**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.** [**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" \l "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).