EBook Gratis

APRENDIZAJE gulp

Free unaffiliated eBook created from
Stack Overflow contributors.

#gulp

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gulp

It is an unofficial and free gulp ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gulp.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con gulp

Observaciones

Gulp es un sistema de compilación de JavaScript, basado en node.js, como Grunt. Le permite automatizar tareas comunes durante su proceso de desarrollo. Gulp usa las secuencias -concepto y código-sobre-configuración para un proceso de compilación más simple e intuitivo. El concepto de código sobre configuración permite crear tareas más legibles y sencillas, mientras que las tareas intensivas están altamente sobre configuradas.

Versiones

Versión	Fecha de lanzamiento
3.4	2014-01-17
3.7	2014-06-01
3.8	2014-06-10
3.9	2015-06-02
3.9.1	2016-02-09
4.0.0	2016-06-21

Examples

Instalación o configuración

1. Instale Node.js y NPM:

Gulp requiere Node.js y NPM, el administrador de paquetes de Node. La mayoría de los instaladores incluyen NPM con Node.js. Consulte la documentación de instalación o confirme que ya está instalado ejecutando el siguiente comando en su terminal,

```
npm -v
// will return NPM version or error saying command not found
```

2. Instala gulp globalmente:

Si ya instaló una versión de gulp globalmente, ejecute `npm rm --global gulp` para asegurarse de que su versión anterior no coincida con **gulp-cli** .

```
$ npm install --global gulp-cli
```

3. Inicialice el directorio de su proyecto:

```
$ npm init
```

4. Instale gulp en las devoluciones de su proyecto:

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

5. Cree un gulpfile.js en la raíz de su proyecto:

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

gulp.task('default', function() {
  // place code for your default task here
});
```

6. Ejecutar trago:

```
$ gulp
```

La tarea por defecto se ejecutará y no hará nada.

Para ejecutar tareas individuales, use `gulp <task> <othertask>` .

Dependencia de la tarea

Puede ejecutar tareas en serie, pasando un segundo parámetro a `gulp.task()` .

Estos parámetros son una serie de tareas que deben ejecutarse y completarse antes de que su tarea se ejecute:

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

gulp.task('one', function() {
   // compile sass css
});

gulp.task('two', function() {
   // compile coffeescript
});

// task three will execute only after tasks one and two have been completed
// note that task one and two run in parallel and order is not guaranteed
gulp.task('three', ['one', 'two'], function() {
   // concat css and js
});
```

```
// task four will execute only after task three is completed
gulp.task('four', ['three'], function() {
   // save bundle to dist folder
});
```

También puede omitir la función si solo desea ejecutar un paquete de tareas de dependencia:

```
gulp.task('build', ['array', 'of', 'task', 'names']);
```

Nota: Las tareas se ejecutarán en paralelo (todas a la vez), así que no asuma que las tareas se iniciarán / finalizarán en orden. Iniciando gulp v4 , `gulp.series()` debe usarse si el orden de ejecución de las tareas de dependencia es importante.

Archivo concat js en subcarpeta usando gulp

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

// include plug-ins
var uglify = require('gulp-uglify'),
    concat = require('gulp-concat');

// Minified file
gulp.task('packjsMin', function() {
   return gulp.src('node_modules/angular/*.js')
      .pipe(concat('script.js'))
      .pipe(uglify())
      .pipe(gulp.dest('Scripts'));
});

//Not minified file
gulp.task('packjs', function () {
    return gulp.src('node_modules/angular/*.js')
      .pipe(concat('allPackages.js'))
      .pipe(gulp.dest('Scripts'));
});
```

gulp documentos de la CLI

Banderas

gulp tiene muy pocas banderas que conocer. Todas las demás banderas son para tareas para usar si es necesario.

- `-v` o `--version` mostrará las versiones globales y locales.
- `--require <module path>` requerirá un módulo antes de ejecutar el gulpfile. Esto es útil para transpilers pero también tiene otras aplicaciones. Puedes usar múltiples banderas `--require`
- `--gulpfile <gulpfile path>` configurará manualmente la ruta de gulpfile. Útil si tienes múltiples gulpfiles. Esto establecerá la CWD en el directorio gulpfile también
- `--cwd <dir path>` configurará manualmente la CWD. La búsqueda del gulpfile, así como la relatividad de todos los requisitos será desde aquí.
- `-T` o `--tasks` mostrará el árbol de dependencias de la tarea para el gulpfile cargado

- `--tasks-simple` mostrará una lista de tareas de texto `--tasks-simple` para el gulpfile cargado
- `--color` forzará a los complementos de trago y trago a mostrar colores incluso cuando no se detecte soporte de color
- `--no-color` forzará que los complementos de trago y trago no muestren colores incluso cuando se detecta soporte de color
- `--silent` deshabilitará todo el registro de `--silent`

La CLI agrega process.env.INIT_CWD, que es el cwd original desde el que se inició.

Indicadores de tareas específicas

Consulte este enlace de StackOverflow para saber cómo agregar indicadores específicos de tareas

Tareas

Las tareas se pueden ejecutar ejecutando `gulp <task> <othertask>` . Simplemente ejecutando `gulp` ejecutará la tarea que registró llamada por `default` . Si no hay una tarea por `default` , se producirá un error.

Compiladores

Puede encontrar una lista de idiomas compatibles en interpretar . Si desea agregar soporte para un nuevo idioma, envíe la solicitud de extracción / abra los problemas allí.

Lea Empezando con gulp en línea: https://riptutorial.com/es/gulp/topic/1341/empezando-con-gulp

Capítulo 2: Compresión sin pérdida de imagen (con gulp-imagemin)

Sintaxis

1. imagemin ([plugins], {opciones})

Parámetros

Argumento	Descripción
`sourcePath`	Directorio fuente de las imágenes (por ejemplo: `/assets/images`)
`buildPath`	Ruta de destino (por ejemplo: `/static/dist/`)

Observaciones

El primer argumento de `imagemin` constructor es la matriz de plugins. De forma predeterminada, se utilizan los siguientes complementos: `[imagemin.gifsicle(), imagemin.jpegtran(), imagemin.optipng(), imagemin.svgo()]`

Segundo argumento son las opciones. En el ejemplo anterior se utilizan las siguientes opciones:

