这个数组包含与指定的通配符模式任意匹配的所有文件路径。patterns和filepaths参数可以是一个单一的字符串或者也可以是一个字符串数组.如果匹配模式以!开头，就会从返回的数组从排除模式匹配的路径。模式会按指定的顺序进行处理，因此包含和排除文件的顺序是重要的。

grunt.file.match([options, ] patterns, filepaths)

options对象也支持[minimatch库](https://github.com/isaacs/minimatch)提供的所有选项。例如：如果options.matchBase为true，即使模式中不带斜线，这个模式也会匹配包含斜线的基本名称。例如：\*.js模式将匹配path/to/file.js文件路径。

**[grunt.file.isMatch](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.ismatch)**

这个方法与grunt.file.match方法包含同样的签名和逻辑，但是如果它匹配任意文件，就会简单的返回ture，否则返回false。

**判断文件类型**

**[grunt.file.exists](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.exists)**

检测给定的路径是否存在，返回boolean类型的值。

和Node.js 中的 [path.join](http://nodejs.org/docs/latest/api/path.html#path_path_join_path1_path2) 方法一样，此方法将所有参数连接在一起，并对结果做规范化。

grunt.file.exists(path1 [, path2 [, ...]])

**[grunt.file.isLink](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.islink)**

给定的路径是否是符号链接，返回boolean类型的值。

和 Node.js 中的 [path.join](http://nodejs.org/docs/latest/api/path.html#path_path_join_path1_path2) 方法一样，此方法此方法将所有参数连接在一起，并对结果做规范化。

grunt.file.isLink(path1 [, path2 [, ...]])

如果路径不存在则返回false。

**[grunt.file.isDir](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.isdir)**

指定的路径是否是一个目录？返回boolean类型的值。

和 Node.js 中的 [path.join](http://nodejs.org/docs/latest/api/path.html#path_path_join_path1_path2) 方法一样，此方法此方法将所有参数连接在一起，并对结果做规范化。

grunt.file.isDir(path1 [, path2 [, ...]])

如果路径不存在它也会返回false。

**[grunt.file.isFile](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.isfile)**

指定的路径是否是一个文件？ 返回boolean类型的值。

和 Node.js 中的 [path.join](http://nodejs.org/docs/latest/api/path.html#path_path_join_path1_path2) 方法一样，此方法此方法将所有参数连接在一起，并对结果做规范化。

grunt.file.isFile(path1 [, path2 [, ...]])

如果路径不存在将返回false。

**路径**

**[grunt.file.isPathAbsolute](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.ispathabsolute)**

指定的文件路径是否是绝对路径？ 返回boolean类型的值。

和 Node.js 中的 [path.join](http://nodejs.org/docs/latest/api/path.html#path_path_join_path1_path2) 方法一样，此方法此方法将所有参数连接在一起，并对结果做规范化。

grunt.file.isPathAbsolute(path1 [, path2 [, ...]])

**[grunt.file.arePathsEquivalent](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.arepathsequivalent)**

所有给出的路径是否都是同一个路径？返回boolean类型的值。

grunt.file.arePathsEquivalent(path1 [, path2 [, ...]])

**[grunt.file.doesPathContain](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.doespathcontain)**

所有descendant路径是否全部包含在指定的ancestor路径中？返回boolean类型的值。

*注意：不需要检查路径是否真的存在。*

grunt.file.doesPathContain(ancestorPath, descendantPath1 [, descendantPath2 [, ...]])

**[grunt.file.isPathCwd](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.ispathcwd)**

指定的文件路径是否是CWD？返回boolean类型的值。

和 Node.js 中的 [path.join](http://nodejs.org/docs/latest/api/path.html#path_path_join_path1_path2) 方法一样，此方法此方法将所有参数连接在一起，并对结果做规范化。

grunt.file.isPathCwd(path1 [, path2 [, ...]])

**[grunt.file.isPathInCwd](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.ispathincwd)**

指定的文件路径是否在在CWD中？注意：CWD不在CWD*中* 。返回boolean类型的值。

和 Node.js 中的 [path.join](http://nodejs.org/docs/latest/api/path.html#path_path_join_path1_path2) 方法一样，此方法此方法将所有参数连接在一起，并对结果做规范化。

grunt.file.isPathInCwd(path1 [, path2 [, ...]])

