

# 用 grunt 搭建自动化的 web 前端开发环境



jQuery 在使用 grunt，bootstrap 在使用 grunt，百度 UEditor 在使用 grunt，你没有理由不学、不用！

## 1. 前言

各位 web 前端开发人员，如果你现在还不知道 grunt 或者听说过、但是不会熟练使用 grunt，那你就真的真的真的 out 了（三个“真的”重复，表示重点）。至于 grunt 的作用，这里不详细说了，总之你如果做 web 前端开发，你一定要用 grunt。还有一点，它完全免费，没有盗版。既强大又免费的东西，为何不用？当然了，你如果你能找到更好的替代 grunt 的其他工具也是可以的，例如 gulp。Gulp 未来有可能替代 grunt，但是现在来说市场占有率还是不如 grunt。而这种工具咱们是现在就需要用的，所有不要再犹豫了，拿来主义，先用 grunt，即学即用。

本文章旨在讲解 grunt 入门，以及讲解 grunt 最常用的几个插件的使用。篇幅可能会比较长，大家耐心看。本文例证详细、清晰的讲解每一个知识点。但是——如果你看完本文还是不会，我还有最后一个大招。不过你可能需要付出一顿饭钱 + 一包烟钱的代价——去看看我录制的视频教程《[用 grunt 搭建自动化的 web 前端开发环境](#)》（教程中有源码下载），保证你看完就会用。

（PS：碰巧，本文基于 windows 环境写的，而视频教程是基于 mac os 录制的，两者兼备了）

废话不多少，视频教程你也先别看，钱别着急花。先挑战一下自己的理解能力，看下文讲述是否清晰、是否能看懂。

## 2. 安装 nodejs

Grunt 和所有 grunt 插件都是基于 nodejs 来运行的，如果你的电脑上没有 nodejs，就去安装吧。安装 nodejs 非常简单，完全傻瓜式、下一步下一步下一步、的安装方式，这里不再赘述。去 <https://nodejs.org/> 上，点击页面中那个绿色、大大的“install”按钮即可。

安装了 nodejs 之后，可以在你的控制台中输入“node -v”来查看 nodejs 的版本，也顺便试验 nodejs 是否安装成功。

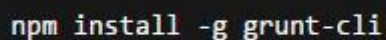
A terminal window with a black background and white text. The prompt is 'C:\Users\福朋>' and the command entered is 'node -v'. The output is 'v0.10.33'.

注意两点。第一，grunt 依赖于 nodejs 的 v0.8.0 及以上版本；第二，奇数版本号的版本被认为是不稳定的开发版，不过从官网上下下载下来的应该都是偶数的稳定版。

## 3. 安装 grunt-CLI

注意，如果你的电脑不联网，以下操作你都做不了，所以，首先保证电脑联网。“CLI”被翻译为“命令行”。要想使用 grunt，首先必须将 grunt-cli 安装到全局环境中，使用 nodejs 的“npm install...”进行安装。如果你不了解 nodejs 的 npm 如何安装软件，这里就先不要问了，先照着我说的做。

打开控制台（注意：**windows 系统下请使用管理员权限打开**），输入：

A terminal window with a black background and white text. The command entered is 'npm install -g grunt-cli'.

**注意，mac os 系统、部分 linux 系统中，在这句话的前面加上“sudo”指令。**

回车，命令行会出现一个转动的小横线，表示正在联网加载。加载的时间看你网速的快慢，不过这个软件比较小，一般加载时间不会很长，稍一会儿，就加载完了。你会看到以下界面。

```
C:\Windows\system32\cmd.exe

C:\Users\福朋>npm install -g grunt-cli
C:\Users\福朋\AppData\Roaming\npm\grunt -> C:\Users\福朋\AppData\Roaming\npm\node_modules\grunt-cli\bin\grunt
grunt-cli@0.1.13 C:\Users\福朋\AppData\Roaming\npm\node_modules\grunt-cli
├── resolve@0.3.1
├── nopt@1.0.10 (abbrev@1.0.7)
├── findup-sync@0.1.3 (lodash@2.4.2, glob@3.2.11)
└─
C:\Users\福朋>
```

这时候要验证一下 `grunt-cli` 是否安装完成并生效，你只需要继续在命令行中输入“`grunt`”命令即可。如果生效，则会出现以下结果：

```
C:\Users\福朋>grunt
grunt-cli: The grunt command line interface. (v0.1.13)

Fatal error: Unable to find local grunt.

If you're seeing this message, either a Gruntfile wasn't found or grunt
hasn't been installed locally to your project. For more information about
installing and configuring grunt, please see the Getting Started guide:

http://gruntjs.com/getting-started

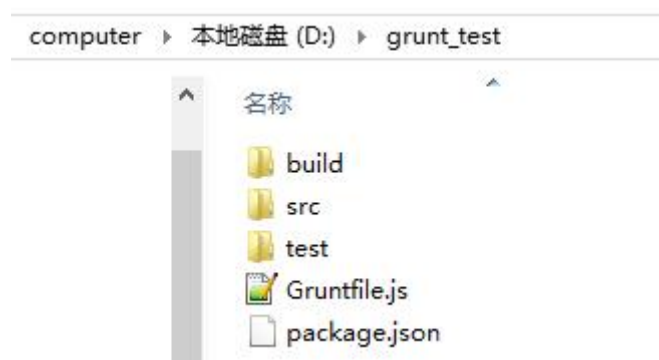
C:\Users\福朋>_
```

