## What is gulp?

　　gulp是前端开发过程中一种基于流的代码构建工具，是自动化项目的构建利器；她不仅能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，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。



## 流(stream)

　　流,简单来说就是建立在面向对象基础上的一种抽象的处理数据的工具。在流中，定义了一些处理数据的基本操作，如读取数据，写入数据等，程序员是对流进行所有操作的，而不用关心流的另一头数据的真正流向。流不但可以处理文件，还可以处理动态内存、网络数据等多种数据形式。

　　而gulp正是通过流和代码优于配置的策略来尽量简化任务编写的工作。这看起来有点“像jQuery”的方法，把动作串起来创建构建任务。早在Unix的初期，流就已经存在了。流在Node.js生态系统中也扮演了重要的角色，类似于\*nix将几乎所有设备抽象为文件一样，Node将几乎所有IO操作都抽象成了stream的操作。因此用gulp编写任务也可看作是用Node.js编写任务。当使用流时，gulp去除了中间文件，只将最后的输出写入磁盘，整个过程因此变得更快。

## 特点

**易于使用**

　　通过代码优于配置的策略，gulp 让简单的任务简单，复杂的任务可管理。

**构建快速**

　　利用 Node.js 流的威力，你可以快速构建项目并减少频繁的 IO 操作。

**易于学习**

　　通过最少的 API，掌握 gulp 毫不费力，构建工作尽在掌握：如同一系列流管道。

**插件高质**

　　gulp 严格的插件指南确保插件如你期望的那样简洁高质得工作。

## 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：

1. 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：

1. 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：

1. npm install --save-dev gulp

　　这样就完成了gulp的安装，接下来就可以在项目中应用gulp了。

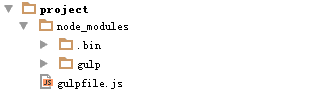
## gulp的使用

1 建立gulpfile.js文件

　　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。

1. var gulp = require('gulp');
2. gulp.task('default',function(){
3. console.log('hello world');
4. });

此时我们的目录结构是这样子的：



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。

## 工作方式

　　在介绍gulp API之前，我们首先来说一下gulp.js工作方式。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，这也是我们应用gulp的一个原因。

　　gulp的使用流程一般是：首先通过gulp.src()方法获取到想要处理的文件流，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。例如：

1. var gulp = require('gulp');
2. gulp.src('script/jquery.js') // 获取流的api
3. .pipe(gulp.dest('dist/foo.js')); // 写放文件的api

　　我们将在本章内容中来给同学们讲解gulp API，其中包括gulp.src()，gulp.task()，gulp.dest()，gulp.watch()，gulp.run()。

## globs的匹配规则

　　我们重点说说gulp用到的globs的匹配规则以及一些文件匹配技巧，我们将会在后面的课程中用到这些规则。

　　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：

|  |  |  |
| --- | --- | --- |
| **匹配符** | **说明** |  |
| \* | 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

## src

　　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。其语法为：

1. 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;类型为String或 Array。我们在前一节中已经讲过了globs的匹配规则，这里就不在详述。

当有多种匹配模式时可以使用数组

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

options为可选参数。以下为options的选项参数:

**options.buffer**

类型： Boolean 默认值： true

　　如果该项被设置为 false，那么将会以 stream 方式返回 file.contents 而不是文件 buffer 的形式。这在处理一些大文件的时候将会很有用。注意：插件可能并不会实现对 stream 的支持。

**options.read**

类型： Boolean 默认值： true

如果该项被设置为 false， 那么 file.contents 会返回空值（null），也就是并不会去读取文件。

**options.base**

类型： String ， 设置输出路径以某个路径的某个组成部分为基础向后拼接。

如, 请想像一下在一个路径为 client/js/somedir 的目录中，有一个文件叫 somefile.js ：

1. gulp.src('client/js/\*\*/\*.js') // 匹配 'client/js/somedir/somefile.js' 现在 `base` 的值为 `client/js/`
2. .pipe(minify())
3. .pipe(gulp.dest('build')); // 写入 'build/somedir/somefile.js' 将`client/js/`替换为build
5. gulp.src('client/js/\*\*/\*.js', { base: 'client' }) // base 的值为 'client'
6. .pipe(minify())
7. .pipe(gulp.dest('build')); // 写入 'build/js/somedir/somefile.js' 将`client`替换为build

