**前端工作流，Grunt上手指南**

*Posted @2013-04-20 7:15 a.m.*

*Categories* [***Grunt***](http://www.hulufei.com/tag/Grunt) *,* [***javascript***](http://www.hulufei.com/tag/javascript)

我想先花点时间回忆一下作为一个前端需要做的工作(Loading...)

* JS合并
* JS压缩
* CSS压缩
* CSS Sprite
* 图片优化
* 测试
* 静态资源缓存(版本更新)
* ...

对应的，一个全副武装的前端可能会是这样的：

* JSHint
* CSSLint
* Jade
* CoffeeScript
* RequireJS/SeaJS
* Compass/Stylus/Less
* QUnit/Mocha/Jasmine
* ...

就是这么苦逼的设定，但其实也正是有了这些天才的工具和解决方案才让我们的工作变得更美好，WTF

随身装备这么多武器，拔刀的姿势很重要，手忙脚乱焦头烂额的狼狈样肯定是不行的。如果每个工具（任务）对应一个招数的话，Grunt就是用来组合各种华丽连续技的辅助装备。这比喻略显白烂，高端大气国际化的说法叫“工作流”(Workflow)。所以开始进入正题： **Grunt锻造指南，参见!**

* [安装](http://www.hulufei.com/post/grunt-introduction#install)
* [关于插件](http://www.hulufei.com/post/grunt-introduction#plugin)
* [配置任务](http://www.hulufei.com/post/grunt-introduction#task)
  + [模板变量](http://www.hulufei.com/post/grunt-introduction#template)
  + [任务选项](http://www.hulufei.com/post/grunt-introduction#options)
  + [指定文件](http://www.hulufei.com/post/grunt-introduction#files-config)
    - [Compact Format](http://www.hulufei.com/post/grunt-introduction#compact-format)
    - [Files Object Format](http://www.hulufei.com/post/grunt-introduction#file-object-format)
    - [Files Array Format](http://www.hulufei.com/post/grunt-introduction#file-array-format)
    - [通配符](http://www.hulufei.com/post/grunt-introduction#glob-match)
    - [动态匹配](http://www.hulufei.com/post/grunt-introduction#dynamic-filename)
  + [自定义任务](http://www.hulufei.com/post/grunt-introduction#custom-task)

**Note**：这些都是基于Grunt 0.4.x版本，需要Nodejs版本>=0.8.0

命令行下安装：

# 如果之前有装过grunt，卸载之

npm uninstall -g grunt

# 安装grunt运行工具

npm install -g grunt-cli

一个grunt项目需要两个文件：package.json和Gruntfile.js，前者用于nodejs包管理，比如grunt插件安装，后者是grunt配置文件，配置任务或者自定义任务。

先生成一个package.json文件，在grunt项目的目录下运行npm install就可以生成一个空的package.json。

安装grunt到当前目录：npm install grunt --save-dev

再生成一个Gruntfile.js的模板文件，这时候可以用[grunt-init](https://github.com/gruntjs/grunt-init-gruntfile)，或者直接手写一个：

module.exports = function(grunt) {

grunt.initConfig({

// task configuration

});

// Load the plugin

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

// Default task(s)

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

});

**关于插件**

grunt利用不同的插件完成不同的任务，比如用uglifyJS压缩js对应的插件就是grunt-contrib-uglify。

使用插件（以grunt-contrib-uglify为例）:

* 在grunt项目目录下安装对应的插件 npm install grunt-contrib-uglify --save-dev
* 在 Gruntfile.js 中加载插件 grunt.loadNpmTasks('grunt-contrib-uglify')
* 在 Gruntfile.js 中配置对应的插件任务，指定要压缩的js文件

关于配置和怎么运行任务往下细说。

在[这里](http://gruntjs.com/plugins)可以看到可用的插件，基本上大部分你能想到或没想到的任务都能找到对应的插件，需要做什么就装什么。

以后如果要重用一个grunt项目的配置，只需要有package.json和Gruntfile.js这两个文件，然后npm install即可安装所有依赖的插件。

一个插件就是对应一个任务，一般来说，所有插件都会遵循下面将要说到的任务配置规则，很多插件的文档都不会很详细，所以你只能根据插件提供的示例套用这些规则看有没有更多配置的可能性。

**关于任务**

任务分为两种："Basic" Tasks和"Multi" Tasks

Multi-tasks有所谓的target，比如下面的concat任务有foo和bar两个targets，而uglify任务有一个叫bar的target

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.