你不要管这些结果是什么意思，总之出现这些提示，证明你的 `grunt-cli` 安装成功了。

#### 4. 创建一个简单的网站

Grunt 是应用于实际项目的，所以我们得有一个简单的测试网站来演示 `grunt` 的安装、使用。

首先，我在电脑的 D 盘下面建了一个“`grunt_test`”文件夹，里面建了三个空文件夹、两个空文档，名称如下图。（注意 `Gruntfile.js` 文件的首字母大写！）



其他的東西先不要管，先把 `package.json` 这个文件写一些东西。记住，既然文件后缀名叫“`json`”，那么文件中的格式肯定是严格的 `json` 格式。什么，不熟悉

json? 作为一个 web 前端程序猿，json 是必备课。有一个教程《[json2.js 源码解读](#)》能让你彻底了解 json。

书归正传。Package.json 的内容我们写成如下格式：



```
1 {
2   "name": "grunt_test",
3   "version": "1.0.0",
4   "devDependencies": {
5
6   }
7 }
```

很简单，我们把这个网站或系统的名称、版本写了一下。但是，不光是写在这里占空的，以后会用到的，具体如何用，我们下文会有介绍，先别着急。

还有，最后一个“devDependencies”是什么意思？字面意思解释是“开发依赖项”，即我们现在这个系统，将会依赖于哪些工具来开发。现在代码一行都没有写，依赖于谁？谁也不依赖！所以，就先写一个空对象。但是下文会慢慢把它填充起来。

另外，其实 package.json 中你可以增加任何符合 json 格式的代码，它生来自由，仅仅受 json 代码格式的限制。

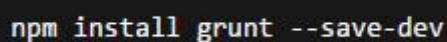
怎么样，看到这里，是不是觉得下文很有悬念，很想看下去呀？那就继续！

## 5. 安装 grunt

主角总是姗姗来迟。《三国演义》在开篇三分一之后才请出来诸葛亮，《水浒传》在第十八回才请出了宋江。而我们本文的主角，也出来的不早。

接下来我们会有一系列插件的安装，他们的安装过程和 grunt 一样。但是他们的执行都是基于 grunt 的，因此才能把 grunt 叫做一个“构建工具”。Grunt 没有具体的作用，但是它能把有具体作用的一个一个插件组合起来，形成一个整体效应。例如，你需要检查 js 语法错误，然后再去压缩 js 代码。如果这两步你都去手动操作，会耗费很多成本。Grunt 就能让你省去这些手动操作的成本。

书归正传。注意，这里安装 grunt 不再是全局安装了，需要你在控制台进入到网站或系统的具体目录下。这里我们进入 D:\grunt\_test 目录下。然后输入以下命令。



```
npm install grunt --save-dev
```

注意，先不要按回车，先不要执行，先看看下文的描述！

这里需要解释的是，“--save-dev”的意思是，在当前目录安装 grunt 的同时，顺便把 grunt 保存为这个目录的开发依赖项。看到“开发依赖项”这一次，是不是觉得有些眼熟？上文在配置 package.json 时，其中的“devDependencies”中就会存储开发依赖项。

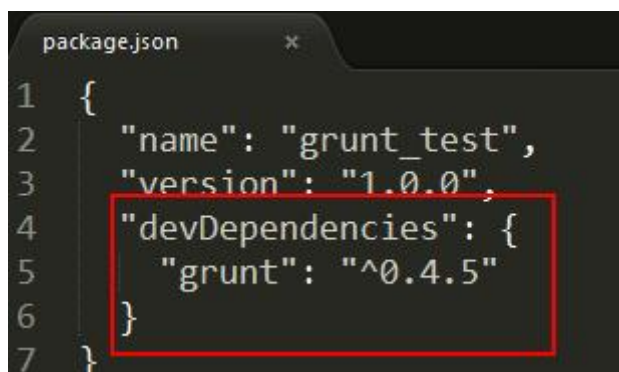
具体保存为开发依赖项是一个什么效果？动手安装一下就是了。首先，在运行安装 grunt 的命令之前，package.json 中的“devDependencies”对应的是空对象。

现在我们来运行这行命令。你会看到几条 warning 提示，不用管它。然后接下来就是加载状态，一个旋转的小横线。稍等待一会儿，会提示你安装成功。如下图所示：

```
D:\grunt_test>npm install grunt --save-dev
npm WARN package.json grunt_test@1.0.0 No description
npm WARN package.json grunt_test@1.0.0 No repository field.
npm WARN package.json grunt_test@1.0.0 No README data
grunt@0.4.5 node_modules\grunt
├── dateformat@1.0.2-1.2.3
├── which@1.0.9
├── eventemitter2@0.4.14
├── getobject@0.1.0
├── rimraf@2.2.8
├── colors@0.6.2
├── async@0.1.22
├── hooker@0.2.3
├── grunt-legacy-util@0.2.0
├── exit@0.1.2
├── lodash@0.9.2
├── coffee-script@1.3.3
├── iconv-lite@0.2.11
├── underscore.string@2.2.1
├── nopt@1.0.10 (abbrev@1.0.7)
├── minimatch@0.2.14 (sigmund@1.0.1, lru-cache@2.6.4)
├── glob@3.1.21 (inherits@1.0.0, graceful-fs@1.2.3)
├── findup-sync@0.1.3 (lodash@2.4.2, glob@3.2.11)
├── grunt-legacy-log@0.1.2 (grunt-legacy-log-utils@0.1.1, underscore.string@2.2.1, lodash@2.4.2)
└── js-yaml@2.0.5 (esprima@1.0.4, argparse@0.1.16)

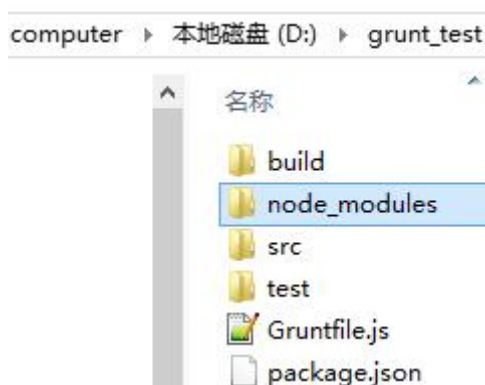
D:\grunt_test>
```

