

Javascript 高级自动化构建工具

随着互联网的飞速发展，前端代码复杂度和规模增加，使用构建工具实现自动化的前端开发流程很有必要。前端自动化构建工具具有代码压缩、编译、监控等功能，主要完成前端一些反复重复的任务。目前主流的前端自动化构建工具有 Grunt、Gulp 等，我们就以 gulp 为例来带大家看看自动化构建工具的使用。

Gulp

什么是 GULP

GULP 是前端开发过程中对代码进行构建的工具，是自动化项目的构建利器；她不仅可对网站资源进行优化，而且在开发过程中很多重复的任务能够使用正确的工具自动完成；使用她，我们不仅可以很愉快的编写代码，而且大大提高我们的工作效率。

GULP 是基于 Nodejs 的自动任务运行器，她能自动化地完成 javascript/coffee/sass/less/html/image/css 等文件的测试、检查、合并、压缩、格式化、浏览器自动刷新、部署文件生成，并监听文件在改动后重复指定的这些步骤。在实现上，她借鉴了 Unix 操作系统的管道（pipe）思想，前一级的输出，直接变成后一级的输入，使得在操作上非常简单。

gulp 和 grunt 非常类似，但相比于 grunt 的频繁 IO 操作，gulp 的流操作，能更快地更便捷地完成构建工作。

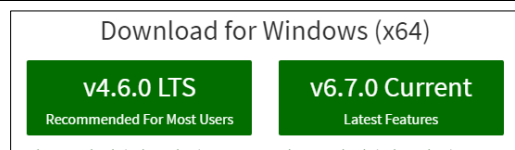
安装 gulp

Gulp 是基于 node.js 环境运行的，所以我们首先需要安装 node.js 环境。

Node.js 的安装

1. 下载 nodejs 安装包

到 nodejs 官网下载软件 <https://nodejs.org>



2. 安装 Nodejs

在 Window 系统中直接下一步下一步式安装。安装好后 WIN+R 输入 cmd 调出 DOS 窗口, 输入 `node -v` 看看是否有版本信息, 如果没有则查看系统变量 Path, 把 path 配置上去。

```
C:\Users\Administrator>node -v
v4.3.1

C:\Users\Administrator>
```

如果在 cmd 中能够输出上图的信息, 说明 node.js 的环境已经安装成功了, 下面就可以开始 gulp 的安装了。

3. 推荐安装 cnpm

npm 服务器在国外, 网络影响大, 甚至还会遇到需要翻墙才能下载插件的情况, 因此推荐安装 cnpm。注: cnpm 跟 npm 用法完全一致, 只是在执行命令时将 npm 改为 cnpm。后面的演示均使用 cnpm【淘宝 npm 镜像, 这是一个完整 npmjs.org 镜像, 你可以用此代替官方版本(只读), 同步频率目前为 10 分钟 一次以保证尽量与官方服务同步】。镜像地址: <http://npm.taobao.org/>

3.1 安装 cnpm

执行 `npm install cnpm -g --registry=https://registry.npm.taobao.org`

3.2 检测 cnpm 是否安装成功

执行 `cnpm -v` 显示版本号即安装成功

gulp 的安装

全局安装 gulp

打开 Node.js command prompt 或者 cmd 命令提示符

输入命令：npm install -g gulp

然后输入 gulp -v，看看是否有版本信息。

项目文件根目录新建 package.json

注：package.json 是基于 nodejs 项目必不可少的配置文件，它是存放在项目根目录的普通 json 文件重点内容

➤ 进入你的项目文件

示例：进入 D:/WWW/test 项目文件夹中

gulp 安装教程，使用教程，简单的自动化任务教程

执行命令 **cnpm init** 来新建 package.json

gulp 安装教程，使用教程，简单的自动化任务教程

检测 package.json 是否成功新建

查看项目文件根目录，是否新建 package.json，且内容是否和你终端中输入的一致。

注：可不使用 cnpm init 方式，可选择手动创建 package.json 配置文件

本地安装 gulp

建立项目，创建 **gulpfile.js** 和 package.json 文件。运行命令

cnpm install gulp --save-dev

安装本地 gulp。

gulp 的实施与使用

创建工作目录与发布目录

工作目录：即没有经过压缩合并等处理的文件存放目录。

发布目录：即项目发布所引用的文件目录地址，这里的文件是经过 **gulp** 压缩合并等处理后生成的文件，处理过后的名字可以自定义，也可以不处理即与未处理文件名相同。