```
{
  progressive: true,
  interlaced: true,
  svgoPlugins: [{removeUnknownsAndDefaults: false}, {cleanupIDs: false}]
}
```

Esos son completamente opcionales.

`progressive` es utilizado por `imagemin-jpegtran` .

`interlaced` es utilizado por `imagemin-gifsicle` .

`removeUnknownsAndDefaults` y `cleanupIDs` son utilizados por `imagemin-svgo` .

Examples

Instalación y uso

Instalación de dependencia (https://www.npmjs.com/package/gulp-imagemin)

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

Uso

```
/*
 * Your other dependencies.
 */

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

/*
 * `gulp images` - Run lossless compression on all the images.
 */
gulp.task('images', function() {
  return gulp.src(sourcePath) // e.g. /assets/images
    .pipe(imagemin({
      progressive: true,
      interlaced: true,
      svgoPlugins: [{removeUnknownsAndDefaults: false}, {cleanupIDs: false}]
    }))
    .pipe(gulp.dest(buildPath + 'images')); // e.g. /static/dist/
});
```

Lea Compresión sin pérdida de imagen (con gulp-imagemin) en línea:
https://riptutorial.com/es/gulp/topic/6549/compresion-sin-perdida-de-imagen--con-gulp-imagemin-

Capítulo 3: Concatenando archivos

Examples

Concat todos los archivos css en uno usando gulp-concat

Primero, instale el plugin `gulp` y `gulp-concat` a su proyecto localmente

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

y agregue la tarea gulp-concat a su gulpfile.js

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

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

gulp.task('css', function() {
    return gulp.src('css/*.css')
        .pipe(concat('concat.css'))
        .pipe(gulp.dest('css/dist'));
});

gulp.task('default', ['css']);
```

Después de iniciar el comando `gulp` , el complemento gulp-concat tomará todos los archivos CSS ubicados en el directorio `css/` y los concatenará en un archivo `css/dist/concat.css`

Concat y Uglify JS y CSS archivos

Recuerde npm instalar primero todos los archivos en devDependencies. P.ej

```
npm install --save-dev gulp gulp-concat gulp-rename gulp-uglify gulp-uglifycss
```

Gulpfile.js

```
var gulp = require('gulp');
var gulp_concat = require('gulp-concat');
var gulp_rename = require('gulp-rename');
var gulp_uglify = require('gulp-uglify');
var uglifycss = require('gulp-uglifycss');

var destDir = './public/assets/dist/'; //or any folder inside your public asset folder
var tempDir = './public/assets/temp/'; //any place where you want to store the concatenated,
but unuglified/beautified files
//To concat and Uglify All JS files in a particular folder
gulp.task('js-uglify', function(){
    return gulp.src(['./public/js/**/*.js','./public/assets/js/*.js']) //Use wildcards to
select all files in a particular folder or be specific
        .pipe(gulp_concat('concat.js')) //this will concat all the files into concat.js
```

```
        .pipe(gulp.dest(tempDir)) //this will save concat.js in a temp directory defined above
        .pipe(gulp_rename('uglify.js')) //this will rename concat.js to uglify.js
        .pipe(gulp_uglify()) //this will uglify/minify uglify.js
        .pipe(gulp.dest(destDir)); //this will save uglify.js into destination Directory
defined above
});
//To Concat and Uglify all CSS files in a particular folder
gulp.task('css-uglify', function () {
  gulp.src('./public/assets/css/*.css') //Use wildcards to select all files in a particular
folder or be specific
  .pipe(gulp_concat('concat.css')) //this will concat all the source files into concat.css
        .pipe(gulp.dest(tempDir)) //this will save concat.css into a temp Directory
        .pipe(gulp_rename('uglify.css')) //this will rename concat.css into uglify.css, but
will not replace it yet.
    .pipe(uglifycss({
      "maxLineLen": 80,
      "uglyComments": true
    })) //uglify uglify.css file
    .pipe(gulp.dest(destDir)); //save uglify.css
});
```

Ejecutarlos siguiendo los comandos

```
gulp js-uglify
gulp css-uglify
```

Lea Concatenando archivos en línea: https://riptutorial.com/es/gulp/topic/4398/concatenando-archivos

Capítulo 4: Crear documentación con gulp-jsdoc3.

Examples

Instalación

En primer lugar, instale `gulp` y `gulp-jsdoc3` en su proyecto:

```
npm install gulp-jsdoc3 --save-dev
```

Luego agrégalo a tu gulpfile.js

```
var gulp = require('gulp');
var jsdoc = require('gulp-jsoc3');



gulp.task('doc', function (cb){
    gulp.src('src/*.js')
    .pipe(jsdoc(cb));
});
```

Para documentar, por ejemplo, una función, debe agregar un comentario justo en la parte superior de la función, como esto:

```
/**
 * @function example
 * @summary This is a short description of example
 * @author Whoever
 * @param {any} cb
 * @returns
 */
function example(cb){
    //Code
}
```

Si desea saber más etiquetas de bloque para usar, visite usejsdoc.org

Lea Crear documentación con gulp-jsdoc3. en línea: https://riptutorial.com/es/gulp/topic/7365/crear-documentacion-con-gulp-jsdoc3-

Capítulo 5: Crear un observador

Examples

Tarea de vigilante

`config.paths.html` representa la ruta a su archivo HTML.

```
gulp.task("watch", function() {
    gulp.watch(config.paths.html, ["html"]);
});
```

La tarea también debe agregarse al valor predeterminado:

```
gulp.task("default", ["html", "watch"]);
```

Lea Crear un observador en línea: https://riptutorial.com/es/gulp/topic/5026/crear-un-observador

Capítulo 6: Eliminar archivos usando Gulp

Observaciones

Nota sobre el uso del patrón globbing (`**`):

El patrón global coincide con todos los `children` **y** los `parent` . Para evitar esto, agregamos `'!public'` a nuestra tarea del para que el directorio `public` no se borre.

