Sunday, November 2, 2014

注意点:

1. PPT修改成16:9 比比例布局, 一一般用用黑黑字, 重点用用红色色
2. 代码编辑器使用用白白底,字体24以上,网网网页页放大大显示示,调整工工具的字体也放大大
3. 讲课时,尽量不要用用老老师、同学什么的,用用第二二人人称、交流。
4. 举例什么的,尽量避开一一些政治、法律、热议问题等。
5. 所有写代码和命令行行的操作界面面要求全屏显示示,建议采用用白白底黑黑字,微软雅黑黑字体,字号不得
小小于24号,屏幕分辨率为1080*720或1366*768时,字号在24号或以上,分辨率为1920*1080 时字号设置为28号或以上

Grunt-Beginner

大家好, 我是mater, 很高兴能跟大家一起分享一些Grunt相关的知 识和经验, 讲到Grunt呢, 就一定绕不开一个命题, 那就是什么是前端集 成解决方案呢。

(翻页页)下面面我们就来聊一一下什么是前端集成解决方方案。(翻页页)

不知道大大家是否喜欢看电影, 我个人人呢是非非常的喜欢, 在十十一一国庆档, 我们诞生生 了一一位50亿影帝-黄黄渤, 他参演的三部电影《亲爱的》《心心花路放》《痞子子英雄二二》的总 票房达到了50亿,超过了中国最红的喜剧片片导演冯小小刚和最红的好莱坞导演,指导变形金金 刚系列的迈克尔·贝贝,下来呢就有人人分析,说黄黄渤为什么能成功? 这时候就一一定少不了专 家分析,专家说: 是因为黄黄渤生生于草根,成于学院。说到这里里大大家就一一定明白白什么意思 了。那我们今天也来用用即草根又又学院的方方式来描述一一下, 草根派的说法呢就是: 前端集 成解决方方案就是用用来解决前端工工程的根本问题。 学院派说法呢: 一一套包含框架和工工具, 便于开发者快速构建美丽实用用的Web应用用程序的稳健强壮的工工作流。 大大家注意这里里边有 几几个关键词: 框架和工工具, 快速, 美丽实用用。 美丽实用用是我们要达到的目目目的,框架和工工 具,是我们实现目目目的的基础,后边重点介绍的GruntJS就属于其中,而而快速则是框架和工工 具在过程中所能给予我们的。

(翻页页)那么学完本次课程, 到底可以解决我们的哪些前端问题呢? 开发团队代码风风格不统一一,如何强制开发规范; 先期开发的组件库如何维护和使用用; 如何模块化前端项目目目; 服务 器部署前必须的压缩,检查流程如何简化, 如何完善,诸如此类的问题,都是我们将要一一一一讨论的。在这里里, 有一一个小小点, 特别想跟大大家探讨一一下, 就是刚才提到的美丽和实 用用,如果不能同时具有的话, 到底哪个对于一一款产品的成功来讲更重要呢? 我个人人呢, 倾向于实用用, 因为我认为实用用是一一种客观的使用用感受, 它是切实存在的。而而美丽呢, 则 是大大众主观的使用用感受,每个人人都不尽相同。 所以你说苹果推的这个扁平化,有人人喜 欢, 有人人不喜欢, 不喜欢扁平化的人人是否就绝不购买使用用苹果的产品了呢,喜欢拟物的 人人是不是就一一定得花上3000元为所谓的情怀买单呢? 显然不是, 因为不管扁平还是拟 物, 简单实用用才是影响用用户是否买账的关键因素。

Sunday, October 26, 2014

(翻页页)讲完了概念,我们要讲讲实现: 目目目前有哪几几种主流的前端集成解决方方案 呢:

第一一种就是我们课程在后边将要重点介绍的 Yeoman, Bower, Grunt的方方式, 同时 会对比比介绍Grunt的一一个强有力力的竞争对手手Gulp。这几几个我们会有非非常深入入和细致的讨 论。

在mac平台上,有一一个把这套流程整合可视化了的app, 叫做codekit。稍等我给大大 家看一一眼。

然后就是国内有些团队在用用的百度出的FIS,和我的老老东家,腾讯AllowTeam出的 Spirit(sbirit 发音音), 这两个大大家感兴趣可以去它们各自自的网网网站查看。

接下来,我们看一一眼codekit (停止止录屏)

(打开codekit全屏,开始录屏)

codekit在我们的本地起了一一个server, 点击preview可以在浏览器中浏览本地代 码, 打开配置项,我们可以看到codekit支支持的语言言类型,使用用的代码检查库有JShint等, 代码压缩是用用的uglify,以及支支持的库和框架,各项配置呢,我们就不细说了, 因为 codekit有一一个致命的缺点, 那就是只支支持mac平台。 当然,全员配置rmbp的土土豪团队, 不在我们讨论范围内。(停止止录屏)

(翻页页) 刚才我们提到, 主流的方方式是Yeoman, Bower和Grunt, 那Grunt在其中是一一个 什么样的定位呢? 我们把它称之为Build tool。

(翻页页) 同为Build Tool的还有很多, 比比如说: 曾经叱姹风风云很多年的Ant, 最近新兴的Gulp, 还有一一些偏小小众的如Buildy, jasy, 还有在Linux平台大大行行其道的,Gmake,顺带 提一一句,如雷贯耳耳的jQuery早期就是使用用Gmake作为它的构建工工具的,而而今也早已叛变到 Grunt的阵营。Ant和Gulp在后边我们会对比比介绍, Build,Jasy和Gmake就不再做赘述,大大 家有兴趣可以去他们各自自的官网网网上查看。

(翻页页)听完了这些介绍, 大大家是不是已经跃跃欲试, 按耐不住想要掀起Grunt的盖头 来呢。 只要你具备HTML, CSS, JS的基础知识, 并且对NODEJS 有所了解,只是了解哦。同时愿意花时间和精力力来了解前端集成解决方方案,认为其将能够为我们构建现代化 web项目目目锦上添花的话。 那就来吧。

好, 这堂课我们就讨论到这里里, 下一一节, 我将跟大大家一一起搭建我们必须的本地环境。 谢谢大大家。

上一节， 我们初步介绍了Grunt所能为我们带来的好处，Grunt本身呢， 是依赖于NodeJs构建的， 如果你以前已经接触过NodeJS了， 可以直接跳过这节， 或者跟我们一起温故而知新， It’s up to you ! 在这一节， 我们将了解Node环境的安装以及命令行的使用和Node项目的基本结构。

（翻页） 首先我们先来问一个是什么的问题？ NodeJS到底是个什么东西呢？ 为了说明这个问题，我们就必须追溯一下JavaScript的历史， JavaScript最早出现是作为一门客户端脚本语言， 发展到今天已经一统浏览器江湖。人也是一样，当你在一个领域孤独求败的时候， 你就会去尝试涉足另一个领域。 那哪个领域是JavaScript没有涉足但是又跟它关系很紧密的呢？对， 就是服务端语言， 感谢Node让我们可以用JavaScript编写服务端程序， 这就好比给JavaScript插上了翅膀， 让我们从浏览器飞到了服务器。 所以情形就很

清晰了， 只要你学会了JavaScript这本语言， 从浏览器到服务器就全包了， 再也不用JavaScript切到PHP, 或者C#， 或者Python， 或者Ruby等等。 但话也不能这么说， 除了语言上的挑战，其实还有很多东西需要学习。 下边我们先来安装一下Node的环境。

（翻页）在我们的机器上安装Node实在是不能更简单了， Node的开发团队在这方面确实下了很大的功夫， 让我们可以在各个平台上非常容易和直观的安装好Node。 我们到Node的官网上看一下， 点击大大的绿色安装按钮， 会将安装包进行下载， 直接点击，一键式傻瓜安装。 除了这种安装方式呢， 在mac和linux上我们还可以使用相关的包管理工具进行安装， mac上我们可以使用homebrew, 如果有使用homebrew的同学， 这里我给大家推荐一款小工具， 叫做cakebrew， 可以免去我们记命令行的烦恼。 。。。linux下， ubuntu可以使用 apt-get， fedora可以使用yum， 等我就不一一细说了。

（翻页）我个人呢， 是一个命令行或者说控制台的狂热分子， 认为这玩意比图形界面，鼠标点来点去要高效和强大的多。 而node中经常给我们带来某些不必要的麻烦的就是空格的存在，此话怎讲呢。 因为有些同学的机器的用户名里边含有空格， 而全局安装的node程序包在Windows下是放在我们的Home目录的， 很多node程序包因为用户名中含有空格往往不能正常工作。 所以我的建议是大家在取用户名的时候尽量不要包含中文，如果你已经这么做了， 可以新建一个账户来规避这个问题。

接下来就是选择一个适合自己的命令行工具， mac和linux用户就不用说了， 本身就已经自带了非常强大的控制台， 推荐mac用户可以试试iTerm, 我认为可以完美替代mac原生的控制台。 问题就出在windows平台上， windows虽然提供了一个类似于控制台的体验，cmd窗口， 但并不支持bash脚本， 为了保持操作上的一致性， 我建议使用windows的同学安装一下git， git提供了一个git bash shell， 在一定程度上弥补了这个缺陷。在课程中初次遇到某个bash命令的时候，我都会介绍一下， 所以之前没有接触过bash的同学也不需要担心。

npm, 一言以蔽之， 就是Node的包管理和分发工具。 如果你熟悉ruby的gem， Python的pypi， setuptools, PHP的pear, 那么不用多说你就已经知道npm的作用了。 Nodejs自身提供了基本的模块，但是开发实际应用过程中仅仅依靠这些基本模块则还需要较多的工作。幸运的是，Nodejs库和框架为我们提供了帮助，让我们减少工作量。但是成百上千的库或者框架管理起来又很麻烦，有了NPM，可以很快的找到特定服务要使用的包，进行下载、安装以及管理已经安装的包。 这个名字起得也很有创意， 我们知道PHP最早起名字是取自： Personal Home Page 的首字母缩写， GNU则是： GNU's Not Unix的递归缩写。NPM则是 Node Package Manager 的首字母缩写。

那么npm是否需要我们单独安装呢， 不必的， 在我们安装Node的时候npm就已经一起安装了。 我们将使用最多的npm命令将是 install, 举例来说， 安装Grunt这个node程序， 我们只需在本地运行 npm install -g grunt-cli 这里-g的意思呢， 则是全局安装， 意味着整个系统都可以使用grunt。 那如果运行install命令的时候不带g呢， 则会在当前目录生成一个node_modules的目录， 将相关的node程序安装在这个node_modules目录中， 我们看一下。如果没有在install后边跟任何的package name呢， npm则会去当前目录寻找一个package.json文件， 找到了， 就按照package.json文件中的声明去进行安装， 没找到，

则会报一个错。 我们看一眼。 而package.json的语法也很简单， 内容呢本身是一个js的object literal， 就是一个键值对的配置项。 我们提几个简单的。

第三章 ——————————— Yeoman Bower Grunt 的安装 ———————————————

大家好，我是mater。在上一节呢，我们一起学习了Node的安装和使用。 这一节我们就来一起看看，怎么在Node环境下安装Yeoman， Bower 和 Grunt。

首先， 我们来看一眼yeoman的官方站点：Yeoman的logo是一个八字胡卡通老头，Yeoman本身的释义呢是自由民，公民，大家记住我们看图学单词的第一个单词了吧 。好， 言归正传，Yeoman很霸气的给了一句介绍： 现代Web App的脚手架工具。 脚手架作何解呢？清代孔尚任的《桃花扇》中有一句： 眼见他起高楼，眼见他宴宾客，眼见他楼塌了。本来是用来形容世事无常。这里我们套用一下，所谓万丈高楼平地起， 靠的是什么， 就是脚手架， 有了这些施工现场为工人操作并解决垂直和水平运输而搭设的各种支架才大大方便了起高楼的速度和质量。Yeoman的作用就不言而喻了， 在Web项目的立项阶段， 使用yeoman来生成项目的文件，代码结构， yeoman自动将最佳实践和工具整合进来， 大大加速和方便了我们后续的开发。这时候就一定会有同学问了， 说有那么多的最佳实践和工具， 所谓没有最好只有更好； 没有好的， 只有合适的。 Yeoman怎么保证自己用到的就是最适合我当前项目的呢？Yeoman的开发者很早就想到了这点， 所以他们维护了一套生成器的生态。 举个形象点的例子， 如果我们把Yeoman看做是一堆橡皮泥， 那么生成器就是一个个的模(mu)具， 通过模具， 我们最终创造出，方的，圆的，五星的成品。 比如说我要在将来的项目中主要使用Angularjs框架来进行开发， 我们就可以找angular web app的生成器； 我的项目主要是在mobile上跑的， 我们就可以找 mobile web app的生成器。 但不管怎么变， Yeoman还是为我们统一了一些现阶段公认的： 代码校验， 测试， 压缩等最基本功能的流程， 其余部分则交给生成器自由发挥。接下来我们就来安装一下Yeoman，安装方法非常简单， 就是标准的一个npm包的安装方式， 通过使用-g我们将yeoman安装在全局， 这里要注意的是yeoman在npmjs.org注册的名字叫做yo，而不是yeoman的全称。当我们使用npm进行在线安装的时候，如果提供的是一个名字， 则npm首先去npmjs.org 找到这个名字对应的程序，然后才下载安装。在后边我跟大家分享如何发布我们自己的npm的grunt的插件的时候， 会再详细介绍npmjs.org的使用。安装完以后， 执行yo -v， 控制台成功打出yeoman的版本号则说明安装成功。大家看到截止我们视频录制的时候，yo最新的版本是1.3.3 。

接下来我们看一眼bower的官方站点： Bower的logo是一只鸟， 我之前只知道bower的释义是凉亭， 荫凉。学了Bower我才知道， 有一种澳洲产的鸟叫做BowerBird，而BowerBird名字的来源也很有意思， 就是因为雄鸟在求偶时用树枝搭建“凉亭”，并用色彩鲜艳的小物品装饰其间。大家记住我们看图学单词的第二个单词了吧。 好， 言归正传。Bower官网给出的介绍是： Web的包管理器。 一个Web站点由很多部分组成， 框架，库， 公共部分等， 而Bower则用来跟踪管理这些。打个比方， 以前我们有两个项目A和

B， A使用了jquery1.11.1，B使用了jquery2.1.1 。 我们以前的做法是跑到jquery的官网上， 把两个版本分别下载下来， 然后丢到A和B项目中。有了Bower再也不用这么麻烦了， 我们只需要告诉Bower， 我需要jquery1.11.1, Bower就会自动帮我们把jquery1.11.1下载到我们本地的项目中，省去了我们很大的麻烦。贺岁片之王，葛大爷在电影《手机》里边有一句经典台词：麻烦，诶我说您这可是顶着麻烦就上了啊。希望有了Bower以后，大家都不再顶着麻烦就上了。接下来我们安装一下Bower， 安装方法非常简单， 依然是标准的一个npm包的安装方式，而且注册名跟商标名一致， 都是bower。执行bower -v，控制台成功打出bower的版本号则说明安装成功。大家看到截止我们视频录制的时候，bower最新的版本是1.3.12 。

接下来我们再看一眼Grunt的官方站点： Grunt的logo是一头面目可憎的野猪，其原意是形容咕噜声， 大家记住我们看图学单词的第三个单词了吧。 好， 言归正传。Grunt官网给出的解释是： JavaScript Task Runner。经典台词： “现在年轻人都这样，旅行不叫旅行，叫行走。” 我们也给Grunt改个描述： Build tool。 为什么我们需要Build tool呢, 一言以蔽之就是： 自动化。 像压缩， 编译， 单元测试， 代码校验这种重复且无业务关联的工作我们做的越少， 就可以留出更多的时间和精力专注于我们的业务代码。那为什么我们选择Grunt呢， 因为Grunt的生态实在是太强大了， 只要你能想到的任务， 基本上都可以找到Grunt相关的自动化插件， 并且Grunt的生态一直在持续增长。包括到课程的最后， 我也会跟大家一起尝试为Grunt贡献几个插件。我们不能给Grunt全世界， 但是Grunt的世界， 全部给了我们。好， 接下来我们看一下Grunt如何安装。安装方法非常简单，依然是标准的一个npm包的安装方式， 通过使用-g我们将yeoman安装在全局， 这里要注意的是跟在后边的包名是：grunt-cli, cli是command line interface的缩写， 安装完之后在我们的system path 中会有grunt的命令， grunt-cli干的事情也很简单， 我们在命令行执行一下grunt， 看到下面这个信息说明安装成功，grunt-cli会执行本地package.json文件中指定安装的grunt版本，读取我们当前所在目录下的Gruntfile配置文件，然后按照其中的配置项来对我们的项目执行自动化。这些在我们后边Grunt一节都会有详细的讲述。

好， 这堂课我们就讨论到这里, 下一节, 我将跟大家一起使用我们安装好的Yeoman来搭建我们的web项目。

第四章 ——————————— Yeoman实战 ———————————————

大家好，我是mater。在上一节呢，我们一起初步了解和安装了Yeoman，Bower和Grunt。 这一节我们就来一起深入学习一下Yeoman，看在实际生产中如何对Yeoman加以使用。

我们本地已经安装好了Yeoman，Yeoman自身呢只是一团橡皮泥，想要化腐朽为神奇，还需要各式各样的模具，那就是Generator，（切换到yeoman的网站）在Yeoman’的官网第二个tab就是Generator的归纳页， 截止到这一刻， Yeoman已经有了1188个

Generator，生成器，浩如瀚海啊。我们大概浏览一下。默认是按照Generator收获的赞的个数拍序的。排在第一位的：（这里使用屏幕颜色笔） angular， 可能有同学还没用过angular，但也一定听说过它的大名了，Angular.js 是一个MV*（Model-View-Whatever，不管是MVC或者MVVM，统归MDV(model Drive View)）JavaScript框架，是Google推出的SPA（single-page-application）应用框架。它为什么这么火呢， 我们稍等管中窥豹，顺带了解angular的一两个特性，以及它所解决的前端痛点问题。大家注意Angular名字前面的这个八字胡标识，有这个标识则说明这个生成器是官方出品，没有的则是第三方热心团队或个人贡献的。往下看， 我们看到有（这里使用屏幕颜色笔）mobile， 移动优先的webApp的生成器，继续往下是（这里使用屏幕颜色笔）webapp，最常规，常用的现代web app的生成器。

（移到命令行窗口）来， 进入实战。Generator并不是随yeoman安装的，需要我们各取所需， 自行安装。我们首先来安装一下这三个Generator，都是标准的npm程序包。Generator的npm包的命名管理是在自身名字的前边加一个generator-, 比如angular就是generator-angular , mobile 就是 generator-mobile, webapp 就是 generator-webapp 。

。。。。。安装。。。。。。

好， 安装好了， 我们来看一下怎么用，

mkdir yo-in-action

cd yo-in-action

用法也很简单， 在yo 命令后边跟generator的名字， 然后是我们项目的名字，这里注意项目是生成在当前所在目录的， 所以执行之前我们先生成一个项目目录。

mkdir angular-in-action

cd angular-in-action

好生成一个angular项目

yo angular learnangular 为了说明这里指定的项目名，和我们自己生成的项目目录的不同， 我特地换了个名字， 我们叫做learnangular 。