## dest

　　gulp.dest()方法是用来写文件的，其语法为：

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

path为写入文件的路径；

options为一个可选的参数对象，以下为选项参数：

**options.cwd**

类型： String 默认值： process.cwd()

输出目录的 cwd 参数，只在所给的输出目录是相对路径时候有效。

**options.mode**

类型： String 默认值： 0777

八进制权限字符，用以定义所有在输出目录中所创建的目录的权限。

1. var gulp = require('gulp');
2. gulp.src('script/jquery.js')　 // 获取流
3. .pipe(gulp.dest('dist/foo.js')); // 写放文件

　　 　　下面再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：

1. var gulp = reruire('gulp');
2. //有通配符开始出现的那部分路径为 \*\*/\*.js
3. gulp.src('script/\*\*/\*.js')
4. .pipe(gulp.dest('dist')); //最后生成的文件路径为 dist/\*\*/\*.js
5. //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

　　用gulp.dest()把文件流写入文件后，文件流仍然可以继续使用。

## 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

1. 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。

1. gulp.task('uglify',function(){
2. //do something
3. });
4. gulp.task('reload',function(){
5. //do something
6. });
7. gulp.watch('js/\*\*/\*.js', ['uglify','reload']);

　　gulp.watch()还有另外一种使用方式：

1. 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。

1. gulp.watch('js/\*\*/\*.js', function(event){
2. console.log(event.type); //变化类型 added为新增,deleted为删除，changed为改变
3. console.log(event.path); //变化的文件的路径
4. });

## task

　　gulp.task方法用来定义任务，内部使用的是Orchestrator(用于排序、执行任务和最大并发依赖关系的模块)，其语法为：

1. 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。

当你定义一个简单的任务时，需要传入任务名字和执行函数两个属性。

1. gulp.task('greet', function () {
2. console.log('Hello world!');
3. });

　　执行gulp greet的结果就是在控制台上打印出“Hello world”。

　　你也可以定义一个在gulp开始运行时候默认执行的任务，并将这个任务命名为“default”：

1. gulp.task('default', function () {
2. // Your default task
3. });

　　前面已经介绍了gulp.task的语法，但是当有多个任务时，需要知道怎么来控制任务的执行顺序。

　　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：

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

　　如果任务相互之间没有依赖，任务就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：

1. gulp.task('one',function(){
2. //one是一个异步执行的任务
3. setTimeout(function(){
4. console.log('one is done')
5. },5000);
6. });
8. //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
9. gulp.task('two',['one'],function(){
10. console.log('two is done');
11. });

　　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。

## run

　　gulp.run()表示要执行的任务。可能会使用单个参数的形式传递多个任务。如下代码：

1. gulp.task('end',function(){
2. gulp.run('task1','task3','task2');
3. });

　　注意：任务是尽可能多的并行执行的，并且可能不会按照指定的顺序运行。

## 插件安装

　　在我们编写gulp代码时候，需要用到一些gulp的插件，可以通过npm install --save-dev 插件名称 来安装。如下代码来安装自动加载插件：

1. 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。

1. var gulp=require('gulp'),
2. plugins=require('gulp-load-plugins')(),
3. uglify = require('gulp-uglify'),
4. minifyHtml = require('gulp-minify-html'),
5. rename = require('gulp-rename');

　　gulp的插件有很多种，我们将在本节内容中为同学们讲解几个插件的用法。如还想了解更多插件，请查阅相关资料。

## 自动加载

　　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:

1. {
2. "devDependencies": {
3. "gulp": "~3.6.0",
4. "gulp-rename": "~1.2.0",
5. "gulp-ruby-sass": "~0.4.3",
6. "gulp-load-plugins": "~0.5.1"
7. }
8. }

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

1. var gulp = require('gulp');
2. //加载gulp-load-plugins插件，并马上运行它
3. 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。

## 重命名

　　gulp-rename插件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。

1. var gulp = require('gulp'),
2. rename = require('gulp-rename'),
3. uglify = require("gulp-uglify");
5. gulp.task('rename', function () {
6. gulp.src('src/1.js')
7. .pipe(uglify()) //压缩
8. .pipe(rename('1.min.js')) //会将1.js重命名为1.min.js
9. .pipe(gulp.dest('js'));
10. });

1. 在project目录中，先建立一个src目录；
2. 再建立一个1.js的文件，写入一些内容；
3. 然后再gulpfile.js中输入以上代码保存；
4. 然后命令行转入到project目录运行，刷新project目录看看会出现什么结果呢！

