

2018-05-29 陈皓，杨爽

2011年，我在[CoolShell](#)上发表了《程序员技术练级攻略》一文，得到了很多人的好评。并且陆续收到了一些人的反馈，说跟着这篇文章找到了不错的工作。几年过去，也收到了一些邮件，希望我把这篇文章更新一下，因为他们觉得有点落伍了。

是的，**老实说，抛开这几年技术的更新迭代不说，那篇文章写得也不算特别系统，同时标准也有点低，当时是给一个想要入门的朋友写的。所以，非常有必要从头更新一下《程序员练级攻略》这一主题。**

前言导读

升级版的《程序员练级攻略》会比Coolshell上的内容更多，也更专业。这篇文章有【入门篇】、【修养篇】、【专业基础篇】、【软件设计篇】、【高手成长篇】五大篇章。它们会帮助你从零开始，一步步地，系统地，完成从陌生到熟悉，到理解掌握，从编码到设计再到架构，从码农到程序员再到工程师再到架构师的进阶，实现从普通到精通到卓越的完美转身……

在**入门篇**中，我先推荐的是Python和JavaScript作为入门语言，并给出了相应的学习资源和方法。Python语法比较简单，有大量的库和语法糖，是零基础的人学习编程的不二之选。而JavaScript是前端语言，更容易让你获得编程的成就感。

随后，我们需要学习使用操作系统Linux、编程工具Visual Studio Code等入门必学内容。Web互联网作为第三次工业革命信息化浪潮中最大的发明，也是每个程序员都不能错过的。而学习编程还是要多多动手，因此我给出了Web编程入门的学习要点，并给出了一个实践项目，帮助你理解和巩固所学的内容。

如果你跟着我的这个教程走过来，并能自己去解决遇到的问题，那么，我相信你能够做一点东西了，而且你还可能会对编程非常感兴趣了。但是你千万不要以为自己已经入门了。我只是用这些内容给你一些成就感，并激发你持续学习兴趣的。

正式入门，我推荐的语言是Java，因为我认为，它是所有语言里综合实力最强的。随后，推荐了更为专业实用的编程工具，如编程的IDE、版本管理工具Git、调试前端程序和数据库设计工具等，并且给出了一个实践项目。我同时设置了业务和技术两方面的需求，跟着做一遍，相信你对学习编程会有更多的理解和感悟。

接下来，我要带你进入更为专业更为复杂的编程世界中。进入之前，我们需要树立正确的三观和心态，这对于程序员是至关重要。这就好像民工建筑队和专业的工程队的区别，就好像小作坊和工厂的差别，他们不仅仅是差别在技能和技术上，更是差别在做事的心态和三观上。

因此，在学习专业的软件开发知识之前，我们来谈谈**程序员修养**。它看似与程序员练级关系不大，实际上却能反映出程序员的工程师特质和价值观，决定了这条路你到底能走多远，是精髓所在。**有修养的程序员才可能成长为真正的工程师和架构师，而没有修养的程序员只能沦为码农，这是码农和工程师的关键区分点。**

在“修养篇”，我给出了一些相关的学习资料，并指出了我认为比较重要的几个方面：英文能力、问问题的能力、写代码的修养、安全防范意识、软件工程和上线规范、编程规范等。这些能力的训练和培养将为后续的学习和发展夯实基础。

此时，相信你已经迫不及待地想进入**专业基础篇**了。这部分内容主要涵盖编程语言、理论学科和系统知识三方面知识。在编程语言方面，推荐学习C、C++和Java这三个工业级的编程语言。理论学科方面，需要学习算法、数据结构、网络模型、计算机原理等内容。系统知识方面会讲述Unix/Linux、TCP/IP、C10K挑战等专业的系统知识。最后给出了你可以选择的几大从业方向。

- 如果对你操作系統、文件系統、数据库、网络等比较感兴趣，那么可以考虑从事底层方面的工作。
- 如果对分布式系統架构、微服务、DevOps、Cloud Native等有热情，那么可以从事架构方面的工作。
- 如果是对大数据、机器学习、人工智能等比较关注，那么数据领域可以成为你一展身手的地方。
- 如果对你用户体验或者交互等更感兴趣，那么前端工程师也是个不错的选择。
- 此外，安全开发、运维开发、嵌入式开发等几大方向，也为你提供了丰富多彩的发展空间。

以我之见，该如何选择应该完全遵从于你的本心，你更愿意在哪个领域里持续奋斗和学习。这个答案，在你的手中，在你的心中。**这里我只想和你说两观点：各种技术方向不是鱼和熊掌，是可以兼得的；很多技术是相通的，关键是你是学在表面还是深入本质。**

软件设计能力是每个程序员都需要具备的基本素质。我结合各主流语言讲述了泛型编程、函数式编程、面向对象编程等多种编程范式，分享了DRY-避免重复原则、KISS-简单原则、迪米特法则（又称“最少知识原则”）、面向对象的S.O.L.I.D原则等等多个经典的软件设计原则。

同时，给出了软件设计领域的一些重要的学习资料。**软件设计是工程师非常重要的能力，这里描述了软件工程自发展以来的各种设计方法，这是从工程师通往架构师的必备技能。**

登峰造极，是每个武林高手都渴望达到的境界，对于每个有理想有追求的程序员也是如此。因此，我特意在《程序员练级攻略（2018）》这一系列内容的最后设置了**高手成长篇**。

相较前面的内容，这部分内容相当全面和丰富，涵盖系统、数据库、分布式架构、微服务、容器化和自动化运维、机器学习、前端方向和技术论文等几方面内容，而且深度一下子拔高了好几个数量级。

同时，这也是我留给你的再一次做选择的机会，平凡还是卓越？自在悠闲，还是猛啃书本，“不破楼兰终不还”？还是遵循你内心的选择吧。偷偷地告诉你，我选的是后者。

你应该不难看出这一系列文章比我在CoolShell上的那一篇更为专业，标准也会更高，当然，难度也会更大。但是，也会让你有更坚固的技术基础，并能有更高更广泛的提高。

通过这一系列文章，我主要想回答以下几个问题。

- **理论和现实的差距。**你是否觉得自己从学校毕业的时候只做过小玩具一样的程序？走入职场后哪怕没有什么经验也可以把文中提到的这些课外练习走一遍。学校课程总是从理论出发，作业项目都看不出有什么实际作用，到了工作上发现自己什么也不会干。
- **技术能力的瓶颈。**你又是否觉得，在工作当中需要的技术只不过是不断地堆业务功能，完全没有什么技术含量。而你工作一段时间后，自己都感觉非常地迷茫和彷徨，感觉到了提高的瓶颈，完全不知道怎么提升了。
- **技术太多学不过来。**你是否又觉得，要学的技术多得都不行了，完全不知道怎么学？感觉完全跟不上。有没有什么速成的方法？

对此，我有如下的一些解释，以端正一下你的态度。

- 并不是理论和现实的差距大，而是你还没有找到相关的场景，来感受到那些学院派的知识的强大威力。算法与数据结构、操作系统原理、编译原理、数据库原理、计算机原理……这些原理上的东西，是你想要成为一个专家必需要学的东西。**这就是“工人”和“工程师”的差别，是“建筑工人”和“建筑架构师”的差别。**如果你觉得这些理论上的东西无用，那么只能说明，你只不过在从事工人的工作，而不是工程师的工作。
- **技术能力的瓶颈，以及技术太多学不过来，只不过是你自己的能力不足或是懒惰找的借口罢了。**技术的东西都是死的，这些死的知识只要努力就是可以学会的。只不过聪明的人花得时间少，笨点的人花得时间多点罢了。这其中的时间差距主要是由学习方法的不同，基础知识储备的不同造成的。只要你的方法得当，多花点时间在基础知识上，会让你未来学习应用知识的时间大大缩短。**以绝大多数人努力的程度，和为自己不努力找借口的程度为参考，只要你坚持正常的学习就可以超过大多数人了。**
- **这里没有学习技术的速成的方法，真正的牛人不是能够培训出来的，一切都是要靠你自己去努力和持续地付出。**如果你觉得自己不是一个能坚持的人，也不是一个想努力的人，而是一个想找捷径的人，那么，这篇文章并不适合你。**这篇文章中的成长路径是需要思考、精力和相关的经验的，这都需要时间，而且是不短的时间。你先问问自己有没有花十年磨一剑的决心，如果没有，那这篇文章对你没有任何作用。**

这里有一篇传世之文《[Teach Yourself Programming in Ten Years](#)》（[中英对照版](#)）。还有在我Cooshell上的这篇《[程序员的荒谬之言还是至理名言？](#)》。

我希望你在学习编程前先读一读这两篇文章。如果你觉得可以坚持的话，那么，我这一系列文章会对你很有帮助。否则，我相信你只要大致浏览一下目录及其中的某些章节，就会选择放弃走这条路的。是的，这个系列的文内容也会让一些想入行但又不愿意付出努力的同学早点放弃。

最后，给出我的几点**学习建议**。

- 一定要坚持，要保持长时间学习，甚至终生学习的态度。
- 一定要动手，不管例子多么简单，建议至少自己动手敲一遍看看是否理解了里头的细枝末节。
- 一定要学会思考，思考为什么要这样，而不是那样。还要举一反三地思考。
- 不要乱买书，不要乱追新技术新名词，基础的东西经过很长时间积累，会在未来至少10年通用。
- 回顾一下历史，看看历史时间线上技术的发展，你才能明白明天会是什么样的。

另外，这篇文章的标准会非常高。希望不会把你吓坏了。《易经》有云：“**取法其上，得乎其中，取法其中，得乎其下，取法其下，法不得也**”。所以，我这里会给你立个比较高标准，你要努力达到。相信我，就算是达不到，也会比你一开始期望的要高很多……

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 -[.....](#)



程序员练级攻略（2018）：零基础启蒙

2018-05-31 陈皓

如果你从来没有接触过程序语言，这里给你两个方面的教程，一个偏后端，一个偏前端。对从零基础开始的人来说，最重要的是能够对编程有兴趣，而要对编程有兴趣，要有成就感。而成就感来自用程序打造东西，所以，我推荐下面这份不错的入门教程。

第一份入门教程，主要是让你体会编程是什么。

- 与孩子一起学编程 这本书以Python语言教你如何写程序，是一本老少咸宜的编程书。其中会教你编一些小游戏，还会和你讲基本的编程知识，相当不错。
- 两个在线编程入门的网站：[Codecademy: Learn Python](#) 和 [People Can Program](#)，你也可以在这两个网站上学习Python，只不过是英文的。

然后，你可以到[CodeAbbey](#)上去做一些在线编程的小练习。

第二份入门教程，主要是让你做点实际有用的东西。嗯，做个网页吧。

- MDN的 Web开发入门 MDN全称是Mozilla Developer Network，你可以认为是Web方面的官方技术网站。这个教程会带着你建立一个网站。然后，你可以把你的网页发布在GitHub上。

这两份教程都很不错，都是从零开始，带着你从环境准备开发，一点一点地从一些简单而有意的东西入手，让你感觉一下编程世界是什么样的，相信这两个教程可以让零基础的你喜欢上编程。