第一个提示： 我们是否会在项目中使用Sass，我们选择是。简单提一句Sass：我们可以把Sass看作是CSS的一种扩展语言（类似的还是有Less, Stylus) ，完美向下兼容css的语法，并在css的基础上提供了一些编程的能力。在开发过程中使用Sass文件，最终还是生成CSS文件。即使不知道也没关系，并不影响我们理解后边的知识，看代码的过程中，我也会捎带着介绍一两个sass的知识点。但如果让我给建议， 我是一定建议大家尝试在项目中使用Sass的，需要注意的Sass是基于Ruby构建的，要求我们的机器上已经安装ruby环境和Sass以及Compass，所以希望本地没有安装的同学可以自行谷歌把这几样东西装上并用起来，Compass 是在Sass的基础上二次开发的一套框架，提供了更加丰富的功能。这里套用后会无期中的一句（后会无期台词）要先学会分辨，然后再去信任，所以到底用不用由大家自行决定。继续往下， 是否包含bootstrap， bootstrap是一个简洁、直

观、强悍的前端开发框架，主要发力点还是整站风格和一些通用组件的使用。为了方便我们后边的页面演示，我们把它也包含上。 y ——- bootstrap最早的版本是使用less进行css的编写和管理， 随着Sass越来越火， Bootstrap官方也维护了一个Sass的版本，既然我们已经决定使用Sass了， 选用这个版本。后边呢，则是可选用的angular的基本组件之外的扩展插件，我们全都选上。 好， 项目生成完了，我们首先看一下项目目录结构。 pwd， 我们目前仍然是在刚刚创建的项目目录， 看看下边都生成了哪些文件 ls -al (ls 是列出当前目录下文件的bash命令）-al 表示把隐藏文件，即命名以点开头的也一起用详细信息的方式列出。我们看到项目代码是直接生成在当前目录的。我们把这个项目拖到编辑器进一步阅读。

yeoman生成的这个项目本身是一个基于node构建的项目，读一个node项目我们首先阅读的就是它的package.json，这里边包含了这个项目最直观的描述。我们打开

name: 我们之前指定的learnangular，这就是我们在运行 yo angular 时指定的项目名，注意它并不是我们项目所在目录的名字。version，当前版本，随着我们项目的不断开发迭代，这个值是一直递增的。dependencies和devDependencies 这两个值都是我们项目需要依赖的node包，但是devDependencies 指的是我们在开发过程中所要依赖到的包，而 dependencies 则是我们的项目在生产环境中需要的依赖， 何谓生产环境， 这么说， 我们将来开发一个node程序，发布出去， 别人通过 npm install 直接安装的时候， 会去查看我们node程序的package.json的dependencies字段，将我们项目中依赖的node程序也一并安装了， 但是声明在devDependencies中的node包则不会被安装。 这里的 dependencies为什么是空呢？ 因为虽然我们当前的开发项目是一个node项目， 但是我们最终发布的时候是一个站点， 并不是一个node项目， 声明也是没有意义的。

value是一个obj 键值对的形式， key是包名， value是版本号, 尖括号是啥意思呢， 是一个比较宽松的对版本的限制，它只限制主版本号， 就是说，在当前主版本号内的版本都ok，当然npm会自动帮你更新最新的。举例来说，这里要求grunt 0.4.5 说明， grunt 0.4.5 肯定是已经发布了的， 那如果以后有0.4.6 发布了， 我的程序就用0.4.6， 如果有0.5.0发布了，就用了0.5.0, 一直到grunt有了一个很大的变化， 发布了 1. X.X 主版本号变了， 那就停留在1.X.X之前的最后一个0.X.X版本。如果把 ^ 换成 ~ 则是一个比较严格的版本限制， 要求只能是最小的版本号的更新。 依然拿Grunt 0.4.5举例子， 0.4.6发了， 我就用 0.4.6 但是0.5.0 发了，更新吗？ 不更新。套用后会无期里边的一句台词： 喜欢就会放肆，但爱就是克制。 这是一种克制的方式，请大家记住。

engines 指定当前项目所需的node版本 >= 0.10.0

scripts 可以直接使用 npm 运行脚本命令。 这里test的意思就是 我们在当前项目运行 npm test 其实是去执行的 grunt test， node支持的scripts很多， 最常用的，其实是 install，当按照package.json的指定， 把依赖的程序都安装好了， 执行什么命令。

大家还记得我们在第二节讲如果执行 npm install 的时候不带任何参数会发生什么吗？我们先把本地yeoman生成的的node_modules目录删掉， rm是bash删除目录的任务，-rf则是指定递归删除，且不需要二次确认。好，删掉了。 我们在当前目录执行一下npm install。发现， npm会根据我们package.json 中指定的devDependencies 去下载必须的node包。所以毫无疑问， node_modules 就是用来存放我们开发过程中所依赖到的node包。好安装好了， 我们来看一下其他的目录及文件

第6节 Grunt In Action - Grunt的Task，Target和Options

大家好，我是mater，上一节呢， 我们一起学习了解了bower在实际生产中的应用， 同时呢， 也认识到了单独使用bower的局限性。从这一节开始，我们就将真正的深入到Grunt的各项功能中，一步步了解其在实际生产中是如何使用的。 首先呢， 我们来探寻一下yeoman是如何把Grunt应用于其生成的项目的，并以此来了解Grunt最基本的三个概念 Task, Target 和 options。

还是老规矩， 首先呢，我们创建一个测试目录， grunt-in-action。 。。 进入

创建一个项目目录 grunt-by-yo 。。。 进入

我们之前呢已经学习了如何用yeoman的Generator来快速的搭建我们的web项目，这一次呢，我们使用webapp这个Generator来搭建一个测试项目。

。。。。

是否使用Sass

。。。 bootstrap， sass和bootstrap在之前的章节我们都已经简单介绍过是做什么用的

modernizr： 是用来检测用户浏览器是否支持某些HTML5和CSS3新特性的一个JS库。

是否使用node-sass， 大家都知道，啊，我们之前聊过了， sass是基于ruby实现的， 后边为了追求更好的性能和可移植性，CSS预编译器这块就换用了C和C++来实现，这部分就叫做libsass, Node-sass干的事情呢就是在libsass的基础上加以封装，用nodejs调用，我个人呢不太建议使用node-sass, 因为我在用的过程中发现了一些莫名奇妙的问题，感觉支持的并不是很好， 没有直接使用compass来的方便，省心。这里我们不选。

大家注意看， 在项目的生成过程中，脚本自动运行了 bower install, 根据bower.json 这个配置文件去安装我们必须的库和组件。在这里我们所依赖的库和组件有什么？ 有 bootstrap, 有 modernizer, 有 jquery。 好， 我们看到已经安装好了。

大概浏览一下项目的目录结构。 。。 重点介绍 Gruntfile.js 这一Grunt的配置项。

使用自动收起代码

‘use strict’ 不用多说， 这是声明以下代码全部遵守ES5的严格模式，严格模式呢， “严格模式”体现了Javascript更合理、更安全、更严谨的发展方向。

继续往下， 是module.exports = function (grunt) {} 的形式， grunt作为函数参数传递。 每一个Gruntfile 和 gruntplugin 都有这样一个最外层的基本格式， 我们把它称为 wrapper function。 所有的Grunt的配置以及逻辑代码都要放在这个函数里边。

继续往下则涉及到了Grunt最重要的一个概念， 那就是Task， 任务。Grunt把代码压缩，目录清除， 创建目录等等这样一个又一个的操作， 称为一个又一个的Task。在initConfig中的，我们配置的是一个又一个的单元Task，每一个Task都可以单独存在和执行。那这里的属性， 或者说配置应该怎么写呢？ 答案就是随便写。我们注意看到第一个属性是config， 第二个属性是watch， 这两个的不同在于？ 我们有一个真实存在的Task叫做grunt-contrib-watch， 而这个task在运行的时候会尝试着去读取initConfig 中的同名属性，watch这个配置项，然后按照这个配置项的设定来运行。但并没有哪个Task叫做config，也就意味着没有哪个task会直接读取initConfig中的config这个配置项，对于这种不针对于任何task的属性，Grunt会把它的值作为一个常量储备起来， 已备我们后边通过<% %>这种模板的方式， 对常量加以引用（到watch:JS配置项中）。所谓随便写就是愿意写多少，怎么写都行， 只要不跟正常的Task所读取的属性冲突即可。当然了这里可以是任何有效的JS表达式。也就是说，其实我们可以动态的生成我们的Grunt配置。比如说有一种常用的方式就是：pkg : grunt.file.readJSON(‘package.json’) 把package.json文件中的信息读取进来，用作配置项常量储存起来, 我们看到这里生成的配置项是config， 用来存放app源文件目录和dist目标文件目录。 那这么做的好处是啥？ 是不是我们可以把文件路径之类的东西写在这里， 一来统一，二来方便将来修改。所谓人无远虑，必有近忧。

那我们如何指定Grunt去加载哪个task呢？grunt也提供了统一的写法：

grunt.loadNpmTask(‘grunt-contrib-watch’);

"grunt-contrib-clean": "^0.6.0",

"grunt-contrib-concat": "^0.5.0",

"grunt-contrib-connect": "^0.8.0",

"grunt-contrib-copy": "^0.5.0",

"grunt-contrib-cssmin": “^0.10.0",

那如果我这么一个个的去写是不是太累了。所以有个哥们就写了一个load-grunt-tasks 的npm组件， 一句话， （屏幕切到 require(‘load-grunt-tasks’)(grunt)）就将所有package.json（切到package.json） 中声明的grunt-* 的 dependencies 全部加载进来了。

顺便插一句：写这个组件的哥们叫做：Sindre Sorhus（xindere suosi）挪威的一个25岁的小伙。在github上相当有名，基本上github几大前端项目组织，他都在其中。对前端有兴趣的同学可以关注一下他。（切到Sindre Sorhus的github地址）。 回到我们的话题，如果不小心dependencies 加载多了会不会有问题呢？ 当然不会， 只要我们在initConfig中没有同名配置项， 就不会造成任何的副作用。

每一个Task呢又会包含自己的target 和 options， 为了说明两者的区别呢， 我们继续往下看。 watch下边包含了很多的属性，我们认为除 options （移动到下边的connect）之外的任意命名的属性，都是target。而Options属性对应的， 则是我们可以为这个Task（connect）或者这个Task下的某个target （watch）设置的配置项。

这里呢， 我来做一个形象点的类比， 至于是不是恰当，大家只好凑合着听了。想想《生活启示录》里边的那句经典台词： 这个世界有95%的婚姻是凑合的。您也就别要求太高了。

(我们每个人从小都有一个大志向，那就是维护世界和平。反恐+解放

我们实施目标的过程就是运行Grunt的过程。

这个过程中，我们有若干的任务需要执行， 比如说消除贫困， 消灭恐怖头脑， 这就是一个又一个Task。

单就消灭恐怖头脑这个Task， 我们的目标可以是 本拉登， 也可以是金三胖， 这就是我们的Target。 在执行消灭恐怖头脑的时候， 我们有一个指导原则那就是： 除恶务尽。 这就是配置在我们消灭恐怖头脑这个Task下的options。 对于金三胖这种暗杀成功率会高一些， 暗杀就是我们配置在金三胖这个Target下的options， 而本拉登呢， 则可以直接使用最先进的军事打击。最先进的军事打击就是我们配置在本拉登这个Target下边的options。 - 因涉及政治敏感话题此举例被拿掉)

假设我们正在进行一盘三国杀8人局的游戏，当前的身份是忠臣，那我们这一局有三个任务要坚守。1. 保护主公 2. 杀死反贼 3. 干掉内奸， 保护主公，杀死反贼，干掉内奸这就是一个又一个的Task。

单就杀死反贼这个Task来说，我们的目标可以是黄盖，可以是貂蝉，可以是孙尚香，还可以是张角。（这里我们假设反贼是这四个人物），那黄盖，貂蝉，孙尚香，张角就是我们的Target。

在打击反贼的时候，对孙尚香出杀， 但是对张角呢，我们以决斗，南蛮入侵为主。出杀和决斗， 南蛮入侵 就是我们为不同的Target设定的options。

具象到Grunt，这些是如何使用的呢?

我们先继续往下看还有哪些Task的配置项：（记得使用sb的收起功能）

好，浏览完了， 我们挑其中一个来说明Task，Target，options具象到grunt是如何使用的。就挑Sass这个task吧。sass这个task干的事情就是把scss文件生成对应的css文件到相应的目录。dist 和server target呢都是生成到.tmp这个目录，为了演示方便，这里我们把dist的生成目录改成dist。 而在我们的源文件目录 app/styles/ 有一个main.scss。我们首先看当前项目并没有一个.tmp目录，也没有一个dist目录。task的运行方式呢就是直接在grunt后边跟task的名字，比如说我们运行sass，就是 grunt sass。我们发现项目目下分别多了一个.tmp 和 dist 目录， 进去看一眼。 ….. 默认情况下，grunt会运行task下的所有target。 如果我只想运行dist这个target呢？ 先把.tmp 和 dist目录删掉， 在运行 grunt sass的时候，把target用分号的方式跟在后边， 就是指定target的意思。（演示。。。）

在sass task下已经配置了一个options，指定了sass的import路径，我们把这个配置项移到dist target中。这时候再运行 dist这个target ok，没问题。 但是运行 server这个target呢？必然报错， 因为这个配置项并不对server 这个target生效。

孙权， 对就是那个三国杀中可以 制衡 的孙权曾经说过：能用众力，则无敌于天下矣；能用众智，则无畏于圣人矣。 那我们维护世界和平的过程中， 为了加快进程，提高效率，是不是也要多个task同时运行啊。那如何组合呢？ 我们回到Gruntfile.js , 在 initConfig 之后有几处 registerTask, 首先呢， 指定一个组合Task 的名字，后边的参数呢既可以是一个function，像server这样。又可以直接是一个数组。像build这样。 函数的方式，在其中呢， 通过 grunt.task.run() 指定要运行的一个个的Task。数组则直接跟需要运行的一个个task。而组合后的task呢， 又可以被再次组合， 比如说build， 就又被组合进了default这个task。 组合的任务和单个任务的运行方式一致， 比如说 build 这个task， 我们直接运行grunt build。当我们直接运行grunt，后边不跟任何参数的时候， grunt会默认执行default这个task。

大家注意执行完了有一个 执行时间的汇总。我们回到gruntfile.js 的开头部分， 发现这里引入了一个time-grunt 的npm组件。只需要require一下， 无需任何配置， 它便会帮统计所有grunt执行任务的时间。顺便提一句，这个组件也是上边膜拜过的xindere suosi 写的。

我们今天扫过的这些个task，都是我们实际生产中经常用到的，详细讲解每一个任务都会耗费一定的时间， 这节课呢，我们主要是对grunt的task,target,options有一个初步的认识。在下一节呢，我们开始深入到这些task的细节当中。

好， 这节课就到这里， 谢谢到家。

第7节 Grunt In Action - 从无到有打造Grunt配置

大家好，我是mater，上一节呢， 我们通过yeoman生成的webapp项目，一起初识了Grunt的Task，Target和Options。这一节呢，抛开yeoman不用，这次不是麻烦来找我们，是我们主动找上了麻烦，只为一探老项目如何整合Grunt的神奇魔力。

引用《霸王别姬》中的一句经典台词： 老规矩了，多少年的老规矩了！ 我们应该 。。

首先创建一个测试目录， grunt-in-action, .. 然后创建一个项目目录 grunt-empty…

为了打造一个传统项目的样子， 我们在目录下创建index.html js/index.js

接下来我们就开始在这个传统项目的基础上来整合Grunt的神奇魔力。我们先创建一个app目录，然后把所有的老项目源代码放到这个目录下。

然后我们使用npm init 命令，生成最基本的package.json的配置。老项目下一样可以在项目目录运行这个命令，不会对原有代码造成任何影响。如果老项目本身就是一个包含package.json文件的node项目，则可以忽略这一步。

description: learn grunt by the empty project

entry point: 是指我们在项目目录运行node 命令的时候的入口文件， 这里我们用不到，用它默认的index.js 就好了。

test command: 测试命令， 留空

git repository: git 仓库地址， 留空

keywords： 项目的关键词， grunt， empty， merge

author： 作者，materliu

版权协议：ISC , 这里顺便跟大家提一下，我们常用的几种开源许可证协议。ISC呢功能上跟两句版的BSD协议相同，基本上允许开发者为所欲为，可以自由的使用，修改源代码，也可以将修改后的代码作为开源或者专有软件再发布。 但”为所欲为”的前提当你发布使用了BSD协议的代码，或则以BSD协议代码为基础做二次开发自己的产品时，需要满足三个条件：

1. 如果再发布的产品中包含源代码，则在源代码中必须带有原来代码中的BSD协议。

2. 如果再发布的只是二进制类库/软件，则需要在类库/软件的文档和版权声明中包含原来代码中的BSD协议。
3. 不可以用开源代码的作者/机构名字和原来产品的名字做市场推广。 BSD 代码鼓励代码共享，但需要尊重代码作者的著作权。
   BSD由于允许使用者修改和重新发布代码，也允许使用或在BSD代码上开发商业软件发布和销售，因此是对商业集成很友好的协议。而很多的公司企业在选用开源产品的时候都首选BSD协议，因为可以完全控制这些第三方的代码，在必要的时候可以修改或者二次开发。

类似的还有Apache Licence 2.0， 需要在二次开发的代码中包含原来代码中的协议，商标，专利声明和其他原来作者规定需要包含的说明。

我们很熟悉的Linux使用的是GPL协议， 这个跟BSD，Apache License等鼓励代码重用的许可很不一样。GPL的出发点是代码的开源/免费使用和引用/修改/衍生代码的开源/免费使用，但不允许修改后和衍生的代码做为闭源的商业软件发布和销售。

而最宽泛的莫过于MIT了，作者只想保留版权,而无任何其他了限制.也就是说,你必须在你的发行版里包含原许可协议的声明,无论你是以二进制发布的还是以源代码发布的.

如果按照几者宽严的程度来排个序的话，MIT 》 BSD（ISC） 》 Apache 》 GPL

看一眼，预览， ok。

这时候我们看到目录下边多了一个package.json文件，我们看一下眼， 这个时候并没有 devDependencies 或者 dependencies 的依赖项。

那如何将grunt引入呢，我们之前说过grunt其实本身只是一个npm的package，那我们就安装这个npm的package就好了， npm install grunt 如果想在安装的时候，同时更新 package.json 文件中的依赖项呢， 在后边跟一个 —save-dev 的命令， 就在安装package 的同时，将它添加到devDependencies中了， 如果是 —svae 的命令，则是在安装 package的同时，将它添加到 dependencies 中。

好， 安装完了，我们看一眼。发现多了一个devDependencies的项， 里边有grunt。

在上一节，我们讲到了Sindre Sorhus这哥们开发的两个插件， load-grunt-tasks 和 time-grunt 把这两个插件也安装上。

好， 安装完了，我们看一眼。发现devDependencies的项里多了load-grunt-tasks 和 time-grunt 。

接下来就要编写我们的Gruntfile.js 文件了。 生成一个Gruntfile.js 文件， 在编辑器中打开我们的项目。

```
'use strict';  （声明使用ES5的严格模式）

module.exports = function (grunt) {  (声明Grunt最基本的框架）


    // Load grunt tasks automatically

    require('load-grunt-tasks')(grunt); （将load-grunt-tasks组件引入)


    // Time how long tasks take. Can help when optimizing build times

    require('time-grunt')(grunt); (将time-grunt 组件引入)


   // Configurable paths

    var config = {  (配置我们的项目路径）

        app: 'app',

        dist: 'dist'

    };


   grunt.initConfig({    (任务配置）


        // 引入 我们的config设置

       config: config

});

}
```

这个时候其实还没有任何的task以及配置， 对吧， 那我们如果运行grunt会怎样呢?

