1. 什么是构建工具

一句话：自动化。对于需要反复重复的任务，例如压缩（minification）、合并、编译、单元测试、lint代码质量检测等，自动化工具可以减轻你的劳动，简化你的工作。当**配置文件**中定义好任务后，任务运行器就会自动帮你或你的小组完成大部分无聊的工作。

Web中的构建工具：其实就是简化开发，提高效率。

1. 构建工具之Grunt
   1. Grunt是什么



Grunt 是一个基于Nodejs平台的自动化构建工具（任务执行工具），简化开发，提供效率。

官方网站：<http://www.gruntjs.net/>

* 1. Grunt的优势

Grunt生态系统非常庞大，并且一直在增长。由于拥有数量庞大的插件（压缩，合并，less...）可供选择，因此，你可以利用Grunt自动完成任何事，并且花费最少的代价。如果找不到你所需要的插件，那就自己动手创造一个Grunt插件，然后将其发布到npm上吧。

* 1. Grunt入门
     1. 快速入门

Grunt和 Grunt 插件是通过 npm 安装并管理的，npm是 Node.js 的包管理器。

Grunt 0.4.x 必须配合Node.js >= 0.8.0版本使用。；奇数版本号的 Node.js 被认为是不稳定的开发版。

命令：

node -v 查看node当前版本

npm -v 查看npm当前版本

* + 1. 安装Grunt

Grunt是一个命令行工具，所以你需要先将Grunt命令行（CLI）安装到全局环境中。安装时可能需要使用sudo（针对OSX、\*nix、BSD等系统中）权限或者作为管理员（对于Windows环境）来执行以下命令。

**npm i -g grunt-cli**

上述命令执行完后，grunt 命令就被加入到你的系统路径中了，以后就可以在任何目录下执行(**grunt**)此命令了。

**注意：**安装grunt-cli并不等于安装了 Grunt！Grunt-cli只是运行grunt模块的一个命令，你需要在本地（自己的项目中:npm i grunt）进行安装。

* + 1. 准备一份新的 Grunt 项目

Grunt项目：其实就是一个普通的项目，只是里面加入了grunt模块和使用grunt任务。

* + 1. 加入Grunt必备文件

一般需要在你的项目中添加两份文件：**package.json 和 Gruntfile。**

一、package.json: 此文件被npm用于存储项目的元数据（项目的基本描述），以便将此项目发布为npm模块。你可以在此文件中列出项目依赖的grunt和Grunt插件，放置于devDependencies配置段内。

1. 可以通过npm init命令会创建一个基本的package.json文件
2. 拷贝一下模板， 与上面相比多出“devDependencies”开发依赖模块，这个列表中就包含了grunt和grunt任务。

{

"name": "my-project-name",

"version": "0.1.0",

**"devDependencies": {**

**"grunt": "~1.0.1",**

**"grunt-contrib-uglify": "~0.5.0"**

**}**

}

1. Gruntfile: 此文件被命名为 Gruntfile.js，用来配置或定义任务（task）并加载Grunt插件的。

**先创建一个Gruntfile.js文件与package.json在同一目录中，js文件中内容稍后添加。**

注意：package.json是本项目配置文件，Gruntfile是grunt的配置文件，必须分清楚。

三、安装项目中依赖的模块

**npm install （cnmp i）**

* + 1. Gruntfile文件案例

grunt虽然是自动化构建工具，但是grunt不知道需要执行那些任务（压缩，合并，混淆），并且grunt也不知道我们需要那些文件进行任务（压缩，合并，混淆）。

Gruntfile.js文件主要提供 “执行任务” 和 “待处理文件路径”的配置；

Gruntfile.js配置文件模板如下：(先无需关注内容的作用，后面详细讲解)

module.exports = function(grunt) {

// Project configuration.

grunt.initConfig({

pkg: grunt.file.readJSON('package.json'),

uglify: {

options:{

banner:'/\* 代码混淆（丑化） 模块名：<%= pkg.name %> 时间：<%= grunt.template.today("yyyy-mm-dd") %> \*/\n'

