EBook Gratuito

APPRENDIMENTO gulp

Free unaffiliated eBook created from
Stack Overflow contributors.

#gulp

Sommario

Di ... 1

Capitolo 1: Iniziare con Gulp ... 2

Osservazioni ... 2

Versioni ... 2

Examples ... 2

Installazione o configurazione ... 2

1. Installa Node.js e NPM: ... 2

2. Installa gulp a livello globale: ... 2

3. Inizializza la directory del tuo progetto: ... 3

4. Installa gulp nel tuo progetto devDependencies: ... 3

5. Crea un gulpfile.js nella radice del tuo progetto: ... 3

6. Esegui gulp: ... 3

Dipendenza delle attività ... 3

Concat js file nella sottocartella usando gulp ... 4

deglutire i documenti CLI ... 4

bandiere ... 4

Bandiere specifiche dell'attività ... 5

Compiti ... 5

I compilatori ... 5

Capitolo 2: Compressione senza perdita di immagini (con gulp-imagemin) ... 6

Sintassi ... 6

Parametri ... 6

Osservazioni ... 6

Examples ... 6

Installazione e utilizzo ... 6

Capitolo 3: Concatenazione di file ... 8

Examples ... 8

Concat tutti i file CSS in uno utilizzando gulp-concat ... 8

Concat e Uglify JS e CSS ... 8

Capitolo 4: Crea documentazione con gulp-jsdoc3 ... 10

Examples ... 10
Installazione ... 10
Capitolo 5: Crea un osservatore ... 11
Examples ... 11
Compito di osservatore ... 11
Capitolo 6: Elimina i file usando Gulp ... 12
Osservazioni ... 12
Examples ... 12
Elimina i file usando del ... 12
Capitolo 7: Guida completa a un flusso di lavoro front-end con Gulpjs 2 di 2 ... 13
Osservazioni ... 13
Riferimenti ... 13
Examples ... 13
Impostazione della sincronizzazione del browser e configurazione degli osservatori per sti ... 13
NOTA ... 13
Gulp Watchdog Task ... 13
Snippet - 1 - all'interno del boiler watchdog ... 14
snippet - 2 - all'interno del boiler watchdog ... 15
Nota ... 16
Snippet - 3 - inside boilerplate task watchdog ... 16
Nota ... 17
Definizione di un'attività predefinita ... 17
Capitolo 8: Guida completa all'automazione del flusso di lavoro front-end con Gulpjs -1 di ... 18
Sintassi ... 18
Osservazioni ... 18
Examples ... 18
Caricamento di tutti i plugin da Package.JSON ... 18
Nota ... 19
NOTA ... 19
Installazione di plugin per immagini reattive | Css Minification | Js minification ... 19
Plugin per l'elaborazione delle immagini ... 19

Plugin ottimizzatore di asset 20
Anatomia di una funzione di sorso 20
Nota 20
$ -> gulp 20
$$ -> gulp-load-plugins 20
Ottimizzazione e minimizzazione delle risorse 20
Style Optimizer 21
Nota 22
Script Optimiser 22
Nota 23
Generare immagini reattive 23
Nota 25
Ulteriori riferimenti 25
HTML Minifier 25
Anatomia di un compito ingannevole 26
Aggiunta di attività Gulp 26
Capitolo 9: Gulp Path 28
Examples 28
Creazione di Simple Gulp Path per l'app 28
Capitolo 10: Minificando JS 30
Sintassi 30
Osservazioni 30
Examples 30
Minimizza JS usando gulp-minify 30
Capitolo 11: Minificazione CSS 32
Examples 32
Usando gulp-clean-css e gulp-rename 32
Sass e Css - Pre-elaborazione con Gulp 32
Capitolo 12: Minimizzazione dell'HTML 34
Examples 34
Minimizza HTML usando gulp-htmlmin 34

Capitolo 13: Mostra errori con gulp-jslint 35
Examples 35
Installazione e utilizzo 35
Capitolo 14: Utilizzando Browserify 36
Parametri 36
Examples 36
Utilizzando Browserify con Vanilla Javascript 36
Usando Browserify con Coffeescript 36
Capitolo 15: Utilizzo dei filtri di file. 38
Examples 38
Creazione di regole Images, JS e CSS (SASS) per ordinare i file 38
Installazione di Gulp e dei suoi compiti 38
Determinazione della struttura delle cartelle 38
Preprocessing di Gulp 38
Gulp Task 38
Gulp Watch 39
Titoli di coda 40

Di

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.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

Capitolo 1: Iniziare con Gulp

Osservazioni

Gulp è un sistema di compilazione JavaScript, un task runner basato su node.js come Grunt. Ti consente di automatizzare attività comuni durante il tuo processo di sviluppo. Gulp utilizza gli stream -concept e code-over-configuration per un processo di compilazione più semplice e intuitivo. Il concetto di code-over-configuration consente di creare attività più leggibili e semplici, mentre i task grunt sono altamente sovra-configurati.