编程入门

在这时，我们使用Python和JavaScript作为入门语言。Python就不用多说了，语法比较简单，有大量的库和语法糖，是零基础的人学习编程的不二之选。而JavaScript则是前端的语言，为了让你更有编程的成就感，所以，这也成了一门要学习的语言。（注意：对于计算机专业的学生来说，一般会使用Pascal做为入门的编程语言，但我觉得编程入门还是要以培养兴趣为主，所以，还是选一些能让人有成就感的语言会更好）。

入门语言Python

如果你想更为系统地学习一下Python编程，我强烈推荐你阅读一下下面这两本书。它们是从零基础入门非常非常不错的书，里面有大量的更为实用的示例和项目，可以让你有会编程会有超能力的感觉。

- Python编程快速上手
- Python编程：从入门到实践

这两本书除了编程语法方面的讲述有所不同之外，其他都差不多，主要是通过书中的示例来强化你对编程的学习。第一本偏文本处理一些，包括处理Word、Excel和PDF，第二本中有一些Web项目和代码部署方面的内容。如果可能的话，你可以把两本书中的示例都跑一遍。如果你时间有限的话，我推荐你看第二本。

编辑插入：极客时间上也有Python入门的视频课程。

入门语言JavaScript

如果想入门学习JavaScript，我主要推荐以下在线教程。

- MDN JavaScript教程 你可以认为这是最权威的JavaScript官方教程了，从初级到中级再到高级，就是讲语言方面。非常全。
- W3School JavaScript教程 W3School的这个教程比较偏Web方面的编程。
- JavaScript全线教程（廖雪峰） 这是廖雪峰的一个比较偏应用的教程，也是偏Web方面的编程，同时包括涉及后端的Node.js方面的教程。

操作系统入门Linux

学习编程你还需要会玩Linux，虽然Windows占据着更多的桌面市场，但是你还是要了解Linux。这里，你可以看一下，W3School上的在线教程[Linux教程](#)。

编程工具Visual Studio Code

这里主要推荐时下最流行也是最好用的Visual Studio Code，这个工具潜力十足，用其开发Python、JavaScript、Java、Go、C/C++都能得心应手（[教程](#) - 本文发布时还基本没怎么翻译完）。

Web编程入门

如果玩到这里，你觉得有趣的话，可以学习一下Web方面的入门知识。**为什么是Web而不是别的什么呢？因为你正身处于第三次工业革命的信息化浪潮中，在这个浪潮中，Web互联网是其中最大的发明，所以，这是任何一个程序员都不能错过的。**

关于Web编程，有下面几个方向你要学习一下。

- 前端基础。要系统地学习一下前端的知识，也就是CSS、HTML和JavaScript这三个东西。这里还是给出MDN的相关技术文档页面[CSS文档](#) 和 [HTML文档](#)。文档很大，你要学习的并不是所有的东西，而是了解CSS和HTML是怎么相互作用来展示数据的，然后，不用记忆文档中的内容，这两个文档是用来查找知识的。另外，你可以简单地学习使用JavaScript操纵HTML元件。理解DOM和动态网页（可以参看[W3Schools的JavaScript HTML DOM的教程](#)）。
- 后端基础。如果你想省点事，不想再学一门新的语言了，那么你可以直接用Python或者Node.js，这两个技术在前面提到的廖雪峰的那个教程里提到过。当然，如果你想试试另外一种脚本型的也是比较主流的编程语言，那么可以搞搞PHP，它也是很快就可以上手的语言。学习PHP语言，你可以先跟着[W3School的PHP教程](#) 玩玩（其中有连接数据库的MySQL的教程）。然后，以[PHP的官网文档](#) 作为更全的文档来学习或查找相关的技术细节。

下面是一些学习要点。

- 学习HTML基本语法。
- 学习CSS如何选中HTML元素并应用一些基本样式。
- 学会用 Firefox + Firebug 或 Chrome 查看你觉得很炫的网页结构，并动态修改。
- 在一台Linux机器上配置LEMP - Ubuntu/Nginx/PHP/MySQL这个环境。
- 学习PHP，让后台PHP和前台HTML进行数据交互，对服务器相应浏览器请求形成初步认识。实现一个表单提交和反显的功能。
- 把PHP连接本地或者远程数据库 MySQL（MySQL 和 SQL现学现用够了）。

这里，你可能会问我，入门时有三个后端语言，一个是Python，一个是Node.js，一个是PHP，你对这三门语言怎么看？老实说，Python我还看好一些，PHP次之，Node.js最后。原因是：

- Python语言的应用面还是很多的。（当然，性能可能会有一些问题，但是用于一些性能不敏感的和运维或是一些小工具相关的，还是非常好用的。另外，Python的应用场景其实还是很多的，包括机器学习和AI也有Python的身影。用Python来做一些爬虫、简单的中间件、应用或是业务服务也是很不错的。）
- PHP也是一个比较主流的简单的语言（PHP在目前来说还是一个比较主流的语言，但其发展潜力有限，虽然可以让你找得到工作，但是一般玩玩就行了）。
- Node.js 号称 JavaScript 的后端版，但从目前发展来说，在后端的世界里，并不能承担大任，而且问题很多。一些前端程序员用它来做后端的粘合层，我个人觉得这样做只是掩盖前后端配合有问题，或是接口设计思维上的懒惰，我还是希望前端程序员应该认真学习一门真正的后端语言。

当然，这里只是让你感觉一下，Web前端编程的感觉，只是为了入门而已。所以，对于这些语言你也不用学得特别精通，感觉一下这几个不同的语言就可以了，然后知道相关的文档和知识在哪里，这样有助于你查阅相应的知识点。

实践项目

无论你用Python，还是Node.js，还是PHP，我希望你能做一个非常简单的Blog系统，或是BBS系统，需要支持如下功能：

- 用户登录和注册（不需密码找回）。
- 用户发贴（不需要支持富文本，只需要支持纯文本）。
- 用户评论（不需要支持富文本，只需要支持纯文本）。

你需要从前端一直做到后端，也就是说，从HTML/CSS/JavaScript，到后面的PHP（Python/Node.js），再到数据库。这其中需要你查阅很多的知识。

这里有几个技术点你需要关注一下。

1. 用户登录时的密码不应该保存为明文，应该用MD5+Salt来保存（关于这个是什么，希望你能自行Google）

1. 用户登录时的密码不能使用明文，应该用MD5+Salt来保存（关于这个是什么，希望你能自行Google一下）。
2. 用户登录后，对于用户自己的帖子可以有“重新编辑”或“删除”的功能，但是无权编辑或删除其它用户的帖子。
3. 数据库的设计，你需要三张表：用户表、文章表和评论表，它们之间是怎么关联的，你需要学习一下。这里有个PHP的blog教你怎么建表，你可以[前往一读](#)。

如果你有兴趣，你可以顺着这个小项目，研究一下下面这几个事。

- 图片验证码。
- 上传图片。
- 阻止用户在发文章或评论时输入带HTML或JavaScript的内容。
- 防范SQL注入。参看[PHP官方文档](#) 或 [微软官方文档](#)，或者你自己Google一下。

上面这些东西，不是什么高深的东西，但是可以让你从中学到很多东西。相信你只需要自己Google一下就能搞定。

小结

总结一下今天所说学的内容。我先推荐Python和JavaScript作为入门语言，以让你尽快上手，获得成就感，从而激发你想持续学习的热情。随后介绍了Linux操作系统、Visual Studio Code编程工具、Web编程入门等方面的学习资料，并给出了学习要点。最后，我给出了一个实践项目，帮助你理解和巩固今天所学的内容。

消化好了今天的内容，就准备好精力迎接后面的挑战吧。下篇文章中，我们将正式入门学习该如何编程。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 -



学习了前面文章中的入门级经验和知识后，你可能会有两种反应。

- 一种反应可能是，你对编程有一点的兴趣了，甚至有一点点小骄傲，可能还会四处炫耀。我想说，请保持这种感觉，但是你也要清醒一下，上面的那些东西，还不算真正的入门，你只是入门了一条腿。
- 另一种反应也可能是，你被吓着了，觉得太难了。感觉不是一般人能玩的，如果是这样的话，我想鼓励你一下—“**无论你做什么事，你都会面对各式各样的困难，这对每个人都是一样的，而只有兴趣、热情和成就感才能让你不怕这些困难**”。所以，你问问你自己，是否从中收获了成就感或是兴趣，如果没有的话，可能这条路并不适合你。如果有的话，哪怕一丁点儿，你也应该继续坚持下来。

这篇文章，我主要是让你成为更为专业的入门程序员。请注意，此时，你可能需要读一些比较枯燥的书，但我想说，这些是非常非常重要的。你一定要坚持住。

编程技能

在系统地学习编程技能之前，我希望你能先看一下“[The Key To Accelerating Your Coding Skills](#)”，这篇文章会告诉你如何有效地快速提高自己的编程能力。

然后接下来是下面几大块内容，但还只是入门级的。

- **编程技巧方面** - 你可以开始看怎么把程序写好的书了，这里推荐的是《[代码大全](#)》。这本书好多年没有更新了，其中有一些内容可能有点过时，但还是一本非常好的书，有点厚，你不需要马上就看完。在你的编程路上，这本书可以陪你走很久，因为当你有更多的编程经验时，走过更多的坑后，再把这本书拿出来看看，你会有更多的体会。**好的书和不好的书最大的不一样的是，好的书在你不同的阶段来读，你会有不同的收获，而且还会产生更多的深层次的思考！** 《代码大全》就是这样的一本书。
- **编程语言方面** - 这个阶段，你可以开始了解一下Java语言了，我个人觉得Java是世界上目前为止综合排名最好的语言。你一定要学好这门语言。推荐《[Java核心技术（卷1）](#)》，除了让你了解Java的语法，它还会让你了解面向对象编程是个什么概念（如果你觉得这本书有点深，那么，你可以降低难度看更为基础的《[Head First Java](#)》）。然后，既然开始学习Java了，那就一定要学Spring，推荐看看《[Spring in Action](#)》或是直接从最新的Spring Boot开始，推荐看看《[Spring Boot 实战](#)》。关于Spring的这两本书，里面可能会有很多你从来没有听说过的东西，比如，IoC和AOP之类的东西，能看懂多少就看懂多少，没事儿。
- **操作系统** - 这里你可以看看《[鸟哥的Linux私房菜](#)》，这本书会让你对计算机和操作系统，以及Linux有一个非常全面的了解，并能够管理或是操作好一个Linux系统。当然，这本书有很多比较专业的知识，你可能会看不懂，没关系，就暂时略过就好了。这本书的确并不适合初学者，你能看多少就看多少吧。
- **网络协议** - 你需要系统地了解一下HTTP协议，请到MDN阅读一下其官方的[HTTP的文档](#)。你需要知道HTTP协议的几个关键点：1) HTTP头，2) HTTP的请求方法，3) HTTP的返回码。还有，HTTP的Cookie、缓存、会话，以及链接管理，等等，在MDN的这个文档中都有了。对于HTTP协议，你不需要知道所有的东西，你只需要了解这个协议的最关键的那些东西就好了。
- **数据库设计** - 你需要系统地了解一下数据库设计中的那些东西，这里推荐慕课网的一个在线课程—“[数据库设计的那些事](#)”。每个小课程不过5-6分钟，全部不到2个小时，我相信你一定能跟下来。你需要搞清楚数据的那几个范式，还有SQL语句的一些用法。当然，你还要学习和使用一下数据库，这里推荐学习开源的MySQL。你可以看官方文档，也可以看一下这本书《[MySQL必知必会](#)》。
- **前端方面** - 前端的东西不算复杂，你需要学习几个东西。一个是和JavaScript相关的[jQuery](#)，另一个是和CSS相关的[Bootstrap](#)，学习这两个东西都不复杂，直接上其官网看文档就好了。最重要的是，你要学习一下，JavaScript的Ajax请求后端的API接口的方式，而不是再像前面那样用后端来向前端返回HTML页面的形式了。这里，你需要学习一下，JavaScript的Promise模式。[阮一峰翻译的ES6的教程中有相关的内容](#)。当然，你Google一下，也可以找到一堆学习资料。
- **字符编码方面** - 在你处理中文时有时会发现有乱码出现，此时需要了解ASCII和Unicode这样的字符编码。这里推荐一篇文章—“[关于字符编码，你所需要知道的（ASCII, Unicode, Utf-8, GB2312...）](#)”或是英文文章“[The history of Character Encoding](#)”以及[Wikipedia - Character encoding](#)。还有GitHub上的这两个Awesome仓库：[Awesome Unicode](#) 和 [Awesome Code Points](#)。

