无双
博学善思，自强不息

前端构建工具**gulpjs**的使用介绍及技巧

gulpjs是一个前端构建工具，与gruntjs相比，gulpjs无需写一大堆繁杂的配置参数，API也非常简单，学习起来很容易，而且gulpjs使用的是nodejs中stream来读取和操作数据，其速度更快。如果你还没有使用过前端构建工具，或者觉得gruntjs太难用的话，那就尝试一下gulp吧。

本文导航：

1、gulp的安装

首先确保你已经正确安装了nodejs环境。然后以全局方式安装gulp：

```
npm install -g gulp
```

全局安装gulp后，还需要在每个要使用gulp的项目中都单独安装一次。把目录切换到你的项目文件夹中，然后在命令行中执行：

```
npm install gulp
```

如果想在安装的时候把gulp写进项目package.json文件的依赖中，则可以加上--save-dev：

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

这样就完成了gulp的安装。至于为什么在全局安装gulp后，还需要在项目中本地安装一次，有兴趣的可以看下stackoverflow上有人做出的回答：why-do-we-need-to-install-gulp-globally-and-locally、what-is-the-point-of-double-install-in-gulp。大体就是为了版本的灵活性，但如果没理解那也不必太去纠结这个问题，只需要知道通常我们是要这样做就行了。

2、开始使用gulp

2.1 建立gulpfile.js文件

就像gruntjs需要一个 `Gruntfile.js` 文件一样，gulp也需要一个文件作为它的主文件，在gulp中这个文件叫做 `gulpfile.js` 。新建一个文件名为 `gulpfile.js` 的文件，然后放到你的项目目录中。之后要做的事情就是在 `gulpfile.js` 文件中定义我们的任务了。下面是一个最简单的 `gulpfile.js` 文件内容示例，它定义了一个默认的任务。

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

此时我们的目录结构是这样子的：

```
├── gulpfile.js
├── node_modules
│   └── gulp
└── package.json
```

2.2 运行gulp任务

要运行gulp任务，只需切换到存放 `gulpfile.js` 文件的目录(windows平台请使用cmd或者Power Shell等工具)，然后在命令行中执行 `gulp` 命令就行了， `gulp` 后面可以加上要执行的任务名，例如 `gulp task1` ，如果没有指定任务名，则会执行任务名为 `default` 的默认任务。

3、gulp的API介绍

使用gulp，仅需知道4个API即可： `gulp.task()` ， `gulp.src()` ， `gulp.dest()` ， `gulp.watch()` ，所以很容易就能掌握，但有几个地方需理解透彻才行，我会在下面一一说明。为了避免出现理解偏差，建议先看一遍官方文档。

3.1 gulp.src()

在介绍这个API之前我们首先来说一下Grunt.js和Gulp.js工作方式的一个区别。Grunt主要是以文件为媒介来运行它的工作流的，比如在Grunt中执行完一项任务后，会把结果写入到一个临时文件中，然后可以在这个临时文件内容的基础上执行其它任务，执行完成后又把结果写入到临时文件中，然后又以这个为基础继续执行其它任务...就这样反复下去。而在Gulp中，使用的是Nodejs中的stream(流)，首先获取到需要的stream，然后可以通过stream的 `pipe()` 方法把流导入到你想要的地方，比如Gulp的插件中，经过插件处理后的流又可以继续导入到其他插件中，当然也可以把流写入到文件中。所以Gulp是以stream为媒介的，它不需要频繁的生成临时文件，这也是Gulp的速度比Grunt快的一个原因。再回到正题上来， `gulp.src()` 方法正是用来获取流的，但要注意这个流里的内容不是原始的文件流，而是一个虚拟文件对象流(Vinyl files)，这个虚拟文件对象中存储着原始文件的路径、文件名、内容等信息，这个我们暂时不用去深入理解，你只需简单的理解可以用这个方法来读取你需要操作的文件就行了。其语法为：

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

globs参数是文件匹配模式(类似正则表达式)，用来匹配文件路径(包括文件名)，当然这里也可以直接指定某个具体的文件路径。当有多个匹配

模式时，该参数可以为一个数组。
options为可选参数。通常情况下我们不需要用到。

下面我们重点说说Gulp用到的glob的匹配规则以及一些文件匹配技巧。Gulp内部使用了node-glob模块来实现其文件匹配功能。我们可以使用下面这些特殊的字符来匹配我们想要的文件：