Versioni

Versione	Data di rilascio
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

Installazione o configurazione

1. Installa Node.js e NPM:

Gulp richiede Node.js e NPM, il gestore di pacchetti di Node. La maggior parte degli installatori include NPM con Node.js. Fare riferimento alla documentazione di installazione o confermare che sia già installato eseguendo il seguente comando nel terminale,

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

2. Installa gulp a livello globale:

Se hai precedentemente installato una versione di gulp a livello globale, esegui `npm rm --global gulp` per assicurarti che la tua vecchia versione non `npm rm --global gulp` in collisione con **gulp-cli**

.

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

3. Inizializza la directory del tuo progetto:

```
$ npm init
```

4. Installa gulp nel tuo progetto devDependencies:

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

5. Crea un gulpfile.js nella radice del tuo progetto:

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

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

6. Esegui gulp:

```
$ gulp
```

L'attività predefinita verrà eseguita e non farà nulla.

Per eseguire singole attività, utilizzare `gulp <task> <othertask>` .

Dipendenza delle attività

È possibile eseguire attività in serie, passando un secondo parametro a `gulp.task()` .

Questo parametro è una serie di attività da eseguire e completare prima dell'esecuzione dell'attività:

```
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
});
```

È inoltre possibile omettere la funzione se si desidera eseguire solo un pacchetto di attività di dipendenza:

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

Nota: le attività verranno eseguite in parallelo (tutte in una volta), quindi non dare per scontato che le attività vengano avviate / completate in ordine. Avviare gulp v4 , `gulp.series()` dovrebbe essere usato se l'ordine di esecuzione delle attività di dipendenza è importante.

Concat js file nella sottocartella 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'));
});
```

deglutire i documenti CLI

bandiere

Gulp ha pochissime bandiere da sapere. Tutti gli altri flag servono per le attività, se necessario.

- `-v` o `--version` visualizzerà le versioni globali e locali del gulp
- `--require <module path>` richiederà un modulo prima di eseguire il file gulp. Questo è utile per i transporter ma ha anche altre applicazioni. È possibile utilizzare più flag - `--require`
- `--gulpfile <gulpfile path>` imposterà manualmente il percorso di gulpfile. Utile se hai più gulpfiles. Questo imposterà anche il CWD nella directory gulpfile
- `--cwd <dir path>` imposterà manualmente il CWD. La ricerca del gulpfile e della relatività di

tutti richiederà da qui
- `-T` o `--tasks` visualizzerà l'albero delle dipendenze dell'attività per il gulpfile caricato
- `--tasks-simple` mostrerà un elenco di compiti in chiaro per il gulpfile caricato
- `--color` forzerà i plugin gulp e gulp per visualizzare i colori anche quando non viene rilevato alcun supporto colore
- `--no-color` costringerà i plugin gulp e gulp a non visualizzare i colori anche quando viene rilevato il supporto del colore
- `--silent` disabiliterà la registrazione di gulp

La CLI aggiunge process.env.INIT_CWD, che è il cwd originale da cui è stato lanciato.

Bandiere specifiche dell'attività

Fare riferimento a questo collegamento StackOverflow per informazioni su come aggiungere contrassegni specifici dell'attività

Compiti

Le attività possono essere eseguite eseguendo `gulp <task> <othertask>` . Basta eseguire `gulp` per eseguire l'attività che hai registrato chiamata `default` . Se non ci sono task gulp `default` verificherà un errore.

I compilatori

Puoi trovare un elenco di lingue supportate da interpretare . Se si desidera aggiungere supporto per una nuova lingua, inviare richiesta pull / aprire i problemi lì.

Leggi Iniziare con Gulp online: https://riptutorial.com/it/gulp/topic/1341/iniziare-con-gulp

Capitolo 2: Compressione senza perdita di immagini (con gulp-imagemin)

Sintassi

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

Parametri

Discussione	Descrizione
`sourcePath`	Directory sorgente delle immagini (ad esempio: `/assets/images`)
`buildPath`	Percorso di destinazione (ad esempio: `/static/dist/`)

Osservazioni

Il primo argomento per `imagemin` constructor è l'array di plugin. Per impostazione predefinita, vengono utilizzati i seguenti plugin: `[imagemin.gifsicle(), imagemin.jpegtran(), imagemin.optipng(), imagemin.svgo()]`

Il secondo argomento sono le opzioni. Nell'esempio sopra riportato vengono utilizzate le seguenti opzioni:

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

Quelli sono completamente opzionali.

`progressive` è usato da `imagemin-jpegtran` .

`interlaced` è usato da `imagemin-gifsicle` .

`removeUnknownsAndDefaults` e `cleanupIDs` sono usati da `imagemin-svgo` .

Examples

Installazione e utilizzo

Installazione delle dipendenze (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/
});
```

