**Nodejs**

# 安装Nodejs

## Windows

安装 ：<https://nodejs.org/> 可以安装 node 和 npm

<https://nodejs.org/download/>

## CentOS

安装 ：<https://nodejs.org/> 可以安装 node 和 npm

## npm

<https://www.npmjs.com/>

## WebStrome

下载 ： WebStorm 11.0.3.rar 或 WebStorm-2017.1.tar.gz

然后用docker 见 ： <https://hub.docker.com/r/woailuoli993/jblse/>

|  |
| --- |
| Full Description jetbrains license server on docker 悄悄的干活，打枪的不要。 感谢[@ilanyus](http://idea.lanyus.com/)发布的认证服务及认证工具。认证方法 （在使用后面的的步骤搭建好服务后） http://qiniu.heyuhua.com/jblse_show.png 认证jetbrains系列ide时点选license, 并填写http://127.0.0.1:20701. 如果要提供服务给团队使用，记得把127.0.0.1 替换为 相应的外网ip。 临时认证搭建 docker run -it --rm -p 20701:20701 woailuoli993/jblse:latest version 0.2.0 增加了自定义认证用户名参数。 usage  1. docker pull woailuoli993/jblse:0.2.0  * 使用默认认证用户名 * docker run -d --name="jblse" -p 20701:20701 woailuoli993/jblse:0.2.0 * 使用自定义用户名 * docker run -d --name="jblse" -p 20701:20701 woailuoli993/jblse:0.2.0 -u coldplay   以上命令建立了一个认证用户名为coldplay的认证服务。  如果不想用这个端口， -p 后面的 第一个20701 也可以修改成别的。 |

## Web 框架

各种框架介绍：

<http://www.cnblogs.com/lhb25/p/10-best-node-js-mvc-frameworks.html>

<http://ourjs.com/detail/15%E4%B8%AA%E6%9C%80%E5%A5%BD%E7%94%A8%E7%9A%84node-js%E5%90%8E%E7%AB%AF%E6%A1%86%E6%9E%B6>

<http://www.techweb.com.cn/network/system/2015-12-11/2240069.shtml>

<http://www.csdn.net/article/2014-03-25/2818964-web-application-frameworks-for-node-js>

## Express

<https://www.npmjs.com/package/express-generator>

npm i -g express-generator

<https://github.com/expressjs/generator>

中文站点： <http://www.expressjs.com.cn/>

## AngularJS

英文网站： <https://angularjs.org/>

中文网站： <https://angular.cn/>

NgNice：　站点：<http://ngnice.com/>　NgNice 站点是由一批 Angular.js 爱好者发起的，致力于打造一个 Angular.js 的学习和经验分享平台　　<https://github.com/angular-cn/ng-nice>