为什么转成Java语言？

相信你可能会问，为什么之前学习的Python和JavaScript不接着学，而是直接切到Java语言上来，这样会不会切得太快了。这是一个好问题，这里需要说明一下，为什么我会切到Java这个语言上来，主要是有以下几方面考虑。

1. Java是所有语言里面综合实力最强的，这也是为什么几乎所有大型的互联网或是分布式架构基本上都是Java技术栈。所以，这是一个工业级的编程语言（Python和JavaScript还达不到这样的水准）。
2. 之所以，没有用Java来做入门语言而是用了Python，这是因为编程是一件比较费脑子的事，一开始学习时，兴趣的培养很重要。Python比较简单，容易上手，能够比较容易地提起兴趣，而用Java则可能比较难。
3. 在你有了些编程语言的基础后，有了些代码的逻辑后，切到工业级的编程语言上来，更为专业地学习编程，是非常有帮助的。像Python和JavaScript这样的动态语言用着是很爽，但是，只有像C、C++和Java这样的静态语言才可以让你真正地进阶。
4. 对于一个合格的程序员，掌握几门语言是非常正常的事情。一方面，这会让你对不同的语言进行比较，让你有更多的思考。另一方面，这也是一种学习能力的培养，会让你对于未来的新技术学习得更快。很多时候，一些程序员只在自己熟悉的技术而不是合适的技术上工作，这其实并不好，这会让你的视野受限，而视野会决定你的高度。综上所述，这就是在入门的时候我故意让你多学几门语言的原因。

编程工具

编程工具方面，你需要开始学习使用下面这些工具了。

- **编程的IDE。** 传统一点的，你可以使用Eclipse ([教程](#))。当然，我推荐你使用IntelliJ IDEA ([教程](#))。这两个工具都可以开发各种语言，但是主要用在Java。如果想玩得更时髦一些的话，使用Visual Studio Code也不错，这个工具潜力十足，用其开发Python、JavaScript、Java、Go、C和C++都能得心应手 ([教程](#) - 本文发布时还基本没怎么翻译完)。
- **版本管理工具。** 版本管理工具是非常重要的编程工具。传统的有P4、SVN、CVS等，但都会被Git取代，所以，你就只用学习Git就好了。学习Git的教程网上有很多，这里我推荐非常系统的[Pro Git 第二版](#)（如果你觉得Pro Git比较枯燥的话，备选[猴子都能懂的Git入门](#)），然后你要学会使用GitHub。关于一些Git环境安装和准备以及GitHub使用，你可以自行Google（比如：这篇[Github and Git 图文教程](#) 或是这篇[Git图文教程及详解](#)）。
- **调试前端程序。** 你需要学会使用Chrome调试前端程序，Google一下会有很多文章，你可以看看[超完整的Chrome浏览器客户端调试大全](#)。
- **数据库设计工具。** 你需要学会使用MySQL WorkBench，这个工具很容易使用。相关的手册，你可以看一下[官方文档](#)。

实践项目

这回我们需要设计一个投票系统的项目。

业务上的需求如下：

- 用户只有在登录后，才可以生成投票表单。
- 投票项可以单选，可以多选。
- 其它用户投票后显示当前投票结果（但是不能刷票）。
- 投票有相应的时间，页面上需要出现倒计时。
- 投票结果需要用不同颜色不同长度的横条，并显示百分比和人数。

技术上的需求如下：

- 这回要用Java Spring Boot来实现了，然后，后端不返回任何的HTML，只返回JSON数据给前端。
- 由前端的jQuery来处理并操作相关的HTML动态生成在前端展示的页面。
- 前端的页面还是要响应式的，也就是可以在手机端和电脑端有不同的呈现。这个可以用Bootstrap来完成。

如果你有兴趣，还可以挑战以下这些功能。

- 在微信中，通过微信授权后记录用户信息，以防止刷票。

- 可以不用刷页面，就可以动态地看到投票结果的变化。
- Google一些画图表的JavaScript库，然后把图片弄得风骚一些。

小结

上面那些书和知识你要看完，还要能理解并掌握，我估计你最少也要花1-2年左右的时间。如果你能够走到这里，把前面的那些知识都了解了，不用精通，能独立地做出上面的那些实践项目，那么，你就算是真正的入门了。

而且，你已经是一个“全栈工程师”的样子了，在这里我要给你一个大大的赞。如果这个时候，你对编程还有很大的热情，那么我要恭喜你了，你可能会是一个非常不错的程序员。加油啊！

上面的那些技术已经算是比较专业的了。如果你已经大致掌握了，我相信你可以找到至少年薪20万以上的工作了，而且你的知识面算是有不错的广度了。但是深度还不够，这个时候，是一个比较关键点了。

你可能已经沉醉在沾沾自喜的骄傲的情绪中，那么你也可以就此止步，加入一些公司，在那里按部就班地完成一些功能性的开发，成为一个搬砖的码农。你也可以开始选择一个方向开始深入。

我给你的建议是选择一个方向开始深入。**因为你并不知道你未来会有多大的可能性，也不知道你会成为什么样的人，所以为什么不再更努力一把呢？**

后面，我们就开始非常专业的程序员之路了。这也是一般程序员和高级程序员的分水岭了，能不能过去就看你的了。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 -



程序员练级攻略（2018）：程序员修养

2018-06-07 陈皓

在完成上述的入门知识学习之后，我们要向专业的计算机软件开发进军了。但是在学习那些专业的知识前，我们先要抽一部分的篇幅来说一下程序员的修养。这是程序员的工程师文化，也就是程序员的价值观，因为如果你的技术修养不够的话，你学再多的知识也是没有用的。

要了解程序员的修养，你可以先从Quora上的这个帖子开始“What are some of the most basic things every programmer should know?”，我摘录一些在这里供你参考。

- Bad architecture causes more problems than bad code.
- You will spend more time thinking than coding.
- The best programmers are always building things.
- There's always a better way.
- Code reviews by your peers will make all of you better.
- Fewer features for better code is always the right answer in the end.
- If it's not tested, it doesn't work.
- Don't reinvent the wheel, library code is there to help.
- Code that's hard to understand is hard to maintain.
- Code that's hard to maintain is next to useless.
- Always know how your business makes money, that determines who gets paid what.
- If you want to feel important as a software developer, work at a tech company.

然后是《97 Things Every Programmer Should Know》，其中有97个非常不错的编程方面的建议。这篇文章是比较经典的，别被“97”吓住，你可以快速浏览一下，会让你有不同的感觉。另外，在工作一段时间后再来读，你会更有感觉。

英文能力

必须指出，再往下走，有一个技能非常重要，那就是英文。如果对这个技能发怵的话，那么你可能无缘成为一个程序员高手了。因为我们所有的计算机技术全部来自于西方国家，所以如果你要想成为一个高手的话，那么必须到信息的源头去。英文的世界真是有价值的信息的集散地。你可以在那里，到官网上直接阅读手册，到StackOverflow上问问题，到YouTube上看很多演讲和教学，到GitHub上参与社区，用Google查询相关的知识，到国际名校上参加公开课……

如果你的英文能力有问题的话，那么基本上来说，你无法成为一个高手。因此，学好英文是非常有必要的，我说的不只是读写，还有听和说。相信你在学校里学过英文，有一定的基础。所以，我给你下面的这些建议。

1. 坚持Google英文关键词，而不是在Google里搜中文。
2. 在GitHub上只用英文。用英文写代码注释，写Code Commit信息，用英文写Issue和Pull Request，以及用英文写Wiki。
3. 坚持到YouTube上每天看5分钟的视频。YouTube上有相关的机器字幕，实在不行就打开字幕。
4. 坚持用英文词典而不是中文的。比如：[剑桥英语词典](#) 或是 [Dictionary.com](#)。你可以安装一个Chrome插件 [Google Dictionary](#)。
5. 坚持用英文的教材而不是中文的。比如：[BBC 的 Learning English](#)，或是到一些ESL网站上看看，如 [ESL: English as a Second Language](#) 上有一些课程。
6. 花钱参加一些线上的英文课程，用视频和老外练习。

问问题的能力

提问的智慧 ([How To Ask Questions The Smart Way](#)) 一文最早是由Eric Steven Raymond所撰写的，详细描述了发问者事前应该做好什么，而什么又是不该做的。作者认为这样能让问题容易令人理解，而且发问者自己也能学到较多东西。

此文一经发出，就广受好评，被广泛转载并奉为经典。该文也有[简体中文翻译版](#)被流传着，所以在华人界也是篇很有名的文章。有两个著名的缩写STFW (Search the fxxking web) 以及RTFM (Read the fxxking manual) 就是出自本文。

另外，还有一个经典的问题叫 [X-Y Problem](#)。对我来说，这是一个很容易犯的错误，所以，你也要小心避免（我曾经在我的Coolshell上写过这个事《[X-Y问题](#)》）。