Leggi Compressione senza perdita di immagini (con gulp-imagemin) online: https://riptutorial.com/it/gulp/topic/6549/compressione-senza-perdita-di-immagini--con-gulp-imagemin-

Capitolo 3: Concatenazione di file

Examples

Concat tutti i file CSS in uno utilizzando gulp-concat

Innanzitutto, installa `gulp` e il plugin `gulp-concat` sul tuo progetto localy

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

e aggiungi l'attività gulp-concat al tuo 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']);
```

Dopo aver avviato il comando `gulp` , plugin gulp-concat prenderà tutti i file CSS presenti nella directory `css/` e li concatenerà in un unico file `css/dist/concat.css`

Concat e Uglify JS e CSS

Ricordarsi di npm installare prima tutti i file in devDependencies. Per esempio

```
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
});
```

Eseguili seguendo i comandi

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

Leggi Concatenazione di file online: https://riptutorial.com/it/gulp/topic/4398/concatenazione-di-file

Capitolo 4: Crea documentazione con gulp-jsdoc3

Examples

Installazione

Prima di tutto, installa `gulp` e `gulp-jsdoc3` nel tuo progetto:

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

Quindi aggiungilo al tuo gulpfile.js

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



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

Per documentare, ad esempio, una funzione, devi aggiungere un commento nella parte superiore della funzione, in questo modo:

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

Se vuoi sapere più tag di blocco da utilizzare, visita usejsdoc.org

Leggi Crea documentazione con gulp-jsdoc3 online: https://riptutorial.com/it/gulp/topic/7365/crea-documentazione-con-gulp-jsdoc3

Capitolo 5: Crea un osservatore

Examples

Compito di osservatore

`config.paths.html` rappresenta il percorso del tuo file HTML.

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

L'attività dovrebbe essere aggiunta anche a default:

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

Leggi Crea un osservatore online: https://riptutorial.com/it/gulp/topic/5026/crea-un-osservatore

Capitolo 6: Elimina i file usando Gulp

Osservazioni

Nota sull'uso del modello di globbing (`**`):

Il modello globbing corrisponde a tutti i `children` **e** al `parent` . Per evitare che aggiungiamo `'!public'` alla nostra attività del modo che la stessa directory `public` non venga cancellata

Examples

Elimina i file usando del

Prima di tutto, installa `gulp` e `del` per localizzare la directory `del` progetto

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

Quindi aggiungi l'attività `clean` al tuo `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']);
```

Questa attività cancella tutti i file nella directory pubblica

L'attività nel codice viene aggiunta come dipendenza per l'attività `'default'` modo che ogni volta che verrà eseguita l' `default` , `clean` verrà eseguito prima di esso.

Puoi anche chiamare manualmente l'operazione `clean` eseguendo il comando:

```
gulp clean
```

Leggi Elimina i file usando Gulp online: https://riptutorial.com/it/gulp/topic/7189/elimina-i-file-usando-gulp

Capitolo 7: Guida completa a un flusso di lavoro front-end con Gulpjs 2 di 2

Osservazioni

Freddo. Quindi abbiamo finito con l'automazione del nostro flusso di lavoro.

Ora abbiamo un file gulp, quello

- Responsive e minimizza le immagini
- pulisce, autoprefixes, concatena e riduce Css
- Concatena e minimizza JS
- Guarda le modifiche alle risorse sia HTML che CSS | JS e attivano i task associati
- Crea una directory di build e memorizza tutti i codici pronti per l'implementazione elaborati al suo interno. E tutto ciò, sullo sfondo, mentre sviluppi la tua app.