## 框架选择

|  |
| --- |
| [**Node.js**](http://lib.csdn.net/base/nodejs)非常适用于Web开发，但是现在无论是一个网站，还是Web App都已经成为包括很多不同部分，如前端、[**数据库**](http://lib.csdn.net/base/mysql)、业务模块、功能模块等等的大型项目，使用Node.js从零开始进行Web开发，也许大中型团队能够胜任，但对于个人和小型团队来说是不现实的。这时候框架就成为Web开发利器，对于个人开发来说几乎是必不可少。那么如何选择Node.js Web开发框架呢？  首先，我们必须要弄清楚的是，我们需要的是—— 程序 or 框架？ 程序是已经成型的应用，你需要的是为它搭建环境、添加配置，然后就可以运行起来；框架则是应用的骨架，你需要为它添加数据模型、业务逻辑，它才能成为应用，开始提供服务。  事实上，对于Web开发来说，程序和框架的区别正越来越模糊，比如几乎妇孺皆知的Wordpress，它是一个博客程序，但它丰富的插件以及高度的自定义能够支持很大程度上的二次开发，在这点上它比起一些[**PHP**](http://lib.csdn.net/base/php)框架也并不逊色。我个人认为，如果重心在于提供服务而不是掌握技术，有WordPress这样的程序是没有必要使用框架的。  可惜的是，由于Nodejs还很年轻，目前还没有WordPress这样的程序，因此目前在Node.js开发里，如果想做出自己想要的作品，框架是必然的选择。如果是某些特定类型的应用，可以尝试一些开源的程序，比如要用Nodejs做博客，有Hexo、Ghost等。 Node.js Web框架有哪些？ Node.js里的Web框架分为API框架和Web应用框架。前者能够开发出RESTful的API，后者也能开发出RESTful API，但还包括模板、渲染等为前端所准备的功能。  API框架的使用场景是为跨平台应用提供统一的数据模型，而渲染由前端/客户端自行解决。目前比较知名的API框架有   * **restify**（[文档](http://mcavage.github.io/node-restify/)、[Github](https://github.com/mcavage/node-restify)、[NPM](https://www.npmjs.org/package/restify)） * **ActionHero.js**（[官网](http://actionherojs.com/)、[Github](https://github.com/evantahler/actionhero)、[NPM](https://www.npmjs.org/package/actionhero)） * **LoopBack**（[官网](http://strongloop.com/node-js/loopback/)、[Github](https://github.com/strongloop/loopback)、[NPM](https://www.npmjs.org/package/loopback)） * **Fortune.js**（[官网](http://fortunejs.com/)、[Github](https://github.com/daliwali/fortune)、[NPM](https://www.npmjs.org/package/fortune)） * **~~Frisby~~**（[官网](http://frisbyjs.com/)、[Github](https://github.com/vlucas/frisby)、[NPM](https://www.npmjs.org/package/frisby)）经提醒修正，这是一个用于测试RESTful API的框架，并不是API框架。   Web应用框架顾名思义，就是为了打造Web应用所开发的框架。这里有两种风格的Web应用框架。  一个是Sinatra风格，另一个是Rails风格。Sinatra和Rails都是Ruby语言的Web框架，后者的影响力更大也更为知名。这里简单的解释一下两种风格是什么意思。  Sinatra风格是指高度可配置，注重开发的自由度。代表性的Nodejs Web框架有：   * **Express**（[官网](http://expressjs.com/)、[Github](https://github.com/strongloop/express)、[NPM](http://npmjs.org/express)）TJ大神开发，Node.js官方推荐 * **hapi**（[官网](http://hapijs.com/)、[Github](http://github.com/hapijs/hapi)、[NPM](https://www.npmjs.org/package/hapi)） * **koa.js**（[官网](http://koajs.com/)、[Github](https://github.com/koajs/koa)、[NPM](https://www.npmjs.org/package/koa)） * **flaliron**（[官网](http://flatironjs.org/)、[Github](https://github.com/flatiron/flatiron)、[NPM](https://www.npmjs.org/package/flatiron)） * **total.js**（[官网](http://www.totaljs.com/)、[Github](https://github.com/totaljs/framework)、[NPM](https://npmjs.org/package/total.js)） * **locomotive**（[官网](http://locomotivejs.org/)、[Github](https://github.com/jaredhanson/locomotive)、[NPM](https://www.npmjs.org/package/locomotive)）   Rails风格则是指不重复自己和约定优于配置，以及严格遵循MVC结构开发。代表性的框架有：   * **Sails.js**（[官网](http://sailsjs.org/)、[Github](https://github.com/balderdashy/sails/)、[NPM](https://www.npmjs.org/package/sails)） * **geddy**（[官网](http://geddyjs.org/)、[Github](https://github.com/geddy/geddy)、[NPM](https://www.npmjs.org/package/geddy)） * **CompoundJS**（[官网](http://compoundjs.com/)、[Github](https://github.com/1602/compound)、[NPM](https://www.npmjs.org/package/compound)） 原railswayjs   这两种风格无所谓谁优谁劣，全凭使用者的偏好。  