Gulp

LoftSchool
от мыслителя к создателю

Содержание

1

Что такое Gulp

Gulp — это инструмент сборки веб-приложения, позволяющий автоматизировать рутинные, повторяющиеся задачи, такие как сборка и минификация CSS- и JS-файлов, запуск тестов, перезагрузка браузера и т.д. При помощи **Gulp** можно написать любой другой конфиг (конфигурационный файл, в котором прописаны настройки), поэтому справедливо будет сказать, что **Gulp** - это система для описания произвольного вида задач.

Задачи, которые помогает решить **Gulp:**

- Создание веб-сервера и автоматическая перезагрузка страницы в браузере при сохранении кода, слежение за изменениями в файлах проекта.
- Использование различных JavaScript-, CSS- и HTML-препроцессоров (CoffeeScript, Less, Sass, Stylus, Jade и т.д.).
- Минификация CSS- и JS-кода, а также оптимизация и конкатенация отдельных файлов проекта в один.
- Автоматическое создание вендорных префиксов (приставок к названию CSS-свойства, которые добавляют производители браузеров для нестандартных свойств) для CSS.
- Управление файлами и папками в рамках проекта: создание, удаление, переименование.
- Работа с изображениями: оптимизация, создание спрайтов.
- Создание различных карт проекта и автоматизация другого ручного труда.
- Деплой — отправка на внешний сервер.

2

Установка Gulp (gulpjs.com)

Для работы **Gulp** в первую очередь необходимо установить **node.js**. Затем давайте перейдем в заранее созданную директорию с проектом и создадим файл **packaje.json** при помощи команды **`npm init`**, которую нужно выполнить в терминале, находясь в директории с проектом.

Для работы необходим:

- nodejs и npm (**npm** устанавливается вместе с **nodejs**).

- Официальный сайт

- Поиск плагинов

- Курс по gulp

Алгоритм работы:

1) Проверка, установлен ли **Gulp**:

```
$ gulp –v
```

2) Устанавливаем глобально:

```
$ npm install ––global gulp
```

сокращённо - **`$ npm i –g gulp`**

3) Создаем в корне проекта файл **package.json**

```
$ npm init
```

Либо создать файл самостоятельно.

- **name** — название проекта

- **version** — версия проекта

- **description** — описание

- **...**

остальные поля ссылка

4) Устанавливаем в проект в **devDependencies**

```
$ npm install --save-dev gulp
```

сокращённо - **`$ npm i -D gulp`**

5) Создаем в корне проекта файл **gulpfile.js**

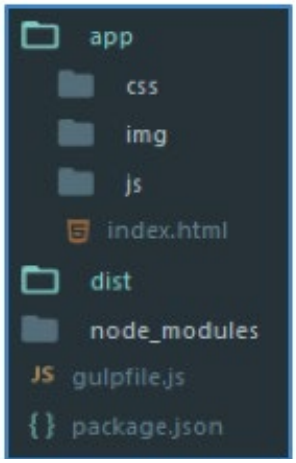
```
var gulp = require('gulp');

gulp.task('default', function()

{ // задача по умолчанию

});
```

6) Запускаем в консоли




```
$ gulp
```

Структура:

app	рабочая директория проекта (исходники)
dist	директория сборки (генерируется сборщиком)
node_modules	директория модулей, установленных через npm
package.json	файл с глобальными настройками проекта
gulpfile.js	файл для управления проектом

Обратите внимание, что мы работаем в одной папке app — здесь мы создаём код и ведём разработку, а в папке **dist** будет код, который мы потом отдаём дальше или выкладываем на хостинг, эта папка генерируется **gulp.**

3

Задачи в gulp

Если сейчас мы запустим команду gulp в терминале, то увидим ошибку, так как мы ещё не создали файл **gulpfile.js**. Это конфигурационный файл, в котором собраны описания различных задач проекта.

Создадим в корневой директории проекта **gulpfile.js** и опишем первую задачу.

```
var gulp = require('gulp');
gulp.task('default', function(callback) {
console.log('Hello world');
callback();
});
```

Если теперь выполнить команду **gulp**, мы увидим в консоли сообщение “Hello world”.