Riferimenti

Esegui-Sequenza di sincronizzazione del browser

Examples

Impostazione della sincronizzazione del browser e configurazione degli osservatori per stile e script

NOTA

> **Questa pagina illustra l'uso di plug-in gulp come browser-sync, gulp-watch e sequenza di esecuzione e continua a discutere di gulp-workflow-automation da dove abbiamo interrotto Gulpjs-workflow-automation-1 di 2. Se sei atterrato qui, considera prima di** passare per quel post.

- Task predefinito
- Attività di watchdog: per costruire continuamente le risorse pronte per l'implementazione al volo, ogni volta che si tratta di un'immagine | JS | cambiamenti di css nel corso dello sviluppo.

Cominciamo con la sincronizzazione del browser.

Gulp Watchdog Task

Cominciamo con il compito di watchdog.

L'obiettivo è quello di osservare le modifiche che apporti durante lo sviluppo. Qualsiasi cambiamento, dovrebbe innescare il compito gulp corrispondente.

Inoltre, abbiamo bisogno di una funzionalità che sincronizzi le modifiche sul browser.

Sincronizzazione del browser

Quindi, dobbiamo installare Browser Sync.

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

Con questa premessa, apriamo il nostro gulpfile.js e aggiungiamo la funzionalità dell'orologio. Cerchiamo di richiedere la sincronizzazione del browser e definire alcune variabili per utilizzare la sua funzionalità.

Nella parte superiore del file gulp, aggiungi lo snippet di sotto. Mettilo appena sotto le dichiarazioni di compressione dell'immagine.

così:

```
//Browser-sync

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

La sincronizzazione del browser per sincronizzare lo sviluppo con il browser è una semplice configurazione. Cerchiamo di creare un'attività chiamata watchdog.

così:

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


})
```

Ora, se navighiamo attraverso le opzioni di sincronizzazione del browser qui e cerchiamo l'impostazione del server, possiamo vedere quanto sia facile.

Abbiamo solo bisogno di inserire il sotto all'interno del nostro compito di watchdog

Snippet - 1 - all'interno del boiler watchdog

```
/*
 Initiate Browser sync
 @documentation - https://www.browsersync.io/docs/options/
 */
 sync.init({
  server: {
    baseDir: "./"
```

```
},
port: 8000 //change it as required
});
```

Inserisci qui sopra il boilerplate del tuo watchdog sopra.

Il snippet successivo consiste nel definire un watcher per gli stili, con l'obiettivo di rielaborare i file css modificati o nuovi e attivare automaticamente il ricaricamento del browser.

snippet - 2 - all'interno del boiler watchdog

```
$.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')
});
```

Inserisci qui sopra il boilerplate del tuo watchdog sopra.

Quindi stiamo monitorando `" [fonts/google/**/*.css , /**/*.css ]"` cioè,

tutti i file css sotto css / tutti i file css sotto fonts / google / Quando qualcosa cambia, o viene aggiunto un nuovo file, si innesca il metodo di ricarica, che è definito nella parte superiore del nostro gulpfile, nella dichiarazione browsersync.

> Nota: Si potrebbe notare, che abbiamo un gestore di eventi **.on** attaccato al watcher.

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

Fondamentalmente, qualsiasi CUD (Crea | Aggiorna | Elimina) attiva la funzione di ricarica e passa un oggetto evento come parametro alla funzione di callback.

Il callback è una funzione vitale, dove possiamo ottenere operazioni come uncache sulla cancellazione degli asset. Ora l'oggetto evento ha parametri simili

- sentiero
- tipo - Crea / Aggiorna / Elimina

Se una risorsa viene eliminata, dobbiamo assicurarci che le cache che abbiamo creato nelle nostre precedenti funzioni di minificazione, tramite gulp-cache e gulp-remember, abbiano bisogno di aggiornamento.

lo gestiamo nel frammento di seguito, che è all'interno del callback sul cambiamento.

```
if (event.type === 'deleted') {
uncache('styles', event.path);
```

```
$$.remember.forget('auto-prefixed-stylesheets', event.path);
}
```

Nota

$ -> alias per gulp

$$ -> alias per gulp-load-plugins

potreste anche notare che ho una `sequence('optimizeStyles');` dopo che ho scritto l'invocazione uncache

Il metodo di sequenza garantisce che il metodo sincrono venga eseguito in modo asincrono per impostazione predefinita javascript.

installarlo è semplice

FARE

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

quindi, dichiaralo nel tuo gulpfile appena sotto la dichiarazione di sincronizzazione del browser.