而在这两种Web框架之外，还有更大型的框架，即全栈框架，其中的代表是MEAN。 MEAN？ MEAN指[**MongoDB**](http://lib.csdn.net/base/mongodb)+Express+Angular.js+Node.js，这一组合包括运行环境、数据库、Web框架和前端引擎。被称为全栈框架（Full-stack framework）。这其中除了Node.js之外，每一个都是可替换的，目标是创建从前端到后端，全部使用[**JavaScript**](http://lib.csdn.net/base/javascript)的Web应用。  由于这一框架的完善性，有人将其称为LAMP的接班人。LAMP即PHP的典型运行环境，[**Linux**](http://lib.csdn.net/base/linux)+Apache+[**MySQL**](http://lib.csdn.net/base/mysql)+PHP，被大量的用于各种虚拟主机上。  MEAN看似庞大，但事实上要构建完整的现代化Web应用，特别是SPA（单页面应用），这几个组件都是难以缺少的，并且，其中每一项几乎都是目前情况下的最佳选择，因此用于学习和重头开始打造新的Web应用是非常合适的。但由于实际业务的独特性，很可能要替换其中的组件，比如用Mysql来替换MongoDB，因此，学习其中的原理和[**架构**](http://lib.csdn.net/base/architecture)，打造自己的类MEAN框架也是一种选择。  作为个人和小团队来说，全栈框架MEAN基本上足够了，但目前大多数全栈框架还包含一项特性，那就是实时，拥有实时功能的框架我们又称为实时框架。 实时框架好吗？ 实时框架（Real-time framework）指包含了webSocket的双向通信功能，能够在服务器和客户端做到实时通信的框架。  服务端和客户端自由通信的需求一直都在，但由于HTTP协议本身的局限性，因此催生了Comet等变通的方法，但即使这样也离实时相距甚远。而当Node.js兴起后，另一个[**HTML5**](http://lib.csdn.net/base/html5)技术webSocket也渐渐成熟，人们突然发现，实时通信一下子变得触手可及，于是webSocket技术在Node.js中得到大量的应用，其中最为知名的模块就是socket.io，而各种全栈框架也纷纷加入实时特性来应对更广阔的开发需求。  目前有代表性的实时框架有：   * **Meteor**（[官网](http://meteor.com/)、[Github](https://github.com/meteor/meteor)、[NPM](https://www.npmjs.org/package/meteor)） * **MEAN.io**（[官网](http://mean.io/)、[Github](https://github.com/linnovate/mean)、[NPM](https://www.npmjs.org/package/meanio)） * **Derby**（[官网](http://derbyjs.com/)、[Github](https://github.com/derbyjs/derby)、[NPM](https://www.npmjs.org/package/derby)） * **SocketStream**（[官网](http://socketstream.org/)、[Github](http://github.com/socketstream/socketstream)、[NPM](https://npmjs.org/package/socketstream)）   不过说实话，目前能看到的实时通信的应用场景其实不多，其中大多集中于聊天室、to-do、实时图表、在线游戏等领域。其他领域使用实时特性不但没必要，而且是对服务器资源的浪费。因此目前是否要采用实时框架，要看具体的项目而定。  以上基本就是Node.js Web框架的现状了，相信看到这里，对于选择何种框架读者已经心里有数了吧。最后再介绍一个容易搞混的概念，和解释一下我的选择。 YEOMAN？ 第一次见到这个词，我还以为它和MEAN有什么联系。事实上，它们是截然不同的两个东西。YEOMAN由YO（脚手架）、grunt（构建工具）、bower（包管理器），它代表的是一种工作流，与框架开发的思维方式完全不同。具体的介绍可见[这里](http://blog.fens.me/nodejs-yeoman-intro/)。  YEOMAN能够和框架达到类似的目的，都是为构建一个Web应用做好准备，但是要不要采用YEOMAN，则是见仁见智。我个人的看法是，学习YEOMAN本身就需要不少时间，并且有一定的学习门槛。至少在目前，使用框架开发还是相对经济的，而如果以后YEOMAN这种模式推广开来，再来学习也不迟，更何况有一定的Node.js项目经验之后再来学习YEOMAN要轻松很多。  事实上，我还是很认可YEOMAN这种Generator+package Manager的模式的，这是因为Node.js本身崇尚**微模块**的概念，即无论是多么小的功能，都将它们模块化，甚至大的模块也要拆分成小的模块，然后通过搭积木的方式来构建应用。这样能够彻底的解耦，对于不容易调试的Javascript来说，也有助于定位和修复应用中的问题。Generator就是这种理念催生下的产物，通过选择不同的配置和选项，将积木搭起来。不过对于这种模式目前大家也还处于实验当中，不急于进行实际应用。 为什么我选择了Hackathon Starter？ 在我的个人项目中，使用的是Hackathon Starter，一个[Node.js Web应用脚手架](http://idlelife.org/archives/491)。  我使用它的原因是，要求高度可配置，同时又讨厌写一些配置的代码，因此它对于我来说是很好的选择。一些全栈框架对我来说，封装过多，将原生的Node.js/Express API隐藏掉了，要使用还需要一定的学习成本。而Express这样的框架又太过简洁，在实际的项目中使用还需要大量的插件和配置，而这些在Hackathon Starter中都已经帮我们做好了，同时还有一些示例代码以供学习，对于新人来说非常友好，可以避免过多的挫折感。  上面一段可以看做是免费为[Hackathon Starter](https://github.com/sahat/hackathon-starter)做的广告吧，开源项目需要宣传和布道才能让更多人所关注。  最后，本文里的框架大多来源于[nodeframework](http://nodeframework.com/)网站，本文可以看做是该站的注释版，在扫清我自己的一些疑惑的同时，也希望对读者有所帮助。 |