## js文件压缩

　　gulp-uglify插件用来压缩js文件。

1. var gulp = require('gulp'),
2. uglify = require("gulp-uglify");
4. gulp.task('minify-js', function () {
5. gulp.src('src/\*.js') // 要压缩的js文件
6. .pipe(uglify()) //使用uglify进行压缩
7. .pipe(gulp.dest('dist/js')); //压缩后的路径
8. });

1. 在project目录中，先建立一个src目录；
2. 再建立一个1.js的文件，写入一些内容；
3. 然后再gulpfile.js中输入以上代码保存；
4. 然后命令行转入到project目录运行，刷新project目录看看会出现什么结果呢！

## Css文件压缩

　　gulp-minify-css插件用来压缩css文件。

1. var gulp = require('gulp'),
2. minifyCss = require("gulp-minify-css");
4. gulp.task('minify-css', function () {
5. gulp.src('src/\*.css') // 要压缩的css文件
6. .pipe(minifyCss()) //压缩css
7. .pipe(gulp.dest('dist/css'));
8. });

1. 在project目录中，先建立一个src目录；
2. 再建立一个1.css的文件，写入一些内容；
3. 然后再gulpfile.js中输入以上代码保存；
4. 然后命令行转入到project目录运行，刷新project目录看看会出现什么结果呢！

## html文件压缩

　　gulp-minify-html插件用来压缩html文件。

1. var gulp = require('gulp'),
2. minifyHtml = require("gulp-minify-html");
4. gulp.task('minify-html', function () {
5. gulp.src('src/\*.html') // 要压缩的html文件
6. .pipe(minifyHtml()) //压缩
7. .pipe(gulp.dest('dist/html'));
8. });

1. 在project目录中，先建立一个src目录；
2. 再建立一个1.html的文件，写入一些内容；
3. 然后再gulpfile.js中输入以上代码保存；
4. 然后命令行转入到project目录运行，刷新project目录看看会出现什么结果呢！

## js代码检查

　　使用gulp-jshint插件，用来检查js代码。

1. var gulp = require('gulp'),
2. jshint = require("gulp-jshint");
4. gulp.task('jsLint', function () {
5. gulp.src('src/\*.js')
6. .pipe(jshint())
7. .pipe(jshint.reporter()); // 输出检查结果
8. });

1. 在project目录中，先建立一个src目录；
2. 再建立一个1.js的文件，写入一些不合语法的内容；
3. 然后再gulpfile.js中输入以上代码保存；
4. 然后命令行转入到project目录运行，看看会出现什么结果呢！

## 文件合并

　　使用gulp-concat插件，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。

1. var gulp = require('gulp'),
2. concat = require("gulp-concat");
3. gulp.task('concat', function () {
4. gulp.src('src/\*.js') //要合并的文件
5. .pipe(concat('all.js')) // 合并匹配到的js文件并命名为 "all.js"
6. .pipe(gulp.dest('dist/js'));
7. });

1. 在project目录中，先建立一个src目录；
2. 再分别建立两个文件1.js与2.js，写入一些内容；
3. 然后再gulpfile.js中输入以上代码保存；
4. 然后命令行转入到project目录运行，刷新project目录看看会出现什么结果呢！

## 图片压缩

可以使用gulp-imagemin插件来压缩jpg、png、gif等图片。

1. var gulp = require('gulp');
2. var imagemin = require('gulp-imagemin');
3. var pngquant = require('imagemin-pngquant'); //png图片压缩插件
4. gulp.task('default', function () {
5. return gulp.src('src/images/\*')
6. .pipe(imagemin({
7. progressive: true,
8. use: [pngquant()] //使用pngquant来压缩png图片
9. }))
10. .pipe(gulp.dest('dist'));
11. });

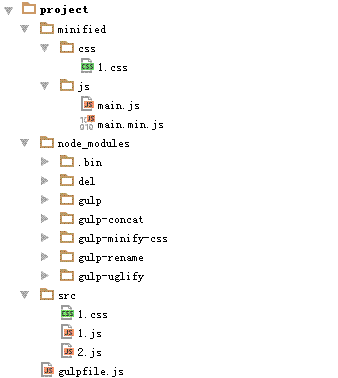
## 自动刷新

　　使用gulp-livereload插件，当代码变化时，它可以帮我们自动刷新页面。

1. var gulp = require('gulp'),
2. less = require('gulp-less'),
3. livereload = require('gulp-livereload');
4. gulp.task('less', function() {
5. gulp.src('less/\*.less')
6. .pipe(less())
7. .pipe(gulp.dest('css'))
8. .pipe(livereload());
9. });
10. gulp.task('watch', function() {
11. livereload.listen(); //要在这里调用listen()方法
12. gulp.watch('less/\*.less', ['less']);
13. });

