Contents

[Yeoman (http://yeoman.io/) 2](#_Toc137582148)

[angular + bootstrap + sass (please refer to http://yeoman.io/codelab/index.html) 2](#_Toc137582149)

[pre-condition: node, ruby/sass/compass and git 2](#_Toc137582150)

[project template 3](#_Toc137582151)

[less 6](#_Toc137582152)

[gulp 6](#_Toc137582153)

[creating gulp project 7](#_Toc137582154)

[gulpfile.js 7](#_Toc137582155)

[task 7](#_Toc137582156)

[compile Sass to CSS in Gulp-- gulp-sass 7](#_Toc137582157)

[Watching Sass files for changes 8](#_Toc137582158)

[Live-reloading with Browser Sync 8](#_Toc137582159)

[Optimizing CSS and JavaScript files 9](#_Toc137582160)

[Cleaning up generated files automatically 11](#_Toc137582161)

[Combining Gulp tasks 12](#_Toc137582162)

[gulp命令行 13](#_Toc137582163)

[var gulp = require('gulp'); 13](#_Toc137582164)

[$ npm install --save-dev gulp-load-plugins 13](#_Toc137582165)

[$npm install --save-dev gulp-inject 13](#_Toc137582166)

[$npm install --save-dev gulp-plumber 16](#_Toc137582167)

[$npm install --save-dev gulp-sourcemaps 17](#_Toc137582168)

[$npm install --save-dev gulp-angular-filesort 17](#_Toc137582169)

[$ npm install --save wiredep 17](#_Toc137582170)

Sublime

Hide files with certain extension in sublime?

<http://stackoverflow.com/questions/17632108/hide-files-with-certain-extension-in-sublime-text-editor>

Preferences → Settings – User

{

"file\_exclude\_patterns":

}

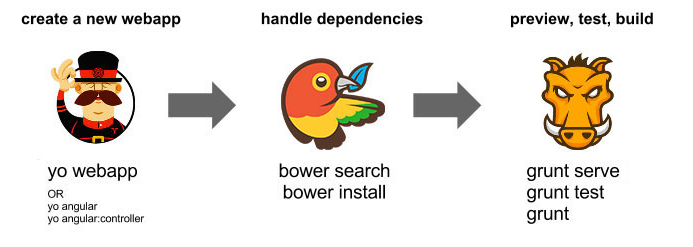
# Yeoman (http://yeoman.io/)

Yeoman = YO(脚手架工具) + GRUNT(构建工具) + BOWER(包管理器)

**YO ：Yeoman核心工具，项目工程依赖目录和文件生成工具，项目生产环境和编译环境生成工具，（创建项目模板，自带web server, live reload, compile sass, unit test, minimize code, optimize images）**

**BOWER ：Web开发的包管理器**，概念上类似npm，npm专注于nodeJs模块，而bower专注于CSS、JavaScript、图像等前端相关内容的管理。需要注意的是，Bower的运行，依赖于版本控制工具git，从github拉取以来信息。 如《Node.js介绍》所说，很多前端工具，都是由Node.js所编写的，Bower也不例外。所以要想成功安装Yeoman,需先安装 Git。

**GRUNT ：前端构建工具**，jquery就是使用这个工具打包的。(C/C++程序通过makefile管理编译测试打包的过程，Java程序通过gradle, Maven,Ant实现项目构建管理功能，Python有pip，Ruby有gem。在Nodejs的领域，我们同样需要一个项目构建工具，这就是Grunt。Grunt可以执行像压缩, 编译, 单元测试, 代码检查以及打包发布的任务)



实战: (please refer to : <http://yeoman.io/>)

generator-gulp-angular

>npm install -g yo gulp bower

>npm install -g generator-gulp-angular

myapp>yo gulp-angular

# angular + bootstrap + sass (please refer to <http://yeoman.io/codelab/index.html>)

## pre-condition: node, ruby/sass/compass and git

**set sys env: ruby\bin; npm; git\bin**

after install npm, set proxy for npm

**> npm config set proxy** [**http://165.225.96.34:10015**](http://165.225.96.34:10015)

**> npm config set https-proxy** [**http://165.225.96.34:10015**](http://165.225.96.34:10015)

(验证：>npm config get proxy)

