Learn Coding With Open Source

teamhost

Published with GitBook

目錄

开放文档：《借助开源项目，学习软件开发》。

诚邀您的参与！

著作权申明

开始之前

这份文档的目标读者

至少1年以上的软件开发者人员

有如下困惑

- 希望如何写出让别人能看懂自己代码的开发者
- 如何能看懂别人的代码

软件开发初学者

这是一份面向软件开发初学者的文档，所谓初学者，可以定义为：学过的语言不超过2种，在已经学过的语言技能方面，能够完成课程上的大部分习题。从高校教育的通常情况来说，基本能够完成老师布置的最后的课程大作业。如果要说学得很出色，大概谈不上。要想进一步提高，也很困难。

对于大学毕业（正负1～2年）的同学们来说，他们很难接触到真正较为复杂的项目，即使参与到复杂的项目之中，也会是其中非常细枝末节的部分。他们渴望快速的提升自己的软件开发能力，而恰恰最缺乏提升自身能力的机会。

因此，本文希望能够介绍一种较为合理的方法，帮助各位"同学"，以较为科学合理的方式，提高软件开发的实力。

开源软件的初次使用者

在软件开发这个领域，完全不接触开源项目，几乎是不可能的事情。在日常开发工作中，如何借助开源项目提高开发效率，减少重复劳动；如何从开源而受惠，而不是因开源而受害。也是一个很有意思的课题。本文也希望能够通过分享过来人的经验，帮助那些初次接触开源的朋友。

开源社区外徘徊的爱好者

想要成为开源社区的一份子，在享受开源带来的好处之后，也能够回馈一些帮助给开源社区？开源社区到底是怎么一回事？有很多人对于开源有着天然的好感，希望能够加入进来，能不能更快的融入开源社区，能不能为开源做出更多的贡献，甚至启动一个自己的开源项目，拉起一支队伍来做些了不起的事情。有这样想法的朋友，应该也不少，希望本文的介绍能够起到一定的帮助。

由于这是一份开放式写作的文档，因此，当我写下这段话的时候，我并没有特别确定的把握。这份文档将只有以上三类目标用户。也许，它能够对更多的人产生价值.....

基本条件

一台能够上网的电脑

如果我说，以上就是全部的基本条件，似乎的确是太不负责任了。但是，真的也就够了。假设你只是研究JavaScript的开源项目的话，装一个FireFox/Chrome这样的对开发者友好的浏览器，基本上就可以开始学习了。

再进一步，当然可以有更多的考究。比如：究竟是选择Windows还是Linux/MacOS？Linux又应该选择什么发行版？等等等等，一旦深入，自然有无穷无尽的问题在等着你。

首选**Ubuntu**

首先解释为什么要选择Linux，因为开源软件，在很多情况下都是Linux的版本更加稳定可靠，在解决版本依赖问题的时候，也更加容易些。当然，在Windows下面，也有非常非常多靠谱的开源项目，因此选择Linux更多的是出于一种多多使用开源的偏好。金庸先生的《神雕侠侣》中提到的寒玉床，可以用来解释这其中的奥妙。当你完全处在一个开源的环境，当你的各种操作都会接触到与开源相关的各种概念时，你就时时刻刻都在进步中，这样获得进步自然会更快一些。

再说为什么选择Ubuntu，这就更是个人偏好了，因为Ubuntu的易用性目前看来还是最好的。对于初学者来说，也更加友好一些。网络上的中文资源，也更多一些。

另外，网络上有一篇非常著名的文章叫做：《开发人员为何应该使用 Mac OS X 兼 OS X 小史》，也写得非常有说服力，推荐阅读一下。

如果你真的喜欢Windows

必须承认，在Windows环境下，还是可以学习开源的。也有很多很多开源人，努力的在Windows平台下工作。很多环境的搭建工具，被一点一点的开发出来。比如：RubyInstaller；XAMPP；以及cygwin等等。（具体的名词这里不解释）

但是，很多时候，你会遭遇莫名其妙的报错，很多人会在某个深夜，突然抬头望天，破口大骂："这个烂Windows！"不是没有道理的。

不会被阻隔的网络

是的，这篇文章如果有幸被翻译为英文，这一段话可以被完全删除。因为他们无法想象我们还会遇到这样的困难。而克服这种困难，对于学习软件开发，又是绝对必须的一种技能，所以.....如果你真的发现目标网站无法访问，寻求帮助吧。(抱歉，无法在这份文档里提供帮助。不过，我留了email。)

Google在大多数时候，对于软件开发来说，都是更好的搜索引擎，所以，哪怕你费尽千辛万苦，也一定要用Google来搜索想要寻找的项目、文档和资料，必须的！

你需要明确的一些事情

你真的想学习软件开发吗？

在正式开始学习之前，我实在是忍不住，我想一遍又一遍的询问你：你真的想吗？你真的想成为一个"码农"吗？你真的想掌握软件开发这门手艺，甚至以此来谋生吗？那么好，我得告诉你一些事实：

- 软件开发绝非你想象中的事少钱多责任轻的那种高薪白领。很多程序员会自称"码农"，就是因为这份职业非常的辛苦，而且做好不易。
- 另外，这是一份需要终生学习的行当，很多很多的其他领域，没有那么快的知识更新速度。但是软件开发这个领域，1～2年不接触技术最新的进展，你就OUT了。
- 还有，软件开发这个行当，真的未必那么好找工作。就业前景什么的，并非想那些传说的成功故事一样光明。

但是，真的有一些人，热爱这个行业，编程不但是他的工作，他的业余爱好也是Coding。

如果，你确信自己不仅仅是靠编程来维生，更是将编程作为自己最大的爱好，那么欢迎您，来到一个神奇而充满魅力的世界。这里有智慧、有乐趣、更有热心的朋友和充满前途的事业！

你真的适合软件开发吗？

虽然，软件开发并非像传说中的那样，是一个需要高智商的Nerd的行业。但是，他的确需要一些品质和能力，如果你发现自己并不具备，或者要很辛苦才能做到。那么，你不适合这个行当。

- 懒惰：有一句名言这么说："懒惰是程序员的美德"，因为真正的程序员，一定痛恨反复做同一件事情，至少他们会写一个函数来替自己完成。如果发现代码里重复出现相同的段落，他们会无法抑制的想要消除这种重复。
- 条理：如果这是一个复杂的事情，那么我可以分成三个阶段来着手去做它。如果仔细想想，第一个阶段，还可以分为5个部分。在开始第一阶段之前，还有4个准备工作，必须首先考虑。
- 耐心：很多时候，程序里的麻烦会来找你，如何解决？解决"bug"需要洞察力，需要细心，而最需要的，则是耐心。有些时候，我会非常享受这种"破案"的过程。
- 好奇心：值得好奇的事情太多了，永远学不完的新技术；最近的进展和最佳实践。甚至其他行业和领域的究竟，我们都有充沛的好奇心，因为无论哪个行业，他们早晚都会来找到我们，帮他们编写代码的。
- 较真：很多事情，据说都差不多就可以了。但是，计算机是那么严格，快百分之一毫秒，也是快了。一个万分之一概率下会出的bug，还是bug。如果你不是一个足够较真的人，就会放过很多问题，而那些问题，往往就会酿成大祸。

OK，闲话少叙，咱们开始吧！

下一章

开始

学习软件开发的几条主要途径

一万小时的神话

前一阵子我在网上与人讨论一个一万小时的话题。有一本叫做《异类》的书中说到这样的观点：世上本没有绝对的天才。天才也需要超过1万小时的训练。人人都有可能成为顶级高手。具体可以参考一篇流传甚广的文章《怎样练习一万小时》

在这种理论的基础上，有人提出了“一万小时编程训练”的概念，似乎经过一万小时以上的训练，普通人也有可能成为编程领域的“大师/高手”。但是，这个理论其实是一个貌似建立在统计学基础上的伪理论。

首先是范围，多大的一个范围，算是一个领域呢？编程是一个大的范围，编译器或者数据库编程是其中两个不同的领域。如果有人用一万小时专注于编译器的开发，他们对于数据库方面的编程，可能是一窍不通的。那么，这种人，算不算编程领域的大师呢？如果有人用一万小时专注于汇编语言的编程，那么对于新出现的面向对象的脚本语言，能够有多高的水平呢？

其次是训练，在一万小时的相关统计中，提到了舞蹈、音乐等众多需要反复练习的领域，而这种针对熟练程度的训练，真的是编程领域需要的吗？当然，我相信提高打字速度，的确有助于提高代码编写的速度，但是这真的有助于提高编程的能力吗？要想在软件开发这个领域，讨论如何才算是训练了一万小时的编程，会非常困难。

最后是关于知识更新，在一个每天都在诞生新名词，新技术，新思路的领域，一个曾经埋头苦练一万小时的高手，在3年不接触最新知识以后，还能称之为高手吗？

总之，匆匆忙忙的接受了一万小时的概念，被激励得热血沸腾，打算下定决心奋力学习一万小时编程，通常不过是“励志书中毒”的症状而已。

软件开发的能力体系是怎样的？

在很多领域，我们都可以用一个金字塔模型，来描述该领域的能力体系，在软件开发领域，同样如此。

- 高
  - 创造能力
- 中
  - 逻辑能力
  - 理解能力
- 低
  - 基础知识
  - 编程技能
  - 领域知识

简单解释一下：

低的三项，属于知识类。基础知识包括计算机、数学、算法、逻辑等等知识，这些知识，通过认真的学习书本教材，基本能够掌握。

编程技能，往往是跟具体的语言相关的，当然，多学几门不同的语言，对于快速掌握一门新的语言，大有帮助。

领域知识，则是与工作的具体方向有关，比如针对多媒体领域的编程，自然要熟悉图形、图形、声音等等的相关领域知识。针对企业级应用的开发，对于管理制度、财务、成本、仓储的东西，总得搞清楚才行。

中与高的两项，属于超越编程局限的通用能力，不仅仅是软件开发上用得到，在各方面都非常需要这三类能力。逻辑能力，可以通过训练提高；理解能力，可以通过经验积累；而创造能力，的确比较难，有天赋的成分在其中。

有哪些途径，可以锻炼这些能力？

- 阅读与习题：找到一堆的经典教科书，狠狠的读，认真的把书里的习题都给做了，这样对于打下扎实的基础，将会有极大的帮助。
- 视频教程/ScreenCast：每次讲解一个主题，学习一下总会有收获，只是效率不高。
  - MOOC课程的出现，正在改变这样的现状，不过国内的MOOC课程大多是渣（简单的把视频教程搬上来就称为MOOC）
  - 我后面会推荐一些较好的MOOC课程，基本都是E文。
- PPT/Slide/PDF：这种属于某次技术会议上的演讲稿，如果能够配合视频看，效果还好些，否则通常会不知所云。
- Wiki：针对某个词条，某个特定的问题，会有相当清晰的解释，不过要看运气，有些词条的解释就非常粗略，甚至过时。
- Blog：在分享知识与经验的过程中，blog是很不错的载体，如果你能够找到的话。
- BBS：曾经是最主要的学习方式，很多人通过泡论坛来提高自己，不过说实话，效率很低，而且容易跑题。
- 邮件组：的确存在着不错的一些邮件组，不过不好找，欢迎多多推荐。
- 问答社区（StackOverFlow/Quora/知乎）：新兴的交流社区，在面临特定问题时，可以尝试搜索或提问。平时泡泡，努力回答别人的问题，也有助于自己的提高。
- 工作中的项目：当然，老板给你发工资，肯定希望你尽快完成，在压力之下，通常进步都会很快。只是这种进步也许是你无法选择的。
- QQ群：真的有人借助QQ群来学习吗？
- 开源项目/开源社区：当然，这个是最重要的，咱们下节详细说。

为什么要学习

学习有不同的目的：有人学习是因为兴趣或者好奇；有人是为了增加生存的技术，把学作为改变工作、生活状态的手段；当然，也有些人，学习是为了思想的交流，与周围的人交流，与远方的朋友交流，与过世的先哲交流。

静下心来，仔细想想自己为什么要学习很重要。如果学习的目的不明确，学习就缺少源动力。这种思考在学习之初是需要的，在学习过程中也同样是需要的。因为随着学习的进行，个人对学习的态度、感受也会发生变化，学习的目的也需要及时的调整。

一个善于学习的人，是能充分利用各种学习机会进行学习实践的人。有人七十多岁开始学画油画，也有人利用每天坐地铁的时间学会一门外语，甚至还有些人把微博、网络公开课作为学习的重要工具。只要学习目的明确了，学习就变成了一件有意义的事，因而才可能持久。

生物进化的几百万年，才使人类有了学习的能力，这种能力是区别与一般动物的。人类的学习是一个觉醒的过程，近百年人类文明高速演进，特别是互联网的出现，使学习的从原始的环境适应演进为主动的创造并迅速转为社会向上的推动力，或者破坏力。

为什么借助开源学习是最有效的

知识是多样的，学习的目的不同，学的内容也不同。计算机的出现，使学的工具发生了根本的变化，而程序的灵魂，因此学习编程不仅仅是软件工程师的事，它应该成为每个社会成员的一个基本的技能。正如语言是人类交流的基本技术，编程是人与机器交流的基本技能。学会编程，可以使机器按照你的意志运行，使每个人按自己的兴趣整合信息资源，以利于更有效的学习。

软件是近百年发展最快的技术之一，特别是随着智能手机与平板电脑的普及，软件技术更是渗透到了我们生活的各个方面。学会编程，并不意味着要去要去建一个复杂的系统。其实写的个报表的计算公式或者做一个小动画可以是一种编程的体验。

软件编程需要的基本环境就是一台电脑，当然如果有互联网的接入则更利于交流与技术信息的查询。

使用Linux最大的好处是它本身就是一个软件开发的开放平台，你可以方便地下载各种开发工具，比如gnu c/c++，python或者其它。你应该学会使用apt-get，这是一个Ubuntu下强大的软件包管理工具。

在网站kernel.org上，有各种版本的内核源代码，如果你想从根本上学习操作系统，也可以通过LFS快速地学习内核构建的过程。

源代码开放的最大的好处是我们不需要重复设计和制造轮子。每个人都可以在软件巨人的臂膀上构造自己的梦想代码天堂。

无论从美国的facebook、谷歌、苹果还是从中国的华为成功的经验中我们都可以看到，开源的代码以及开源的项目是当前众多商业公司的技术立足之根本。开源已经造就了无数商业神话。我们大部分人只知道苹果的酷，但很少有人在苹果的版权说明中，看有关开源项目的罗列。

微软的比尔盖茨以及苹果的乔布斯大家耳熟能详，但对软件产业最有影响力的人应该是出生于芬兰的李纽斯(Linus Torvalds)。谷歌正是采用Linux为内核，才使android几乎在一夜之间窜红并重创诺基亚。而李纽斯在软件界的影响力，堪比罗马教皇。

有一部电影叫源代码，也许对代码开源化的一个隐喻。如果你读到了关键的源代码，也许你真的可以改写历史。当然，李纽斯说得很好，开源应该是快乐的，“Just for fun”。我们不需要太多的使命感与焦虑，改变世界也许只是一个顺带的结果。

源代码是我们最好的营养。

方向不对，努力白费

在中国，如果你想面朝大海，应该是一路向东。当然向西也是可以的，不过要多费些周折。技术的更新非常快，但如果把握了大势往往可以事半功倍。

举一个实际的例子：十多年前，PHP是一个相对冷门的编程工具。在很多场合，很多人都不好意思说自己是搞PHP开发的。而如今，PHP已经成为主流的开发工具，很多搞.net的人出于生计的考虑，不得不转向Java或者PHP。

> 这个例子不妥，语言只是载体，工具，任何语言都能达成自己的目的。 以前父母总是以这样的方式去教育孩子，这个专业（语言）很热门，薪水很高...