Examples

Eliminar archivos usando del

Primero, instale `gulp` y `del` directorio de proyectos localmente

```
npm install --save-dev gulp del
```

Luego agrega la tarea `clean` a tu `gulpfile.js`

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

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

// Task to delete target build folder
gulp.task('clean', function() {
  return del(['public/**', '!public']);
});

gulp.task('default', ['clean']);
```

Esta tarea borra todos los archivos en el directorio público.

La tarea en el código se agrega como una dependencia para la tarea `'default'` , por lo que cada vez que se ejecute el `default` , la `clean` se ejecutará antes.

También puede llamar a la tarea `clean` manualmente ejecutando el comando:

```
gulp clean
```

Lea Eliminar archivos usando Gulp en línea: https://riptutorial.com/es/gulp/topic/7189/eliminar-archivos-usando-gulp

Capítulo 7: Guía completa para un flujo de trabajo frontal con Gulpjs 2 de 2

Observaciones

Guay. Así que todos hemos terminado con nuestra automatización de flujo de trabajo.

Ahora tenemos un archivo trago, que

- Responde y minimiza imágenes.
- limpia, repara automáticamente, concatena y minimiza Css
- Concatena y minimiza JS
- Observa los cambios en los activos, ya sea HTML | CSS | JS y desencadena tareas asociadas.
- Crea un directorio de compilación y almacena todo el código listo para la implementación procesada dentro de él. Y todo eso, en el fondo, mientras que usted acaba de desarrollar su aplicación.

Referencias

Secuencia de ejecución del navegador de sincronización

Examples

Configuración de la sincronización del navegador y configuración de observadores para estilo y secuencias de comandos

NOTA

> **Esta página ilustra el uso de complementos de truco como browser-sync, gulp-watch y run-sequence, y continúa discutiendo gulp-workflow-automation desde donde lo dejamos en Gulpjs-workflow-automation-1 de 2. En caso de que haya aterrizado aquí, Considera** pasar por ese post primero.

- Tarea predeterminada
- Tarea de vigilancia: para construir continuamente sus activos listos para el despliegue sobre la marcha, siempre que haya algo de imagen | JS | Cambios de css en el curso del desarrollo.

Comencemos con la sincronización del navegador.

Gulp Watchdog Task

Comencemos con la tarea de vigilancia.

El objetivo es observar los cambios que realice durante el desarrollo. Cualquier cambio, debe desencadenar la correspondiente tarea engullida.

Además, necesitamos una funcionalidad que sincronice sus cambios en el navegador.

Sincronización del navegador

Por lo tanto, tenemos que instalar la sincronización del navegador.

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

Con esa premisa, abramos nuestro gulpfile.js y agreguemos la funcionalidad del reloj. Necesitamos la sincronización del navegador y definir algunas variables para usar su funcionalidad.

En la parte superior del gulpfile, agregue el siguiente fragmento de código. Colóquelo justo debajo de las declaraciones de compresión de la imagen.

al igual que:

```
//Browser-sync

var sync = require('browser-sync').create();
var reload = sync.reload;
```

Tener la sincronización del navegador sincroniza su desarrollo con el navegador, es una configuración simple. Vamos a crear una tarea llamada perro guardián.

al igual que:

```
$.task('watchdog', function() {


})
```

Ahora, si exploramos las opciones de sincronización del navegador aquí y buscamos la configuración del servidor, podemos ver lo fácil que es.

Solo tenemos que colocar el interior debajo de nuestra tarea de vigilancia.

Fragmento - 1 - placa de calderín guardián interior

```
/*
 Initiate Browser sync
```

```
@documentation - https://www.browsersync.io/docs/options/
*/
sync.init({
 server: {
   baseDir: "./"
},
port: 8000 //change it as required
});
```

Inserta lo anterior en el interior de tu placa de control de arriba.

El siguiente fragmento, es definir un observador para los estilos, con el objetivo de volver a procesar los archivos CSS modificados o nuevos, y desencadenar una recarga del navegador automáticamente.

fragmento - 2 - placa de calderín de perro guardián interior

```
$.watch(['css/**/*', 'fonts/google/**/*.css'], reload).on('change', function(event) {
console.log(event.type + ':' + event.path)
if (event.type === 'deleted') {
uncache('styles', event.path);
$$.remember.forget('auto-prefixed-stylesheets', event.path);
}
sequence('optimizeStyles')
});
```

Inserta lo anterior en el interior de tu placa de control de arriba.

Entonces estamos monitoreando `" [fonts/google/**/*.css , /**/*.css ]"` es decir,

todos los archivos css bajo css / todos los archivos css bajo fonts / google / Cuando algo cambia, o se agrega un nuevo archivo, activa el método de recarga, que se define en la parte superior de nuestro archivo gulp, en la declaración de browsersync.

> Nota: Es posible que observe que tenemos un controlador de eventos **.on** adjunto al observador.

```
$.watch(['css/**/*', 'fonts/google/**/*.css'], reload).on('change', function(event)
```

Básicamente, cualquier CUD (Crear | Actualizar | Eliminar) activa la función de recarga y pasa un objeto de evento como parámetro a la función de devolución de llamada.

La devolución de llamada es una función vital, en la que podemos lograr operaciones como la eliminación de caché en la eliminación de activos. Ahora el objeto de evento tiene parámetros como

- camino

- tipo - Crear / Actualizar / Eliminar

Si se elimina un activo, debemos asegurarnos de que las cachés que construimos en nuestras funciones de minificación anteriores, a través de gulp-cached y gulp-remember, necesitan actualización.

estamos manejando eso en el fragmento siguiente, que está dentro de la devolución de llamada en el cambio.

```
if (event.type === 'deleted') {
uncache('styles', event.path);
$$.remember.forget('auto-prefixed-stylesheets', event.path);
}
```

Nota

$ -> alias para trago

$$ -> alias para gulp-load-plugins

También puede notar que tengo una `sequence('optimizeStyles');` despues escribi la invocacion unacheque

El método de secuencia, asegura, método síncrono se ejecuta de forma asíncrona por javascript predeterminado.

instalarlo es simple

HACER

```
bash $ npm install run-sequence
```

luego, declare en su archivo gulp justo debajo de la declaración de sincronización del navegador.