1) install yo and other required tools

**> npm install -g yo bower grunt-cli gulp**

after install bower, set proxy for bower

set env

**HTTP\_PROXY =** [**http://165.225.96.34:10015**](http://165.225.96.34:10015)

**HTTPS\_PROXY =** [**http://165.225.96.34:10015**](http://165.225.96.34:10015)

Please restart cmd to make setting effective

或者

在.bowerrc文件添加代理

{"directory": "bower\_components",

"registry": "http://bower.herokuapp.com",

"proxy": "http:// 165.225.96.34:10050/",

"https-proxy": "http://161.92.51.225:8080/"}

或者

C:\Users\310031267\AppData\Roaming\npm\node\_modules\bower\node\_modules\bower-config\lib\util\default.js

"proxy": "http://161.92.51.225:8080/",

"https-proxy": "http://161.92.51.225:8080/"

set env

HTTP\_PROXY = http://161.92.51.225:8080/

HTTPS\_PROXY = http://161.92.51.225:8080/

## project template

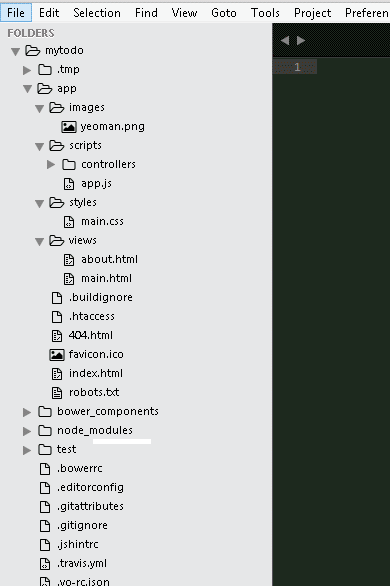
**project-parent-directory> mkdir mytodo && cd mytodo**

**mytodo > node --version && npm --version && git --version**

**mytodo > yo --version && bower --version && grunt --version**

**mytodo > npm install -g generator-angular**

**mytodo > yo angular**



In mytodo, we have:

*app:* a parent directory for our web application

index.html: the base html file for our Angular app

*404.html, favicon.ico, and robots.txt:* commonly used web files so you don’t have to create them yourself

*scripts:* our own JS files

*app.js:* our main Angular application code

*controllers:* our Angular controllers

*styles:* our CSS files

*views:* a place for our Angular templates

*bower\_components, bower.json:* our JavaScript/web dependencies, installed by Bower

*Gruntfile.js, package.json, and node\_modules:* configuration and dependencies required by our Grunt tasks

*test:* a scaffolded out test runner and the unit tests for the project, including boilerplate tests for our controllers.

**mytodo > grunt serve //start the server (local Node-based http server on localhost:9000**

若port:9000被占用，用如下脚本杀死

cmd> netstat –ano|findstr 9000 //查看谁占用9000端口

//杀死占用9000的应用程序

cmd> taskkill /pid 6856 /f //其中6865是使用9000端口的进程

从而可以在浏览器访问localhost:9000

**mytodo > ctrl + c**  // terminate

Note: could not delete file “ because the path is too long?” (because npm nests dependencies)

**mytodo> npm install –g rimraf**

**mytodo> rimraf node\_modules**

// use bower to install packages

**mytodo > bower list** // list current packages

**mytodo > bower search angular-ui-sortable** // search for packages

**mytodo > bower search jquery-ui**

现有项目当中，添加包

**mytodo> bower install --save angular-ui-sortable jquery-ui**

则会下载包到bower\_components文件夹下，且--save会自动更新bower.json,执行grunt serve会自动更新index.html, 执行grunt test会自动更新test/karma.conf.js

//write your code

modify views/\*.html

styles/\*.scss

scripts/app.js

controllers/\*.js

(note: yeoman will autowatch and update browser)

//write unit test

modify test/spec/controller/\*.js

mytodo > grunt test

(note: if error “jit-grunt plugin for the karma task not found”, please install)

mytodo > npm install grunt-karma karma-phantomjs-launcher karma-jasmine --save-dev

//deploy

lint our code, run our tests, concatenate and minify our scripts and styles to save on those network requests, optimize images if we were using any, compile the output of any preprocessors we are using, and generally make our application really lean

mytodo >**grunt**

(note:if error “Running imagemin:dist task failed, use grunt –force)

dist文件夹整个作为应用，可以发布到Server上

mytodo >**grunt serve:dist** //启动服务，运行dist

//创建自定义模块

bower\_components/common/scripts/filters/customFilter.js

angular.module(“customFilters”, []) ; //自定义模块

index.html:

<script src=” bower\_components/common/scripts/filters/customFilters.js”></script>

app.js:

angular.module(‘mytodoApp’, [‘customFilters’]) ; //声明依赖

//给模块mytodoApp添加控制器，过滤器, 指令

mytodo > **yo angular:controller form**

**mytodo > yo angular: filter form**

**mytodo > yo angular:directive form**

**mytodo > yo angular:view form**

// npm clean cache, bower clean cache

mytodo> **npm cache clean**

mytodo> **bower cache clean**

# less

download and install WinLess from <http://winless.org>

import project bootstrap

right click less/bootstrap.less -> choose “select output file” -> output file name “bootstrap.css” -> compile

# gulp

<https://css-tricks.com/gulp-for-beginners/>

Gulp configurations tend to be much shorter and simpler when compared with Grunt. Gulp also tends to run faster.