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

Quindi con questa comprensione, l'osservatore delle sceneggiature è facile da scrivere. solo diversi globi!

Quindi, aggiungi questo snippet sotto l'osservatore di stile all'interno del boilerdog del watchdog.

Snippet - 3 - inside boilerplate task 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

Abbiamo utilizzato due funzioni nei nostri frammenti sopra.

- rimozione dati dalla cache
- $$. remember.forget Nota:

$ -> Alias per gulp

$$ -> Alias per gulp-load-plugins

Definiamo la funzione uncache da qualche parte nel nostro gulpfile.js, prima che venga invocato.

```
/*
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)
}
```

Definizione di un'attività predefinita

Quindi ora, completiamo il codice gulpfile, definendo un'attività predefinita.

l'attività predefinita è quella che viene eseguita, quando si dice

```
gulp
```

su un prompt dei comandi sotto la radice del tuo progetto.

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

Leggi Guida completa a un flusso di lavoro front-end con Gulpjs 2 di 2 online: https://riptutorial.com/it/gulp/topic/6343/guida-completa-a-un-flusso-di-lavoro-front-end-con-gulpjs-2-di-2

Capitolo 8: Guida completa all'automazione del flusso di lavoro front-end con Gulpjs -1 di 2

Sintassi

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

Osservazioni

Continua a leggere: creazione di un'attività predefinita e impostazione della sincronizzazione del browser

Riferimenti

- Gulp-Clean-Css
- Gulp-Uglify
- Gulp-Autoprefixer - per Css Prefissi che mantengono la compatibilità del browser
- Gulp-Copia cache
- Gulp-Remember
- Gulp-Jshint

Guida passo passo a Gulp Workflow Automation per principianti assoluti che ho documentato nel mio blog

Examples

Caricamento di tutti i plugin da Package.JSON

Supponendo che tu abbia una conoscenza di come installare gulp, lasciaci andare fino a richiedere tutte le dipendenze di gulp da package.json nella cartella principale dei tuoi progetti.

Nel caso in cui non si abbia ancora un file gulp, si prega di creare un file vuoto con il nome

gulpfile.js

Per prima cosa, abbiamo bisogno di un gulp. così:

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

Successivamente, carichiamo tutti i nostri plugin, nel nostro file gulp, dal frammento di seguito

Nota

Leggi i commenti in tutti i frammenti che includeremo in questa lettura, forniscono più informazioni su ogni funzionalità

```
/*
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

Attraverso tutti i molti esempi che avremo nell'articolo, noi alias

1. deglutire come $ e
2. gulp-load-plugins come $$

puramente per facilità di battitura.

Installazione di plugin per immagini reattive | Css Minification | Js minification

Il nostro obiettivo è quello di

1. Rendi le tue immagini conformi alle larghezze e alla scala in modo appropriato, generando una batteria di immagini di larghezze diverse e per ridimensionarle
2. Lascia il tuo javascript
3. Riduci al minimo le risorse: JS / CSS / HTML, consentendo così di ospitare un codice più leggero di quello più leggero
4. Guarda i file css / js / image per il cambiamento e ricostruisci le ottimizzazioni
5. Sincronizzazione delle modifiche durante lo sviluppo, su un browser che serve il tuo sito.

Abbiamo bisogno di un numero di plugin, quindi installiamoli tutti. Si prega di eseguire tutti questi nella cartella principale del progetto.

Plugin per l'elaborazione delle immagini

```
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
```

Plugin ottimizzatore di asset

```
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
```

Anatomia di una funzione di sorso

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

$.src(glob)

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

}
```

Nota

pipe è un metodo che esegue lo streaming di tutti i file corrispondenti all'ingresso glob, ai nostri plug-in (minifyhtml in questo caso).

È semplice immaginarlo così:

$ .src è ciò che costruisce lo stream e pipe pipe fuori ogni singolo file che corrisponde al glob verso il basso per ogni plugin nella pipeline.Ogni plugin il file viene passato a, modifica il suo contenuto in memoria solo fino a $ .dest è stato raggiunto, che quindi aggiorna / crea file in streaming da $ .src

Dove ,

$ -> gulp

$$ -> gulp-load-plugins

Ottimizzazione e minimizzazione delle risorse

Quindi, prima di scrivere le funzioni di ottimizzazione, è necessario installare un paio di plug-in di memorizzazione nella cache.