},

build:{

files:{

"dest/index.min.js":"js/index.js"

}

}

}

});

// 加载包含 "uglify" 任务的插件。

grunt.loadNpmTasks('grunt-contrib-uglify');

// 默认被执行的任务列表。

grunt.registerTask('default', ['uglify']);

};

* + 1. 运行grunt任务

grunt 或者 grunt 任务名 或者 grunt 任务名:目标

grunt //执行默认命令

或者

grunt uglify//执行具体某个命令

* 1. Gruntfile配置文件
     1. gruntfile结构

Gruntfile.js文件其实就是一个普普通通的模块，还记得“nodeJs中一个js文件就是一个模块”。

模块导出对象是一个函数，该函数在我们执行grunt的时候被调用。

module.exports = function(grunt) {

//完成grunt配置和任务

}

* + 1. grunt.initConfig()初始化配置

initConfig:方法用于配置grunt参数和任务。

grunt.initConfig(**{**

**//配置参数和任务**

**}**)

项目配置文件对象：

pkg : **grunt.file.readJSON('package.json')**,

读取package.json中的数据交给pkg属性.

pkg = {

"name": "my-project-name",

"version": "0.1.0",

**"devDependencies": {**

**"grunt": "~1.0.1",**

**"grunt-contrib-uglify": "~0.5.0"**

**}**

}

uglify混淆任务配置：

uglify: **{**

**options:{**

**banner:'/\* 代码混淆（丑化） 模块名：<%= pkg.name %> 时间：<%= grunt.template.today("yyyy-mm-dd") \*/\n'**

**},**

**build:{**

**files:{**

**"dest/index.min.js":"js/index.js"**

**}**

**}**

**}**

build:具体每一个混淆任务配置，build属性名可以任意编写。

files: 配置输出文件地址和源代码地址；

options:任务的全局配置参数。

banner：生成文件的头部内容；

banner中其实就是一个字符串，但我们可以通过**<%= %>**动态的输出内容

比如：<%=pkg.name%>获取pkg对象的name属性值；

<%= grunt.template.today("yyyy-mm-dd") 获取当前时间；

* + 1. grunt.loadNpmTask加载任务

grunt.loadNpmTasks(“任务名”);

通过任务名加载grunt的任务。 类似于require(“模块名”);

* + 1. grunt.registerTask注册任务

//一个任务列表注册成一个注册任务，方便使用。 使用被注册的任务，就会执行列表中所有任务

grunt.registerTask('任务名', ['任务列表']);

//默认被执行的任务列表。

grunt.registerTask('default', ['uglify']);

* 1. Grunt配置细节
     1. initConfig

initConfig({})方法，设置grunt配置参数，initConfig方法需要接受一个JSON对象。

此配置主要是以任务名称命名的属性，也可以包含其他任意数据。一旦这些代表任意数据的属性与任务所需要的属性相冲突，就将被忽略。

grunt.initConfig({

concat: {

// 这里是concat任务的配置信息。

},

uglify: {

// 这里是uglify任务的配置信息

},

// 任意数据。

my\_property: 'whatever',

my\_src\_files: ['foo/\*.js', 'bar/\*.js'],

});

蓝色部分是任务配置，黑色部分时任意数据，任意数据可以通过<%=%>使用。

* + 1. 任务配置和目标

当运行一个任务时，Grunt会自动查找配置对象中的同名属性。

多任务（multi-task）可以通过任意命名的“目标（target）”来定义多个配置。在下面的案例中，uglify任务有名为foo和bar两个目标，而concat任务仅仅只有一个名为bar目标。

例如：grunt uglify，会在配置文件中查找uglify属性,并且执行下面的foo,bar目标。

grunt.initConfig({

concat: {

bar: {

// concat task "bar" target options and files go here.

},

},

uglify: {

foo: {

// concat task "foo" target options and files go here.

},

bar: {

// uglify task "bar" target options and files go here.