选择开发工具只是软件工匠们需要认真定夺的一个方面。其它如系统构架、测试方法、团队管理、决策者眼光等等，更是关系每个程序员未来的诸多要素。

开源的精神内涵使学习变得更加有意义

在商业极度发展的今天，人们对物质的无限追求使很多人忘记了生活的本质。人被异化为物的附属品，价值被虚拟的概念、标签重置。

互联网的出现，促进了人类相互之间的沟通。软件高速更新发展的自然需求和因团队协作所带来的有效性、高效性造就了一个全新的文化：开源文化。软件便与分享、开源代码便于扩展的特质，使以Linux操作系统为代表的开源项目迅速崛起。大批的软件工程师不仅通过开源项目找到的精神寄托、同道中人，而且还找到了与商业社会有效融合的模式与渠道，解决了事业与兴趣结合问题，实现了生活、学习、工作甚至社会公益有完美统一。

在开源精神的感召下，学习变得更加积极主动。

在分享、贡献的核心价值体系下，人们能充分体会人心温情的另一面。与传统商业社会利用信息不对称在交易中图谋利益最大化不同，开源世界里的人们在创造、协作的过程中完成一个又一个不断成长的软件系统，这些系统有些使整个社会运行更有效、当然也有的在损坏甚至危及社会的安全。人类精神世界的两面性在开源世界里更直接、更激烈地表现出来，正在影响着现实的诸多方面。

开源社区是最好的学校

软件是构建虚拟世界的基础，而开源社区则是软件新技术产生、发展的主要场所，因而也是学习软件技术最好的学校。

当前最大的网络社区应该算游戏社区，这个社区的人大多是在消费社会资源。而开源社区则分化成两个阵营：一个是以创新、创造为目地的，创造社会价值；另一个则是以破坏、非法取得信息资源为目地，损毁社会资源与体系。

在开源社区里，有大量热心的程序员他们乐于分享自己对技术的理解、心得，他们通过各自的行动扩大自己的影响力，在协助别人的同时不断加深自己对技术的理解程度和实践能力。而新的社区加入者也可以在与社区互动的过程中找到自己技术与精社的导师(Mentor)，正如电影黑客帝国(Matrix)中尼欧(Neo)遇到摩菲(Morpheus)。

如果说我们生活的世界是上帝创造的，那么我们对面的这个数字的世界则是由程序员创造的。数字世界与现实世界不断地融合，使现实世界与虚拟世界的边界变得越来越模糊。在学校课堂里，陈旧的教学方法、过时的教学内容、有限的学习资源是无法与互联网上丰富的开源社区资源相比的。社区内部团队协作的自发性、自主性、可靠性也极大地提高了社区成员学习的效率，并使个体超常规成长成为可能。

在媒体上经常看到十、三四岁的少年创造一个个软件项目的奇迹，殊不知这与国外成熟的开源社区发展息息相关。如果国内开源社区渐渐发展起来了，我们有理由相信在不久的将来，我们的周围会出现众多皮尔斯·富里曼(Pierce Freeman)这样的天才少年。

移动互联网时代，学习是开放的更是开源的

随着智能手机、平板电脑的普及，学校以及教室的功能将被弱化，人们可以在各种公共场所组成形式多样的学习社区。而开源社区提供多种专业技术人员以及业余爱好者面对面交流的机会。有的地方还出现了包括软件、硬件开源的创客空间。大家在无线网络环境下快速组成学习社区，分享交流最新的技术，互相协助解决各种技术问题。发现的志同道合的朋友，有的技术团队在天使投资者的支持下，在学习的过程中还可以建立创业团队。

在企业的内部，根据企业的发展战略，也可以形成企业内部的开源社区，通过开源项目整合企业内部与外部的技术资源。开放的心态使企业以开源文化的发展为契机引领技术的潮流。

在学习一门语言之前

你至少应该对计算机科学有所了解，如果你是个文人骚客，淫得一首好诗，但是对计算机一窍不通。 那么你应该首先去认知计算机科学的各种概念。这里推荐一个不错的计算机科学入门MOOC课程: 斯坦福大学 计算机科学入门课程 它面向无任何基础的学生，介绍计算机科学的基本知识:

- 计算机和代码的本质，它们能做哪些事情，不能做哪些事情

- 计算机硬件如何运作：芯片、CPU、内存和存储设备
- 必备术语：bits（二进制位）、bytes（字节）、megabytes（百万字节）、gigabytes（千兆字节）
- 软件如何运作：什么是程序，什么叫做“运行”
- 数码图片是如何实现的
- 计算机代码：循环和逻辑
- 大概念：抽象、逻辑和程序缺陷
- 结构化数据是如何实现的
- 互联网是如何实现的：ip 地址、路由、以太网、wi-fi
- 计算机安全：病毒、木马和密码，噢我的天啊！
- 模拟和数字
- 数字媒体、图片、声音、视频、压缩

在了解了计算机科学之后，你就可以开始选择一门语言来进行学习。

选择一门语言

首先需要说明，这里所讨论的选择语言，并非工作中开发语言的选择，而是出于学习软件开发，提高软件开发能力的目的，讨论如何选择语言。

一些基本的判断依据

- 最好是跨平台/平台无关的语言。比如Java、Ruby、Python、PHP、JS这样的语言。.NET平台的众多语言，因为mono的存在，现在也算是跨平台的了。
- 这门语言所创建的开源项目，要足够多，可供选择。C/C++、Java、Python、PHP都 可以非常好的满足这个条件，随着github平台的出现，Ruby的开源项目现在也越来越 多的了。随着nodejs的兴起，JS的开源项目也呈现明显的上升趋势。
- 语言以及语法本身，要具备较好的可读性。这里我非常推崇Ruby，因为这门语言从创立之初，就是极端重视代码可读性的，整个Ruby社区的风格，也非常强调代码的简洁优雅。
- 相关的文档资料容易查找，这方面大多数流行语言都符合条件。C/C++、Java、PHP、C#、 Python都已经极为丰富了。
- 最好身边有能够随时请教的这门语言的高手。有这一条，基本就足够了。
- 最好能够至少分别学习一门静态类型语言与一门动态类型语言。

推荐一些语言学习网站

- 《笨办法学语言》系列，详见下节的介绍。
- 指导开发者快速学习编程的网站推荐
- 当然，各个语言的官方网站，是必须常去的。

一些忠告

- 不要根据流行的编程语言排行榜，选择语言。
- 不要根据某某语言最容易找工作，薪酬水平最高，来选择语言。
- 不需要贪图掌握太多的语言，越是深入的学习一门语言A， 越是能够快速的学习另一门语言B。A和B可以是任何两种语言。 努力深入透彻的掌门目前正在使用的这门语言，深入、再深入。 这些努力，在你以后要学习其他语言的时候，一定会有回报的。

一点建议

- 选择深入的语言,其实就是选择一种生活方式
- 各种开发语言都有最适合的工作领域
- 各种开发语言的技术社区在中国各自有各自的文化
- 选择日常可以用到的开发语言深入进去，事半功倍 ;-)
- 选择hacker 界公认的好语言，坚持学习，总是能得到养份的,获得多少,看能坚持多久！
- 这里公认的是指 Lisp —— Paul Graham在The root of LISP 里曾经曰过：

> “吾观之，自古语言模型者，惟 C 与 LISP 简洁而隽永，高山仰止， 后来者皆取 C 之形，循 LISP 之神也。”

推荐语言

这个列表，可以不断扩充，也欢迎大家补充自己的推荐语言与推荐理由

首选语言:

如果以常规的应用开发为目的来学习一门语言，那么面向对象类型的语言(java,freePascal,.net)能够让你养成严谨的基于对象分类思索模式的开发习惯，这会是一个好的开始。 如果是为了快速开发建站，那么php,ruby, nodejs,python不失为一个选择。 不过可能到了后面需要补坑。

语言名称	主要开源项目	推荐理由
ruby	Ruby On Rails	动态语言，简洁清新
java	Tomcat	经典的面向对象静态语言，长盛不衰，优秀项目多如牛毛
python	scipy, nltk, django, ansi	最接近人类语言的通用语言，在开源的科学计算领域一骑绝尘，在数据挖掘、web开发、系统管理等领域也为表现突出
c	lua,mongrel2	面向过程的，存在于所有平台，历史悠久的语言。如果你的未来应用方向是嵌入式开发的话这是首选
nodejs	sails.js,angular,coffee	疯狂的科学家们在用js做一系列的不可思议的事情，有一统的趋势

对比语言元素: PHP, Perl, Python, Ruby

其实选用哪一个语言并不重要，关键是你要能使用该语言去控制，去驱使计算机。

必须初步掌握的基本功

以下所讨论的基本功，其实是一个相当宽泛的概念。很难确切的定义一个门槛：不到某种程度，你就无法学习开源了。而是说，在掌握了一些必要的能力之后，再开始学习，会学得不那么幸苦。

计算机基础知识

计算机相关的基础知识，其实相当琐碎，很多人都是在日常的使用与开发过程中，逐步掌握的。在了解各种各样的零零碎碎的知识同时，对于各种知识及其相互关联，有一个整体上的把握，我称之为“地图思维”，是非常重要的。

简单来说，计算机相关的基础知识主要包括：基本操作与使用；计算机体系结构；网络基础知识；算法导论等等。

- 计算机操作与使用：
  - 会初步使用至少一种操作系统吧；
  - 会自己装一个虚拟机（VirtualBox什么的都可以）；
  - 知道常见的文件格式如何打开；
  - 知道去哪里下载并安装相关的应用软件；
  - 诸如此类的知识。推荐一个网站：我爱电脑网真心觉得不错，那些乱七八糟的广告可以略过。
- 计算机体系结构(组成原理)：
  - 《深入理解计算机系统》 书评:《NB学校的NB课程的NB教材--CSAPP》

- 网络基础知识：
  - 《计算机网络》，
    - 书评:《我看过的最好的网络入门书》
- 算法与设计：
  - 《Algorithms》
    - 书评:《大家好，我是译者》
  - 《设计模式 可复用面向对象软件的基础》
  - 《代码大全》
  - 《程序设计实践》
- 数学基础:
  - 《具体数学-计算机科学基础》
- 语言:
  - java编程思想
  - Google Java 编程规范
  - C编程一站式学习

至少掌握一门编程语言

听上去似乎是废话，如果连语言都没有掌握，怎么可能开始学习开源软件，看人家的源代码呢？不过，怎么才算掌握了一门语言呢？能够写出Hello World，自然是不算的。掌握这门语言的基本语法，肯定也是不够的。也许找一本某某语言的经典教材，然后把后面的习题都给做出来，的确算是一个简单的办法。不过，编程语言实在太多，相关的经典教材，就更是多不胜数。这里就不再一一推荐了。

不过特别想推荐一个《笨办法学语言》系列，目前有：Python、Ruby、C、Regex、SQL、CLI六种。引用我觉得最有道理的一段话：“不要复制粘贴。你必须手动将每个练习打出来。复制粘贴会让这些练习变得毫无意义。这些习题的目的是训练你的双手和大脑思维，让你有能力读代码、写代码、观察代码。如果你复制粘贴的话，那你就是在欺骗自己，而且这些练习的效果也将大打折扣。”相关链接如下：

- Learn code the hard way
- 《笨办法学 Python 》在线中文版
- Python初学者（零基础学习Python、Python入门）书籍、视频、资料、社区推荐

熟练掌握搜索引擎的使用

- 第一戒律：尽可能在Google，而不是Baidu搜索。对于软件开发而言，Google才是最佳武器。
- 不断的积累关键字：一个内容你搜索不到，只是因为你没有听说过那个关键字。比如，我想要找一个图像处理的开源项目，如果你知道 “Computer Vision”是指计算机视觉，那么直接搜“Open Source Computer Vision”,OpenCV一定就会是第一个结果。如果你知道

OpenCV，那么想要找一个2D图像转换成3D图像的技术，有没有开源实现，就可以试着搜“2D to 3D OpenCV”，也许就会更快的找到想要的内容。当你对某个领域完全没有了解时，可以先试着搜索一些周边词汇，看看相关的文档，然后了解行内人是用哪些关键词的，然后再去搜索，就会迅速的缩小范围。

- 搜索出错信息：当然，当你遇到错误时，直接把错误输出放到Google里去搜索，也说不定就会遇到和你有相同遭遇的同学，看看别人是怎么解决的。
- 尝试各种专业的、垂直的搜索引擎：比如StackOverFlow或者Quora这样的专业问答社区，koders 则是一个源代码搜索的引擎。google search里的Blog、Discussions里也有不少好东西。
- 到百度试试手气：毕竟人家也抓了不少网页了，说不定会有Google没抓到的呢？

英语不能太差

当然，这个更加是没底的事情，只是我自身英文也非常差，所以没资格教育别人，推荐余晟老师的一篇博客，供大家学习：《关于程序员学英语的经验》

MOOC 课程

注意: 绝大部分是英文课程

Python

- 计算机科学及Python编程导论
  - 这个是非常棒的计算机开发入门课程(授课老师Prof. Eric Grimson是MIT的副校长) 不过是用python语言作为示范讲解的.
  - 中文字幕(建议和英文字幕对照看)

Java

1. Intro to Java Programming Building Programs with Classes & Objects
2. Java编程导论 - 第1部分：开始使用Java编程
3. Introduction to Java Programming – Part 1
   - (香港大学 港式英语很High)

C

- 程序设计入门——C语言（浙大）
- 计算机程序设计-C 台湾大学

计算机体系结构(组成原理)

- The Hardware/Software Interface
  - 这门Coursera的课程几乎完全取材于《深入理解计算机系统》这本书，连编程作业都是完全照搬CSAPP在CMU的作业
  - 很棒
  - 先修知识:
    - Introductory programming in C or Java
    - familiarity with binary numbers

算法

- 算法
  - 算法，第一部分(普林斯顿大学)
  - 算法，第二部分(普林斯顿大学)
  - 教材：《Algorithms》

关于开发工具

- 关于这个话题，我在知乎上发起了一个话题，欢迎参考。关于开发/编程工具，你有哪些心得或给初学者的建议？
- 过犹不及：所有的工具，都是为了提高我们的开发效率而存在的，但是，如果为了那些工具，投入了太多的精力，则可能舍本逐末，忘记了自己的根本目标。也许有人会说，磨刀不误砍材功，但是，千万不要只顾磨刀，忘记砍材啊。
- 对于初学者而言，GUI是更加自然的方式。虽然，计算机的发展过程，是先有命令行，后有图形界面。但是，键盘、鼠标的综合运用，会更加容易被人掌握。当然，随着熟练程度的增加，纯用键盘操作，加上花样繁多的快捷键，会大大的提高操作的效率。再进一步，有一些优秀的GUI工具，极其丰富的支持各种快捷方式，同时又兼顾图形界面的美观与方便，是最值得推荐的。但是，还是回到那句话：过犹不及。
- 好的教程，会帮助我们的熟悉开发工具，我一直认为TextMate的迅速普及，是由于RailsCast的功劳。因此，花时间找一些靠谱的教程，尤其是视频教程，会很有效。
- 开发人员，往往会有两个误区，或者频繁的挑选、比较多种开发工具，或者长时间埋头于自己最常用的那个工具，对于外面世界的发展毫无兴趣。我的个人建议是，定期抬起头来，看看新工具的进展与介绍。1～2年为一个周期就好，不用太频繁。
- 知其然，更要知其所以然。千万不要变成被工具惯坏了的程序员。
- 不要总试图要汉化它（by 李焕林）
- 工具是给懒人用的，不是给傻子用的。（by 程劭非）
- 还会不定期的补充一些心得进来...

