

Advanced GruntJS

Rong Shen (沈嵘)
rong.shen@nxmix.com
@Jacobubu



我是谁？

- > 4 年程序员 - CS&S
- 6 年顾问 - Microsoft
- 6 年售前顾问 - Microsoft
- 1 产品总监 - k-touch
- 2 年杂役? -  NEXT EXPERIENCE INTERACTIVE nxmix.com

本节目标

- 了解 GrungJS 的设计哲学和关键概念
- 节省你的阅读和探索的时间
- 便于快速复习
- Level = 300 (技术贴, 没有市场营销)

课程挑战

40 分钟值 430 元？

对不同程度的读者

从未用过



演示

用过，但是没有深入了解



要点



事后阅读

避免使用英文

便于会后阅读

节省1, 2天的工时

GRUNTJS.COM

Ben Alman (cowboy)

github.com/cowboy
benalman.com



我们在每一个项目中使用



<http://nextday.im>

埃及

这次流浪不同以往

Alexandria

它是一次身不由己的浪漫

麻雀不大，事情不少

- 生成每日图片
- 生成每日 Preview 图片
- Image Minification
- CoffeeScript to JS
- Stylus to CSS
- Uglify Javascript
- CSS Minification
- Embedding CSS & JS into Jade File
- Jade to HTML
- HTML Minification
- 部署到测试或生产环境



第三方的 Plugins

```
"grunt-contrib-clean": "~0.4.1",  
"grunt-contrib-coffee": "~0.7.0",  
"grunt-contrib-stylus": "~0.5.0",  
"grunt-contrib-jade": "~0.7.0",  
"grunt-contrib-imagemin": "~0.1.4",  
"grunt-contrib-concat": "~0.3.0",  
"grunt-contrib-uglify": "~0.2.2",  
"grunt-contrib-htmlmin": "~0.1.3",  
"grunt-contrib-cssmin": "~0.6.1",  
"grunt-contrib-copy": "~0.4.1",  
"grunt-sync": "0.0.4",  
"grunt-base64": "~0.1.0",  
"grunt-rsync": "~0.1.1",  
"grunt-contrib-watch": "~0.4.4"
```

GruntJS 节省了我大量时间!

为什么节省时间?

- “自动化”领域的最佳重用模型
- 站在 npm 的肩膀上
- 大量的既有 Plugins
- 用 JavaScript 来构建 JavaScript

关键概念

- Grunt CLI 和 Grunt Module
- Task 和 Target
- File Mapping
- Grunt Object 

Grunt CLI 和 Grunt Module

安装 Grunt CLI

```
sudo npm install -g grunt-cli
```

在你的工程目录安装 Grunt Module

```
~/myproject/npm install grunt
```

Grunt CLI 是 Grunt Module 的 Launcher

运行 grunt

~/myproject/grunt

运行的是 Grunt CLI, 不是 Grunt Module



Symbolic Link

通过 Symbolic Link 找到 Grunt CLI 的 JS 程序

/usr/local/bin/grunt -> lib/node_modules/grunt-cli/bin/grunt



Grunt CLI 从 CWD 开始, Find-Up node_modules/grunt (向上查找 **Grunt Module**), 载入并传入命令行参数执行

~/myproject/node_modules/grunt

Grunt Module 也从 CWD 开始, Find-Up **Gruntfile.js** or coffee, 载入并执行

要点

- Grunt CLI 和 Grunt Module 是分离的
 - 每个项目都可以使用不同的 Grunt Module 版本，或者一系列的子项目共用父项目的 Grunt Module
 - **Grunt CLI** 负责实现公共任务的入口逻辑: help, version, bash completion
 - **Grunt CLI** 负责把命令行参数传递给找到的 **Grunt Module** 完成剩余部分
- 随后 **Grunt Module** 会去寻找“最近”（Find-Up）的 Gruntfile，载入并执行 Default Task





```
#!/usr/bin/env node
var findup = require('findup-sync');
var resolve = require('resolve').sync;

// Internal libs.
var options = require('../lib/cli').options;
var completion = require('../lib/completion');
var info = require('../lib/info');
var path = require('path');

var basedir = process.cwd();
var gruntpath;

// Do stuff based on CLI options.
if ('completion' in options) {
  completion.print(options.completion);
} else if (options.version) {
  info.version();
} else if (options.gruntfile) {
  basedir = path.resolve(path.dirname(options.gruntfile));
}
```