},

}

});

如果同时指定任务（task）和目标（target），例如grunt uglify:foo或者grunt uglify:bar，将只会处理指定目标（target）的配置。

* + 1. options属性

在一个任务配置中，options属性可以用来指定覆盖内置属性的默认值。此外，每一个目标（target）中还可以拥有一个专门针对此目标（target）的options属性。目标（target）级的平options将会覆盖任务级的options。

options对象是可选的，如果不需要，可以忽略。

grunt.initConfig({

concat: {

**options: {**

**// 这里是任务级的Options，覆盖默认值**

**},**

foo: {

**options: {**

**// "foo" target options may go here, overriding task-level options.**

**},**

},

bar: {

// No options specified; this target will use task-level options.

},

},

});

* + 1. 处理文件

由于大多的任务都是执行文件操作，Grunt有一个强大的抽象层用于声明任务应该操作哪些文件。

文件分为：源文件（src） , dest(目标文件)

源文件：需要处理的文件；

目标文件：被处理后输出到的文件；

比如：把源文件js/index.js文件混淆后输出到目标文件dest/index.min.js

src -----> grun 任务处理----> 目标文件

Grunt有好几种定义src-dest(源文件-目标文件)文件映射的方式，你可以根据需要选择。

* + 1. 简洁格式

grunt.initConfig({

pkg: grunt.file.readJSON('package.json'),

uglify: {

foo:{

**src**:"js/index.js",

**dest**:"dest/index.js"

},

bar:{

**src**:["js/a.js","js/b.js"],

**dest**:"dest/all.js"

}

}

});

uglify任务中foo目标含义：对象源文件js/index.js进行混淆然后输出到目标文件dest/index.js中。

uglify任务中bar目标含义：对象源文件[“js/a.js”,”js/b.js”]进行合并然后混淆在输出到目标文件dest/index.js中。

注意：uglify任务自带合并功能。独立的合并任务是grunt-contrib-concat

* + 1. 文件对象格式

简洁模式文件使用非常简单，但是有一个问题，每次只能设置一对src-dest.

对象格式形式支持每个目标对应多个src-dest形式的文件映射，属性名就是目标文件，源文件就是它的值(源文件列表则使用数组格式声明)。

grunt.initConfig({

pkg: grunt.file.readJSON('package.json'),

uglify: {

foo:{

src:"js/index.js",

dest:"dest/index.js"

},

bar:{

src:["js/a.js","js/b.js"],

dest:"dest/all.js"

},

**baz:{**

**files:{**

**"dest/all.js":["js/a.js","js/b.js"],**

**"dest/index.js":"js/index.js"**

**}**

**}**

}

});

犹豫是多个，所以我们这里必须使用files，然后每一个键值对（ 键:值 ）就是一个文件映射。

另外注意： 键是目标文件，值是源文件。

"dest/all.js":["js/a.js","js/b.js"] 含义：js/a.js和js/b.js进行合并，然后混淆输出到目标文件dest/all.js中。

最后一句:使用这种方式指定数个src-dest文件映射，但是不能够给每个映射指定附加的属性。

* + 1. 文件数组格式

grunt.initConfig({

pkg: grunt.file.readJSON('package.json'),

uglify: {

foo:{

src:"js/index.js",

dest:"dest/index.js"

},

bar:{

src:["js/a.js","js/b.js"],

dest:"dest/all.js"

},

baz:{

files:{

"dest/all.js":["js/a.js","js/b.js"],

"dest/index.js":"js/index.js"

}

},

**qux:{**

**files:[**

**{**

**src:"js/index.js",**

**dest:"dest/index.js"**

**},**

**{**

**src:["js/a.js","js/b.js"],**

**dest:"dest/all.js"**

**}**

**]**

**}**

}

});