接下来呢， 我们就要添加一些Task， 最常用的Task是： 文件删除clean，文件拷贝copy， 总之都是跟文件相关的。这些最基本的Task，Grunt都提供了官方版本。那么我们就来看看Grunt里边的文件是如何被操作的。

首先我们安装一下这几个最基本的task(别忘了加 —save-dev)。Grunt的文件拷贝依赖于官方的Task，叫做grunt-contrib-copy, 文件删除依赖于官方的grunt-contrib-clean, 安装好了以后，我们就来配置一下。

首先我们来配置一下copy命令

```
// Copies remaining files to places other tasks can use
    copy: {
        dist: {  // 添加一个target， 取名dist， 表示这个target是把文件拷贝到dist目录去。下边就是指定要拷贝哪个文件，拷贝到哪里去。
            ——-  files 这段后边一段叙述完了在添加。
            files: [{
                expand: true,
                dot: true,
                cwd: '<%= config.app %>',
                dest: '<%= config.dist %>',
                src: [
                    '{,*/}*.html',
                ]
            }
        },
```

因为大部分任务都有文件操作， 所以Grunt在这块呢进行了一定的抽象。一共有几种定义源文件到目标文件的文件映射方式。Grunt的Task一般都能支持到， 所以就需要我们在实际需求中选择哪种是最适合我们的。不管哪种都支持 src 和 dest的声明， src表示源文件， dest表示目标文件

1. 第一种我们称之为： Compat格式

这种模式下，每个Target下，可以放置一对 src-dest 的文件映射，一般我们是在那些只读的任务里边使用， 这个时候就只需要写一个src参数， 而不需要dest参数。这里呢， 我们在copy dist这个target中演示一下。

```
        dist: {
            src: '<%= config.app%>/index.html',
```

```
        dest: '<%= config.dist%>/index.html'
    }
```

声明一个src，这里既可以是一个数组， 也可以是一个单独的字符串。这里我们引用之前保存的Grunt的config配置。

好， 我们来运行一下。

我们发现项目目录下多了一个dist目录， 其中拷贝进了 index.html

好， 有了dist目录， 我们配置一下目录清除的任务: clean。同样给他创建一个dist target，

```
clean: {
    dist: {
        src: '<%= config.dist %>/index.html'
    }
}
```

只需要配置src， 无需dist。 运行一下。我们发现dist目录下的index.html 文件被清除了。

这个时候我们再回到copy任务， 修改一下target， 我们改为 dist_html和dist_js 分别来拷贝js和html。

```
copy: {
    dist_html: {
        src: '<%= config.app%>/index.html',
        dest: '<%= config.dist%>/index.html'
    },
    dist_js: {
        src: '<%= config.app%>/js/index.js',
        dest: '<%= config.dist%>/js/index.js'
    }
},
```

直接运行 copy这个task 。。。。。

这个时候如果我们想clean多个文件呢， 我前边说了 src支持数组的参数形式，我们修改 clean task

```
clean: {
    dist: {
        src: ['<%= config.dist %>/index.html', '<%= config.dist %>/js/index.js']
    }
}
```

……

回到我们的copy命令， 无故多出了一个Target 是不是很烦。 怎么办呢？ 这时候就可以用我们要讲的第二种 Files Array Format。

以数组的形式， 组织多个 源文件到目标文件的映射。好， 我们改造一下 copy 这个 Task。

```
concat: {
  foo: {
    files: [
      {src: ['src/aa.js', 'src/aaa.js'], dest: 'dest/a.js'},
      {src: ['src/aa1.js', 'src/aaa1.js'], dest: 'dest/a1.js'},
    ],
  },
```

好， 先运行一下 clean， 然后运行 copy task ……

还是要写很多对 src 和 dest 也够烦人的， 那有没有啥更好的方式呢？ 这就涉及到了我们要讲的第三种 。 Files Object Format

变成了一个object键值对的形式， 只不过key就是 目标文件， value则是源文件。

好， 我们改造一下 copy 这个 Task。

```
concat: {
  foo: {
    files: {
      'dest/a.js': ['src/aa.js', 'src/aaa.js'],
```

```
      'dest/a1.js': ['src/aa1.js', 'src/aaa1.js'],
    },
  },
  bar: {
    files: {
      'dest/b.js': ['src/bb.js', 'src/bbb.js'],
      'dest/b1.js': ['src/bb1.js', 'src/bbb1.js'],
    },
  },
},
```

好， 先运行一下 clean， 然后运行 copy task ……

这时候回到我们的clean命令， 我们的目的其实很明确，我们想删除dist目录下的所有文件，其实并不需要指定的这么详细。这时候利用 globbing patterns 即可轻松实现, 在计算机编程里边，尤其是在类unix环境下， 基于通配符的类型匹配我们就称之为Globbing。因为Grunt是基于Node构建，而Node本身呢也支持Globbing。 好，我们改造一下clean命令。

```
clean: {
    dist: {
        src: ['<%= config.dist %>/**/*']
    }
}
```

好， 运行一下。

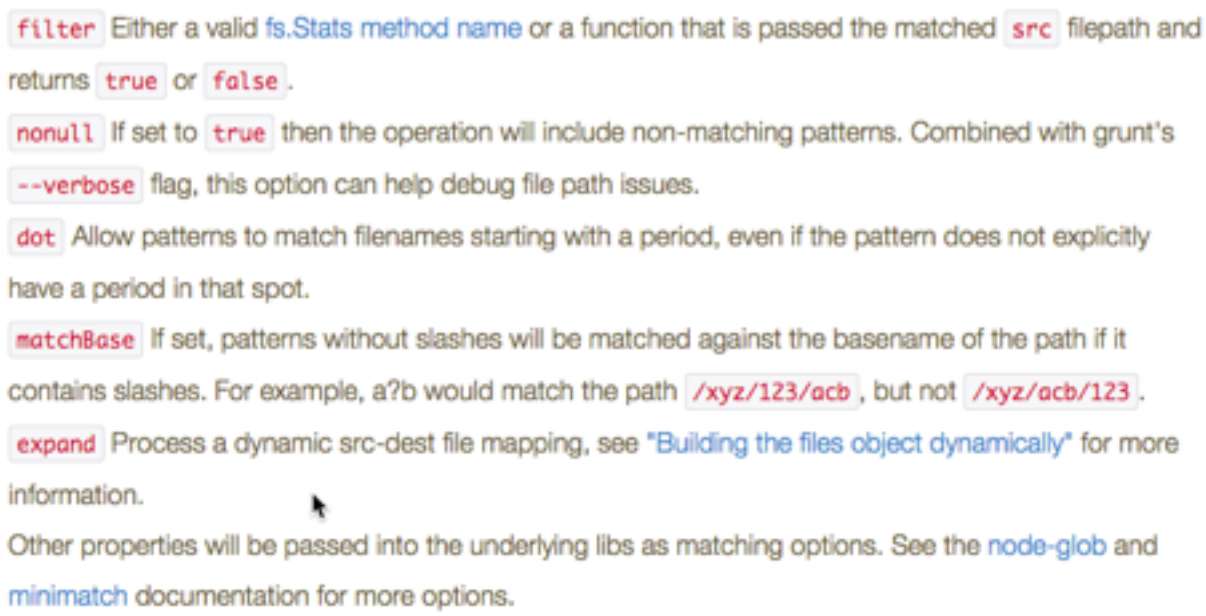
大家注意，我们这里 用了两个** 后边又跟了一个反斜杠 一个*

- `*` matches any number of characters, but not `/`
- `?` matches a single character, but not `/`
- `**` matches any number of characters, including `/`, as long as it's the only thing in a path part
- `{}` allows for a comma-separated list of "or" expressions
- `!` at the beginning of a pattern will negate the match

这些大家在我们后边详细讲解其他常用Task的时候都会碰到。

这个时候我来创造一个伪需求。先运行一下copy命令， 生成dist目录。如果在清除的过程中， 我并不希望js这个文件夹也被清除， 那应该怎么过滤呢？这时候就引出了我们要谈的下一个问题， 那就是额外参数。

注意，额外参数仅支持 Compat formats 和 Files Array formats，就是我们讲到的第一，第二种。额外参数都有哪些呢？



- `filter` Either a valid fs.Stats method name or a function that is passed the matched `src` filepath and returns `true` or `false`.
- `nonull` If set to `true` then the operation will include non-matching patterns. Combined with grunt's `--verbose` flag, this option can help debug file path issues.
- `dot` Allow patterns to match filenames starting with a period, even if the pattern does not explicitly have a period in that spot.
- `matchBase` If set, patterns without slashes will be matched against the basename of the path if it contains slashes. For example, a?b would match the path `/xyz/123/acb`, but not `/xyz/acb/123`.
- `expand` Process a dynamic src-dest file mapping, see "Building the files object dynamically" for more information.
- Other properties will be passed into the underlying libs as matching options. See the node-glob and minimatch documentation for more options.

filter： filter的值既可以是nodejs中 fs.Stats类下的一个函数的名字。比如说isFile http://nodejs.org/docs/latest/api/fs.html#fs_class_fs_stats 也可以是我们自己构造的一个函数， 这个函数通过返回true或者false来指示当前的filepath是否命中。比如我们这里可以借助Grunt的内置函数来如此实现：

```
filter: function(filepath) {
        return (!grunt.file.isDir(filepath));
    },
```

nonull: 不多说， 一般用于调试。

dot： 如果为真的话， 则尝试命中 以点开头的文件。比如说 我们写 **/index.html 会命中 .index.html

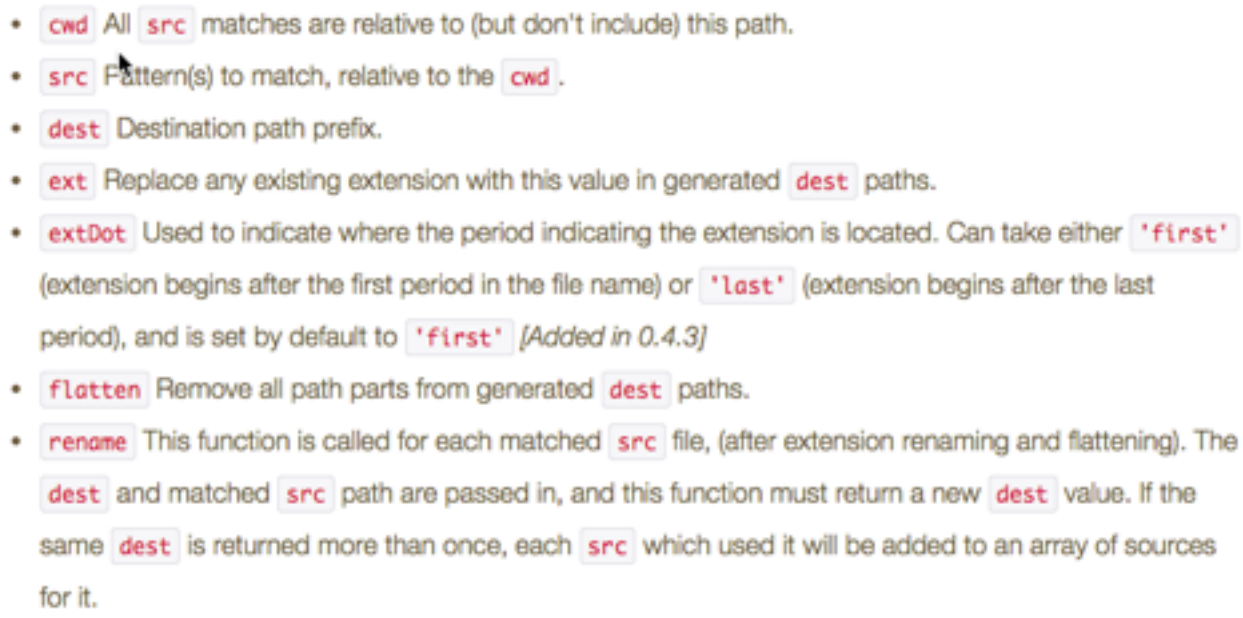
matchBase: 直接举例说明。 a?b 默认会命中 /xyz/123/acb, 和 /xyz/acb/123， 但如果我们把 matchBase 设为true， 则a?b 就只会命中 /xyz/123/acb 不会命中 /xyz/acb/123 了。就是只去匹配 路径的 basename

expand: 如果设为true， 则意味着处理动态的 src-dest的文件映射。

其他的属性则会直接透传给grunt依赖的node底层的 node-glob和 minimatch ， 一般我们都用不到， 这里就不再赘述。

回到expand， 我们来看看动态的src-dest 文件映射是怎么用的。

首先，我们将copy task改为file array formats 的形式， 将expand参数设置为true。



- `cwd` All `src` matches are relative to (but don't include) this path.
- `src` Pattern(s) to match, relative to the `cwd`.
- `dest` Destination path prefix.
- `ext` Replace any existing extension with this value in generated `dest` paths.
- `extDot` Used to indicate where the period indicating the extension is located. Can take either `'first'` (extension begins after the first period in the file name) or `'last'` (extension begins after the last period), and is set by default to `'first'` *[Added in 0.4.3]*
- `flatten` Remove all path parts from generated `dest` paths.
- `rename` This function is called for each matched `src` file, (after extension renaming and flattening). The `dest` and matched `src` path are passed in, and this function must return a new `dest` value. If the same `dest` is returned more than once, each `src` which used it will be added to an array of sources for it.

动态的src-dest 支持以下这些参数。 官网 接下来我们就来一一应用一下这些个参数。

。。。。。

好， 这节课呢，我们通过grunt-contrib-copy 和 grunt-contrib-clean 这两个Task，彻底讲解了Grunt的几种file formats 和 如何将Grunt引入到遗留项目中。 打好基础我们接下来就要深入认识一下Grunt的各个Task了。欲知后事如何，且听下回分解。 好， 谢谢大家。

第8节 Grunt-In-Action 常用Grunt Task剖析

大家好，我是mater，上一节呢，抛开yeoman不用，我们学习了如何在传统项目中整合Grunt，并在这个过程中了解了Grunt中相当重要的几种Files Format。加上我们之前讲过的Grunt的Task, Target 和 Options 从而为我们奠定了继续窥秘Grunt的基础。 接下来我们则将回到yeoman创建的webapp，逐个逐个的讲解我们常用的Task， 以及这些Task是如何组合应用的。

借用《闪闪的红星》中的一段经典台词， “我胡汉三又回来啦”， 道出yeoman的心声。 好， 这里我们创建测试目录 grunt-in-action, .. 创建一个项目目录 yo-webapp-grunt, 利用 webapp 这个generator 生成我们的测试项目。

生成好了以后， 我们直扑主题： 一起来看看Gruntfile.js 文件中的各项任务配置， 不过呢， 我们采用倒叙的方式， 从组合任务看起。 为什么这样呢?

所谓兄弟齐心 其利断金， 成功需要组合拳， 在实际的项目生产中，我们往往是直接应用这些组合任务。

第一个组合任务 serve， 我们直接运行看效果。 发现我们的浏览器窗口被直接打开定位到了 localhost:9000， 而页面内容呢则是yeoman自动生成的我们的项目站点， 这个组合任务让我们可以实时预览本地项目在浏览器中的渲染情况，当我们修改项目中的文件的时候， 浏览器呢也会自动刷新。 好，我们来试一下， 这里呢， 我们把提示语 Allo, Allo 改为， Hello Grunt! 根据我们之前讲到的项目结构， 源代码全部是放在app目录的，打开app目录找到index.html 文件，搜索 Allo, 修改， 保存。 我们看到浏览器自动刷新页面了。

serve呢是之前讲过的， 在registerTask 的函数调用中， 通过传递函数参数来自定义。 我们看一下函数中的定义。首先判断，在启动这个命令的时候， 是否带有参数 allow-remote 如果带这个参数， 会将connect 这个task中的options配置项中的 hostname 的值改为 0.0.0.0，connect task呢简而言之就是在本地起一个webserver， 等一下我们会详细看， 这里只看这个配置项， hostname的默认值是 localhost ， 本地serve起来之后， 我们只能通过 localhost:9000 或者 127.0.0.1:9000 的方式访问，我们来看一下我机器现在的ip， 我机器现在的ip是： 10.237.210.7， 那我直接通过这个ip在我的本机访问能访问到这个站点吗？ 我们来试一下。。。 发现访问不到。 那跟我在同一局域网内的机器，可以通过 10.237.210.7:9000端口访问到我的站点吗？ 在这个配置下依然是不可以的。 怎么样就

可以了呢， 那就是把hostname的配置项改为0.0.0.0 。也就是这里所说的允许 remote access 。 （演示 ）怎么在执行的时候添加这个参数呢， 我们先将当前的server ctrl + c 终止掉，在执行 grunt serve 的时候 通过 —allow-remote 将这个参数跟在后边， 这个时候，我们注意到浏览器默认打开的已经是 0.0.0.0:9000 了， 这时候再用我的本机ip访问试一下， 发现访问ok。 一样的道理， 只要能连通我的机器的用户， 现在都可以直接我的站点。

继续往下看， 是判断函数参数， grunt在执行这个函数的时候， 会把 当前组合task的target传递进来， 这里的判断， 如果target的值为dist， 那么则执行 build 和 connect,

执行connect这个task的时候呢， 选择的是dist这个target， 后边跟了一个keepalive 是啥意思呢？根据我们之前讲到的target的知识， 很多同学会错误的把这个参数也认为成是一个target, 但我们到 connect 这个task中呢，并没有看到 keepalive 这样的一个target配置，其实Grunt本身呢也不支持这样配置多个target的语法。这是什么原因呢？其实 keepalive 是grunt-contrib-connect 这个task的一个配置项， 默认情况下呢， 一旦Grunt的任务完成了， web server 也就会随之退出（这种场景在执行自动化测试的时候用的比较多，server跑起来执行测试代码， 测试代码执行完， server退出）， 但这里，我们要想让server一直起着， 方便我们调试， 就得指定connect的keepalive options 配置项。普通的做法应该是在connect task的dist target 的 options 里边这么配置, 这样做可以吗， 当然是没问题的。 但是文艺青年就不这么干了。grunt对于 grunt task:abc:def:ghi 的这种写法，在执行task的时候呢， 在task的注册函数体内， this对象下会有一个 falgs的变量，this.flags 它的值呢就是 { abc: true, def: true, ghi: true }, 而connect就在这里做了一下文章，在执行的时候先去取 this.flags.keepalive的值，如果为truty的值则直接使用， 如果没有或者为falsy的值， 再去取 options.keepalive, 如果为truty的值则直接使用， 否则就用默认的keepalive false的值， 从而避免了 options 的使用。这跟我们常说的冷热启动，冷热插拔的概念有点像，一个是运行前就已经决定了， 一个是运行的时候才知道配置。这里额外提一句， 如果abc是一个target的名字同时这个task， 又是通过grunt的grunt.registerMultiTask 注册的， 那么abc这个target的名字， 是不会被设置为 this.flags 的一个参数的。 而一般我们用到的grunt的原子插件， 都是用 grunt.registerMultiTask 注册的， 它与 我们现在说的 grunt.registerTask的不同， 就在于 grunt.registerMultiTask 允许我们为一个task配置多分属性，参数， 其实就是配置多个target的意思。 这些我们在后边一课将为grunt编写我们自己的插件的时候会详细说到。