# 包管理工具

## 工具比较

<https://www.zhihu.com/question/24414899>

## npm

<https://www.npmjs.com/>

## bower

## jamjs

## volojs

## component

## gulp

## grunt

## koa

## yoman

# 全栈

## Book【[电子书]MEAN Web Development(PACKT,2014)】



代码下载： [www.packtpub.com](http://www.packtpub.com)

## MEAN.IO

|  |
| --- |
| http://mean.io/wp-content/uploads/sites/164/2016/08/banner-top-ninja-1.png  网站：<http://mean.io/>  代码：<https://github.com/linnovate/mean>  从属于：<http://www.linnovate.net/>  学习： <http://learn.mean.io/> **<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<**  MEAN =（Mongo DB，Express，Angular 和 Node.js）    $ sudo npm install -g mean-cli   $ mean init yourNewApp |

## MEAN.JS

|  |
| --- |
| MEAN.JS  网站： <http://meanjs.org/>  代码： <https://github.com/meanjs/mean>  文档见：<http://meanjs.org/generator.html>  先安装 [yo scaffolding tool](https://github.com/yeoman/yo) 【**yo 即 Yeoman 见**<http://yeoman.io/>】  $ npm install -g yo  再安装 MEAN.JS generator  $ npm install -g generator-meanjs |