这种形式支持每个目标对应多个src-dest文件映射，同时也允许每个映射拥有额外属性：

* + 1. 通配符模式

通常分别指定所有源文件路径是不切实际的，因此Grunt通过内置支持node-glob 和 minimatch 库来匹配文件名。

\* 匹配任意数量的字符，但不匹配 /

? 匹配单个字符，但不匹配 /

\*\* 匹配任意数量的字符，包括 /，只要它是路径中唯一的一部分

{} 允许使用一个逗号分割的“或”表达式列表

! 在模式的开头用于排除一个匹配模式所匹配的任何文件

每个人都需要知道的是：foo/\*.js将匹配位于foo/目录下的所有的.js结尾的文件；而foo/\*\*/\*js将匹配foo/目录以及其子目录中所有以.js结尾的文件。

此外, 为了简化原本复杂的通配符模式，Grunt允许指定一个数组形式的文件路径或者一个通配符模式。所有模式按顺序处理，模式处理的过程中，带有!前缀的模式所匹配的文件将不包含在结果集中。 而且其结果集中的每一项也是唯一的。

例如：

// 指定单个文件：

{src: 'foo/this.js', dest: ...}

// 指定一个文件数组：

{src: ['foo/this.js', 'foo/that.js', 'foo/the-other.js'], dest: ...}

// 使用一个匹配模式：

{src: 'foo/th\*.js', dest: ...}

// 一个独立的node-glob模式：

{src: 'foo/{a,b}\*.js', dest: ...}

// 也可以这样编写：

{src: ['foo/a\*.js', 'foo/b\*.js'], dest: ...}

// foo目录中所有的.js文件，按字母顺序排序：

{src: ['foo/\*.js'], dest: ...}

// 首先是bar.js，接着是剩下的.js文件，并按字母顺序排序：

{src: ['foo/bar.js', 'foo/\*.js'], dest: ...}

// 除bar.js之外的所有的.js文件，按字母顺序排序：

{src: ['foo/\*.js', '!foo/bar.js'], dest: ...// 按字母顺序排序的所有.js文件，但是bar.js在最后。

{src: ['foo/\*.js', '!foo/bar.js', 'foo/bar.js'], dest: ...}

// 模板也可以用于文件路径或者匹配模式中：

{src: ['src/<%= basename %>.js'], dest: 'build/<%= basename %>.min.js'}

// 它们也可以引用在配置中定义的其他文件列表：

{src: ['foo/\*.js', '<%= jshint.all.src %>'], dest: ...}

* + 1. 动态构建文件对象

当你希望处理大量的单个文件时，这里有一些附加的属性可以用来动态的构建一个文件列表。

expand 设置为true用于启用下面的选项：

cwd 所有src指定的匹配都将相对于此处指定的路径（但不包括此路径） 。

src 相对于cwd路径的匹配模式。

dest 目标文件路径前缀。

ext 对于生成的dest路径中所有实际存在文件，均使用这个属性值替换扩展名。

extDot 用于指定标记扩展名的英文点号的所在位置。可以赋值 'first' （扩展名从文件名中的第一个英文点号开始） 或 'last' （扩展名从最后一个英文点号开始），默认值为 'first' [添加于 0.4.3 版本]

flatten 从生成的dest路径中移除所有的路径部分。

rename 对每个匹配的src文件调用这个函数(在重命名后缀和移除路径之后)。dest和匹配的src路径将被作为参数传入，此函数应该返回一个新的dest值。 如果相同的dest返回不止一次，那么，每个返回此值的src来源都将被添加到一个数组中作为源列表。

* + 1. 模板

使用<%= %>分隔符指定的模板会在任务从它们的配置中读取相应的数据时将自动扩展扫描。

grunt.initConfig({

concat: {

sample: {

options: {

banner: '/\* <%= baz %> \*/\n', // '/\* abcde \*/\n'

},

src: ['<%= qux %>', 'baz/\*.js'], // [['foo/\*.js', 'bar/\*.js'], 'baz/\*.js']

dest: 'build/<%= baz %>.js', // 'build/abcde.js'

},

},

//用于任务配置模板的任意属性

foo: 'c',

bar: 'b<%= foo %>d', // 'bcd'

baz: 'a<%= bar %>e', // 'abcde'

qux: ['foo/\*.js', 'bar/\*.js'],

});