现在你应该第一时间打开 package.json 去看看，那里的“devDependencies”有什么变化。我这里的变化如下图，看看你的是不是和我的一样？



```
package.json
1 {
2   "name": "grunt_test",
3   "version": "1.0.0",
4   "devDependencies": {
5     "grunt": "^0.4.5"
6   }
7 }
```

然后你再看看文档目录中的文件或者文件夹有什么变化？我这里多了一个“node\_modules”文件夹，其中有一个“grunt”文件夹，再其中有若干文档。这里就是存储 grunt 源文件的地方。



这是见证奇迹的时刻，别着急，奇迹还没完呢。然后你在控制台运行“grunt”命令。如果你得到一个 warning 提示，那说明 grunt 已经起作用了。如下图：



```
D:\grunt_test>grunt
Warning: Task "default" not found. Use --force to continue.

Aborted due to warnings.

D:\grunt_test>
```

经过以上三步，说明 grunt 已经在这个目录下成功安装。

那么，为何我们在刚才执行 grunt 时候会有 Warning 提示呢？根据提示，我们得知的信息是：Task “default” not found，如何搞定这个问题？——当然是继续往下看啊。

## 6. 配置 Gruntfile.js

顾名思义，Gruntfile.js 这个文件，肯定是为了 grunt 做某种配置的。按照 grunt 的规定，我们首先把 Gruntfile.js 配置成如下格式。



```

package.json  x  Gruntfile.js  x
1  // 包装函数
2  module.exports = function(grunt) {
3
4      // 任务配置,所有插件的配置信息
5      grunt.initConfig({
6
7          //获取 package.json 的信息
8          pkg: grunt.file.readJSON('package.json')
9
10     });
11
12     // 告诉grunt当我们在终端中输入grunt时需要做些什么（注意先后顺序）
13     grunt.registerTask('default', []);
14
15 };

```

在以上代码中，我们看到了刚才运行 `grunt` 命令，warning 提示中的“default”字眼。不妨我们此时再运行一下 `grunt` 命令，看看会不会再次出现“warning”、“default”等字眼。

```

D:\grunt_test>grunt
Done, without errors.
D:\grunt_test>

```

运行结果告诉我们“Done, without errors”。那就继续往下吧。

另外请注意 Gruntfile.js 中的一句话：

```

//获取 package.json 的信息
pkg: grunt.file.readJSON('package.json')

```

这句话是在 Gruntfile.js 中获取 package.json 中的内容。在上文配置 package.json 时我们说过：package.json 中的内容不光是用来占位置的，还可以在其他地方获取。这里不是已经获取了 package.json 内容了吗？至于获取了如何使用，下文会有介绍，还是继续往下看吧。

## 7. Grunt 插件介绍

进入 `grunt` 官网的插件列表页面 <http://www.gruntjs.net/plugins>，我们能看到 `grunt` 到目前位置的所有插件。现在里面有 4560 个插件，这个数量每天都在增加。而且，这些既然出现在官网，肯定是经过审核的。

插件分为两类。第一类是 `grunt` 团队贡献的插件，这些插件的名字前面都带有“contrib-”前缀，而且在插件列表中有星号标注。第二类是第三方提供的插件，不带有这两个特征。

和 `jquery` 一样，插件那么多，肯定不会全部用。`grunt` 官网插件列表的前几个插件中的前几个插件，下载量最多，它们肯定是大家都在用的插件。第一名 `jshint` 插件最近 30 天下载量将近 89 万次，这是多么惊人的下载量！

咱们可以把前几名插件的作用简单描述一下，看看你在实际编码过程中是否需要？其实不用看就知道答案——肯定需要——要不然怎么会下载量那么高呢？

- `Contrib-jshint`——javascript 语法错误检查；
- `Contrib-watch`——实时监控文件变化、调用相应的任务重新执行；
- `Contrib-clean`——清空文件、文件夹；
- `Contrib-uglify`——压缩 javascript 代码
- `Contrib-copy`——复制文件、文件夹
- `Contrib-concat`——合并多个文件的代码到一个文件中
- `karma`——前端自动化测试工具

以上这些插件，本文不会全部讲到。但是随着讲解其中的一部分，我想你就能掌握使用 `grunt` 插件的方法。知道方法了，然后你就可以参考官方文档去使用你想要的插件。

`grunt` 集全世界 web 前端开发的智慧于一身，比你想想的更加强大，它的插件库能应对你在 web 前端开发遇到的任何事情。

还等什么，继续往下看。

## 8. 使用 `uglify` 插件（压缩 javascript 代码）

`Uglify` 插件的功能就是压缩 javascript 代码。至于 javascript 代码压缩的必要和意义，这里就不用赘述了吧？几乎每一个 javascript 类库或者框架，都有一个 `**min.js` 压缩版。