关于开发工具的分类（by 李路）

我认为开发工具分三类，需区别对待：

1. 可以使用一辈子的工具，学习路径几乎没有尽头，值得在职业初期就好好考虑，仔细斟酌进行选择，并在整个生涯中不断努力力求学到更多，你的工作效率会因为这种努力不断提高。如：
    - 编辑器： emacs, vim
    - 基本操作系统环境：如bash
    - 基本编程语言: c / lisp
2. 任何时候都需要掌握的工具，这类工具总是每隔一个周期就有新的产品出现，取代掉旧有的产品，但相对来说是值得学习的，能保持一个较长的时代的有效期，如：
    - 版本控制系统：git
    - 社交网络: stack overflow / github
    - 写作工具: markdown / reStructureText / latex / html
    - 通用编程语言： python / ruby / javascript
3. 特定领域需要的工具，此类工具往往时效性较短，不断被新产品取代，一旦掌握，能在特定领域获得非常高的效率，但缺点是很快会过期，通常是几年之内
    - 各类编程框架： xcode / rails / backbone
    - 各类测试框架：xunit / rspec
    - 用户行为分析工具： ga
    - 各类设计工具： balsamiq
    - 各类项目管理，代码集成工具: github / trac / basecamp / redmine

各种参考资料

以下链接尽可能都给英文维基百科的link，分类页尽可能给维基百科里的Category页

- Programming tool
- Text editors
    - Emacs
    - Vim
    - JEdit
    - Notepad++
    - Sublime Text
    - Atom Github 出品 纯coffeescript(nodejs)开发，我喜欢
- Integrated development environments / IDE List
    - Emacs
    - Dev-C++
    - Eclipse
    - NetBeans
    - Cloud9 IDE
    - Code::Blocks
    - KDevelop

- Revision control software
  - SVN
  - Git
  - Mercurial
- Test-driven development / Behavior Driven Development
  - xUnit
  - RSpec
- Rapid prototyping tools
  - Balsamiq Mockups
  - Axure RP Pro
- Project management software
  - Bugzilla
  - XMind
  - Redmine
  - Trac
  - MediaWiki

关于英文资料

回应Shi YiMin的评论："大学一、二年级的新生看这个英文的资料是不是会有点困难？"。

我觉得，还是要努力的去看吧，如果发现理解有困难，再去寻求各种帮助。毕竟相比对应的中文版，英文版的内容质量要好得多。再者，维基百科里的这些内容，主要还是简介性质的，不算太难。

上一章 | 下一章

Hello World

下载源代码的N种办法

关于源代码管理与版本控制

首先需要介绍一些基础的概念，这里只是简要的介绍，比较详细的介绍，可以参见 Understanding Version-Control Systems，期待有人能够将其翻译为中文。（或者已经有中译本了，欢迎告知我。）

源代码(**Source Code**)：也就是通常一个软件，由程序员编写，并且可以被其他程序员阅读的，可以被直接执行/或编译后执行的文本代码。

源代码管理与版本控制(**Version control/Revision control**)：由于源代码数量的急剧膨胀、变更的越来越频繁、可能修改同一个源文件的人也越来越多，需要将这些代码管理起来，于是每次变更被称之为一次修正 (Revision)。版本控制更准确的说法应该是“Revision control”，每当我们修改一个源代码文件并再次保存时，就出现了两个不同的版本，一个是修改前的，一个是修改以后的。而版本控制，就是确保源文件的每一次修改，都被记录下来，并且可以知道是被谁修改的，是因为什么原因而修改的。必要时，可以找回任何一个版本的源代码。

软件版本号(**Software Version**)：这里的版本，是另外一个概念，源代码中的任何一个文件，都存在一个修订版本号，而作为整个软件，无论对内称呼还是对外发布，都需要一个更加正式的，完整的版本号。前者的英文是Revision，而后者的英文是：Version。因此，当我们谈到版本管理的时候，很可能是同时谈到两者：一个是源代码的 Revision，一个是整个项目的Version。

版本控制工具(**Revision control software**)：为了帮助更好的管理源代码，程序员们开发出了林林种种的版本控制工具，有闭源的，也有开源的。而现在市面上流行的，已经几乎全是开源的了。简单的列出几种在下面：

- 仅管理本地源文件
  - 免费/开源：SCCS (1972) RCS (1982)
  - 闭源：PVCS (1985)
- C/S方式管理源代码
  - 免费/开源：CVS (1990) CVSNT (1998) Subversion (2000)
  - 闭源：Software Change Manager (1970s) ClearCase (1992) CMVC (1994) Visual SourceSafe (1994) Perforce (1995) StarTeam (1995) MKS Integrity (2001) AccuRev SCM (2002) SourceAnywhere (2003) SourceGear Vault (2003) Team Foundation Server (2005) Rational Team Concert (2008)
- 分布式管理源代码
  - 免费/开源：GNU arch (2001) Darcs (2002) DCVS (2002) SVK (2003) Monotone (2003) Codeville (2005) Git (2005) Mercurial (2005) Bazaar (2005) Fossil (2007) Veracity (2011)
  - 闭源：TeamWare (1990s?) Code Co-op (1997) BitKeeper (1998) Plastic SCM (2006)

更多参考资料：

- 上述概念所附带的四个link，都是来自于英文版的wikipedia，有些也有附带的中文版本，顺着这四个主条目的介绍，可以点击阅读更多的词条，加深了解。
- 版本控制工具历史的10个里程碑 本文中的一些link，也非常值得点击过去细细阅读。
- 为什么软件项目从集中式迁移到分布式版本控制系统的热情持续不减? 最新趋势，值得了解。

寻找早期开源项目的源代码

取得源代码的方式千千万万，现在有越来越多的开源项目，已经开始逐步采用规范的，统一的方式，提供自己的源代码以供下载，但是在开源项目发展的早期，还有很多是以并非规范的方式提供的。而在寻找一些开源项目的源代码时，google与wikipedia，将会是我们的好帮手。当然，还有更早期的一些开源项目，就是在邮件列表里发一个带附件的邮件，要找到那种项目的源代码，就得有考古的功力了。

这里举一个例子，来描述一下我寻找某一个开源项目源代码的过程。

有一个项目叫做GForge，在早期还是一个较为著名的开源托管平台的项目，这个托管平台的代码本身也是以GNU许可开源的。假设，我首先是在wikipedia上发现了这个项目：http://en.wikipedia.org/wiki/GForge 看到这个项目的介绍，大概我觉得不错，于是我就看到下面的External links，列出了四个外部链接：

- GForge official website
- GForge project open source page
- GForge Advanced Server page
- FusionForge project open source page

一个比较合理的猜测，自然是首先访问GForge project open source page因为这个项目托管在自己的开源平台上，应该是顺理成章的。进入这个页面之后，我迷惑了一下，他竟然是一个中文界面，里面有文件、档案发行和SVN三个疑似可以下载的地方。经过试错，我发现：文件里没啥东西。SVN的存取信息里告诉我，可以svn checkout http://gforge.org/svn/gforge获得最新的源代码。而在档案发行里，我发现了各个历史版本的列表而最新的那个5.0版本并非一个压缩包，而是一个txt文件，里面告诉我：GForge项目已经被全部重新设计和重写，目前可以在as-help这个项目里看到。另外，如果要下载最新的版本，可以到http://gforgegroup.com/es/download.php

而如果要下载早期的版本，那么历史版本列表的那些gforge-x.x.tag.bz2，就可供我挑选了。

更加糟糕的情况也许会需要我们在google里搜索：“XXX source code download”这样的关键字，才能找到，这里不再赘述。

SVN、Git、Mercurial快速介绍

三本值得一看的书

- Subversion 与版本控制
- progit 中文版
- hginit.com 中文版

另外

- Mercurial 权威指南(这本书的翻译尚未完成，因此推荐直接阅读英文版)

基本上，如果想要开始使用并深入掌握这三种版本管理工具的其中一种，都可以由此入门，并最终融会贯通。

SVN如何获取代码

如果你在windows平台下，推荐安装TortoiseSVN，然后，就在图形界面下简单操作。

填入SVN来源URL，直接CheckOut出最新的版本(HEAD revision)，或者选择某个具体的版本(Revision)，选择CheckOut的目录即可。

如果你在linux/mac平台下，那么命令行会非常方便：

```
svn checkout http://SiteDomain/path/ProjectName 获取最新的版本
svn checkout http://SiteDomain/path/ProjectName --revision {....} 获取某个版本
svn checkout http://SiteDomain/path/ProjectName/tags/Release_x.xx 获取上某某tag的具体发行版本
```

Git如何获取代码

如果一个git管理的项目放在github这样的成熟项目托管平台，获取代码是非常简单的，而且网站还会有详细的操作指南，教你一步一步的如何操作。

至少，你可以先参照GitHub的指南，搭建自己的git开发环境。

然后，获取代码也极其简单：

```
git clone GitRepoURL
```

Mercurial如何获取代码

正如GitHub是主打Git的开源托管平台，BitBucket则是一个主打Mercurial的开源托管平台。因此，我们可以在这里找到关于Mercurial的操作指南。而因为Git的飞速发展，现在BitBucket也开始同时支持Git，所以在bitbucket 101，可以同时看到两种工具的安装指南。

```
hg clone HgRepoURL
```

即可获得Mercurial仓库的完整副本。（Mercurial又简称为hg）

基于包管理的方式获取源代码

这是新增的一个小节，因为在很多动态脚本语言中，都有类似的包管理工具，在wikipedia中有一个词条List of software package management systems，其中的Application-level package managers小节，就介绍了十多种不同语言的包管理工具。

想要更进一步的了解软件包管理系统，可以阅读英文版 中文版

与过去的很多开源项目不同，这种脚本语言的包管理工具实在是方便，一行命令，连下载，带安装、就都有了。本章4.2节的让代码运行起来里，会举几个这样的例子。这里就不再赘述了。

Application-level package managers

语言	包管理工具	相关文档与资源
Perl	CPAN	[1] [2]
PHP	PEAR PECL	[1] [2]
Ruby	RubyGems Bundler	[1] [2]
Java	Maven	[1] [2]
Python	EasyInstall PyPI	[1] [2]
NET	NuGet	[1]
NodeJS	npm	[1]

让代码运行起来

经过思考，决定将这一小节以实例的方式写出。也欢迎大家补充各种不同语言的how to install。

Ruby版

在**Windows**环境下 推荐安装RubyInstaller，进入下载页面，选择一个ruby版本，比如Ruby 1.9.3-p125，下载、运行安装即可。

在**Linux/Mac**环境下 推荐安装RVM或者rbenv

```
$ bash -s stable < <(curl -s https://raw.github.com/wayneeseguin/rvm/master/binscripts/rv
ubuntu下，将下两行中的.bash_profile改为.profile
$ echo '[[ -s "$HOME/.rvm/scripts/rvm" ]] && . "$HOME/.rvm/scripts/rvm" # > Load RVM func
$ source ~/.bash_profile
$ rvm requirements
根据提示，安装其他必要的软件包
$ rvm install 1.9.3
$ rvm use 1.9.3
```

安装一个开源项目

```
$ gem install sinatra
```

搞定收工。。。

如果没有安装过rubygems这个包，则会困难一些。首先在RubyGems这个页面挑一个文件格式下载，并解压缩。然后：

```
$ cd directory
$ ruby setup.rb
```

搞定收工。。。

参考：使用RVM在ubuntu下安装ruby&rails

PHP版

在**Windows**环境下 推荐安装WAMP（Windws+Apache+MySQL+PHP）套件，XAMPP

在**Linux**环境下 以下将简单介绍在Ubuntu下安装LAMP的过程，其他的linux平台以及Mac平台，请自行搜索。

```
$ sudo apt-get install mysql-client mysql-server
根据提示，输入两次MySQL的root密码。
$ sudo apt-get install apache2
在浏览器中访问 http://localhost，应该可以看到It Works等文字。表明Apache2安装成功。
$ sudo apt-get install php5 libapache2-mod-php5 libapache2-mod-auth-mysql php5-mysql
$ sudo /etc/init.d/apache2 restart #重启Apache服务器，完成配置PHP的工作
$ sudo vim /var/www/info.php
将以下内容写入文件
<?php
phpinfo();
?>
在浏览器中访问 http://localhost/info.php 应该能够看到PHP的版本及模块说明等内容。这样就表明，Linux+A
```

安装一个开源项目 以在Ubuntu上安装一个wordpress为例

```
$ wget http://cn.wordpress.org/wordpress-3.3.1-zh_CN.tar.gz #需要寻找最新的版本
$ tar -zxvf wordpress-3.3.1-zh_CN.tar.gz
$ mysql -uroot -p
mysql> CREATE DATABASE IF NOT EXISTS wordpress default charset utf8 COLLATE utf8_general_
mysql> GRANT ALL PRIVILEGES
               ON wordpress.*
               TO 'wp_user'@'localhost'
               IDENTIFIED BY 'wp_password'
               WITH GRANT OPTION;
mysql> exit
$ cp wp-config-sample.php wp-config.php
$ vim wp-config.php
（最主要是修改以下四个值的定义）
define('DB_NAME', 'wordpress');
define('DB_USER', 'wp_user');
define('DB_PASSWORD', 'wp_password');
define('DB_HOST', 'localhost');
```

在浏览器中访问 http://localhost/wordpress/wp-admin/install.php 按提示完成各个步骤，就搞定了，整个时间不超过5分钟。

最后一句的提示，很有趣味："WordPress 安装完成。您是否还沉浸在愉悦的安装过程中？很遗憾，一切皆已完成！ :)"

参考文档 1 参考文档 2

Java版

在Ubuntu下以源代码方式安装Tomcat

```
$ wget http://apache.etoak.com/tomcat/tomcat-7/v7.0.27/src/apache-tomcat-7.0.27-src.tar.g
$ tar -zxvf apache-tomcat-7.0.27-src.tar.gz
$ cd apache-tomcat-7.0.27-src
$ vim BUILDING.txt # 阅读编译指南，按照指示操作
$ 下载 jdk-6u31-linux-x64.bin #tomcat7.0.x的源代码编译，要求jdk6的版本，jdk7在编译时，会报错
$ cd
$ ./jdk-6u31-linux-x64.bin
$ export JAVA_HOME=~/jdk1.6.0_31/
$ wget http://apache.etoak.com//ant/binaries/apache-ant-1.8.3-bin.tar.gz
$ tar -zxvf apache-ant-1.8.3-bin.tar.gz
$ export PATH=%PATH:~/apache-ant-1.8.3/bin:~/jdk1.6.0_31/bin
$ export ANT_HOME=~/apache-ant-1.8.3/
$ cd apache-tomcat-7.0.27-src
$ ant
$ cd output/build/bin/
$ ./catalina.sh run
```

至此，在浏览器中，访问http://localhost:8080/ 可以看到Tomcat的欢迎页。

Python版

在Ubuntu上安装Python Webpy

```
$ sudo easy_install web.py
$ python
  Python 2.7.5 (default, Mar  9 2014, 22:15:05)
  [GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
  Type "help", "copyright", "credits" or "license" for more information.
     >>> import web
```

在Ubuntu上安装Python Webpy -- 另一种方法

```
$ sudo pip install web.py
$ python
  Python 2.7.5 (default, Mar  9 2014, 22:15:05)
  [GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
  Type "help", "copyright", "credits" or "license" for more information.
     >>> import web
```

到这一步，Webpy就算是安装完成了。太简单了，我们还得再用这个框架干点什么。

```
$ cat hello.py
#!/bin/env python
#-*- coding:utf-8 -*-
import web
urls = (
        '/', 'index'
        )

class index:
        def GET(self):
            return "Hello, world!"

if __name__ == "__main__":
        app = web.application(urls, globals())
        app.run()
$ python hello.py
  http://0.0.0.0:8080/
```

这是，用浏览器访问 http://localhost:8080/ 你将看到 Hello, world.

