Contents

[代码静态检查器 1](#_Toc456692104)

[Interaction between frontend and backend 2](#_Toc456692105)

[NPM (node package manager) 2](#_Toc456692106)

[introduction 2](#_Toc456692107)

[How to use npm with wall? 2](#_Toc456692108)

[how to use gem with wall? 3](#_Toc456692109)

[Git download 3](#_Toc456692110)

[Babel 3](#_Toc456692111)

[JavaScript Source Map 3](#_Toc456692112)

[Yeoman (http://yeoman.io/) 5](#_Toc456692113)

[angular + bootstrap + sass (please refer to http://yeoman.io/codelab/index.html) 5](#_Toc456692114)

[pre-condition: node, ruby/sass/compass and git 5](#_Toc456692115)

[project template 6](#_Toc456692116)

[Sass & Compass 9](#_Toc456692117)

[Pre-condition: ruby\bin 9](#_Toc456692118)

[ install sass and compass 9](#_Toc456692119)

[create compass framework with 960: 10](#_Toc456692120)

[create compass framework with Bootstrap: 10](#_Toc456692121)

[less 10](#_Toc456692122)

[gulp 10](#_Toc456692123)

[creating gulp project 11](#_Toc456692124)

[gulpfile.js 11](#_Toc456692125)

[task 11](#_Toc456692126)

[compile Sass to CSS in Gulp-- gulp-sass 11](#_Toc456692127)

[Watching Sass files for changes 12](#_Toc456692128)

[Live-reloading with Browser Sync 12](#_Toc456692129)

[Optimizing CSS and JavaScript files 13](#_Toc456692130)

[Cleaning up generated files automatically 15](#_Toc456692131)

[Combining Gulp tasks 16](#_Toc456692132)

# 代码静态检查器

how to use jslint?

cmd:

E:\javascript\WebstormProject\learnJQueryUI> **java -jar jslint4java-2.0.4.jar** *spa/js/spa.js*

## Interaction between frontend and backend

via tool curl

post data as follows:

cmd:

E:\ javascript\docs\curl\curl-7.33.0-win64-nossl>curl localhost:3000/user/create -d {}

via Chrome Post man

# NPM (node package manager)

## introduction

大多数程序平台都有一个用来下载、安装，管理第三方模块的系统，在Node里，我们使用Node包管理器（NPM: Node Package Manager）

NPM包含三部分：一个用来存放第三方包的代码库，一个管理本地已经安装包的机制，一个用来定义包依赖关系的标准。**NPM提供了一个公共的注册服务，它包含了大家发布的所有包，并提供了一个命令行工具，用来下载，安装和管理这些包**。你可以按照Node的包格式标准来制定你的包或者应用需要依赖的其他第三方包。

使用NPM来安装，升级和卸载包

NPM的操作主要有两种模式：全局和本地。这两种模式会影响包存放的目录结构，以及Node加载包时的顺序。本地模式是NPM的默认操作模式，在这个模式下，NPM只工作在工作目录下，不会造成系统范围的修改，这个模式让你在某个Node程序下尽情地安装，测试模块，而不会影响你电脑上的其他Node程序。

全局模式适合那些将被很多程序使用，而且总是被全局加载的公共模块，比如命令行工具这些公不会被应用程序直接使用的模块。在全局模式下面，NPM会把包安装到/usr/local/lib/node\_modules

## How to use npm with wall?

使用国内淘宝镜像:

E:\Work\Script\Javascript\WebstormProject\spa> npm --registry=https://registry.npm.taobao.org

E:\Work\Script\Javascript\WebstormProject\spa> npm install

## how to use gem with wall?

Install rubyinstaller-2.1.5.exe

>gem sources --a <https://ruby.taobao.org/> --remove https://rubygems.org/

>gem install sass

(若需要proxy>gem install sass –http-proxy=http://165.225.96.34:10015)

>gem install compass

# Git download

1. 某folder下右击git bash
2. 若有代理需要设置 $ git config –global http.proxy 161.92.64.42:8080

Or ENV HTTP\_PROXY

1. $ git clone –depth=14 <https://github.com/angular/angular-phonecat.git>

Download npm tool dependencies

1. Cd 某个angularjs 应用程序文件
2. npm config set proxy <http://161.92.64.42:8080>
3. npm install

# Babel

## JavaScript Source Map

参考：<http://www.ruanyifeng.com/blog/2013/01/javascript_source_map.html>

常见的源码转换，主要是以下三种情况：

　　（1）压缩，减小体积。比如jQuery 1.9的源码，压缩前是252KB，压缩后是32KB。

　　（2）多个文件合并，减少HTTP请求数。

　　（3）其他语言编译成JavaScript。最常见的例子就是CoffeeScript。

这三种情况，都使得实际运行的代码不同于开发代码，除错（debug）变得困难重重。

通常，JavaScript的解释器会告诉你，第几行第几列代码出错。但是，这对于转换后的代码毫无用处。举例来说，jQuery 1.9压缩后只有3行，每行3万个字符，所有内部变量都改了名字。你看着报错信息，感到毫无头绪，根本不知道它所对应的原始位置。