然后，你可以到StackOverflow上看看如何问问题的一些提示-- “[FAQ for StackExchange Site](#)”。

作为一个程序员，不做伸手党，你必需要读一读这几篇文章，并努力践行。

写代码的修养

除了《代码大全》外，你还需要补充一些如何写好代码的知识，有以下几本书推荐。

- [《重构：改善既有代码的设计》](#)，这本书是Martin Fowler的经典之作。这本书的意义不仅仅在于“改善既有代码的设计”，也指导了我们如何从零开始构建代码的时候避免不良的代码风格。这是一本程序员必读的书。
- [《修改代码的艺术》](#)，这本书是继《重构》之后探讨修改代码技术的又一里程碑式的著作，而且从涵盖面和深度上都超过了前两部经典（《代码大全》和《重构》）。作者将理解、测试和修改代码的原理、技术和最新工具（自动化重构工具、单元测试框架、仿对象、集成测试框架等），与解依赖技术和大量开发和设计优秀代码的原则、最佳实践相结合，许多内容非常深入。这本书可以让你不仅能够掌握最顶尖的修改代码技术，还可以大大提高对代码和软件开发的领悟力。
- [《代码整洁之道》](#)，这本书提出一种观念：代码质量与其整洁度成正比。干净的代码，既在质量上较为可靠，也为后期维护和升级奠定了良好基础。本书作者给出了一系列行之有效的整洁代码操作实践。这些实践在本书中体现为一条条规则（或称“启示”），并辅以来自现实项目正反两面的范例。
- [《程序员的职业素养》](#)，这本书是编程大师Bob大叔40余年编程生涯的心得体会，讲解成为真正专业的程序员需要什么样的态度、原则，需要采取什么样的行动。作者以自己以及身边的同事走过的弯路、犯过的错误为例，意在为后来人引路，助其职业生涯迈上更高台阶。

另外，作为一个程序员，Code Review是非常重要的程序员修养。Code Review对我的成长非常有帮助，我认为没有Code Review的公司都没有必要呆（因为不做Code Review的公司一定不尊重技术的）。下面有几篇我觉得还不错的Code Review的文章，供你参考。

- [Code Review Best Practices](#)
- [How Google Does Code Review](#)
- [LinkedIn's Tips for Highly Effective Code Review](#)

除了Code Review之外，Unit Test也是程序员的一个很重要的修养。写Unit Test的框架一般来说都是从JUnit衍生出来的，比如CppUnit之类的。学习JUnit使用的最好方式就是到其官网上看 [JUnit User Guide \(中文版\)](#)。然后，有几篇文章你可以看看（也可以自行Google）：

- [You Still Don't Know How to Do Unit Testing](#)
- [Unit Testing Best Practices: JUnit Reference Guide](#)
- [JUnit Best Practices](#)

安全防范

在代码中没有最基本的安全漏洞问题，也是我们程序员必需要保证的重要大事，尤其是对外暴露Web服务的软件，其安全性就更为重要了。对于在Web上经常出现的安全问题，有必要介绍一下 [OWASP - Open Web Application Security Project](#)。

OWASP是一个开源的、非盈利的全球性安全组织，致力于应用软件的安全研究。其被视为Web应用安全领域的权威参考。2009年，国际信用卡数据安全技术PCI标准将其列为必要组件，美国国防信息系统局、欧洲网络与信息安全局、美国国家安全局等政府机构所发布的美国国家和国际立法、标准、准则和行业实务守则参考引用了OWASP。

美国联邦贸易委员会（FTC）强烈建议所有企业需遵循OWASP十大Web弱点防护守则。所以，对于https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project OWASP Top 10项目是程序员非常需要关注的最基本的是最严重的安全问题，现在其已经成了一种标准，这里是其中文版《[OWASP Top 10 2017 PDF 中文版](#)》。

下面是安全编程方面的一些Guideline。

- [伯克立大学的Secure Coding Practice Guidelines](#)。
- [卡内基梅隆大学的 SEI CERT Coding Standards](#)。

此外，有一篇和HTTP相关的安全文章也是每个程序员必需要读的——《[Hardening Your HTTP Security Headers](#)》。

最后想说的是“防御性编程”，英文叫[Defensive Programming](#)，它是为了保证对程序的不可预见的使用，不会造成程序功能上的损坏。它可以被看作是为了减少或消除墨菲定律效力的想法。防御式编程主要用于可能被滥用，恶作剧或无意地造成灾难性影响的程序上。下面是一些文章。

- [The Art of Defensive Programming](#)。
- 当然，也别过渡了，这篇文章可以看看，[Overly defensive programming](#)。

软件工程和上线

系统上线是一件比较严肃的事，这表明你写的软件不是跑在自己的机器上的玩具，或是实验室里的实验品，而是交互给用户使用的，甚至是用户付费的软件。对于这样的软件或系统，我们需要遵守一些上线规范，比如，需要认真测试，并做上线前检查，以及上线后监控。下面是几个简单的规范，供你参考。

- 关于测试，推荐两本书。
 - 《[完美软件：对软件测试的各种幻想](#)》，这本书重点讨论了与软件测试有关的各种心理问题及其表现与应对方法。作者首先阐述软件测试之所以如此困难的原因—人的思维不是完美的，而软件测试的最终目的就是发现对改善软件产品和软件开发过程有益的信息，故软件测试是一个信息获取的过程。
 - 《[Google软件测试之道](#)》，描述了测试解决方案，揭示了测试架构是如何设计、实现和运行的，介绍了软件测试工程师的角色；讲解了技术测试人员应该具有的技术技能；阐述了测试工程师在产品生命周期中的职责；讲述了测试管理，并对在Google的测试历史上或者主要产品上发挥了重要作用的工程师的访谈，这令那些试图建立类似Google的测试流程或团队的人受益很大。
- 当你的系统要上线时，你是不是已经做好上线的准备了？这里有两个Checklist供你做上线前的一些检查。
 - [Server Side checklist](#)
 - [Single Page App Checklist](#)
- 《[Monitoring 101](#)》这是一篇运维方面的入门文章，告诉你最基本的监控线上运行软件的方法和实践。

小结

好了，总结一下今天分享的主要内容。程序员修养看似与程序员练级关系不大，实际上却能反映出程序员的工程师特质和价值观，决定了这条路你到底能走多远。**有修养的程序员才可能成长为真正的工程师和架构师，而没有修养的程序员只能沦为码农。**

因此，在这篇文章中，我指出了我认为比较重要的几个方面：英文能力、问问题的能力、写代码的修养、安全防范意识、软件工程和上线规范等。这些能力的训练和培养将为后续的学习和发展夯实基础。

附录：编程规范

我们在写代码时，最好参考一些已有的最佳实践。为什么要有编程规范和最佳实践，要让所有人按一定的规范来编程呢？有下面几个主要原因。

- 可以让你的代码很规整，这有利于代码易读性，从而更容易地维护。
- 提升开发效率，我们知道，效率来自于结构化，而不是杂乱。
- 可以让你的软件避免一些容易掉坑的陷阱，也让Bug更少，质量更高。
- 可以让团队成员更高效率地协作。

如果一个程序员没有这类规范和最佳实践的沉淀，那么是很难成为真正的程序员的，只能沦为码农。

当然，对于一些代码风格方面的东西，比如左大括号是否要换行，缩进是用tab还是空格等等，我觉得没有对错，只要团队统一就好了。

下面，我罗列了一堆各种语言的编程规范，供你参考。

编程语言相关

C语言

- [NASA C Style](#)。
- [C Coding Standard](#)。
- [C Programming/Structure and style](#)。
- [Linux kernel coding style](#)。
- [GNU Coding Standard](#)，GNU的编码规范。

C++语言

- [C++ Core Guidelines](#)，这个文档是各种C++的大拿包括原作者在内持续讨论更新的和C++语言相关的各种最佳实践。
- [Google C++ Style Guide](#)。

Go语言

- [Effective Go](#)，Go的语法不复杂，所以，Go语言的最佳实践只需要看这篇官方文档就够了。

Java语言

- [Code Conventions for the Java™ Programming Language](#)，Java官方的编程规范。
- [Google Java Style Guide](#)，Google的Java编码规范。

JavaScript语言

- [JavaScript The Right Way](#)，一个相对比较容读的JavaScript编程规范，其中不但有代码规范，还有设计模式，测试工具，编程框架，游戏引擎.....
- [Google JavaScript Style Guide](#)，Google公司的JavaScript的编码规范，一个非常大而全的编程规范。
- [Airbnb JavaScript Style Guide](#)，Airbnb的JavaScript编程规范。没Google的这么大而全，但是也很丰富了。
- [jQuery Core Style Guide](#)，jQuery的代码规范。
- [JavaScript Clean Code](#)，前面推荐过的《代码整洁之道》一书中的JavaScript的实践。

还有一些其它相对比较简单的JavaScript编程规范。

- [JavaScript Style Guides And Beautifiers](#)，这是一篇推荐JavaScript编程规范的文章，你可以看看。
- [JavaScript Style Guide and Coding Conventions](#)，这是W3Schools的JavaScript。
- [Code Conventions for the JavaScript](#)。

PHP语言

- [PHP FIG](#)，PHP编码规范及标准推荐。
- [PHP The Right Way](#)，除了编码规范之外的各种PHP的最佳实践，还包括一些设计模式，安全问题，以及服务部署，Docker虚拟化以及各种资源。
- [Clean Code PHP](#)，《代码整洁之道》的PHP实践。

Python语言

- [Style Guide for Python Code](#)，Python官方的编程码规范。
- [Google Python Style Guide](#)，Google公司的Python编码规范。

- [Google Python Style Guide](#), Google公司的Python编程规范。
- [The Hitchhiker's Guide to Python](#), 这不只是Python的编程规范，还是Python资源的集散地，强烈推荐。

Ruby语言

- [Ruby Style Guide](#), Airbnb公司的Ruby编程规范。
- [Ruby Style Guide](#)。

Rust语言

- [Rust Style Guide](#)。
- [Rust Guidelines](#) 开源社区里最好的Rust编程规范。

Scala语言

- [Scala Style Guide](#), Scala官方的编程规范。
- [Databricks Scala Guide](#) - Databricks的Scala编程规范。
- [Scala Best Practices](#)。

Shell语言

- [Google Shell Style Guide](#), Google的Shell脚本编程规范。

Node.js相关

- [npm-coding-style](#)。
- [Microsoft + Node.js Guidelines](#)。
- [Node.js Style Guide](#)。

Mozilla的编程规范

- [Mozilla Coding Style Guide](#), 其中包括C、C++、Java、Python、JavaScript、Makefile和SVG等编程规范。

前端开发相关

- [CSS Guidelines](#), CSS容易学，但是不好写，这篇规范会教你如何写出一个健全的、可管理的，并可以扩展的CSS。
- [Scalable and Modular Architecture for CSS](#), 这是一本教你如何写出可扩展和模块化的CSS的电子书，非常不错。
- [Frontend Guidelines](#), 一些和HTML、CSS、JavaScript相关的最佳实践。
- [Sass Guidelines](#), Sass作为CSS的补充，其要让CSS变得更容易扩展。然而，也变得更灵活，这意味着可以被更容易滥用。这里这篇"富有主见"的规范值得你一读。
- [Airbnb CSS / Sass Styleguide](#), Airbnb的CSS/Sass规范。
- 说了Sass就不得不提LESS，这里有几篇和LESS相关的：[LESS Coding Guidelines](#)、[LESS Coding Guidelines](#)、[LESS coding standard](#)。
- [HTML Style Guide](#), 一个教你如何写出性能更高，结构更好，容易编程和扩展的HTML的规范。
- [HTML + CSS Code Guide](#), 如何写出比较灵活、耐用、可持续改进的HTML和CSS的规范。
- [CoffeeScript Style Guide](#), CoffeeScript的最佳实践和编程规范。
- [Google HTML/CSS Style Guide](#), Google的HTML/CSS的编程规范。
- [Guidelines for Responsive Web Design](#), 响应式Web设计的规范和最佳实践。
- [U.S. Web Design Standards](#), 这是美国政府网端要求的一些UI交互可视化的一些规范。