**[grunt.file.setBase](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.setbase)**

改变Grunt的当前工作目录(CWD)。默认情况下，所有文件路径都是参照 Gruntfile 文件的相对路径。此函数和 --base 命令行参数的工作方式一致。

grunt.file.setBase(path1 [, path2 [, ...]])

和 Node.js 中的 [path.join](http://nodejs.org/docs/latest/api/path.html#path_path_join_path1_path2) 方法一样，此方法此方法将所有参数连接在一起，并对结果做规范化。

**外部工具库**

*不建议使用*

**下面列出的所有外部工具库已经不再建议使用了。**

请使用 **npm** 管理项目中对第三方工具库的依赖。

例如，如果你需要使用 [Lo-Dash](https://npmjs.org/package/lodash)，首先通过 npm install lodash 安装，然后在 Gruntfile 文件中使用即可： var \_ = require('lodash');

**[grunt.file.glob](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.glob)**

*不建议使用*

[glob](https://github.com/isaacs/node-glob) - File globbing utility.

**[grunt.file.minimatch](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.minimatch)**

*不建议使用*

[minimatch](https://github.com/isaacs/minimatch) - File pattern matching utility.

**[grunt.file.findup](http://www.gruntjs.net/api/grunt.file" \l "grunt.file.findup)**

*不建议使用*

[findup-sync](https://github.com/cowboy/node-findup-sync) - Search upwards for matching file patterns.

Found an error in the documentation? [File an issue](https://github.com/gruntjs/grunt-docs/issues).

# 6.[grunt.log](http://www.gruntjs.net/api/grunt.log#grunt.log)

向控制台输出信息。

参见 [log lib source](https://github.com/gruntjs/grunt/blob/master/lib/grunt/log.js) 以获取更多信息。

## [The log API](http://www.gruntjs.net/api/grunt.log" \l "the-log-api)

Grunt的输出应当看上去一致、美观。因此， 就有了这些log方法和一些有用的模式。所有用于输出日志的方法都可以链式调用。

注意：只有在命令行中指定*--verbose*选项时，所有*grunt.verbose*中的方法才会输出日志，并且完全像*grunt.log*中的方法一样工作。

### [grunt.log.write / grunt.verbose.write](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.write-grunt.verbose.write)

输出指定的 msg 字符串日志，结尾不带换行符（newline）。

grunt.log.write(msg)

### [grunt.log.writeln / grunt.verbose.writeln](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.writeln-grunt.verbose.writeln)

输出指定的 msg 字符串日志，结尾带换行符（newline）。

grunt.log.writeln([msg])

### [grunt.log.error / grunt.verbose.error](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.error-grunt.verbose.error)

如果省略msg 字符串，将会以红色字体输出ERROR，否则输出>> msg，并且尾部带有换行符。

grunt.log.error([msg])

### [grunt.log.errorlns / grunt.verbose.errorlns](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.errorlns-grunt.verbose.errorlns)

使用grunt.log.error记录一个错误日志，使用grunt.log.wraptext可以将日志以每行80个字符的形式输出。

grunt.log.errorlns(msg)

### [grunt.log.ok / grunt.verbose.ok](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.ok-grunt.verbose.ok)

如果省略msg字符串，将会以绿色字体输出OK, 否则输出>> msg，并且尾部带有换行符。

grunt.log.ok([msg])

### [grunt.log.oklns / grunt.verbose.oklns](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.oklns-grunt.verbose.oklns)

使用grunt.log.ok记录一条ok消息，使用grunt.log.wraptext可以将日志以每行80个字符的形式输出。

grunt.log.oklns(msg)

### [grunt.log.subhead / grunt.verbose.subhead](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.subhead-grunt.verbose.subhead)

记录指定的msg字符串并**加粗**，尾部带有换行符。

grunt.log.subhead(msg)

### [grunt.log.writeflags / grunt.verbose.writeflags](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.writeflags-grunt.verbose.writeflags)

记录obj的属性列表（用于调试标志最好）。

grunt.log.writeflags(obj, prefix)

