Introduction to Grunt

Shuhei Kagawa

I was surprised to find that **80-90% of the time spent by users waiting for pages to load is spent on the frontend**, all the work that needs to be done after the HTML document has arrived.

—Steve Souders (2007). High Performance Web Sites

フロントエンドでやりたいことは
増える一方・・・

- JS, CSS の結合・最小化
- 画像, SVG のサイズ削減
- Sass, CoffeeScript のコンパイル
- JS, CSS, HTML のチェック
- 開発用サーバーの起動
- JavaScript の自動テスト
- スタイルガイドの作成
- などなど

```
$ python -m SimpleHTTPServer
$ uglify foo.js bar.js -o out.js
$ cssmin foo.css > foo.min.css
$ imagemin foo.png > foo.min.png
$ svgmin foo.svg > foo.min.svg
$ jshint foo.js
$ jscs foo.js
$ csslint foo.css
$ compass
...
```

$ grunt

JavaScript で書かれたタスクランナー。

豊富なプラグイン（3,000 個以上！）が提供するタスクを実行するツール。

フロントエンド開発のデファクトスタンダード。
Google, Twitter, Adobe, jQuery, Mozilla などで利用。

Grunt

設定ファイル

Gruntfile.js

各種プラグイン

grunt-contrib-less

grunt-contrib-requirejs

grunt-contrib-sass

各種ツール

公式プラグイン（grunt-contrib-*）

ファイル操作	clean, copy, concat, symlink
コードチェック	jshint, csslint
開発支援	watch, connect, livereload
圧縮	uglify, cssmin, imagemin, htmlmin, requirejs, compress
AltXXX	less, coffee, compass, sass, stylus
テンプレート	jade, handlebars, jst
JS 単体テスト	qunit, nodeunit, jasmine
ドキュメント生成	yuidoc

いいところ

- タスクを自動化できる。
- タスクの明文化によりチームで開発しやすくなる。
- 一つのツールで様々なタスクを実行できる。
- プラグイン（タスク）が豊富。
- ユーザー・情報が多い。
- Windows でも利用しやすい。

使い方

用意するもの

初回だけ

Node

npm

grunt-cli

プロジェクトごと

package.json

Gruntfile.js

- JavaScript の実行環境。ブラウザの外で JavaScript を動かすことができる。
- Grunt を動かすのに必要。
- Grunt 本体や各種 Grunt プラグインも Node.js のプログラム。

- Node のモジュール管理システム。
- Node で使うプログラムの部品（モジュール）をインストールすることができる。
- Grunt 本体も各種 Grunt プラグインも npm モジュールとして管理されている。
- Grunt プラグインが利用するツール自体も npm のモジュールであることが多い。jshint や csslint など。
  - この場合は、ツール自体も Grunt プラグインと一緒にインストールできる。
  - Compass は Ruby 製なので、npm モジュールではない。

Installation on Windows

- 事前にインストール
  - Node 0.10.x
    - ユーザ環境変数の PATH に %USERPROFILE%\AppData\Roaming\npm を追加すること
  - Ruby 2.x (Compass 使う場合)
    - インストール時に PATH 追加するオプションをチェックすること

Installation on Mac

- $ brew update
- $ brew install node
  - 余裕があれば nvm がおすすめ
    - $ curl https://raw.githubusercontent.com/creationix/nvm/v0.12.2/install.sh | bash
    - $ nvm install v0.10.29
    - $ nvm alias default 0.10.29
- $ brew install ruby
  - 余裕があれば rbenv がおすすめ
    - $ brew install rbenv ruby-build
    - $ rbenv install 2.1.2
    - $ rbenv global 2.1.2
    - $ rbenv rehash

grunt コマンドのインストール

$ npm install -g grunt-cli

以下のようになれば成功。

$ grunt
grunt-cli: The grunt command line interface.
(v0.1.13)

Fatal error: Unable to find local grunt.

package.json

- npm モジュールの定義ファイル。
- プロジェクト毎に作成する。自分たちのプロジェクトも npm モジュールという考え方。
- そのプロジェクトに必要な npm パッケージを記述しておく（Grunt 本体、Grunt の各種プラグインなど）。すると、どこでも npm を使って必要なパッケージをインストールできる。

package.json の作成

まず package.json を作成。

$ npm init

モジュールをインストールしつつ、package.json の devDependencies に記録する。（自分で書くより楽。）

$ npm install -save-dev grunt
$ npm install -D grunt-contrib-connect

プロジェクトに途中から参加するなど、package.json がすでにできている場合は、書かれているモジュールをインストール。

$ npm install

Gruntfile.js

- Grunt の設定ファイル。
- JavaScript で記述する（CoffeeScript でも可）。
- プロジェクト毎に作成。そのプロジェクトで必要なタスクとその設定を記述する。
- これが書ければ、あとは実行するだけ！
- サンプル http://gruntjs.com/sample-gruntfile

```
module.exports = function(grunt) {
  // Task, Target の設定

  grunt.initConfig({
    // Config Object
  });

  // プラグインの読み込み（プラグインには Task が入っている）
  grunt.loadNpmTasks(‘plugin1’);
  grunt.loadNpmTasks(‘plugin2’);

  // 独自タスクの登録（プラグインに入っているタスクを組み合わせる）
  grunt.registerTask(‘sometask’, [‘task1:target1’, task2:target2’]);
  grunt.registerTask(‘default’, [‘sometask’]);
};
```