- `*` 匹配文件路径中的0个或多个字符，但不会匹配路径分隔符，除非路径分隔符出现在末尾
- `**` 匹配路径中的0个或多个目录及其子目录,需要单独出现，即它左右不能有其他东西了。如果出现在末尾，也能匹配文件。
- `?` 匹配文件路径中的一个字符(不会匹配路径分隔符)
- `[...]` 匹配方括号中出现的字符中的任意一个，当方括号中第一个字符为 `^` 或 `!` 时，则表示不匹配方括号中出现的其他字符中的任意一个，类似js正则表达式中的用法
- `!(pattern|pattern|pattern)` 匹配任何与括号中给定的任一模式都不匹配的
- `?(pattern|pattern|pattern)` 匹配括号中给定的任一模式0次或1次，类似于js正则中的(pattern|pattern|pattern)?
- `+(pattern|pattern|pattern)` 匹配括号中给定的任一模式至少1次，类似于js正则中的(pattern|pattern|pattern)+
- `*(pattern|pattern|pattern)` 匹配括号中给定的任一模式0次或多次，类似于js正则中的(pattern|pattern|pattern)*
- `@(pattern|pattern|pattern)` 匹配括号中给定的任一模式1次，类似于js正则中的(pattern|pattern|pattern)

下面以一系列例子来加深理解

- `*` 能匹配 `a.js` , `x.y` , `abc` , `abc/` ,但不能匹配 `a/b.js`
- `*.*` 能匹配 `a.js` , `style.css` , `a.b` , `x.y`
- `*/*/*.js` 能匹配 `a/b/c.js` , `x/y/z.js` ,不能匹配 `a/b.js` , `a/b/c/d.js`
- `**` 能匹配 `abc` , `a/b.js` , `a/b/c.js` , `x/y/z` , `x/y/z/a.b` ,能用来匹配所有的目录和文件
- `**/*.js` 能匹配 `foo.js` , `a/foo.js` , `a/b/foo.js` , `a/b/c/foo.js`
- `a/**/z` 能匹配 `a/z` , `a/b/z` , `a/b/c/z` , `a/d/g/h/j/k/z`
- `a/**b/z` 能匹配 `a/b/z` , `a/sb/z` ,但不能匹配 `a/x/sb/z` ,因为只有单 `**` 单独出现才能匹配多级目录
- `?.js` 能匹配 `a.js` , `b.js` , `c.js`
- `a??` 能匹配 `a.b` , `abc` ,但不能匹配 `ab/` ,因为它不会匹配路径分隔符
- `[xyz].js` 只能匹配 `x.js` , `y.js` , `z.js` ,不会匹配 `xy.js` , `xyz.js` 等,整个中括号只代表一个字符
- `[^xyz].js` 能匹配 `a.js` , `b.js` , `c.js` 等,不能匹配 `x.js` , `y.js` , `z.js`

当有多种匹配模式时可以使用数组

```
//使用数组的方式来匹配多种文件
gulp.src(['js/*.js','css/*.css','*.html'])
```

使用数组的方式还有一个好处就是可以很方便的使用排除模式，在数组中的单个匹配模式前加上 `!` 即是排除模式，它会在匹配的结果中排除这个匹配，要注意一点的是不能在数组中的第一个元素中使用排除模式

```
gulp.src([*.js,'!b*.js']) //匹配所有js文件，但排除掉以b开头的js文件
gulp.src(['!b*.js',*.js]) //不会排除任何文件，因为排除模式不能出现在数组的第一个元素中
```

此外，还可以使用展开模式。展开模式以花括号作为定界符，根据它里面的内容，会展开为多个模式，最后匹配的结果为所有展开的模式相加起来得到的结果。展开的例子如下：

- `a{b,c}d` 会展开为 `abd` , `acd`
- `a{b,}c` 会展开为 `abc` , `ac`
- `a{0..3}d` 会展开为 `a0d` , `a1d` , `a2d` , `a3d`
- `a{b,c{d,e}f}g` 会展开为 `abg` , `acdfg` , `acefg`
- `a{b,c}d{e,f}g` 会展开为 `abdeg` , `acdeg` , `abdeg` , `abdfg`