При запуске команды **gulp** без параметров подразумевается, что выполнится задача со специализированным зарезервированным словом **default**.

Если запустить команду **gulp** и передать параметр, то этот параметр — имя задачи, которая описана в **gulpfile.js**.

Метод **gulp.task** принимает два параметра. Первый — имя задачи, второй - функция, которая выполняет задачу. Зачем в описанной таске используется **callback**? Для того, чтобы сигнализировать о завершении задачи. Есть четыре основных способа:
1. Вызов **callback**-функции, как в примере выше.
2. Возвратить промис.
3. Возвратить поток.
4. Возвратить дочерний процесс.

Как правило, волноваться об этом не стоит, так как в большинстве случаев вы всегда будете возвращать поток.

Для выборки файлов используется метод **gulp.src** Первым параметром он принимает строку или массив строк, которые ни что иное, как пути до обрабатываемых в таске файлов. Вторым — объект настроек.

Описывая пути, можно использовать специальные паттерны, за обработку которых отвечает модуль **minimatch**. Например:

```
gulp.src('app/**/*.*');
gulp.src('app/**/*.js');
gulp.src('app/img/*.{png,jpg,gif}');
gulp.src(['app/style/*.sass', '!app/style/_*.sass']);
```

Еще один метод, который нам сейчас понадобится — это **gulp.dest**, он будет принимать поток файлов и записывать их в указанную директорию. Подробнее о нём можно прочитать в документации.

Напишем простую задачу “**copy**”, которая будет копировать файлы **js** из одной директории в другую:

```
var gulp = require('gulp');
gulp.task('copy', function(){
return gulp.src(app/js/*.js')
.pipe(gulp.dest(dist/js'));
});
```

И выполним ее в терминале:

```
gulp copy
```

Вот что произойдет:

- **gulp.src** найдёт все файлы в директории '**app/js**' с расширением **.js** создаст из них специальные объекты vinyl и передаст дальше по потоку.
- метод **pipe** примет поток и передаст его в **gulp.dest**.
- **gulp.dest** запишет все файлы из потока в директорию **dist/js.**

Зачем нужен **return**? Как раз для того, чтобы сигнализировать о завершении задачи. Как мы говорили выше, один из способов сказать о том, что задача закончена — это вернуть поток. Если не поставить **return**, задача никогда не завершится.

Теперь мы можем описать задачу, которая будет не только копировать файлы, но и как-то их преобразовывать. Например, объединять все js-файлы в один, обфусцировать, минифицировать и переименовывать итоговый файл.

```
var gulp = require('gulp');
var uglify = require('gulp-uglify');
var concat = require('gulp-concat');

gulp.task('js', function(){
   return gulp.src(app/js/*.js') // Берем все файлы *.js
   из директории
   .pipe(concat('app.js')) // объединим их в один файл
    app.js
   .pipe(uglify()) // минифицируем
   .pipe(rename({ // добавим суффикс, т.е преименуем в
    app.min.js
    suffix: '.min'
   }))
   .pipe(gulp.dest('dist/js')); // положим в нашу папку
    для хостинга
});
```

Мы хотим запускать задачу каждый раз, когда в отслеживаемом файле или директории что-то изменилось. Для этого используется метод **gulp.watch.**

Опишем задачу, которая будет запускать слежение:

```
gulp.task('watch', function(){
   gulp.watch('app/js/**.*.js', gulp.series('js'));
});
```

После запуска этой задачи **Gulp** начнет следить за файлами, переданными первым параметром. Если в них произойдет изменение, то **Gulp** автоматически запустит соответствующий таск.

4

Работа с browser-sync

Browser-sync - это инструмент, который позволяет производить автоматическое отображение приложения (проекта) в браузере и перезагрузку при внесении изменений в код. Причем эти изменения могут вносится и отслеживаться синхронно в нескольких браузерах или устройствах.