```
var sequence = require('run-sequence');
```

Así que con esa comprensión, el observador de guiones es fácil de escribir. Sólo diferentes globos!

Entonces, agregue este fragmento debajo del observador de estilos dentro de la placa de control del perro guardián.

Fragmento - 3 - dentro de la placa de control de la tarea Watchdog

```
/*
on addition or change or deletion of a file in the watched directories
the change event is triggered. An event object with properties like
path,
event-type
is available for perusal passed to the callback

*/
$.watch('js/**/*', reload).on('change', function(event) {
console.log(event.type + ':' + event.path)
if (event.type === 'deleted') {
uncache('scripts', event.path);
$$.remember.forget('linted-scripts', event.path);
}
sequence('optimizeScripts');
});
```

Nota

Usamos dos funciones en nuestros fragmentos arriba.

- desentrañar
- $$. recordar. olvidar Nota:

$ -> Alias para trago

$$ -> Alias para gulp-load-plugins

Definamos la función uncache en algún lugar de nuestro gulpfile.js, antes de invocarla.

```
/*
Deletes a cache entry
*/
var uncache = function(cacheName, cacheKey) {
        var cache = $$.cached;
        if (cache.caches[cacheName] && cache.caches[cacheName][cacheKey])
            return delete cache.caches[cacheName][cacheKey];
        return false;
    }
    /*
    logs current cache created via gulp-cached
    */
var viewCache = function() {
    console.log($$.cached.caches)
}
```

Definir una tarea predeterminada

Así que ahora, terminemos el código gulpfile, definiendo una tarea predeterminada.

la tarea predeterminada es la que se ejecuta, cuando se acaba de decir

```
gulp
```

en un indicador de comando debajo de la raíz de su proyecto.

```
$.task('default', ['generateResponsiveImages'], function() {
 $.start('watchdog');
 console.log('Starting Incremental Build');
});
```

Lea Guía completa para un flujo de trabajo frontal con Gulpjs 2 de 2 en línea: https://riptutorial.com/es/gulp/topic/6343/guia-completa-para-un-flujo-de-trabajo-frontal-con-gulpjs-2-de-2

Capítulo 8: Guía completa para una automatización de flujo de trabajo de front-end con Gulpjs -1 de 2

Sintaxis

- npm install [nombre del plugin] --save-dev
- npm install [nombre del plugin] --save
- Función `<function-name>` (glob) {$ .src (glob) .pipe ([plugin 1]). Pipe ([plugin 2]) .... pipe ([plugin n]). Pipe ($ .dest (`<destination-name>`)}
- `$.task(<taskname> , [dependencies] , <body>);`

Observaciones

Continuar leyendo - Crear una tarea predeterminada y configurar la sincronización del navegador

Referencias

- Trago limpio css
- Gulp-Uglify
- Gulp-Autoprefixer - para prefijos Css que mantienen la compatibilidad del navegador
- Gulp-Cached
- Tragar-Recordar
- Gulp-Jshint

Guía paso a paso para la automatización del flujo de trabajo de Gulp para principiantes absolutos que documenté en mi blog

Examples

Cargando todos los complementos de Package.JSON

Suponiendo que tiene una idea de cómo instalar trago, permítanos sumergirnos hasta requerir todas las dependencias de gulp de package.json en la carpeta raíz de su proyecto.

En caso de que todavía no tenga un gulpfile, cree un archivo vacío con el nombre

gulpfile.js

Primero, requerimos un trago. al igual que:

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

A continuación, vamos a cargar todos nuestros complementos, en nuestro gulpfile, por el siguiente fragmento de código

Nota

Lea los comentarios en todos los fragmentos que incluiremos en esta lectura, ya que brindan más información sobre cada funcionalidad.

```
/*
require gulp-load-plugins, and call it
gulp-load-plugins will require individually,
all the plugins installed in package.json at the root directory
these required plugins, are lazy loaded
that is, gulp-load-plugins will only require them when they are referenced in this file
plugins are available for perusal, via camelcased names, minus gulp
eg: gulp-clean-css is accessed by $$.cleanCss
*/
var $$ = require('gulp-load-plugins')();
```

NOTA

A lo largo de los muchos ejemplos que tendremos en el artículo, tenemos alias

1. trago como $ y
2. gulp-load-plugins como $$

puramente para facilitar la escritura.

Instalación de complementos para imágenes receptivas | Minificación Css | Minificación Js

Nuestro objetivo, es

1. Haga que sus imágenes se ajusten a los anchos y las escalas de manera adecuada, generando una batería de imágenes de anchos variados, y para minimizarlas
2. Lint tu Javascript
3. Minimice sus activos: JS / CSS / HTML, lo que le permite alojar un código más ligero que el más ligero
4. Mire los archivos css / js / image para ver si hay cambios y reconstruya las optimizaciones
5. Sincronizar los cambios durante el desarrollo, en un navegador que sirva su sitio.

Necesitamos una serie de complementos, así que instalémoslos todos. Por favor, ejecute todos estos en la carpeta raíz del proyecto.

Complementos de procesamiento de imágenes

```
bash $ npm install --save-dev gulp-responsive
bash $ npm install --save-dev gulp-imagemin
bash $ npm install --save-dev imagemin
bash $ npm install --save-dev imagemin-jpeg-recompress
bash $ npm install --save-dev imagemin-pngquant
```

Complementos optimizador de activos

```
bash $ npm install --save-dev gulp-clean-css
bash $ npm install --save-dev gulp-uglify
bash $ npm install --save-dev gulp-minify-html
bash $ npm install --save-dev gulp-jshint
bash $ npm install --save-dev gulp-concat
bash $ npm install --save-dev gulp-autoprefixer
```

Anatomía de una función trilla.

```
[Function <name>] (glob) {

$.src(glob)

.pipe([plugin 1])
.pipe([plugin 2])
.
.
.
.pipe([plugin n])
.pipe( $.dest(<destination-name>)

}
```

Nota

> pipe es un método que transmite todos los archivos que coinciden con la entrada global, a nuestros complementos (minifyhtml en este caso).
>
> Es simple de imaginarlo así:
>
> $ .src es lo que construye el flujo y la tubería canaliza cada archivo individual que hace coincidir el globo hacia abajo a cada complemento en la tubería. Cada complemento al que se pasa el archivo, modifica su contenido en la memoria solo hasta que se alcanza $ .dest, que luego se actualiza / crea archivos transmitidos por $ .src

Cuando,

$ -> trago

$$ -> gulp-load-plugins

Entonces, antes de escribir las funciones del optimizador, necesitamos instalar un par de complementos de almacenamiento en caché.