JavaScript(nodejs)版

现在的JavaScript发展的确飞快，已经不仅在浏览器里可以使用了，这里先简单介绍一下服务器端的node.js的安装与使用。

在**Ubuntu**下安装**node.js**、**npm**与**express.js**

```
首先确认GCC编译环境的正确安装，否则请：“apt-get install build-essential”
$ wget http://nodejs.org/dist/v0.6.15/node-v0.6.15.tar.gz
$ tar -zxvf node-v0.6.15.tar.gz
$ cd node-v0.6.15
$ sudo apt-get install libssl-dev
$ ./configure
$ make
$ sudo make install
$ node -v
v0.6.15
$ curl http://npmjs.org/install.sh | sudo sh
$ npm -v
1.1.18
$ npm install express
$ vim test.js
#File Begin
var app = require('express').createServer();

app.get('/', function(req, res){
  res.send('Hello World!');
});

app.listen(3000);
#File End
$ node test.js
```

打开浏览器，访问http://localhost:3000/ 将看到 Hello World!

C/C++版

因为王越的系列文章《Mac OS X背后的故事》，其中第八章《三好学生Chris Lattner的LLVM编译工具链》对LLVM的介绍，使我决定尝试把LLVM，作为c语言的hello world项目。

说实话，在开源项目中，C语言的各种开源项目的编译安装，都是非常类似的。绝大多数命令都是一下三行：

```
$ ./configure
$ make
$ make install
```

当然，在windows下，会麻烦得多。LLVM的安装也非常简单，命令如下：

```
$ cd where-you-want-llvm-to-live
$ svn co http://llvm.org/svn/llvm-project/llvm/trunk llvm
$ cd llvm/tools
$ svn co http://llvm.org/svn/llvm-project/cfe/trunk clang
$ cd ../projects
$ svn co http://llvm.org/svn/llvm-project/compiler-rt/trunk compiler-rt
$ svn co http://llvm.org/svn/llvm-project/test-suite/trunk test-suite
$ cd ..
$ mkdir build
$ cd build
$ ../configure
$ make
$ make check-all
$ make update
$ make install
```

然后，我们可以尝试用LLVM的编译工具链，来编译c语言的代码。 创建一个文件hello.c

```
#include <stdio.h>

int main() {
  printf("hello world\n");
  return 0;
}
```

编译hello.c，得到hello

```
$ clang hello.c -o hello
```

编译c代码，得到LLVM的bitcode文件。

```
$ clang -O3 -emit-llvm hello.c -c -o hello.bc
```

第一行以普通方式，执行编译好的可执行文件，第二行调用LLVM JIT，命令为lli，来执行LLVM的bitcode。

```
$ ./hello
$ lli hello.bc
```

更多内容可以参考：

- Getting Started with the LLVM System
- 参考文档 漫谈C语言及如何学习C语言

如何克服可能遇到的困难

1. 仔细看文档

LLVM的文档很有意思，在他开篇头三条是：1. Read the documentation. 2. Read the documentation. 3. Remember that you were warned twice about reading the documentation.

是的，认真的，非常认真的阅读相关文档，是最重要的方法。其他的一切方法，都是排在这个后面的。

当然，文档的正确性和完整性，也是可以怀疑的。有很多开源项目，往往存在文档bug、文档版本与代码版本不同步等问题。比如LLVM的安装文档，也并非完全正确，我到现在都还不明白，在1.2.3.条那么正经的提示之后，他的第8条，只有configure、make、make check-all、make update，居然没有make install！

但是，先看文档，尤其是项目本身的文档，而非二手、三手文档，是必须的第一步。

2. 注意依赖，注意版本号

优秀的安装手册，会提示你各种版本依赖问题。在开始正式安装之前，核对各种相关系统、工具、类库的版本，是非常重要的。之前我在安装apache tomcat的时候，就发现用java sdk的最新版JDK7，是不能正确编译的。必须将JDK，退回到JDK6.x，才能正确。

说实话，在开源软件安装的时候，估计有60%的问题，都是出在版本错误上。而剩下的40%，多半是看文档不认真的原因。

3. 仔细看出错提示信息，并且在网上搜索答案

在编译LLVM的时候，我遇到了一个古怪的bug，“collect2: ld terminated with signal 9 [Killed]”完全不理解是怎么会事。只能在Google上找答案，幸好，类似的悲剧也在别人身上发生过。我找到了一个简明扼要的回答：“You probably ran out of memory.”。于是，我重新调整了Ubuntu虚拟机的内存大小，就编译通过了。

事实上，将出错信息的关键一行原文，加上开源项目的名称作为关键词，在google上搜索。通常能够找到相关的答案。

4. 不要太早钻研细节

因为是完全从源代码开始安装，所以其实可以有比二进制方式安装多得多的选择。比如configure能够配置的各种参数，一开始最好全部选择缺省，过早陷入安装的细节，会分散我们学习的注意力。

5. 在网上找一找攻略

有很多前人，愿意将自己的安装步骤仔仔细细的记录下来，从而节约我们大量的时间。找到优质的攻略，是事半功倍之道。但是， 要特别注意攻略提及的软件版本，相关环境。在网上搜索攻略时，也类似。比如：不要简单的搜索“ubuntu apache source install”，而是搜

索“ubuntu 10.04 64 apache source install”。这样能够避免照着不同版本的攻略，一个劲的死拼。另外，再一次提醒，不要在百度搜索。在国内的技术圈，有无数的“开发者”，喜欢反复的转载别人的blog和文章。不但不注明原文来源，甚至连格式、排版都丢失了，内容也难免错漏。简直就是一堆垃圾。而在百度搜索，这种垃圾又特别多，躲都躲不开。

所以，最好是在Google搜索！

6. 如果很难搞定，不要一条道走到黑，换个方向

RP问题，是存在的。喘口气，歇一歇，换个方向。试试别的开源项目吧。

上一章 | 下一章

3. 选择一个开源项目

3.1. 到哪里寻找开源项目

开源基金会

大部分开源项目都来自于开源社区，而大部分开源社区背后都有基金会在运作，比较知名的有Apache基金会（专注于Java技术的开源软件），Linux基金会（专注于Linux系统的开源软件），Eclipse基金会（专注于基于Eclipse IDE的开源软件），Jboss基金会（专注于JAVA EE方面的开源项目）等。每个基金会都会有目前该基金会正在进行的项目列表，我们可以从其中寻找自己感兴趣的项目。

- Apache：http://projects.apache.org/indexes/quick.html
- Linux：http://www.linuxfoundation.org/programs
- Eclipse：http://www.eclipse.org/projects/listofprojects.php
- Jboss：http://www.jboss.org/projects

开源项目托管网站

另外，每个开源项目都需要有自己的管理平台，各大基金会的开源项目也不例外，因此，各个开源项目的托管网站是也是一个寻找开源软件的好去处。

目前在业界比较知名的项目托管网站主要有SourceForge，GitHub。

微博上的@OpenERP_Jeff 补充说：launchpad是一个很重要的开源hosting网站，著名的开源项目有ubuntu、mysql和zope。项目计划、蓝图、代码库、bug管理、翻译都很完整。

服务提供商	SVN	Git	Mercurial	介绍
Google Code	支持	支持	支持	Google Code属于“富二代”，其在速度上，使用体验上都优于其他几个托管网站，尤其是其丰富的帮助文档，其中有很多还有对应的中文版。因此，对于初学者来说，比较容易上手，也可以获得一个很好的学习机会。当然，因为都是google.com的域名，所以时不时的会访问不了管理界面，原因大家懂的。另外，因为使用简单方便，相应的其在功能上就相对较弱一点。但是应付一般的项目还是绰绰有余。正在关闭中...
Sourceforge	支持	支持	支持	SourceForge可以算是开源界的托管始祖，很多古老的，知名的开源项目都托管在它上面，其对开源界的贡献估计可以和Apache对 Java界的贡献相提并论了。因此，其在功能上经过了长时间的考验，大家想要的功能都能找到。不过，因为强大，其上手难度也相对较高，而且全英文界面，对于英文较弱的同学来说也是一件很痛苦的事。
Github	支持	支持	不支持	GitHub属于托管界的新贵，伴随着Git的蓬勃发展而发展。越来越多的开源软件使用Github托管。最近因Rails漏洞被Hacker黑了也让其处于媒体的风口浪尖。这个事件从一个侧面反映了其在业界的影响力。
CodePlex	Yes	Yes	Yes	微软的开源项目基地
BitBucket	No	Yes	Yes	Python的项目在上面比较多
launchpad	No	No	No	使用bzr管理代码，项目很多，如Ubuntu等，中国大陆访问速度巨慢无比

维基百科有一个词条，列出了非常非常多的开源托管服务的比较：

- Comparison of open source software hosting facilities
- 自由软件主机服务比较

技术社区

像ITeye，infoQ,OSChina，CSDN等国内知名的技术社区都非常关心开源软件的发展，在社区新闻，月刊等载体上都有最新的，流行的开源软件介绍。同时在社区，还有一个很大的好处，你会找到很多志同道合的朋友，这可是学习过程中的一大乐事。

各大科技公司的研究院

Google的BigTable论文，Amazon的Dynamo论文开启了目前火热的云计算时代，各大科技公司对技术领域的引领能力毋庸置疑，同时，很多公司也是开源运动的忠实支持者，像国外的Google，Facebook，Yahoo，国内的淘宝，百度，盛大，豆瓣等，连曾经的坚决反对者Microsoft也开始通过赞助Apache基金会扭转自己在开源界的形象。关注各大科技公司的研究院最新的开源软件可以了解目前整个业界的技术趋势，而且他们开源的软件的未来发展前景都比较看好，因此，是一个寻找开源软件的有效途径。

谷歌编程之夏(Google Summer of Code)

我个人觉得GSoC是一个寻找开源项目的好地方。原因有以下几点：

1. GSoC有一个历年来的参与者列表(http://www.google-melange.com/gsoc/accepted_orgs/google/gsoc2012)，这个列表中的有各种各样的开源组织可以让想参与的人挑选（今年的列表上有180个组织）。
2. 参与者列表上的组织是经过一定的审核才选出来的，选拔出来的组织一般质量比较好，社区活跃度也很高。
3. 列表上的组织为了吸引参加人员，一般在项目的主页上会有专门的wiki为参加人员导航(比如这个DragonFlyBSD项目(http://www.dragonflybsd.org/docs/developer/gsoc2012student/))，很适合新手快速了解这个项目。 我想即使无法参加GSoC本身，通过GSoC的预热也是一个很不错的机会，可以让想要参与开源项目的人有一个很好的入口。

注：本小节来自于Ashi的邮件，在此表示感谢！

搜索引擎

如果您对某类技术特别感兴趣，或者遇到某个技术难题想找业界成熟的解决方案，搜索引擎（Google，Baidu，Bing）都是一个不错的工具，其在我们寻找开源软件的过程中也起同样的作用。通过一些关键词的罗列就能帮助我们找到感兴趣的开源项目。

博客和微博

关注各个领域大牛们的博客和微博，他们会紧跟技术节奏给我们推荐他们领域最新的开源软件，甚至会为我们讲解这些开源软件的底层实现原理，带领我们入门，这是多么高尚的行为啊。

关注各大科技公司的博客和微博，他们也会经常透露自己的开源计划，以及分析业界的一些开源软件在自己公司内部的应用，这对我们深入了解一个开源软件的技术价值有很大的帮助。

技术问答网站

这是最直接的一个方式，在知乎，StackOverflow，Quora等问答网站直接请人推荐几款开源软件，工程师的热情会让你受宠若惊的，很有可能向你推荐某个开源软件的就是这个开源软件的创始人。

维基百科

在维基百科上，有很多辛勤的人们，在整理着各种相关的资料，例如：

- http://en.wikipedia.org/wiki/Open-source_software
- http://en.wikipedia.org/wiki/List_of_open_source_software_packages
- http://en.wikipedia.org/wiki/Comparison_of_open_source_software_hosting_facilities

等等，在其中搜索相关的词条，还能发现很多的宝藏。

3.2. 什么样的开源项目适合初学者

缘起

说实话，在当初列这个提纲的时候，我并没有想好如何写这一节。但是，开放地做事情，就常常会有奇妙的事情发生，佛家称之为“助缘”，各种对这件事情有帮助的缘分，都会在不经意间出现。

一位叫李军的朋友，给我发来邮件，信中写道：“我想是否我们能够通过沟通，然后你在对我有些了解，给我指出点建议，并且是详细的建议，我看学apache开源框架应该不错的，不知道我是否适合，谢谢。期待你的回复。”

在与他的往来邮件中，我也真的将这一节渐渐的想清楚了。另外，在与李军的讨论中，我还发现，需要开辟一个专门的章节，讨论：“学习开源项目，能够提升软件开发中的哪些能力。”

在此，我想对李军表示感谢，更希望有越来越多的朋友，参与到这个文档的讨论中来，相信它会变得越来越完善。

明确自己的目的

选择一个开源软件，首先要明确的，是自己的动力何在。是出于兴趣？还是出于工作需要？比如，有人对于搜索引擎特别感兴趣，想了解搜索引擎是怎么做出来的？那么首先可以考虑先寻找一些专业的书籍，来了解一些关键的知识点。如果对于某一领域的知识点，缺乏必要的了解，可能完全无法理解一个项目里的代码。在掌握初步的知识以后，自然可以去找Lucene、Sphinx来学习。也可能是出于工作需要，比如平时是用PHP开发Web应用，已经在用某一个常见的PHP框架了，希望能够对这个框架有一个深入的学习了解，甚至希望横向的比较多个不同的PHP Web框架，这些都是非常清晰的目的。自然在学习的过程中，不太会迷失方向。

比较危险的一种，是听说某某项目很有名气，甚至是为了将来找工作比较容易，就贸然一头扎进某个项目中去了。这种学习目的，往往会选择到那种很庞大，也很成熟的项目，打开文件夹一看，成百上千的源文件，根本无法看完，一下子就蒙了，再就是颓了。心想自己大概不是学软件开发的料吧～～

优先选择能够独立运行的项目

开源的项目有很多种类，能够独立运行的项目，当然很多。但是也有不少项目，是其他开源项目的插件，类库，扩展包之类的东西，这些在一开始接触开源的时候，最好不要涉猎，因为理解他们，可能会需要理解他们背后的那个庞然大物，往往会遭遇很多难解的细节，一不小心，就进行不下去了。 当然，还有一类项目，他们虽然是独立运行，但是想要让他们独立运行成功，还得安装、配置很多其他的依赖项目，这个往往会让初学者特别绝望，搞了一个礼拜，居然这个项目都还没有运行起来。。。

