nano.sapegin.ru

Grunt 0.4: система сборки для фронтенд-разработчиков

Эта статья — пересказ моего доклада на Web Standards Days в ноябре прошлого года, учитывающий все изменения новой версии 0.4. Будет полезен как новичкам в Гранте, так и пользователям Гранта 0.3, переходящим на новую версию.

Зачем это нужно

У хороших сайтов есть две версии:

1. Версия для разработки: JS/CSS разбиты на много файлов и не сжаты, долго загружается, легко отлаживать.

2. Боевая версия: минимум запросов к серверу, ничего лишнего, всё сжато, быстро загружается.

Грант автоматически делает из первой версии вторую: склеивает файлы, минифицирует JavaScript, проверяет код с помощью JSHint, прогоняет тесты, запускает CSS-препроцессоры и компилятор CoffeeScript. Всего не перечислишь. И может делать это как по команде, так и автоматически, отслеживая изменения исходных файлов.

В отличии от других подобных инструментов (Ant, Make и т. д.), Грант создавался специально для фронтенд-разработчиков. И сам Грант, и расширения для него, и даже конфиг написаны

на знакомом им языке — JavaScript. Он легко настраивается и расширяется. А большинство готовых расширений устанавливаются одной командой вместе со всеми зависимостями. (Конечно, есть немало расширений, использующих внешние библиотеки и утилиты, которые не всегда работают на всех платформах.)

Установка

Для использования Гранта вам понадобится установить Node.js (на маке просто brew install node). Вместе с Нодой установится менеджер пакетов npm, который понадобится для установки самого Гранта и его плагинов.

Если вы уже пользуетесь предыдущей версией Гранта, то перед установкой её нужно удалить: npm uninstall -g grunt.

Установим консольную утилиту grunt (ключ `-g` означает, что пакет будет установлен глобально), которая будет запускать Грант, установленный в папке вашего проекта. Таким образом у каждого проекта будут свои версии Гранта и плагинов — можно не бояться, что при обновлении сборка поломается.

```
$ npm install grunt-cli -g
```

Вероятно, вам понадобится запустить npm через sudo или открыть консоль под администратором.

Настройка

Теперь нужно создать в папке проекта два файла:

- package.json — описание проекта для npm. Содержит список зависимостей (в нашем случае это Грант и его плагины) и позволяет потом устанавливать их все одной командой.

- Gruntfile.js или Gruntfile.coffee — файл конфигурации Гранта (грантфайл). (До версии 0.4 этот файл назывался grunt.js.)

(Примеры к статье есть в репозитории на Гитхабе.)

package.json

package.json можно создать вручную или командой `npm init`. В нём есть два обязательных поля — имя проекта и версия. Если вы делаете сайт, а не библиотеку, то их содержимое не имеет значения:

```
{
    "name": "MyProject",
    "version": "0.0.0"
}
```

Теперь нужно установить (и добавить в package.json) зависимости нашего проекта. Грант:

```
$ npm install grunt --save-dev
```

И все необходимые плагины:

```
$ npm install grunt-contrib-concat grunt-contrib-
uglify --save-dev
```

Ключ `--save-dev` в дополнение к установке добавляет ссылку на пакет в package.json. Установить все зависимости, уже перечисленные в файле, можно командой npm install.

Грантфайл

Грантфайл выглядит примерно так:

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

    grunt.initConfig({

        concat: {
            main: {
                src: [
                    'js/libs/jquery.js',
                    'js/mylibs/**/*.js'
                ],
                dest: 'build/scripts.js'
            }
        },

        uglify: {
            main: {
                files: {

                    'build/scripts.min.js': '<%=
concat.main.dest %>'
                }
            }
```

```
        }
    });


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


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

Этот грантфайл склеивает JS-файлы (jQuery и все JS-файлы из папки js/mylibs, задача concat) и минифицирует их с помощью UglifyJS (задача uglify).

Обратите внимание на задачу по умолчанию default — это просто «ссылка» на задачи concat и uglify. Она обязательно должна быть в грантфайле.

Задачи, подзадачи, параметры

Конфигурация большинства задач выглядит примерно так:

```
concat: {
    options: {
        separator: ';'
    },
    libs: {
        src: 'js/libs/**/*.js',
        dest: 'build/libs.js'
    },
```

```
    main: {
        src: [
            'js/mylibs/*.js',
            'js/main.js'
        ],
        dest: 'build/scripts.js'
    }
}
```

concat:libs и concat:main — это подзадачи, они позволяют запускать одну задачу для разных исходных файлов. А в options определяются общие для всех подзадач параметры.