Пример использования :

```
var gulp = require('gulp');
var browserSync = require('browser-sync').create();
var sass = require('gulp-sass');

//Компиляция SASS
gulp.task('sass', function() {
  gulp.src('app/scss/*.scss')
    .pipe(sass())
    .on('error', log)
    .pipe(gulp.dest('app/css'))
    .pipe(browserSync.stream());
});

// Запускаем локальный сервер (только после компиляции
sass)
gulp.task('server', ['sass'], function() {
  browserSync.init({
    notify: false,
    port: 9000,
    server: {
      baseDir: 'app'
    }
  });
});
```

```
// слежка и запуск задач
gulp.task('watch', function() {
  gulp.watch('app/scss/*.scss', ['sass']);
  gulp.watch([
    'app/**/*.html',
    'app/js/**/*.js'
  ]).on('change', browserSync.reload);
  gulp.watch([
    'app/css/**/*.css'
  ]).on('change', browserSync.reload);
});

// Задача по-умолчанию
gulp.task('default', ['server', 'watch']);
```

5

GULP API

Gulp.src()

Возвращает поток, представляющий файлы, который уже может быть передан на вход плагинам (ссылка):

```
gulp.src(globs[, options])
```

- globs — маска файлов (строка или массив).

- options — объект настроек.

Пример использования при «склеивании» файлов:

```
var gulp = require('gulp'),
      concatCss = require('gulp-concat-css');

gulp.task('default', function () {
    return gulp.src('app/**/*.css')
        .pipe( concatCss('fullstyle.css') )
        .pipe( gulp.dest('app/css/') );
});
```

Пример масок файлов:

css/style.css	файл **style.css** в директории css
css/*.css	все файлы с расширением **.css** в директории css
js//*.js**	все файлы с расширением **.js** в директории js и всех дочерних директориях
!js/app.js	исключает определённый файл
***.+(js|css)**	все файлы в корневой директории с расширениями **.js** или **.css**

Пример маски для всех файлов .css в директории css, кроме уже минифицированных:

```
gulp.src(['css/**/*.css', '!css/**/*.min.css']);
```

Gulp.dest()

Записывает в файл из потока; если папок в пути не существует, они будут созданы (ссылка):

```
gulp.dest(path[, options])
```

- path — путь.

- options — объект настроек.

Пример использования при «склеивании» файлов:

```
var gulp = require('gulp'),
      concatCss = require('gulp-concat-css');

gulp.task('default', function () {
    return gulp.src('app/**/*.css')
        .pipe( concatCss('fullstyle.css') )
        .pipe( gulp.dest('app/css/') );
});
```

Gulp.task()

Объявляет задачу (ссылка):

```
gulp.task(name[, deps], fn)
```

- **name** — название задачи;
- **deps** — массив задач, которые должны быть выполнены и завершены до запуска этой задачи;
- **fn** — функция, выполняемая при вызове задачи.

Использование:

```
var gulp = require('gulp');

gulp.task('default', function() {
      console.log('default task');
});
```

Использование с **deps**:

```
var gulp = require('gulp');

gulp.task('default', ['your_task_1', 'your_task_2'],
function() {
      console.log('default task');
});
```

По умолчанию задачи выполняются асинхронно - т. е. запускаются одновременно сразу все задачи без ожидания. Именно поэтому deps-задачи могут не соблюдать очерёдность

Чтобы добиться очерёдности выполнения, следует:

- Сообщить, что задача закончена.

- Указать, что задача зависит от другой задачи.

Чтобы выполнить синхронно задачу:

1) Передать **callback**:

```
gulp.task('callback_task', function(callback_func) {
      // любая асинхронная задача
      setTimeout(function () {
           callback_func();
      }, 500);
});
```

2) Вернуть поток:

```
gulp.task('stream_task', function() {
      return gulp.src('app/**/*.css')
            .pipe( concatCss('fullstyle.css') )
            .pipe( gulp.dest('app/css/') );
});
```

3) Вернуть промис:

```
gulp.task('promise_task', function() {
      var deferred = Q.defer();
      // любая асинхронная задача
      setTimeout(function () {
           deferred.resolve();
      }, 500);

      return deferred.promise;
});
```

Выполняем задачи последовательно:

```
var gulp = require('gulp');

// Первая задача
gulp.task('first_task', function() { });

// Вторая задача
gulp.task('second_task', ['first_task'], function() {
      // Не запустится, пока не выполнится первая
задача });

gulp.task('default', ['first_task', 'second_task'],
function() { });
```

Gulp позволяет определить таск по умолчанию (тот самый, с названием “**default**”), который будет запускаться по команде **gulp**:

```
$ gulp
```

Для вызова конкретного таска — указываем его имя:

```
$ gulp task_name
```

Gulp.watch()

Следит за файлами и выполняет действия в случае их изменения (ссылка):

```
gulp.watch(glob[, opts], tasks) (вариант 1)
```

- **glob** - файлы, за которыми необходимо наблюдать (строка или массив);

- **opts** — объект настроек;

- **tasks** — массив названий задач, запускаемых при изменении файла.