* 1. Grunt常用任务

清理文件和文件夹：grunt-contrib-clean。

module.exports = function(grunt) {

grunt.initConfig({

clean : {

build : ['.tmp', 'build'],

release : ['release']

}

});

grunt.loadNpmTasks('grunt-contrib-clean');

grunt.registerTask('default', ['clean']);

};

使用UglifyJS压缩js文件：grunt-contrib-uglify。

module.exports = function(grunt) {

// 项目配置

grunt.initConfig({

pkg : grunt.file.readJSON('package.json'),

uglify : {

options : {

banner : '/\*! <%= pkg.file %> <%= grunt.template.today("yyyy-mm-dd") %> \*/\n'

},

build : {

src : 'src/<%=pkg.file %>.js',

dest : 'dest/<%= pkg.file %>.min.js'

}

}

});

// 加载提供"uglify"任务的插件

grunt.loadNpmTasks('grunt-contrib-uglify');

// 默认任务

grunt.registerTask('default', ['uglify']);

};

合并文件： grunt-contrib-concat。

module.exports = function(grunt) {

// 项目配置

grunt.initConfig({

pkg : grunt.file.readJSON('package.json'),

concat : {

options : {

separator : ';'

},

dist : {

src : ['src/zepto.js', 'src/underscore.js', 'src/backbone.js'],

dest : 'dest/libs.js'

}

}

});

grunt.loadNpmTasks('grunt-contrib-concat');

// 默认任务

grunt.registerTask('default', ['concat']);

}

合并并压缩文件：

方式一、先合并再压缩，首先将三个文件合并成一个文件libs.js，然后再将文件libs.js压缩成libs.min.js

module.exports = function(grunt) {

// 项目配置

grunt.initConfig({

pkg : grunt.file.readJSON('package.json'),

concat : {

options : {

separator : ';'

},

dist : {

src : ['src/zepto.js', 'src/underscore.js', 'src/backbone.js'],

dest : 'dest/libs.js'

}

},

uglify : {

build : {

src : 'dest/libs.js',

dest : 'dest/libs.min.js'

}

}

});

grunt.loadNpmTasks('grunt-contrib-uglify');

grunt.loadNpmTasks('grunt-contrib-concat');

// 默认任务

grunt.registerTask('default', ['concat', 'uglify']);

}

方式二：先压缩后合并，其实这里只需要grunt-contrib-uglify一个插件就可以完成代码的压缩及合并。

module.exports = function(grunt) {

// 项目配置

grunt.initConfig({

pkg : grunt.file.readJSON('package.json'),

uglify : {

"target" : {

"files" : {

'dest/libs.min.js' : ['src/zepto.js', 'src/underscore.js', 'src/backbone.js']

}

}

}

});

grunt.loadNpmTasks('grunt-contrib-uglify');

// 默认任务

grunt.registerTask('default', ['uglify']);

}

grunt-contrib-copy：复制文件和文件夹。

grunt-contrib-cssmin：CSS文件压缩。

grunt-contrib-imagemin：图片压缩。

grunt-usemin：Replaces references to non-optimized scripts or stylesheets into a set of HTML files (or any templates/views).

grunt-contrib-htmlmin：Minify HTML

1. 构建工具之Gulp
   1. gulp是什么

Gulp是自动化构建工具，使用它可以增强你的工作流程！



**http://www.gulpjs.com.cn/**

* 1. gulp优势

1. 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。

（Gulp管理任务非常简单）

1. 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。

（采用管道流方式，进行加快编译速度）

1. 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。

（高质量的扩展功能）

1. 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。

（上手容易，易于理解）

1. 文档齐全

<http://www.gulpjs.com.cn/docs/>

* 1. 入门指南

1. 全局安装 gulp：

npm install --global gulp

2. 作为项目的开发依赖（devDependencies）安装：

npm install --save-dev gulp

3. 在项目根目录下创建一个名为 gulpfile.js 的文件：

var gulp = require('gulp');