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

Potresti chiederti perché due cache eh !. **gulp-cache** , passa solo il contenuto modificato o nuovo nella pipeline ad altri plugin. Quindi, dato che vogliamo che i file senza modifiche vengano utilizzati per concatenare in un singolo file per risorsa (css | js), abbiamo bisogno di **gulp-remember** oltre a gulp-cache

Per prima cosa usiamo **gulp-cache** per creare un elenco di file che sono stati modificati

In secondo luogo, abbiamo bisogno di **gulp-ricordarsi** di tenere traccia di tutti i file che sono passati attraverso quella lista in memoria.

Prima esecuzione: nessun file viene memorizzato nella cache, quindi gulp-cache li passerà tutti a gulp-remember

Esecuzioni successive: solo i file modificati o nuovi vengono reindirizzati da gulp-cache. Poiché il contenuto del file corrente è cambiato, gulp-remember aggiorna la sua cache.

Cool, scriviamo il nostro primo ottimizzatore

Style Optimizer

```
// 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

$ -> gulp

$$ -> gulp-load-plugins

$ .src -> crea flussi di file corrispondenti al glob passato come src

$ .dest -> salva il file manipolato nel percorso specificato

Script Optimiser

```
// 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

$ -> gulp

$$ -> gulp-load-plugins

$ .src -> crea flussi di file corrispondenti al glob passato come src

$ .dest -> salva il file manipolato nel percorso specificato

Generare immagini reattive

Passiamo ora all'elaborazione delle immagini. Quindi, lo scopo è avere una serie di dimensioni per ogni immagine che stai per servire.

Perché?

> Bene, per capire perché abbiamo bisogno di una batteria di immagini con una gamma di larghezze, dobbiamo riflettere sul fatto che probabilmente ci sono milioni di dispositivi con risoluzioni diverse. Abbiamo bisogno di un'immagine da ridimensionare senza molti pixel. Allo stesso tempo, dobbiamo migliorare i tempi di caricamento della pagina, scaricando solo l'immagine, che si adatta alla larghezza in cui è contenuta, ed è anche con la dimensione più piccola possibile, per farlo. Ci sono blog accademici come quello scritto da Eric Portis, che mette in evidenza l'inefficacia delle sole domande dei media e funge da guida completa alla comprensione di concetti come srcset e dimensioni.

Puoi riferirti all'epica scrittura di Eric Portis qui

Ora, la nostra funzione, ha bisogno di prendere un glob e una larghezza come input, e fare la sua magia e spingere il file ogni corsa genera, verso una destinazione e minimizzare l'immagine responsiva.

Abbiamo installato due plug- in di compressione delle immagini nel nostro primo esempio

Dato che questi plugin **NON** iniziano con un "gulp-" prefissato, dobbiamo caricarli manualmente sul nostro gulpfile.

SO Richiediamoli manualmente, dopo la dichiarazione gulp-load-plugins nella parte superiore del nostro gulpfile.

così:

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

Vale la pena notare che gulp-responsive viene fornito con il nitido processore di immagini, che è migliore di imagemagick di FAAAAR !. Sharp è ciò che viene utilizzato da gulp-responsive per ritagliare le immagini alle larghezze desiderate.

potresti dare un'occhiata a gulp-responsive-configuration-options, per un elenco completo di parametri di configurazione. Ho solo usato

- larghezza - per ritagliare le nostre immagini ad una larghezza w, passata come parametro
- rinomina - per aggiungere un suffisso al nome dell'immagine, in modo che rimanga univoco

nella mia funzione di configurazione di seguito. quindi la nostra funzione taglierà l'immagine alla larghezza passata come input, per tutte le immagini corrispondenti decifrate tramite l'input glob. quindi, ogni immagine viene compressa usando jpeg-recompress o pngquant e salvata all'interno di build / images.