作者原创同步版本 findup

实现 Node.js require.resolve 的算法，但是可以通过配置实现更灵活地查找

http://nodejs.org/api/modules.html#modules_all_together

解析命令行参数

安装和与某个 Shell 匹配的 auto completion 执行脚本

目前缺省仅提供BASH的Auto Completion脚本，实现对当前目录（或之上目录）的Gruntfile中的tasks实现自动完成
需要在.bashrc中添加 `eval "$(grunt --completion=bash)"`

命令行可以指定特定的 gruntfile 的路径以替换从当前目录查找

Global Grunt File

/usr/local/lib/node_modules/grunt-cli/bin/grunt



```
completion.print(options.completion);
} else if (options.version) {
  info.version();
} else if (options.gruntfile) {
  basedir = path.resolve(path.dirname(options.gruntfile));
}
```

```
try {
  gruntpath = resolve('grunt', {basedir: basedir});
} catch (ex) {
  gruntpath = findup('lib/grunt.js');
  // No grunt install found!
  if (!gruntpath) {
    if (options.version) { process.exit(); }
    if (options.help) { info.help(); }
    info.fatal('Unable to find local grunt.', 99);
  }
}
```

```
// Everything looks good. Require local grunt and run it.
require(gruntpath).cli();
```

从 cwd 或者 指定目录中的 'node_modules' 中查找 grunt module, 如果失败, 向上层目录的 'node_modules' 中查找

如果都找不到, 则从当前目录开始, 向上查找 'lib/grunt.js', 这是为了 grunt module 内查找 'lib/grunt.js'

找到了, 载入 grunt module, 并且调用其 cli 函数, 传入外部的命令行参数

gruntfile.js

 00-gettingStarted



```
module.export= 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'  
      }  
    }  
  });  
  
  // Load the plugin that provides the "uglify" task.  
  grunt.loadNpmTasks('grunt-contrib-uglify');  
  
  // Default task(s).  
  grunt.registerTask('default', ['uglify']);  
};
```

每个 gruntfile 都必须有一个输出函数，运行时传入 **grunt** 对象

为 **uglify** task 指定配置 (必须和注册的 Task 同名)。Options 是保留属性，不会被理解为 target

使用 **LoDash** Template 语法

build 是 uglify 的 target, 一个 task 可以有多个 targets, 每个 target 可以享有独立的 options 和文件集合

src 指定源文件集合

dest 在这里指定输出文件

从 CWD 下面的 node_modules 中载入指定的 Plugin Module (没有用 find-up 的方式)

uglify 这个 task 是在 **grunt-contrib-uglify** Module 里注册的

需要 `npm install grunt-contrib-uglify --save-dev`

注册 default task, 使之顺序执行数组中的 tasks。'default' 也称之为 ['uglify'] tasks 数组的别名

Task 和 Target

- Task 在运行前需要注册和配置
 - 注册：
 - registerTask, registerMultiTask, loadNpmTasks, loadTasks
 - 配置：
 - File-Set 和 Options
 - 或者先设定一个或多个 Target, 每个 Target 包含独立的 File-Set 和 Options
- Alias (别名) Task 则可以将 n 个 Targets 封装为新的 Task

多个 Targets

```
grunt.initConfig({
  uglify: {
    foo: {
      options: {},
      files: {}
    },
    bar: {
      options: {},
      files: {}
    }
  }
});

grunt.registerTask('default', ['uglify']);
grunt.registerTask('shortcut', ['uglify:bar']);
```

~/myproject/grunt -> 顺序执行 uglify:foo 和 uglify:bar

~/myproject/grunt uglify:foo

~/myproject/grunt shortcut -> 执行 uglify:bar

Options 和 File-Set

 02-sloc

👁️ 精简格式 (Compact Format)



```
module.exports = function(grunt) {
```

```
  grunt.config.init({
    sloc: {
      options: {
        reportType: 'stdout',
        jsonSpace: '..'
      },
      all: {
        src: 'files/**'
      },
      js: {
        options: {
          reportType: 'json'
        },
        src: 'files/**/*.js'
      }
    }
  })
}
```