```
bash $ npm install --save-dev gulp-cached
bash $ npm install --save-dev gulp-remember
```

Usted podría preguntarse por qué dos cachés eh !. **gulp-cached** , pasa solo contenido modificado o nuevo por la tubería a otros complementos. Por lo tanto, dado que queremos que los archivos sin cambios se utilicen para concatenar en un solo archivo por activo (css | js) también, necesitamos un **truco,** además de gulp-cached

Primero usamos **gulp-cached** para construir una lista de archivos que han cambiado

En segundo lugar, necesitamos un **trago, recordar** para mantener un seguimiento de todos los archivos que pasan por esa lista en la memoria.

Primera ejecución: no hay archivos en caché, por lo que gulp-caché los pasará a todos para recordarlos.

Ejecuciones subsiguientes: solo los archivos nuevos o modificados se canalizan por gulp-cached. Dado que el contenido del archivo actual ha cambiado, gulp-remember actualiza su caché.

Genial, escribamos nuestro primer optimizador.

Optimizador de estilo

```
// Goal

/*

1. cache existing files on the first run
2. each file ,
       a. Is autoprefixed with cross browser compatible prefixes for any css property (
justify-content for e.g)
       b. Is concatenated into app.css
3. app.css is minified
4. On subsequent runs , the above process is implemented on file modifications/additions only

*/

/*
*@src - input a glob pattern - a string eg 'images/**/*' or '**/*.css' or, an array eg
['glob1','glob2']
*/
var optimizeStyles = function(src) {

return $.src(src).
pipe($$.cached('styles')).
pipe($$.autoprefixer({
```

```
browsers: ['last 2 versions']
})).
pipe($$.remember('auto-prefixed-stylesheets')).
pipe($$.concat('app.css')).
pipe($.dest('build/css')).
pipe($$.cleanCss()).
pipe($$.rename({
suffix: '.min'
})).
pipe($.dest('build/css'))
}
```

Nota

$ -> trago

$$ -> gulp-load-plugins

$ .src -> crea secuencias de archivos que coinciden con el globo pasado como src

$ .dest -> guarda el archivo manipulado en la ruta especificada

Optimizador de secuencias de comandos

```
// Goal

/*

1. cache existing files on the first run
2. each file ,
       a. Is linted with jshint
       b. Is concatenated into app.js
3. app.js is minified
4. On subsequent runs , the above process is implemented on file modifications/additions only

*/

/*
*@src - input a glob pattern - a string eg 'js/**/*' or '**/*.js' or, an array eg
['glob1','glob2']
*/

var optimizeScripts = function(src) {

    return $.src(src).
    pipe($$.cached('scripts')).
    pipe($$.jshint()).
    pipe($$.jshint.reporter('default')).
    pipe($$.jshint.reporter('fail')).
    pipe($$.remember('linted-scripts')).
    pipe($$.concat('app.js')).
    pipe($.dest('build/js')).
    pipe($$.uglify()).
    pipe($$.rename({
        suffix: '.min'
```

```
    })).
    pipe($.dest('build/js'))


}
```

Nota

$ -> trago

$$ -> gulp-load-plugins

$ .src -> crea secuencias de archivos que coinciden con el globo pasado como src

$ .dest -> guarda el archivo manipulado en la ruta especificada

Generar imágenes responsivas

Pasemos ahora al procesamiento de imágenes. Por lo tanto, el objetivo es tener una variedad de tamaños para cada imagen que se va a servir.

¿Por qué?

> Bueno, para comprender por qué necesitamos una batería de imágenes con un rango de anchos, debemos reflexionar sobre el hecho de que probablemente haya millones de dispositivos con resoluciones variadas. Necesitamos una imagen a escala sin mucha pixelación. Al mismo tiempo, necesitamos mejorar los tiempos de carga de la página, descargando solo una imagen, que se ajuste al ancho que contiene, y también con la dimensión más pequeña posible, para hacerlo. Hay blogs académicos como el que escribió Eric Portis, que resalta la ineficacia de las consultas de los medios de comunicación y sirve como una guía completa para comprender conceptos como conjuntos y tamaños.

Puede referirse a la epopeya de Eric Portis aquí.

Ahora, nuestra función, necesita tomar un globo, y un ancho como entradas, y hacer su magia y empujar el archivo que genera cada ejecución, a un destino y minimizar la imagen respondida.

Hemos instalado dos complementos de compresión de imagen en nuestro primer ejemplo

Dado que estos complementos **NO** comienzan con un prefijo "trago", debemos cargarlos manualmente en nuestro archivo de gulp.

Por lo tanto, exigámoslos manualmente, después de la declaración gulp-load-plugins en la parte superior de nuestro archivo gulp.

al igual que:

```
var compressJpg = require('imagemin-jpeg-recompress');
var pngquant = require('imagemin-pngquant');
```

Vale la pena tener en cuenta que el procesador de imagen nítido responde a tragos, lo que es mejor que imagemagick BY FAAAAR !. Sharp es lo que utiliza gulp-responsive para recortar sus imágenes a los anchos deseados.

Puede consultar las opciones de configuración que responden rápidamente, para obtener una lista completa de los parámetros de configuración. Solo he usado

- ancho: para recortar nuestras imágenes al ancho w, pasado como parámetro
- renombrar - para agregar un sufijo al nombre de la imagen, para que siga siendo único

en mi función de configuración a continuación. por lo tanto, nuestra función recortará la imagen al ancho pasado como entrada, para todas las imágenes coincidentes descifradas a través de la entrada global. luego, cada imagen se comprime con jpeg-recompress o pngquant y se guarda dentro de build / images.

Con eso en mente, nuestra función sería así:

```
/*
@generateResponsiveImages
*@Description:takes in a src of globs, to stream matching image files , a width,
*to resize the matching image to, and a dest to write the resized and minified files to
*@src - input a glob pattern - a string eg 'images/** /*' or 'images/*' or, an array
eg ['glob1','glob2']
*@return returns a stream
*/
var generateResponsiveImages = function(src, width, dest) {

    //declare a default destination
    if (!dest)
        dest = 'build/images';
    //case 1: src glob -  images/**/*
    // the base is the directory immediately preceeding the glob - images/ in this case
    //case 2: images/fixed/flourish.png : the base here is images/fixed/ - we are overriding
    // that by setting base to images.This is done so that, the path beginning after images/
    // - is the path under our destination - without us setting the base, dest would be,
    //build/images/flourish.png.But with us setting the base, the destination would be
    // build/images/fixed/flourish.png
    return $.src(src, {
        base: 'images'
    })

    //generate resized images according to the width passed
    .pipe($$.responsive({
            //match all pngs within the src stream
            '**/*.png': [{
                width: width,
                rename: {
                    suffix: '-' + width
                },
                withoutEnlargement: false,
            }],
            //match all jpgs within the src stream
```

```
            '**/*.jpg': [{
                width: width,
                rename: {
                    suffix: '-' + width
                },
                progressive: true,
                withoutEnlargement: false,
            }]
        }, {

            errorOnEnlargement: false,
            errorOnUnusedConfig: false,
            errorOnUnusedImage: false

        }))
        //once the file is resized to width, minify it using the plugins available per format
        .pipe($$.if('*.jpg', compressJpg({
            min: 30,
            max: 90,
            target: 0.5
        })()))
        //use file based cache gulp-cache and it will minify only changed or new files
        //if it is not a new file and if the contents havent changed, the file is served from
cache
        .pipe($$.cache($$.imagemin({
            verbose: true
        })))


    //write to destination - dest + path from base
    .pipe($.dest(dest));
}
```

Nota

$ -> trago

$$ -> gulp-load-plugins

$ .src -> crea secuencias de archivos que coinciden con el globo pasado como src

$ .dest -> guarda el archivo manipulado en la ruta especificada

Referencias adicionales

- Gulp-Responsive
- Gulp-Imagemin

Minificador de HTML

```
*
 *@minifyHtml
```

```
 *Description:takes in a glob src, and minifies all '.html' files matched by the glob
 *@src - input a glob pattern - a string eg '/**/*.html /*' or '*.html' or, an array eg
['glob1','glob2']
 *@dest=file.base means, the modified html file will be in the same directory as the src file
being minified
 *often means, the process is just a modification on the existing src file
 *@return returns a stream
 */
var minifyHtml = function(src) {
    return $.src(src)
        .pipe($$.minifyHtml())
        .pipe($.dest(function(file) {
            //file is provided to a dest callback -
            // Refer here https://github.com/gulpjs/gulp/blob/master/docs/API.md#path
            return file.base;
        }));
}
```

Anatomía de una tarea trillada.

La anatomía de una definición de tarea es así:

```
$.task(<taskname> , [dependencies] , <body>);
```

dependencias, es una serie de tareas que TIENEN que finalizar antes de que se ejecute la tarea actual que está definiendo. Más como forzar una ejecución síncrona en lugar de la funcionalidad asíncrona predeterminada.

Añadiendo tareas Gulp

Así que ahora tenemos

- Una función definida arriba para optimizar los estilos.
- Una función definida arriba para optimizar scripts.
- Una función definida arriba para optimizar HTML.
- Una función para generar múltiples imágenes por imagen arriba.

Todo lo que tenemos que hacer ahora, es invocarlos cuando sea necesario.

Escribamos nuestras tareas de acuerdo con la sintaxis que definimos anteriormente

```
/*
* $.task('name','dependency array',function)
results in building a task object as below
task:{
'name':name,
'dep':[array of dependencies],
'fn':function
}
*/


//*@return returns a stream to notify on task completion
$.task('optimizeHtml', function() {
    var src = ['**/*.html', '!{node_modules}/**/*.html'];
```

```
    return minifyHtml(src);
});

//*@return returns a stream to notify on task completion
$.task('optimizeScripts', function() {
    var src = ['js/**/*.js'];
    return optimizeScripts(src);
});

//*@return returns a stream to notify on task completion
$.task('optimizeStyles', function() {
    var src = ['css/**/*.css', 'fonts/google/**/*.css'];
    return optimizeStyles(src);
});

//Take in a callback to ensure notifying the gulp engine, that the task is done
//required since, you are not returning a stream in this task
$.task('generateResponsiveImages', function(callback) {
    var src = ['images/**/*.{jpg,png}'];
    for (var i = widths.length - 1; i >= 0; i--) {
        generateResponsiveImages(src, widths[i]);
    }
    callback();

});
```

Lea Guía completa para una automatización de flujo de trabajo de front-end con Gulpjs -1 de 2 en línea: https://riptutorial.com/es/gulp/topic/6342/guia-completa-para-una-automatizacion-de-flujo-de-trabajo-de-front-end-con-gulpjs--1-de-2

Capítulo 9: Gulp Path

Examples

Creando Simple Gulp Path a la aplicación

Antes de ejecutar gulp necesitas instalar `Node.JS` y `npm`

El jefe de la Gulp.js es:

```
var gulp = require('gulp');
    clean = require('gulp-clean'); // A gulp plugin for removing files and folders.
    imagemin = require('gulp-imagemin'); // Minify PNG, JPEG, GIF and SVG images
```

A continuación, asigne la ruta de los archivos trilla

```
var bases = {
 app: 'app/',         // path to your app
 dist: 'dist/',       // path to the compiled application
};

var paths = {
 scripts: ['scripts/**/*.js', '!scripts/libs/**/*.js'],
 libs: ['scripts/libs/jquery/dist/jquery.js', 'scripts/libs/underscore/underscore.js',
'scripts/backbone/backbone.js'],
 styles: ['styles/**/*.css'],
 html: ['index.html', '404.html'],
 images: ['images/**/*.png'],
 extras: ['crossdomain.xml', 'humans.txt', 'manifest.appcache', 'robots.txt', 'favicon.ico'],
};
```

Ejemplo 1:

```
// Copy all other files to dist directly
    gulp.task('copy', ['clean'], function() {
 // Copy html
       gulp.src(index.html)
         .pipe(gulp.dest(bases.dist)); // same as .pipe(gulp.dest('dist'));
    });
```

Ejemplo 2:

```
// Imagemin images and ouput them in dist
gulp.task('imagemin', ['clean'], function() {
 gulp.src(paths.images, {cwd: bases.app})
 .pipe(imagemin())
 .pipe(gulp.dest(bases.dist + 'images/')); // same as .pipe(gulp.dest('dist/images/'));
});
```

Reloj predeterminado

```
// Define the default task as a sequence of the above tasks
gulp.task('default', ['clean', 'imagemin', 'copy']);
```

Lea Gulp Path en línea: https://riptutorial.com/es/gulp/topic/7759/gulp-path

Capítulo 10: Minificar CSS

Examples

Usando gulp-clean-css y gulp-rename

Primero, instale `gulp` , `gulp-clean-css` y `gulp-rename` en el directorio del proyecto localy

```
npm install --save-dev gulp gulp-clean-css gulp-rename
```

A continuación, agregue la siguiente tarea `gulpfile.js` `minify-css` a su `gulpfile.js`

```
var gulp = require('gulp');
var cleanCSS = require('gulp-clean-css');
var rename = require('gulp-rename');

gulp.task('minify-css', function() {
    return gulp.src('css/dist/dist.css')
        .pipe(cleanCSS())
        .pipe(rename('dist.min.css'))
        .pipe(gulp.dest('css/dist'));
});

gulp.task('watch', function(){
  gulp.watch('css/dist/**/*.css', ['minify-css']);
  // Other watchers
});

gulp.task('default', ['minify-css', 'watch']);
```

Aquí `.pipe(cleanCSS())` ejecuta la `css/dist/dist.css` archivo `css/dist/dist.css` y `.pipe(rename('concat.min.css'))` cambia el nombre a `dist.min.css`

Sass y Css - Preprocesamiento con Gulp

Antes de iniciar gulp necesitamos instalar `node.js` y `npm` . Luego instale gulp-sacc

```
$ npm i gulp-sass --save-dev // i = install
```

Gulp head

```
var gulp = require('gulp');
// Requires the gulp-sass plugin
var sass = require('gulp-sass');
```

Gulp Body

```
gulp.task('sass', function(){
  return gulp.src('app/scss/*.scss') // searching for sass files
```

```
    .pipe(sass()) // Converts Sass to CSS with gulp-sass
    .pipe(gulp.dest('app/css')) // destination of the css file
});
```

Reloj Gulp

```
gulp.task('watch', function(){
  gulp.watch('app/scss/**/*.scss', ['sass']);
  // Other watchers
})
```

Lea Minificar CSS en línea: https://riptutorial.com/es/gulp/topic/4396/minificar-css

Capítulo 11: Minificar HTML

Examples

Minificar HTML utilizando gulp-htmlmin

Primero, instale `gulp` y `gulp-htmlmin` en el directorio del proyecto localmente

```
npm install --save-dev gulp gulp-htmlmin
```

Luego agregue la tarea `gulpfile.js` `minify-html` a su `gulpfile.js`

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

// Task to minify HTML
gulp.task('minify-html', function() {
return gulp.src('source/*.html')
.pipe(htmlmin())
.pipe(gulp.dest('public/'));
});

gulp.task('watch', function (){
    gulp.watch('source/*.html', ['minify-html']);
// other tasks
});

gulp.task('default', ['minify-html', 'watch']);
```

Esta tarea busca todos los archivos en el directorio de `source` con una extensión `.html` , los minimiza y luego envía los archivos resultantes al directorio `public` .

La tarea en el código se agrega como una dependencia para la tarea `'default'` , por lo que cada vez que se ejecute el `default` , `minify-html` se ejecutará antes.

También puede llamar a la tarea `minify-html` manualmente ejecutando el comando:

```
gulp minify-html
```

Lea Minificar HTML en línea: https://riptutorial.com/es/gulp/topic/7187/minificar-html

Capítulo 12: Mostrar errores con gulp-jslint

Examples

Instalación y uso

Instalación

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

Uso

En gulpfile.js añadir:

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

gulp.task('lint', function(){
    return gulp.src(['source.js'])
        .pipe(jslint({ /* this object represents the JSLint directives being passed down */
}))
        .pipe(jslint.reporter( 'my-reporter' ));
});
```

para utilizar esta tarea:

```
$ ./node_modules/gulp/bin/gulp.js lint
```

Lea Mostrar errores con gulp-jslint en línea: https://riptutorial.com/es/gulp/topic/7415/mostrar-errores-con-gulp-jslint

Capítulo 13: Reduciendo JS

Sintaxis

- `ext` Un objeto que especifica la fuente de salida y las extensiones de archivo minimizadas.
- `source` La cadena de sufijo de los nombres de archivo con los que se generan los archivos de origen finaliza.
- `min` When string: la cadena de sufijo de los nombres de archivo con los que se imprimen los archivos minificados termina con
- When Array: las expresiones regex que se reemplazarán con nombres de archivos de entrada. Por ejemplo: [/.(.*)-source.js$/, '$ 1.js']
- `exclude` No minimizará los archivos en los directorios.
- `noSource` No `noSource` el código fuente en los `noSource` .
- `ignoreFiles` No minimizará los archivos que coincidan con el patrón.
- `mangle` Pase `false` para omitir nombres de mangling.
- `output` Pase un objeto si desea especificar `output options` . Los valores predeterminados están optimizados para la mejor compresión.
- `compress` Pasa un objeto para especificar `compressor options` personalizadas de `compressor options` . Pase falso para omitir la compresión completamente.
- `preserveComments` Una opción de conveniencia para options.output.comments. Por defecto conserva ningún comentario.
- `all` Conservar todos los comentarios en bloques de código
- `some` comentarios de Preserve que comienzan con una explosión `(!)` o incluyen una directiva de compilador de cierre `(@preserve, @license, @cc_on)`
- `function` Especifique su propia función de preservación de comentarios. Se le pasará el nodo actual y el comentario actual y se espera que devuelva `true` o `false` .

Observaciones

Enlaces útiles para gulp-minify

Examples

Minify JS utilizando gulp-minify

Primero, instale `gulp` y `gulp-minify` en el directorio del proyecto localmente

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

Luego agregue la siguiente tarea `min-js` a su `gulpfile.js`

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