}

}

});

target的名字可以任意指定，因为target只是为了用特定配置运行指定的任务，比如grunt concat:foo或者grunt concat:bar会分别运行foo或者bar指定的concat任务。如果只运行grunt concat将会遍历所有concat下的targets按顺序运行。

但是任务的名称比如concat和uglify是固定的，由对应的插件指定，在插件的使用文档里面都会有说明。

每个multi task都必须有至少一个target.

不需要配置的任务就是Basic Task，你可以这样来定义一个Basic Task，grunt.registerTask(taskName, [description, ] taskFunction)

// foo task

grunt.register('foo', function(arg1, arg2) {

// do something

});

这样运行：grunt foo，或者grunt foo:a:b，a和b就是传递给foo的参数

**模板变量**

grunt可以通过类似<%= k.sub.k %>这种格式插入配置的其他属性值

**Options**

在一个任务配置里面，option属性可以用来覆盖默认的配置，另外，每个target都可以有自己的option属性。target的option优先级高于task的。options是可选的。

grunt.initConfig({

concat: {

options: {

// Task-level options may go here, overriding task defaults.

},

foo: {

options: {

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

},

},

bar: {

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

options.

},

},

});

不一定所有的任务都会有option的。

**指定文件**

这应该是刚接触grunt时最让人不知所措的地方了，想想这么多插件，每个插件都需要指定对应要应用到的文件，但是咋一看好像每个插件都有一套自己配置文件的方式，配置方式看上去很随意，所以总是会觉得有一丝不靠谱。

就像之前提到的，其实是有一套通用的规则的：

Grunt提供了几种不同的格式定义**src-dest**形式的文件映射。任何multi-task都支持这几种格式。

文件映射可以有3种格式：Compact Format, Files Object Format和File Array Format, 其中"Compact"和"File Array"这两种形式提供了一些额外的属性可用：

* filter 过滤，接受[fs.Stats](http://nodejs.org/docs/latest/api/fs.html" \l "fs_class_fs_stats)方法定义的名字，比如isFile, isDirectory，或者自定义函数接受一个源文件名做为参数，返回true or false。
* nonull Retain src patterns even if they fail to match files. Combined with grunt's --verbose flag, this option can help debug file path issues.
* matchBase Patterns without slashes will match just the basename part.
* ......（剩下几个看文档吧）
* 另外还有一个动态文件列表生成（批量匹配文件）

以下示例中的属性名src和dest, files都是固定的key名，一开始就不必纠结了。

**Compact Format**

这种形式只允许单个**src-dest**映射在一个target里面，只有src属性是必须的，可以没有dest，这种形式一般用在只读的task，比如jshint

grunt.initConfig({

jshint: {

foo: {

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

},

},

concat: {

bar: {

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

dest: 'dest/b.js',

},

},

});

**Files Object Format**

这种形式支持指定多个src-dest对应多个target，属性名(key)是要输出的目标文件名，value值是源文件列表。*不支持额外的属性*

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'],

},

},

},

});

**Files Array Format**

同上，只是支持额外的属性

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'},

],

},

},

});