使用 minimatch 的算法来匹配文件，但是支持多个 pattern。后续的 pattern 从前面的过滤结果继续过滤

<https://github.com/isaacs/minimatch>
括号扩展、glob 和 globstar

```
  grunt.registerMultiTask('sloc', 'Source lines of code', function() {
```

```
    var options = this.options({
      reportType: 'stdout', //stdout, json
      jsonSpace: ' '
    });
```

this.options 混合缺省参数和 config 的 Options。在 target 中调用 **this.options**, grunt 将自动向上寻找并合并 task 的 **options** 配置

~ 省略 ~

```
    grunt.verbose.writeflags(options, 'Options');
```

grunt.log 和 grunt.verbose 分别实现了各种输出工具函数。例如 grunt.log.error, grunt.log.ok 等。输出的内容符合 grunt 的 color scheme。



```
var options = this.options({
  reportType: 'stdout', //stdout, json
  jsonSpace: '  '
});
```

~ 省略 ~

```
grunt.verbose.writeflags(options, 'Options');
```

```
this.filesSrc.forEach(function(filepath) { -----
```

```
  if (!grunt.file.exists(filepath)) { return; }
  -----
```

```
  var ext = path.extname(filepath);
  if (exts.indexOf(ext) < 0) { return; }
```

```
  var source = grunt.file.read(filepath, {encoding: 'utf8'});
  var stats = sloc(source, ext.substr(1, ext.length));
```

```
  count.loc += stats.loc;
```

this.filesSrc 包含所有匹配出的文件名
(包含相对路径-以 gruntfile 所在目录为
base) 的数组。

使用 **this.filesSrc** 适合只读的 task 或者无
需生成结果文件的场景。否则, 应该使用
this.files。

grunt 中的 file 对象包括一系列的 工具函
数, 例如: **grunt.file.setBase** 可以设置新
的 base 目录; **grunt.file.expand** 可以自
己实现文件匹配

Mapping 和 Expanding

 03-base64

Files Object

```
grunt.config.init({
  base64: {
    main: {
      files: {
        'dist/1-2.b64': ['files/{1..2}.txt'],
        'dist/3.b64': ['files/3.txt']
      }
    }
  }
});
```

适合多个源文件产生一个目标文件的场景

File Array Format

```
grunt.config.init({
  base64: {
    main: {
      files: [
        {
          cwd: 'files/'
          , src: ['{1..2}.txt']
          , dest: 'dist/1-2.b64'
          }
        , {
          src: ['files/3.txt']
          , dest: 'dist/3.b64'
          , filter: 'isFile'
          }
      ]
    }
  }
});
```

可以指定更多的配置参数

<http://gruntjs.com/configuring-tasks#files-array-format>

不同的源文件集合，每个集合产生一个目标文件

Expanding

```
grunt.config.init({
  expand: {
    files: [
      {
        expand: true
        , cwd: 'files/'
        , src: ['*.png']
        , dest: 'dist/'
        , ext: '.png.b64'
        , flatten: 'false'
      }
      , {
        expand: true
        , cwd: 'files/'
        , src: ['*.jpg']
        , dest: 'dist/'
        , ext: '.jpg.b64'
      }
    ]
  }
});
```

dest 的文件名将根据 src 的文件名来确定
src 从哪个路径开始查找

dest 的文件名的扩展名

输出文件在 dest 目录下是否保留在
src 目录结构，缺省是 false

为过滤出的源文件分别生成目标文件

this.files

```
this.files.forEach(function(mapping) {  
  var contents = mapping.src.map(function(filepath) {  
    return grunt.file.read(filepath);  
  }).join('');  
  
  grunt.file.write(mapping.dest  
    , (new Buffer(contents)).toString('base64'));  
});
```

files 包括一个 mapping 数组，每个 mapping 元素又包括一个 src 数组和 dest 属性

src 是过滤出的源文件名的数组，dest 则是对应的目标文件

要点

- **this.files[x].src** 和 **this.files[x].dest** 之间可以是 1 vs. 1 或者 N vs. 1 的关系
 - 1 vs. 1 用例: 将 .coffee 文件编译为 .js 文件, 或者将 .png 文件压缩; 可以选择是否保有源目录结构
 - N vs. 1 用例: 将多个 .js 文件合并为一个 .js 文件
- 如果无需输出结果文件, 则可以用简化版的 **this.filesSrc**
 - 例如: 删除某些文件, 统计行数, rsync 文件到服务器...