问一句，你平时做 javascript 开发，都用什么工具来压缩代码？我想这个问题会有许多个答案。但是，使用 `grunt` 的 `uglify` 插件进行压缩，效果绝对不输于任何插件。

要安装一个插件，你首先要进入这个插件在 `grunt` 官网的说明文档页面。我们在 `grunt` 官网插件列表页面，找到“contrib-uglify”点击进入。你要看这里面的说明，然后根据说明进行安装。这里我们只讲重点。



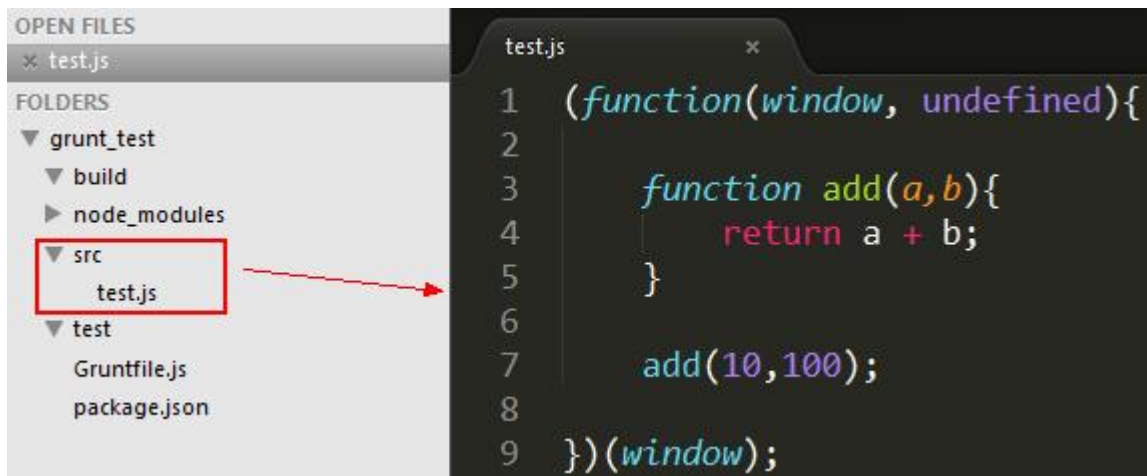
安装 uglify 插件的方式，和安装 grunt 是一样的。还记得 grunt 是怎么安装的吗？

```
npm install grunt-contrib-uglify --save-dev
```

这里又出现了熟悉的“--save-dev”，具体的作用在上文安装 grunt 时已经说过了，不再赘述。运行这句命令。安装完成之后，你会看到 package.json 中“devDependencies”节点的变化，以及“node\_modules”文件夹里的变化。这两点都在安装 grunt 时已经详细说过。

现在还不能用，还需要在 Gruntfile.js 做配置。

不过，先别着急。我们既然要使用 uglify 来压缩 javascript 代码，当然得创建一个 js 文件来做实验。我们在现有的“src”文件夹中新建一个“test.js”，并随便写一些代码。显然，我们无法通过双手和键盘写出压缩后的代码。



测试文件建立好了。我们接下来就要把这个 js 文件进行压缩。

当然，要压缩谁？往哪里压缩？这些都需要配置，在 Gruntfile.js 中配置。分为三步：

第一步，在 grunt.initConfig 方法中配置 uglify 的配置参数。如下图：

```
// 任务配置,所有插件的配置信息
grunt.initConfig({

  //获取 package.json 的信息
  pkg: grunt.file.readJSON('package.json'),

  // uglify插件的配置信息
  uglify: {
    options: {
      stripBanners: true,
      banner: '/*! <%=pkg.name%>-<%=pkg.version%>.js <%= grunt.template.today("yyyy-mm-dd") %> */\n',
    },
    build: {
      src: 'src/test.js',
      dest: 'build/<%=pkg.name%>-<%=pkg.version%>.js.min.js'
    }
  }
});
```

上图中，对 uglify 的配置有两项。

“options”中规定允许生成的压缩文件带 banner，即在生成的压缩文件第一行加一句话说明。注意，其中使用到了 pkg 获取 package.json 的内容。

“build”中配置了源文件和目标文件。即规定了要压缩谁？压缩之后会生成谁？注意，我们这里将目标文件的文件名通过 pkg 的 name 和 version 来命名。

（PS：上文中说过的 package.json 的内容终于找到了他被应用的地方了。这样的好处是：例如，对文件版本的管理，你只需要在 package.json 中修改即可，grunt 会自动根据最新的版本号生成相应版本的文件。你不用手动去修改文件的文件名。）

最后，这里只是对“options”和“build”的基本应用，还有许多中使用方式，可以去官网查阅。

第二步，在 grunt.initConfig 方法之后，要让 grunt 去加载这一个插件。光配置了，不加载上，如何用啊？

```
// 任务配置,所有插件的配置信息
grunt.initConfig({

});

// 告诉grunt我们将使用插件
grunt.loadNpmTasks('grunt-contrib-uglify');
```

第三步，在 grunt 命令执行时，要不要立即执行 uglify 插件？如果要，就写上，否则不写。我现在是需要的，所以我写上。也有可能不需要，这种情况谁知道呢？

```
// 任务配置,所有插件的配置信息
grunt.initConfig({

});

// 告诉grunt我们将使用插件
grunt.loadNpmTasks('grunt-contrib-uglify');

// 告诉grunt当我们在终端中输入grunt时需要做些什么（注意先后顺序）
grunt.registerTask('default', ['uglify']);
```