好， 抽丝剥茧，我们继续往下， 等一下再看build和connect:dist:keepalive 干了什么。

深入到单元Task，每一个单元Task在github上都有对应的项目，上边有关于这个Task的所有详细的参数讲解， 当遇到实际生产中解决不了的问题的时候， 大家可以先上相关的github项目地址看看有没有啥解决方案。

clean:server，跳到clean这个task， 在上一节呢， 我们就已经介绍过这个task，其功能，单一，实用， 就是删除文件以及文件夹。 看一下server这个target， 大家注意， 这里的filePath 是直接跟在target name之后的， 这是我们之前没有讲到的grunt 的files format，它的应用场景很小， 只能应用于没有destination文件路径的task, 比如说clean。 server这个target是删除.tmp文件夹， 说明我们的临时文件都是生成在.tmp这个文件夹的。在运行serve组合任务之前先清空一下， 以免留下什么后患。

wiredep： 这个则是我们要讲的重头戏， 为什么呢？ 大家还记得在bower那一章我们曾经说过， 用原始的方式引入bower install的js， css 依然有点麻烦吗， 当时的悬念便留到了这一节， 而wiredep就是用来解决这个麻烦的。

wiredep 通过bower.json文件配置，找到我们的components依赖， 并将其引入到我们指定的HTML文件中。我们看一下它的配置，有两个target， app和sass， 通过src指定待处理文件， app指定我们的index.html 文件， sass指定我们的sass文件， ignorePath指定在插入我们的引用文件的时候，排除掉filepath中的哪一部分。 我们看一眼index.html 文件对bower components 中 js 和 css的引用， 注意， 只有一句简单的 bower:css …. endbower bower:js endbower， 现在我们看到 bower:js 区间内已经有内容了， 是因为我们之前运行grunt serve的时候，顺带运行过grunt wiredep了， 这里我们删掉这些内容， 重新运行一下。发现相关js都被填充进来了。注意看这里的路径，直接上来就是bower_components ， 我们将app配置中的， ignorePath 注释掉， 再重新看一下效果，发现这时候多了 ../ 这是因为默认wiredep， 在插入的时候， 路径是取的 components 相对于被插入文件的路径， 而我们在最后将index.html 和 components 文件进行进一步的grunt task的处理的时候， 实际上改变了他们的相对路径， 变成了同级文件结构， 所以这里通过ignorePath 进行这个处理。 sass中的配置也是同理， 我们看一眼main.scss, 也是一样的 bower:scss 和 endbower , 我们先删掉这句 @import, 这是sass中的引入语法。运行 wiredep 发现这句又自动加回来了。

app 中多了一个exclude的配置项：指定在插入过程中， 我们要排除的被插入的引用文件。这里指定排除 bootstrap.js。 为什么呢？因为在bootstrap的components中，这些分散的组件全部加起来其实就是bootstrap.js 如果这时候再引入bootstrap.js 就相当于重复引用， 没有意义。

好， 回到grunt serve， 我们继续往下看

concurrent:server 说文解字， 看到这个单词，我们大家应该已经明白什么意思了。默认情况下grunt的task都是串行的， 一个执行完了，另一个才继续执行， 这对于那些没有先后依赖关系的task其实完全没有必要， 这时候我们就可以依靠concurrent 来将它们并行执

行。server target 指定同时运行sass:server 和 copy:styles, 我们进一步看一下这两个task都干了些啥。

sass Task 之前我们已经讲过， 干的事情很简单， 就是将sass文件编译成对应的css文件，并输出到指定目录， sass：server 将app/styles/下的sass文件编译成css输出到.tmp/styles 目录

copy Task 我们之前也已经讲过， 简单方便快捷的文件拷贝。copy:styles 将app/styles 下的css文件拷贝到 .tmp/styles 目录。

好， 回到grunt serve， 我们继续往下看

autoprefixer: 是一个css处理task， 我们知道有些css属性， 比如说：（在main.scss 中演示） transform transform: scale(2, 4); 这一css3属性， 在不同浏览器下需要添加不同的浏览器前缀，以往我们的做法是在写的时候， 就一条一条的写好， 非常麻烦。而 autoprefixer 就是帮我们把css文件中需要厂商前缀的这样的属性自动添加上厂商前缀。

这里的配置， 我们看到是处理.tmp/styles 目录下的css文件， 处理完覆盖输出到.tmp/styles文件夹。options指定了浏览器的过滤条件

市场占有率 > 1%, 每种浏览器最新的两个版本， 最新的Firefox ESR 版本， 以及opera的12.1

我们运行看一下

好， 回到grunt serve， 我们继续往下看

connect:livereload

grunt-contrib-connect 这个task 是在非常出名的 node connect 上二次开发为grunt做的封装， 前边我们说了， 就是在本地起一个webserver, 看一下它的配置项

指定我们server的端口号开在9000端口， 起来之后是否默认打开浏览器， 这里是默认打开。

livereload 配置项， 我们知道， 仅仅是在本地开一个webserver, 是没办法做到我们修改本地文件的时候让浏览器自动刷新的， 这里呢， connect 通过 connect-livereload这个组件来部分实现这个能力。 为什么说是部分实现呢， 先卖个关子。讲下一个Task的时候，我们再说。这里是指定livereload 占用的端口号。

hostname 前边已经花了很大的篇幅说它， 不多讲了。

livereload target的配置，middleware 可以是一个数组，也可以是一个函数， 函数最终也是返回一个数组。 connect作为函数的形参被传进来。 node connect 呢，我们不多讲，大家有兴趣，可以去看它的各种用法。 这里简单描述一下： static 指定serve的内容是哪些， 第一句指定 将.tmp 目录下的内容添加到/ 根路径的路径匹配， 第三句指定 将app 目录下的内容添加到/ 根路径下的路径匹配， 第二句为bower_components 下的内容修改了一下路径匹配， 这次不是匹配根路径了， 匹配 /bower_components 路径。

好， 回到grunt serve， 我们继续往下看

watch Task 用于监听本地文件的修改， 一旦监听到了， 则运行options中配置的task。

我们看一下watch的配置, watch下有N多的target， 我们一个一个来过。

files 用来指定监听的文件， tasks用来指定当监听到对应文件变化，应该执行的任务。

bower， 当监听到bower.json文件修改了， 重新执行一下wiredep这个task。

js： 当监听到我们的app/scripts 目录下的js修改了， 执行jshint校验，检查我们的代码书写是否符合规范。

livereload 配置项， 这个的值可以为真， 为真的情况下默认取 35729, 这其实就是我们connect task中指定的端口号，当然也可以自定义一个端口号。watch遇到livereload配置项的时候会在后台起一个livereload 的server， 用来记录是否有事件发生需要reload, 有了这个配置项， watch监听到我们这里配置的js 文件变化的时候，执行完jshint之后，便会通知 livereload server， 需要 reload。那仅仅服务器知道了，并没有用。怎么通知到浏览器，让浏览器reload呢，传统的做法是在我们的html页面中嵌入一段 js， <script src="//localhost:35729/livereload.js"></script> js会跟服务器建立连接， 一旦需要reload， js便触发浏览器的reload。 但是这样需要我们手动写的方式多土啊。 所以connect 在serve我们的html文件的时候， 通过 connect-livereload这个组件 自动为我们的html文件插入了这段script 标签，从而避免了我们的手工劳作， 所以这也是我为什么在前边说connect只是部分实现了livereload, 因为它要跟watch 这个task相配合。

jstest： 当监听到我们test 目录下的js文件变化的时候， 执行 test:watch ， test我们先不展开， 后边细表。

gruntfile: 监听Gruntfile.js 的修改， 然后什么都不干。 什么都不干也不是一点用都没有，会在控制台给你个提示消息。

sass： 当监听到app/styles/目录 sass文件变化的时候， 执行 sass:server 和 autoprefixer 。

styles： 当监听到app/styles/目录 css文件变化的时候， 执行copy:styles 和 autoprefixer, 我们发现前边多了 newer ， 这其实是grunt的一个task plugin, 叫做 grunt-newer, 它的作用非常清晰， 就是只对新的，变化了的文件执行跟在后边的task。我们知道文件读写都是耗时操作，而copy:styles 呢，不管你具体变化的是哪个css文件， 会一股脑的将styles目录下的所有css文件都拷贝一遍，其实是非常没有必要的。将newer加在前边， 则只会拷贝新增的或者实际发生变化的css文件。使用方法就是这么简单， 将newer 跟在修饰的task前边。

livereload target，这里通过变量引用的方式， 获取到connect中配置的 livereload 端口号。

当监听到 app/ 目录下的html文件 或者 images文件， 以及 .tmp/styles 目录下的css文件变化时， 不执行额外的task， 直接触发 reload 通知。

watch task呢， 执行起来就会一直watch在这里， 即整个grunt程序不会退出。之前我们说如果没有 keepalive 选项， 一旦Grunt的任务完成了，connect 起的 web server 也就会随之退出， 那因为watch task， connect 起的webserver也会一直起在这里， 直到我们 control c 强制整个程序终止。

好，我们继续往下看

有一个 server的组合任务， 看它的提示语， 。。。 我们便知这其实是一个历史遗留问题。最早的 yoeman webapp generator 自动生成的项目中的grunt 这个任务就叫做 grunt server, 后来才改名动词 grunt serve. grunt server呢， 它实际上是去运行了 grunt serve, 透传我们指定的target。

好， 还剩三个组合任务呢， 我们留待下节再剖析。谢谢大家。

第9节 Grunt-In-Action 常用Grunt Task剖析2

大家好，我是mater，上一节呢，基于yeoman创建的webapp，我们深入学习了grunt serve这一组合task，及其所涉及的单元Task。这一节，我们就来了解一下剩余的test组合Task，及其背后的小老婆们。

实践是检验真理的唯一标准， 让我们直接运行一下grunt test， 看看发生了什么。在控制台输出了 1passing， 1passed 的消息， 这其实是告诉我们有一个测试用例， 这个测试用例成功通过。

好， 回到Gruntfile， 让我们看看在这个过程中， Grunt都干了些什么事情， 好看一眼:

首先判断target是否为watch， 当不为watch的时候执行以下三个前置操作：

clean:server 之前已经讲过， 首先清空 .tmp 目录。

concurrent:test concurrent 上一节，我们已经讲过其并行执行Task的作用，test target这里其实也只有一个task：就是将app/styles 下的css文件拷贝到 .tmp/styles 目录。

autoprefixer上节也已经说过， 就是帮我们把css文件中需要厂商前缀的这样的属性自动添加上厂商前缀。

当我们不希望CSS重新生成的时候， 就把 watch 这个target加上。感觉这里，它这个target的名字起得不是太好， 大家也想想有没有更好的名字。

接下来就是connect;test

上一节呢，我们也已经详细介绍过了connect这个task，以及其参数作用， 就不一一详细赘述了， 我们只说connect test 干了什么：将对根路径的访问，映射到.tmp test app 这几个目录寻路， 将对 bower_components 的路径访问， 映射到 bower_components 这个目录寻路。 test target 与 livereload 这个target的在路径映射上的不同之处就在于新增了 test 这个目录， 同时， 它的服务器端口开在了9001， 默认不打开浏览器。

回到test组合task， connect运行完了之后， 是mocha这个task， 我们接下来看看mocha干了些什么。

（打开mocha的主页）mocha 是一个javascript测试框架， 可跑在nodejs环境上， 也可以跑在浏览器中。 通过一个个执行我们指定的test case，最终生成精准的测试报告。因为时间关系， mocha我们不过细展开， 只讲跟我们当前 task相关的。

urls: 指定访问哪个webserver页面内容，来执行测试用例，（注意，这里我并没有说是通过浏览器去访问这个页面，下边会讲到底是如何访问的）。 一般mocha配置urls的时候都跟connect task 配合使用。 这里我们看到是访问test目录下的index.html， 通过变量引用的方法， 来使用在connect task的test target中指定的域名和端口号。hostname这一配置型， test target是从connect task的全局options中继承的。 细心的同学应该发现了， 在

connect task的test target中， 我们把app目录也映射到了根路径下， 而app目录中也有一个index.html 文件， 那难道不会冲突吗？ 这里呢， nodejs的寻路策略是， 先声明， 先寻路， 一旦命中， 不再继续。所以大家就明白了， 一旦找到 test/index.html， nodejs就不会再命中 app/index.html了。 而这里把app目录映射进来的原因是，我们的测试往往是针对app中的js文件的，是不可或缺的。

好，我们就看一下 test/index.html 文件都干了些啥。mocha 每个版本都有两个基本文件，那就是mocha.css和mocha.js 当在浏览器中运行的时候， 我们需要在html文件中，引入这两个文件， 这里，利用bower来进行的相关文件的管理下载，hardcode引入。接下来就是要告诉mocha，我们希望使用哪种接口，mocha本身呢，支持BDD，即： Behaviour Driven Development（行为驱动开发）行为驱动开发的根基是一种“通用语言”。这种通用语言同时被客户和开发者用来定义系统的行为; TDD, 即Test Driven Development （测试驱动开发）TDD的原理是在开发功能代码之前，先编写单元测试用例代码，测试代码确定需要编写什么产品代码；甚至是由你自己来自定义暴露你喜欢的风格的接口。这里我们声明使用BDD这种类型的接口。

同时，mocha允许用户使用任何的断言库，这一点也非常的灵活， 这里加载的是chai.js 这个断言库（打开chai.js 的主页）。然后将chai的变量，方法赋值到全局, 变相暴露。 加载存放测试用例的js， 加载完之后, 通过执行 mocha.run() 开始我们的自动化测试。我们看一眼test.js 里边的测试用例， 是一个自执行函数， 声明使用ES5的严格模式， describe 可以理解为定义一个模块的意思。 最外层模块的描述是： 来点内容吧。 里边子模块的内容是： 这里的内容得详细点。 it呢则负责描述测试用例， 第一个参数的描述是： 应当运行几个断言，(当然了， 上边这些描述都是毫无意义的戏虐的话， 真正编写测试用例的时候， 需要我们精准，明确的描述) 而真正的断言是写在它的第二个参数，即它的回调函数中的。 我们发现这里置空，留待我们真正开发项目的时候填写。 至于测试用例应该怎么写， 我们这里只简单演示一个： 利用equal断言， 即 表达式一的结果， 应该等于 表达式二。…. assert.equal(1, 3-1);…. 至于更复杂的测试，需要大家下去自行实践， 因为 mocha 我们完全可以另开一个系列来讲， 这里展开就说不完了。

那么index.html 也好， test.js 也好， 这些只是跑在浏览器里边的代码， 它是怎么把测试结果反馈到我们的命令行的呢?

我们这里通过一个小实验来说明： 我们上一节已经讲过， 如果我们没有指定connect的keepalive属性， 同时connect 后续的task 又没有阻塞grunt的退出， 那一旦grunt退出了，我们的connect 起的server也就关闭了， 所以这里为了让server能一直起着， 我们在后边加一个watch task。

执行。。。

好， 现在呢， 我们在浏览器里边打开， 发现测试结果被输出在页面上了。而页面上的内容跟我们控制台打出来的内容其实并不是完全一致的。那到底是怎么回事呢？这时候就不

得不介绍另一个东东了。（打开浏览器PhantomJS(/ˈfæntəm/)的主页）PhantomJS, 见名知意， Phantom本意事幻影。PhantomJS 我们可以把它理解成一个没有界面展现的浏览器。如幽灵般跑在后台，去执行相关的DOM解析，CSS渲染，JS操作。 Grunt-mocha就是基于PhantomJS来做我们这个自动化测试的。在PhantomJS实例通过我们指定的url加载我们的页面的时候，grunt-mocha在其中插了一段JS脚本，就像 connect task 在实现 livereload 时做的那样。而这段被插入的JS脚本则负责监听我们的测试用例执行情况， 最终将结果呈现在命令行上。在这段js脚本插入之后， grunt-mocha会为grunt添加一系列的监听器来监听每一种mocha Events, 这时候就要讲到mocha的另一个配置项， run: run的值默认为false， 当run为true的时候， 指定这些监听器都添加好以后， 插入的JS脚本去执行mocha.run() . 可能这时候就有同学要犯嘀咕了， 我们看到index.html 不是已经写了

<script> mocha.run() </script> 了吗， 为什么还要在去执行一次呢？ 大家注意， 被插入的这段用来监听mocha执行情况的JS脚本，是插入在我们文档的末尾的， 插入的时候 mocha.run() 已经被执行过了，因此grunt的监听函数也是在这之后才被注册的，如果这段JS脚本不再次执行mocha.run()， 那这些监听函数自然收不到任何的测试用例执行情况通知， 最后被触发超时。 我们把 run options 改为false 看一下。。。。。 我们发现出现超时的错误提示。

所以这里的写法如果改成：

```
<script type="text/javascript" charset="utf-8">
  // Only tests run in real browser, injected script run if options.run == true
  if (navigator.userAgent.indexOf('PhantomJS') < 0) {
    mocha.run();
  }
</script>
```

就完美了， 只有在真正浏览器中访问的时候，才执行我们提前写好的mocha.run, 在PhantomJS中，即使执行也没有意义， 干脆就不执行。

因为mocha.run() 是支持多次运行的， 所以这里直接写mocha.run 其实也是没有啥问题的。

好， mocha我们就讲这么多， 大家如果对mocha感兴趣， 觉得意犹未尽的话， 我们可以在留言中交流，再或者单独讲讲mocha。

本节就先介绍这么多， 下一节我们一气呵成， 讲完剩下的build和default 组合task。 谢谢大家。

第10节 Grunt-In-Action 常用Grunt Task剖析3

大家好，我是mater，前面呢，基于yeoman创建的webapp，我们深入学习了grunt serve， grunt test这两个组合task，及其所涉及的单元Task。这一节，我们就来聊一聊最后一个重头组合task， build。

依然还是那句，实践是检验真理的唯一标准， 让我们直接运行一下grunt build， 看看发生了什么。我们发现在我们的项目目录下的dist目录被生成。

好， 看一眼这个目录的内容。

有bootstrap依赖的字体文件， 压缩合并之后的js文件， 并且这些js文件都被重命名了，基于文件内容计算的md5值被加在了文件名的前边，md5算法，只要文件的内容不变，那么算出来的这个值就一定是不变的, 这里我们取这个值的前8个字符。大家晓得这样做的好处吗?