👁 Basic Task

```
grunt.registerTask('default', 'log');
```

```
grunt.registerTask('foo', 'A 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);
  }
});
```

- Basic Task 的限制
 - 可以有 Config，但是不支持 Target
 - 不支持 files 对象

Task 异步执行

熟悉的模式

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

```
    var done = this.async();
```

----- 强制 Grunt 将任务的执行转入“异步”模式，获得“Done”函数的句柄

```
    // Run some sync stuff.
```

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

```
    // And some async stuff.
```

```
    setTimeout(function() {
```

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

```
        done();
```

----- 告诉 Grunt 当前 Task 执行完毕

```
    }, 1000);
```

```
});
```

👁️ 'this' 里有什么？

🗨️ 01-multiTask

```
~/myproject/grunt log:bar
```

<code>this.name</code>	当前 Task 的名字, 例如: 'log'
<code>this.target</code>	当前 Target 的名字, 例如: 'log:bar'
<code>this.data</code>	'hello world'
<code>this.async()</code>	告诉 Grunt 当前的 task 是一个异步 task
<code>this.requires(taskList)</code>	当前 task 执行之前, 必须先执行哪些 tasks
<code>this.requiresConfig()</code>	当前 task 在执行之前, 哪些配置信息必须先被初始化
<code>this.options()</code>	根据 Config 和 缺省值构造当前 task 执行的 options 对象
<code>this.files [Getter]</code>	返回 Mapping 的数组, 每个 Mapping 元素包含一个 src 文件集合, 以及一个 dest 文件名
<code>this.filesSrc [Getter]</code>	返回 src 文件集合

要点

- 重用的是 Task
- 变化的是 Task 的 Config、Src File Set 和 Dest File Set
- 每一种变化类型组织为一个 Target
- Alias 可以重用既有的 Task 和 Target
- Template 让 Config 实时反应运行时的状态



Plugins

Grunt Plugins

This plugin listing is automatically generated from the npm module database. Officially maintained "contrib" plugins are marked with a star ★ icon and are shown at the top of every search by default.

In order for a Grunt plugin to be listed here, it must be published on [npm](#) with the [gruntplugin](#) keyword. Additionally, we recommend that you use the [gruntplugin grunt-init template](#) when creating a Grunt plugin.

★	contrib by Grunt Team The entire grunt-contrib suite.	Modified 23 days ago Grunt Version ~0.4.0
★	contrib-clean by Grunt Team Clean files and folders.	Modified 4 months ago Grunt Version ~0.4.0
★	contrib-coffee by Grunt Team Compile CoffeeScript files to JavaScript.	Modified 7 months ago Grunt Version ~0.4.0