以上说的这三步已经 OK 了，接下来我们去试试。在控制台中运行 grunt 命令，看得到什么结果。

控制台将输入如下信息：

```
D:\grunt_test>grunt
Running "uglify:build" (uglify) task
>> 1 file created.

Done, without errors.

D:\grunt_test>
```

再去看看，是否生成了一个压缩后的 js 文件？



果然。根据 package.json 中的 name 和 version 生成了文件名。而且，压缩后的代码的 banner 也是符合 Gruntfile.js 中的配置要求的。

以上就是 uglify 插件的详细安装、配置说明。Javascript 使用 uglify 压缩，css 可使用 **cssmin** 插件压缩。安装、配置方法一样的，不再赘述。

## 9. 使用 jshint 插件（检查 javascript 语法错误）

如果你仅仅写一个几十行 js 代码做一个小测试，语法错误的检查大可不必。但我相信看这篇文章的朋友，肯定不限于此，你可能每天都需要写一大堆的 js 代码来完成自己的工作。即使再牛、再仔细的人也会犯一些低级错误，但是 jshint

不会。因此，你最好的做法就是每时每刻都让 jshint 来帮助你检查刚刚写过的 js 代码，有错误立即发现立即处理。这样一来，你们就没必要每隔几天都相聚在会议室进行人工代码走查了。及时代码走查，你也没必要刻意的关注语法错误。

还有一些 js 初级入门的朋友，或者已经有多年 js 经验，却“不思进取”的朋友。你到现在可能都不知道一些 js 语法归法。例如：你到现在可能都不知道“==”和“===”到底有什么区别，你到现在都不知道在语句块内定义变量的坏处，还有更多更多你不知道的。怎么办？让 jshint 来帮助你。

接下来介绍 jshint 的安装和配置。

插件的安装和安装 grunt、uglify 没有任何差别，这里不再赘述了。直接执行下面的命令

```
D:\grunt_test>npm install grunt-contrib-jshint --save-dev
```

配置 jshint 和配置 uglify 一样。在配置 uglify 时候我们讲到了三个步骤，这里也是三个步骤。

第一步，在 grunt.initConfig 方法中配置 jshint。

```
// 任务配置,所有插件的配置信息
grunt.initConfig({

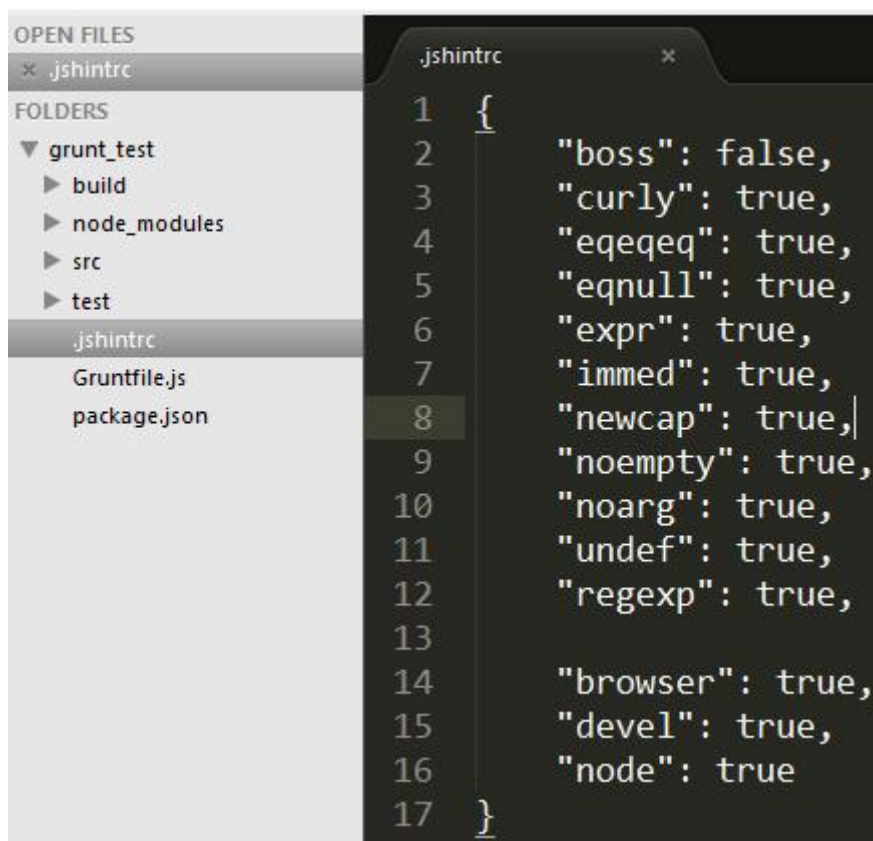
    //获取 package.json 的信息
    pkg: grunt.file.readJSON('package.json'),

    // uglify插件的配置信息
    uglify: {
    },

    //jshint插件的配置信息
    jshint:{
        build: [ 'Gruntfile.js', 'src/*.js' ],
        options: {
            jshintrc: '.jshintrc'
        }
    }

});
```

和 uglify 的配置一样，分为“options”和“build”两个部分。“build”中描述了 jshint 要检查哪些 js 文档的语法。“options”中描述了要通过怎么的规则检查语法，这些规则的描述文件就保存在网站根目录下的一个叫做“.jshintrc”的文件中。因此我们在网站的根目录下面添加上这个文档，并且填写上文件内容。