3 種類の Task

Alias Task

複数の Task を一つにまとめるのに使う。

grunt.registerTask(‘build’, [‘concat’, ‘uglify’, ‘cssmin’]);

$ grunt build # build Task を実行

Multi Task

複数の Target を持つ。

大半のプラグインが用意する Task はこれ。

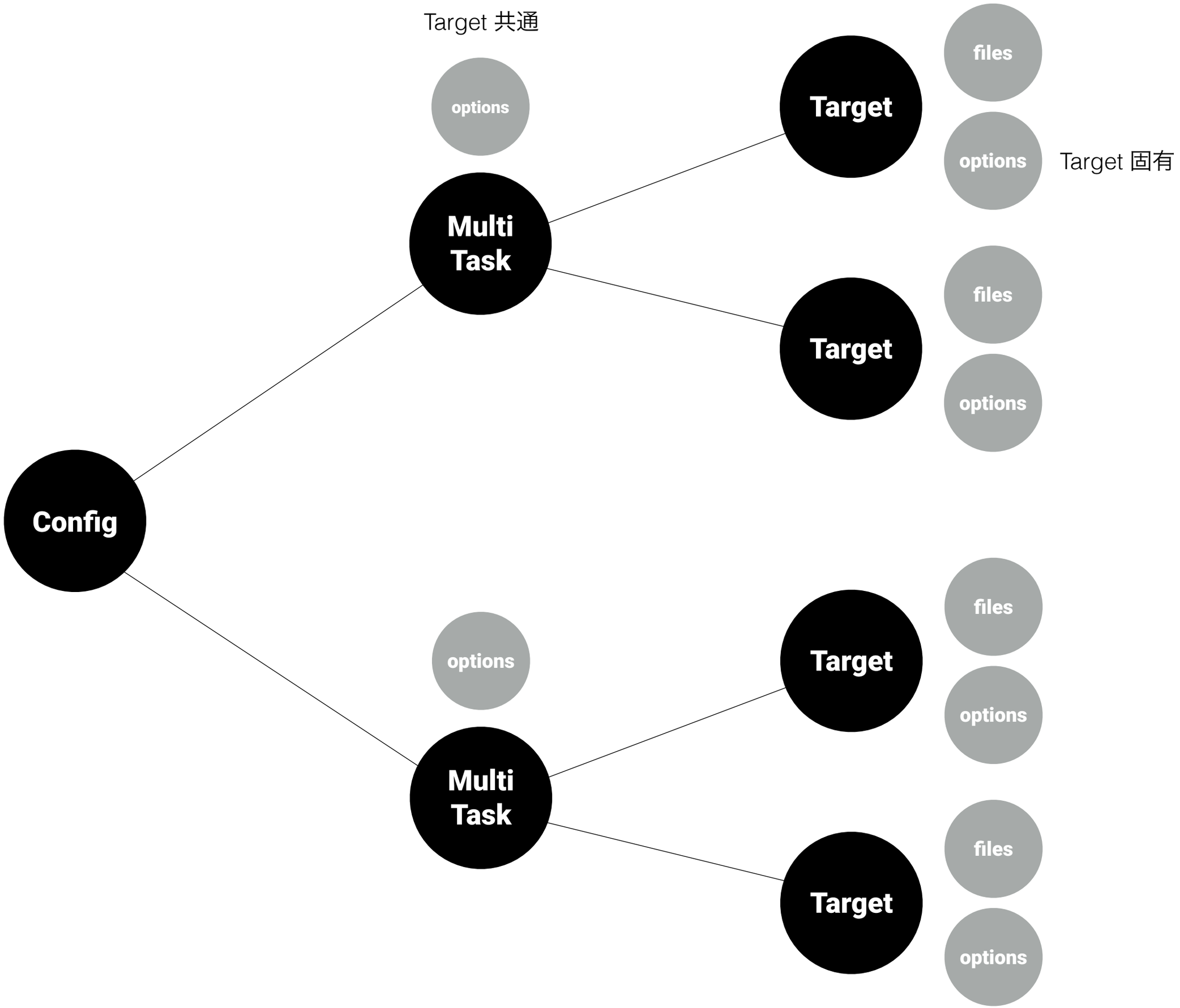
$ grunt foo:bar # foo Task の bar Target を実行