Plugins

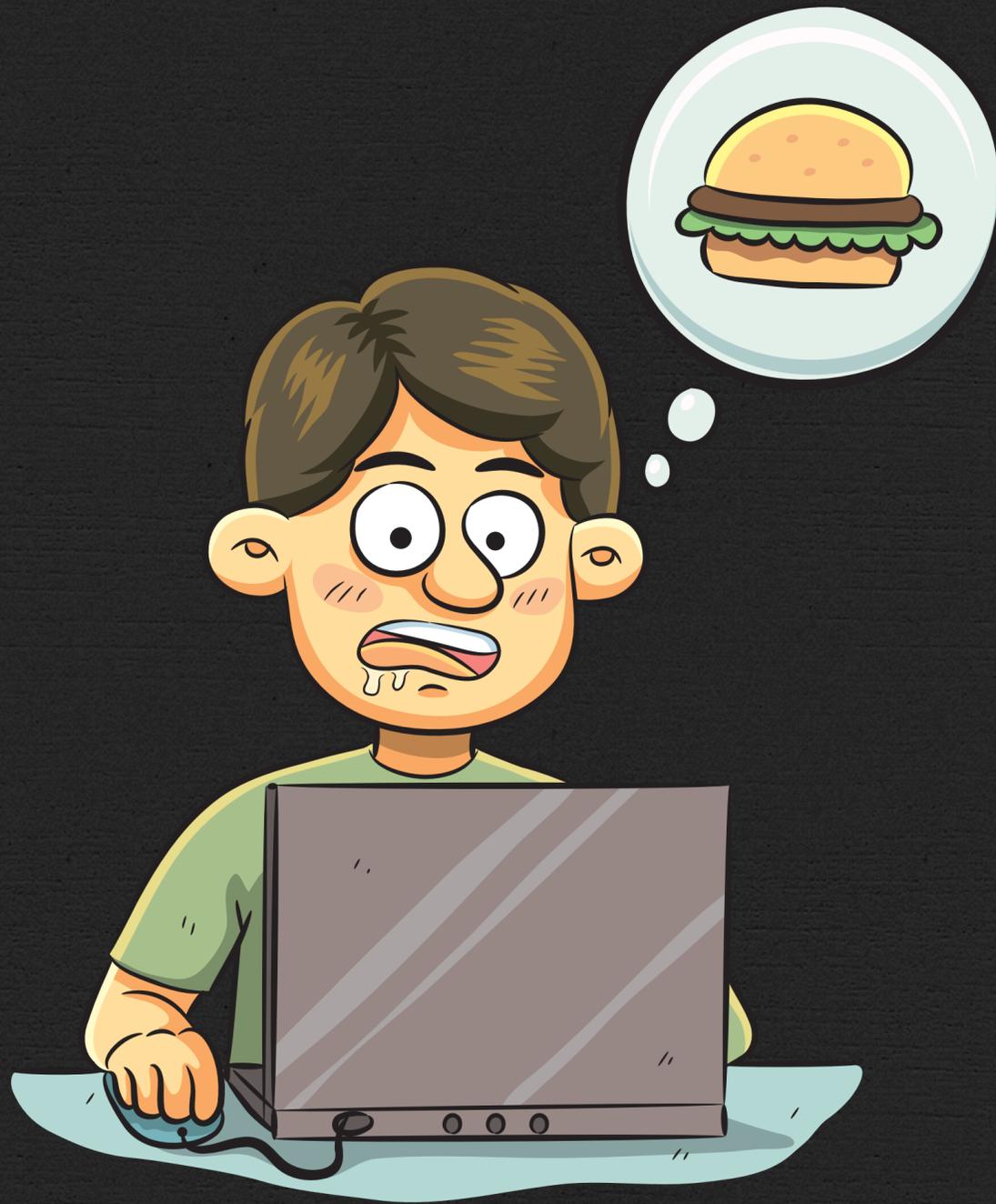
- 一个 npm module, 以 'grunt-' 为名称前缀
- 包含一个 tasks 子目录
 - 包含一个或多个 task 文件
- 相应的 package.json
- 发布到 npm

生成 Plugin 的框架

1. 通过 `sudo npm install -g grunt-init` 安装 `grunt-init`
2. 安装 `gruntplugin` 模板
 - ★ `git clone git://github.com/gruntjs/grunt-init-gruntplugin.git`
`~/grunt-init/gruntplugin`
3. 在一个新目录中运行 `grunt-init gruntplugin`
 - 一系列的问答，例如项目名称，github 地址等，帮你生成文件名和 `package.json`
4. 运行 `npm install`，准备你的开发环境
5. 编写你的 plugin，放置到 `tasks` 子目录中，以及单元测试代码等
6. 用 `npm publish` 发布你的 Grunt plugin 到 npm!

Plugins 保存数据建议

- Plugins 的 Task 生成的临时数据，请保存在 OS 级别的临时目录中 (利用 npm modules 'temporary', 'tmp')
- 如果是需要重复使用的数据 (例如 OAuth Token) ，可以保存在：
 - `~/myproject/.grunt/[npm_module_name]` 中
 - Plugin 自己决定什么时候进行清理



时间到了，下课!