Использование:

```
var watcher = gulp.watch('js/**/*.js', ['task_1',
'task_2']);

watcher.on('change', function(event) {
       console.log('File change – running tasks...');
});

gulp.watch(glob[, opts, cb]) (вариант 2)
```

- **glob** — файлы, за которыми наблюдать (строка или массив)

- **opts** — объект настроек;

- **cb** — callback-функция, вызываемая при изменении.

Использование:

```
 var watcher = gulp.watch('js/**/*.js', function(event) {
       console.log('File change – running tasks...');
});
```

- **event.type** — тип изменения **added, changed** или **deleted;**

- **event.path** — путь к файлу, который вызвал событие.

6

MODULES

Gulp-concat-css

Модуль для склеивания css-файлов (ссылка).

Установка:

```
$ npm install --save-dev gulp-concat-css
```

сокращённо - **$ npm i -D gulp-concat-css**

Использование:

```
var gulp = require('gulp'),
      concatCss = require('gulp-concat-css');

gulp.task('default', function () {
    return gulp.src('app/**/*.css')
        .pipe( concatCss('fullstyle.css') )
        .pipe( gulp.dest('app/css/') );
});
```

concatCss(targetFile, options):

- targetFile — относительный путь генерируемого файла;

- options — объект настроек.

Gulp-minify-css

Модуль для минификации css-файлов (ссылка).

Установка:

```
$ npm install --save-dev gulp-minify-css
```

сокращённо - **$ npm i -D gulp-minify-css**

Использование:

```
var gulp = require('gulp'),
      concatCss = require('gulp-concat-css'),
      minifyCss = require('gulp-minifycss');

gulp.task('default', function () {
    return gulp.src('app/**/*.css')
        .pipe( concatCss('fullstyle.css') )
        .pipe( minifyCss() )
        .pipe( gulp.dest('app/css/') );
});
```

minifyCss (options):

- path — объект настроек;

Gulp-rename

Модуль для переименования файлов (ссылка).

Установка:

```
$ npm install --save-dev gulp-rename
```

сокращённо - **$ npm i -D gulp-rename**

uglify (options):

- options— строка пути, callback-функция, **hash**

Строка:

```
var gulp = require('gulp'),
      rename = require('gulp-rename');

gulp.task('default', function () {
      return gulp.src('app/**/*.js')
          .pipe( rename('new.js') )
          .pipe( gulp.dest('dist/js/') );
});
```

Callback-функция:

```
gulp.task('default', function () {
      return gulp.src('app/**/*.js')
          .pipe( rename( function (path) {
                    path.dirname += '/js';
                    path.basename += 'new';
```

```
                    path.extname = '.js'
            }))
          .pipe( gulp.dest('dist') );
});
```

Hash:

```
gulp.task('default', function () {
      return gulp.src('app/**/*.js')
          .pipe( rename( {
                      dirname : '/js',
                      basename : 'new',
                      prefix : 'pre-',
                      suffix : '.full',
                      extname : '.js'
           } )
          .pipe( gulp.dest('dist') );
});
```

Gulp-uglify

Модуль для минификации js-файлов (ссылка).

Установка:

```
$ npm install --save-dev gulp-uglify
```

сокращённо - **`$ npm i -D gulp-uglify`**

Использование:

```
var gulp = require('gulp'),
     rename = require('gulp-rename'),
     uglify = require('gulp-uglify');

gulp.task('default', function () {
    return gulp.src('app/**/*.js')
        .pipe( uglify() )
        .pipe( rename('main.min.js') )
        .pipe( gulp.dest('dist/js') );
});
```

uglify (options):

- options — объект настроек.

Gulp-autoprefixer

Модуль для управления браузерными префиксами (ссылка).