Con questo in mente, la nostra funzione sarebbe così:

```
/*
@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

$ -> gulp

$$ -> gulp-load-plugins

$ .src -> crea flussi di file corrispondenti al glob passato come src

$ .dest -> salva il file manipolato nel percorso specificato

Ulteriori riferimenti

- Gulp-Responsive
- Gulp-Imagemin

HTML Minifier

```
*
 *@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;
        }));
}
```

Anatomia di un compito ingannevole

L'anatomia di una definizione di attività è così:

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

dipendenze, è una serie di compiti che devono finire prima del compito corrente che si sta definendo, corre. Più simile a forzare un'esecuzione sincrona invece della funzionalità Asincrona predefinita.

Aggiunta di attività Gulp

Quindi, ora abbiamo

- Una funzione definita Sopra per ottimizzare gli stili
- Una funzione definita Sopra per ottimizzare gli script
- Una funzione definita Sopra per ottimizzare l'HTML
- Una funzione per generare più immagini per immagine Sopra

Tutto ciò che dobbiamo fare ora è invocarli quando necessario.

Lascia che scriviamo i nostri compiti in base alla sintassi che abbiamo definito in precedenza

```
/*
* $.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();

});
```

Leggi Guida completa all'automazione del flusso di lavoro front-end con Gulpjs -1 di 2 online: https://riptutorial.com/it/gulp/topic/6342/guida-completa-all-automazione-del-flusso-di-lavoro-front-end-con-gulpjs--1-di-2

Capitolo 9: Gulp Path

Examples

Creazione di Simple Gulp Path per l'app

Prima di eseguire gulp devi installare `Node.JS` e `npm`

Il capo di Gulp.js è:

```
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
```

Quindi assegna il percorso dei file gulp

```
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'],
};
```

Esempio 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'));
    });
```

Esempio 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/'));
});
```

Orologio predefinito

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

Leggi Gulp Path online: https://riptutorial.com/it/gulp/topic/7759/gulp-path

Capitolo 10: Minificando JS

Sintassi

- `ext` Un oggetto che specifica l'origine di output e le estensioni di file miniate.
- `source` La stringa del suffisso dei nomi file che termina con i file di origine.
- `min` Quando stringa: la stringa del suffisso dei nomi file che termina con i file minificati.
- Quando array: espressioni regex da sostituire con nomi file di input. Ad esempio: [/.(.*)-source.js$/, '$ 1.js']
- `exclude` Will non minimizza i file nelle directory.
- `noSource` Non `noSource` il codice sorgente nelle `noSource` .
- `ignoreFiles` Non minimizza i file che corrispondono al modello.
- `mangle` Passa `false` per saltare i nomi di mangling.
- `output` Passa un oggetto se desideri specificare ulteriori `output options` . I valori predefiniti sono ottimizzati per la migliore compressione.
- `compress` Passa un oggetto per specificare `compressor options` personalizzato. Passa falso per saltare completamente la compressione.
- `preserveComments` Un'opzione di convenienza per options.output.comments. Impostazioni predefinite per non conservare commenti.
- `all` Mantieni tutti i commenti nei blocchi di codice
- `some` Mantieni commenti che iniziano con un botto `(!)` o includono una direttiva Closure Compiler `(@preserve, @license, @cc_on)`
- `function` Specificare la propria funzione di conservazione dei commenti. Verrà passato il nodo corrente e il commento corrente e ci si aspetta che restituisca sia `true` che `false` .

Osservazioni

Link utili per gulp-minify

Examples

Minimizza JS usando gulp-minify

Per prima cosa, installa `gulp` e `gulp-minify` nella directory del progetto localmente

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

Quindi aggiungi la seguente attività `min-js` al tuo `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']);
```

Questa attività trova tutti i file js nella directory `lib` , minfy e salva nella directory `lib` con il suffisso `.min.js` . Ad esempio, dopo aver `lib/app.js` file `lib/app.js` verrà creato un file `lib/app.min.js`

Oltre a essere eseguito come dipendenza per l'attività `'default'` gulp `'default'` , questa attività può essere eseguita manualmente digitando il seguente comando:

```
gulp min-js
```

Leggi Minificando JS online: https://riptutorial.com/it/gulp/topic/4397/minificando-js

Capitolo 11: Minificazione CSS

Examples

Usando gulp-clean-css e gulp-rename

Innanzitutto, installa `gulp` , `gulp-clean-css` e `gulp-rename` nella directory di progetto localy

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

Aggiungere la seguente attività `gulpfile.js` `minify-css` al file `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']);
```

Qui `.pipe(cleanCSS())` esegue la minificazione del file `css/dist/dist.css` e `.pipe(rename('concat.min.css'))` rinomina in `dist.min.css`

Sass e Css - Pre-elaborazione con Gulp

Prima di avviare gulp, è necessario installare `node.js` e `npm` . Quindi installare 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
});
```

Orologio Gulp

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

Leggi Minificazione CSS online: https://riptutorial.com/it/gulp/topic/4396/minificazione-css