对于js和css，图片这些静态资源，为了提高它们的加载速度， 我们往往会把这些静态资源发布到CDN上来进行长缓存。

这里多说两句，CDN是利用Http Header 的Cache-Control: max-age 来进行缓存控制。一旦静态资源被缓存在浏览器端，页面再发起同一路径的请求的时候，只要没达到我们设置的过期时间，浏览器就直接去缓存中取了，不会再发起网络请求。

那这些资源修改了以后，我们怎么触发浏览器的更新呢？一般来说，我们的入口html文件，比方说 index.html , 是不做浏览器长cache的，使用的是http header Cache-Control:no-cache 来进行缓存控制。这种做法呢， 浏览器依然会把index.html放在缓存中，但每次访问的时候都会跑到服务器问一下，我目前的缓存有没有过期啊，服务器如果返回304说没有过期，那么浏览器则直接使用index.html文件的缓存，如果服务器说我们有新版本发布了，这个文件过期了，那浏览器则去向服务器请求新版的index.html文件，并更新自己的index.html文件的缓存。

以前的做法都是通过在html中被请求的文件url里边加一个querystring的 时间戳或者版本号，每次发布都更改一下这个querystring，这种做法跟利用md5重命名文件的方式相比有三点不足： 1. 如果js，css静态资源文件没有变更，理论上我们是不应该触发浏览器的刷新的，文件没变化，md5值就不会变化，文件名自然也不会变化。但是时间戳或者版本号的方式则比较粗犷，不管你的文件内容是否变更，时间戳和版本号在我们发布新版本的时候，都会更新。这就浪费了一次没有必要的网络请求更新。2. 版本号或者时间戳的方式在发布的时候是覆盖线上的同名文件。不管怎样，在发布的过程中，index.html和a.js总有一

个先后的顺序，从而中间出现一段或大或小的时间间隔。对于一个大型互联网应用来说即使在一个很小的时间间隔内，都有可能出现新用户访问。在这个时间间隔中，访问了网站的用户会发生什么情况呢？要么是新的index.html配合旧的js,css文件的情况，要么是新的js，css配合旧的index.html文件的情况。都有可能出现问题。3. 一些CDN的服务商，为了节省成本，降低带宽流量，遇到querystring变更的请求，并不会去上级服务器取，而依然使用自己机器上缓存的老版本的文件。从而导致更新失败。

继续往下看是从sass编译出来的css文件，同时也经历了压缩和重命名。

默认的Apache服务器的配置文件 .htaccess

favicon.ico 是显示在我们的浏览器地址栏或者收藏夹里边的网站icon(这里使用慕课网的站点来进行说明)。 一般的做法是直接把这个文件丢在我们网站的根目录里，浏览器则会自己去寻找并使用。但也可以在html文件里边显示调用，这里就是在index.html文件里边通过 link icon 的方式来显示调用的。显示调用的话名字也就随意了。1. 一般来说一个网站的图标不会随意更换 2. 很多时候， 我们的站点并不是只被浏览器访问的， 比如说1password，这种软件（实用软件演示）你输一个网址进去， 他会生成一个站点的缩略图， 它其实就是去取的你站点下的favicon.ico 图标。 所以这里做这个md5重命名其实意义不大，甚至还会有负作用。 我把这个问题提给了咱们之前聊过的那哥们Sinder sorhus, Webapp generator也是他开发的，所以等到你们亲自实践的时候发现favicon.ico不再md5重名了，也就不必惊讶了。

index.html 文件不用多说了， 就是我们的首页。我们发现其中对于静态资源的引用也全部自动更新了新的名字。

robots.txt 爬虫协议文件，我们通过这个文件告诉搜索引擎，站点里边的哪些页面可以被抓取，哪些不能。

这里边只有一句话： 以下规则针对所有的搜索引擎种类。 User-agent 用来指定搜索引擎的种类，*是一个通配符。 规则呢需要我们自己填充。详细的规则我们这里不展开，大家有需要的话自行谷歌robots.txt的语法。

通观下来， 毫无疑问，这就是被处理后的整站代码，我们现实发布上线我们的站点的时候，只需发布 dist目录下的文件即可。

下边我们就来一步步看看build到底干了哪些事情。

clean: dist clean的功能不用多说了，dist target 清空 .tmp 以及 dist目录。这里指定dot的值为true， 就说明 以.开头的隐藏文件也要一并清除。但是不清除dist目录下git相关的信息。

wiredep： 在 grunt serve一节，我们已经讲过，自动引入bower管理的第三方组件的js和css， 这里不再赘述。

接下来需要注意了， useminPrepare 和 usemin 是 同属grunt-usemin这一组件 的两个Task，相当于姐妹，usemin的主要作用就是在我们的js和css等静态资源文件合并，压缩，处理，重命名以后，对应修改我们html文件（或者更精准一点表述，引用这些静态资源的文件）里边的引用（为了表述的方便，后边我就直接说HTML文件了，大家知道这种说法包含了其他格式的引用静态资源的文件就好）， 刚才我们也已经看过dist目录下的index.html 文件了，其中对于静态资源的引用路径都被相应修改了。

那这姐妹两个是怎么分工的呢?

useminPrepare 根据我们HTML文件的注释中的指令， 来生成注释指令中包含的合并，压缩，文件重命名等配置。注意，这里只是生成相关task的配置，执行还是需要我们依赖这些相关task插件去实现，比如说合并依赖的是 grunt-contrib-concat 插件， js压缩依赖的是grunt-contrib-uglify插件，css压缩依赖的是grunt-contrib-cssmin插件，文件md5值计算和重命名依赖的是grunt-rev插件。 所谓生成的相关task的配置，其实就是生成相关Task的一个叫做 generated 的target配置。所以我们在编写的时候，可以像这样（PPT）声明，generated的target在Gruntfile.js文件中找不到， 它是useminPrepare 动态生成的。

而usemin 则负责将HTML文件中的资源引用替换成处理后的文件。

回到我们的Gruntfile.js, 我们发现 useminPrepare 和 usemin 中间包的任务， 并没有跟一个我们上边提到的generated target，为什么呢?

我们之前说过 当一个task后边没有跟任何的target的名字的时候，默认是执行这个task下的所有target， 那这种写法 generated 这个target自然是跑不掉的， 所以并没有啥问题。

那接下来，我们就来一一看看这些包含其中的task。

concurrent:dist, concurrent不用多说了， 并行执行task。

dist target 执行，

sass： 将sass文件编译，并输出css到指定的.tmp目录。

copy:styles ： 之前我们也已经说过： app目录下的css文件，拷贝到 .tmp 目录。

imagemin和svgmin: 这两个task 是我们第一次遇到， 见文知意： 压缩图片和svg文件，配置也很简单， 只需要指定源文件app目录， 处理哪些格式的文件 imagemin 处理gif, jpeg, jpg, 以及png， svgmin 处理 svg文件， 输出到dist目录。 到目前为止，我都数不清楚我说了多少次见文知意了，不是有那么句话吗： 计算机科学中最难的两件事是命名和缓存失效， 命名反映了一个程序员对领域模型的分析能力，写得一手好代码，先从命名开始。

好， 闲话少扯， 回到我们的build, concurrent 执行完了， 接下来就是autoprefixer, 帮我们把css文件中需要厂商前缀的这样的属性自动添加上厂商前缀。

concat: 全局搜索一下， 发现Gruntfile.js 文件中，并没有有效的concat 配置， 有一个还是被注释掉的， 为什么呢， 默认情况下， concat 行为是usemin动态生成的， 这里留这么一手的意思是， 如果你不想使用usemin， 直接把注释去掉， 这些配置能就着用，只能说 sindresorhus 这帮人想的还挺多。好， 借花献佛， 我们来看一下concat单独使用的语法。

因为我们现在只有一个js文件， 我们再增加两个， main2.js , main3.js, 改一下里边的内容

```
concat: {
    options: {
      separator: ';'  // separator 用来指定 被拼接的文件之间的分隔符。因为这里我们演示合并js， 为了避免合并后的js在压缩的时候出错，我们惯例都是加上一个分号。
    },
    dist: {
      src: ['<%= config.app %>/scripts/main.js','<%= config.app %>/scripts/main1.js', '<%= config.app %>/scripts/main2.js'],  src 指定要合并的文件列表
      dest: '<%= config.dist %>/scripts/concated.js'  dest 指定要合并到那个文件
    }
  },
```

看一下合成后的文件， 我们看到三个文件的内容已经被合并进了同一个文件。

回到usemin， 那usemin是如何指定js，css文件如何合并， 或者说它的配置是如何编写的呢?

我们看一眼 index.html

注意html注释内的内容， build：css —endbuild 这个就是usemin的配置， build：后边紧跟的是要处理的文件类型， 我们的页面里边有css和js， 这个值也可以通过usemin提供的方式来自定义，这里我们涉及不到。 后边的（.）是一个可选项， 可有可无，默认情况下usemin 去找这些静态资源的时候，相对路径是相对于我们被处理文件所在的路径的，这里就是index.html文件所在的路径，如果我们这里写的相对路径不是相对于index.html文件写的， 则需要通过（.) 的声明，指明从哪里开始寻找这些资源文件。 最后的参数 则用来指定，合并，压缩处理后的文件的生成路径及名字。

usemin 是怎么知道应该去哪些文件里边找这些注释配置项呢？ 这就需要我们回到 useminPrepare 的配置项看一眼。

有一个叫做html的target， 我们之前已经讲过这种语法， 直接指定src， 处理app/index.html 这个文件，options 里边的 dest 用来指定合并，压缩之后的js， css等文件，的输出根目录。

好， 我们继续回到build往下看。

cssmin， 同样的全局搜索一下， 发现Gruntfile.js 文件中，并没有有效的cssmin 配置，有一个还是被注释掉的, 道理跟concat一样，usemin帮我们自动生成了相关的cssmin配置，并在这里调用。 cssmin的语法没什么好说的, 没啥特殊的options， 就按照我们之前讲的Grunt files format配置要操作的文件目录即可。

继续往下是uglify, 依然还是， 全局搜索一下， 好， 此处略去XX个字。 我们知道uglify本身呢就是一个非常优秀的js压缩工具， 支持非常多的配置。这里我只讲一个在某些情况下可能会用到的:

sourceMap: 我们都知道， 调试压缩之后的代码， 就好像瞎子点灯一样，基本是白费蜡， 那压缩之后的文件在调试的时候怎么恢复呢， 就靠sourceMap文件。 源代码在生成压缩代码的同时生成了一份对应的sourceMap文件， 它跟我们的目标文件同名， 只不过后缀名是.map, 当我们在chrome中访问一个站点的时候， 打开控制台， chrome便会去线

上js文件的同级目录找有没有这个js对应的sourceMap文件， 一旦找到了， 就把它下载下来，将我们的js文件恢复成源代码格式， 这时候调试起来就得心应手的多了。

如果我们把uglify 的 sourceMap 设置为true，那么uglify在压缩代码的同时就会生成一份sourcemap文件， 反之则不会，这个配置项的默认值为false。

但是， 注意， 我还是建议大家不要轻易的把sourceMap文件发布到外网， 原因你懂得。

至于uglify 文件的配置没啥好说了， 依然是遵循 grunt-files-format 的规范即可。

好， 回到build， 下边的task， 我们下节接着讲。

第11节 Grunt-In-Action 常用Grunt Task剖析4

接上一节的内容。

copy: dist 看一下dist 这个target都指定了哪些拷贝操作。

首先是把app根目录下的ico, png, txt 拷贝到dist目录， 这就包括了 favicon.ico 和 robots.txt 。 然后是images目录下的webp格式的图片， 大家知道这里为啥只指定webp格式吗？ 还记得，上边（屏幕上要有演示）concurrent:dist 里边的imagemin操作吗？ imagemin在压缩gif, png, jpg 图片的同时， 已经把它们输出到了dist目录。

顺便说一下webp这种图片格式： 是谷歌新推出的影像技术，它可让网页图片有效进行压缩，同时又不影响图片格式兼容与实际清晰度，进而让整体网页下载速度加快。听起来很美好：但是注意：除了blink内核， 也就是Webkit这一支的浏览器， 即chrome， opera， Android Browser 其他浏览器都不支持这种格式， 慎用。

另外这里还有一个知识点， 之前一直没有跟大家说： 就是 {,*/}*.webp 的这种写法， 我们知道， 它其实就相当于 *.webp 和 */*.webp 诈一看好像是包含了目录下所有的webp文件，其实不然， 因为*号不能表示反斜杠，** 才能表示包括反斜杠在内的所有字符， 所以这里只往下寻路了一级目录。为什么不用**呢， 大家想想。用脚后跟也能想到是基于性能的考虑， 大部分的web站点， 两级目录就已经够深了， 所以没有必要再往深层次遍历。

将app跟目录下的html文件和一级目录下的html文件拷贝到dist目录。

将styles 中的字体文件 拷贝到 dist 目录

将apache的配置文件拷贝到dist目录， 现代web站点， 讲究前后端分离， 大部分时候我们都已经用nginx做静态web资源的服务器了，所以这步操作， 我个人认为也是没什么意义了。

将 bootstrap 用到的字体文件 拷贝到 dist目录, 注意这里cwd的配置是当前目录，就意味着src中的这一大串目录也要在dist目录对应生成， 我们看一眼是不是。（balala）

好， 继续回到build。截止到这一步，我们大部分的文件都已经进入到了dist这个目录。包括合并压缩后的js和css，图片等。

modernizr： 我们前边已经说过，modernizr 是一个用来检测用户浏览器知否支持某些HTML5和CSS3特性的JS库。传统的做法是把整个modernizr库都加载到我们的站点，但其实我们可能只用到了HTML5和CSS3的一两个特性，所以我们只需要检测这一两个特性有没有就可以了，加载一个大而全的检测库会拖慢我们站点的整体加载速度。

grunt-modernizr 通过遍历我们的项目文件， 找到我们都用Modernizr来检测哪些HTML5和CSS3特性了，然后输出一个定制版的Modernizr，从而避免了大量冗余代码的引入。

因为有些同学可能没用过modernizr， 我们这里稍微科普一下。

Modernizr执行完之后， 会在window上挂载一个 Modernizr 的对象，这个对象下有很多跟html5特性一一对应的属性，(PPT) 对应列表在Modernizr 的官网上都能找到。属性的值为true表示当前浏览器支持此属性， 属性的值为false，表示当前的浏览器不支持此属性。我们这里演示一个用的比较多的， localStorage.

打开main.js

```
;(function () {
  alert(Modernizr.localstorage ? 'support localstorage' : 'not support localstorage');
})();
```

好， 我们利用 grunt serve 执行一下

我们在控制台看一下Modernizr下挂载的所有属性。

```
;(function () {
  console.log(Modernizr);
  alert(Modernizr.localstorage ? 'support localstorage' : 'not support localstorage');
})();
```

大概有4，5十个属性， 具体等大家用到的时候再细查。

我们先把之前生成的modernizr的文件另存一下 好， 我们重新运行一下grunt build， 利用对比工具， 对比一下这两个文件的不同。（使用 kaleidoscope）

我们发现多了一段对localstorage 属性的操作语句。

好， 这时候再来看一下 modernizr 的配置项：

devFile, 指定我们本地的modernizr.js的路径，如果我们是用的线上的modernizr.js 文件，这个值就设置为remote 。

outputFile, 指定定制处理后的modernizr.js 的输出路径。

默认情况下 grunt-modernizr 会爬我们项目目录下所有的文件， 来看我们都用到了哪些 modernizr.js 的特性检测， src用来将扫描范围缩小， 指定爬哪些文件， 这里爬我们的dist 目录下的js， css， 但js里边第三方js要排除掉。

uglify：为true的时候， 指定在输出定制化的modernizr.js的时候， 顺带手把它给压缩了。

好， 回到build, 前边我们已经讲了很多利用文件的md5值重命名文件的好处了， grunt-rev 就是用来实现这个目的的。

配置非常简单， 按照 grunt-files-format 指定哪些文件需要计算一下它们的md5值，并重命名即可。这里指定，我们的dist目录下的js， css， 图片， 字体， ico， 基本上全覆盖了。 这样就有一个潜在的风险， rev只负责重命名， 可不管你这些文件名字改完之后，原来引用这些文件的地址还对不对。 所以我们才要配合着usemin使用。

usemin 一共干了两件事情， 注意， 我们这里说的usemin不包括 useminPrepare 这个步骤。

(切到index.html)第一步： 使用 useminPrepare 自动生成的一系列task，concat, cssmin, uglify ，以及rev处理后文件路径， 替换掉我们引用文件里边的 build: endbuild 块级内容。

第二步， 看一看不在build endbuild 块内的被引用文件有没有对应的重命名了的版本，如果有，用重命名了的文件路径，替换掉重命名之前的文件路径。

我们看一下usemin的配置。

它下边有两个target， html 和 css， 用来指定我们要处理其中引用的html和css文件，assetsDirs 则用来指定查看哪些被引用文件有没有对应的重命名版本， 好进行第二步操作。注意assetsDirs 中没有指定scripts路径， 这是因为script的重命名，已经在第一步替换build endbuild块级内容的时候干了。

好， 我们做个实验， 为我们的项目添加一个图片， 然后在css中使用这个图片。（桌面上calendar.png)

```
.test-usemin {
  background-image: url(../images/calendar.png);
}
```

好， 我们运行一下， 看一下这里的引用会不会被自动修改。

拉到最后， 我们发现， 这里对图片的引用也已经自动重命名了。所以说 rev 要配合着usemin用， 这就好像bower 要 配合着 grunt-wiredep 用一样， 这里引用《大秦帝国》里边的一句经典台词， 正可谓是： 公如青山，我为松柏；同心同德，永为知音。相辅相成，相得益彰，粉身碎骨，绝不相负！

htmlmin, 见名知义， 用来负责我们的html文件压缩。文件路径配置很简单， 按照grunt-files-format 指定要压缩哪些html文件即可， 配置项略多， 得唠叨两句。

PPT

html中有一类属性叫做 boolean 属性， 比如说 selected, disalbed, checked, readonly 这类属性， 属性的有无就直接决定了这个属性的值， 当这个属性存在的时候， 那么它就是true的， 不管你给它真实的赋值是啥。 collapseBooleanAttributes 为true的时候， 去掉boolean 属性的属性值。

我们在写html文件的时候， 为了代码的可读性， 我们会在适当的时候，空格，换行。(/kə'læps/)collapseWhitespace为true的时候， 将空格，换行清除掉，但不清除script, style, pre, textarea 里边的空格，以免造成副作用。 但即使如此， 依然可能存在副作用，对于 我跟你 中间带一个空格 无冤无仇 的这种写法， 浏览器渲染出来也是我跟你 中间一个空格 无冤无仇， HTML标记中的空格是被作为一个text 节点处理的， 而这个text节点的值就是一个空白符， 如果两个相邻的元素是行内元素， 那么我们看上去， 我跟你 和 无冤无仇 中间就会多一个空白符。（我跟你　无冤无仇　为何加害于我， 读起来很是有一种正气）， 但是如果把这个空格去掉的话，浏览器渲染出来就是 我跟你无冤无仇， （给人一种 我跟你什么仇，什么冤， 我跟你什么仇，什么冤的猥琐感） 表达的意思就变了。英文就更可能出这种乌龙了， May Day 断开是 五月一日的意思， 而连起来的意思则是： 遇险的飞机、船只等使用的国际无线电呼救信号。

这个时候就用到了 (/kən'sɜ˞vətɪv/)conservativeCollapse 当它为true的时候， 指定无论多少个空格压缩， 都压缩成一个空格， 而不是完全去掉。 而连续的空格 和 一个空格 是被同等对待的，别管你有多少个，在浏览器渲染中只做一个空格字符渲染。

在HTML5规范中， 并不要求所有的属性值都必须被双引号括起来， 所以把 removeAttributeQuotes设为true， 连引号都省了。

removeCommentsFromCDATA: 为true的时候， 去掉scripts和styles中的html注释语法注释掉的内容。不用太过关注， 因为我们极少在scripts 和 styles 使用 html 的注释语法。

removeEmptyAttributes: 为true的时候， 删除掉值为空的html属性。所谓的为空就包括空格和换行符。

当我们的html文件缺少某些tag或者某些tag的结束标签， 比如说head， body， 在浏览器解析的时候，它会自动帮我们把这些缺失的tag补全，页面依然可以正常渲染。 removeOptionalTags: 就是用来去掉这些拿掉对浏览器渲染也没有影响的tag。 这种做法我送它三个字: 蛇精病。请记住追求极致的路上充满了蛇精病。 我们看一眼生成出来的 index.html 文件就明白， 我们发现head的结束标签不见了， body html的结束标签都没有了。

拿input举例， 我们知道， 当不包含 type 属性的时候， 浏览器默认这个input 元素的type 值就是text， 所以这个时候 type=“text” 就是多余的。 removeRedundantAttributes 就管这事， 但是， 注意但是， 如果这个配置项设置为true的话， 就不要再使用 input[type=“text”] 的属性选择器去写CSS样式了。

useShortDoctype 使用短 doctype声明方式， 其实就是html5的doctype 声明方式， 现在已然进入html5的时代， 我们新建的页面一般都会用html5 的doctype， 所以这个配置项意义也不是特别大了。 对使用HTML4.x doctype声明的老页面有效，能减少那么几个字节，又一个追求极致路上的蛇精病。

好， 到这一步我们的dist目录就彻底准备好了， 可以发布上线了。 但是慢着， 是不是代码校验和自动化测试我们还没再确认一遍?