### [grunt.log.debug / grunt.verbose.debug](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.debug-grunt.verbose.debug)

记录一条调试信息，但是，仅当在命令行中指定 --debug选项时才会输出。

grunt.log.debug(msg)

## [Verbose 和 Notverbose](http://www.gruntjs.net/api/grunt.log" \l "verbose-notverbose)

所有grunt.verbose下可用的日志记录方法的工作都酷似它们所对应的grunt.log方法，但是它们只在指定--verbose命令行选项的情况下才一样。还有一个对应"notverbose"适用于grunt.log.notverbose和grunt.log.verbose.or。实际上，.or属性也可以用于在verbose和notverbose两者之间有效的进行切换。

### [grunt.verbose / grunt.log.verbose](http://www.gruntjs.net/api/grunt.log" \l "grunt.verbose-grunt.log.verbose)

这个对象包含grunt.log下的所有方法，但是只在指定--verbose命令行选项情况下它才会输出日志信息。

grunt.verbose

### [grunt.verbose.or / grunt.log.notverbose](http://www.gruntjs.net/api/grunt.log" \l "grunt.verbose.or-grunt.log.notverbose)

这个对象也包含grunt.log下的所有方法，但是只在不指定--verbose命令行选项情况下它才会输出日志信息。

grunt.verbose.or

## 工具方法

这些方法实际上不记录日志，它们只返回字符串，返回的字符串可以用于其他方法。

### [grunt.log.wordlist](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.wordlist)

Returns a comma-separated list of arr array items. arr 数组中的条目将会以逗号分割的形式返回。

grunt.log.wordlist(arr [, options])

options 对象拥有以下属性和默认值：

var options = {

// The separator string (can be colored).

separator: ', ',

// The array item color (specify false to not colorize).

color: 'cyan',

};

### [grunt.log.uncolor](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.uncolor)

从字符串中移除所有颜色信息，使其适合检测其 .length 或写入日志文件。、

grunt.log.uncolor(str)

### [grunt.log.wraptext](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.wraptext)

以 width 个字符为一组将 text 字符串进行分解并添加 \n 字符，除非绝对必要，否则将尽量确保不会从中间截断单词。

grunt.log.wraptext(width, text)

### [grunt.log.table](http://www.gruntjs.net/api/grunt.log" \l "grunt.log.table)

以 width 个字符为一组将 text 字符串进行分解。Wrap texts array of strings to columns widthscharacters wide. A wrapper for the grunt.log.wraptext method that can be used to generate output in columns.

grunt.log.table(widths, texts)

## 案例

通常的模式是，只有在 --verbose 模式或发生错误时才输出日志，如下所示：

grunt.registerTask('something', 'Do something interesting.', function(arg) {

var msg = 'Doing something...';

grunt.verbose.write(msg);

try {

doSomethingThatThrowsAnExceptionOnError(arg);

// Success!

grunt.verbose.ok();

} catch(e) {

// Something went wrong.

grunt.verbose.or.write(msg).error().error(e.message);

grunt.fail.warn('Something went wrong.');

}

});

解释以上代码：

1. grunt.verbose.write(msg); 只有在 --verbose 模式时才会输出日志信息（没有换行符）。
2. grunt.verbose.ok(); 以绿色输出日志信息，末尾输出换行符。
3. grunt.verbose.or.write(msg).error().error(e.message); 做了以下几件事：
   1. grunt.verbose.or.write(msg) 如果不在 --verbose 模式则输出日志信息，然后返回notverbose 对象。
   2. .error() 以红色输出ERROR日志，结尾输出换行符，然后返回 notverbose 对象。
   3. .error(e.message); 输出实际的错误信息（并返回 notverbose 对象）。
4. grunt.fail.warn('Something went wrong.'); 以嫩黄色输出警告信息。除非在命令行指定了--force 选项，否则输出退出码1，然后退出 Grunt。

查看 [grunt-contrib-\* 任务的源码](https://github.com/gruntjs) 以获取更多案例。

# 7.[grunt.task](http://www.gruntjs.net/api/grunt.task#grunt.task)

注册、执行和加载外部任务。

参见 [task lib source](https://github.com/gruntjs/grunt/blob/master/lib/grunt/task.js) 和 [task util lib source](https://github.com/gruntjs/grunt/blob/master/lib/util/task.js) 获取更多信息。