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

// 将你的默认的任务代码放在这

});

1. 运行 gulp：

gulp

默认的名为 default 的任务（task）将会被运行，在这里，这个任务并未做任何事情。

想要单独执行特定的任务（task），请输入 gulp <task> [<othertask>]。

* 1. 插件使用

gulp任务定义非常简单，但是，任务中需要去做具体的事情，比如：压缩、合并、编译less等操作，那么这些具体任务功能都是一个一个gulp插件。

* + 1. 查找 gulp 插件

gulp 插件总是会包含 **gulpplugin** 关键字。在这搜索 gulp 插件 或者 在 。

gulp插件地址 ：<http://gulpjs.com/plugins/>

npm 所有插件 ：<https://npmjs.org/browse/keyword/gulpplugin>

* + 1. gulp插件安装

使用gulp插件首先就的安装插件，插件安装：

cnpm i 插件名

cnpm i gulp-uglify --save-dev 安装压缩插件

* + 1. gulp插件的使用方式

gulp是基于流的自动化构建工具，采用管道流方式对文件进行处理（非所有，大部分）。 其中文件分为：src源文件, dest目标文件。

src源文件：被执行任务的文件。

dest目标文件：执行任务完毕后输出到的文件。

任务执行代码，如下：

gulp.**src("js/\*.js") //获取源文件列表**

.pipe(uglify()) //对象源文件执行压缩任务

.pipe(**gulp.dest("dest")**); //输出到dest目录

src与dest之间可以有多个pipe管道任务。

* + 1. gulp src源路径

gulp.src(path); 方法，设置源文件路径非常重要，可以设置一个明确的路径，比如：./js/index。但是更多时候我们需要通过匹配模式来简化路径的设置。

gulp支持node-glob匹配模式 ：<https://github.com/isaacs/node-glob>

\* 匹配任意数量的字符，但不匹配 /

? 匹配单个字符，但不匹配 /

\*\* 匹配任意数量的字符，包括 /，只要它是路径中唯一的一部分

{} 允许使用一个逗号分割的“或”表达式列表

**! 在模式的开头用于排除一个匹配模式所匹配的任何文件**

每个人都需要知道的是：foo/\*.js将匹配位于foo/目录下的所有的.js结尾的文件；而foo/\*\*/\*js将匹配foo/目录以及其子目录中所有以.js结尾的文件。

gulp.src(path);

gulp.src([path,path]);

* 1. gulp-uglify压缩js

//安装

cnpm i gulp-uglify

//导入gulp压缩插件

var uglify = require("gulp-uglify");

gulp.task("task1",,function(){

gulp.src("js/\*\*.js").pipe(

uglify()

).pipe(gulp.dest("dest"));

});

* 1. gulp-rename重命名

一般开发中，为了区分压缩和非压缩的文件，压缩文件后缀一般为.min。 此时就可以使用gulp-rename来修改名字。

//安装

cnpm i gulp-rename

//导入rename重命名插件

var rename = require("gulp-rename");

gulp.task("task2",function(){

//重名名某一个文件

gulp.src("./hello.txt")

**.pipe(rename("./aaaa.txt"))**

.pipe(gulp.dest("./dist")); // ./dist/aaaa.txt

});

gulp.task("task2",function(){

//通过函数动态修改文件路径

gulp.src("./hello.txt")

**.pipe(rename(function (path) {**

**path.dirname += "/dir";**

**path.basename += "-mm";**

**path.extname = "text"**

**}))**

.pipe(gulp.dest("./dist"));//dist/dir/hello-mm.text

});

gulp.task("task2",function(){

//通过配置对文件名进行设置

gulp.src("./hello.txt", { base: process.cwd() })

**.pipe(rename({**

**//dirname: "",//文件夹**

**//basename: "",//文件名（不含后缀）**

**prefix: "前缀-",//文件名的前缀**

**suffix: "-后缀",//文件名的后缀**

**extname: ".min.js"//扩展名**

**}))**

.pipe(gulp.dest("./dist")); // ./dist/前缀-hello-后缀.min.js

});