```
gulp.task('min-js', function() {
    return gulp.src('lib/*.js')
        .pipe(minify({
            ext: {
                min: '.min.js'
            },
            ignoreFiles: ['-min.js']
        }))
        .pipe(gulp.dest('lib'))
});

gulp.task('watch', function(){
  gulp.watch('lib/*.js', ['min-js']);
  // Other watchers
});

gulp.task('default', ['min-js', 'watch']);
```

Esta tarea encuentra todos los archivos js en el directorio `lib` , lo minfy y guárdelo en el directorio `lib` con el sufijo `.min.js` . Por ejemplo, después de `lib/app.min.js` archivo `lib/app.js` se creará un archivo `lib/app.min.js`

Además de ejecutarse como una dependencia para la tarea `'default'` , esta tarea se puede ejecutar manualmente escribiendo el siguiente comando:

```
gulp min-js
```

Lea Reduciendo JS en línea: https://riptutorial.com/es/gulp/topic/4397/reduciendo-js

Capítulo 14: Utilizando Browserify

Parámetros

Opciones	Detalles
transformar	Especifica una tubería de funciones (o nombres de módulos) a través de la cual se ejecutará el paquete explorado.
depurar	Habilitar soporte de mapa de origen. `!gulp.env.production` funcionaria bien.
extensiones	La matriz de extensiones que desea omitir `require()` llamadas `require()` además de .js y .json. No se olvide .
ignorar	Conjunto de rutas que deben pasarse a la función de ignorar de browserify.
resolver	Función de resolución de nombre de módulo personalizado.
nobuiltins	Elimine los módulos incorporados definidos en `lib/builtins.js` (módulo browserify). `opts.builtins` debe estar definido y `opts.nobuiltins` puede ser un Array of Strings o simplemente un String.

Examples

Usando Browserify con Vanilla Javascript

Primero instale gulp y browserify a través de `npm i gulp gulp-browserify` . Esto instalará browserify en su carpeta `node_modules` .

gulpfile.js

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

gulp.task('script', function() {
    gulp.src('./src/script.js')
        .pipe(browserify({
            insertGlobals: true
        })
        .pipe(gulp.dest('./build/'));
}
```

Usando Browserify con Coffeescript

Primero instale gulp y browserify vía npm i gulp gulp-coffeeify. Esto instalará browserify en su carpeta node_modules.

gulpfile.js

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

gulp.task('script', function() {
    gulp.src('./src/script.coffee')
        .pipe(coffeeify())
        .pipe(gulp.dest('./build/'));
}
```

Lea Utilizando Browserify en línea: https://riptutorial.com/es/gulp/topic/6364/utilizando-browserify

Capítulo 15: Utilizando filtros de archivos.

Examples

Creación de reglas de Imágenes, JS y CSS (SASS) para ordenar archivos

Instalación de Gulp y sus tareas

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

Determinar la estructura de la carpeta

En esta estructura, utilizaremos la carpeta de la aplicación para fines de desarrollo, mientras que la carpeta dist se utiliza para contener archivos optimizados para el sitio de producción.

```
|-app
    |-css/
    |-images/
    |-index.html
    |-js/
    |-scss/
|- dist/
|- gulpfile.js
|- node_modules/
|- package.json
```

Preproceso de Gulp

```
// Requires the gulp-sass plugin
var gulp = require('gulp');
    sass = require('gulp-sass');
    uglify = require('gulp-uglify');
    imagemin = require('gulp-imagemin');
```

Gulp Task

```
gulp.task('sass', function(){
  return gulp.src('app/scss/**/*.scss') //selection all files in this derectory
    .pipe(sass()) // Using gulp-sass
    .pipe(gulp.dest('dist'))
```

```
});
gulp.task('gulp-uglify', function(){
  return gulp.src('app/js/*.js')
    // Minifies only if it's a JavaScript file
    .pipe(uglify())
    .pipe(gulp.dest('dist'))
});
gulp.task('images', function(){
  return gulp.src('app/images/**/*.+(png|jpg|gif|svg)')
  .pipe(imagemin())
  .pipe(gulp.dest('dist/images'))
});
```

Gulp Watch

```
gulp.task('watch', function(){
  gulp.watch('app/js/**/*.js', ['gulp-uglify']);
    gulp.watch('app/scss/**/*.scss', ['sass']);
    gulp.watch('app/images/**/*.*', ['images']);
  // Other watchers
});
gulp.task('build', [`sass`, `gulp-uglify`, `images`], function (){
  console.log('Building files');
});
gulp.task('default', function() {});
```

Lea Utilizando filtros de archivos. en línea: https://riptutorial.com/es/gulp/topic/7818/utilizando-filtros-de-archivos-

Creditos

S. No	Capítulos	Contributors
1	Empezando con gulp	cl3m, Community, fracz, João Silva, naresh, Nhan, Pejman, Pierre-Loup Pagniez, Pyloid, Sender, Tim
2	Compresión sin pérdida de imagen (con gulp-imagemin)	Siegmeyer
3	Concatenando archivos	Andrew Plakhotnyi, fracz, John Strood, maoizm, Mikhail, noob
4	Crear documentación con gulp-jsdoc3.	Manuel
5	Crear un observador	Muaaz Rafi, Nhan
6	Eliminar archivos usando Gulp	Mor Paz
7	Guía completa para un flujo de trabajo frontal con Gulpjs 2 de 2	Schrodinger's cat
8	Guía completa para una automatización de flujo de trabajo de front-end con Gulpjs -1 de 2	Schrodinger's cat
9	Gulp Path	GYTO
10	Minificar CSS	fracz, GYTO, Mikhail, SteveLacy
11	Minificar HTML	GYTO, Mor Paz
12	Mostrar errores con gulp-jslint	jesussegado
13	Reduciendo JS	fracz, GYTO, Mikhail, Mor Paz, Saiful Azad
14	Utilizando Browserify	dome2k

15	Utilizando filtros de archivos.	GYTO