## MEAN.IO VS MEAN.JS

|  |
| --- |
| <http://stackoverflow.com/questions/23199392/difference-between-mean-js-and-mean-io>  They're essentially the same... They both use swig for templating, they both use karma and mocha for tests, passport integration, nodemon, etc.  Why so similar? Mean.js is a fork of Mean.io and both initiatives were started by [the same guy](https://github.com/amoshaviv)... Mean.io is now under the umbrella of the company Linnovate and looks like the guy (Amos Haviv) stopped his collaboration with this company and started Mean.js. You can read more about the reasons [here](http://blog.meanjs.org/post/76726660228/forking-out-of-an-open-source-conflict).  Now... main (or little) differences you can see right now are:  **SCAFFOLDING AND BOILERPLATE GENERATION**  Mean.io uses a custom cli tool named 'mean' Mean.js uses Yeoman Generators  **MODULARITY**  Mean.io uses a more self-contained node packages modularity with client and server files inside the modules. Mean.js uses modules just in the front-end (for angular), and connects them with Express. Although they were working on vertical modules as well...  **BUILD SYSTEM**  Mean.io has recently moved to gulp Mean.js uses grunt  **DEPLOYMENT**  Both have Dockerfiles in their respective repos, and  Mean.io has one-click install on [Google Compute Engine](https://cloud.google.com/solutions/mean/click-to-deploy),  while Mean.js can also be deployed with [one-click install on Digital Ocean](https://www.digitalocean.com/features/one-click-apps/mean/).  **DOCUMENTATION**  Mean.io has ok docs Mean.js has AWESOME docs  **COMMUNITY**  Mean.io has a bigger community since it was the original boilerplate Mean.js has less momentum but steady growth  On a personal level, I like more the philosophy and openness of MeanJS and more the traction and modules/packages approach of MeanIO. Both are nice, and you'll end probably modifying them, so you can't really go wrong picking one or the other. Just take them as starting point and as a learning exercise. ALTERNATIVE “MEAN” SOLUTIONS MEAN is a generic way (coined by [Valeri Karpov](http://blog.mongodb.org/post/49262866911/the-mean-stack-mongodb-expressjs-angularjs-and)) to describe a boilerplate/framework that takes "Mongo + Express + Angular + Node" as the base of the stack. You can find frameworks with this stack that use other denomination, some of them really good for RAD (Rapid Application Development) and building SPAs. Eg:   * [**Meteor**](https://www.meteor.com/). Now with official [Angular support](http://angular-meteor.com/), represents a [great MEAN stack](http://info.meteor.com/blog/thoughts-on-angular-meteor-as-a-great-mean-stack) * [StrongLoop Loopback](http://strongloop.com/mobile-application-development/loopback/) (main Node.js core contributors and [Express maintainers](http://strongloop.com/strongblog/tj-holowaychuk-sponsorship-of-express/)) * [Generator Angular Fullstack](https://github.com/DaftMonk/generator-angular-fullstack) * [Sails.js](http://www.quora.com/Node-js/Should-I-use-a-MEAN-stack-or-Angular-+-Sails-js-for-a-node-js-powered-back-end-structure-or-framework) * [Cleverstack](http://cleverstack.io/) * Deployd, etc (there are more)   You also have [Hackathon Starter](https://github.com/sahat/hackathon-starter). It doesn't have A of MEAN (it is 'MEN'), but it rocks..  Have fun! |

## MEAN on Google

|  |
| --- |
| Overview  Bitnami MEAN Stack provides a complete development environment for mongoDB and Node.js that can be deployed in one click. It includes the latest stable release of mongoDB, Express, Angular, Node.js, Git, PHP and RockMongo. |