**通配符支持**

由nodejs内置的node-glob库支持，这些都可以用在上面所说的各种文件配置中

* \*匹配任何字符，除了/
* ?匹配单个字符，除了/
* \*\*匹配任何字符，包括/，所以用在目录路径里面
* {}逗号分割的“或”操作(逗号后面不要有空格)
* ! 排除某个匹配
* // You can specify single files:
* {src: 'foo/this.js', dest: ...}
* // Or you can generalize with a glob pattern:
* {src: 'foo/th\*.js', dest: ...}
* // This single node-glob pattern:
* {src: 'foo/{a,b}\*.js', dest: ...}
* // Could also be written like this:
* {src: ['foo/a\*.js', 'foo/b\*.js'], dest: ...}
* // All files in alpha order, but with bar.js at the end.
* {src: ['foo/\*.js', '!foo/bar.js', 'foo/bar.js'], dest: ...}
* // Templates may be used in filepaths or glob patterns:

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

**动态生成文件名**

* expand 设置为true打开以下选项
* cwd 所有src指定的文件相对于这个属性指定的路径
* src 要匹配的路径，相对与cwd
* dest 生成的目标路径前缀
* ext 替换所有生成的目标文件后缀为这个属性
* flatten 删除所有生成的dest的路径部分
* rename 一个函数，接受匹配到的文件名，和匹配的目标位置，返回一个新的目标路径
* grunt.initConfig({
* minify: {
* dynamic\_mappings: {
* // Grunt will search for "\*\*/?.js" under "lib/" when the "minify" task runs
* 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.
* }
* ]
* }
* }

});

**自定义任务**

这里总结一些遇到的问题吧

**获取/设置配置（模板变量）**

* 可以读取json配置文件：config: grunt.file.readJSON('config.json')
* 获取json对象的属性：grunt.config('config.key.subkey')
* 对应的模板变量：'<%= config.key.subkey %>'
* 设置配置段：grunt.config('config', 'value')

**动态更改任务配置，循环执行某个任务**

grunt的任务都会放入一个队列顺序执行，但是队列本身是异步执行的，所以下面的这种做法是不会如预期输出：

grunt.registerTask('demo', function() {

for (var i = 0; i < 5; i++) {

grunt.task.run('t');

}

// 期望执行完5次`t`任务之后打印输出

// 实际上会立即输出，在`t`任务开始之前

console.log('run after t');

// 执行5次`t`任务之后才会执行这个`final`任务

grunt.task.run('final');

});

动态更改任务配置可以利用模板变量来做，由于如上所说的异步，所以不能直接在循环中给模板变量赋值，而是要额外做一个任务来接受配置：

// 假如有这样的一个配置

t: {

target: 'some <%= param %>'

}

// 在这个demo任务中需要多次调用t任务，每次都要设置param

grunt.registerTask('demo', function() {

for (var i = 0; i < 5; i++) {

// 要一个额外任务去更改配置

grunt.task.run('t\_wrapper:' + i);

}

});

// 更改`t`配置并运行

grunt.register('t\_wrapper', function(i) {

grunt.config('param', i);

grunt.task.run('t');

});

还有一种方法可以克隆一个新的target，然后直接更改这个cloned target的配置

**grunt.config和grunt.option的区别**

grunt.config如上所述可以用来动态更改模板变量，但是grunt.option不能这样，如果在配置中直接使用grunt.option，则option在运行时就已经确定了，不能再更改，假设这样配置：

t: {

target: 'some ' + grunt.option('param')

}

运行grunt t --param=0，则target对应就是'some 0'，不能再通过grunt.option(param, 1)这样来更改配置

grunt.option和grunt.config都可以用来在任务之间共享一些信息，但option更多用来接受额外的任务参数。

**设置输出文字颜色**

直接在字符串后面点一个颜色：grunt.log('test color'.green)

**References**

* [Grunt API](http://gruntjs.com/api)
* [gruntjs](https://github.com/gruntjs)
* [workflow-vips](https://github.com/hulufei/workflow-vips)

假如你已经熟悉了Grunt，可以去看看[Yeoman](http://yeoman.io/)，也许能为你提供更多灵感。

**UPDATE: 2013-6-29**

关于这个主题，最近在w3ctech广州站做了一次分享，浓缩了一个slides：[利用Grunt打造前端工作流](https://speakerdeck.com/hulufei/li-yong-gruntda-zao-qian-duan-gong-zuo-liu)