所以，小的，能够独立运行的，不依赖于太多其他项目的开源项目，可以优先选择。

选择活跃的项目

项目的活跃程度，包括两个部分，一个是开发者提交新代码的频繁程度。另一个是在社区中对于这个项目的讨论热烈程度。提交代码越是活跃，提交的人越多，越能证明这个项目是很有价值的，也证明这个项目是值得你花精力去学习的。而项目在社区讨论的热烈程度，则能够确保当你遇到问题的时候，能够搜索到别人的答案，或者你自己提问以后，能够有人热心回答你。

当然，活跃程度都是相对的，如果你真的对一个项目感兴趣，可以直接试着给这个项目的作者发邮件，提问题。大多数开发者都会很高兴有人关注他的项目，也会通常会热心的回答你的问题的。

判断代码质量

并非所有的开源项目，都是高手写的，都值得你去学习。事实上，有很多垃圾开源项目，代码仔细一看，写得真是一塌糊涂。所以，试着阅读一下这个项目的代码。至于如何判断一个项目的代码质量，之前我在知乎回答过一个类似的问题《如何让自己写的代码易维护？》。

推荐各位朋友参考一下。

当然，更加推荐的，是阅读《Clean Code》一书，非常好的一本介绍如何提交代码质量的书。附一篇书评，可以一读：《写代码犹如写文章》

选择合适的版本

最后，面对已经发展了多年的开源项目，最好不要选择最新的版本。如果你是在工作中要想使用这个项目，当然应该选择最新的稳定版，甚至测试版、 beta版。但是如果是出于学习的目的，为了减少复杂度，快速的理解这个项目的核心结构与开发思想，选择第一个稳定版，是一个比较妥当的办法。 然后，在初步理解了第一个版本的代码之后，再不断的通过阅读changelogs，追踪最新的版本中的代码变更，体会作者修改代码的目的、手法与技巧。这样应该会有很大的收获。

3.3. 值得推荐给大家的开源项目

项目名称	语言（平台）	所属领域	推荐理由
Petshop4.0	NET	企业开发	1.微软官方用来展示.Net企业开发实力的项目,架构优雅;2.涉及中小型企业开发常用技术;3.网络资源较多
Haozesfx	NET	Winform,SIP协议	1.基于飞信实现的开源项目，实用性强,方便二次开发;2.结构清晰，设计成熟;3.实现了SIP协议
ThinkPHP	PHP	Web框架	1.应用较为广泛的Web开发框架，成熟稳定;2.集成多项先进的设计思想;3.活跃的社区支持和完善的中文文档(帮助手册,Demo,代码注释)
Nginx	C/Linux/Unix/windows/MAC	基础设施	代码优秀简洁，插件系统牛逼，HTTP协议可谓当今码农的必备知识
bottle	Python/Linux/Unix/windows/MAC	web开发	代码优秀简洁，3000多行代码（包括注释）搞定web server

其他：memcached c语言 Free & open source, high-performance, distributed memory object caching system。by Jun Guo

上一章 | 下一章

理解源代码

在这一章中，只打算讨论以命令式编程范型为主的语言，因为其他的编程范型的开源项目，笔者接触太少了（期待各类达人多多补充）。

静态理解

阅读一个开源项目的源代码，通常都很容易。大多数开源项目的托管网站，都提供了无需下载，直接阅读源代码的功能，比较有趣的是，大家可以比较一下 sourceforge、google code 以及github的查看源代码的功能。这分别代表了老、中、青三代开源托管平台，对于查看代码的重视程度。

目录结构

好的开源项目，通常会选择合理的目录结构，来组织自己的代码。而所谓合理，通常意味着遵循最常见的约定俗成。比如：

目录名	含义
conf/configure	各种配置文件
src/source	项目的源代码
doc/document	项目文档
test/unittest	单元测试
tools/utils	相关工具
lib	库文件
app	应用相关的文件（在web项目中经常出现）
controllers	控制器，在遵循MVC模式的Web项目中，经常出现
models	模型，在遵循MVC模式的Web项目中，经常出现
views	视图，在遵循MVC模式的Web项目中，经常出现
db	数据库相关文件
demo/example	相关示例代码
misc	其他杂项
include	头文件所在目录，c/c++项目中常见
out/build	编译结果输出目录
third_party/vender	第三方库
install	安装所需的相关文件

包名与文件名

在软件体系中，包（Package）是一个很重要的概念，与模块（Module）类似，但是又有所区别。一个项目，在初始设计时，就需要做模块划分，每一个大小合适的模块，往往就可以作为一个开发工作单元，分配给某个开发者完成。当然，对于那种大型的、复杂的项目，还需对模块做进一步的细分，比如：子模块（Sub Module）。而包（Package），则往往具有一定的可重用性。我们可以认为，一个模块，开源出去未必会有人来用。而一个设计良好的包，本身就可以作为一个开源项目，放出去给被人使用。

因此，从更加有利于软件开发的协作的角度来说，合理的包命名，就变得非常重要。

越是现代的开源项目，越是懂得不必一切从零开始搭建，所以，我们常常会发现，一个开源项目，他们自己会开发一组Package，同时也依赖一批别人开发的Package。在静态理解项目时，了解一个项目项目有哪些包，以及依赖哪些包，就非常重要。

举例之一：rails是一个著名的ruby开源项目。我们访问它的github主页，就可以看到这个项目的源代码结构。

- 首先需要选择查看某一个稳定版本，比如3.2版
  - https://github.com/rails/rails/tree/3-2-stable
- 然后阅读install.rb文件
  - https://github.com/rails/rails/blob/3-2-stable/install.rb
- 我们可以看到的内容，主要包含两大部分：
  - 编译rails的依赖包：activesupport、activemodel、activerecord、activeresource、actionpack、actionmailer、railties。这些都是rails项目自己开发的包
  - 然后再编译rails本身
- rails本身的包描述文件是：rails.gemspec
  - https://github.com/rails/rails/blob/3-2-stable/rails.gemspec
- 通过阅读rails.gemspec，我们可以了解到，rails这个项目，依赖的包还包括：bundler、sprockets-rails。
  - 事实上，通过阅读activesupport等一系列包的.gemspec文件，我们还会发现更多的外部依赖包。

不同的项目，描述包文件，以及包依赖关系，有各种不同的格式。需要一一分别学习。这里就不再详细解说了。

在一个开源项目中，代码当然是由一个一个的源代码文件组成的。通过查看文件名，往往可以了解一个文件的大概内容。例如：

- errors.rb，通常会与出错处理有关
- i18n.rb，通常会与国际化有关
- logger.rb，通常会与日志有关
- json目录下的两个文件decoding.rb和encoding.rb，自然是JSON格式的编解码相关代码

通常，要迅速的辨认出一个文件名的含义，与领域知识大有关系。例如：http.rb，通常会是处理http协议相关。而request和response，则通常是网络协议中的请求与响应相关的处理代码。对于这些单词的熟悉程度，决定了我们阅读与理解代码的迅捷程度。

类名、函数名与变量名

java是一门很讲究规范的语言,所以他的每一个类，就会对应一个同名的.java文件（内部类除外）。这使得我们寻找类所在的源文件，变得非常简单。当然，这样会造成源文件数量的增加，也许会有人不喜欢。

不同的语言，对于命名有其自己的规范，我们可以做一个列表，来简单列出这些规范。

语言	包/命名空间命名	类命名	函数/方法命名
Java	domainname.package 全部都是小写的单词，以.区隔	ThisClassName 每个单词都以大写字母开头	theMethodName 第一个单词 字母开头
C#	DomainName.Package 每个单词都以大写字母开头，以.区隔	ThisClassName 每个单词都以大写字母开头	TheMethondName 与类名一
PHP	domainname.package 全部都是小写的单词，以.区隔	ThisClassName 每个单词都以大写字母开头	theMethodName 第一个单词 字母开头
C/C++	std::hex 全部小写，以::区隔	CThisClassName 以大写C开头，后续是大写字母开头的单词	TheMethodName 每个单词 母开头
Delphi	MyUnit.Unit2 遵循Pascal命名法:一个名字里如果包含多个单词，每个单词的首字母都要大写，以.区隔	TThisClassName 以大写T开头，后续是大写字母开头的单词	TheMethodName 每个单词 母开头
Ruby	Module::SubModule 每个单词以大写开头，以::区隔	ThisClassName 每个单词都以大写字母开头	the_method_name 全小写单 下划线分隔，!?有特定的含义
Python	mod*submod* 全部小写，以区隔	ThisClassName 每个单词都以大写字母开头	the_method_name/theMeth 全小写单词，以下划线分隔， 类似Java的命名，私有函数 划线开头
JavaScript	无	ThisClassName 每个单词都以大写字母开头	thePrivateMethod/ThePubli 私有函数小写字母开头，公 大写字母开头

这是一个非常粗略，挂一漏万的表格，详细的命名规范，请参考各种具体语言的命名规范文档。

注释与**Readme**

很多时候，开源项目代码里的注释，会给你带来误导。或者会让人不知所云，或者只是写给自己看到TODO，或者代码改了，注释忘记改。种种原因，因此我强烈的不建议过于重视注释。

但是，在业内有一种流派，非常重视注释，而且信奉从源代码的注释，就可以直接生成项目的开发文档。比如JavaDoc这样的东西，在java的开源项目里，简直用到泛滥，也造成了java的很多项目，注释数量比代码的数量还要多。而且，格式规范，千篇一律（为了生成Document），真正有意义的注释内容，少之又少。纯粹是干扰阅读。

很多时候，我都建议阅读代码，就真的去读代码。然后试着从类名、方法名、变量名中，大致“猜出”代码的意义。然后再实际的将代码运行起来，看看执行过程中，这些代码是如何工作的。

总之，不到万不得已，不要先看注释。

虽然我个人对于注释，相当的不重视，但是却非常认同Readme的价值。令我倍感欣慰的是，Github的创立者，也完全赞成这一点。他们将一个项目的首页，直接规定为“代码展示+Readme”，也就强迫所有在Github上安家的开源项目，将更多的精力，投注到Readme的撰写中去。这样形成的良性循环，使得我们可以乐观的预期：越来越多的开源软件，将会越来越重视项目根目录下的Readme文件的价值，将一个项目最为重要的内容，以最为精炼的方式，在Readme中，以结构良好的方式，展现出来。

因此，首先阅读Readme，对于了解一个开源项目，是一个非常好的选择。

UML图

UML是一种软件建模语言，全称为：统一建模语言（UML，Unified Modeling Language）。非常巧合的是，在打算写这一小节的今天，@蔡学庸 发了两条微博：

> 有时候我会一边读源码，一边将我的理解画成图。这是我最近读源码画的图，还没有美化处理。
>
> 阅读源代码时，将代码顺手图像化，有助于我思考与记忆。位置、色彩、形状，这些都是我将源代码图像化时会采用的手段。

我非常期待，他能够就这个问题，谈到更多的心得。在我看来，在阅读源代码的时候，不断记录，在脑海里形成整个项目的全景图像，是非常有帮助的。

关于UML的定义，可以参考维基百科：UML，以下引用一段：

> 统一建模语言（UML，Unified Modeling Language）是非专利的第三代建模和规约语言。UML是一种开放的方法，用于说明、可视化、构建和编写一个正在开发的、面向对象的、软件密集系统的制品的开放方法。UML展现了一系列最佳工程实践，这些最佳实践在对大规模，复杂系统进行建模方面，特别是在软件架构层次已经被验证有效。
>
> UML集成了Booch，OMT和面向对象软件工程的概念，将这些方法融合为单一的，通用的，并且可以广泛使用的建模语言。UML打算成为可以对并发和分布式系统的标准建模语言。
>
> UML 并不是一个工业标准，但在Object Management Group的主持和资助下，UML正在逐渐成为工业标准。OMG 之前曾经呼吁业界向其提供有关对象导向的理论及实现的方法，以便制作一个严谨的软件建模语言（Software Modeling Language）。
>
> 有很多业界的领袖亦真诚地回应OMG，帮助她建立一个业界标准。

在UML系统开发中有三个主要的模型：

- 功能模型： 从用户的角度展示系统的功能，包括用例图。
- 对象模型： 采用对象，属性，操作，关联等概念展示系统的结构和基础，包括类图。
- 动态模型： 展现系统的内部行为。包括序列图，活动图，状态图。

在学习开源软件时如何使用UML，有以下一些经验和忠告：

- 不用使用工具，自动化的生成UML。自己手绘或者用Umbrello这样的开源工具自己绘制，将大大提高阅读并理解代码的能力。
- 首先建立对象的静态模型，也就是先画“类图”。
- 在UML规范之外，可以加一些辅助自己记忆的符号，这个没有一定的规矩，方便好记就行。
- 其次在动态理解的过程中，对关键的执行路径，画出时序图（Sequence Diagram），将有助于深入理解项目的执行过程。
- 对于非面向对象的软件项目，可以参照类图与组件图的模式，画出模块图。也有助于加深理解。
- UML本身有越来越复杂，越来越学术化的倾向，要适可而止。

外部文档

开源项目的文档水平，有如下几个层次：

- 没有文档（比这个更糟糕的，是有一些错漏的，长时间没有更新的垃圾文档）
- 有一个根据代码注释自动生成的XXDoc，通常这样的文档，价值很低。还不如直接去看代码。
- 有一个简单的综述性质的文档，至少告诉你一个项目的大概。
- 有完整的项目文档，这样的项目已经非常罕见了。
- 有各国志愿者帮助翻译的多语言文档。这样的项目，通常已经是世界一流的项目了。

- 有专门的文档、博客、甚至图书，《XX项目源码解读》之类。一般只有Linux、MySQL这样的项目，才有这样的待遇吧。

一般来说，外部文档的可信度并不高，而且往往过时。不到走投无路，我不建议找外部文档来帮助理解。当然，一些太复杂的项目，作为入门导引，看看也无妨。

ZoomQuiet补充：可以借用 CMMI 的几个层次来类比出文档的层次:

- 0级: 无文档
- 1级: 号称有文档
- 2级: 有可用文档
- 3级: 文档完备
- 4级: 文档丰富
- 5级: 文档开放,可持续完善

动态理解

所谓动态理解，就是让项目运行起来，在运行项目的过程中，理解一个项目是如何运行的。（这个有点像绕口令了。。。）

输出日志

所谓日志，在软件领域，通常是指程序运行的一种记录。开发者与维护人员，可以通过分析日志，了解程序的运行状况。日志输出的数量多寡，可以分为以下几种：

- 完全没有输出（这不是一种好的做法）
- 有出错与崩溃时的输出日志（主要用于排除故障）
- 打开某个参数配置的开关，例如将日志级别修改为debug，将会输出更多的日志信息（主要用于调试程序）
- 为了理解特定的片段，直接修改代码，增加更多的日志输出，甚至将代码执行过程中的所有相关变量，全都输出出来。（用于查找疑难杂症，深入理解源代码等目的。）