安装相关插件

Gulp 提供了大量的常用插件，供我们使用，下面列举一些大家可能经常要使用的插件：

- ✓ sass 的编译 (gulp-sass)
- ✓ less 编译 (gulp-less)
- ✓ 重命名 (gulp-rename)
- ✓ 自动添加 css 前缀 (gulp-autoprefixer)
- ✓ 压缩 css (gulp-clean-css)
- ✓ js 代码校验 (gulp-jshint)
- ✓ 合并 js 文件 (gulp-concat)
- ✓ 压缩 js 代码 (gulp-uglify)
- ✓ 压缩图片 (gulp-imagemin)
- ✓ 自动刷新页面 (gulp-livereload, 谷歌浏览器亲测，谷歌浏览器需安装 livereload 插件)
- ✓ 图片缓存，只有图片替换了才压缩 (gulp-cache)
- ✓ 更改提醒 (gulp-notify)

less 和 sass 的编译 (gulp-less, gulp-sass)

Less 插件：

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

在 Gulpfile.js 执行如下代码：

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

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

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

Sass 插件:

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

在 Gulpfile.js 执行如下代码:

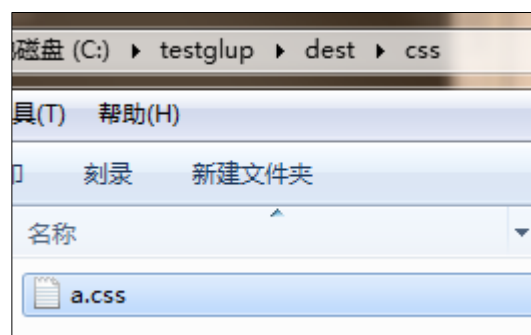
```
var gulp = require('gulp'),
    sass = require("gulp-sass");

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

我们以 gulp-less 插件为例: 我们在的代码中定义了一个任务—test-less, 那么这个任务就会去找我们定义的第二个参数回调函数中的 gulp.src 中的目录, 之后执行 less 方法 (就是 require 得到的 less 插件对象), 之后这个插件就会将 src 目录下匹配的 less 文件编译到 gulp.dest 方法对应路径下的 CSS 文件:

```
>gulp test-less
Using gulpfile C:\testglup\gulpfile.js
Starting 'test-less'...
Finished 'test-less' after 7.69 ms
```

得到 CSS 文件:



那么这就是 less 插件, 完成 less 文件的编译。

一般而言, less 和 sass 我们不会传递参数, 如果确实存在了参数的传递的话, 在 less() 方法中可以传递一个对象 (json) 来控制, 如:

```
gulp.task('less', function(){
  return gulp.src('src/less/*.less')
    .pipe(less({globalVars:{ENV:dev, version:10}}))
    .pipe(gulp.dest('dest/css'));
});
```

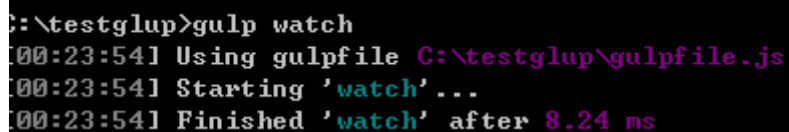
当然我们每次等写完 less 或者 sass 之后在执行一下 cmd 命令有时候也感觉很麻烦，我们可以使用自动编译任务：

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

gulp.task("test-less",function(){
  gulp.src("src/less/*.less")
    .pipe(gulp_less())
    .pipe(gulp.dest("dest/css"));
});

gulp.task("watch",function(){
  //watch 方法会自动的执行我们传递的任务（第二个参数）
  //第一个参数是任务需要监控的路径
  gulp.watch("src/less/*.less",["test-less"]);
});
```

此时，我们再去执行 gulp watch，gulp 就会执行我们的 watch 方法，这个方法会自动的编译（时时检测 less 或者 sass 文件的变化）来编译我们的文档，减少我们不断的去敲命令来完成编译。



```
C:\testglup>gulp watch
[00:23:54] Using gulpfile C:\testglup\gulpfile.js
[00:23:54] Starting 'watch'...
[00:23:54] Finished 'watch' after 8.24 ms
```