$ grunt foo # foo Task の全 Target を実行

Basic Task

Basic なのに、あまり使われない。

自分で任意のタスクを追加する際に使うと良いかも。


Target 共通
options
Multi Task
Target
files
options
Target 固有
Target
files
options
Config
options
Multi Task
Target
files
options
Target
files
options

```
grunt.initConfig({
  concat: {
    options: {
      separator: ';'
    },
    js: {
      src: ['src/**/*.js'],
      dest: ‘dist/grunt-helloworld.js’
    }
  }
});
```

Task

Target

ファイルパス

Target には処理したいファイルのファイルパスを指定。
src, dest, files, filter 等、書き方はいろいろある。
まずは src, dest の書き方がわかりやすいのでおすすめ。
ファイルを出力しない Task には dest は必要ない。
jshint など。

options

ファイルパスの他、options にその Task 固有のオプションを指定。
Target に指定すると、その Target だけ。
Task に指定すると、Target 共通で適用される。
両方で同じ項目が指定されていたら Target の方が優先。

```
grunt.initConfig({
  concat: {
    options: {
      separator: ';'
    },
    js: {
      src: ['src/**/*.js'],
      dest: ‘dist/grunt-helloworld.js’
    }
  }
});
```

Options

Files

プラグイン紹介

grunt-contrib-jshint

JavaScript をチェックして問題があれば警告を表示。
設定は .jshintrc というファイルに記述しておくと、エディタで jshint を利用する際にも使えて良い。ライブラリなどチェックしないファイルは .jshintignore に書いておく。

```
jshint: {
  all: {
    src: ['Gruntfile.js', 'src/js/**/*'],
    options: {
      jshintrc: true
    }
  }
}
```

grunt-contrib-uglify

JavaScript を結合・圧縮・難読化。

```
uglify: {
  all: {
    src: [‘src/js/foo.js’, ‘src/js/bar.js’],
    dest: ‘public/js/bundle.js’
  }
}
```

grunt-contrib-imagemin

画像を圧縮。GIF, JPEG, PNG, SVG に対応。
options はファイルの種類ごとに別なので、ファイルの種類で Target を分ける必要はない。
（src にファイルパターンだとうまく動かなかったので、files で記述しています。）

```
imagemin: {
  all: {
    files: {
      expand: true,
      cwd: ‘src/img/‘,
      src: [‘*.{jpg,svg,png,gif}’],
      dest: ‘dist/img/’,
    }
  }
}
```

grunt-contrib-compass

compass で Sass をビルド。
config.rb を指定するだけで OK。config.rb に指定するような内容を Gruntfile.js 内で指定する方法もある。

```
compass: {
  all: {
    options: {
      config: ‘config.rb’
    }
}
```

grunt-contrib-watch

ファイルを監視し、変更があれば Task を実行する。
設定が特殊で files, tasks と options。files に変更があれば、tasks を実行する。他の Task と一緒に実行する際は一番最後にすること。

```
watch: {
  js: {
    files: [‘src/js/**/*.js’],
    tasks: [‘jshint’, ‘uglify’]
  }
}
```

grunt-contrib-connect

開発用のサーバを起動する。
Grunt の実行が終わるとサーバも終了するので注意。終わらないようにするには grunt connect:server:keepalive とする。watch と一緒に使えば keepalive は不要。

```
connect: {
  server: {
    options: {
      port: 8000,
      base: 'dist'
    }
  }
}
```

grunt-spritesmith

画像から CSS Sprite 用 CSS を生成。設定が特殊で options を使わない。Compass と比べて画像の並べ方を柔軟に決められる（らしい）。

```
spritesmith: {
  all: {
    src: ‘src/img/sprites/*.png’,
    destImg: ‘dist/img/spritesheet.png’,
    destCSS: ‘dist/css/sprites.css’,
    algorithm: ‘alt-diagonal’
  }
}
```

詳細は、各プラグインのドキュメントを参照

設定にはプラグインごとに差異があるので、ドキュメントに乗っている例を参考にするのが無難です。

終

Q. Rails などのサーバサイドフレームワークでも同じことができるのでは？

A. たしかに Assets Pipeline でもある部分（CoffeeScript, Sass, Minify など）については、簡単にできる。しかし、それ以上のことをやろうとすると大変。
Grunt はプラグインさえあれば何でもできる。
また、サーバーサイドのフレームワーク関係なく使えるのが最大のメリット。

Q. gulp など類似ツールとの違いは？

A. Grunt はタスク毎に結果をファイルに書き込むので遅い。設定ファイルも長くなりがち。
しかし、先発なのでプラグインや情報が充実している。また Grunt は Node っぽくないので、Node 慣れしていない人にも使いやすいかも？

gulp は、Stream など Node の知識が前提になっているが、設定ファイルを非常にシンプルに書けるし、処理も速い。プラグインもそれなりに充実。一度見てみて、気に入った方を使うと良いかも。

Broccoli などについては、よく知らない。

```
$ grunt task:target:flag1:flag2
```

```
$ grunt connect:server:keepalive
```

task target flag