ZoomQuiet补充：同前,应该使用相同的递进分析层次

- 0级: 无日志
- 1级: 号称有日志
- 2级: 有可用日志
- 3级: 日志级别完备
- 4级: 日志粒度可调节
- 5级: 有通用日志服务,可集中分析

一个比较完善的开源项目，通常会输出一些日志，如果你搜索整个项目的源代码，都找不到（log，logger，logging）这样的关键字，那就比较糟糕了。

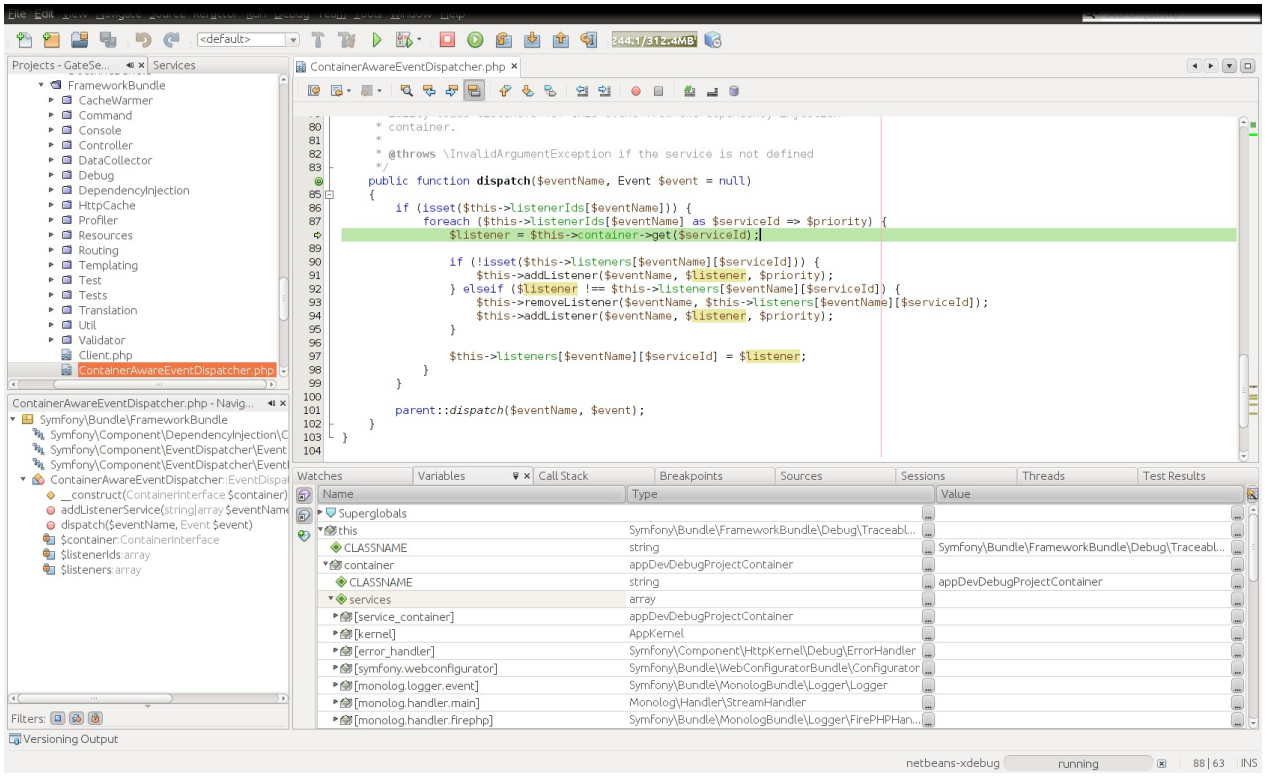
通过修改源代码，以增加更多的日志输出，是我们常用的一种手段，最好是能够找到项目中可供参考的输出日志的办法，照着那个例子来改写。如果实在找不到，或者调用存在一些陷阱，也可以自己纯粹手工的添加日志输出代码。简单的举一个ruby的例子，下面的这一行代码，就可以输出一段内容到日志文件里去了。

```
File.open("temp.log","a") { |f| f.puts("log info") }
```

输出日志，能够解决大多数情况下的理解需求，唯一可能会有陷阱的，则是多线程程序输出的日志，因为每次输出的内容可能次序不一致，因此需要特别小心。

设置断点与单步跟踪

IDE有一个重要的好处，就是可以帮助程序员调试程序。在一个图形化界面里，跟踪调试程序，有着纯文字界面难以比拟的便利性。



Netbeans IDE 7.0 调试PHP的程序片段

在一个宽屏的显示器里，同时显示源代码树、对象结构、当前执行到的代码行、当前的各种变量值、调用序列、各个线程、输出内容等等等，还是很爽的。 当然，要使得IDE能够调试一个开源项目，还是有很多琐碎的事情需要处理。例如：

- 如何在IDE中打开一个项目
- 如何在IDE中配置一个项目的依赖项
- 如何在IDE中编译并运行一个项目
- 如何在IDE中设置断点

这些如何，因语言、平台、IDE、版本、具体项目的不同，而有所区别。这里没法给出一个周到全面的解决方案，但是可以给一些搜索方面的建议：

- 假设要在Netbeans 7中打开一个开源的Java项目，可以搜索“how to open java project in netbeans 7”，然后我们可以找到一些文档：
  - http://netbeans.org/kb/docs/java/project-setup.html
  - http://stackoverflow.com/questions/4382619/how-can-i-open-non-netbeans-java-project-using-netbeans

当然，一定会有各种让人挠头的问题，各位多多尝试吧。另外推荐一本书，是张银奎写的《软件调试》，大部头。但是的确是一本好书！

http://book.douban.com/subject/3088353/

抛出异常

有很多种语言，都支持异常处理，以及手动抛出异常。在特定的位置，将整个调用序列打印出来，可以方便我们快速的找到整个项目，是从何处开始，又是如何一层一层的调用，最终到达我们设置抛出异常的位置的。

在Java语言中，我们可以这么写：(new Exception()).printStackTrace(); 在PHP语言中，我们可以直接调用函数：debug_backtrace();或者debug_print_backtrace(); 在Ruby语言中，我们可以这么写：

```
begin
  1/0
rescue => exception
  puts exception.backtrace
end
```

其他种类的语言，建议各位可以自行Google。

在调用了类似的函数之后，我们可以或者类似如下这样的输出：

```
java.lang.Exception
        at org.jruby.lexer.yacc.LexerSource.getSource(LexerSource.java:147)
        at org.jruby.parser.Parser.parse(Parser.java:122)
        at org.jruby.Ruby.parseFile(Ruby.java:1965)
        at org.jruby.Ruby.parseFile(Ruby.java:1969)
        at org.jruby.Ruby.parseFromMain(Ruby.java:364)
        at org.jruby.Ruby.runFromMain(Ruby.java:327)
        at org.jruby.Main.run(Main.java:214)
        at org.jruby.Main.run(Main.java:100)
        at org.jruby.Main.main(Main.java:84)
```

以上信息，清楚的现实了某某源文件的某某行，发生了一个调用，进而在下一个源文件中的某个函数，被调用了。于是，我们就可以从抛出异常的代码，开始逐级回溯，阅读相应的代码片段。

修改代码，破坏性尝试

在初步理解了项目代码之后，可以尝试做一些简单的修改，看看会发生什么。

- 编译错误

静态类型的语言，在这方面有更多优势，通过编译时给出的错误提示，你可以知道自己改坏了什么，以及为什么它被改坏了。比较令人头痛的，是用了各种复杂的模板语法的C++语言，这时候编译器可能会给出一些令人莫名其妙的报错信息，那你就抓瞎了。不过，开源初学者，一上手就去啃复杂的C++代码，也很难说是明智的选择。

- 运行时报错

与编译期报错类似，你也可以通过阅读一堆一堆的报错信息，了解代码是如何运作的，以及为什么原来不报错，现在就开始报错了。 不过，触发运行时报错，可能会有些困难，这涉及到你修改的代码，是处于主线还是支线位置，以及有多少相关功能，依赖于你所修改的代码。

- 功能失效

可以尝试注释掉一些代码，看看会发生什么事情，比如某个功能按钮失灵甚至消失。当然，对于JavaScript代码，我们常常会喜欢在某些位置加上alert，看看什么时候会弹出一个消息提示框，却什么事都没有完成。

- 走走捷径

我们常常会看到的一类代码，是长期发展完善后的产物。从接收消息，要处理某某事件开始，到真正去完成某某功能，其间还做了很多的额外工作，比如数据校验，纪录日志，触发其他消息等等，将这些代码全部注释掉，大多数功能都还是正常的，但是，会出各种意外，可以进一步做出各种尝试。（详见5.3. 主线与支线）

工具

有很多辅助代码阅读工具，简单的介绍一些

- IDE类

既然是IDE，自然大多数都已经集成了相当不错的代码阅读功能，索引，反查，标注，自由跳转，通常都应有尽有。比较好的IDE，有Eclipse、Netbeans、IntelliJ IDEA、Visual Studio(Express)等等，各种面向专门语言的IDE还有很多，可以自行在Google搜索“Best XXX IDE”。

进阶阅读：18 个最佳代码编辑器/IDE推荐

- 传统神器＋超牛插件

所谓传统神器，自然是指Emacs与VIM，这两款神器不能简单的以是否IDE来划分。他们都带有众多的插件，可以扩展出惊人的灵活性。

推荐阅读两篇文章： Emacs和它的朋友们——阅读源代码篇 自己动手定制一个高效阅读源代码的vim

- 专业工具

专业工具，通常也是由商业公司开发的收费软件，人家要挣钱，自然也要有拿得出手的好东西。这里推荐两款软件，更多介绍，请自行搜索相关介绍与教程：

- Understand
- source insight

- 软件工程相关工具

所谓软件工程的相关工具，主要是一些与面向对象概念相关的UML工具，比如IBM的Rational Rose、Sybase的Power Desinger、还有Borland的Together，都是大公司出品的一些大家伙。可以将面向对象语言（主要是Java）的项目，转换输出成为UML的各种图，有一定的参考价值，玩玩也不错。

主线与支线

一个开源项目，我们总可以从中找到主要的功能与次要的、辅助的功能。一个系统从启动开始，到完成最核心的功能、任务，最后成功结束，就是一条主线。而各种可能达到的次要功能，就是支线。

一个项目在最初创建的时候，总有一组想要实现的核心功能，其他都是次要的目标。比如：我做一个BBS，要让用户能够登录，发言（或者回复评论他人的发言），退出登录。这就是主线。而：设置用户头像，版主的各种权限与管理功能，顶、 踩功能，帖子内投票，站内短信等等等等，都可以算是次要功能。

在一个项目中，首先清理出主线以及其相关的代码，是一个非常必要的工作，主要的入手处，是项目的Readme文档。另外，对于某一领域的基础知识，也非常必要。

寻找入口

看一堆代码，总要找一个入手处。而不同的项目，入手处是不一样的。举几个例子：

- 一个满足MVC模式的Web项目，通常会有多个Controller，这些Controller就是代码的入口处。如何才能找到某个访问路径对应的Controller呢？这就需要去看Route的定义了。

- 一个较为普通的C/C++/Java项目，通常会有一个或者好几个含有main()函数的文件，那个就是代码的入口处。
- Make、NMake、Rake、Ant、Maven等等，都是各种不同语言与项目，可能会用到的批处理任务管理工具，有很多的项目，是以这些配置文件中描述的文件，作为入口的，这自然需要理解相应的配置文件的语法规则。
- 5.2.3节中，描述的抛出异常的方法，也是找到入口的办法。

跟踪关键流程

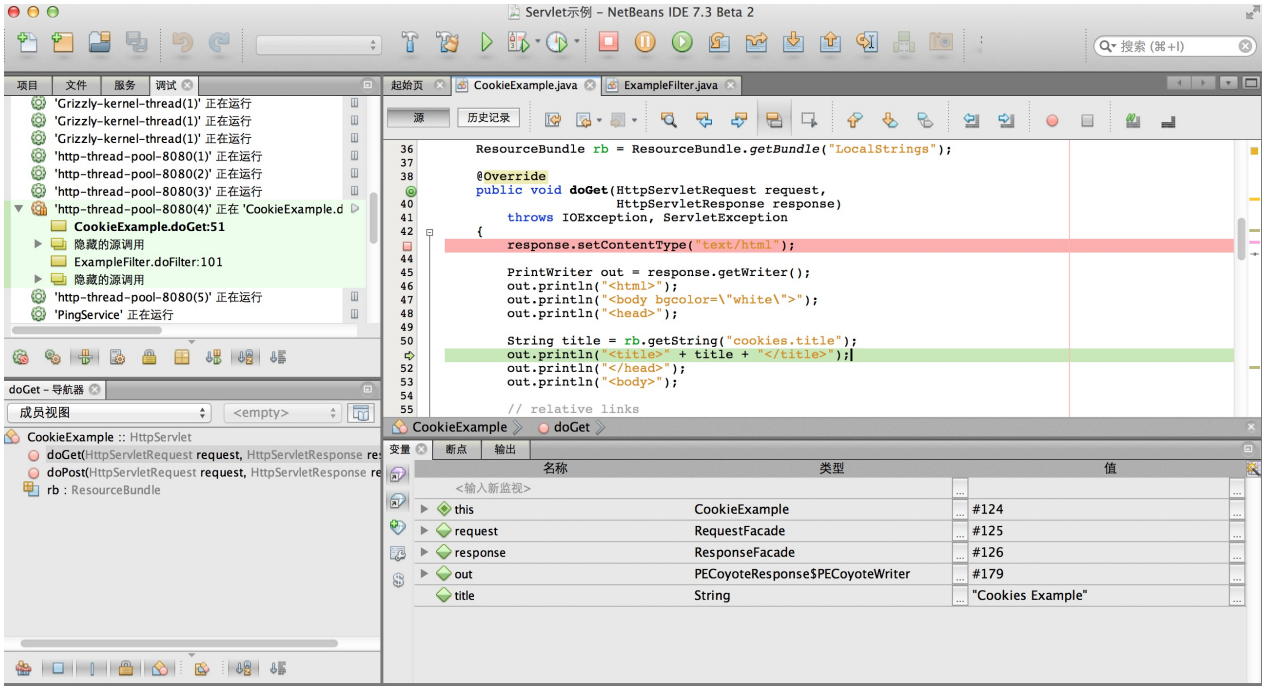
所谓跟踪关键流程，核心要义，就是学会使用调试器。包括集成开发环境（IDE）与命令行方式的调试器（例如GDB）。

无论何种调试工具，其基本的用法都是一致的：

- 设置断点
- 运行时中断
- 单步跟踪
- 查看或修改内存中的各种变量

在下图所示的Java代码debug截图中：

- 我们将断点设置在CookieExample.java的第43行（红色行）。
- 以debug方式运行后，代码将会停留在第43行。
- 我们连续按六次F8，表示代码向下执行六行，目前停留在第51行（绿色行）。
- 在窗口的下方，我们可以看到当前内存中的各种变量，例如title这个变量，就是String类型，值等于："Cookies Example"
- 在屏幕的左下角，我们可以看到这个java源文件包含两个方法与一个实例变量。
- 在屏幕的左上角，我们可以看到这个项目执行的调用序列，撇开那些外部框架的代码，我们可以认为是ExampleFilter.java的doFilter方法在第101行时调用了CookieExample.java中的doGet方法。



最笨的办法，就是一行一行的跟着代码的执行，反复的观察并理解，这些代码究竟做了哪些事情。随着对代码理解的加深，我们可以每次多执行几行，跳过一些已经没有疑问的代码，在更多的地方设置断点，甚至可以在运行时，试着修改某些变量的值，看看会发生什么变化。