## Mean on DigitalOcean

## 权威对比：

<http://dancancro.com/meanio_vs_meanjs.html>

|  |  |  |
| --- | --- | --- |
| MeanJS.org. provides these benefits that MEAN.io. doesn't Help:     \* Dedicated/searchable user group for questions, using github issues     \* There's a book about it File Organization:     \* Basic sourcecode organization, module(->submodule)->side     \* Module directories hold directives Code Modularization:     \* Approach to AngularJS modules, Only one module definition per file     \* Approach to AngularJS modules, Don’t alter a module other than where it is defined Model:     \* Object-relational mapping     \* Server-side validation, server-side example     \* Client side validation, using Angular 1.3 View:     \* Approach to AngularJS views, Directives start with "data-"     \* Approach to data readiness, Use ng-init Control:     \* Approach to frontend routing or state changing, URLs start with '#!'     \* Approach to frontend routing or state changing, Use query parameters to store route state Support for things:     \* Languages, LESS     \* Languages, SASS Syntax, language and coding:     \* JavaScript 5 best practices, Don't use "new" Testing:     \* Testing, using Mocha     \* End-to-end tests     \* End-to-end tests, using Protractor     \* Continuous integration (CI), using Travis Development and debugging:     \* Command line interface (CLI), using Yeoman Build:     \* Build configurations file(s)     \* Deployment automation, using Azure     \* Deployment automation, using Digital Ocean, screencast of it     \* Deployment automation, using Heroku, screencast of it Code Generation:     \* Input application profile     \* Quick install?     \* Options for making subcomponents     \* config generator     \* controller (client side) generator     \* directive generator     \* filter generator     \* route (client side) generator     \* service (client side) generator     \* test - client side     \* view or view partial generator     \* controller (server side) generator     \* model (server side) generator     \* route (server side) generator     \* test (server side) generator Implemented Functionality:     \* Account Management, Forgotten Password with Resetting     \* Chat     \* CSV processing     \* E-mail sending system     \* E-mail sending system, using Nodemailer     \* E-mail sending system, using its own e-mail implementation     \* Menus system, state-based     \* Paypal integration     \* Responsive design     \* Social connections management page Performance:     \* Creates a favicon Security:     \* Safe from IP Spoofing     \* Authorization, Access Contol List (ACL)     \* Authentication, Cookie     \* Websocket and RESTful http share security policies | MEAN.io. provides these benefits that MeanJS.org. doesn't Quality:     \* Sponsoring company Help:     \* Docs with flatdoc Code Modularization:     \* Share code between projects     \* Module manager View:     \* Approach to data readiness, Use state.resolve() Control:     \* Approach to frontend code loading, Use AMD with Require.js     \* Approach to frontend code loading, using wiredep     \* Approach to error handling, Server-side logging Client/Server Communication:     \* Centralized event handling     \* Approach to XHR calls, using $http and $q Syntax, language and coding:     \* JavaScript 5 best practices, Wrap code in an IIFE (SEAF, SIAF) Development and debugging:     \* API introspection report and testing interface, using Swagger     \* Command line interface (CLI), using Independent command line interface Build:     \* Development build, add IIFEs (SEAF, SIAF) to executable copies of code     \* Deployment automation     \* Deployment automation, using Heroku Code Generation:     \* Scaffolding undo    (mean package -d <name>)     \* FEATURE (a.k.a. module, entity) generator, Menu items added for new features Implemented Functionality:     \* Admin page for users and roles     \* Content Management System    (Use special data-bound directives in your templates. Switch to edit mode and you can edit the values right where you see them)     \* File Upload     \* i18n, localization     \* Menus system, submenus     \* Search     \* Search, actually works with backend API     \* Search, using Elastic Search     \* Styles, using Bootstrap, using UI Bootstrap AngularJS directives     \* Text (WYSIWYG) Editor     \* Text (WYSIWYG) Editor, using medium-editor Performance:     \* Instrumentation, server-side Security:     \* Serverside authenticated route restriction     \* Authentication, using Oauth, Link multiple Oauth strategies to one account     \* Authentication, JSON Web Token (JWT) | MEAN.io. and MeanJS.org. both provide these benefits Quality:     \* Version Control, using git Platforms:     \* Client-side JS Framework, using AngularJS     \* Frontend Server/ Framework, using Node.JS     \* Frontend Server/ Framework, using Node.JS, using Express     \* API Server/ Framework, using NodeJS     \* API Server/ Framework, using NodeJS, using Express Help:     \* Dedicated/searchable user group for