В других системах сборки задачи обычно называют целями (target).

Списки файлов

Список исходных файлов можно задать двумя способами: один файл или массив файлов. Можно использовать маски (glob). С масками нужно иметь ввиду, что порядок файлов может оказаться любым. Это может привести к проблемам, например, при склейке CSS- или JS-файлов.

```
'js/main.js'
[ 'js/utils.js', 'js/main.js' ]
[ 'js/libs/*.js', 'js/mylibs/**/*.js' ]
```

Шаблоны

Внутри параметров конфига можно использовать шаблоны. Грант использует шаблонизатор из библиотеки Lo-Dash.

С помощью шаблонов можно ссылаться на другие параметры конфига, вставлять текущую дату в имя результирующего файла, и использовать любые конструкции Яваскрипта.

```
concat: {
    main: {
        src: 'js/*.js',
        dest: 'build/scripts.js'
    }
},
uglify: {
    main: {
        files: {

            'build.<%= grunt.template.today("m-
d-yyyy") %>.js': '<%= concat.main.dest %>'
        }
    }
}
```

А вот так можно использовать данные из JSON-файла:

```
pkg: grunt.file.readJSON('package.json'),
banner: '/* <%= pkg.name %> v<%= pkg.version %>
*/'
uglify: {
    main: {
        files: {
            '<%= pkg.name %>.min.js': '<%=
pkg.name %>.js'
        }
    }
```

```
}
```

Запуск

```
$ grunt
$ grunt concat
$ grunt concat:main

$ grunt --debug
```

Во время разработки удобно запускать Грант с ключом `--debug`. Задачи могут использовать его по-разному. Например, grunt-contrib-stylus в отладочном режиме не сжимает CSS-код.

grunt watch

Задача watch запускает задачи при каждом изменении исходных файлов.

Например, можно заново склеивать JS-файлы при каждом их изменении:

```
concat: {
    main: {
        src: 'js/*.js',
        dest: 'build/scripts.js'
    }
}
watch: {
    concat: {
        files: '<%= concat.main.src %>',
```

```
        tasks: 'concat'
    }
}
```

Не забудьте добавить в грантфайл плагин grunt-contrib-watch:

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

И установить соответствующий пакет из npm:

```
$ npm install grunt-contrib-watch --save-dev
```

Веб-сервер

Простейший веб-сервер для статических сайтов — задача connect.

```
connect: {
    test: {
        options: {
            port: 8000,
            base: '.'
        }
    }
}
```

Запускается так:

```
$ grunt connect
```

Теперь ваш сайт доступен по адресу http://localhost:8000/.

JSHint

Раньше (до версии 0.4) нужно было перечислять все опции JSHint прямо в грантфайле, сейчас можно хранить их в файле .jshintrc. Этот же файл могут использовать и консольный JSHint, и SublimeLinter. Задача jshint.

```
jshint: {
    options: {
        jshintrc: '.jshintrc'
    },
    files: 'js/**/*.js'
}
```

Конфигурации

Конфигурации позволяют переключаться между боевой и отладочной версиями сайта. В явном виде в Гранте их нет. Но можно сделать, например, так:

```
concat: {
    main: {
        src: 'js/*.js',
        dest: 'build/scripts.js'
    }
},
uglify: {
    main: {
        files: {
            '<%= concat.main.dest %>': '<%=
concat.main.dest %>'
        }
```

```
    }
}
...
grunt.registerTask('default', ['concat',
'uglify']);
grunt.registerTask('debug', ['concat']);
```

Отладочная версия собирается так: `grunt debug`, а боевая: просто `grunt`. HTML в обоих случаях не меняется, но в последнем код будет сжат.

Для более крупных проектов может понадобится что-то сложнее. Например, RequireJS и/или подключение разных файлов в шаблонах.

Быстрое подключение плагинов

Писать для каждого плагина `grunt.loadNpmTasks('имяплагина')` быстро надоест, поэтому лучше сразу заменить все вызовы loadNpmTasks одной строчкой:

```
require('load-grunt-tasks')(grunt);
```

И установить load-grunt-tasks:

```
$ npm install load-grunt-tasks --save-dev
```

Это заклинание вызовет loadNpmTasks для всех плагинов установленных с ключом `--save-dev`.

grunt-init

Утилита grunt-init упрощает инициализацию проектов (в английском для обозначения этого процесса есть удобное слово scaffolding):

- создаёт файлы и структуру папок;
- позволяет использовать шаблоны везде, где только можно;
- переименовывает файлы при копировании;
- задаёт пользователю уточняющие вопросы.