3.2 gulp.dest()

gulp.dest()方法是用来写文件的，其语法为：

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

path为写入文件的路径
options为一个可选的参数对象，通常我们不需要用到

要想使用好 `gulp.dest()` 这个方法，就要理解给它传入的路径参数与最终生成的文件的关系。
gulp的使用流程一般是这样子的：首先通过 `gulp.src()` 方法获取到我们想要处理的文件流，然后把文件流通过pipe方法导入到gulp的插件中，最后把经过插件处理后的流再通过pipe方法导入到 `gulp.dest()` 中， `gulp.dest()` 方法则把流中的内容写入到文件中，这里首先需要弄清楚的一点是，我们给 `gulp.dest()` 传入的路径参数，只能用来指定要生成的文件的目录，而不能指定生成文件的文件名，它生成文件的文件名使用的是导入到它的文件流自身的文件名，所以生成的文件名是由导入到它的文件流决定的，即使我们给它传入一个带有文件名的路径参数，然后它也会把这个文件名当做是目录名，例如：

```
var gulp = require('gulp');
gulp.src('script/jquery.js')
    .pipe(gulp.dest('dist/foo.js'));
//最终生成的文件路径为 dist/foo.js/jquery.js,而不是dist/foo.js
```

要想改变文件名，可以使用插件gulp-rename

下面说说生成的文件路径与我们给 `gulp.dest()` 方法传入的路径参数之间的关系。

`gulp.dest(path)` 生成的文件路径是我们传入的path参数后面再加上 `gulp.src()` 中有通配符开始出现的那部分路径。例如：

```
var gulp = reruire('gulp');
//有通配符开始出现的那部分路径为 **/*.js
gulp.src('script/**/*.js')
    .pipe(gulp.dest('dist')); //最后生成的文件路径为 dist/**/*.js
//如果 **/*.js 匹配到的文件为 jquery/jquery.js ,则生成的文件路径为 dist/jquery/jquery.js
```

再举更多一点的例子

```
gulp.src('script/avalon/avalon.js') //没有通配符出现的情况
    .pipe(gulp.dest('dist')); //最后生成的文件路径为 dist/avalon.js

//有通配符开始出现的那部分路径为 **/underscore.js
gulp.src('script/**/underscore.js')
    //假设匹配到的文件为script/util/underscore.js
    .pipe(gulp.dest('dist')); //则最后生成的文件路径为 dist/util/underscore.js

gulp.src('script/*') //有通配符出现的那部分路径为 *
    //假设匹配到的文件为script/zepto.js
    .pipe(gulp.dest('dist')); //则最后生成的文件路径为 dist/zepto.js
```

通过指定 `gulp.src()` 方法配置参数中的 `base` 属性，我们可以更灵活的来改变 `gulp.dest()` 生成的文件路径。

当我们没有在 `gulp.src()` 方法中配置 `base` 属性时， `base` 的默认值为通配符开始出现之前那部分路径，例如：

```
gulp.src('app/src/**/*.css') //此时base的值为 app/src
```

上面我们说的 `gulp.dest()` 所生成的文件路径的规则，其实也可以理解成，用我们给 `gulp.dest()` 传入的路径替换掉 `gulp.src()` 中的 `base` 路径，最终得到生成文件的路径。

```
gulp.src('app/src/**/*.css') //此时base的值为app/src,也就是说它的base路径为app/src
     //设该模式匹配到了文件 app/src/css/normal.css
    .pipe(gulp.dest('dist')) //用dist替换掉base路径，最终得到 dist/css/normal.css
```

所以改变base路径后， `gulp.dest()` 生成的文件路径也会改变

```
gulp.src(script/lib/*.js) //没有配置base参数，此时默认的base路径为 script/lib
    //假设匹配到的文件为script/lib/jquery.js
    .pipe(gulp.dest('build')) //生成的文件路径为 build/jquery.js
```

```
gulp.src(script/lib/*.js, {base:'script'}) //配置了base参数，此时base路径为script
    //假设匹配到的文件为script/lib/jquery.js
    .pipe(gulp.dest('build')) //此时生成的文件路径为build/lib/jquery.js
```