Source map就是一个信息文件，里面储存着位置信息。也就是说，转换后的代码的每一个位置，所对应的转换前的位置。

有了它，出错的时候，除错工具将直接显示原始代码，而不是转换后的代码。

目前，暂时只有Chrome浏览器支持这个功能。在Developer Tools的Setting设置中，确认选中"Enable source maps"。

如何启用Source map

只要在转换后的代码尾部，加上一行就可以了。

//@ sourceMappingURL=/path/to/file.js.map

如何生成Source map

最常用的方法是使用Google的Closure编译器。

java -jar compiler.jar \

　　　　--js script.js \

　　　　--create\_source\_map ./script-min.js.map \

　　　　--source\_map\_format=V3 \

　　　　--js\_output\_file script-min.js

各个参数的意义如下：

　　- js： 转换前的代码文件

　　- create\_source\_map： 生成的source map文件

　　- source\_map\_format：source map的版本，目前一律采用V3。

　　- js\_output\_file： 转换后的代码文件。

Source map的格式

{

　　　　version : 3,

　　　　file: "out.js",

　　　　sourceRoot : "",

　　　　sources: ["foo.js", "bar.js"],

　　　　names: ["src", "maps", "are", "fun"],

　　　　mappings: "AAgBC,SAAQ,CAAEA"

}

整个文件就是一个JavaScript对象，可以被解释器读取。它主要有以下几个属性：

- version：Source map的版本，目前为3。

　　- file：转换后的文件名。

　　- sourceRoot：转换前的文件所在的目录。如果与转换前的文件在同一目录，该项为空。

　　- sources：转换前的文件。该项是一个数组，表示可能存在多个文件合并。

　　- names：转换前的所有变量名和属性名。

　　- mappings：记录位置信息的字符串，下文详细介绍。

关键就是map文件的mappings属性。这是一个很长的字符串，它分成三层。

　第一层是行对应，以分号（;）表示，每个分号对应转换后源码的一行。所以，第一个分号前的内容，就对应源码的第一行，以此类推。

　　第二层是位置对应，以逗号（,）表示，每个逗号对应转换后源码的一个位置。所以，第一个逗号前的内容，就对应该行源码的第一个位置，以此类推。

　　第三层是位置转换，以VLQ编码表示，代表该位置对应的转换前的源码位置。

位置对应的原理

每个位置使用五位，表示五个字段。

　　- 第一位，表示这个位置在（转换后的代码的）的第几列。

　　- 第二位，表示这个位置属于sources属性中的哪一个文件。

　　- 第三位，表示这个位置属于转换前代码的第几行。

　　- 第四位，表示这个位置属于转换前代码的第几列。

　　- 第五位，表示这个位置属于names属性中的哪一个变量。

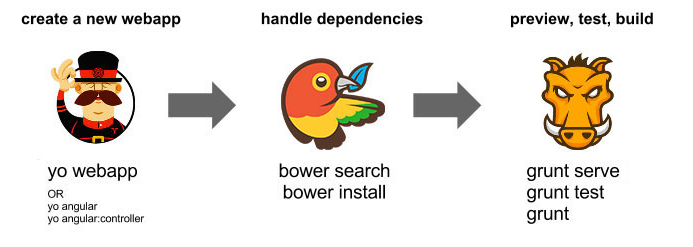
# Yeoman (http://yeoman.io/)

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/>)

# 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

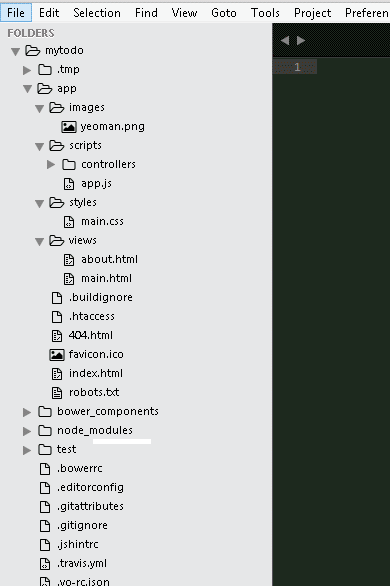
**project-parent-directory> mkdir mytodo && cd mytodo**

**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**

# Sass & Compass

Pre-condition: ruby\bin

Install rubyinstaller-2.1.5.exe

>gem sources --a <https://ruby.taobao.org/> --remove https://rubygems.org/

* install sass and compass

> gem install sass

> gem install compass

(若需要proxy>gem install sass –http-proxy=http://165.225.96.34:10015)

* how to create compass framework with blueprint ?

1. gem install compass-blueprint (安装插件到C:\Ruby193\lib\ruby\gems\1.9.1\gems)
2. 项目父目录> compass create my\_project --using blueprint

or添加给已有项目

b.1)require ‘compass-blueprint (in config.rb)

b.2)已经项目目录> compass install blueprint

c. compile scss to css

my\_project> compass clean

my\_project> compass watch

(note: 编译若出现error: not such file …”.lock”, 原因是compass compile .sasss-cache file full path length over 255 will failed，解决办法是在config.rb文件里添加:

cache = true

cache\_path = 'H:/temp/sass/')

## create compass framework with 960:

1. gem install compass-960-plugin
2. 项目父目录>compass create compass\_blueprint -r ninesixty --using 960

## create compass framework with Bootstrap:

1. gem install bootstrap-sass
2. compass create compass\_bootstrap -r bootstrap-sass --using bootstrap

# 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

)

})