//常用方式

gulp.task("task2",function(){

gulp.src("./js/\*.js")

**.pipe(rename({**

**suffix: ".min",//文件名的后缀**

**}))**

. pipe(gulp.dest("./dist"));

});

* 1. gulp-clean-css 压缩css

安装

cnpm i gulp-clean-css --save-dev

//导入clean-css压缩插件

var cleanCss = require("gulp-clean-css");

gulp.task("task3",function(){

gulp.src("./css/\*.css")

**.pipe(cleanCss())**

.pipe(rename({suffix:".min"}))

.pipe(gulp.dest("./dest/css"));

});

* 1. gulp-imagemin 压缩图片

压缩图片可以减少，图片的体积，提高传输速度

安装

cnpm i gulp-imagemin --save-dev

//导入imagemin压缩图片

var imagemin = require("gulp-imagemin");

gulp.task('task4',function(){

gulp.src('src/images/\*')

.pipe(imagemin())

.pipe(gulp.dest('dist/images'))

});

* 1. gulp.watch监控哪些文件的变动

文件被修改的时候，我们就需要执行相关的任务，但是如果每次修改都要人为的来操作，未免过来繁琐，所以，gulp提供了一个watch方法，可以在文件被修改的时候通过回调执行任务。

//压缩js任务

gulp.task("uglifyJs",function(){

gulp.src("js/\*\*.js")

.pipe(uglify())

.pipe(rename({

suffix:""

}))

.pipe(gulp.dest("dest"))

});

//监测任务

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

**gulp.watch("js/\*.js",["uglifyJs"]);**

});

* 1. del 删除文件或文件夹

你也许会想要在编译文件之前删除一些文件。由于删除文件和文件内容并没有太大关系，所以，我们没必要去用一个 gulp 插件。del模块就可以满足我们的要求。

//安装

cnpm i del --save-dev

//导入删除模块

var del = require("del");

gulp.task("del",function(){

//采用同步进行删除

del.sync("./dest");

});

gulp.task("del",function(cb){

//采用异步删除，删除完毕后，调用cb回调函数，告诉gulp任务已经完成。

del("./dest").then(function(){

cb();

});

});

* 1. gulp 任务依赖

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

一个任务可以依赖其他任务，[deps]一个包含任务列表的数组，这些任务会在你当前任务运行之前完成。

你的任务是否在这些前置依赖的任务完成之前运行了？请一定要确保你所依赖的任务列表中的任务都使用了正确的异步执行方式：使用一个 callback，或者返回一个 promise 或 stream。

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

setTimeout(function(){

console.log("t1");

cb();//回调函数调用告知gulp任务执行完毕

},1000);

});

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

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

.pipe(minify())

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

return stream;

});

返回一个 promise

var Q = require('q');

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

var deferred = Q.defer();

// 执行异步的操作

setTimeout(function() {

deferred.resolve();

}, 1);

return deferred.promise;

});

* 1. run-sequence串行任务

gulp任务执行都是异步，也就是说如果多个任务是同时执行的，但是某些任务需要等待一个任务完成后，在继续执行下一个任务，那么我们就需要使用到run-sequence串行任务。

//安装

cnpm i --save-dev run-sequence

//导入sequence串行任务

var runSequence = require("run-sequence");

//任务一

gulp.task("t1",function(cb){

setTimeout(function(){

console.log("t1");

cb();//回调函数调用告知gulp任务执行完毕

},1000);

});

//任务二

gulp.task("t2",function(cb){

setTimeout(function(){

console.log("t2");

cb();//回调函数调用告知gulp任务执行完毕

},500);

});

//任务三

gulp.task("t3",function(cb){

setTimeout(function(){

console.log("t3");

cb();//回调函数调用告知gulp任务执行完毕

},200);

});

//定义串行gulp任务

gulp.task("run1",function(){

runSequence("t1","t2","t3")

});

gulp.task("run2",function(){

runSequence("t1",["t2","t3"])