questions     \* Dedicated/searchable user group for questions, using Google Groups     \* Dedicated/searchable user group for questions, using Facebook     \* Dedicated/searchable user group for questions, response time mostly under a day     \* Example application     \* Tutorial screencast in English     \* Tutorial screencast in English, using Youtube     \* Dedicated chatroom File Organization:     \* Basic sourcecode organization, module(->submodule)->side, with type subfolders     \* Module directories hold controllers     \* Module directories hold services     \* Module directories hold templates     \* Module directories hold unit tests     \* Separate route configuration files for each module Code Modularization:     \* Modularized Functionality     \* Approach to AngularJS modules, No global 'app' module variable     \* Approach to AngularJS modules, No global 'app' module variable without an IIFE Model:     \* Setup of persistent storage     \* Setup of persistent storage, using NoSQL db     \* Setup of persistent storage, using NoSQL db, using MongoDB View:     \* No XHR calls in controllers     \* Templates, using Angular directives     \* Approach to data readiness, prevents Flash of Unstyled/compiled Content (FOUC) Control:     \* Approach to frontend routing or state changing, example of it     \* Approach to frontend routing or state changing, State-based routing     \* Approach to frontend routing or state changing, State-based routing, using ui-router     \* Approach to frontend routing or state changing, HTML5 Mode     \* Approach to frontend code loading, using angular.bootstrap() Client/Server Communication:     \* Serve status codes only as responses     \* Accept nested, JSON parameters     \* Add timer header to requests     \* Support for signed and encrypted cookies     \* Serve URLs based on the route definitions     \* Can serve headers only     \* Approach to XHR calls, using JSON     \* Approach to XHR calls, using $resource (angular-resource) Support for things:     \* Languages, JavaScript (server side)     \* Languages, Swig Syntax, language and coding:     \* JavaScript 5 best practices, Use 'use strict' Tool Configuration/customization:     \* Separate runtime configuration profiles Testing:     \* Testing, using Jasmine     \* Testing, using Karma     \* Client-side unit tests     \* Continuous integration (CI)     \* Automated device testing, using Live Reload     \* Server-side integration & unit tests     \* Server-side integration & unit tests, using Mocha Development and debugging:     \* Command line interface (CLI) Build:     \* Build-time Dependency Management, using npm     \* Build-time Dependency Management, using bower     \* Build tool / Task runner, using Grunt     \* Build tool / Task runner, using gulp     \* Development build, script     \* Development build, reload build script file upon change     \* Development build, copy assets to build or dist or target folder     \* Development build, html page processing     \* Development build, html page processing, inject references by searching directories     \* Development build, html page processing, inject references by searching directories, injects js references     \* Development build, html page processing, inject references by searching directories, injects css references     \* Development build, LESS/SASS/etc files are linted, compiled     \* Development build, JavaScript style checking     \* Development build, JavaScript style checking, using jshint or jslint     \* Development build, run unit tests     \* Production build, script     \* Production build, concatenation (aggregation, globbing, bundling)    (If you add debug:true to your config/env/development.js the will not be  uglified)     \* Production build, minification     \* Production build, safe pre-minification, using ng-annotate     \* Production build, uglification     \* Production build, make static pages for SEO Code Generation:     \* FEATURE (a.k.a. module, entity) generator    (README.md feature css routes controller view additional menu item) Implemented Functionality:     \* 404 Page     \* 500 Page     \* Account Management     \* Account Management, register/login/logout     \* Account Management, is password manager friendly     \* Front-end CRUD     \* Full-stack CRUD     \* Full-stack CRUD, with Read     \* Full-stack CRUD, with Create, Update and Delete     \* Google Analytics     \* Menus system     \* Realtime data sync     \* Realtime data sync, using socket.io     \* Styles, using Bootstrap Performance:     \* Javascript performance thing     \* Javascript performance thing, using lodash     \* One event-loop thread handles all requests     \* Configurable response caching    (Express plugin https://www.npmjs.org/package/apicache)     \* Clustered HTTP sessions Security:     \* JavaScript obfuscation     \* https     \* Authentication, using Oauth     \* Authentication, Basic    (With Passport or others)     \* Authentication, Digest    (With Passport or others)     \* Authentication, Token    (With Passport or others) |