用 `gulp.dest()` 把文件流写入文件后，文件流仍然可以继续使用。

3.3 gulp.task()

`gulp.task` 方法用来定义任务，内部使用的是Orchestrator，其语法为：

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

name 为任务名
deps 是当前定义的任务需要依赖的其他任务，为一个数组。当前定义的任务会在所有依赖的任务执行完毕后才开始执行。如果没有依赖，则可省略这个参数
fn 为任务函数，我们把任务要执行的代码都写在里面。该参数也是可选的。

```
gulp.task('mytask', ['array', 'of', 'task', 'names'], function() { //定义一个有依赖的任务
  // Do something
});
```

`gulp.task()` 这个API没什么好讲的，但需要知道执行多个任务时怎么来控制任务执行的顺序。
gulp中执行多个任务，可以通过任务依赖来实现。例如我想要执行 `one` , `two` , `three` 这三个任务，那我们就可以定义一个空的任务，然后把那三个任务当做这个空的任务的依赖就行了：

```
//只要执行default任务，就相当于把one,two,three这三个任务执行了
gulp.task('default',['one','two','three']);
```

如果任务相互之间没有依赖，任务会按你书写的顺序来执行，如果有依赖的话则会先执行依赖的任务。
但是如果某个任务所依赖的任务是异步的，就要注意了，gulp并不会等待那个所依赖的异步任务完成，而是会接着执行后续的任务。例如：

```
gulp.task('one',function(){
  //one是一个异步执行的任务
  setTimeout(function(){
    console.log('one is done')
  },5000);
});

//two任务虽然依赖于one任务,但并不会等到one任务中的异步操作完成后再执行
gulp.task('two',['one'],function(){
  console.log('two is done');
```

```
});
```

上面的例子中我们执行two任务时，会先执行one任务，但不会去等待one任务中的异步操作完成后再执行two任务，而是紧接着执行two任务。所以two任务会在one任务中的异步操作完成之前就执行了。

那如果我们想等待异步任务中的异步操作完成后再执行后续的任务，该怎么做呢？
有三种方法可以实现：
第一：在异步操作完成后执行一个回调函数来通知gulp这个异步任务已经完成,这个回调函数就是任务函数的第一个参数。

```
gulp.task('one',function(cb){ //cb为任务函数提供的回调，用来通知任务已经完成
  //one是一个异步执行的任务
  setTimeout(function(){
    console.log('one is done');
    cb();  //执行回调，表示这个异步任务已经完成
  },5000);
});

//这时two任务会在one任务中的异步操作完成后再执行
gulp.task('two',['one'],function(){
  console.log('two is done');
});
```

第二：定义任务时返回一个流对象。适用于任务就是操作gulp.src获取到的流的情况。

```
gulp.task('one',function(cb){
  var stream = gulp.src('client/**/*.js')
      .pipe(dosomething()) //dosomething()中有某些异步操作
      .pipe(gulp.dest('build'));
    return stream;
});

gulp.task('two',['one'],function(){
  console.log('two is done');
});
```

第三：返回一个promise对象，例如

```
var Q = require('q'); //一个著名的异步处理的库 https://github.com/kriskowal/q
gulp.task('one',function(cb){
  var deferred = Q.defer();
  // 做一些异步操作
  setTimeout(function() {
     deferred.resolve();
  }, 5000);
  return deferred.promise;
});

gulp.task('two',['one'],function(){
```

```
  console.log('two is done');
});
```

`gulp.task()` 就这些了，主要是要知道当依赖是异步任务时的处理。

3.4 gulp.watch()

`gulp.watch()` 用来监视文件的变化，当文件发生变化后，我们可以利用它来执行相应的任务，例如文件压缩等。其语法为

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

glob 为要监视的文件匹配模式，规则和用法与 `gulp.src()` 方法中的 `glob` 相同。
opts 为一个可选的配置对象，通常不需要用到
tasks 为文件变化后要执行的任务，为一个数组

```
gulp.task('uglify',function(){
  //do something
});
gulp.task('reload',function(){
  //do something
});
gulp.watch('js/**/*.js', ['uglify','reload']);
```