最后是一个前端开发的各种注意事项列表，非常有用。

- [Front-End Checklist](#)，一个前端开发的Checklist，其中包括HTML、CSS和JavaScript，还和图片、字体、SEO、性能相关，还包括一些和安全相关的事项，这个列表真的是太好了。

移动端相关

Kotlin

- [Coding Conventions](#)。

Objective-C语言

- [Objective-C Style guide](#), Style guide & coding conventions for Objective-C projects。
- [Google Objective-C Style Guide](#)。
- [NYTimes Objective-C Style Guide](#) , The Objective-C Style Guide used by The New York Times。

Swift语言

- [API Design Guidelines](#)。
- [Swift](#) - 一个Swift的相关编程规范的教程。
- [Swift style guide](#)。
- [Swift Style Guide](#) - LinkedIn的官方 Swift编程规范。
- [Metova's Swift style guide](#)。
- [Xmartlabs Swift Style Guide](#), Xmartlabs的 Swift编程规范。

API相关

- [HAL](#), 一个简单的API规范教程。
- [Microsoft REST API Guidelines](#), 微软的Rest API规范。
- [API Design Guide](#)。
- [RESTful API Designing guidelines-The best practices](#)。
- [JSON API - Recommendations](#), JSON相关的API的一些推荐实践。
- [API Security Checklist](#) , API的安全问题的检查列表。

开发工具相关

Markdown相关

- [Google Markdown Style Guide](#)。
- [Markdown Style Guide](#)。

JSON

- [Google JSON Style Guide](#)。
- [JSON Style Guide](#)。

- [Git Style Guide。](#)
- [Few Rules from Git Documentation。](#)

正则表达式相关

- [RegexHQ。](#)
- [Learn regex the easy way。](#)

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 -[.....](#)



为了进入专业的编程领域，我们需要认真学习以下三方面的知识。

编程语言。你需要学习C、C++和Java这三个工业级的编程语言。为什么说它们是工业级的呢？主要是，C和C++语言规范都由ISO标准化过，而且都有工业界厂商组成的标准化委员会来制定工业标准。次要原因是，它们已经在业界应用于许多重要的生产环境中。

- C语言不用多说，现今这个世界上几乎所有重要的软件都跟C有直接和间接的关系，操作系统、网络、硬件驱动等等。说得霸气一点儿，这个世界就是在C语言之上运行的。
- 而对于C++来说，现在主流的浏览器、数据库、Microsoft Office、主流的图形界面、著名的引擎等都是用C++编写的。而且，很多公司都用C++开发核心架构，如Google、腾讯、百度、阿里云等。
- 而金融电商公司则广泛地使用Java语言，因为Java的好处太多了，代码稳定性超过C和C++，生产力远超C和C++。有JVM在，可以轻松地跨平台，做代码优化，做AOP和IoC这样的高级技术。以Spring为首的由庞大的社区开发的高质量的各种轮子让你只需关注业务，是能够快速搭建企业级应用的不二之选。

此外，我推荐学习Go语言。一方面，Go语言现在很受关注，它是取代C和C++的另一门有潜力的语言。C语言太原始了，C++太复杂了，Java太高级了，所以Go语言就在这个夹缝中出现了。这门语言已经10多年了，其已成为云计算领域事实上的标准语言，尤其是在Docker/Kubernetes等项目中。Go语言社区正在不断地从Java社区移植各种Java的轮子过来，Go社区现在也很不错。

如果你要写一些PaaS层的应用，Go语言会比C和C++更好，目前和Java有一拼。而且，Go语言在国内外一些知名公司中有了一定的应用和实践，所以，是可以学习的（参看：[《Go语言、Docker 和新技术》](#)一文）。此外，Go语言语法特别简单，你有了C和C++的基础，学习Go的学习成本基本为零。

理论学科。你需要学习像算法、数据结构、网络模型、计算机原理等计算机科学专业需要学习的知识。为什么要学好这些理论上的知识呢？

- 其一，这些理论知识可以说是计算机科学这门学科最精华的知识了。说得大一点，这些是人类智慧的精华。你只要想成为高手，这些东西是你必需要掌握和学习的。
- 其二，当你在解决一些很复杂或是很难的问题时，这些基础理论知识可以帮到你很多。我过去这20年从这些基础理论知识中受益匪浅。
- 其三，这些理论知识的思维方式可以让你有触类旁通，一通百通的感觉。虽然知识比较难啃，但啃过以后，你将获益终生。

另外，你千万不要觉得在你的日常工作或是生活当中根本用不上，学了也白学，这样的思维方式千万不要有，因为这是平庸的思维方式。如果你想等我用到了再学也不晚，那么你有必要看一下这篇文章[《程序员的荒谬之言还是至理名言？》](#)。

系统知识。系统知识是理论知识的工程实践，这里面有很多很多的细节。比如像Unix/Linux、TCP/IP、C10K挑战等这样专业的系统知识。这些知识是你能不能把理论应用到实际项目当中，能不能搞定实际问题的重要知识。

当你在编程的时候，如何和系统进行交互或是获取操作系统的资源，如何进行通讯，当系统出了性能问题，当系统出了故障等，你有大量需要落地的事需要处理和解决。这个时候，这些系统知识就会变得尤为关键和重要了。

这些东西，你可以认为是计算机世界的物理世界，上层无论怎么玩，无论是Java NIO，还是Nginx，还是Node.js，它们都逃脱不掉最下层的限制。所以，你要好好学习这方面的知识。

编程语言

Java语言

学习Java语言有以下入门级的书（注意：下面一些书在入门篇中有所提及，但为了完整性，还是要在这里提一下，因为可能有朋友是跳着看的）。

- [《Java核心技术：卷1基础知识》](#)，这本书本来是Sun公司的官方用书，是一本Java的入门参考书。对于Java初学者来说，是一本非常不错的值得时常翻阅的技术手册。书中有很多地方进行Java与C++的比较，因为当时Java面世的时候，又被叫作“C++ Killer”。而我在看这本书的时候，发现书中有很多C++的东西，于是又去学习了C++。学习C++的时候，发现有很多C的东西不懂，又顺着去学习了C。然后，C -> C++ -> Java整条线融汇贯通，这对我未来的技术成长有非常大的帮助。
- 有了上述的入门后，Java的Spring框架是你玩Java所无法回避的东西，所以下来是两本Spring相关的书，《[Spring实战](#)》和《[Spring Boot实战](#)》。前者是传统的Spring，后者是新式的微服务的Spring。如果你只想看一本的话，那么就看后者吧。

认真学习前面的书可以让你成功入门Java，但想要进一步成长，就要看下面我推荐的几本提升级的书。

- 接下来，你需要了解了一下如何编写高效的代码，于是必需看一下《[Effective Java](#)》（注意，这里我给的引用是第三版的，也是2017年末出版的书），这本书是模仿Scott Meyers的经典图书《Effective C++》的。Effective这种书基本上都是各种经验之谈，所以，这是一本非常不错的书，你一定要读。这里需要推荐一下[Google Guava 库](#)，这个库不但是JDK的升级库，其中有如：集合（collections）、缓存（caching）、原生类型支持（primitives support）、并发库（concurrency libraries）、通用注解（common annotations）、字符串处理（string processing）、I/O 等库，其还是Effective Java这本书中的那些经验的实践代表。
- [《Java并发编程实战》](#)，是一本完美的Java并发参考手册。书中从并发性和线程安全性的基本概念出发，介绍了如何使用类库提供的基本并发构建块，用于避免并发危险、构造线程安全的类及验证线程安全的规则，如何将小的线程安全类组合成更大的线程安全类，如何利用线程来提高并发应用程序的吞吐量，如何识别可并行执行的任务，如何提高单线程系统的响应性，如何确保并发程序执行预期任务，如何提高并发代码的性能和可伸缩性等内容。最后介绍了一些高级主题，如显式锁、原子变量、非阻塞算法以及如何开发自定义的同步工具类。
- 了解如何编写出并发的程序，你还需要了解一下如何优化Java的性能。我推荐《[Java性能权威指南](#)》。通过学习这本书，你可以比较大程度地提升性能测试的效果。其中包括：使用JDK自带的工具收集Java应用的性能数据，理解JIT编译器的优缺点，调优JVM垃圾收集器以减少对程序的影响，学习管理堆内存和JVM原生内存的方法，了解如何最大程度地优化Java线程及同步的性能，等等。看完这本书后，如果你还有余力，想了解更多的底层细节，那么，你有必要去读一下《[深入理解Java虚拟机](#)》。
- [《Java编程思想》](#)，真是一本透着编程思想的书。上面的书让你从微观角度了解Java，而这本书则可以让你从一个宏观角度了解Java。这本书和Java核心技术的厚度差不多，但这本书的信息密度比较大。所以，读起来是非常耗大脑的，因为它会让你不断地思考。对于想学好Java的程序员来说，这是一本必读的书。
- [《精通Spring 4.x》](#)，也是一本很不错的书，就是有点厚，一共有800多页，都是干货。我认为其中最不错的部分是在分析原理，尤其是针对前面提到的Spring技术，应用与原理都讲得很透彻，IOC和AOP也分析得很棒，娓娓道来。其对任何一个技术都分析得很细致和全面，不足之处就是内容太多了，所以导致很厚，但这并不影响它是一本不错的工具书。

当然，学Java你一定要学面向对象的设计模式，这里就只有一本经典的书《[设计模式](#)》。如果你觉得有点儿难度了，那么可以看一下《[Head First设计模式](#)》。学习面向对象的设计模式时，你不要迷失在那23个设计模式中，你一定要明白这两个原则：

- **Program to an ‘interface’, not an ‘implementation’**

- 使用者不需要知道数据类型、结构、算法的细节。
- 使用者不需要知道实现细节，只需要知道提供的接口。
- 利于抽象、封装，动态绑定，多态。符合面向对象的特质和理念。

- **Favor ‘object composition’ over ‘class inheritance’**

- 继承需要给子类暴露一些父类的设计和实现细节。
- 父类实现的改变会造成子类也需要改变。
- 我们以为继承主要是为了代码重用，但实际上在子类中需要重新实现很多父类的方法。
- 继承更多的应该是为了多态。

至此，如果你把上面的这些知识都融汇贯通的话，那么，你已是一个高级的Java程序员了，我保证你已经超过了绝大多数程序员了。基本上来说，你在技术方面是可以进入到一线公司的，而且还不是一般的岗位，至少是高级程序员或是初级架构师的级别了。

C/C++语言

不像我出道那个时候，几乎所有的软件都要用C语言来写。现在，可能不会有多少人学习C语言了，因为一方面有Java、Python这样的高级语言为你屏蔽了很多的底层细节，另一方面也有像Go语言这样的新兴语言可以让你更容易地写出来也是高性能的软件。但是，我还是想说，C语言是你必须学习的语言，因为这个世界上绝大多数编程语言都是C-like的语言，也是在不同的方面来解决C语言的各种问题。[这里，我想放个比较武断话——如果你不学C语言，你根本没有资格说你是一个合格的程序员！](#)