压缩 HTML 页面（gulp-htmlmin）

Gulp-htmlmin 插件的安装：

```
cnpm install gulp-htmlmin --save-dev
```

安装成功后，我们在 gulpfile.js 中编写代码，完成页面的压缩：

```
var gulp = require("gulp");//gulp模块
var gulp_htmlmin = require('gulp-htmlmin');//gulp-htmlmin插件模块

//test-htmlmin任务
gulp.task("test-htmlmin",function(){
  var options = {
    removeComments: true,//清除HTML注释
    collapseWhitespace: true,//压缩HTML
    collapseBooleanAttributes: true,//省略布尔属性的值 <input checked="true"/> ==> <input />
    removeEmptyAttributes: true,//删除所有空格作属性值 <input id="" /> ==> <input />
    removeScriptTypeAttributes: true,//删除<script>的type="text/javascript"
    removeStyleLinkTypeAttributes: true,//删除<style>和<link>的type="text/css"
    minifyJS: true,//压缩页面JS
    minifyCSS: true//压缩页面CSS
  };
  gulp.src("src/*.html")
    .pipe(gulp_htmlmin(options))
    .pipe(gulp.dest("dest/"))
  });
```

执行命令：

```
gulp test-htmlmin
```

此时就会压缩我们的 src 目录下的所有页面。

gulp-autoprefixer

使用 gulp-autoprefixer 根据设置浏览器版本自动处理浏览器前缀。使用她我们可以很潇洒地写代码，不必考虑各浏览器兼容前缀。【特别是开发移动端页面时，就能充分体现它的优势。例如兼容性不太好的 flex 布局。】

本地安装 gulp-autoprefixer

```
cnpm install gulp-autoprefixer --save-dev
```

之后在 gulpfile.js 中书写任务：

```
var gulp = require("gulp");//gulp模块
var gulp_less = require("gulp-less");//gulp-less插件模块
var gulp_sass = require("gulp-sass");//gulp-sass插件模块
var gulp_htmlmin = require('gulp-htmlmin');//gulp-htmlmin插件模块
```

```
var autoprefixer = require('gulp-autoprefixer');//autoprefixer插件模块

//test-autoprefixer任务
gulp.task("test-autoprefixer",function(){
  gulp.src("src/css/*.css")
    .pipe(autoprefixer({
      browsers: ['last 2 versions', 'Android >= 4.0'],
      cascade: true, //是否美化属性值 默认: true 像这样:
        //-webkit-transform: rotate(45deg);
        //      transform: rotate(45deg);
      remove:true //是否去掉不必要的前缀 默认: true
    }))
    .pipe(gulp.dest("dest/css/"))
  });
```

在 cmd 中执行命令:

```
gulp test-autoprefixer
```

我们发现我们的 CSS 文件都被补全前缀了, 再也不担心兼容问题了~~。

gulp-autoprefixer 的 browsers 参数详解:

last 2 versions: 主流浏览器的最新两个版本
last 1 Chrome versions: 谷歌浏览器的最新版本
last 2 Explorer versions: IE 的最新两个版本
last 3 Safari versions: 苹果浏览器最新三个版本
Firefox >= 20: 火狐浏览器的版本大于或等于 20
iOS 7: IOS7 版本
Firefox ESR: 最新 ESR 版本的火狐
> 5%: 全球统计有超过 5%的使用率

各浏览器的标识

发现上面规律了吗, 相信这不难看出, 接下来说说各浏览器的标识:

Android for Android WebView.
BlackBerry or bb for Blackberry browser.
Chrome for Google Chrome.
Firefox or ff for Mozilla Firefox.

Explorer or ie for Internet Explorer.

iOS or ios_saf for iOS Safari.

Opera for Opera.

Safari for desktop Safari.

OperaMobile or op_mob for Opera Mobile.

OperaMini or op_mini for Opera Mini.

ChromeAndroid or and_chr

FirefoxAndroid or and_ff for Firefox for Android.