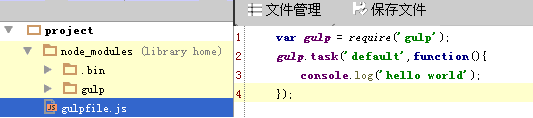
## 项目需求

　　在本章内容中，我们将创建一个自己的gulp，具体的需求是通过gulp把我们自己所编写的JS文件合并压缩、CSS文件进行压缩后，并且生成新的文件。我们所需要的插件为：gulp-minify-css gulp-concat gulp-uglify gulp-rename del 如下图所示，完成后的项目目录结构：

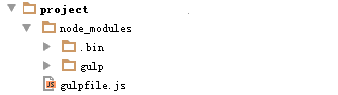


## 创建项目

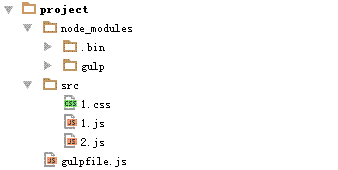
　　在创建目录之前我们要先安装Nodejs，通过在安装好gulp（编辑环境中已经为同学们安装完毕）的环境中，点击【文件管理】展开后，就可以创建目录与文件了。



　　首先我们先来创建一个名为project的目录，然后进入到该目录下面，再将gulp安装到我们项目的目录中，然后在该目录下新建一个名称为gulpfile.js的文件。安装好后的目录结构为：



在该目录下再创建一个src目录，来存放源JS与CSS文件。建立完成后，再src目录分别建立两个js文件与一个CSS文件。完成后的目录结构为：



## 安装插件

　　根据我们项目的需求，安装所需要的插件，可以通过"npm install 插件名" 来安装插件。安装完成后的目录结构如图所示。



　　然后打开gulpfile.js，将我们所用到的插件引用到我们项目中，代码如下:

1. var gulp = require('gulp'),
2. minifycss = require('gulp-minify-css'), //CSS压缩
3. concat = require('gulp-concat'), // 文件合并
4. uglify = require('gulp-uglify'), //js压缩插件
5. rename = require('gulp-rename'), // 重命名
6. del = require('del'); // 文件删除

## 编写代码

　　上一节中已经完成了对插件的引用，下面就开始我们的代码编写，可以通过gulp.start()方法来开始执行我们的任务。

1.gulp默认的执行任务是 "default"，当然你也可以指定别的名称，然后通过"gulp 任务名称" 来运行。

1. gulp.task('default', function() {
2. gulp.start('clean','minifycss', 'minifyjs'); // 要执行的任务
3. });

2.CSS压缩

1. gulp.task('minifycss', function() {
2. return gulp.src('src/\*.css') //压缩的文件
3. .pipe(minifycss()) //执行压缩
4. .pipe(gulp.dest('minified/css')); //输出文件夹
5. });

3.JS 合并压缩

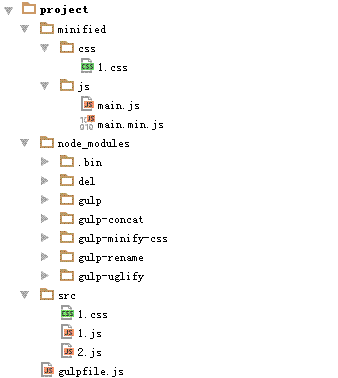
1. gulp.task('minifyjs', function() {
2. return gulp.src('src/\*.js')
3. .pipe(concat('main.js')) //合并所有js到main.js
4. .pipe(gulp.dest('minified/js')) //输出main.js到文件夹
5. .pipe(rename({suffix: '.min'})) //rename压缩后的文件名
6. .pipe(uglify()) //压缩
7. .pipe(gulp.dest('minified/js')); //输出
8. });

4.执行压缩前，先删除目录里的内容

1. gulp.task('clean', function(cb) {
2. del(['minified/css', 'minified/js'], cb)
3. });

好了，这样我们的代码就完成了。

前面我们已经编写完成了代码，在命令行中先转到project目录下，就可以输入gulp命令来运行本项目了，刷新project目录看看会出现什么结果呢。运行完成后的目录如下图：



　　运行过程中的消息如下图所示：