（运行之前别忘了把 main.js 恢复成没加 Modernizr 之前的样子）

grunt default 我们之前已经说过， 当grunt 后边不跟任何task的名字的时候， 默认就是执行组合task， default。 我们执行一下。

newer 之前我们已经说过， 只对新的，变化了的文件执行跟在后边的task。 jshint我们之前讲watch的时候一语带过了， 现在来仔细看一下它的配置项。

jshintrc: 指定我们jshint的配置文件。 jshint本身有很多的配置项， 展开的话又要说好久，大家感兴趣我们可以在留言区讨论或者开一个番外篇课程， 总之您的要求我尽量满足。我的建议是使用jshintrc之前先形成自己团队的代码规范，然后按照定下的代码规范去改造jshint。

reporter： 默认jshint的校验结果是通过Grunt reporter输出的， 通过这个配置指定其他的reporter，来让结果看起来更顺眼，更直观。 这里配置的是 sindresorhus 写的 jshint-stylish reporter .

all这个target 指定了我们要校验哪些文件是否规范。 有 Gruntfile.js， 源文件目录下的 js， 但是排除掉第三方的库和框架， 以及测试用例的js代码。

test 我们之前用了一节的时间讲这个组合task， 这里就不多说了。记住一点组合task可以直接引用组合task。

最后执行build。 这下真的可以上线了。 但上线之前最好再在我们本地的服务器上跑一遍， 看看压缩， 合并 等有没有引入新的问题。

正所谓： 出来混，总是要还的。 大家还记得在serve那一节，我们曾说等一下再看build 和 connect:dist:keepalive 都干了些什么吗？ 这一拖就拖到了今这一节。

（在命令行运行） 在运行 grunt serve的时候， 在后边跟一个 dist target, 则会先运行 build，生成我们的dist目录， 然后是 起一个本地server， 将对根路径的访问映射到 dist 目录， 供我们在本地查看部署后的效果。

好， 我们运行一下， 看到浏览器被默认打开了， 我们现在浏览的就是上线前的代码了。

这里涉及的Grunt task呢，我们终于说完了， 但是正所谓学海无涯， 授人以鱼不如授人以渔。 Grunt的task实在是太多， 我们说上三天三夜也不一定尽兴。 （打开grunt的官网）

在grunt的官网的plugins tab 已经收录了4000多个grunt插件， 带星标的是官方认证的， 支持搜索， 各种规则排序。 有特殊需求或者额外配置的时候先来这里看看， 或者直接上github， 总能找到适合你的那款菜。 如果实在没有， 我们就来自己实现一个grunt插件，(这时候搜索自己写的localhosts) 这是我之前写的一个修改本地host的插件。 而如何编写我们自己的grunt插件， 且听下回分解，谢谢大家。

第11节 Grunt-In-Action Grunt插件编写（一）

（PPT）大家好，我是mater，通过前面的学习， 我们已经彻底掌握了yeoman，bower 和 grunt 的使用， 也见识了浩如烟海的开源生态， 而这一生态依然在滚雪球似的蓬勃发展。我们能想到的正八经的task插件都已经有人贡献了， 接下来只好剑走偏锋， 我们来玩点黑魔法， 只为一窥grunt 插件编写的种种。 好， 那接下来就进入我们今天的主题， 我们来写一个“佛祖保佑， 永无BUG”的插件。

插件实现的任务也很简单， 在我们的源文件js的头部， 加上佛祖保佑的这样一个 注释画。 这画面太美， 我简直不敢看啊。 （点击切换， 加了这些注释的代码）

我一直认为学习一件事物的同时呢， 如果能够适度的延展, 在增加知识的深度的同时， 不忘扩展知识的广度 才是最终成为大家的必要条件。历史上这样的例子不胜枚举， 比如说费马大定理的证明, 一个困扰世人百年的数论问题最终被图论专家拿下。 这里我想跟大家说说佛家的四法印： 所谓：诸行无常， 诸法无我， 诸漏皆苦， 涅槃寂静， 当然这跟迷信无关， 不同的人呢有不同的感悟， 如果能从中参透人生的体会那是再好不过了， 如果不能， 权当增加码农的人文修养吗。 诸行无常， 诸法无我， 诸漏皆苦， 涅槃寂静

好， 闲篇少扯， 接下来进入我们的正题。

说到开源实战， 就一定少不了github， 我们首先在github上创建我们的插件项目。 没有账号的同学先注册账号。

我们给它起个名字， grunt-buddha (/‘budə/), 因为npmjs.org是以组件的名字作为key的，有一个唯一性校验， 所以一旦我在上边注册了， 大家就没办法再注册了， 建议大家在后边加 - 你的英文id或者你自己的唯一标识 。

Description: Buddha's grace illuminates(/ɪ’lumɪnets/) code as sunshine. 佛光普照代码

开源吗， public

是否需要github自动为我们的项目生成一个 README 说明文件， 不需要，为啥， 稍后说。

.gitignore 哪些文件不纳入git版本管理，不需要， 为啥， 稍后说。

是否添加一个代码协议文件， 不需要， 为啥， 稍后说。

好， 创建

这个时候， 把我们的项目clone到本地， 相信大家本地应该都已经安装了git， 如果没有的话， 自行google安装一下， 非常简答。

好， 进入我们的项目目录。

接下来就是我们的插件编写了， 一个从无到有的项目， 当这个念头从你脑海里闪过的时候， 你第一个想到的就应该是Yeoman， 好， 我们先上yeoman的generators收纳页看看有没有相关的生成器, 搜 gruntplugin 果然有， 还是官方出品的。 接下来就是安装这个 generator 和 利用这个generator 生成我们的项目原始代码。

（命令行操作） 大家还记得我们generator的命名规则吗？ 前边跟一个generator- ， 后边是generator的名字。

好， 生成我们的项目

插件名： grunt-buddha 别忘了在后边跟你的英文名或个人标识

描述： Buddha\’s grace illuminates(/ɪ’lumɪnets/) code as sunshine. (这里为什么带一个转义符号，等下我们就知道了)

版本： 0.0.1

这里跟大家聊一下版本一般的命名规则：主版本号.次版本号.修订号，版本号按照如下规则递增:

当你做了不兼容的API 修改时，修改主版本号， 当你做了向下兼容的功能性新增时，修改次版本号， 当你做了向下兼容的问题修正时，修改修订号。 先行版（预览版）版本号及版本编译信息可以加到”主版本号.次版本号.修订号"的后面，作为延伸。

Project git repository: git仓库的地址， 这里把我们的github地址填进去。

Project homepage: 现在还没有， 先置空， 稍后我们一起利用github的pages生成我们的项目主页。

License： 个人最爱MIT

Author name： materliu

…..

Author url： 个人主页， 这里我填我的github主页 http://materliu.github.io/

所需的node版本， 没有特殊要求，我们使用默认给出的就好。

所需的grunt版本， 没有特殊要求， 使用默认的。

好， 生成完了， 我们看一下自动生成的文件， 我们发现generator自动帮我们生成了README， .gitignore 这也是我们为什么刚刚不让github帮我们生成这些文件的原因了。除了tasks， 其他文件和文件夹的作用我们都不用多说了， 前边的课程都介绍过了。 我们的插件代码存放在tasks这个文件夹。下面我们就来一窥究竟。在编辑器打开我们的代码。

我在网上找了两个字符画， 一个是佛祖， 一个是神兽， 加入到我们的tasks文件夹， 给他加一个assets的目录。 tasks文件夹呢， 是grunt规定的plugin代码的目录结构， 必须这样命名。

alpaca （/æl’pækə/）

我们看一下gruntplugin帮我们自动生成的核心工作代码， 整个代码结构非常清晰明了，

‘use strict’ 使用ES5的严格模式

module.exports = function (grunt) {} 的形式， grunt作为函数参数传递。 我们之前就已经讲过每一个Gruntfile 和 gruntplugin 都有这样一个最外层的基本格式， 我们把它称为wrapper function。 这其实就是NodeJS编写时的一种暴露模块的语法，第三方可以使用require() 的方式来调用。

grunt.registerMultiTask 这个函数，我们之前在介绍grunt.registerTask的时候说过， grunt插件一般使用这个函数注册， 因为它允许我们为一个task配置多分属性，参数， 其实就是配置多个target的意思。

第一个参数是我们插件的名字， 在Gruntfile.js中写task配置的时候， 用到就是我们这里声明的task的名字；

第二个参数是对我们的插件的描述， 大家现在知道我们刚刚在命令行输入我们的插件描述的时候要带一个转义符号了吧， 默认生成的描述是被单引号包裹的， 如果这里没有我们的转义符号， 单引号会被我们句子中的单引号截断。

《风中奇缘》有一句经典点台词：一个人这辈子能走多远，并不是由他的腿决定，而是由他的心决定。 第三个参数就是我们插件的心脏， 决定了这个插件到底要能干什么。

我们逐行往下看：

当一个task在运行的时候， grunt会通过this object暴露很多其针对task封装的属性和方法给当前task。 this对象上的属性和方法都在Grunt的inside-task的页面有所说明， 大家记住这个页面， 留待以后需要使用我们今天没有讲到的属性方法的时候备查。

this.options 获取我们在Gruntfile.js文件中配置task时声明的options， 函数参数是我们的默认的 task 的options配置，如果Gruntfile.js 中的task配置了同名的options， 那么 this.options 返回的就是Gruntfile.js中配置的， 这里的默认options配置会被覆盖掉。 gruntplugin在这里帮我们默认填充了punctuation, 标点， separator 分隔符 这两个 options。

this.files 如果我们的grunt task 是通过 registerMultiTask 注册的， 不管Gruntfile.js 中我们是使用何种 Grunt files format 声明我们的文件配置的， 都会被转成 Files Array Format,

大家还记得Files Array Format 的声明方式吗， (在代码中演示)：

```
files: [
    {src: ['src/aa.js', 'src/aaa.js'], dest: 'dest/a.js'},
    {src: ['src/aa1.js', 'src/aaa1.js'], dest: 'dest/a1.js'},
  ],
```

this.files 拿到的就是转换之后的包含src， dest 键值对object的数组。 forEach 是ES5规范中新增的数组方法， 参数是一个function， forEach遍历我们的数组， 对于其中的每一个element调用一次参数function， 实际上， （演示）forEach会给function传递三个参数， array Element的值， array Element的index, 数组本身。 这里我们只用到了array element -》file对象，忽略其他的两个参数。

file对象， 一般我们只需要关注其下的两个key， src和dest， 不管我们Gruntfile.js配置中的src的路径值是字符串还是数组的形式， 这里都会封装成一个数组的形式， 字符串配置转换过来就是只有一个value的数组。所以 （屏幕演示）Array.isArray(file.src) === true ，而dest则是一个string的路径值。

filter 也是ES5中新增的数组方法， 用来返回我们当前操作数组的一个子级， 通过我们传递的function 参数来实现筛选，为了表述方便，我们这里称之为filter function, 如果filter function返回一个trusy的值， 那么当前的array element， 我们这里是文件路径被收入数组子级， 返回的是一个falsy的值，当前的array element则不被纳入子级。 数组调用filter方法的时候，传递给filter function的参数跟forEach中一致，不再赘述。

filter function 中 通过 grunt的api 来校验当前filepath下是否存在文件，筛选出真实有效的文件路径， 避免操作不存在的文件而报错。

这里再强调一下： 见文知意， grunt.file.exists 检验文件是否存在， 存在返回true， 不存在返回 false， 当不存在的时候， 使用grunt的.log 接口， 输出一条warn信息， 文件不存在。 grunt.log只用来输出log信息， 并不会导致grunt task执行的中断， 即使我们这里使用grunt.log.error 输出log信息， 这些只用来标识log的严重，紧急程度， 以及有一个颜色上的不同， 来方便我们 “见文知意”。

所有的 grunt api， 都可以在 http://materliu.grunt.org/api/grunt 页面找到相关的详细文档，虽然我会讲解我们所有遇到的API，依然希望大家能培养一种翻阅文档的习惯，最后对着我吼一句《一步之遥》中的经典桥段： 我不需要你，我自己能走。

拿到filter过滤之后的数组子级，执行map方法。map也是ES5中新增的数组方法，通过我们传入的function 参数来实现对我们每一个数组对象的操作， 然后返回一个包含了每一个操作之后结果的数组。为了表述方便，我们这里称之为map function。 map function的参数跟forEach中一致，不再赘述。

```
a = [1, 2, 3];

b = a.map(function(x) { return x*x; });  // b is [1, 4, 9]
```

干巴巴的说， 大家可能不太理解map到底干了件什么事， 我们在chrome的控制台里敲一小段演示代码大家就明白了。 对每一个数组元素执行map function中指定的操作，然后返回一个新的数组。

map function中， 通过grunt file api 将文件中的内容读取出来，直接将文件内容作为新的数组元素返回。

最后对返回的数组执行一个 join 操作。 其实就是将各个文件的内容拼接起来， 分隔符使用separator配置， grunt utils 是grunt api的工具类操作的命名空间，其下的 normalizelf 用来将string中的换行符转义成适配当前操作系统的换行符。 比如说windows下是 \r\n, mac和linux下都是 \n 。

最后在拼接后的文件内容的末尾添加一个 标点符号配置， 默认是句号。

使用grunt file api 将其写入指定的dest路径中， 第一个参数是写入路径， 第二个参数是写入内容。

最后输出一句log： 文件 已创建。

在《一步之遥》的影评中我写下，经典台词太多， 这里就再用一句： 太平洋的季风，已撩拨起黄浦江的热浪。 下一节就让我们来一起改造一下这里的代码，实现我们的佛祖保佑。

第11节 Grunt-In-Action Grunt插件编写（二）

这一节呢我们就来真正设计一下我们的 grunt-budha， 首先是其在Gruntfile.js 中的配置应该是怎么样的。看看 gruntplugin generator 为我们生成的Gruntfile.js中都写了些啥。

首先加载package.json 中声明的grunt插件， （点开package.json）主要是clean，jshint， nodeunit, jshint-stylish, 这里边除了nodeunit，我们之前都已经讲过， nodeunit跟我们之前讲过的mocha一样， 都属于自动化测试工具。 现在前端的自动化工具还属于群雄逐鹿的状态， 没有哪一个可以说就绝对的好和不好， mocha现在用的稍微多一些，感兴趣的同学可以在 https://github.com/joyent/node/wiki/modules#testing node的modules页面的testing大致浏览一下， 自动化测试框架真的是多如牛毛。 所以nodeunit我们也不展开， 只讲我们用到的。

task配置如下：

对 Gruntfile.js， 我们的插件代码， 自动化测试代码 进行jshint校验， 这里通过变量引用的方式， 读取nodeunit task中配置的自动化测试代码。

在生成新文件之前，先清空 tmp 目录

自动为我们生成的buddha的task配置，一共两个target， 一个使用默认配置项， 一个使用自定义的separtor和punctuation配置项，default_options target 将test/fixtures/ 目录下的testing 和123文件的内容，输出合并到 tmp下的 default_options文件。 custom_options target 功能类似

nodeunit配置， 配置很简单， 指定task的src指向测试脚本即可。

我们看一眼测试脚本：

引入grunt，exports.** node 模块暴露接口的标准写法， 这里声明buhhda 为一个object，在nodeunit 看来， 这就是一组test。

setUp, 在跑这一组测试之前先运行setUp function，nodeunit给setUp函数传递的参数，可以认为是我们等下要说的test.done，调用done函数，通知nodeunit，当前测试单元执行完毕， 进入到下一个测试单元。

default_options: 这是一个测试单元， nodetest 的对象作为参数传递进来。

test.expect 用来声明， 这个测试单元内有多少个断言要执行。 是用来确保所有的callback和断言都被执行到。 这里参数为1， 是说只有一个断言。

将我们实际生成的合并后的文件读取进来， 将测试用例里边提前写好的我们认为正确合并后的结果文件读取进来。

然后利用 test对象下的equal断言， 比较这两个文件的内容是否相等。 第三个参数字符串是对这个断言的描述。

然后调用test.done() 执行下一个测试单元。

custom_options 类似， 不再赘述。

好， 回到Gruntfile.js, 我们的grunt-budha 还没写好发布成一个npm组件，没办法利用load-grunt-tasks 自动从 package.json 文件读取，也没有办法利用我们之前讲的grunt api grunt.loadNpmTask加载， 这个时候怎么引入到Grunt中呢？ grunt 还提供了另一个api ，叫做loadTasks , 从指定目录加载task相关的文件， 一般我们用grunt加载本地的grunt插件都用这个方法， 这里的路径的相对路径是相对于Gruntfile.js 这个文件的。需要指向local grunt插件的tasks子目录， 所以我们这里只需要执行tasks， 就把 grunt-buddha 加载进来了。

注册两个组合task，

第一个， 组合task test， 清空tmp目录， 执行grunt-buddha, 然后通过nodeunit 看我们grunt-buddha的生成结果是否符合预期。

组合task default， 首先使用jshint校验我们的代码， 然后执行组合task test。

接下来我们先来运行一下default组合task。 运行成功， everything is ok 。

接下来就来设计一下我们的grunt-buddha task的配置，(清空buddha的配置内容) 因为我们是在原有文件的基础上添加佛祖保佑，或者神兽保佑的字符画， 直接覆盖原文件， 所以我们这里只需要一个src就够了， 我们声明一个dist target， 指定test fixtures目录下的js， 这时候我们在fixtures 目录下添加两个js测试文件， 填充一些内容。 删掉原来的测试文件。

buddha_grace: {

```
    dist: ['test/fixtures/*.js']
  },
```

配置项设计一个，

who， 用谁来保佑。值可以是buddha, 也可以是alpaca/ælˈpækə/,

设计一个注释符号， commentSymbol， 默认我们使用js的双斜线

```
buddha_grace: {
    options: {
      who: 'buddha',    // buddha|alpaca
      commentSymbol: '//'
    },
    dist: ['test/fixtures/*.js']
  },
```

接下来来实现我们的逻辑代码:

首先替换掉自动生成的默认配置项。使用who和commentSymbol

```
var options = this.options({
    who: 'buddha',
    commentSymbol: '//'
  });
```

使用临时变量存储两个配置项。

```
// get the who options
  var who = options.who,
  commentSymbol = options.commentSymbol;
```

接下来读取我们的字符画内容， 这里我们使用nodeJS的path模块来进行路径的操作， 使用require将其引入进来 var path = require('path');

构造字符画的路径， __dirname是nodeJS环境下的全局变量， 表示当前运行代码所在的目录， 将其拼接上assets目录下对应的字符画文件。

利用grunt api 将其内容读取出来。

```
// 读取我们的字符画内容
  var commentFilepathMap = {
    'buddha': 'assets/buddha.txt',
    'alpaca': 'assets/alpaca.txt'
  };
  var commentFilepath = path.join(__dirname, commentFilepathMap[who]),
      commentContent = grunt.file.read(commentFilepath);
```

在字符画内容前加上我们配置项中的注释符。

拿到字符画每一行的内容， var lineCommentArr = commentContent.split(grunt.utils.normalizelf('\n'));

遍历并操作每一行的内容。

```
lineCommentArr.forEach(function (value, index, arr) {
    arr[index] = commentSymbol + value;
  });
```

重写commentContent

```
commentContent = lineCommentArr.join(grunt.utils.normalizelf('\n'));
```

接下来就是进行我们的读写操作， 将不需要的自动生成的代码先删除掉， 我们不需要join和写dest的操作, 也不需要存储src变量。

在map function中读取到文件的内容之后， 将其跟注释画内容拼接

```
var originalFileContent = grunt.file.read(filepath),
        newFileContent = commentContent +
                grunt.utils.normalizelf('\n') +
                originalFileContent;
```

利用grunt api 将其写入我们原来的文件。

```
 grunt.file.write(filepath, newFileContent);
```

但是到这还没完， 这段代码存在的问题在于 我们没有校验当前文件是不是已经被添加了字符画内容了， 如果已经添加过了， 是不是我们就不应该再继续添加， 重复添加了。

怎么校验呢， 我们就检测字符画里边的特定字符好了。

观察一下两幅字符画， 提取两个字符串， buddha 我们用 /o8888888o/ ….

```
// get the who options
   var who = options.who,
      commentSymbol = options.commentSymbol;

   var testExistRegexMap = {
    'buddha': /o8888888o/,
    'alpaca': / ┗┓┓ ┏━┻┓ ┏┛ /
   };
```

如果文件中已经存在字符画的内容了， 直接return掉。

```
if (testExistRegexMap[who].test(originalFileContent)) {
        return;
```

}

好， 我们运行一下 grunt buddha.