//中括号括起来的表示可以异步执行的

});

* 1. gulp-livereload

//安装

cnpm i --save-dev gulp-livereload

//导入livereload实时刷新模块

var livereload = require("gulp-livereload");

//压缩js任务

gulp.task("uglifyJs",["del"],function(){

gulp.src("js/\*\*.js")

.pipe(uglify())

.pipe(rename({

suffix:""

}))

.pipe(gulp.dest("dest"))

**.pipe(livereload());**

});

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

**livereload.listen();**

gulp.watch("js/\*.js",["uglifyJs"]);

});

//安装Chrome Livereload插件



Chrome工具栏会出现这个图标



点击，变为实心圆点，代表自动刷新可以用



* 1. gulp-webserver

//安装

cnpm i gulp-webserver --save-dev

//导入

var webserver = require('gulp-webserver');

// 使用任务

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

gulp.src( './' ) // 服务器目录（./代表根目录）

.pipe(webserver({ // 运行gulp-webserver

livereload: true, // 启用LiveReload

open: true // 服务器启动时自动打开网页

}));

});

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

* 1. gulp常用插件

No.1、run-sequence

Links: https://www.npmjs.com/package/run-sequence

作用：让gulp任务，可以相互独立，解除任务间的依赖，增强task复用

推荐指数：★★★★★

No.2、browser-sync

Links: http://www.browsersync.io/

作用：静态文件服务器，同时也支持浏览器自动刷新

推荐指数：★★★★★

No.3、del

Links：https://www.npmjs.com/package/del

作用：删除文件/文件夹

推荐指数：★★★★★

No.4、gulp-coffee

Links: https://github.com/wearefractal/gulp-coffee

作用：编译coffee代码为Js代码，使用coffeescript必备

推荐指数：★★★★

No.5、coffee-script

Links: https://www.npmjs.com/package/coffee-script

作用：gulpfile默认采用js后缀，如果要使用gulpfile.coffee来编写，那么需要此模块

推荐指数：★★★

No.6、gulp-nodemon

Links: https://www.npmjs.com/package/gulp-nodemon

作用：自动启动/重启你的node程序，开发node服务端程序必备

推荐指数：★★★★★

No.7、yargs

Links: https://www.npmjs.com/package/yargs

作用：用于获取启动参数，针对不同参数，切换任务执行过程时需要

推荐指数：★★★

No.8、gulp-util

Links: https://www.npmjs.com/package/gulp-util

作用：gulp常用的工具库

推荐指数：★★★★★

No.9、gulp-uglify

Links: https://www.npmjs.com/package/gulp-uglify

作用：通过UglifyJS来压缩JS文件

推荐指数：★★★★

No.9、gulp-concat

Links: https://www.npmjs.com/package/gulp-concat

作用：合并JS

推荐指数：★★★★

No.10、gulp-sourcemaps

Links: https://www.npmjs.com/package/gulp-sourcemaps

作用：处理JS时，生成SourceMap

推荐指数：★★★★

No.11、gulp-less

Links：https://www.npmjs.com/package/gulp-less

作用：将less预处理为css

推荐指数：★★★★

No.12、gulp-sass

Links：https://www.npmjs.com/package/gulp-sass

作用：将sass预处理为css

推荐指数：★★★★

No.13、gulp-autoprefixer

Links：https://www.npmjs.com/package/gulp-autoprefixer

作用：使用Autoprefixer来补全浏览器兼容的css。

推荐指数：★★★★

No.14、gulp-minify-css

Links：https://www.npmjs.com/package/gulp-minify-css

作用：压缩css。

推荐指数：★★★★

No.15、connect-history-api-fallback

Links：https://www.npmjs.com/package/connect-history-api-fallback

作用：开发angular应用必须，用于支持HTML5 history API.

推荐指数：★★★

1. Grunt Gulp 样板

gulp <https://github.com/cferdinandi/gulp-boilerplate>

grunt <https://github.com/Integralist/Grunt-Boilerplate>