.jshintrc 文件中代码的格式也要遵守严格的 json 语法，否则无效。那里面这些配置是什么意思呢？想详细了解可以去百度搜索“**jshint 配置**”关键字，你就能知道答案。这里由于篇幅太多，不过多介绍。总之吧，这个 jshint 是我常用的配置。第二步，加载插件。和 uglify 的加载方法一样。注意，这里没有先后顺序。

```
// 告诉grunt我们将使用插件
grunt.loadNpmTasks('grunt-contrib-uglify');
grunt.loadNpmTasks('grunt-contrib-jshint');
```

第三步，配置 grunt 命令启动时，要执行的任务，这里注意先后顺序。你是希望先检查语法呢？还是先合并呢？——聪明人应该都知道先检查语法比较好，因为语法对，合并了有什么意义？

```
// 告诉grunt当我们在终端中输入grunt时需要做些什么（注意先后顺序）
grunt.registerTask('default', ['jshint', 'uglify']);
```



以上三步配置完了之后，我们可以测试一下这个 jshint 到底怎么用。这里我故意将当前创建的 test.js 文件写一个语法错误。

```
1  (function(window, undefined){
2
3      function add(a,b){
4          a = a + c; //显然，这里 c 未定义
5          return a + b //没写分号
6      }
7
8      add(10,100);
9
10 })(window);
```

然后我们执行“grunt”命令，看 jshint 能给我们识别出来这两个错误吗？结果如下：

```
D:\grunt_test>grunt
Running "jshint:build" <jshint> task

src/test.js
  5 :          return a + b //没写分号
      ^ Missing semicolon.
  4 :          a = a + c; //显然，这里 c 未定义
      ^ 'c' is not defined.

>> 2 errors in 2 files
Warning: Task "jshint:build" failed. Use --force to continue.

Aborted due to warnings.

D:\grunt_test>
```

看到没有，jshint 很清晰的识别出了这两个错误。而且注意到没有，jshint 错误之后呢，其后面的 uglify 就没有再继续执行。这正式我们想要的结果。

我们修改完这些错误，在此执行 grunt 命令，结果没有提示错误，而且 jshint 和 uglify 都顺利执行了。

```
D:\grunt_test>grunt
Running "jshint:build" <jshint> task
>> 2 files lint free.

Running "uglify:build" <uglify> task
>> 1 file created.

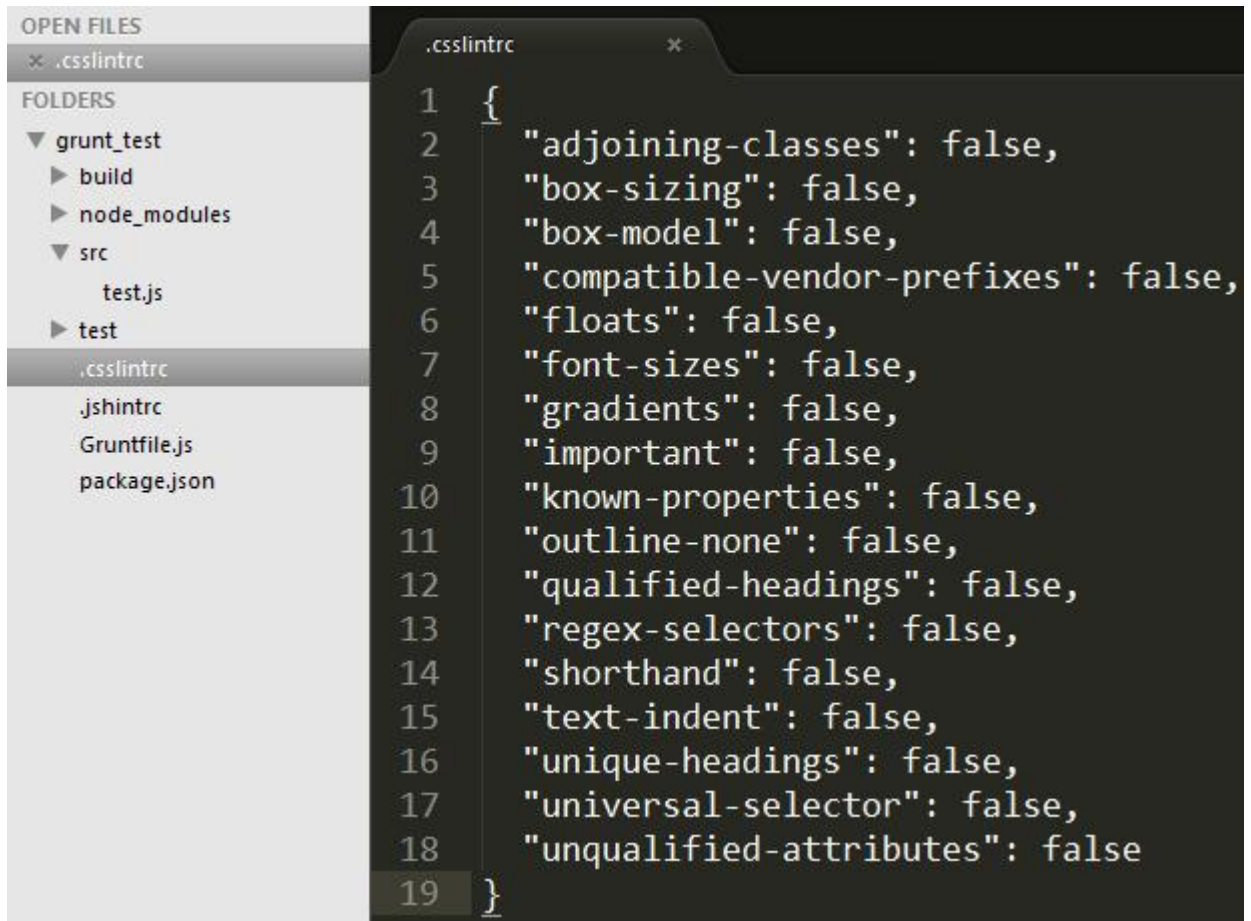
Done, without errors.

D:\grunt_test>
```