## [The task API](http://www.gruntjs.net/api/grunt.task" \l "the-task-api)

当一个任务正在执行时，Grunt 通过this 对象向此任务函数暴露了很多任务特定的属性和方法。参见 [深入任务内幕](http://www.gruntjs.net/inside-tasks)指南，这里可以找到完整的属性和方法列表。

很多属性和方法都可以通过 this 对象访问到。

注意，任何标记为的 ☃ (unicode snowman) 方法也可以直接通过 grunt 对象直接访问到。参见 [API首页](http://www.gruntjs.net/grunt)以获取更多信息。

## 创建任务

### [grunt.task.registerTask ☃](http://www.gruntjs.net/api/grunt.task" \l "grunt.task.registertask)

注册 "别名任务" 或 任务函数。此方法支持以下两种类型：

**别名任务**

如果指定了一个任务列表，那么，新注册的任务将会是这一个或多个任务的别名（alias）。当此"别名任务"执行时，taskList中的所有任务都将按指定的顺序依次执行。taskList 参数必须是一个任务数组。

grunt.task.registerTask(taskName, taskList)

下面这个案例展示的是定义一个"default" 任务，当执行 Grunt 且不通过参数指定任务时， "jshint"、 "qunit"、"concat" 和 "uglify" 任务将自动执行：

task.registerTask('default', ['jshint', 'qunit', 'concat', 'uglify']);

还可以指定任务的参数。下面的这个案例中，别名"dist"执行了 "concat" 和 "uglify" 这两个任务，并且都指定了"dist" 参数：

task.registerTask('dist', ['concat:dist', 'uglify:dist']);

**任务函数**

如果传入description和taskFunction，每当任务运行时，指定的函数（taskFunction）都会被执行。此外，当执行 grunt --help时，前面指定的描述（description）就会显示出来。特定任务的属性和方法在任务函数内部通过this对象的属性即可访问。如果任务函数返回false表示任务失败。

注意，grunt.task.registerMultiTask方法将稍候介绍，它可以被用于定义一种特殊类型的任务，即"复合任务"。

grunt.task.registerTask(taskName, description, taskFunction)

下面这个案例中，当 Grunt 运行grunt foo:testing:123时，日志输出foo, testing 123。如果运行这个任务时不带参数，如grunt foo，日志输出foo, no args。

grunt.task.registerTask('foo', 'A sample task that logs stuff.', function(arg1, arg2) {

if (arguments.length === 0) {

grunt.log.writeln(this.name + ", no args");

} else {

grunt.log.writeln(this.name + ", " + arg1 + " " + arg2);

}

});

参见 [创建任务](http://www.gruntjs.net/creating-tasks) 文档,查看更多任务实现案例和别名任务。

此方法还可以通过[*grunt.registerTask*](http://www.gruntjs.net/grunt)调用。

### [grunt.task.registerMultiTask ☃](http://www.gruntjs.net/api/grunt.task" \l "grunt.task.registermultitask)

注册一个 "复合任务（multi task）"。 复合任务是指在不指定目标（target）时，将依次执行其所包含的所有已命名的子属性（sub-properties） (也就是 目标) 。除了默认的属性和方法外，还可以通过this对象访问复合任务独有的属性。

大多数的contrib任务，包括 [jshint task](https://github.com/gruntjs/grunt-contrib-jshint)、[concat task](https://github.com/gruntjs/grunt-contrib-concat) 和 [uglify task](https://github.com/gruntjs/grunt-contrib-uglify) 都是复合任务。

grunt.task.registerMultiTask(taskName, description, taskFunction)

给定以下配置信息，当执行grunt log:foo时，下面的复合任务将输出日志foo: 1,2,3；当执行grunt log:bar时，将输出日志bar: hello world。如果只是执行grunt log，那么，将先输出日志foo: 1,2,3，然后是bar: hello world，最后是baz: false。

grunt.initConfig({

log: {

foo: [1, 2, 3],

bar: 'hello world',

baz: false

}

});

grunt.task.registerMultiTask('log', 'Log stuff.', function() {

grunt.log.writeln(this.target + ': ' + this.data);