- 这里尤其推荐，已故的C语言之父Dennis M. Ritchie和著名科学家Brian W. Kernighan合作的圣经级的教科书《[C程序设计语言](#)》。注意，这本书是C语言原作者写的，其C语言的标准不是我

们平时常说的ANSI标准，而是原作者的标准，又被叫作K&R C。但是这本书很轻薄，也简洁，不枯燥，是一本你可以拿着躺在床上看还不会看着睡着的书。

- 然后，还有一本非常经典的C语言的书《[C语言程序设计现代方法](#)》。有人说，这本书配合之前的《The C Programming Language》那本书简真是无敌。我想说，这本书更实用，也够厚，完整覆盖了C99标准，习题的质量和水准也比较高。更好的是，探讨了现代编译器的实现，以及和C++的兼容，还揭穿了各种古老的C语言的神话和信条……是相当相当干的一本学习C语言的书。

对了，千万不要看谭浩强的C语言的书。各种误导，我大学时就是用这本书学的C，后来工作时被坑得不行。

在学习C语言的过程中，你一定会感到，C语言这么底层，而且代码经常性地崩溃，经过一段时间的挣扎，你才开始觉得你从这个烂泥坑里快要爬出来了。但你还需要看看《[C陷阱与缺陷](#)》这本书，你会发现，这里面的坑不是一般大。

此时，如果你看过我的《编程范式游记》那个系列文章，你可能会发现C语言在泛型编程上的各种问题，这个时候我推荐你学习一下C++语言。可能会有很多人觉得我说的C++是个大坑。是的，这是世界目前来说最复杂也是最难的编程语言了。但是，C++是目前世界上范式最多的语言了，其做得最好的范式就是“泛型编程”，这在静态语言中，是绝对地划时代的一个事。

所以，你有必要学习一下C++，看看C++是如何解决C语言中的各种问题的。你可以先看看我的这篇文章“[C++的坑真的多吗？](#)”，有个基本认识。下面推荐几本C++的书。

- 《[C++ Primer中文版](#)》，这本书是久负盛名的C++经典教程。书是有点厚，前面1/3讲C语言，后面讲C++。C++的知识点实在是太多了，而且又有点晦涩。但是你主要就看几个点，一个是指向对象的多态，一个是模板和重载操作符，以及一些STL的东西。看看C++是怎么玩泛型和函数式编程的。
- 如果你想继续研究，你需要看另外两本更为经典的书《[Effective C++](#)》和《[More Effective C++](#)》。这两本书不厚，但是我读了10多年，每过一段时间再读一下，就会发现有更多的收获。这两本书的内容会随着你经历的丰富而变得丰富，这也是对我影响最大的两本书，其中影响最大的不是书中的那些C++的东西，而是作者的思维方式和不断求真的精神，这真是太赞了。
- 学习C/C++都是需要好好了解一下编译器到底干了什么事的。就像Java需要了解JVM一样，所以，这里还有一本非常非常难啃的书你可以挑战一下《[深度探索C++对象模型](#)》。这本书是非常之经典的，看完后，C++对你来说就再也没有什么秘密可言。我以前写过的《[C++虚函数表解析](#)》，还有《[C++对象内存布局](#)》属于这个范畴。
- 还有C++的作者 Bjarne Stroustrup 写的《[C++ FAQ \(中文版\)](#)》，也是非常值得一读的。

学习Go语言

C语言太原始了，C++太复杂了，Go语言是不二之选。有了C/C++的功底，学习Go语言非常简单。

首推《[Go by Example](#)》作为你的入门教程。然后，《[Go 101](#)》也是一个很不错的在线电子书。如果你想看纸书的话，《[The Go Programming Language](#)》一书在豆瓣上有9.2分，但是国内没有卖的。（当然，我以前也写过两篇入门的供你参考“[GO语言简介（上）-语法](#)”和“[GO语言简介（下）-特性](#)”）。

另外，《[Effective Go](#)》是必读的，这篇文章告诉你如何更好地使用Go语言，以及Go语言中的一些原理。

Go语言最突出之处是并发编程，Unix老牌黑客罗勃·派克（Rob Pike）在Google I/O上的两个分享，可以让你学习到一些并发编程的模式。

- Go Concurrency Patterns（[幻灯片](#)和[演讲视频](#)）。
- Advanced Go Concurrency Patterns（[幻灯片](#)、[演讲视频](#)）。

然后，Go在GitHub的wiki上有好多不错的学习资源，你可以从中学习到多。比如：

- [Go精华文章列表](#)。
- [Go相关博客列表](#)。
- [Go Talks](#)。

此外，还有个内容丰富的Go资源列表《[Awesome Go](#)》，推荐看看。

小结

好了，最后我们来总结一些今天分享的内容。在编程语言方面，我推荐学习C、C++、Java和Go四门语言，并分别阐释了推荐的原因。

- 我认为，C语言是必须学习的语言，因为这个世界上绝大多数编程语言都是C-like的语言，也是在不同的方面来解决C语言的各种问题。
- 而C++虽然复杂难学，但它几乎是目前世界上范式最多的语言了，其做得最好的范式就是“泛型编程”，这在静态语言中，是绝对地划时代的一个事。尤其要看看C++是如何解决C语言中的各种问题的。
- Java是我认为综合能力最强的语言。其实我是先学了Java，然后又去学了C++，之后去学了C语言的。C -> C++ -> Java整条线融汇贯通，这对我未来的技术成长有非常大的帮助。
- 在文章最末，我推荐了Go语言，并给出了相关的学习资料。

我认为，一个合格的程序员应该掌握几门语言。一方面，这会让你对不同的语言进行比较，让你有更多的思考。另一方面，这也是一种学习能力的培养，会让你对于未来的新技术学习得更快。

下篇文章中，我们将分享每个程序员都需要掌握的理论知识。敬请期待。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 -



耳听风

洞悉技术的本质
享受科技的乐趣



资深技术专家
骨灰级程序员



扫 码 订 阅

进入专业的编程领域，算法、数据结构、网络模型、计算机原理等这样的计算机科学专业需要学习的理论知识是必须要学习的。下面我们先来讲述数据结构和算法。

数据结构和算法

算法是比较难学习的，而且学习“算法”是需要智商的。数组、链表、哈希表、二叉树、排序算法等一些基础知识，对大多数人来说是没什么问题的。但是一旦进入到路径规划、背包问题、字符串匹配、动态规划、递归遍历等一些比较复杂的问题上，就会让很多人跟不上了，不但跟不上，而且还会非常痛苦。是的，解决算法问题的确是可以区分人类智商的一个比较好的方式，这也是为什么有些公司用算法题当面试题来找到智商比较高的程序员。

然而，在很多时候，我们在工作中却发现根本用不到算法，或是一些基本的算法也没有必要实现，只需要使用一下第三方的库就好了。于是，导致社会上出现很多“算法无用论”的声音。

对此，我想说，算法真的很重要。我这20年的经历告诉我，无论是做业务还是做底层系统，经常需要使用算法处理各种各样的问题。比如，业务上我需要用算法比较两个数组中差异的布隆过滤器，或是在做监控系统时实时计算过去一分钟的P99统计时的蓄水池算法，或是数据库的B+树索引，还有Linux内核中的epoll的红黑树，还有在做服务调度里的“背包问题”等都会用算法，真的是会本质上帮助到你，也是会让你非常有成就感的一件事。

虽然算法很难，需要智商，但我还是想鼓励你，这其中是有很多的套路是可以学习的，一旦学会这些套路，你会受益无穷的。

这里有几本书着重推荐一下。

- **基础知识。**《[算法](#)》，是算法领域经典的参考书，不但全面介绍了关于算法和数据结构的必备知识，还给出了每位程序员应知应会的50个算法，并提供了实际代码。最不错的是，其深入浅出的算法介绍，让一些比较难的算法也变得容易理解，尤其是书中对红黑树的讲解非常精彩。其中，还有大量的图解，详尽的代码和讲解，也许是最好的数据结构入门图书。不好的是不深，缺乏进一步的算法设计内容，甚至连动态规划都未提及。另外，如果你觉得算法书比较枯燥的话，你可以看看这本有趣的《[算法图解](#)》。
- **理论加持。**如果说上面这本书偏于实践和工程，而你看完后，对算法和数据结构的兴趣更浓了，那么你可以再看看另一本也是很经典的偏于理论方面的书——《[算法导论](#)》。虽然其中的一些理论知识在《算法》那本书中也有提过，但《算法导论》这本书更为专业一些，是美国计算机科学本科生的教科书。
- **思维改善。**还有一本叫《[编程珠玑](#)》的书，写这本书的人是世界著名计算机科学家乔恩·本特利（Jon Bentley），被誉为影响算法发展的十位大师之一。你可能不认识这个人，但是你知道他的学生有多厉害吗？我例举几个，一个是Tcl语言设计者约翰·奥斯特德奥特（John Ousterhout），另一个是Java语言设计者詹姆斯·高斯林（James Gosling），还有一个是《算法导论》作者之一查尔斯·雷瑟森（Charles Leiserson），还有好多好多。这本书也是很经典的算法书，其中都是一些非常实际的问题，并以其独有的洞察力和创造力，来引导读者理解并学会解决这些问题的方法，也是一本可以改善你思维方式的书。

然后，你需要去做一些题来训练一下自己的算法能力，这里就要推荐 [LeetCode](#) 这个网站了。它是一个很不错的做算法训练的地方。现在也越做越好了。基本上来说，这里会有两类题。

- **基础算法题。**其中有大量的算法题，解这些题都是有套路的，不是用递归（深度优先DFS，广度优先BFS），就是要用动态规划（Dynamic Programming），或是折半查找（Binary Search），或是回溯（Back tracing），或是分治法（Divide and Conquer），还有大量的对树、数组、链表、字符串和hash表的操作。通过做这些题能让你对这些最基础的算法的思路有非常扎实的了解和训练。对我而言，Dynamic Programming是我的短板，尤其是一些比较复杂的问题，在推导递推公式上总是有思维的缺陷（数学是我的硬伤）。做了这些题后，我能感到我在动态编程的思路上受到了很大的启发。
- **编程题。**比如：atoi, strstr, add two nums, 括号匹配，字符串乘法，通配符匹配，文件路径简化，Text Justification，反转单词等，这些题的Edge Case和Corner Case有很多。这些题需要你想清楚了再干，只要你稍有疏忽，就会有几个case让你痛不欲生，而且一不小心就会让你的代码写得又臭又长，无法阅读。通过做这些题，可以非常好地训练你对各种情况的考虑，以及你对程序代码组织的掌控（其实就是其中的状态变量）。

我觉得每个程序员都应该花时间和精力做这些题，因为你会从这些题中得到很大的收益。我在Leetcode上做的一些题的代码在这——我的 [Github](#) 上，可以给你一些参考。

如果能够把这些算法能力都掌握了，那么你就有很大的概率可以很容易地通过这世界上最优的公司的面试，比如：Google、Amazon、Facebook之类的公司。对你来说，如果能够进入到这些公司里工作，那么你未来的想像空间也会大得多得多。