`gulp.watch()` 还有另外一种使用方式：

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

glob和**opts**参数与第一种用法相同
cb参数为一个函数。每当监视的文件发生变化时，就会调用这个函数,并且会给它传入一个对象，该对象包含了文件变化的一些信息，`type` 属性为变化的类型，可以是 `added` , `changed` , `deleted` ； `path` 属性为发生变化的文件的路径

```
gulp.watch('js/**/*.js', function(event){
    console.log(event.type); //变化类型 added为新增,deleted为删除，changed为改变
    console.log(event.path); //变化的文件的路径
});
```

4、一些常用的gulp插件

gulp的插件数量虽然没有grunt那么多，但也可以说是应有尽有了，下面列举一些常用的插件。

4.1 自动加载插件

使用gulp-load-plugins
安装： `npm install --save-dev gulp-load-plugins`
要使用gulp的插件，首先得用 `require` 来把插件加载进来，如果我们

要使用的插件非常多，那我们的 `gulpfile.js` 文件开头可能就会是这个样子的：

```
var gulp = require('gulp'),
    //一些gulp插件,abcd这些命名只是用来举个例子
    a = require('gulp-a'),
    b = require('gulp-b'),
    c = require('gulp-c'),
    d = require('gulp-d'),
    e = require('gulp-e'),
    f = require('gulp-f'),
    g = require('gulp-g'),
    //更多的插件...
    z = require('gulp-z');
```

虽然这没什么问题，但会使我们的 `gulpfile.js` 文件变得很冗长，看上去不那么舒服。 `gulp-load-plugins` 插件正是用来解决这个问题。 `gulp-load-plugins` 这个插件能自动帮你加载 `package.json` 文件里的gulp插件。例如假设你的 `package.json` 文件里的依赖是这样的：

```
{
  "devDependencies": {
    "gulp": "~3.6.0",
    "gulp-rename": "~1.2.0",
    "gulp-ruby-sass": "~0.4.3",
    "gulp-load-plugins": "~0.5.1"
  }
}
```

然后我们可以在 `gulpfile.js` 中使用 `gulp-load-plugins` 来帮我们加载插件：

```
var gulp = require('gulp');
//加载gulp-load-plugins插件，并马上运行它
var plugins = require('gulp-load-plugins')();
```

然后我们要使用gulp-rename和gulp-ruby-sass这两个插件的时候，就可以使用 `plugins.rename` 和 `plugins.rubySass` 来代替了,也就是原始插件名去掉 `gulp-` 前缀，之后再转换为驼峰命名。
实质上 `gulp-load-plugins` 是为我们做了如下的转换

```
plugins.rename = require('gulp-rename');
plugins.rubySass = require('gulp-ruby-sass');
```

`gulp-load-plugins` 并不会一开始就加载所有 `package.json` 里的gulp插件，而是在我们需要用到某个插件的时候，才去加载那个插件。最后要提醒的一点是，因为 `gulp-load-plugins` 是通过你的 `package.json` 文件来加载插件的，所以必须要保证你需要自动加载的插件已经写入到了 `package.json` 文件里，并且这些插件都是已经安装好了的。

4.2 重命名

使用gulp-rename

安装：`npm install --save-dev gulp-rename`

用来重命名文件流中的文件。用`gulp.dest()`方法写入文件时，文件名使用的是文件流中的文件名，如果要想改变文件名，那可以在之前用`gulp-rename`插件来改变文件流中的文件名。

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

gulp.task('rename', function () {
    gulp.src('js/jquery.js')
    .pipe(uglify())  //压缩
    .pipe(rename('jquery.min.js')) //会将jquery.js重命名为
jquery.min.js
    .pipe(gulp.dest('js'));
    //关于gulp-rename的更多强大的用法请参考
https://www.npmjs.com/package/gulp-rename
});
```

4.3 js文件压缩

使用gulp-uglify

安装：`npm install --save-dev gulp-uglify`

用来压缩js文件，使用的是uglify引擎

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

gulp.task('minify-js', function () {
    gulp.src('js/*.js') // 要压缩的js文件
    .pipe(uglify())  //使用uglify进行压缩,更多配置请参考:
    .pipe(gulp.dest('dist/js')); //压缩后的路径
});
```