看一眼效果， 发现佛祖已经保佑上我们的代码了。

至于nodeunit自动化测试， 也非常简单， 原理我们已经说了， 只需要修改一下读取路径和我们的expected的文件内容即可。 这里我就不演示了， 希望大家能够自行修改nodeunit的测试用例， 如果遇到问题自己解决不了， 可以在慕课网我们课程的下边的问答区提问。

这节呢， 我们就说到这里， 下一节和大家一起来看看如何把我们的插件发布到npm.js供所有人使用， 也方便以后我们直接通过npm install 安装。 谢谢大家。

第11节 Grunt-In-Action Grunt插件编写（三）

通过上一节的学习，我们成功构建了我们的grunt插件， 这一节我们就来看看如何发布我们写好的grunt插件。

进入到我们的项目目录， 通过git相关的命令， 将我们的代码提交到github仓库。

git add -A 将所有的代码加入到仓库

git commit -m “finish” 提交我们的代码

git push origin master 将我们的代码推送到github上。

好， 提交好了， 我们上github上看一眼。

在github上有一个setting 选项， 我们点进去， 里边有一个github pages选项， 可以认为是我们这个项目的主页， 既可以我们自己设计这个主页， 也可以利用github自动生成，这里图方便，我们选自动生成。

首先是我们的项目名， 以及站点标语， 都不需要改， 其次就是站点主体部分的内容了，github pages 通过jekyll来处理markdown等标记语言的文件， 将其转成html文件， 最终呈现给我们， 这里我们直接把generator为我们生成的项目README.md说明文件加载进来。 这时候发现一个问题， README.md 里边的内容，我们还用的是generator自动生成的， 这个先不管， 等下我们修改。

好， 接下来选择一个布局。

上边是一个又一个的布局， 下边是预览效果。我们选一个hack。 publish pages。

这时候，我们的项目主页就生成了， github提示说可能30分钟才能生效， 其实用不了，我们进去看一下。发现已经生成了。

记下我们的project的主页地址， 接下来我们来更新一下我们的READM.md 和 package.json 文件，

把我们的homepage 写到我们的package.json文件中， homepage： 。。。。。

更新README.md 说明文件。这里主要修改我们的配置参数说明以及示例。

好，修改好了， 再更新一下Release History

今天我们发布我们的第一个版本

2012-09-10   v0.2.0   Refactored from grunt-contrib into individual repo.

彻底更新好了， 我们再生成一遍 github pages， 首先提交 。。。。。

面子工程做完了， 这时候就该发布我们的npm 组件了。当我们通过 npm install 安装一个npm包的时候， npm便去 https://www.npmjs.com/的注册库查询其对应的代码，下载安装， 所以我们只需要将我们的组件在npmjs.org 注册一下即可。

之前没在npmjs.com 注册过的同学，需要先注册一个自己的https://www.npmjs.com/ 账号， 注册账号这种再简单不过的事情就不赘述了， 大家自行注册。

注册好了以后， 大家将账号，密码添加到本地。

npm adduser 根据提示一步步来就可以了。

在我们的项目主页上有 github帮我们默认打包了的代码的地址， 我们把这个地址拷贝一下，

在本地运行 npm publish https://github.com/materliu/grunt-buddha-grace/tarball/master 后边跟我们项目打包后的地址， 回车， 我们的npm组件就发布到 npmjs.com 上了。

非常easy。

好， 我们最终验证一下是否生效， 在我们的测试目录下， 利用webapp generator 生成生成一个空项目，

在空项目中， 安装我们的npm组件，

在Gruntfile.js 将其配置一下， 这时候运行一下。

换个配置， 在运行一下。

好， 这样我们的grunt插件就算彻底完成了， 以后有新的功能加入， 只需要对应修改package.json中版本号等一众信息，更新README.md文件， 然后重新使用npm publish 发布即可。

关于grunt的插件，我们就说这么多，大家有好的想法也可以在下边留言，我们一起来实现。 好， 谢谢大家。

第12.1节 Grunt-In-Action 扩展知识（gulp）（一）

大家好，我是mater，截止到上一节呢， 有关grunt的知识我们就算彻底介绍完了， 我们之前说过 Grunt呢，我们把它称之为Build Tool， 同为Build Tool的还有很多， 在第一节呢， 我曾说要对比介绍同为Build Tool的Ant 和 Gulp，当时之所以觉得应该介绍一下Ant呢，是因为我觉得好多前端同学是从后端岗位上转过来，比如说我， 写java的同学用惯了Ant，这样一点就通，但看了大家的评论和反馈， 感觉大家还是没接触过java的居多，所以呢，我们这里便不再介绍Ant，转而呢额外说说如何使用npm来做类似的事情。 真是正应了 《少年P的奇幻漂流》里的那句： 人生就是不断地放下，然而痛心的是，我还没来得及与你们好好告别。

好， 我们首先来说说Gulp。

第一步， 直捣它的根据地： 官网。 gulp的官网地址是： http://gulpjs.com/ 非常好记，上边有相关的文档，使用说明和插件收纳页 。 （一一点进去）

下面我们就来安装一下gulp， 安装方法非常简单， 就是一个标准的npm包的安装。

```
npm install --g gulp
```

安装完了执行一下 gulp -v 我们当前的gulp command line interface 版本是3.6.2 当前执行目录没有安装gulp。 这里的设计思想跟grunt相似， 允许不同项目使用不同的gulp版本。

学习一门技术， 最好的方法就是直接看代码，迅速投入实战中。 代码从哪来呢？ 还是那句： 一个从无到有的项目， 当这个念头从你脑海里闪过的时候， 你第一个想到的就应该是Yeoman， 好， 我们先上yeoman的generators收纳页看看有没有相关的生成器, 我们直接搜gulp， 发现第一个就是一个gulp相关的生成器，而且是官方出品， 生成一个webapp， 但是Build Tool呢， 换用gulp。 基本上，包括我们之前聊过的， 使用grunt作为build tool的generator都有一个对应的gulp版本。下边这个就是gulp-angular 对应我们之

前聊过的 angular generator(这里搜索演示)， 只不过grunt出来的早， 讨了个彩头， 它的生成器前边不需要加 grunt- XXX。

创建测试目录-安装 - 生成-打开

在真正开始之前， 我们先来说说市面上常用来说Gulp优于Grunt的抨击点， 稍后我们过的过程中， 大家注意看看Gulp有没有解决这些痛点问题：

（PPT）

1. 插件很难遵守单一责任原则。所谓单一原则无非就是一个插件只干一件事，因为 Grunt 的 API 设计思想，使得许多插件不得不负责一些和其主要任务无关的事情。经常出现“职责不明”， ”越俎代庖“这样的事情，比如说要对处理后的文件进行更名操作，我们可能使用的是 uglify 插件，也有可能使用的是 concat 插件， 这取决于我们工作流的最后一个环节是谁。 当然从另一个角度讲也不能说这就是Grunt的缺陷，因为 Grunt 的设计思想就是把对文件的操作抽象为一个独立的组件，就是之前我们多次提到的Grunt Files, 任何插件都可以以相同的规则来使用它，而遗憾在于，使用它的过程发生在每个插件的独立配置对象里，而这些插件的编写过程受限于贡献者的技能，经验不同，协调起来不尽完美。
2. 用插件做一些本来不需要插件来做的事情。因为 Grunt 提供了统一的 命令行指令入口，子任务由插件定义，由 command line Interface 命令来调用执行，因此哪怕是很简单的外部命令（比如说打开浏览器，并制定打开的页面路径 一个命令行就搞定的事情）都得有一个插件来负责封装它，然后再变成 Grunt command line interface 命令的参数来运行，多少有点多此一举。
3. 因为Grunt依赖Gruntfile.js配置文件来完成所有事，这对于开发者就提出了比较高的要求，需要对流程有一个清晰的掌控， 否则结果就是混乱不堪。尤其是对那些规模较大，构建／分发／部署流程较为复杂的项目，其 Gruntfile则会相当庞杂。而 gulp 奉行的是“写程序而不是写配置”，它走的是一种 node way。对于 node.js 开发者来说这当然是好事，符合他们的一贯作风；不过对于纯前端工程师来说，这似乎没有什么显著的改善。况且近来 Grunt 社区涌现了不少插件比如说usemin，来帮助开发者组织／管理／简化臃肿的 Gruntfile，我们看到效果也是非常的好的。
4. 鉴于以上3点，整个流程控制会产生了一些让人头痛的临时文件和文件夹， 这会导致性能上的滞后和不必要的操作步骤。这是 gulp 下刀子的重点，其标榜的“流式构建”所解决的根本问题。流式构建改变了底层的流程控制，大大提高了构建工作的效率和性能，给用户的直观感觉就是：更快。但是如果我们需要对构建流程进行debug， 这些临时文件往往对于我们发现问题大有裨益，方便我们迅速发现，定位构建流程中的问题。

所以以上4点到底能不能说是Grunt的不足，我们大家自己多思辩，还有就是整个前端集成解决方案是一套生态，单一的build tool最后能不能胜出，成功，其自身所占的比重可能只占到50%， 另一半要交给其周边生态，比如说插件库是否完备等。所以经常听到有些同学动不动说啥Grunt要死了， Gulp必胜之类的话， 你就会觉得特别可笑， 你知道吗？喊了这么多年java已死，java程序员不一样活的好好的，而且行业薪资水准不低。 大而化之也一样， 当有人告诉我们，这个世界当怎样一下就好了， 当有人告诉我们 他们都不行，让我来， 你就可以跟他说， 哪三个字？ 蛇精病。永远记住选择适合自己的方式就是最完美的方案。

好， 我们看一眼 生成的目录结构， 跟使用grunt作为build tool的项目大体结构上没什么不同， 只是gruntfile.js 对应的变成了 gulpfile.js 配置文件，整体我们就不再赘述了， 我们直奔主题， 直接看一下 gulpfile.js。

我们先来了解4个gulp的API。

gulp.task(name[, deps], fn)：注册任务

name 是任务名称；deps 是可选的数组，其中列出需要在本任务运行前要执行的任务；fn 是任务体，这是 gulp的核心了。 这个方法可以对应我们grunt里边的grunt.registerTask

gulp.src(globs[, options])：指明源文件路径，读取其数据流, 我们介绍Grunt 的时候，已经说过globs， 大家肯定不陌生了， 在计算机编程里边，尤其是在类unix环境下， 基于通配符的类型匹配我们就称之为Globbing。 这里也一样，gulp.src 支持 globs语法； 毫无疑问，对应我们grunt里边的 src 配置项。

gulp.dest(path)：指明任务处理后的目标输出路径，输出数据流。 毫无疑问，对应我们grunt里边的dest配置项。

gulp.watch(glob[, options], tasks)／gulp.watch(glob[, options, cb])：监视文件的变化并运行相应的任务。watch 居然作为核心 API 出现在了 gulp 里。 在Grunt中我们是通过一个plugin实现这个目的的。

下一节， 我将和大家一起来一个个大致的过一遍这些Task。

第12.1节 Grunt-In-Action 扩展知识（gulp）（二）

好， 接下来我们就来一个个大致的过一遍这些Task， 还是老办法， 先看头部， 然后从组合task看起。

var gulp = require('gulp'); 使用require将gulp对象引入进来。

var $ = require('gulp-load-plugins')(); 大家还记得我们学习grunt的时候有一个插件叫做load-grunt-tasks, gulp-load-plugins跟load-grunt-tasks 干的事情很像， 读取package.json文件中的dependencies等配置项， 调用gulp API 将相关的gulp插件加载进来， 返回对象是一个object， 这个object的key 值跟插件同名，指向每个插件, 这里我们把这个对象赋值给$符， 方便后续直接通过$符获取到每个插件的引用。

如果不使用gulp-load-plugins, 那么插件的引入方法跟gulp一样， 直接使用require（）后边跟插件的名字即可。（演示 require('gulp-imagemin')）

然后我们直接跳到后边， 有三个组合task， 分别对应grunt的 serve， build 和 default。

首先看serve。

还是老规矩， 实践是检验真理的唯一标准， 我们直接运行gulp watch， command line interface 设计的跟grunt很像， 前边是gulp 后边跟task的名字。 我们发现命令行提示， 本地起了一个server， 监听在了9000端口，浏览器被自动打开， 定位到localhost:9000 端口。 我们发现站点已经起来了。 同时控制台信息提示本地有一个livereload 服务器也起来了， 那我们编辑一下页面， 看是否会实时生效。 (balabala)

好， 接下来我们看一下serve task 是如何配置的。执行之前， 先执行connect, 和 watch 这两个task。 其实这里connect没有必要写， 这也算是自动生成的配置文件的一点小瑕疵， 但不影响使用。 为什么呢？ 我们直接看watch就明白了。

在运行watch之前， 先执行connect。 watch本身就依赖connect先执行， 所以即使serve task不配置connect， connect都会先执行。 这个问题我已经提交给generator-gulp-webapp 项目组， 并且已经被merge了， 下个版本就会修复， 只是不知道大家在用的时候，下个版本发布没有。

我们看一眼connect， connect在执行之前先执行style 这个task， 我们看一眼style。

style task：

style task 没有依赖于其他的前置task， 直接上来就是他的function 任务体。

将app/styles/main.scss 文件的内容转化成数据流，执行pipe（） pipe 就是 stream 模块里负责传递流数据的方法，至于最开始的 return 则是把整个任务的 stream 对象返回出去，以便任务和任务可以依次传递执行。说的通俗一点： 将多个小的变换操作进行组合，连接成管道， 这种方式就叫做流， stream。我们将数据丢入管道顶部，它将下落并穿过所有的变换，最后在底部得到我们想要的内容。流系统的灵活性能够很好地解决文件变换的需求。 但是标准流的方式也存在一个问题， 那就是当其中的一个管道运行出错的时候， 就会触发unpipe事件， 这个事件将告诉其他流不再向他写入数据。对gulp的影响，就是导致gulp报错并强制退出。这不是gulp的问题， 是node stream本身的问题。 我们现在面对的这个场景， 我们监听sass文件的变化， 一旦监听到了文件变化， 则执行sass命令， 但是 假设我写的sass文件， 本身有语法错误， 那么sass在执行的时候是不是要报错， 这就意味着rubySass这个管道运行出错， 导致的结果是gulp报错，并强制退出， 那么watch是不是失效了， 然后我得重启gulp， 之后不停的面对失败再重启的问题。

那怎么解决呢？ 这种致命的烦人问题我们肯定不能允许它的存在。所以有了，gulp-plumber (/ˈplʌmə˞/) 这个插件， plumber是水管工的意思， 这么起名字好像是我们的管道漏了需要修理一样， 其实不然，管道只是把错误抛出而已，这么做反而是管道的标准行为。 gulp-plumber 干的事情恰恰相反， 通过替换pipe方法， 和移除onerror处理函数，它把管道变成了哑巴， 有错吗， 有错， 但是不许你喊， 知道吗， 知道， 但是不许你说。 这样， 即使某一个管道某一次处理数据流出问题了， 也不会影响其他管道以及后续进来的数据流的再处理。 这只能算是一个无奈的折中方案吧。

当我们的task需要兼容我上边提到的这种情况的时候， 就需要让我们的数据流首先经过gulp-plumber

然后将数据流传给 gulp-rubySass这个管道，依然还是那句， gulp和grunt一样，它自己可不会帮你安装ruby和sass， 请自行谷歌ruby， sass的安装方法。so easy。 rubySass function里边的参数即rubySass的配置， 这里配置编译出来的css为expanded模式，sass

最终编译的css一共有4种模式： nested, expanded, compact, and compressed. sass的默认配置是nested的。 我们直接上sass的官网看一下这四种模式最终输出的文件是什么样子的： http://sass-lang.com/documentation/file.SASS_REFERENCE.html#_13

nested： 看上去层级选择器之间好像是嵌套的。

expanded 是最接近我们手动写成的css的样式。

compat： 将属性全部合并到一行。

compressed： 将所有的CSS指令合并到一行， 压缩工具常干的事。

precision: 指定我们在sass当中书写的非整数属性保留几位小数点， 这里指定保留10位。估计这里是为了把 3.1415926写进去， 所以搞了10位， 玩笑话哈， 你可别当真。 流式布局中， 我们常常要计算一个宽度的百分比， 比如说960宽度下，占宽340， 那么340/960的值就是：0.3541666666666667， 那对于宽度百分比的建议是如果小数点后的位数很大， 为了保持最高的精确度呢， 尽可能的完整保留这些小数， 反正浏览器都能识别。

rubySass管道处理完了， 把数据流传给 autoprefixer, 这个是干嘛的不用多说了吧， 帮我们把css文件中需要厂商前缀的这样的属性自动添加上厂商前缀。 插件配置一样是通过函数参数进行配置，last 1 version 指定适配 每种浏览器最新的一个版本。

autoprefixer 处理完传递给 grunt.dest 管道， 输出流到 .tmp目录下的 styles目录。

这一个task我们就发现， gulp并不是说能完全避免临时文件的产生，还是要依赖于你的流程设计， 只是它产生的临时文件更少而已。 比如说sass的执行这一步便没有临时文件产生了， 而grunt中，sass的执行也要产生临时文件。

好， 回到connect。 styles task 执行完了， 便是function， connect的任务体了。 require(‘connect’) node connect 之前我们已经讲过， 大名鼎鼎的， node的扩展Http 服务器框架， 通过使用各种中间件让http服务配置变得非常的容易。 serve-static， serve-index， connect-livereload 这些都是它的中间件，通过require引入进来。 这里就顺便讲一下node connect的语法和工作原理，之前在讲grunt的时候没有扩展这里， 我一直想找个机会说说， 趁这个机会刚好弥补这个缺憾。

node connect 通过一个又一个的中间件来处理接收到的请求，首先创建一个connect 对象， 我们习惯性的把这个对象称为 Connect App。中间件的添加就通过调用Connect App的use方法， 请求进来的时候， 会依次通过所有的中间件， 被中间件处理。 除了像 serve-static, serve-index 这种中间件， use 方法第一个参数是一个字符串路径， 这种调

用方法， 我们把它称之为 mount 中间件。 建立一个对子级url路径 寻路目录映射 的修改。

connect-livereload 中间件： 讲Grunt的时候我们就已经说过livereload的工作原理， 当时也说了connect-livereload的任务就是 自动为我们的html文件插入reload 的script脚本。

通过函数参数传递配置项。按照江湖规矩， livereload的端口监听在35729, 说道江湖规矩了， 这里想跟大家分享一个上海青帮大亨杜月笙的故事。 （balabala）

serve-static 中间件， 函数参数就是服务器要服务的文件目录， 默认会把对根路径的请求映射到这个目录， 至于子级请求路径则去寻找同名的子级文件夹来寻路和匹配， 如果某个文件找不到， 则返回一个404的response。

这里把对根路径的访问映射到 .tmp app 两个目录。 把对 /bower_components 路径的访问映射到 bower_components 目录。

serve-index 中间件， 函数参数为我们本地的一个文件目录，当路径命中这个文件目录，而路径下又没有index 文件的时候，返回一个这个目录的目录索引页面，列出这个目录下的所有文件夹和文件。 好我们看一眼。