Capitolo 12: Minimizzazione dell'HTML

Examples

Minimizza HTML usando gulp-htmlmin

Innanzitutto, installa `gulp` e `gulp-htmlmin` per proiettare la directory localmente

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

Quindi aggiungi l'attività `gulpfile.js` `minify-html` al tuo `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']);
```

Questa attività trova tutti i file nella directory di `source` con un'estensione `.html` , li minimizza e quindi restituisce i file risultanti alla directory `public` .

L'attività nel codice viene aggiunta come dipendenza per l'attività `'default'` modo che ogni volta che verrà eseguita l' `default` , `minify-html` verrà eseguito prima di esso.

Puoi anche chiamare manualmente l'attività `minify-html` eseguendo il comando:

```
gulp minify-html
```

Leggi Minimizzazione dell'HTML online: https://riptutorial.com/it/gulp/topic/7187/minimizzazione-dell-html

Capitolo 13: Mostra errori con gulp-jslint

Examples

Installazione e utilizzo

Installazione

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

uso

In gulpfile.js aggiungi:

```
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' ));
});
```

per utilizzare questo compito:

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

Leggi Mostra errori con gulp-jslint online: https://riptutorial.com/it/gulp/topic/7415/mostra-errori-con-gulp-jslint

Capitolo 14: Utilizzando Browserify

Parametri

Opzioni	Dettagli
trasformare	Specifica una pipeline di funzioni (o nomi di moduli) attraverso cui verrà eseguito il pacchetto browserizzato.
mettere a punto	Abilita il supporto della mappa di origine. `!gulp.env.production` funzionerebbe bene.
estensioni	Serie di estensioni che si desidera ignorare in `require()` oltre a .js e .json. Non dimenticare `.`
ignorare	Matrice di percorsi che dovrebbero essere passati alla funzione ignora di browserify.
risolvere	Funzione di risoluzione del nome del modulo personalizzato.
nobuiltins	Rimuovi i moduli incorporati definiti in `lib/builtins.js` (modulo browserify). `opts.builtins` deve essere definito e `opts.nobuiltins` può essere una matrice di stringhe o semplicemente una stringa.

Examples

Utilizzando Browserify con Vanilla Javascript

Prima installa gulp e browserify via `npm i gulp gulp-browserify` . Questo installerà browserify nella cartella `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

Prima installa gulp e browserify via npm i gulp gulp-coffeeify. Questo installerà browserify nella

cartella 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/'));
}
```

Leggi Utilizzando Browserify online: https://riptutorial.com/it/gulp/topic/6364/utilizzando-browserify

Capitolo 15: Utilizzo dei filtri di file.

Examples

Creazione di regole Images, JS e CSS (SASS) per ordinare i file

Installazione di Gulp e dei suoi compiti

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

Determinazione della struttura delle cartelle

In questa struttura, utilizzeremo la cartella dell'app per scopi di sviluppo, mentre la cartella dist viene utilizzata per contenere file ottimizzati per il sito di produzione.

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

Preprocessing di 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() {});
```

Leggi Utilizzo dei filtri di file. online: https://riptutorial.com/it/gulp/topic/7818/utilizzo-dei-filtri-di-file-

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Gulp	cl3m, Community, fracz, João Silva, naresh, Nhan, Pejman, Pierre-Loup Pagniez, Pyloid, Sender, Tim
2	Compressione senza perdita di immagini (con gulp-imagemin)	Siegmeyer
3	Concatenazione di file	Andrew Plakhotnyi, fracz, John Strood, maoizm, Mikhail, noob
4	Crea documentazione con gulp-jsdoc3	Manuel
5	Crea un osservatore	Muaaz Rafi, Nhan
6	Elimina i file usando Gulp	Mor Paz
7	Guida completa a un flusso di lavoro front-end con Gulpjs 2 di 2	Schrodinger's cat
8	Guida completa all'automazione del flusso di lavoro front-end con Gulpjs -1 di 2	Schrodinger's cat
9	Gulp Path	GYTO
10	Minificando JS	fracz, GYTO, Mikhail, Mor Paz, Saiful Azad
11	Minificazione CSS	fracz, GYTO, Mikhail, SteveLacy
12	Minimizzazione dell'HTML	GYTO, Mor Paz
13	Mostra errori con gulp-jslint	jesussegado
14	Utilizzando Browserify	dome2k

15	Utilizzo dei filtri di file.	GYTO