Его работа основана на сайте

Установка:

```
$ npm install --save-dev gulp-autoprefixer
```

сокращённо - **$ npm i -D gulp-autoprefixer**

Использование:

```
var gulp = require('gulp'),
      autoprefixer = require('gulp-autoprefixer');

gulp.task('default', function () {
    return gulp.src('app/**/*.css')
        .pipe( autoprefixer({
            browsers: ['last 2 versions'],
            cascade: false
        }))
        .pipe( gulp.dest('dist/js') );
});
```

autoprefixer (options):

- options — объект настроек.

Gulp-sass

Модуль для компиляции sass-файлов (ссылка).

Установка:

```
$ npm install --save-dev gulp-sass
```

сокращённо - **$ npm i -D gulp-sass**

Использование:

```
var gulp = require('gulp'),
       sass = require('gulp-sass');
gulp.task('sass_task', function () {
      gulp.src('app/**/*.sass')
            .pipe( sass({outputStyle: 'compressed'}) )
            .pipe(gulp.dest('dist/css'));
});
```

sass (options):

- options — объект настроек.

Gulp-uncss

Модуль для оптимизации css-файлов на основе использования в html (ссылка).

Установка:

```
$ npm install --save-dev gulp-uncss
```

сокращённо - **$ npm i -D gulp-uncss**

Использование:

```
var gulp = require('gulp'),
       uncss = require('gulp-uncss');

gulp.task('sass_task', function () {
      gulp.src('app/**/*.css')
```

```
            .pipe( uncss({ html: [ 'app/index.html' ] })
)
            .pipe(gulp.dest('dist/css'));
});
```

uncss (options):

- options — объект настроек.

Gulp-imagemin

Модуль для сжатия изображений (ссылка).

Установка:

```
$ npm install --save-dev gulp-imagemin
```

сокращённо - **$ npm i -D gulp-imagemin**

Использование:

```
var gulp = require('gulp'),
       imagemin = require('gulp-imagemin');

gulp.task('imagemin_task', function () {
      gulp.src('app/img/**/*')
            .pipe( imagemin( {
                  progressive: true,
                  interlaced: true
            } )
            .pipe(gulp.dest('dist/css'));
      ) );
});
```

clean (options):

- options — объект настроек.

Del

Модуль для удаления папок и файлов (ссылка).

Установка:

```
$ npm install --save-dev del
```

сокращённо - **$ npm i -D del**

Использование:

```
var del = require('del');

del(['tmp/*.js', '!tmp/unicorn.js'], function (err,
paths) {
    console.log('Deleted files/folders:\n', paths.
join('\n'));
});
```

del (patterns, [options], callback):

- patterns — строка или массив паттернов;

- options — объект настроек;

- callback — callback-функция.

Gulp-util

Модуль утилит (ссылка).

Установка:

```
$ npm install --save-dev gulp-util
```

сокращённо - **$ npm i -D gulp-util**

Использование:

```
var gutil = require('gulp-util');

gutil.log('stuff happened', 'Really it did', gutil.
colors.magenta('123'));
```

Vinyl-ftp

Модуль для работы по ftp (ссылка).

Установка:

```
$ npm install --save-dev vinyl-ftp
```

сокращённо - **$ npm i -D vinyl-ftp**

ВНИМАНИЕ!!! Ни в коем случае не заливайте данные FTP-доступа на github, это приведет к взлому вашего сайта!!!

Использование:

```
var gulp = require('gulp'),
      gutil = require('gulp-util'),
      ftp = require('vinyl-ftp');

gulp.task('deploy', function () {
      var conn = ftp.create({
                host: 'your_host',
                user: 'your_user',
                password: 'your_password',
                parallel: 10,
              log: gutil.log
       });

       var globs = [
               'dist/**/*'
       ];

       return gulp.src(globs, { base: 'dist/', buffer:
false})
           .pipe(conn.dest('public_html/'));

});
```

clean (options):

- **options** — объект настроек.

Browser-sync

Модуль для синхронизации с браузером (ссылка).