});

参见 [创建任务](http://www.gruntjs.net/creating-tasks) 文档以获取更多复合任务的案例。

此方法还可以通过[*grunt.registerMultiTask*](http://www.gruntjs.net/grunt)调用。

### [grunt.task.requires](http://www.gruntjs.net/api/grunt.task" \l "grunt.task.requires)

Fail the task if some other task failed or never ran.

grunt.task.requires(taskName);

### [grunt.task.exists](http://www.gruntjs.net/api/grunt.task" \l "grunt.task.exists)

Added in 0.4.5

Check with the name, if a task exists in the registered tasks. Return a boolean.

grunt.task.exists(name)

### [grunt.task.renameTask ☃](http://www.gruntjs.net/api/grunt.task" \l "grunt.task.renametask)

重命名任务。如果你希望覆盖某个任务的默认行为，并且希望保留原来的名字，这个函数将会很有用。

注意，如果一个任务已经被重命名了，[*this.name*](http://www.gruntjs.net/inside-tasks#this.name)和[*this.nameArgs*](http://www.gruntjs.net/nside-tasks#this.nameargs)属性都会发生相应的变化。

grunt.task.renameTask(oldname, newname)

此方法还可以通过[*grunt.renameTask*](http://www.gruntjs.net/grunt)调用。

## 加载外部定义的任务

对于多数项目来说，[Gruntfile](http://www.gruntjs.net/getting-started) 文件中可能会定义很多任务。对于大型项目或者需要在多个项目中共享任务的情况，可以从一个或多个外部目录加载任务，或者从npm安装的 Grunt 插件加载任务。

### [grunt.task.loadTasks ☃](http://www.gruntjs.net/api/grunt.task" \l "grunt.task.loadtasks)

从指定的目录（注意：相对于 [Gruntfile](http://www.gruntjs.net/getting-started) 所在目录）加载任务相关的文件。此方法可以从本地Grunt插件加载任务相关的文件，只需指定包含"tasks"子目录的插件目录即可。

grunt.task.loadTasks(tasksPath)

此方法还可以通过[*grunt.loadTasks*](http://www.gruntjs.net/grunt)调用。

### [grunt.task.loadNpmTasks ☃](http://www.gruntjs.net/api/grunt.task" \l "grunt.task.loadnpmtasks)

从指定的 Grunt 插件中加载任务。此插件必须通过npm安装到本地，并且是参照 [Gruntfile](http://www.gruntjs.net/getting-started) 文件的相对路径。Grunt插件还可以通过 [grunt-init grunt插件模版](https://github.com/gruntjs/grunt-init)创建： grunt init:gruntplugin。

grunt.task.loadNpmTasks(pluginName)

此方法还可以通过[*grunt.loadNpmTasks*](http://www.gruntjs.net/grunt)调用。

## 队列任务

Grunt自动将命令行中指定的任务加入队列并执行，但是，一些独特的任务可以向队列中加入额外需要执行的任务。

### [grunt.task.run](http://www.gruntjs.net/api/grunt.task" \l "grunt.task.run)

将一个或多个任务放入队列中。 taskList 中的每个任务都会按照其在队列中的顺序，在当前任务执行完毕后立即执行。任务列表可以是一个任务数组或单个任务。

grunt.task.run(taskList)

### [grunt.task.clearQueue](http://www.gruntjs.net/api/grunt.task" \l "grunt.task.clearqueue)

完全清空任务队列。除非将额外的任务加入队列，否则将不会执行任何任务。

grunt.task.clearQueue()

### [grunt.task.normalizeMultiTaskFiles](http://www.gruntjs.net/api/grunt.task" \l "grunt.task.normalizemultitaskfiles)

将目标（target）的配置对象标准化为一个src-dest文件映射数组。此方法在内部由复合任务系统的[this.files / grunt.task.current.files](http://www.gruntjs.net/grunt.task#wiki-this-files)属性调用。

grunt.task.normalizeMultiTaskFiles(data [, targetname])

Found an error in the documentation? [File an issue](https://github.com/gruntjs/grunt-docs/issues).