4.4 css文件压缩

使用gulp-minify-css

安装：`npm install --save-dev gulp-minify-css`

要压缩css文件时可以使用该插件

```
var gulp = require('gulp'),
    minifyCss = require("gulp-minify-css");

gulp.task('minify-css', function () {
    gulp.src('css/*.css') // 要压缩的css文件
    .pipe(minifyCss()) //压缩css
    .pipe(gulp.dest('dist/css'));
});
```

4.5 html文件压缩

使用gulp-minify-html
安装： `npm install --save-dev gulp-minify-html`
用来压缩html文件

```
var gulp = require('gulp'),
    minifyHtml = require("gulp-minify-html");

gulp.task('minify-html', function () {
    gulp.src('html/*.html') // 要压缩的html文件
    .pipe(minifyHtml()) //压缩
    .pipe(gulp.dest('dist/html'));
});
```

4.6 js代码检查

使用gulp-jshint
安装： `npm install --save-dev gulp-jshint`
用来检查js代码

```
var gulp = require('gulp'),
    jshint = require("gulp-jshint");

gulp.task('jsLint', function () {
    gulp.src('js/*.js')
    .pipe(jshint())
    .pipe(jshint.reporter()); // 输出检查结果
});
```

4.7 文件合并

使用gulp-concat
安装： `npm install --save-dev gulp-concat`
用来把多个文件合并为一个文件,我们可以用它来合并js或css文件等，这样就能减少页面的http请求数了

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

gulp.task('concat', function () {
    gulp.src('js/*.js')  //要合并的文件
    .pipe(concat('all.js'))  // 合并匹配到的js文件并命名为 "all.js"
    .pipe(gulp.dest('dist/js'));
});
```

4.8 less和sass的编译

less使用gulp-less,安装： `npm install --save-dev gulp-less`

```
var gulp = require('gulp'),
    less = require("gulp-less");
```

```
gulp.task('compile-less', function () {
    gulp.src('less/*.less')
    .pipe(less())
    .pipe(gulp.dest('dist/css'));
});
```

sass使用gulp-sass,安装： `npm install --save-dev gulp-sass`

```
var gulp = require('gulp'),
    sass = require("gulp-sass");

gulp.task('compile-sass', function () {
    gulp.src('sass/*.sass')
    .pipe(sass())
    .pipe(gulp.dest('dist/css'));
});
```

4.9 图片压缩

可以使用gulp-imagemin插件来压缩jpg、png、gif等图片。
安装： `npm install --save-dev gulp-imagemin`

```
var gulp = require('gulp');
var imagemin = require('gulp-imagemin');
var pngquant = require('imagemin-pngquant'); //png图片压缩插件

gulp.task('default', function () {
    return gulp.src('src/images/*')
        .pipe(imagemin({
            progressive: true,
            use: [pngquant()] //使用pngquant来压缩png图片
        }))
        .pipe(gulp.dest('dist'));
});
```

gulp-imagemin的使用比较复杂一点，而且它本身也有很多插件，建议去它的项目主页看看文档

4.10 自动刷新

使用gulp-livereload插件，安装:
`npm install --save-dev gulp-livereload` 。
当代码变化时，它可以帮我们自动刷新页面
该插件最好配合谷歌浏览器来使用，且要安装livereload chrome extension扩展插件,不能下载的请自行FQ。

```
var gulp = require('gulp'),
    less = require('gulp-less'),
    livereload = require('gulp-livereload');

gulp.task('less', function() {
  gulp.src('less/*.less')
```

```
    .pipe(less())
    .pipe(gulp.dest('css'))
    .pipe(livereload());
});

gulp.task('watch', function() {
  livereload.listen(); //要在这里调用listen()方法
  gulp.watch('less/*.less', ['less']);
});
```

如对gulp还有什么不明白之处，或者本文有什么遗漏或错误，欢迎一起交流和探讨~

分类: js

« 上一篇：用js动态生成css代码

posted on 2015-02-05 00:28 无双 阅读(32802) 评论(29) 编辑 收藏