最后，我们要知道这个世界上的数据结构和算法很多很多，下面给出了两个网站。

- [List of Algorithms](#)，这个网站罗列了非常多的算法，完全可以当成一个算法字典，或是用来开阔眼界。
- 还有一个数据结构动画图的网站 [Data Structure Visualizations](#)。

其它理论基础知识

下面这些书，基本上是计算机科学系的大学教材。如果你想有科班出生的理论基础，那么这些书是必读的。当然，这些理论基础知识比较枯燥，但我觉得如果你想成为专业的程序员，那么应该要找时间读一下。

- [《数据结构与算法分析》](#)，这本书曾被评为20世纪顶尖的30部计算机著作之一，作者Mark Allen Weiss在数据结构和算法分析方面卓有建树，他在数据结构和算法分析等方面的著作尤其畅销，并广受好评，已被世界500余所大学用作教材。
- [《数据库系统概念》](#)，它是数据库系统方面的经典教材之一。国际上许多著名大学包括斯坦福大学、耶鲁大学、德克萨斯大学、康奈尔大学、伊利诺伊大学、印度理工学院等都采用本书作为教科书。这本书全面介绍了数据库系统的各种知识，透彻阐释数据库管理的基本概念。不仅讨论了数据库查询语言、模式设计、数据仓库、数据库应用开发、基于对象的数据库和XML、数据存储和查询、事务管理、数据挖掘与信息检索以及数据库系统体系结构等方面的内容，而且对性能评测标准、性能调整、标准化以及空间与地理数据、事务处理监控等高级应用主题进行了广泛讨论。
- [《现代操作系统》](#)，这本书是操作系统领域的经典之作，书中集中讨论了操作系统的基本原理，包括进程、线程、存储管理、文件系统、输入/输出、死锁等，同时还包含了有关计算机安全、多媒体操作系统、掌上计算机操作系统、微内核、多核处理器上的虚拟机以及操作系统设计等方面的内容。
- [《计算机网络》](#)，这本书采用了独创的自顶向下方法，即从应用层开始沿协议栈向下讲解计算机网络的基本原理，强调应用层范例和应用编程接口，内容深入浅出，注重教学方法，理论与实践相结合。新版中还增加了无线和移动网络一章，并扩充了对等网络、BGP、MPLS、网络安全、广播选路和因特网编址及转发方面的材料。是一本不可多得的教科书。
- [《计算机程序的构造和解释》](#)，这本书也很经典，是MIT的计算机科学系的教材。这本书中主要证实了很多程序是怎么构造出来的，以及程序的本质是什么。整本书主要是使用Scheme/Lisp语言，从数据抽象、过程抽象、迭代、高阶函数等编程和控制系统复杂性的思想，到数据结构和算法，到编译器/解释器、编程语言设计。
- [《编译原理》](#)，这本书又叫“龙书”，其全面、深入地探讨了编译器设计方面的重要主题，包括词法分析、语法分析、语法制导定义和语法制导翻译、运行时刻环境、目标代码生成、代码优化技术、并行性检测以及过程间分析技术，并在相关章节中给出大量的实例。与上一版相比，本书进行了全面的修订，涵盖了编译器开发方面的最新进展。每章中都提供了大量的系统及参考文献。

小结

好了，最后我们来总结一些今天分享的内容。在这篇文章中，我建议想进入专业编程领域的人，一定要算法、数据结构、网络模型、计算机原理等理论知识，并推荐了相应的学习素材，给出了我的思考和建议。

我认为，虽然这些理论知识枯燥难学，而且通常学完了在工作中也并不是马上就能用上，但这些知识是必须要学好的。这些理论知识可以说是计算机科学这门学科最精华的知识了，认真学习，理解其背后的逻辑和思维方式，会令你受益匪浅。不管是未来学习新知识，还是解决什么疑难问题，都能在这些知识中获得灵感或者启发。

下篇文章中，我们将分享每个程序员都需要掌握的系统知识。敬请期待。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)

- 理论学科
- 系统知识
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络 \(系统底层知识\)](#)
 - [异步I/O模型和Lock-Free编程 \(系统底层知识\)](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门 \(分布式架构\)](#)
 - [分布式架构经典图书和论文 \(分布式架构\)](#)
 -

The promotional image features a portrait of Chen Hao, a middle-aged man with short dark hair, wearing a dark blue long-sleeved shirt. He is seated, looking slightly to his right with a faint smile. To his left, large vertical Chinese characters '左耳听风' are written in a light blue, artistic font. Below these characters is a vertical column of smaller text: '洞悉技术的本质' and '享受科技的乐趣'. In the top right corner of the image area, there is a logo consisting of a stylized orange 'Q' shape followed by the text '极客时间' and the tagline '重启极客精神·提升技术认知'. To the right of Chen Hao's name, '陈皓' is written in a large, bold, black font. Below it, the text '资深技术专家' and '骨灰级程序员' is displayed in a smaller black font. In the bottom right corner of the image area, there is a QR code with the text '扫码订阅' underneath it.

左耳听风

洞悉技术的本质
享受科技的乐趣

陈皓

资深技术专家
骨灰级程序员

极客时间

重启极客精神·提升技术认知

扫码订阅

程序员练级攻略（2018）：系统知识

2018-06-19 陈皓

进入专业的编程领域，学习系统知识是非常关键的一部分。

首先推荐的是翻译版图书《[深入理解计算机系统](#)》，原书名为《Computer Systems A Programmer's Perspective》。不过，这本书叫做《程序员所需要了解的计算机知识》更为合适。

本书的最大优点是为程序员描述计算机系统的实现细节，帮助其在大脑中构造一个层次型的计算机系统。从最底层的数据在内存中的表示到流水线指令的构成，到虚拟存储器，到编译系统，到动态加载库，到最后的用户态应用。通过掌握程序是如何映射到系统上，以及程序是如何执行的，你能够更好地理解程序的行为为什么是这样的，以及效率低下是如何造成的。

再强调一下，这本书是程序员必读的一本书！

然后就是美国计算机科学家[理查德·史蒂文斯（Richard Stevens）](#)的三套巨经典无比的书。（理查德·史蒂文斯于1999年9月1日离世，终年48岁。死因不详，有人说他是滑雪意外，也有人说他是攀岩意外，也有人说他是滑翔机意外。总之，家人没有透露。大师的[个人主页](#)今天还可以访问。）

- [《Unix高级环境编程》](#)。
- [《Unix网络编程》第1卷 套接口API、第2卷 进程间通信](#)。
- [《TCP/IP详解 卷I 协议》](#)。

这几书的地位我就不多说了，你可以自己看相关的书评。但是，这三本书可能都不容易读，一方面是比较厚，另一方面是知识的密度太大了，所以，读起来有点枯燥和乏味。但是，这没办法，你得忍住。

这里要重点说一下《TCP/IP详解》这本书，是一本很奇怪的书。这本书迄今至少被[近五百篇学术论文引用过](#)。这本写给工程师看的书居然被各种学院派的论文来引用，也是很神奇的一件事了。而且，虽然理查德·史蒂文斯不是TCP的发明人，但是这本书中把这个协议深入浅出地讲出来，还画了几百张时序图，也是令人叹为观止了。

如果你觉得上面这几本经典书比较难啃，你可以试试下面这些通俗易懂的（当然，如果读得懂上面那三本的，下面的这些也就不需要读了）。

- [《Linux C编程一站式学习》](#)。
- [《TCP/IP网络编程》](#)。
- [《图解TCP/IP》](#)，这本书其实并不是只讲了TCP/IP，应该是叫《计算机网络》才对，主要是给想快速入门的人看的。
- [《The TCP/IP Guide》](#)，这本书在豆瓣上的评分9.2，这里给的链接是这本书的HTML英文免费版的，里面的图画得很精彩。

另外，学习网络协议不单只是看书，你最好用个抓包工具看看这些网络包是什么样的。所以，这里推荐一本书《[Wireshark数据包分析实战](#)》。在这本书中，作者结合一些简单易懂的实际网络案例，图文并茂地演示使用Wireshark进行数据包分析的技术方法，可以让我们更好地了解和学习网络协议。当然，也拥有了一定的黑客的技能。

看完《Unix高级环境编程》后，你可以趁热打铁看看《[Linux/Unix系统编程手册](#)》或是罗伯特·拉姆（Robert Love）的[Linux System Programming 英文电子版](#)。其中文翻译版[Linux系统编程](#)也值得一读，虽然和《Unix高级环境编程》很像，不过其主要突出的是Linux的一些关键技术相关的系统调用。

关于TCP的东西，你还可以看看下面这一系列的文章。

- [Let's code a TCP/IP stack, 1: Ethernet & ARP](#)
- [Let's code a TCP/IP stack, 2: IPv4 & ICMPv4](#)
- [Let's code a TCP/IP stack, 3: TCP Basics & Handshake](#)
- [Let's code a TCP/IP stack, 4: TCP Data Flow & Socket API](#)
- [Let's code a TCP/IP stack, 5: TCP Retransmission](#)

对于系统知识，我认为主要有以下一些学习要点。

- 用这些系统知识操作一下文件系统，实现一个可以拷贝目录树的小程序。
- 用fork / wait / waitpid写一个多进程的程序，用pthread写一个多线程带同步或互斥的程序。比如，多进程购票的程序。
- 用signal / kill / raise / alarm / pause / sigprocmask实现一个多进程间的信号量通信的程序。
- 学会使用gcc和gdb来编程和调试程序（参看我的[《用gdb调试程序》二、三、四、五、六、七](#)）。
- 学会使用makefile来编译程序（参看我的[《跟我一起写makefile》二、三、四、五、六、七、八、九、十、十一、十二、十三、十四](#)）。
- Socket的进程间通信。用C语言写一个1对1的聊天小程序，或是一个简单的HTTP服务器。

C10K问题

然后，当你读完《Unix网络编程》后，千万要去读一下“[C10K Problem（中文翻译版）](#)”。提出这个问题的人叫丹·凯格尔（Dan Kegel），目前工作在美国Google公司。

他从1978年起开始接触计算机编程，是Winetricks的作者，也是Wine 1.0的管理员，同时也是Crosstool（一个让gcc/glibc编译器更易用的工具套件）的作者。还是Java JSR 51规范的提交者并参与编写了Java平台的NIO和文件锁，同时参与了RFC 5128标准中有关NAT穿越（P2P打洞）技术的描述和定义。

C10K问题是操作系统处理大并发请求的问题。对于Web时代的操作系统而言，对于客户端过来的大量的并发请求，需要创建相应的服务进程或线程。这些进程或线程多了，导致数据拷贝频繁（缓存I/O、内核将数据拷贝到用户进程空间、阻塞），进程/线程上下文切换消耗大，从而导致资源被耗尽而崩溃。这就是C10K问题的本质。

了解这个问题，并了解操作系统是如何通过多路复用的技术来解决这个问题的，有助于你了解各种I/O和异步模型，这对于你未来的编程和架构能力是相当重要的。

另外，现在，整个世界都在解决C10M问题，推荐看看[The Secret To 10 Million Concurrent Connections -The Kernel Is The Problem, Not The Solution](#)一文。