全文总结

- 写你自己的 Plug-ins，并且尽量重用它们
- 尽可能将 Plug-ins 发布到 npm
- 尽量使用 Grunt 自己的API，尊重 Grunt 的风格和习惯
- 不断重构你的 Plugins，Task，Target 的组织方式
 - 灵活意味着总可能有更优解



Slides 和 Codes 如下

https://github.com/jacobbubu/Advanced_GruntJS

参考

<http://gruntjs.com>

<https://github.com/gruntjs/grunt>

<https://github.com/rhiokim/grunt-sloc/blob/master/tasks/sloc.js>

```
module.
```



'grunt' 对象

```
// Project configuration.
```

```
grunt
```

```
  pkg:
```

```
    options: {
```

```
      banner:
```

```
    },
```

```
    build: {
```

```
      src:
```

```
      dest:
```

```
    }
```

```
  }
```

```
});
```

grunt.config	获取 Gruntfile 中定义的配置值，可以取得模板匹配后的，也可以取得原始值；也提供动态设置 Config 的方法
grunt.event	Node Event的增强版本 (https://github.com/hij1nx/EventEmitter2) grunt 自身并没有 emit 任何事件，Plugins 和你自己的 Task 之间可以共用这套机制（希望不是过度设计:）
grunt.fail	因为错误或异常强制退出Grunt的执行过程，并记录故障信息
grunt.file	文件和目录处理的帮助函数，见后续说明
grunt.log	符合grunt风格（颜色和格式）的log helper，并且遵循命令行参数的控制
grunt.option	获取命令行参数，或在不同的 task 之间传递数据
grunt.task	Register, Load 和 Run 外部的 Tasks；设置依赖断言
grunt.template	基于LoDash Template 完成模板匹配的方法集，以及日期格式化方法
grunt.util	一系列难以归类的工具函数

grunt.config

初始化当前工程的配置对象, `grunt.config.init == grunt.initConfig`

```
grunt.config.init({
  log: {
    today: '<%= grunt.template.today() %>'
  }
});
```

获取配置值, `grunt.config(prop)` 或者 `grunt.config.get(prop)`

```
grunt.config('log.today');
grunt.config(['log', 'today']);
// Fri Nov 08 2013 12:50:39

grunt.config('concat.dist/built\\.js');
```

grunt.config - 续

获取配置的原始值, `grunt.config.getRaw(prop)`

```
grunt.config.getRaw('log.today');  
// grunt.template.today()
```

动态设置配置值, `grunt.config.set`

```
grunt.config.set('log.now', '<%= new Date()%>');
```

对象树会被自动建立

断言某些配置必须存在, `grunt.config.requires(prop [, prop [, ...]])`

grunt.file

<code>grunt.file.defaultEncoding</code>	设置缺省的文件编码，缺省是 'utf8'
<code>grunt.file.read</code>	同步读取文件内容，返回 string。如果 options 的 encoding 是 null，则返回 buffer <code>grunt.file.read(filepath [, options])</code>
<code>grunt.file.readJSON</code>	同步读取文件内容，以 JSON 解析，返回 JS 对象
<code>grunt.file.readYAML</code>	同步读取文件内容，以 YAML 解析，返回 JS 对象
<code>grunt.file.write</code>	同步写文件，建立必要的中间目录。使用这个方法还有一个好处是，如果运行 grunt 时又 <code>-no-write</code> 命令行参数，则不会真正写到磁盘
<code>grunt.file.copy</code>	复制文件，支持通配符和必要的中间目录的建立，也尊重 <code>-no-write</code> 命令行参数
<code>grunt.file.delete</code>	删除文件或目录（包括子目录），也尊重 <code>-no-write</code> 命令行参数