* Spins up a web server
* Compiles Sass to CSS
* Refreshes the browser automatically whenever you save a file
* Optimizes all assets (CSS, JS, fonts, and images) for production

## creating gulp project

install node.js contains NPM(Node Package Manger)

$sudo npm install gulp -g

$npm init

==package.json

$npm install gulp --save-dev

We've added --save-dev, which tells the computer to add gulp as a dev dependency in package.json

## gulpfile.js

### task

a real task may look like:

gulp.task('task-name', function () {

return gulp.src('source-files') // Get source files with gulp.src

.pipe(aGulpPlugin()) // Sends it through a gulp plugin

.pipe(gulp.dest('destination')) // Outputs the file in the destination folder

})

### compile Sass to CSS in Gulp-- gulp-sass

Gulp-sass uses LibSass to convert Sass into CSS. It's much quicker than Ruby-based methods.

$ npm install gulp-sass --save-dev

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

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

return gulp.src('app/scss/styles.scss')

.pipe(sass()) // Converts Sass to CSS with gulp-sass

.pipe(gulp.dest('app/css'))

});

$gulp sass 运行task

\*\*/\*.scss: This is a more extreme version of the \* pattern that matches any file ending with .scss in the root folder and any child directories.

!not-me.scss: The ! indicates that Gulp should exclude the pattern from its matches, which is useful if you had to exclude a file from a matched pattern. In this case, not-me.scss would be excluded from the match.

\*.+(scss|sass): The plus + and parentheses () allows Gulp to match multiple patterns, with different patterns separated by the pipe | character. In this case, Gulp will match any file ending with .scss or .sass in the root folder.

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

return gulp.src('app/scss/\*\*/\*.scss') // Gets all files ending with .scss in app/scss and children dirs

.pipe(sass())

.pipe(gulp.dest('app/css'))

})

### Watching Sass files for changes

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

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

// Other watchers

})

$gulp watch

这时候编辑\*.scss，保存后，会自动执行sass任务

### Live-reloading with Browser Sync

$ npm install browser-sync --save-dev

var browserSync = require('browser-sync').create();

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

browserSync.init({

server: {

baseDir: 'app' // let Browser Sync know where the root of the server should be

},

})

})

Browser Sync can inject new CSS styles (update the CSS) into the browser whenever the sass task is ran.

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

return gulp.src('app/scss/\*\*/\*.scss') // Gets all files ending with .scss in app/scss

.pipe(sass())

.pipe(gulp.dest('app/css'))

.pipe(browserSync.reload({

stream: true

}))

});

Gulp should start both the sass and browserSync tasks concurrently. When both tasks are completed, watch will run

gulp.task('watch', ['browserSync', 'sass'], function (){

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

// Other watchers

});

adding two more watch processes, and calling the browserSync.reload function when a file gets saved

gulp.task('watch', ['browserSync', 'sass'], function (){

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

// Reloads the browser whenever HTML or JS files change

gulp.watch('app/\*.html', browserSync.reload);

gulp.watch('app/js/\*\*/\*.js', browserSync.reload);

});

### Optimizing CSS and JavaScript files

#### concatenate your scripts in the correct order.

Gulp-useref concatenates any number of CSS and JavaScript files into a single file by looking for a comment that starts with "<!--build:" and ends with "<!--endbuild-->". Its syntax is:

<!-- build:<type> <path> -->

... HTML Markup, list of script / link tags.

<!-- endbuild -->

<type> can either be js, css, or remove

<path> here refers to the target path of the generated file.