（演示）我们app目录下有index.html文件， 所以访问根路径是不成了， 默认展示我们的index.html页面， 所以我们访问一下scripts路径， 因为前边有serveStatic 的配置了，scripts路径其实会寻路到app目录下的scripts目录，我们试一下会不会列出这个目录下的文件。 虽然列出来了， 但是上一级点了却是跳到首页了。原因不用再解释了吧。

require('http') 加载node http组件， 调用它的createServer 方法， 来创建一个新的web server object, 其参数为requestListener， 而app本身就是一个requestListener. 将app传入， 便会对request， 执行我们上述设定的一系列条件。 然后web server object 通过listen方法， 监听在9000端口上， 一旦启动成功， 输出一条log信息。 Started connect web server on http://localhost:9000

好， connect执行完了， 回到watch task， 接下来便是function， watch的任务体了。

首先调用 gulp-livereload 插件的listen方法，

gulp.watch('app/styles/**/*.scss', [‘styles']); 监听scss文件的变化， 一旦变化， 执行 styles task， 刚才我们已经说过了， 不再赘述。

gulp.watch('bower.json', [‘wiredep']); 监听bower的配置文件， bower.json, 一旦变化， 执行 wiredep， wiredep 是用来干嘛的， 我们之前讲Grunt的时候已经说过， wiredep 通过bower.json文件配置，找到我们的components依赖， 并将其引入到我们指定的HTML文件中， 这里我们看一下wiredep task的语法。

通过require 引入 wiredep， 拿到它的管道方法， 首先是处理scss文件中对bower components 资源的依赖， 然后是处理 app目录下 html文件，处理其中对bower components 资源的依赖， 通过函数参数传入配置， 排除bootstrap-sass-official引用文件, 不再将其插入html文件， 原因呢， 我们之前也讲过， 如果大家已经记不太清了， 则需要回顾一下我们之前已经讲过的内容了。（注意演示）其在html和scss中的标记语法呢， 跟grunt一节当时讲到的一模一样， 不再赘述。

styles， wiredep task 可能会导致 .tmp 目录下的css文件， app目录下的html文件的变更， 这时候监听 app下html文件， .tmp下的css文件， scripts下的js文件， 图片文件的变更， gulp watch呢， 本身返回的是一个 EventEmitter 对象，EventEmitter呢是Node.js中事件的核心对象，所有的事件基本都是通过这个对象完成构建。 通过on change的方式，我们为这个EventEmitter 对象添加一个事件监听， 当watch监听到这些文件变更的时候，便会触发 EventEmitter 对象的 emit /ɪ’mɪt/ 方法， 抛出一个change事件， 我们的监听函数被触发执行。 这其实就是最简单的订阅者设计模式。

被触发的是 livereload plugin 的 changed 方法， changed方法内部执行的时候， 便会通过事件参数， 拿到我们的变化的文件的文件路径， 通知livereload服务器， 说哪个文件发生变更了。 但是注意在这之前， 我们需要首先调用 livereload plugin 的 listen方法， 主动开启 livereload 服务器 的监听。 至于livereload的实现原理， 我们之前也聊过， 想必大家都已经很熟悉了， 不再赘述了。

好， 回到serve task， watch执行完了， 执行serve task的function主体， 引入node opn， 用来打开浏览器到指定的url地址， 没啥好说的。 好， serve就说完了。

下一节呢， 我们一起来看看build和default。

第12.1节 Grunt-In-Action 扩展知识（gulp）（三）

大家好， 我是mater， 上一节呢， 我们一起学习了gulp 的serve task， 接下来我们看一下build。

build依次依赖 jshint, html, images, fonts, extras task先执行，

看一眼jshint。 jshint不依赖于其他的task， 上来就是function主体。

对 app/scripts/目录下的所有js文件， 执行jshint校验， 不传递参数， 默认使用同级目录下的.jshintrc校验文件，校验结果传递给 jshint-stylish reporter，通过jshint插件的reporter方法，函数参数传递reporter name 来指定reporter， 关于reporter， 我们之前也已经讲过，不再赘述。 而reporter 函数如果传递的是字符串 ‘fail’参数， 在jshint校验没通过的时候， 则会通过 stylish reporter 记录下错误信息， 然后fail掉当前task。

接下来是html， 看一眼html。 html task 依赖 styles task首先执行， styles task我们上一节已经说过， 不再赘述， 用来处理我们的sass文件到.tmp 目录。 然后是html task的function 主体，

首先通过require 引入 node lazypipe 组件， lazypipe 用来缓存一组管道配置， 这组管道配置呢， 不是说立马就得被使用， 是用来存起来稍后再应用到流上。 语法呢， 首先执行 lazypipe() 函数， 然后以接近正常的添加管道的方法， 在lazypipe函数的执行结果上添加即可，为什么说是接近， 因为传递的是插件对象， 而不是插件的执行结果， 注意这后边没有跟执行括号， 这里添加两个管道，

gulp-csso 用来压缩我们的css代码， 用法和职责都比较单一。

gulp-replace 见文知意， 用来进行文本的替换， 将流中 bower_components/bootstrap-sass-official/assets/fonts/bootstrap 这段文本替换成 fonts 文本， 这是因为稍后我们看到的gulp在将bootstrap使用的字体文件拷贝到dist目录的时候，没有按照bootstrap原文件中使用的bower_components/bootstrap-sass-official/assets/fonts/bootstrap 的文件结构，而是将这些文件拷贝到了fonts这个文件夹， 所以这里要对应。

然后将这组管道配置赋值给cssChannel这个局部变量，在后续的使用中呢， 把 cssChannel 当做一个普通的pipe即可。

gulp-useref 跟 grunt-usemin 很像， 通过解析html文件中的注释块来处理替换html文件中对那些未经合并，压缩的js和css等资源的引入。 连html中的注释语法都跟grunt-usemin一模一样（演示） 就不多说了。 用法跟usemin也很像， usemin有两个task， 一个是useminPrepare, 一个是usemin， gulp-useref 也主要分两步， assets 和useref， 首先通过调用 useref 的 assets 方法生成一个用来检索资源文件的管道， 为了表述方便， 后边

我就喊它 assets 管道， 通过函数参数传入配置项，searchPath： 指定在哪些目录中搜索html文件引用到的资源文件， 文件路径相对于当前gulpfile.js所在的目录。

将app目录下的html文件纳入流中， 传给assets 管道， assets管道会检索传进来的html文件所有的引用资源，按照html文件中注释指令， 将这些引用资源进行合并， 然后生成一个新的流， 这个新的流将之前流中的html文件剔除掉了，新增了合并后的资源文件，新的流继续流转，

gulp-if 插件，很像我们管道上的小控制阀门， 只有符合条件的流， 才能进入这个阀门后边的管道， 不然只能沿原管道继续往下流。

第一个参数是控制条件， 第二个参数是阀门后边的管道。

如果，流入的流是一个js文件， 则将这个流放入uglify管道， uglify管道处理完了再回归到主管道， uglify 不用多说了， 非常优秀的js压缩组件， 如果流入的流是一个css文件， 则将这个流放入我们之前通过lazypipe缓存的管道组合， 先压缩css， 然后替换其中的fonts路径。

处理完了这些引用资源， 将流流转到assets.restore 管道， 这个管道负责将之前剔除掉的html文件重新加回到流中， 新的流继续流转， 在useref 管道中， useref 将html文件里边原来注释块内对静态资源的引用， 用处理后的js，css的文件路径替换掉。

流继续流转， 通过 gulp-if 控制， 如果流中的文件是一个html文件， 使用gulp-minifyHtml插件对html文件进行压缩， 通过函数参数填充配置项， 当conditionals配置项为true的时候， 不移除html中专门针对IE的条件注释，loose 为true的时候， 压缩空格的过程中，至少保留一个空格， 原因我们在grunt一节也已经说过了， 这两个配置项的默认值都为false。

最后将流输出到dist目录， 也就是将所有处理后的文件生成到dist目录下。

接下来是images， 看一眼images。 上来就是function 主体， 首先将app images目录下的文件读取进流中， gulp-cache, 非常好的说明了我们之前比较的一点gulp和grunt的异同，那就是 插件的单一原则， grunt-imagemin 呢， 本身维护了一份cache， 来避免对已经压缩过的文件再一次压缩， 导致图片过渡压缩失真。 而gulp-imagemin 呢， 才不管你这些， 我就只负责一件事， 压缩图片， 才不理会你这个图片之前有没有被压缩过。 那如何避免二次压缩呢， 就通过gulp-cache 这个插件， gulp-cache也只干一件事， 那就是维护一份临时文件， 记录哪些文件之前已经被处理过了， 如果处理过了， 便不再传递给其包

装的管道。 用法也很简单， 类似于装饰者模式， 其参数就是需要添加缓存文件能力的 plugin 管道， 这里我们传递的是 gulp-imagemin, imagemin不用多说就是负责我们的图片压缩， 其通过函数参数配置的两个选项值得一说， 用过photoshop的同学应该对这两个配置项不陌生，

progressive 图像渐进式扫描，这个选项针对jpg格式的文件， 逐级JPG文件可以让图像先以比较模糊的形式显示，随着图像文件数据不断从网上传过来，图像会逐渐变清晰。这样做的好处是图像可以尽快地显示出来，虽然图像不很完美，但是却让浏览者看到了希望，并且图像在不断地变好。 相比这种方式， 是jpg文件的基线或者说标准格式， 它用逐行扫描的方式显示在屏幕上 在我们看来就是图片被从上到下的一点点展示出来。

interlaced /ˌɪntəˈlest/ 图像隔行扫描， 这个选项针对gif格式的文件， 隔行GIF是指图像文件是按照隔行的方式来显示的，比如先出奇数行，再出偶数行，浏览器下载它的时候隔行下载,这样下载一张图只用一半的时间就可以看到它的样子,只不过只是隔行的图,然后它再下载另一般,这样可以减少你等待看它的时间. 图像也是逐渐变清楚。

最后将压缩完的图片文件输出到 dist下的images目录。

接下来是 fonts， 看一眼fonts。

node main-bower-files 组件通过读取我们的bower.json配置文件， 返回一个包含 我们依赖的第三方框架和库文件路径的数组， 我们可以在node环境下执行一下， 看一下返回的内容 (演示)， 我们看到有jquery， bootstrap 等等。

然后将这个数组拼接上 app/fonts/** 文件路径， 作为gulp src 的配置项， 将这些文件的内容读取进流， gulp-filter 跟 gulp-if 很像， 不同的是， 满足gulp-filter 条件的流内容继续流转， 不满足则被剔除掉。 这里利用gulp-filter 过滤 后缀为 .eot, .svg, .ttf, .woff 的文件流内容

讲grunt 动态的 src 到 dest 的文件映射， files format的时候， 我们曾介绍过一个配置项 flatten, 大家还记得这个配置项是用来干什么的吗？ gulp-flatten 插件干的事情一样， 用来将长长的相对路径移除掉。 最后将字体文件输出到dist的fonts目录， 这也对应上了前边我们说的html task中替换css文件中的 fonts路径。

接下来是 extras, 看一眼extras。 extras 干的事情就比较简单了， 将app目录下除html文件外的文件拷贝到dist目录， 但是注意这里只有一个*.*，也就意味着不会向下级文件夹寻路， 不会拷贝js， css， images 这些文件夹。之所以不需要拷贝html文件, js, css， 图片文件是因为这些文件都已经在前边task的处理中被放到了dist目录， 将node_modules 下apache-server-configs组件目录下的.htacess apache 配置文件拷贝到dist目录， 如果你

的项目不是使用apache部署， 那么这一句其实可以删掉。 gulp.src 的第一个参数是 文件路径， 第二个参数是读取文件时的配置项， 是一个可选参数， 以object的键值对的形式配置读取选项。 dot 我们在讲grunt的时候也说过， 这其实是底层 node-glob 的一个配置项， 为true的时候， 文件路径 在匹配的时候， 同样匹配以点开头的文件和文件夹。

好， 回到build， 是build的function主体。

利用gulp-size 用来将我们项目中各个文件的大小，以及项目总大小展示出来。

两个配置项， gzip： 为true的话， 展示文件被gzip压缩之后的文件大小， 为什么要展示gzip压缩之后的大小呢？ 是因为为了加快网络传输速度， 在网络上传输的网页文件在经服务器输出的时候都会有一个gzip压缩， 浏览器收到响应之后对应的gzip解压。 gzip压缩后的大小， 才是网络中传输的真实大小。 一般我们认为开启了gzip可以减少40%的流量。

title： 控制台输出的时候， 给这块内容加个标题， 方便我们阅读。 这里就叫build。好我们看一眼控制台。

build 就说完了， 我们看最后一个组合task default， 首先依赖于 clean task。

通过require引入node-del 组件来进行文件清楚， node-del 的语法也很简单， 这里作者利用了ES5里边的一个新增方法， bind，bind()方法会创建一个新函数,称为绑定函数， bind主要是用来修改某一函数的this上下文，第一个参数就是指定原来函数的this对象， 这里设置成null。 后续的参数则是传递给原来函数的真实参数。 我们稍微改造一下， 相信大家一眼就能看懂。

```
gulp.task('clean', function() {
  require('del')(['.tmp', 'dist']); node-del 的参数为要清理的文件目录。
});
```

好， 我们运行一下。 发现.tmp 和 dist 目录都被清除掉了。

然后是default的function主题， 通过gulp的start api运行 build 组合task， 这里其实可以改写成

```
gulp.task('default', ['clean', 'build']);
```

也更清晰明了， 不清楚generator 的作者这么写的目的是不是为了让我们多学习一个gulp启动task的 API。 gulp.start 参数跟 task的名字。

好，最后我们总结一下gulp的特点，大概有5点：

PPT

最后给大家几点对比学习grunt和gulp的建议：

1. 花点时间浏览一下grunt 和 gulp 的 插件库，大致了解下利用已有的插件你都可以做哪些事情 （页面操作）
2. 对于常用的插件，仔细阅读它们自己的文档，以便发挥出它们最大的功效
3. 抽时间学习grunt 和 gulp 自身的API。
4. 尝试编写适合自己工作流程和习惯的plugin，把它发布出来分享给大家， 大家一同完善， 共同进步！

好， 关于gulp我们就说这么多，至于如何在已有项目中添加gulp编译，方式方法跟在已有项目中添加Grunt编译如出一辙，我们已经讲了， 大家只要掌握了那一节， 照猫画虎也一定不会有什么问题， 我们就不在赘述了。

下一节就让我们一起来看看如何利用npm来做类似的事情。 谢谢大家。

第12.2节 Grunt-In-Action 扩展知识（npm）

大家好， 我是mater， 前边呢说完了Gulp，接下来我们来看看如何利用npm来做类似的事情。我们就着上一节生成的test-gulp-webapp项目用，不用生成新的测试项目了。 npm呢， 我们也已经见识了其是一个极其出色的工具， 可以说它是支撑nodejs社区繁荣发展的脊梁骨。但我们所见只是其作为包管理工具的一面， 其实npm还有很多子功能点， 利用这些子功能点呢还可以做很多， 比如说前边在讲 nodejs 的package.json 文件的时候呢， 我们说过 有一个 scripts directive可以直接使用 npm 来运行脚本命令。 这也就意味着可以使用这个directive来做构建工具比如说grunt， gulp所做的那些事情。

```
"scripts": {
   "opn": "opn \"http://www.imooc.com/\""
 }
```

前边我们一直使用的是npm的install能力， npm run-script的能力我们一直没咋涉及， 如果在项目package.json 所在目录运行 npm run ** npm 则会去运行package.json 中的scripts directive 对应的key的命令， 而运行方式也非常简单， 就是把 scripts directive 中的key对应的value值拿出来， 作为当前所在系统的默认shell环境下的一个command去执行， 通常呢， 这个shell环境是bash，除了windows。 比如这里我们执行 npm run opn 发现同样的，浏览器被打开， 定位到了我们指定的url地址。 而bash运行 opn \"http://www.imooc.com/\" 的时候， 毫无疑问是去PATH的环境变量里找opn的， 那理论上来说我们的PATH环境变量里不应该有opn， opn只是一个我们这里下载的npm组件而已。 这要得益于， shell帮我们把当前目录下的node_modules目录下的.bin文件夹也加到了临时PATH中。

如果我们在运行 npm run 的时候， 后边不跟任何的参数， npm则会列出我们当前的可用脚本。

我们也可以通过添加 “env” 获取当前的系统环境，

```
"scripts": {
   "opn": "opn \"http://www.imooc.com/\"",
   "env": "env"
 }
```

相信明眼的同学一定已经发现了一点问题， 那就是我们这里只能配置.bin目录下已有bash运行文件的指令或者全局Path下的指令。 那要想运行 jshint 这些怎么办呢?

我们知道gulp做jshint校验依靠的是 gulp-jshint 组件， 其实就是在node-jshint上做的二次开发封装， 这里我们安装一下node-jshint 看会发生什么。 npm install jshint

安装好了， 我们发现 .bin 目录下默认增加了 jshint的bash运行文件。 这时候我们就可以配置 jshint指令了，

```
"scripts": {
    "opn": "opn \"http://www.imooc.com/\"",
    "env": "env",
    "lint": "jshint app/scripts/**.js”
  }
```

(balabala)

举一反三， 其他的对应的grunt， gulp的task 都可以通过类似的方式构建。

这里， 我不打算带大家一步步完全走完整个npm 构建流程， 我希望这个能够作为一个终极作业留给大家， 验证一下自己通过我们这几小时的学习对于前端自动化解决方案的理解到底到了什么程度。

但是有几个知识点还是要跟大家科普一下， 避免踩坑。

npm的短语： // TODO

npm的前置，后置钩子: // TODO

npm如何传递命令行参数: // TODO

npm 配置变量: // TODO

windows 环境如何兼容： glob rm // TODO

指定多任务： // TODO

npm 如何利用管道： // TODO

最后披露scripts 完整版。

好， npm作为构建工具， 我们就说这么多， 这种做法呢， 现在看还没有被广泛应用，将来这种做法火了， 我们今天做的就是走在了业界的最前沿。 没火起来呢， 要安慰自己，开阔了眼界和思路。 好， 谢谢大家。

第13节 总结

大家好， 我是mater， 通过前边几个小时的学习， 我们终于啃下了前端自动化解决方案。首先呢， 要恭喜各位。 再有个人敢跟我们说前端不就是切个图， 写个页面吗， 前端没有成熟的自动化工具， 管理工具？ 甩他一句： 知道什么叫集成解决方案吗？ 知道什么叫现代前端项目架构吗？ 知道什么叫yeoman generator吗？知道什么叫bower吗？ 知道什么叫无知吗？呵呵， 开个玩笑。当然集成解决方案， 只是这几年前端业界飞速发展的产物之一，有太多新生的东西需要我们学习掌握， 比如说Sass和Compass， css预编译预言； 比如说AngularJS， 颠覆了我们之前的前端最佳实践；比如说mobile web，retina屏的适配；比如说hybird web app 的构建，通信方式； 比如说如何利用node快速创建git hook。 等等等等， 总之一入前端深似海， 从此周末改充电。 最后引用姜文《一步之遥》中的一句经典台词给我们的课程结个尾： 女士们，先生们，today is history, today we make history, today is part of history。 恭喜大家完成Grunt Beginner 系列课程，多谢大家听我唠叨了这么久， 谢谢大家。

记得为大家附带讲解 pre-commit 的知识。https://app.yinxiang.com/shard/s2/sh/6fe86f3b-8466-4040-aa88-91d06b2c0be4/173c7d74e91f313f