Gulp学习之task做了什么事情

gulp.task()函数用来定义一个gulp要执行的任务。其格式如下

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

该函数有三个参数：name，deps和fn

**name** 任务的名称，字符串类型，这个参数是必须的。

gulp.task(‘mytask’)

执行该任务

gulp] # gulp mytask

Starting 'mytask'...

Finished 'mytask' after 120 μs

我们定义的这个任务其实什么也没有做。通常情况下是需要有一个回调函数来执行该任务的内容。也就是fn参数

**fn** 回调函数，要执行的任务都在该回调函数中实现。

gulp.task(‘mytask’,function(){

console.log(‘mytask’);

})

执行该任务

gulp] # gulp mytask

Starting 'mytask'...

mytask

Finished 'mytask' after 360 μs

通过上面两个例子，我们看到，在命令行执行gulp的时候，都需要加上任务名称。如果想省略的话需要使用默认的任务名称’default’

gulp.task(‘default’,function(){

console.log(‘default’);

})

继续执行上面代码

gulp] # gulp

Starting 'default'...

default

Finished 'default' after 360 μs

但是一般情况下，我们只是使用default作为一个入口。并不定义实际的内容。其实真正的工作都是由其依赖的任务来做。这就需要使用到参数’deps’。

**deps** 当前任务执行所依赖的任务，该参数是一个数组。也就是说只有在deps中定义的任务完成的情况下当前任务才会执行。

gulp.task('mytask',function(){  
 console.log('mytask');  
})  
gulp.task('task',function(){  
 console.log('task');  
})

gulp.task('default',['mytask','task'],function(){  
 console.log('default');  
});

为了说明其顺序，我们在default中加入一个回调函数。执行代码：

gulp] # gulp

Starting 'mytask'...

mytask

Finished 'mytask' after 309 μs

Starting 'task'...

task

Finished 'task' after 179 μs

Starting 'default'...

default

Finished 'default' after 170 μs

我们知道，js是事件驱动的。所以说所有的这些任务都是异步执行的。它们之间的运行没有先后之分，互相之间是平行的。

gulp.task('mytask',function(){  
 setTimeout(function(){  
 console.log('mytask')  
 },1000)  
})

gulp.task('default',['mytask'],function(){  
 console.log('default');  
});

执行结果如下

gulp] # gulp

Starting 'mytask'...

Finished 'mytask' after 983 μs

Starting 'default'...

default

Finished 'default' after 168 μs

mytask

其中，mytask是在整个程序执行完的1秒以后打印出来的。这种异步的方式导致的一个比较严重的问题就是在实际编译文件过程中，我们会看到程序已经运行完成了，但是我们的文件还在编译。

要解决这个问题就要使程序是同步执行的而不是异步。在官方文档中给出了三种方法：一是在fn中接收一个回调函数作为参数；二是返回一个stream；三是返回一个promise。

gulp.task('mytask',function(){  
 setTimeout(function(){  
 console.log('mytask')  
 },1000)  
})

我们看这个例子

gulp] # gulp mytask

Starting 'mytask'...

Finished 'mytask' after 1.01 ms

mytask

在显示Finished以后才打印的mytask。下面可以通过第一种回调函数的方式来实现同步执行。

gulp.task('mytask',function(cb){  
 return setTimeout(function(){  
 console.log('mytask');  
 cb();  
 },1000);  
})

执行结果如下

gulp] # gulp mytask

Starting 'mytask'...

mytask

Finished 'mytask' after 1 s

现在的结果就是我们想要的了。但是，有一点需要注意的是如果我们在fn种接收了一个回调函数作为参数，那么就必须调用该回调函数，否则程序是没法向下执行的。尤其是在有多个依赖任务的程序中，假如在依赖的任务的fn中接收了回调函数为参数，但是并没有调用该回调函数，那这样造成的结果就是程序执行到该任务就结束了。也就是说后续的任务不再被执行。

gulp.task('task',function(cb){  
 console.log('task');  
})

gulp.task('default',['task'],function(){  
 console.log('default');  
});

运行该程序发现结果会是惊人的

gulp] # gulp

Starting 'task'...

task

很明显这段程序没有执行完就停止了。解决的方法就是在task任务中调用cb();

gulp.task('task',function(cb){  
 console.log('task');  
 cb();  
})

下面回到我们的主线，在说完接收回调函数来使程序达到同步执行后。我们继续看剩下的两种方式。

由于这篇文章尽量不涉及到其它的函数，旨在说明gulp.task的工作方式。所以这里对于剩下的两种方式我们直接使用官网给出的例子

**返回 stream**

gulp.task('somename', function() {

var stream = gulp.src('client/\*\*/\*.js')

.pipe(minify())

.pipe(gulp.dest('build'));

return stream;

});

**返回 promise**

var Q = require('q');

gulp.task('somename', function() {

var deferred = Q.defer();

// do async stuff

setTimeout(function() {

deferred.resolve();

}, 1);

return deferred.promise;

});

**总结**

默认情况下，gulp对于是一次性加载所有的任务，这些任务是并行执行的。我们要想有一个按照一定顺序执行的任务序列，需要做的是：首先为某些任务指定其依赖完成的任务；然后是给任务完成以后通知程序执行下一个任务（通过我们上面介绍的三种方式，可根据实际情况选择）。

同样，我们摘用官网的例子。

var gulp = require('gulp');

// takes in a callback so the engine knows when it'll be done

gulp.task('one', function(cb) {

// do stuff -- async or otherwise

cb(err); // if err is not null and not undefined, the run will stop, and note that it failed

});

// identifies a dependent task must be complete before this one begins

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

// task 'one' is done now

});

gulp.task('default', ['one', 'two']);

参考：[gulp官网](https://github.com/gulpjs/gulp/blob/master/docs/API.md)