跟踪关键流程，最容易犯的错误，就是单步跟踪时，迷失在代码的丛林里，晕头转向。及时识别出不太重要的代码（比如，试着注释掉一些代码段，看看会不会造成太大的影响），并且跳过这些段落，是保证有效追踪的关键。

说实话，要将代码调试这个事情讲解清楚，值得写整整一本书，有兴趣深入学习的同学，可以找一些专门的书籍来看，例如：《软件调试的艺术》、《Debug Hacks中文版》

理解插件体系

在较为现代的开源项目之中，常常会出现某种插件体系结构，整个系统可以分为三个主要的部分：核心功能+插件管理框架+扩展插件。

通过这样的体系结构，开源项目的主创人员，可以集中精力实现自己设想中的功能，而将其他扩展功能的开放任务，交给有兴趣的外部开发者，一方面可以吸引更多的人来参与，另一方面，也可以有效的隔离外部开发者对于核心架构的干扰。

因此，现在很多参与开源项目的方式，并非一开始就可以加入核心代码的开发，而是围绕着插件体系，为系统新增一个插件，或者修改其中的一个插件。对于复杂的系统而言，有时候一个插件，也成为了一个独立的开源项目，可以自由的参与其中。

不同的语言，不同的开源项目，可能具有不同的插件体系，较为著名的，有Eclipse的插件体系OSGi、FireFox的插件体系、jQuery的插件体系、Redmine的插件体系以及Joomla！的插件体系等等。

在维基百科上有一个关于插件的概要介绍： http://en.wikipedia.org/wiki/Plug-in_(computing)

要理解某个项目的插件体系，通常需要找到专门的介绍文档，仔细阅读。例如Java的OSGi，已经非常的复杂，值得专门写N本书来深入介绍：豆瓣搜索OSGi的结果

这里做一些简单的介绍的，是在不看文档的情况下，试着从代码寻找答案的初步方法。以Joomla!CMS为例：

将Joomla的源代码解压缩以后，我们发现有三个目录的名字，较为可疑：components、modules、plugins，看上去都是作为扩展插件而存在的。

首先查看components目录，里面有一堆的以com_开头的文件夹，看起来就是每一个文件夹，是一个组件。进入其中的一个文件夹：com_banners。打开banners.php，发现有三行关键代码：

```
$controller = JControllerLegacy::getInstance('Banners');
$controller->execute(JFactory::getApplication()->input->get('task'));
$controller->redirect();
```

其中Legacy这个单词，引起了我的注意：看来component是一种遗留类型的插件，每个插件，也以MVC的风格写成。在源文件里搜索“class JControllerLegacy”，于是发现了一组文件：

```
/libraries/legacy/controller/admin.php
/libraries/legacy/controller/form.php
/libraries/legacy/controller/legacy.php
```

看来都是用来处理遗留的扩展系统的controller的。

继续阅读com_banners里的代码，在models目录下，有一个banner.php文件，在其中我们又看到了一行值得注意的代码： class BannersModelBanner extends JModelLegacy 看来是与处理遗留扩展系统中的controller类似，用来处理model的。在源文件里搜索“class JModelLegacy”，于是发现了另一组文件：

```
/libraries/legacy/model/admin.php
/libraries/legacy/model/form.php
/libraries/legacy/model/item.php
/libraries/legacy/model/legacy.php
/libraries/legacy/model/list.php
```

从这8个文件入手，我想就可以看明白component的扩展机制了。

在看modules目录下的文件，也是一堆以mod_开头的文件夹，打开其中一个mod_banners/mod_banners.php

也能发现一行值得注意的代码：

```
require JModuleHelper::getLayoutPath('mod_banners', $params->get('layout', 'default'));
```

在源文件中搜索“class JModuleHelper”，就会另外发现一个文件：/libraries/legacy/module/helper.php

看来是处理遗留扩展系统中的modules的，读懂这个文件，应该就可以理解Module的扩展机制了。

最后看看plugins下的代码，与component和module不同，plugins是可以分层的，在authentication目录下，还有gmail、joomla、ldap三个目录，每个目录下，又有两个关键的文件，而且都与目录名（插件名）相同，gmail.php、gmail.xml。看来一个是gmail认证的实现文件，一个是这个插件的配置文件。

在源文件中搜索“class JPlugin”，可以发现两个文件：

```
/libraries/joomla/plugin/helper.php
/libraries/joomla/plugin/plugin.php
```

配合这两个文件，加上插件目录下的.php与.xml文件，应该就可以较为清楚的理解plugins的实现机制了。

外围代码

必须存在的外围功能

当我们阅读一个开源项目的代码时，当时首先是找到主线代码，然后努力去理解其核心的内容。但是，一个完整的开源项目，必须存在很多外围的功能与代码，除了文档之外，还有其他一些必不可少的内容。

- 例如，一个C/C++的开源项目，通常会有的Makefile文件，已经较为正规一点的项目都会提供的configure脚本文件。没有这样的文件，想要自己编译整个项目，几乎是不可能的。
- 再比如：一个项目如果与数据库相关，必备的数据库建表、初始化数据灯工作，就得有一些工具来辅助。最糟糕的，是写一个txt文档，让你照着做，好一点的是提供一个或一组SQL，供人执行，这方面Rails做得非常的到位，我们通常可以在一个Rails的开源项目下，找到一个/db/migrate的目录，这里面提供的脚本，是专门用于数据迁移的。
- 一些好的开源项目，会允许多种不同的运行模式，例如在运行时加上debug参数，就能够输出更多的调试信息，供使用者查找可能存在的问题。在阅读理解代码的时候，这些调试代码也是辅助理解开源项目的好方法。

- 依赖管理也是一个麻烦的事情，这方面有两个较为好的榜样，Ruby的开源项目往往会在根目录下附带一个Gemfile文件，下载源码之后，只要再执行一个bundle install即可。Node.JS也有类似的好习惯，根目录下的package.json文件，就是类似的定义包依赖的文件，只要执行npm install，就万事大吉。
- 负责任的开源项目，在提供源代码之前，就已经做了充分的测试，而且通常是按照单元测试的规范来做的。对于C/C++项目，我们在make之后，一般可以make test，或者对于Java项目，我们可以ant test。对于ruby项目，我们可以rake test或者rspec xxx.rb
- 当然，负责任的开源框架，通常还会提供demo与example，连带单元测试的介绍，我们都会再下面的两节，详细展开。

单元测试

在软件开发的各种最佳实践中，我认为最具有其他性的，是“测试驱动开发（TDD）”，而最容易误导开发者的，则是“设计模式（DP）”，在阅读开源项目的代码时，如果我看到编写完整而全面的单元测试代码，我自然会对整个项目的质量高看一眼。而如果我在源代码里看到了充斥着陈腐味道的各种设计模式的名字时，我会猜测这是一个“玩具项目”——也就是那种经验不足的开发，学习练手的项目。

单元测试，也有高下之分，好的单元测试，会测试关键逻辑与核心算法；而敷衍了事的单元测试，则会去测试get/set这样的代码，看起来一堆assert，却毫无价值。

阅读单元测试的代码，也有一种弊端，就是因为陷入点点滴滴的琐碎细节，进而迷失了方向。所以，不要在一开始就去阅读单元测试的代码，而是先对项目的整体有所了解之后，再开始阅读。

再者，好的单元测试的代码，会特别注重使用场景的构建：某个功能点，在什么情况下，被预期会做出什么样的反应。而构建使用场景的代码，主要集中在startUp这样的函数里，以及各种Mock对象的构造之中，因此，关注startUp与Mock创建，而不是仅仅去看一个一个的testXXX()方法，会更容易理解开发者的意图。

demo/example

我们可以简单的将开源项目划分成两类，一类是给最终用户使用的项目；一类是基于这个项目，可以继续做开发的。 对于第二类项目，有可以分为几类：开发框架（各种Web MVC框架）、基础服务（MySQL、Message Queue）、可以被插件扩展的软件（FIrefox、Chrome）、编程语言（Ruby、Python、NodeJS）、模板引擎（SaSS、Less、HAML）等等。

所有这第二类项目，都需要告诉使用者，应该如何使用自己的项目，给出开发文档、入门教程、指南、手册之类，当然多多益善。但是，最能够指导用户学习的，却是demo或者example。

一个负责任的开源项目，就应该给出足够覆盖自身功能特性的例子，以指导开发者的使用，同时也是对自身功能特性的一种全面的检验。

对于初学者而言，运行demo，尝试修改，嫁接组合，其实就是类似理解一个开源项目的，较小规模的阅读与理解。所以，好的demo，不仅应该是丰富，最好还是循序渐进的。从Hello World级的demo，到PetShop级的demo，都会对开发者有很大的帮助。

至于如何阅读理解这些demo，其实就是将之看成一个小型的开源项目，然后应用本章的各种方法而已，在此不再赘述。

知其所以然

有一句俗语叫做："知其然，更要知其所以然"。用在任何学习科目上，几乎都是恰当的。本章叫做《理解开源项目》，而之前的4个小节，可以说都是属于"知其然"的功夫。如何才能知其所以然呢？

所以然包括哪些内容？

往大了说，整个这份文档，希望帮助读者达到的，就是能够对于开源软件"知其所以然"。这样才算是真正提高了软件开发的能力。因此，我们可以将"架构决策"、"代码风格"、"领域知识"、"编程技巧"等等内容，都算作是所以然的一部分。

架构决策 通过深入阅读和分析源代码，理解整个项目，为何像这样，而不是那样做架构设计。其间蕴含着项目作者的经验和智慧，理解了这个，将是一种巨大的收获。

代码风格 每一种语言、每一个社区、每一个开发者群体，甚至每一个开源项目，都有其独特的代码风格，这种风格，有其背后的合理性，也有很多是来源于某种开发哲学的思考。理解一种代码风格，就是理解一种思考的模式，一种思想的体系。能够多了解一些不同种类的代码风格，对于提高软件开发能力，将有很大的帮助。

领域知识 有些代码不容易看懂，很重要的一个原因，是这个项目所涉及的领域，我们没有什么深入的了解。多年的程序员经验告诉我，要做好某一个行业的软件，一定要成为某一个行业的内行。甚至要比那个本行业的业内人士，更加精通。因此，一个优秀的程序员，通常是能够跟你聊多个不同行业的话题的。强大到你几乎无法分别他的是不是业内人士。因此，通过理解开源项目，进而理解相关的领域知识，会有很多收获。

编程技巧 阅读优秀的开源项目的代码，有时候很像是看一本好书。细细品味，慢慢的体会。我们会发现一点一滴的"妙处"。这些妙处凝聚了程序员的巧思妙想，能够体会得越多，对我们的帮助也就越大。

所以然还包括的一些内容：

个人偏好 开源作者也是普通人，他们有很多观点和取舍，未必能够说服他人，只能算是他们自己的偏好。而他们将自己的偏好表达在代码里，有些时候，我们能够很容易理解（因为我们也是这样想的）。有些时候，我们就会感觉很不解，而且，常常会发生的一类故事就是：某某大牛写了一个开源项目，另一个大牛有感觉不爽的地方。提了意见建议，人家又不肯改。结果，这另外一个大牛，就一怒之下，另起炉灶，写了一个新的开源项目。

历史原因 有一篇很有意思的文章，解释了《为什么Vim使用HJKL键作为方向键》，其实原因很简单。当Bill Joy创建Vi文本编辑器时，他使用的机器机器是ADM-3A终端机，这机器就是把HJKL键作为方向键的。

如何搞清楚这些所以然呢？

思考 当然，这种思考应该伴随我们“通过开源项目，学习软件开发”的始终。但是，从方法论来说，可以尝试从以下一些角度出发来思考：

- 如果我来做一个，会如何去做？
- 如果能够对这个项目做减法，我可以去掉那些模块和代码？真的能够去掉吗？
- 通过阅读单元测试，理解开发者的设计思路。
- 尝试做一些破坏或者修改，来理解项目中的那些做法。这个在下一章会更详细的讨论。

讨论 到开源社区去，发起一些讨论。当然，前提是你必须经过足够深入的思考。不要尽是问一些傻问题

向作者提问 与上面的讨论类似，前提是要先思考。当然，还有一个讨巧的办法，你可以提出：翻译这个开源项目的英文文档，然后，对方当然会很高兴——国际化了嘛。然后，你可以在翻译的过程中，提出各种各样的问题。。。

阅读指南 有些著名的开源项目，本身也非常复杂，所以会有一些文档与书籍：linux内核代码分析、MySQL源码解读、PHP源码分析等等。如果有心学习这样的大型开源项目，这种入门指南，也是很有价值的。但是，这毕竟是别人嚼过的饭，肯定不如自己去啃来的香。所以，还是回到那句老话，源代码才是最有营养的。

参考文档

- 看源代码那些事
- Tomcat7调试运行环境搭建与源代码分析入门
- Redis代码阅读
- 通过修改JRuby，给我的ruby代码加密

上一章 | 下一章

修改开源项目

经过挑选，我决定拿最近比较火爆的开源游戏2048来练手，因为这个项目的规模不大，改起来应该不难。

从需求出发，修改代码

众多的2048改编版，都非常类似，就是将数字的加法，变成某种等级序列的递进。比如：两个夏合在一起，变成商；两个商合在一起，变成西周。

确立了目标之后，我们需要找到在哪里修改代码，当然，前提是我们得把代码搞到本地来。

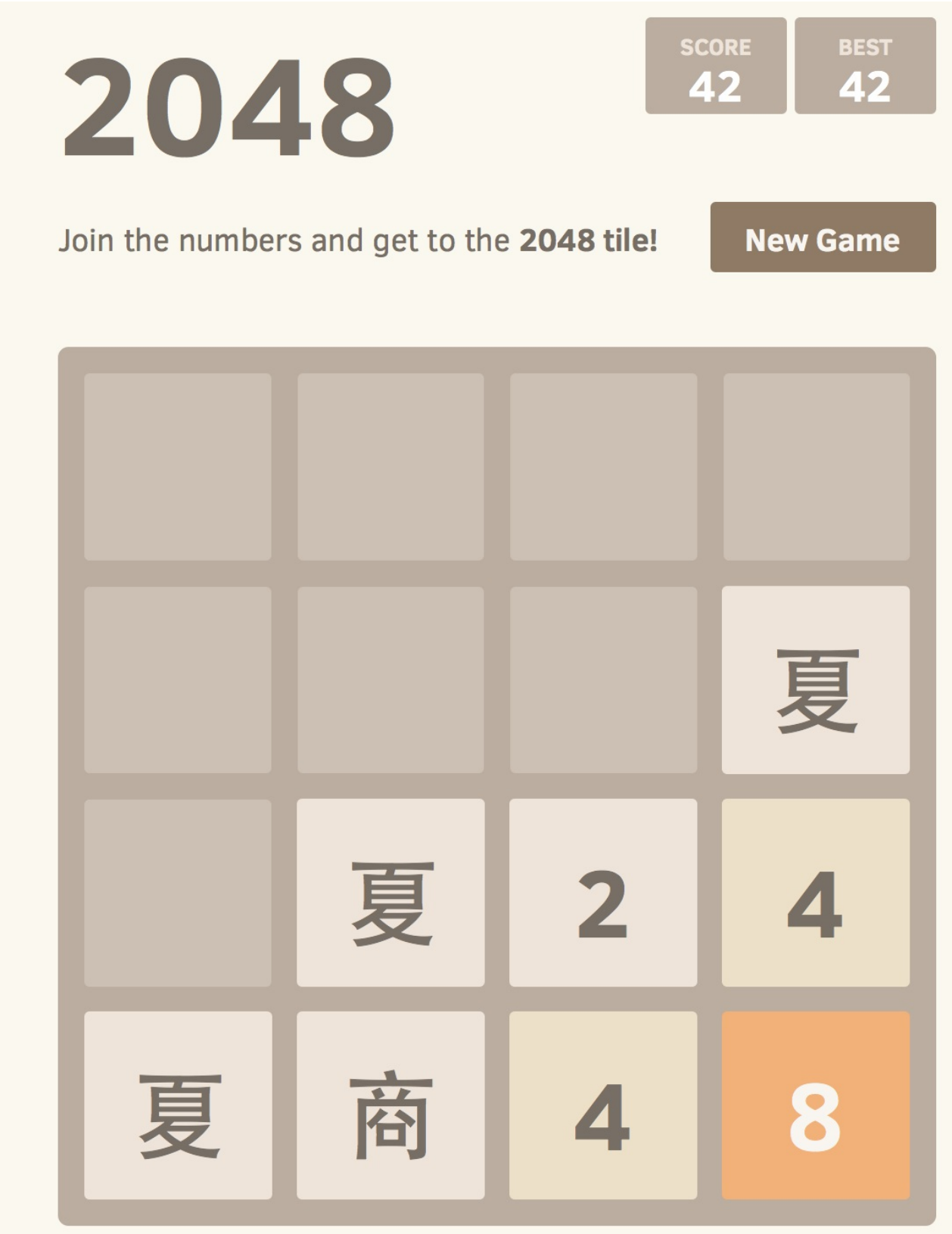
```
首先访问：https://github.com/gabrielecirulli/2048 然后fork出一个自己的版本
例如：https://github.com/zhuangbiaowei/2048

然后，在自己的机器上，执行：
git clone git@github.com:zhuangbiaowei/2048.git
```