Устанавливается отдельно:

```
$ npm install grunt-init -g
```

Из коробки есть шаблоны для грантфайлов, jQuery-плагинов, проектов на Node.js и другие (полный список можно посмотреть набрав `grunt-init --help`). Например, если выполнить `grunt-init node`, то получится вот такое дерево файлов:

```
$ tree
.
├── Gruntfile.js
├── LICENSE-MIT
├── README.md
├── lib
│   └── MyCoolProject.js
├── package.json
└── test
    └── MyCoolProject_test.js
```

Это очень мощный инструмент, которому можно найти немало применений. Я уже писал о нём более подробно.

Собственные задачи

Свои задачи делать довольно просто. Для примера сделаем задачу, которая будет запускать из Гранта консольный оптимизатор веб-графики imgo.

Стоит рассматривать эту задачу только как пример. Для реальной работы лучше использовать grunt-contrib-imagemin.

Конфиг

Задача будет принимать список изображений и запускать imgo для каждого файла. Вот так будет выглядеть конфиг. Всего один параметр:

```
imgo: {
    images: {
        src: 'images/**'
    }
}
```

Код задачи

Добавить задачу можно двумя функциями:

- grunt.registerMultiTask — задача с подзадачами, как concat, uglify и как описано в разделе «Задачи, подзадачи, параметры» выше. Нам нужна именно такая.

- grunt.registerTask — используется для задач-ссылок (как

default и debug выше) или задач, где несколько наборов входных данных не имеют смысла.

```
grunt.registerMultiTask('imgo', 'Optimize images
using imgo', function() {

    var done = this.async();




    grunt.util.async.forEach(this.filesSrc,
function(file, next) {

        grunt.util.spawn({
            cmd: 'imgo',
            args: [file]
        }, next);
    }, done);
});
```

Задача должна быть асинхронной, потому что мы будем вызвать внешнюю программу, а в Node.js это асинхронная операция. `this.async()` возвращает функцию, которую необходимо вызвать, когда, все файлы будут обработаны.

Сам цикл по исходным файлам тоже асинхронный. Для этого используется метод forEach из модуля async.

Хранение и использование

Задачи можно класть прямо в грантфайл, а можно в отдельные файлы или публиковать в npm (если ваша задача может быть полезна и другим людям).

Первый способ самый простой. Для этого надо разместить код задачи где-нибудь перед `grunt.registerTask('default', […])`.

Во втором случае нужно создать для задач отдельную папку и поместить код задачи в такую же обёртку, как и у грантфайла:

```
module.exports = function(grunt) {
    grunt.registerMultiTask('imgo', 'Optimize
images using imgo', function() {

    });
};
```

А в грантфайле написать:

```
grunt.loadTasks('tasks');
```

Если будете делать свои задачи, обязательно посмотрите документацию API — в Гранте уже есть множество полезных функций.

Ссылки

- Код примеров к статье
- Плагины из коллекции contrib (поддерживаются разработчиками Гранта):

    - concat — склеивание файлов;
    - uglify — минификация JS (UglifyJS);
    - jshint — проверка JS (JSHint);
    - watch — отслеживание изменений в файлах;
    - connect — простой веб-сервер для статики;
    - imagemin — оптимизация картинок;
    - CSS-препроцессоры: sass, less, stylus;
    - тестовые фреймворки: qunit, jasmine;
- Ещё полезные плагины:
    - exec — запуск исполняемых файлов;
    - csso — оптимизация CSS;
    - remove-logging — удаляет из JS вызовы console.log();
    - string-replace — замена строк в файлах;
- Мои плагины:
    - webfont — сборка веб-шрифта из набора SVG-файлов;
    - bower-concat — автоматическая склейка Bower-компонентов;
    - fingerprint — версии (для сброса кэша) статических

файлов;

- shower-markdown — генератор презентаций Shower из Markdown;

- Сайт Grunt, документация, плагины
- Getting Started With Grunt
- Upgrading from 0.3 to 0.4
- Грантфайл jQuery (весьма продвинутый)
- Подборка шаблонов grunt-init
- Создание файлов и структуры проекта по шаблонам с помощью Grunt