$ npm install gulp-useref --save-dev

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

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

return gulp.src('app/\*.html')

.pipe(useref())

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

});

for example:

<!--build:js js/main.min.js -->

<script src="js/lib/a-library.js"></script>

<script src="js/lib/another-library.js"></script>

<script src="js/main.js"></script>

<!-- endbuild -->

will be concatenated to

dist/js/main.min.js

#### minify

$ npm install gulp-uglify --save-dev

var gulpIf = require('gulp-if');

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

return gulp.src('app/\*.html')

.pipe(useref())

// Minifies only if it's a JavaScript file

.pipe(gulpIf('\*.js', uglify()))

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

});

minify and concatenate css files

$ npm install gulp-cssnano

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

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

return gulp.src('app/\*.html')

.pipe(useref())

.pipe(gulpIf('\*.js', uglify()))

// Minifies only if it's a CSS file

.pipe(gulpIf('\*.css', cssnano()))

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

});

#### optimize images

$ npm install gulp-imagemin --save-dev

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

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

return gulp.src('app/images/\*\*/\*.+(png|jpg|gif|svg)')

.pipe(imagemin())

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

});

Optimizing images however, is an extremely slow process that you'd not want to repeat unless necessary. To do so, we can use the gulp-cache plugin.

$ npm install gulp-cache --save-dev

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

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

return gulp.src('app/images/\*\*/\*.+(png|jpg|jpeg|gif|svg)')

// Caching images that ran through imagemin

.pipe(cache(imagemin({

interlaced: true

})))

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

});

#### Copying Fonts to Dist

Since font files are already optimized, there's nothing more we need to do. All we have to do is to copy the fonts into dist.

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

return gulp.src('app/fonts/\*\*/\*')

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

})

### Cleaning up generated files automatically

The del function takes in an array of node globs which tells it what folders to delete.

Note: We don't have to worry about deleting the dist/images folder because gulp-cache has already stored the caches of the images on your local system.

npm install del --save-dev

var del = require('del');

gulp.task('clean:dist', function() {

return del.sync('dist');

})

To clear the caches off your local system, you can create a separate task that's named `cache:clear`

gulp.task('cache:clear', function (callback) {

return cache.clearAll(callback)

})

### Combining Gulp tasks

a development process, where we compiled Sass to CSS, watched for changes, and reloaded the browser accordingly.

gulp.task('watch', ['browserSync', 'sass'], function (){

// ... watchers

})

an optimization process, where we ready all files for the production website. We optimized assets like CSS, JavaScript, and images in this process and copied fonts over from app to dist.

$ npm install run-sequence --save-dev

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

gulp.task('task-name', function(callback) {

runSequence('task-one', 'task-two', 'task-three', callback);

});

Gulp will run task-one first. When task-one finishes, Gulp will automatically start task-two

gulp.task('task-name', function(callback) {

runSequence('task-one', ['tasks','two','run','in','parallel'], 'task-three', callback);

});

Gulp first runs task-one. When task-one is completed, Gulp runs every task in the second argument simultaneously. All tasks in this second argument must be completed before task-three is run.

gulp.task('build', function (callback) {

runSequence('clean:dist',

['sass', 'useref', 'images', 'fonts'],

callback

)

})

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

runSequence(['sass','browserSync', 'watch'],

callback

)

})

## gulp命令行

$gulp --version

$gulp --gulpfile gulpfile\_test.js 手动指定 gulpfile路径

$gulp --tasks 显示所指定gulpfile的task依赖树

$gulp --tasks-simple 显示所载入gulpfile中的task列表

### var gulp = require('gulp');

gulp.src(globs[, options])

gulp.dest(path[, options])

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

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

gulp.start('watch');

### $ npm install --save-dev gulp-load-plugins

assume package.json

{

"dependencies": {

"gulp-jshint": "\*",

"gulp-concat": "\*"

}

}

var plugins = require('gulp-load-plugins')();

等价于

plugins.jshint = require('gulp-jshint');

plugins.concat = require('gulp-concat');

### $npm install --save-dev gulp-inject

Each pair of comments are the injection placeholders

<https://www.npmjs.com/package/gulp-inject>

**src/index.html:**

<!DOCTYPE html>

<html>

<head>

<title>My index</title>

<!-- inject:css -->

<!-- endinject -->

</head>

<body>

<!-- inject:js -->

<!-- endinject -->

</body>

</html>

**The gulpfile.js:**

var gulp = require('gulp');

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

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

var target = gulp.src('./src/index.html');

var sources = gulp.src(['./src/\*\*/\*.js', './src/\*\*/\*.css'], {read: false});

return target.pipe(inject(sources))

.pipe(gulp.dest('./src'));

});

**src/index.html after running gulp index:**

<!DOCTYPE html>

<html>

<head>

<title>My index</title>

<!-- inject:css -->

<link rel="stylesheet" href="/src/style1.css">

<link rel="stylesheet" href="/src/style2.css">

<!-- endinject -->

</head>

<body>

<!-- inject:js -->

<script src="/src/lib1.js"></script>

<script src="/src/lib2.js"></script>

<!-- endinject -->

</body>

</html>

**Injecting files relative to target files**

Project structure:

└── src

├── module

│ ├── module.js

│ └── module.html

└── app

├── main.js

└── index.html

**src/app/index.html:**

<!DOCTYPE html>

<html>

<head>

<title>My Index</title>

</head>

<body>

<h1>Home</h1>

<!-- inject:js -->

<!-- endinject -->

</body>

</html>

**src/module/module.html:**

<!DOCTYPE html>

<html>

<head>

<title>Module</title>

</head>

<body>

<h1>Module</h1>

<!-- inject:js -->

<!-- endinject -->

</body>

</html>

gulpfile.js:

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

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

.pipe(**inject(gulp.src('./src/\*\*/\*.js', {read: false}), {relative: true})**)

.pipe(gulp.dest('./src'));

Resulting src/app/index.html:

<!DOCTYPE html>

<html>

<head>

<title>My Index</title>

</head>

<body>

<h1>Home</h1>

<!-- inject:js -->

<script src="main.js"></script>

<script src="../module/module.js"></script>

<!-- endinject -->

</body>

</html>

Resulting src/module/module.html:

<!DOCTYPE html>

<html>

<head>

<title>Module</title>

</head>

<body>

<h1>Home</h1>

<!-- inject:js -->

<script src="../app/main.js"></script>

<script src="module.js"></script>

<!-- endinject -->

</body>

</html>

Injecting files from multiple source streams

$npm install --save-dev event-stream

var series = require('stream-series'),

inject = require('gulp-inject');

var vendorStream = gulp.src(['./src/vendors/\*.js'], {read: false});

var appStream = gulp.src(['./src/app/\*.js'], {read: false});

gulp.src('./src/index.html')

.pipe(inject(series(vendorStream, appStream))) // This will always inject vendor files before app files

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

### $npm install --save-dev gulp-plumber

Briefly it replaces pipe method and removes standard onerror handler on error event, which unpipes streams on error by default.

var $ = require('gulp-load-plugins')();

gulp.src('app/\*/styles/!(\_)\*.scss')

.pipe($.plumber())

### $npm install --save-dev gulp-sourcemaps

write inline source maps, inline source maps are embedded in the source file

gulp.src('app/\*/styles/!(\_)\*.scss')

.pipe($.plumber())

.pipe($.sourcemaps.init())

.pipe($.sass.sync().on('error', $.sass.logError))

.pipe($.sourcemaps.write())

$npm install --save-dev gulp-natural-sort

Sort stream by path name using a natural sort

gulp.src(paths.jsFiles)

.pipe($.plumber()) // use plumber so watch can start despite js errors

.pipe($.naturalSort())

### $npm install --save-dev gulp-angular-filesort

Automatically sort AngularJS app files depending on module definitions and usage

Used in conjunction with gulp-inject to inject your AngularJS application files (scripts) in a correct order, to get rid of all Uncaught Error: [$injector:modulerr]. To work correctly, each angular file needs to have a uniquely named module and setter syntax (with the brackets), i.e. angular.module('myModule', []).

var angularFilesort = require('gulp-angular-filesort'),

inject = require('gulp-inject');

gulp.src('./src/app/index.html')

.pipe(inject(

gulp.src(['./src/app/\*\*/\*.js']).pipe(angularFilesort())

))

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

### $ npm install --save wiredep

Wire Bower dependencies to your source code.

<html>

<head>

**<!-- bower:css -->**

**<!-- endbower -->**

</head>

<body>

**<!-- bower:js -->**

**<!-- endbower -->**

</body>

</html>

将bower\_components依赖包注入index.html中

// inject bower components into index.html

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

return gulp.src('app/index.html')

// exclude ionic scss since we're using ionic sass

.pipe(**wiredep.stream({exclude: ['bower\_components/ionic/release/css']})**)

.pipe(gulp.dest('app/'));

});

main-bower-files

mainBowerFiles returns an array of files

var gulp = require('gulp');

var mainBowerFiles = require('main-bower-files');

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

return gulp.src(mainBowerFiles(/\* options \*/), { base: 'path/to/bower\_components' })

.pipe(/\* what you want to do with the files \*/)

});

$ npm install --save-dev gulp-changed

Only pass through changed files, compare the current files with the destination files.

return gulp.src(fontFiles)

.pipe($.changed(DEST))

.pipe(gulp.dest(DEST));