grunt.file - 续

<code>grunt.file.mkdir</code>	以类似 <code>mkdir -p</code> 的方式创建目录，尊重 <code>--no-write</code> 命令行参数 <code>grunt.file.mkdir(dirpath [, mode])</code>
<code>grunt.file.recurse</code>	递归一个目录，为每个文件调用 <code>callback</code> <code>grunt.file.recurse(rootdir, callback)</code>
<code>grunt.file.expand</code>	返回匹配的文件集合，和 <code>this.filesSrc</code> 的算法一样，支持 <code>inclusive</code> 和 <code>exclusive</code> 等 <code>grunt.file.expand([options,] patterns)</code>
<code>grunt.file.expandMapping</code>	返回 <code>src-dest file mapping</code> 对象，和 <code>this.files</code> 的算法一样 <code>grunt.file.expandMapping(patterns, dest [, options])</code>
<code>grunt.file.match</code>	对一个或多个路径进行通配符匹配，返回匹配的文件路径数组 <code>grunt.file.match([options,] patterns, filepaths)</code>
<code>grunt.file.isMatch</code>	和 <code>grunt.file.match</code> 算法一样，只是返回 <code>Boolean</code> 值来表示是否找到匹配的文件 <code>grunt.file.isMatch([options,] patterns, filepaths)</code>
<code>grunt.file.exists</code>	给定的路径是否存在？ <code>path.join(path1, path2...)</code> 会先执行 <code>grunt.file.exists(path1 [, path2 [, ...]])</code>
<code>grunt.file.isDir</code> ， <code>grunt.file.isFile</code> ， <code>grunt.file.isLink</code>	给定路径是否是一个目录，文件， <code>Symbolic Link</code>

grunt.file - 续

<code>grunt.file.isPathAbsolute</code>	给定路径是否为绝对路径；会先执行 <code>path.join(path1, path2...)</code> <code>grunt.file.isPathAbsolute(path1 [, path2 [, ...]])</code>
<code>grunt.file.arePathsEquivalent</code>	以下路径是否是同一个路径 <code>grunt.file.arePathsEquivalent(path1 [, path2 [, ...]])</code>
<code>grunt.file.doesPathContain</code>	后续路径是否都是第一个路径的子目录 <code>grunt.file.doesPathContain(ancestorPath, descendantPath1 [, descendantPath2 [, ...]])</code>
<code>grunt.file.isPathCwd</code>	给定的文件路径是否是 CWD, 会先执行 <code>path.join(path1, path2...)</code> <code>grunt.file.isPathCwd(path1 [, path2 [, ...]])</code>
<code>grunt.file.isPathInCwd</code>	给定的路径是否是 CWD 的子目录，同样会先 <code>path.join</code>
<code>grunt.file.setBase</code>	改变 grunt 执行时的工作目录（CWD）；Plugins 要谨慎使用这个方法。

grunt.option

命令行参数参数会被自动填入 `grunt.option`

```
grunt deploy --dest=staging
```

```
var dest = grunt.option('dest');  
// dest = "staging"
```

Target 在执行时可以设置 `grunt.option`，以在 Task:Target 之间传递参数

```
grunt.option('staging', false);  
var isStaging = grunt.option('staging');  
// isStaging === false  
  
// 'no-' 是语法糖，自动为后面的 key 值取反  
var isDev = grunt.option('no-staging');  
// isDev === true
```

grunt.task

<code>grunt.task.registerTask</code> <code>grunt.registerTask</code>	注册一个 Task Function 或者为既有的 Tasks 设置别名
<code>grunt.task.registerMultiTask</code> <code>grunt.registerMultiTask</code>	注册一个 'Multi Task'，即支持多 Targets 配置的 Task
<code>grunt.task.renameTask</code> <code>grunt.renameTask</code>	为既有的 Task 改名
<code>grunt.task.loadTasks</code> <code>grunt.loadTasks</code>	在载入指定目录下的所有 Plugins
<code>grunt.task.loadNpmTasks</code> <code>grunt.loadNpmTasks</code>	从 Gruntfile 所在目录的 <code>node_modules</code> 下面载入 Plugins
<code>grunt.task.run</code>	将一个或多个 Task 放入执行队列。这一般是用于在一个 Task 执行过程中，调用另一个 Task 执行的需求
<code>grunt.task.clearQueue</code>	清空 Task 执行队列
<code>grunt.task.normalizeMultiTaskFile</code>	将 target 中的文件集合配置，转换为标准的 <code>src-dest</code> 文件映射对象。 <code>this.files</code> 就是用这个函数生成的