随后，挑选自己的喜欢的编辑器，打开2048的目录，开始查找可以下手的地方。

在js目录下的game_manager.js，我们发现了一个函数，叫做：addRandomTile。在这个函数里，有一行是这么写的：`var value = Math.random() < 0.9 ? 2 : 4;` 这种搞法，真的非常粗暴，随机加入一个方块，有90%的概率是2，有10%的概率是4。

所以，我也选择了比较粗暴的搞法，在这个js的第一行，加了一句：`NameArray = ['夏','商','西周','春秋','战国','秦','汉','三国','两晋','南北朝','隋','唐','五代十国','宋','元','明','清','民国','新中国'];` 然后把刚才的那句改成：`var value = Math.random() < 0.9 ? NameArray[0] : NameArray[1];`

这是唯一的修改，我们可以在浏览器里打开index.html，看看效果。



每次出来的方块，的确是朝代名称了，但是加在一起之后，却全都变成了2。

还是在game_manager.js，我们发现了如下一段代码：

```
var merged = new Tile(positions.next, tile.value * 2);
merged.mergedFrom = [tile, next];

self.grid.insertTile(merged);
self.grid.removeTile(tile);

// Converge the two tiles' positions
tile.updatePosition(positions.next);

// Update the score
self.score += merged.value;

// The mighty 2048 tile
if (merged.value === 2048) self.won = true;
```

看来这就是我们要修改的关键所在了。`tile.value`的值现在是一个字符串，所以不能简单的乘以2，我们可以先找到它在NameArray里的位置，然后取他的下一个值。

```
var pos =NameArray.indexOf(tile.value);
var merged = new Tile(positions.next, NameArray[pos+1]);
```

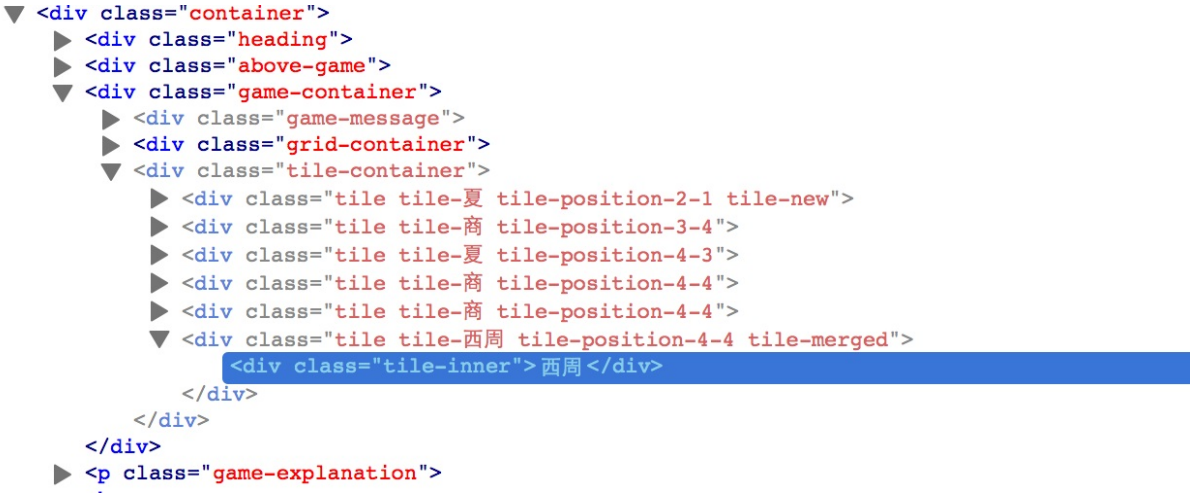
另外，`merged.value`也不能直接加到`self.score`里去了，要改成:

```
now_value=Math.pow(2,pos+2);
self.score += now_value;
if (now_value === 2048) self.won = true;
```

再运行一下看看，发现了一些丑陋的bug。

1. 颜色不对，不同的朝代应该有不同的颜色
2. 字体大小不对，两个字的朝代，有一个字就到下面去了

打开Firefox里的Firebug，我们可以查看到问题所在：



原版的2048，相当粗暴的直接将方块里的内容，当成css的class的内容。因为现在的方块里都变成了汉字，所以我们得将汉字换算成实际的数字。

在html_actuator.js中，我们找到了这样一句：`var classes = ["tile", "tile-" + tile.value, positionClass];`

我们可以在这一行的前面，补上两句：

```
var pos =NameArray.indexOf(tile.value);
var tile_value=Math.pow(2,pos+1);
//再修改一下刚才的那句：
var classes = ["tile", "tile-" + tile_value, positionClass];
```

于是，正常的颜色就会出现了。至于字体大小的问题，我们得回到main.css中去找答案。

```
.tile .tile-inner {
  border-radius: 3px;
  background: #eee4da;
  text-align: center;
  font-weight: bold;
  z-index: 10;
  font-size: 55px; }
```

我们可以简单粗暴将 `font-size: 55px;`，改成 `font-size: 35px;`。我们看一下现在的效果：

OK，打完收工!

指导开发者快速学习编程的网站推荐

- 英文原文：9 Useful Websites to Learn How to Code Quickly
- 翻译: wangguo

互联网是一个飞速发展的领域，从HTML到HTML5、从CSS到CSS3、从JavaScript到JavaScript框架，技术在不断地更新换代。如果你不能跟上这个形势，你将会被淘汰。因此，快速掌握一门语言或一项技术，对于你的Web开发工作将是百利无一害的。

本文为你带来了9个实用的在线教程，一些是互动形式的，还有一些则以全面的指南和可视化形式帮助你快速掌握一门语言。它们有共同的目的，让你的学习更加有乐趣，以及更容易让你掌握这些知识。

LifeHacker Learn to code

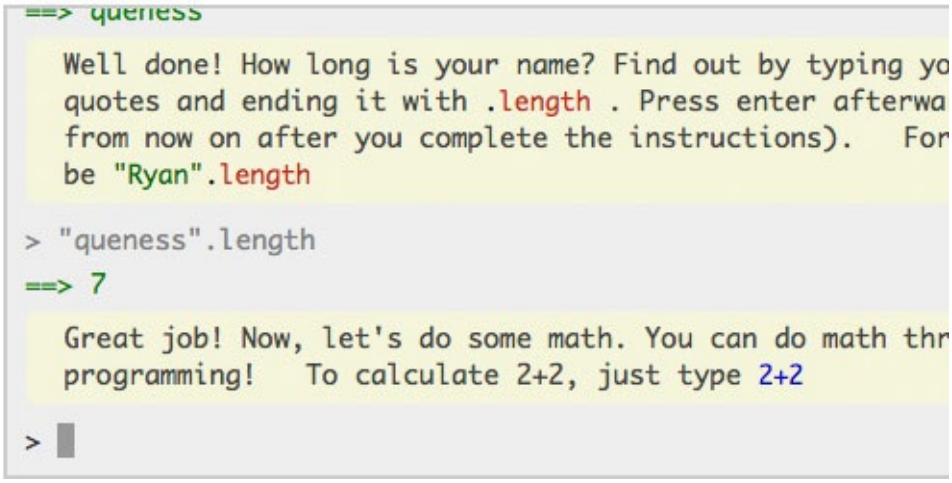
语言：JavaScript

4节课+最佳实践/资源，教你如何使用Javascript进行编程。每节课有一个视频和详细的文章，来确保你理解这些内容。

CodeCademy

语言：JavaScript

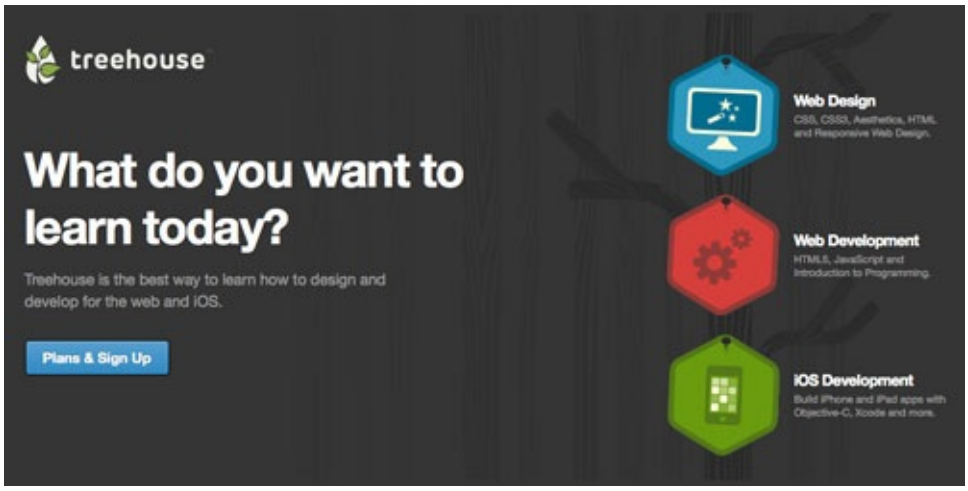
这是一个互动的教学形式，让你一点一点掌握JavaScript。每完成一个课程，你可以得到一些成就点和徽章，通过该形式来激发你的学习兴趣。



TeamTreeHouse

语言：CSS、CSS3、HTML、HTML5、JavaScript、Basic编程和iOS开发

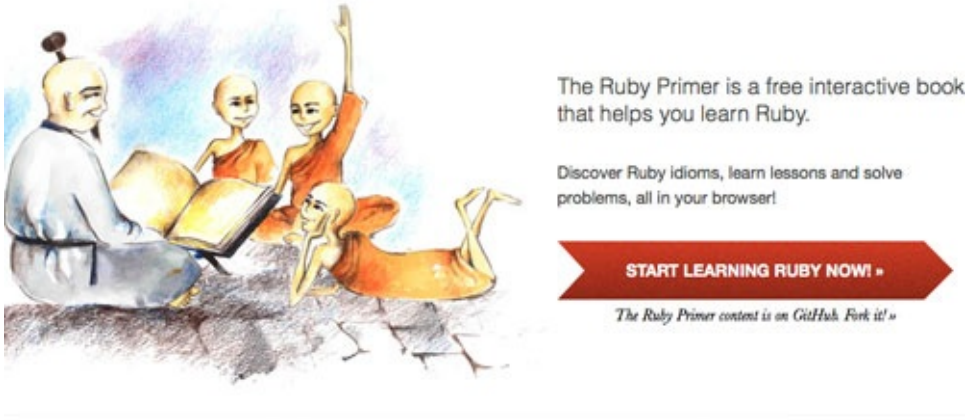
这是一个付费会员在线学习服务，完全覆盖了Web设计和开发等主题，也包括目前非常流行的iOS开发。



RubyMonk

语言：Ruby

RubyMonk是一个互动的学习平台。你可以通过课程、问题解决或者相关文章来学习Ruby。



Hackety

语言：Ruby

Hackety将会教你掌握绝对的编程基础知识，即使你没有任何编程经验。

jQuery Air

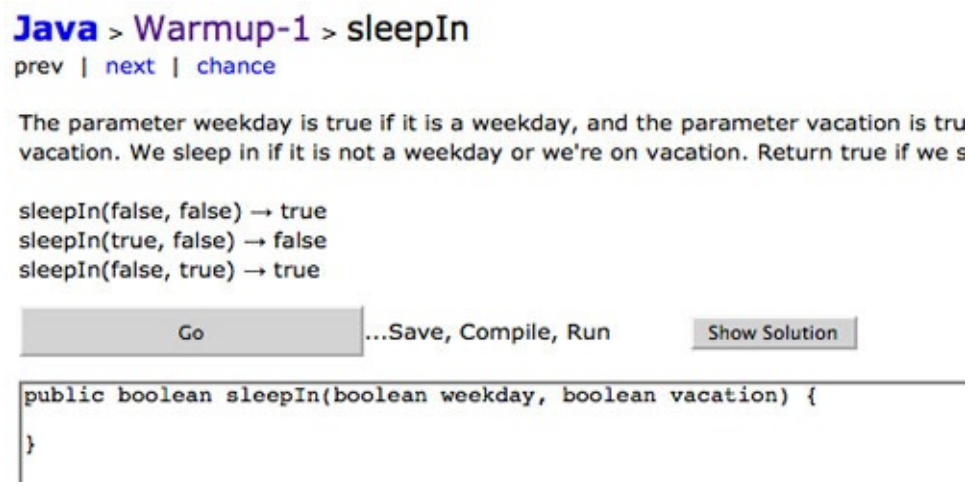
语言：jQuery

jQuery是最著名的JavaScript框架。现在，你可以直接在浏览器中学习jQuery。开始学习jQuery的过程应该是充满乐趣的，jQuery Air通过大量实践的方式来达到这个目的。

CodingBat

语言：Java、Python

通过解决示例问题以及在线编码实践的形式，来学习Java和Python。



PHP Know How

语言：PHP、MySQL

这并不是一个互动的学习方式，但它有一个很好的书面教程，包含了大量示例和指南，教你学习PHP和MySQL的基本知识。

PHP KnowHow
PHP in Plain English

MongoDB

语言：MongoDB

这是MongoDB的官方网站。MongoDB是一个可扩展、高性能、开源的NoSQL数据库。该网站中包含一个教程，让你通过一些命令来开始你的MongoDB学习旅程。

Contributor Name	Github
BiaoWei Zhuang	zhuangbiaowei
Riceball LEE	snowyu
Zhuang Ma	mzlogin
DukeAnt	-
myourdream	-
mieyou8mie	-
guanmu	-
zhouyilong	-
易枭寒	-
wgwang	-
杨松	-
Chunpeng Wen	wcp1231