## 10. 使用 csslint 插件（检查 css 语法错误）



检查 css 文件的语法错误要使用 csslint 插件，其安装配置方法和 jshint 几乎一模一样。只不过 csslint 依赖于一个叫做“.csslintrc”的文件作为语法检验的规则，我的“.csslintrc”文件如下。其他内容我们就不在此赘述了。



## 11. 使用 watch 插件（真正实现自动化）

你可以一直有一个疑问，上面讲的插件中，每次执行插件功能，都得执行一遍“grunt”命令，这样的操作非常繁琐，说好的“**自动化**”呢？别着急，接下来就解决这个问题——通过 watch 插件解决这个问题。

首先安装 watch 插件，如果安装不再赘述了。接下来要配置 watch 插件，配置分为三个步骤，不再详细描述了，只贴图说明。

第一步。配置 watch 将监控哪些文件的变化，以及这些文件一旦变化，要立即执行哪些插件功能。如下图，watch 将监控 src 文件夹下所有 js 文件和 css 文件的变化，一旦变化，则立即执行 jshint 和 uglify 两个插件功能。

```
// watch插件的配置信息
watch: {
  build: {
    files: ['src/*.js', 'src/*.css'],
    tasks: ['jshint', 'uglify'],
    options: { spawn: false}
  }
}
```

第二步，直接贴图

```
grunt.loadNpmTasks('grunt-contrib-watch');
```

第三步，直接贴图

```
// 告诉grunt当我们在终端中输入grunt时需要做什么（注意先后顺序）
grunt.registerTask('default', ['jshint', 'uglify', 'watch']);
```

这三步执行完了，即 watch 插件配置完成。运行 grunt 命令，控制台提示 watch 已经开始监听。此时要想停止，按 ctrl + c 即可。

```
D:\grunt_test>grunt
Running "jshint:build" (jshint) task
>> 2 files lint free.

Running "uglify:build" (uglify) task
>> 1 file created.

Running "watch" task
Waiting...
```

既然在监听，我们试一试看监听有没有效。我们将 test.js 代码中去掉一个分号，看它能否自动检查出来这个错误。

```
>> File "src\test.js" changed.

Running "jshint:build" (jshint) task

  src/test.js
    4 :          return a + b
      ^ Missing semicolon.

>> 1 error in 2 files
Warning: Task "jshint:build" failed.

Running "watch" task
Waiting...
```

结果显示，watch 检查到了 test.js 文件的变化，而且通过执行 jshint 提示了语法错误。更重要的是，它现在还在监听、并未停止。说明它正在等着你去修改错误，重新监听检查。那我们就不要辜负它了，再把语法错误修复了。看它会如何处理。

```
>> File "src\test.js" changed.

Running "jshint:build" (jshint) task
>> 2 files lint free.

Running "uglify:build" (uglify) task
>> 1 file created.

Running "watch" task
Completed in 0.097s at Mon Jun 08 2015 20:25:34 GMT+0800 (中国标准时间) - Waiting...
```

它又检测到了文件变化，这次语法没有错误，它很顺利地执行了 jshint 和 uglify，执行完毕之后重新进行监听。多听话的工具！

好了，已经回答了你们的问题——自动化。

## 12. 上文中所谓的“build”

上文中描述各个插件的配置时，都是用了“build”这一名称作为一个配置项。

```
// uglify插件的配置信息
uglify: {
  options: {
    stripBanners: true,
    banner: '/*! <%=pkg.name%>-<%=pkg.version%>'
  },
  build: {
    src: 'src/test.js',
    dest: 'build/<%=pkg.name%>-<%=pkg.version%>'
  }
},

// jshint插件的配置信息
jshint:{
  build: [ 'Gruntfile.js', 'src/*.js' ],
  options: {
    jshinttrc: '.jshinttrc'
  }
},

// watch插件的配置信息
watch: {
  build: {
    files: ['src/*.js', 'src/*.css'],
    tasks: ['jshint', 'uglify'],
    options: { spawn: false }
  }
}
```

那么这里是不是必须用“build”这一个名字？答案很明显，当然不是。我之前之所以没直接说，是因为我要先把插件的安装和配置讲明白，不变一次传输太多知识，现在一并和大家说了。

这里可以用任何字符串代替“build”（但要符合 js 语法规则）。甚至，你可以把“build”指向的内容分开来写。这样对多人协同开发很友好。好的设计就是这样：让用户尽情发挥他们的自由来干事，而不是去限制他们。

```
jshint:{
  build: [ 'Gruntfile.js', 'src/*.js' ],
  options: {
    jshinttrc: '.jshinttrc'
  }
},
```



```
jshint:{
  test1: [ 'Gruntfile.js' ],
  test2: [ 'src/*.js' ],
  options: {
    jshinttrc: '.jshinttrc'
  }
},
```

如上图，我对 jshint 的配置做了修改，大家可以去自己修改，然后执行 `grunt` 命令试试。命令行会有“test1”、“test2”的字眼。

### 13. 批量安装插件

请各位看官继续跟随我思考问题，学而不思则罔。

到现在为止，我刚刚安装了 3 个插件，“node\_modules”文件夹所占据的空间就有 18MB 了。大家猜一猜，我在上传代码到开发库的时候，会不会把“node\_modules”中的内容也上传呢？既然我这么问了，答案肯定是不上传。