## Angular Full-Stack Generator

网站： <https://angular-fullstack.github.io/>

代码： <https://github.com/angular-fullstack/generator-angular-fullstack>

|  |
| --- |
| 先安装yo  $ npm install --global yo generator-angular-fullstack gulp-cli  建立项目  $ mkdir example && cd $\_  $ yo angular-fullstack  ...  启动服务  $ gulp serve  得到结果  Home page screenshot |

## 相关文章

[**Nodejs之MEAN栈开发（一）---- 路由与控制器**](http://www.cnblogs.com/stoneniqiu/p/5538109.html)

[**Nodejs之MEAN栈开发（二）----视图与模型**](http://www.cnblogs.com/stoneniqiu/p/5551019.html)

[**Nodejs之MEAN栈开发（三）---- 使用Mongoose创建模型及API**](http://www.cnblogs.com/stoneniqiu/p/5556669.html)

[**Nodejs之MEAN栈开发（四）---- form验证及图片上传**](http://www.cnblogs.com/stoneniqiu/p/5613823.html)

[**Nodejs之MEAN栈开发（五）---- Angular入门与页面改造**](http://www.cnblogs.com/stoneniqiu/p/5641721.html)

[**Nodejs之MEAN栈开发（六）---- 用Angular创建单页应用(上)**](http://www.cnblogs.com/stoneniqiu/p/5659158.html)

[**Nodejs之MEAN栈开发（七）---- 用Angular创建单页应用(下)**](http://www.cnblogs.com/stoneniqiu/p/5669419.html)

[**Nodejs之MEAN栈开发（八）---- 用户认证与会话管理详解**](http://www.cnblogs.com/stoneniqiu/p/5690731.html)

MEAN: AngularJS + NodeJS的REST API开发教程 <http://www.jdon.com/idea/nodejs/web-app-with-angularjs-and-rest-api-with-node.html>

# MEAN 开源实例

## NgNice：Angular.js 学习和经验分享平台

网站： <http://ngnice.com/>

源代码： <https://github.com/angular-cn/ng-nice>

## Nodeclub：使用 Node.js 和 MongoDB 开发的社区系统

网站： <https://cnodejs.org/>

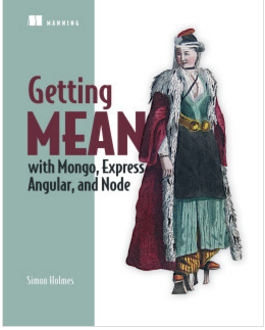
源代码：<https://github.com/cnodejs/nodeclub>

## Imprint - User Authentication with MEAN

网站： <https://boiling-harbor-43090.herokuapp.com/>

源代码：<https://github.com/lushen/imprint>

书：Getting MEAN with Mongo, Express, Angular, and Node <https://www.manning.com/books/getting-mean-with-mongo-express-angular-and-node>



## Mongodb、Express、Nodejs与Angularjs结合的MEAN架构指南系列之CRUD综合DEMO实例

网站：<http://www.upsnail.com/?p=670>

源代码：<https://github.com/lxf/MEAN>

## 一个JS全栈构建的博客应用，MEAN架构

网站： <http://114.215.164.12:3000/> 【无法访问】

源代码： <https://github.com/icyse/mean-blog>

## Blog Engine

网站：<http://zackyang.com/>

源代码：<https://github.com/TossShinHwa/CMS>

## 浙江大学求是潮网站

<http://www.qsc.zju.edu.cn/>

# 微服务

## Book【电子书】Developing Microservices with Node.js



# Mongodb

## Ubuntu 14上安装

<https://docs.mongodb.com/master/tutorial/install-mongodb-on-ubuntu/>

|  |
| --- |
| 1. 得到key   sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0C49F3730359A14518585931BC711F9BA15703C6   1. 查看（for 14）   echo "deb [ arch=amd64 ] http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list   1. 更新   sudo apt-get update   1. 安装包   sudo apt-get install -y mongodb-org   1. 启动   sudo service mongod start   1. 校验启动成功   查看文件 /var/log/mongodb/mongod.log  其中找到：[initandlisten] waiting for connections on port <port>  端口 <port> 定义在文件 /etc/mongod.conf 中，默认为27017   1. 停止与重启   sudo service mongod stop  sudo service mongod restart   1. 删除程序   sudo apt-get purge mongodb-org\*   1. 删除数据   sudo rm -r /var/log/mongodb  sudo rm -r /var/lib/mongodb |