Установка:

```
$ npm install --save-dev browser-sync
```

сокращённо - **$ npm i -D browser-sync**

Использование:

```
var gulp = require('gulp'),
     browserSync = require('browser-sync');

gulp.task('browser_sync', function () {
    browserSync({
        port: 9000,
        open: true,
        notify: false,
        server: './app'
    });
});
```

browserSync (options):

- options — объект настроек

Gulp-useref

Модуль для конкатенации стилей и скриптов через парсинг спец. блоков (ссылка).

Установка:

```
$ npm install --save-dev gulp-useref
```

сокращённо - **$ npm i -D gulp-useref**

Спец. блок

```
<html>
<head>
    <!-- build:css css/combined.css -->
    <link href="css/one.css" rel="stylesheet">
    <link href="css/two.css" rel="stylesheet">
    <!-- endbuild -->
</head>
<body>
    <!-- build:js scripts/combined.js -->
    <script type="text/javascript" src="scripts/one.
js"></script>
    <script type="text/javascript" src="scripts/two.
js"></script>
    <!-- endbuild -->
</body>
</html>
```

Использование:

```
var gulp = require('gulp'),
       useref = require('gulp-useref');

gulp.task('useref_task', function () {
      var assets = useref.assets();

      gulp.src('app/*.html')
            .pipe( assets )
            .pipe( assets.restore() )
            .pipe( useref() )
            .pipe(gulp.dest('dist'));
});
```

useref (options):

- options — объект настроек.

Результат:

```
<html>
<head>

    <link href="css/combined.css" rel="stylesheet">

</head>
<body>

    <script type="text/javascript" src="scripts/combined.
js"></script>

</body>
</html>
```

Wiredep

Модуль для автоподключения bower компонентов через спец. блоки (ссылка).

Установка:

```
$ npm install --save-dev wiredep
```

сокращённо - **$ npm i -D wiredep**

Спец. блок

```
<html>
<head>
    <!-- bower:css -->
    <!-- endbower -->
</head>
<body>
    <!-- bower:js -->
    <!-- endbower -->
</body>
</html>
```

Использование:

```
var gulp = require('gulp'),
       wiredep = require('wiredep').stream;

gulp.task('wiredep_task', function () {
      gulp.src('app/templates/*.html')
             .pipe( wiredep ({
                   ignorePath: /^(\.\.\/)*\.\./
```

```
            }) )
            .pipe(gulp.dest('app/temp/templates'));
});
```

wiredep (options):

- options — объект настроек.

Так же можно работать и с jade-файлами:

```
gulp.task('wiredep_jade', function () {
      gulp.src('app/templates/*.jade')
            .pipe( wiredep ({
                  ignorePath: /^(\.\.\/)*\.\./
            }) )
            .pipe(gulp.dest('app/temp/templates'));
});
```

Пропишем в самом jade-файле:

```
doctype html
html(lang='ru-RU')
    head
       // bower:css
       // endbower

       script(src='bower/modernizr/modernizr.js')

    body
…

       // bower:js
       // endbower
```

modernizr.js лучше подключить вручную, т. к. он должен быть в **head.**

Wiredep автоматически определит, какие файлы в какой последовательности необходимо подключать. Чтобы указать другие файлы (например, чтобы подключить **.min** файл), необходимо в нашем **bower.json**-файле указать следующее:

```
{
   “overrides” : {
        “qtip2” : {
             “main” : [
                  “./jquery.qtip.min.js” ,
                  “./jquery.qtip.min.css”
             ],
             “dependencies” : {
                  “jquery” : “>=1.6.0”
            }
        }
  }
}
```

main — какие файлы подключать;

dependencies — зависимости.

7

Gulp Blacklist

Нерекомендуемые модули (ссылка на полный список):

gulp-clean	use the `del` module
gulp-browserify	use the browserify module directly
gulp-rimraf	use the `del` module
gulp-image-optimization	duplicate of gulp-imagemin
gulp-bower	use the bower module directly
gulp-php	use PHP directly through gulp-spawn or ChildProcess.spawn()
gulp-tinypng	uses fs to create a temp .gulp folder
gulp-css	duplicate of gulp-minify-css
gulp-cssmin	duplicate of gulp-minify-css
gulp-clean-old	duplicate of gulp-clean
gulp-if-else	duplicate of gulp-if
gulp-prettify	duplicate of gulp-html-prettify
...	