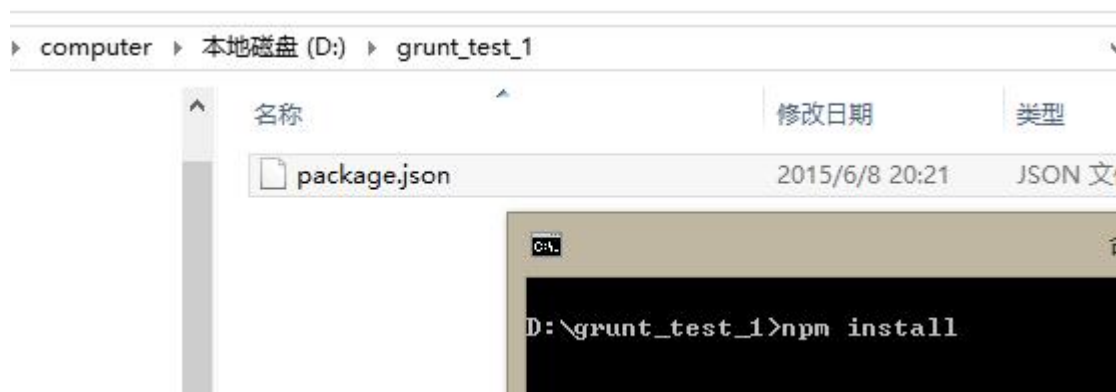
问题又来了，我如果作为开发环境的搭建者，我不把“node\_modules”上传，其他一起开发的人，怎么得到这些 `grunt` 插件和工具呢？有人回答让他们自己去手动一个一个安装——首先这是一个笨方法，其次如果我当年安装的旧版本，而他们现在自己安装的可能是新版本。新旧有可能不兼容啊。

该怎么办？

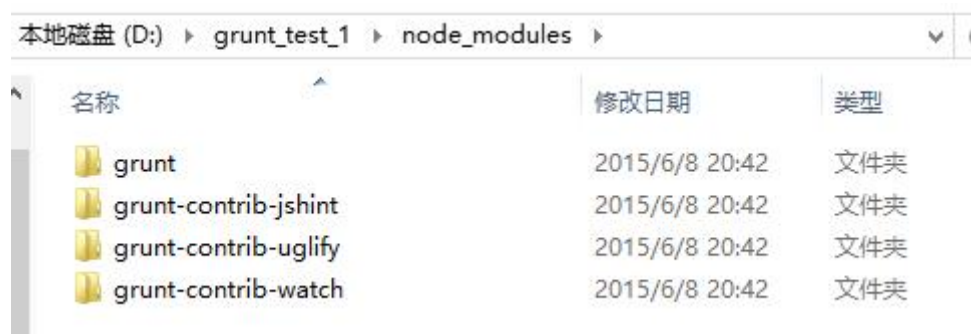
其实答案很简单——我上传源码时候，肯定会把 `package.json` 上传上去，而 `package.json` 中的“devDependencies”就记录了这个系统的开发依赖项，然后我通过 nodejs 的 `npm` 即可批量安装。

做一个试验。我在 D 盘下面新建一个目录“`grunt_test_1`”，然后把“`grunt_test`”中的 `package.json` 拷过去。在打开命令行跳转到“`grunt_test_1`”，执行“`npm install`”命令，看看得到什么结果。





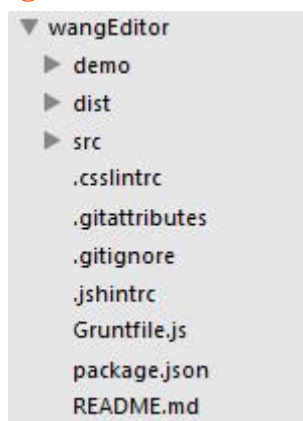
此时按回车执行命令，结果在“grunt\_test\_1”生成了“node\_modules”文件夹，里面安装好了 package.json 中“devDependencies”配置的插件。而且，版本都是一直的。神奇吧！



## 14. 系统文件结构

使用 grunt 来搭建 web 前端开发环境时候，文档目录和之前可能就不一样了。因为你手动写的代码文件，绝对不是你最终输出的文件。这些还需要经过 grunt 各种插件的检验、合并、压缩才能最终输出给用户。

这里我拿自己的开源项目 **wangEditor** 的文档结构举例子。



上图中，“src”文件夹里面存储的是原始的代码文件，“dist”文件夹里面存储的是最终生成的代码文件，“demo”里面存储的是一些测试页面。



当然了，各个系统的文件组织形式不一样，但是我建议大家还是按照这么一个思想去组织文档结构。大家也可以去 github 上参考一下 jquery、bootstrap 这些著名开源项目的文档结构。看看 jquery 输出的虽然是简单的一个 js 文件，但是它的开发环境是多么的复杂。

做好前端，不容易。

## 15. 完结

这篇文章我使用 word 写的，然后粘贴到博客园中发布的。到现在为止，我写了一共有 20 页、6700 多字，全部手写、手动截图。写博客写到这个份上，我也算是拼了！

如果你看完这篇文章还不会 grunt，或者你懒得看文章、嫌累，我建议你去看看我录制的《[使用 grunt 搭建全自动 web 前端开发环境](#)》这个视频教程，视频里讲的肯定详细的多了，而且可以下载源码。