grunt.template

grunt.template.process	<p>模板匹配</p> <pre>var obj = { foo: 'c', bar: 'b<%= foo %>d', baz: 'a<%= bar %>e' }; grunt.template.process('<%= baz %>', {data: obj}) // 'abcde'</pre>
grunt.template.setDelimiters	<p>启用更多的LoDash 预设的分隔符，name 是 LoDash 预设的分隔符的名称</p> <pre>grunt.template.setDelimiters(name)</pre> <p>http://lodash.com/docs/#template</p>
grunt.template.addDelimiters	<p>设置 template 的前后识别符</p> <pre>grunt.template.addDelimiters(name, opener, closer)</pre>
grunt.template.date	<p>格式化某个日期时间值</p> <pre>grunt.template.date(847602000000, 'yyyy-mm-dd') // '1996-11-10'</pre> <p>https://github.com/felixge/node-dateformat</p>
grunt.template.today	<p>格式化今天的日期，是 grunt.template.date(new Date()) 的语法糖</p>

grunt.util

<code>grunt.util.kindOf(value)</code>	返回一个对象的类型, 'number', 'string', 'boolean', 'function', 'regexp', 'array', 'date', 'error', 'null', 'undefined'
<code>grunt.util.error</code>	返回一个 Error 对象的实例 (可以被 throw) <code>grunt.util.error(message or errObj [, origError])</code>
<code>grunt.util.linefeed</code>	根据当前的 OS 返回换行符 <code>\r\n</code> on Windows <code>\n</code> on Mac and Linux
<code>grunt.util.normalizelf</code>	根据当前的 OS, 给输入的字符串添加换行符 <code>grunt.util.normalizelf(string)</code>
<code>grunt.util.recurse</code>	递归一个对象或者数组, 对其中的每一个非 Object 值调用 callback, 对其中的每个值都会调用 continue 来判断是否需要继续执行 <code>grunt.util.recurse(object, callback, continue)</code>
<code>grunt.util.repeat</code>	返回一个重复 n 次的字符串 <code>grunt.util.repeat(n, str)</code>
<code>grunt.util.pluralize</code>	<code>grunt.util.pluralize(1, 'child/children', '/')</code>
<code>grunt.util.spawn</code>	产生一个子进程, 获得其 stdout, stderr 和 exit code, 见下页
<code>grunt.util.toArray</code>	获取grunt中的用户定义数据, 缺省是模板填充值后的, 也可以获得原始定义
<code>grunt.util.callbackify</code>	将同步和异步函数都封装为异步回调方式

grunt.util.spawn

grunt.util.spawn(options, doneFunction)

```
var options = {
  cmd: commandToExecute,
  // 如果设置为 true (缺省是 false) , 将使用当前的 grunt bin 来启动子进程
  grunt: boolean,
  // 命令行参数
  args: arrayOfArguments,
  // Additional options for the Node.js child_process spawn method.
  opts: nodeSpawnOptions,
  // 如果出现错误, fallback 的值将被用来替代原始的错误值
  fallback: fallbackValue
};
```

```
function doneFunction(error, result, code) {
  // 如果 exit code 没有非零, 并且没有设定 fallback, 将返回 error 对象; 否则为 null
  error
  // result object 包含 .stdout, .stderr, .code (exit code) 属性
  result
  // 如果 exit code 是 0, 那么 String(result) 是 stdout 的内容。如果 exit code 非 0, 并且
  // 制定了 fallback, String(result) 是 fallback 的值; 如果出错且没有指定 fallback, 那么
  // String(result) 是 stderr 的内容
  String(result)
  // exit code (numeric)
  code
}
```

grunt 内置的第三方 Modules

grunt.file.glob -> glob

grunt.file.minimatch -> minimatch

grunt.file.findup -> findup-sync

grunt.util._ -> Lo-Dash [已废弃]

grunt.util._.str -> Underscore.string [已废弃]

grunt.util.async -> Async [已废弃]

grunt.util.hooker -> Async [已废弃]