ExplorerMobile or ie_mob for Internet Explorer Mobile

gulp-minify-css 【gulp-clean-css】

使用 gulp-minify-css 压缩 css 文件，减小文件大小，并给引用 url 添加版本号避免缓存。

重要：gulp-minify-css 已经被废弃，请使用 gulp-clean-css，用法一致。

本地安装 gulp-minify-css

github: <https://github.com/jonathanpollack/gulp-minify-css>

命令提示符执行 `cnpm install gulp-minify-css --save-dev`

gulpfile.js 文件:

```
//引入所需的模块
var gulp = require("gulp");//gulp模块
var gulp_less = require("gulp-less");//gulp-less插件模块
var gulp_sass = require("gulp-sass");//gulp-sass插件模块
var gulp_htmlmin = require('gulp-htmlmin');//gulp-htmlmin插件模块
var autoprefixer = require('gulp-autoprefixer');//autoprefixer插件模块
var cssmin_minify = require('gulp-minify-css');
var cssmin_clean = require('gulp-clean-css');

//确保已本地安装gulp-make-css-url-version [cnpm install gulp-make-css-url-version --save-dev]
var cssver = require('gulp-make-css-url-version');

//test-minify任务
gulp.task("test-minify",function(){
    gulp.src("src/css/*.css")
        .pipe(cssmin_minify({
            advanced: false,//类型: Boolean 默认: true [是否开启高级优化(合并选择器等)]
            compatibility: 'ie7',//保留ie7及以下兼容写法 类型: String 默认: 'or'* [启用兼容模式;
```

```
'ie7': IE7兼容模式, 'ie8': IE8兼容模式, '*': IE9+兼容模式]
    keepBreaks: true, //类型: Boolean 默认: false [是否保留换行]
    keepSpecialComments: '*'
    //保留所有特殊前缀 当你用autoprefixer生成的浏览器前缀, 如果不加这个参数, 有可能将会删除你的部
    分前缀
  })
  .pipe(gulp.dest("dest/css/"))
});
```

给 css 文件里引用 url 加版本号 (根据引用文件的 md5 生产版本号), 像这样:

```
.test1 {background:url(.. /img/test.png?v=rn059ZYitm98d8%2Bgnp6v9A%3D%3D)no-repeat}
```

```
//确保已本地安装gulp-make-css-url-version [cnpm install gulp-make-css-url-version --save-dev]
var cssver = require('gulp-make-css-url-version');

//cssmin_clean任务
gulp.task('test-cssmin', function () {
  gulp.src('src/css/*.css')
    .pipe(cssver()) //给css文件里引用文件加版本号 (文件MD5) 防止缓存
    .pipe(cssmin_clean())
    .pipe(gulp.dest('dest/css'));
});
```

此时 CSS 中的图片就会都加上参数, 防止缓存。

gulp-uglify, js 文件压缩

使用 gulp-uglify 压缩 javascript 文件, 减小文件大小。

本地安装 gulp-uglify

github: <https://github.com/terinjokes/gulp-uglify>

命令提示符执行 cnpm install gulp-uglify --save-dev

gulpfile.js 文件:

```
gulp.task('jsmin', function () {
  gulp.src('src/js/index.js')
    .pipe(uglify())
    .pipe(gulp.dest('dist/js'));
});
```

如果有多个文件压缩:

```
//多个文件
gulp.task('jsmin', function () {
```

```
gulp.src(['src/js/index1.js', 'src/js/login.js', 'src/js/js1.js', 'src/js/omg.js'])
//多个文件以数组形式传入
    .pipe(uglify())
    .pipe(gulp.dest('dest/js'));
});
```

也可以使用匹配方式:

```
//排除排除混淆关键字，多个参数
gulp.task('jsmin', function () {
    //压缩src/js目录下的所有js文件
    //除了test1.js和test2.js (**匹配src/js的0个或多个子文件夹)
    gulp.src(['src/js/*.js', '!src/js/**/*.js'])
        .pipe(uglify({
            //      mangle: true, //类型: Boolean 默认: true 是否修改变量名
            mangle: {except: ['require', 'exports', 'module', '$']}, //排除混淆关键字
            compress: true, //类型: Boolean 默认: true 是否完全压缩
            preserveComments: 'all' //保留所有注释
        }))
        .pipe(gulp.dest('dest/js'));
});
```

gulp-concat

使用 gulp-concat 合并 javascript 文件，减少网络请求。

本地安装 gulp-concat

github: <https://github.com/wearefractal/gulp-concat>

命令提示符执行 `cnpm install gulp-concat --save-dev`

gulpfile.js 文件:

```
var concat = require('gulp-concat');
//合并js文件
gulp.task('test_concat', function () {
    gulp.src('src/js/*.js')
        .pipe(concat('all.js')) //合并后的文件名
        .pipe(gulp.dest('test/js'));
});
```

gulp-imagemin

使用 gulp-imagemin 压缩图片文件（包括 PNG、JPEG、GIF 和 SVG 图片）。注意：安装时很容易出错的。

本地安装 gulp-imagemin

github: <https://github.com/sindresorhus/gulp-imagemin>

命令提示符执行 `cnpm install gulp-imagemin --save-dev`

gulpfile.js 文件:

基本压缩:

```
var concat = require('gulp-concat');
//合并js文件
gulp.task('test_concat', function () {
  gulp.src('src/js/*.js')
    .pipe(concat('all.js'))//合并后的文件名
    .pipe(gulp.dest('test/js'));
});
```

gulp-imagemin 其他参数:

```
gulp.task('testImagemin', function () {
  gulp.src('src/img.{png,jpg,gif,ico}')
    .pipe(imagemin({
      optimizationLevel: 5, //类型: Number 默认: 3 取值范围: 0-7 (优化等级)
      progressive: true, //类型: Boolean 默认: false 无损压缩jpg图片
      interlaced: true, //类型: Boolean 默认: false 隔行扫描gif进行渲染
      multipass: true //类型: Boolean 默认: false 多次优化svg直到完全优化
    }))
    .pipe(gulp.dest('dest/img'));
});
```

深度压缩图片

```
var imagemin = require('gulp-imagemin');
//确保本地已安装imagemin-pngquant [cnpm install imagemin-pngquant --save-dev]
var pngquant = require('imagemin-pngquant');

//深度压缩图片
gulp.task('testImagemin', function () {
  gulp.src('src/img/*.{png,jpg,gif,ico}')
    .pipe(imagemin({
      progressive: true,
      svgoPlugins: [{removeViewBox: false}],//不要移除svg的viewbox属性
    }));
});
```

```
        use: [pngquant()] //使用pngquant深度压缩png图片的imagemin插件
    })))
    .pipe(gulp.dest('dest/img'));
});
```

只压缩修改的图片。压缩图片时比较耗时，在很多情况下我们只修改了某些图片，没有必要压缩所有图片，使用”gulp-cache”只压缩修改的图片，没有修改的图片直接从缓存文件读取（C:\Users\Administrator\AppData\Local\Temp\gulp-cache）：

```
//确保本地已安装imagemin-pngquant [cnpm install imagemin-pngquant --save-dev]
var pngquant = require('imagemin-pngquant');
//确保本地已安装gulp-cache [cnpm install gulp-cache --save-dev]
var cache = require('gulp-cache');

gulp.task('testImagemin', function () {
    gulp.src('src/img/*.{png,jpg,gif,ico}')
        .pipe(cache(imagemin({
            progressive: true,
            svgoPlugins: [{removeViewBox: false}],
            use: [pngquant()]
        }))))
    .pipe(gulp.dest('dest/img'));
});
```

Fis

FIS 是专门为解决前端开发中自动化工具、性能优化、模块化框架、开发规范、代码部署、开发流程等问题的工具框架，由中国百度公司开发。

FIS 的官方网站：

```
http://fis.baidu.com/
```

FIS 的解决方案官网：

```
http://oak.baidu.com/
```

FIS 是一系列提升产品质量与开发效率的工程化方案。

Fis 的安装

安装 fis：

```
cnpm install -g fis
```

`fis server start` #启动 8080 服务。在浏览器访问（自动弹出） `http://127.0.0.1:8080/`

如果 8080 端口已被占用，可用 `fis server start -p 8081`

打开服务器端项目的位置：

```
fis server open
```

配置 `fis-con.js` 完成压缩。（注意：也可以不配置就完成压缩，配置就是过滤文件）

```
npm install -g fis-postpackager-simple
```

需要在你的项目的根路径下新建一个文件：`fis-conf.js`。

```
//Step 1. 开启 simple 插件，注意需要先进行插件安装 npm install -g fis-postpackager-simple
fis.config.set('modules.postpackager', 'simple');

//通过 pack 设置干预自动合并结果，将公用资源合并成一个文件，更加利于页面间的共用

//Step 2. 将[]里面的所有文件打包到 pkg/lib.js 目录里
fis.config.set('pack', {
  'pkg/lib.js': [
    '/lib/mod.js',
    '/modules/underscore/**/*.js',
    '/modules/backbone/**/*.js',
    '/modules/jquery/**/*.js',
    '/modules/vendor/**/*.js',
```

```
'/modules/common/**/*.js'
```

```
]
});
```

//Step 3. 除了 Step2 中手动设置合并的 JS 脚本之外。通过 Step3 的配置，可以对其他的脚本进行自动合并。

```
.fis.config.set('settings.postpackager.simple.autoCombine', true);
```

之后在运行 fis 的 release 命令：

fis release -参数

Fis 的命令共有如下几个：

- Fis install 用来安装插件
- Fis server 用来管理服务器
- Fis release 发布打包等

我们可以通过 -h 参数来查看参数的作用

Fis release -o //压缩资源文件，包括 CSS、js 文件

也可以参数联合使用

Fis release -po //打包压缩

Base64 图片

在图片的 URL 后增加?__inline 这样 fis 就会自动生成 base64 图片