实践项目

我们已经学习完了编程语言、理论学科和系统知识三部分内容，下面就来做几个实践项目，小试牛刀一下。实现语言可以用C、C++或Java。

实现一个telnet版本的聊天服务器，主要有以下需求。

- 每个客户端可以使用telnet ip:port的方式连接到服务器上。
- 新连接需要用用户名和密码登录，如果没有，则需要注册一个。
- 然后可以选择一个聊天室加入聊天。
- 管理员有权创建或删除聊天室，普通人员只有加入、退出、查询聊天室的权力。
- 聊天室需要有人数限制，每个人发出来的话，其它所有的人都能看到。

实现一个简单的HTTP服务器，主要有以下需求。

- 解释浏览器传来的HTTP协议，只需要处理URL path。
- 然后把所代理的目录列出来。
- 在浏览器上可以浏览目录里的文件和下级目录。
- 如果点击文件，则把文件打开传给浏览器（浏览器能够自动显示图片、PDF，或HTML、CSS、JavaScript以及文本文件）。
- 如果点击子目录，则进入子目录中，并把子目录中的文件列出来。

实现一个生产者/消费者消息队列服务，主要有以下需求。

- 消息队列采用一个Ring-buffer的数据结构。
- 可以有多个topic供生产者写入消息及消费者取出消息。
- 需要支持多个生产者并发写。
- 需要支持多个消费者消费消息（只要有一个消费者成功处理消息就可以删除消息）。
- 消息队列要做到不丢数据（要把消息持久化下来）。
- 能做到性能很高。

小结

到目前为止，我们已经学习完了专业编程里面最重要的三部分的内容：编程语言、理论学科和系统知识。我们针对这些内容做个小结。如果想看我推荐的那些书和知识，并整理归类和掌握，我建议

到今天，我们已经学习完了专业编程方面最为重要的二部分内容：编程语言、理论学科和系统知识，我们针对这些内容做个小结。如果想看我推荐的那些书和知识，升能理解和掌握，我估计忘得需要4-5年的时间。嗯，是的，就是一个计算机科学系科班出身的程序员需要学习的一些东西。这其中，最重要的是下面这几点。

编程语言。以工业级的C、C++、Java这三门语言为主，这三门语言才是真正算得上工业级的编程语言，因为有工业级的标准化组织在控制着这几门语言，而且也有工业级的企业应用。尤其是Java，还衍生出了大量的企业级架构上的开源生态。你至少需要掌握C语言和Java语言，这对你以后面对各式各样的编程语言是非常重要的。

此外，还推荐学习Go语言，它已成为云计算领域事实上的标准语言，尤其是在Docker、Kubernetes等项目中。而且，Go语言在国内外一些知名公司中有了一定的应用和实践，并且其生态圈也越来越好。

算法和数据结构。这个太重要了，尤其是最基础的算法和数据结构，这是任何一个称职的程序员都需要学习和掌握的。你必需要掌握。

计算机的相关系统。你至少要掌握三个系统的基础知识，一个是操作系统，一个是网络系统，还有一个是数据库系统。它们分别代表着计算机基础构架的三大件——计算、存储、网络。

如果你能够走到这里，把前面的那些知识都了解了（不用精通，因为精通是需要时间和实践来慢慢锤炼出来的，所以，你也不用着急），那么你已经是一个非常非常合格的程序员了，而且你的潜力和可能性是非常非常高的。

如果经历过这些比较枯燥的理论知识，而且你还能有热情和成就感，那么我要恭喜你了。因为你已经超过了绝大多数人，而且还是排在上游的比较抢手的程序员了。我相信你至少可以找到年薪50万以上的工作了。

但是，你还需要很多的经验或是一些实践，以及一些大系统大项目的实际动手的经验。没关系，我们后面会有教你怎么实操的方法和攻略。

但是，往后面走，你需要开始需要术业有专攻了。下面给一些建议的方向。

- **底层方向：**操作系统、文件系统、数据库、网络.....
- **架构方向：**分布式系统架构、微服务、DevOps、Cloud Native.....
- **数据方向：**大数据、机器学习、人工智能.....
- **前端方向：**你对用户体验或是交互更感兴趣，那么你走前端的路吧。
- **其它方向：**比如，安全开发、运维开发、嵌入式开发.....

这些方向你要仔细选择，因为一旦选好，就要勇往直前地走下去，当然，你要回头转别的方向也没什么问题，因为你有前面的这些基础知识在身，所以，不用害怕。只是不同的方向上会有不同的经验积累，经验积累是看书看不来的，这个是转方向的成本。

下篇文章，我们将进入《软件设计篇》。敬请期待。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 -

The promotional image features a portrait of Chen Hao, a middle-aged man with short dark hair, wearing a dark blue button-down shirt. He is seated at a desk, resting his chin on his hand in a thoughtful pose. The background is a solid light blue. On the left side of the image, there is large vertical white text reading '左耳听风'. To the right of Chen Hao, his name '陈皓' is written in large white characters. Below his name, the text '资深技术专家' and '骨灰级程序员' is displayed in smaller white font. In the top right corner, the logo for '极客时间' (Geek Time) is shown, consisting of a stylized orange 'G' icon followed by the text '极客时间' and the tagline '重拾极客精神·提升技术认知'. In the bottom right corner, there is a QR code with the text '扫码订阅' below it.

学习软件设计的方法、理念、范式和模式，是让你从一个程序员通向工程师的必备技能。如果你不懂这些设计方法，那么你将无法做出优质的软件。这就好像写作文一样，文章人人都能写，但是能写得有条理，有章法，有血有肉，就不简单了。软件开发也一样，实现功能，做出来并不难，但是要做漂亮，做优雅，就非常不容易了。

Linus说过，这世界程序员之所有高下之分，最大的区别就是程序员的“品味”不一样。有品位的程序员和没有品位的程序员所写出来的代码，所做出来的软件，差距非常大，而且价值也差别很大。所以，对于我们每一个程序员，如果你想成为软件工程师、设计师或架构师，软件设计是你必须用心学习的事。

然而，软件设计这个事，并不是一朝一夕就能学会的，也不是别人能把你教会的，很多东西需要你自己用实践、用时间、用错误、用教训、用痛苦才能真正体会其中的精髓的。所以，除了学习理论知识外，你还需要大量的工程实践，然后每过一段时间就把这些设计的东西重新回炉一下。你会发现这些软件设计的东西，就像饮茶一样，一开始是苦的，然后慢慢回甘，最终你会喝出真正的滋味。

要学好这些软件开发和设计的方法，你真的需要磨练和苦行，反复咀嚼，反复推敲，在实践和理论中螺旋式地学习，才能真正掌握。 所以，你需要有足够的耐心和恒心。

编程范式

学习编程范式可以让你明白编程的本质和各种语言的编程方式。虽然很多程序员都忽略了这个事，但是其实是非常非常重要的事。因此，我推荐以下一些资料，帮助你系统化地学习和理解。

- 一个是我极客时间写的《编程范式游记》系列文章，目录如下。

- [编程范式游记 \(1\) - 起源](#)
- [编程范式游记 \(2\) - 泛型编程](#)
- [编程范式游记 \(3\) - 类型系统和泛型的本质](#)
- [编程范式游记 \(4\) - 函数式编程](#)
- [编程范式游记 \(5\) - 修饰器模式](#)
- [编程范式游记 \(6\) - 面向对象编程](#)
- [编程范式游记 \(7\) - 基于原型的编程范式](#)
- [编程范式游记 \(8\) - Go 语言的委托模式](#)
- [编程范式游记 \(9\) - 编程的本质](#)
- [编程范式游记 \(10\) - 逻辑编程范式](#)
- [编程范式游记 \(11\) - 程序世界里的编程范式](#)

- [Wikipedia: Programming paradigm](#)，维基百科上有一个编程范式的页面，顺着这个页面看下去，你可以看到很多很多有用的知识。这些东西对你的编程技能的提高会非常非常有帮助。

- [Six programming paradigms that will change how you think about coding](#)，中文翻译版为[六个编程范型将改变你对编程的看法](#)。这篇文章讲了默认支持并发 (Concurrent by default)、依赖类型 (Dependent types)、连接性语言 (Concatenative languages)、声明式编程 (Declarative programming)、符号式编程 (Symbolic programming)、基于知识的编程 (Knowledge-based programming) 等六种不太常见的编程范式，并结合了一些你没怎么听说过的语言来分别进行讲述。

比如在讲Concatenative languages时，以Forth、cat和joy三种语言为例讲述这一编程范式背后的思想——语言中的所有内容都是一个函数，用于将数据推送到堆栈或从堆栈弹出数据；程序几乎完全通过功能组合来构建 (concatenation is composition)。作者认为，这些编程范式背后的思想十分有魅力，能够改变对编程的思考。我看完此文，对此也深信不疑。虽然这些语言和编程范式不常用到，但确实能在思想层面给予人很大的启发。这也是我推荐此文的目的。

- [Programming Paradigms for Dummies: What Every Programmer Should Know](#)，这篇文章的作者彼得·范·罗伊 (Peter Van Roy) 是比利时鲁汶大学的计算机科学教师。他在这篇文章里分析了编程语言在历史上的演进，有哪些典型的、值得研究的案例，里面体现了哪些值得学习的范式。

比如，在分布式编程领域，他提到了Erlang、E、Distributed Oz和Didactic Oz这四种编程语言。虽然它们都是分布式编程语言，但各有特色，各自解决了不同的问题。通过这篇文章能学到不少在设计编程语言时要考虑的问题，让你重新审视自己所使用的编程语言应该怎样用才能用好，有什么局限性，这些局限性能否被克服等。

- [斯坦福大学公开课：编程范式](#)，这是一门比较基础且很详细的课程，适合学习编程语言的初学者。它通过讲述C、C++、并发编程、Scheme、Python这5门语言，介绍了它们各自不同的编程范式。以C语言为例，它解释了C语言的基本要素，如指针、内存分配、堆、C风格的字符串等，并解释了为什么C语言会在泛型编程、多态等方面有局限性。通过学习这门课程，你会对一些常用的编程范式有所了解。

一些软件设计的相关原则

- [Don't Repeat Yourself \(DRY\)](#)，DRY是一个最简单的法则，也是最容易被理解的。但它也可能是最难被应用的（因为要做到这样，我们需要在泛型设计上做相当的努力，这并不是一件容易的事）。它意味着，当在两个或多个地方发现一些相似的代码的时候，我们需要把它们的共性抽象出来形成一个唯一的新方法，并且改变现有地方的代码让它们以一些合适的参数调用这个新的方法。

- [Keep It Simple, Stupid \(KISS\)](#)，KISS原则在设计上可能最被推崇，在家装设计、界面设计和操作设计上，复杂的东西越来越被众人所鄙视了，而简单的东西越来越被人所认可。宜家 (IKEA) 简约、高效的家居设计和生产思路；微软 (Microsoft) “所见即所得”的理念；谷歌 (Google) 简约、直接的商业风格，无一例外地遵循了“KISS”原则。也正是“KISS”原则，成就了这些看似神奇的商业经典。而苹果公司的iPhone和iPad将这个原则实践到了极至。

- [Program to an interface, not an implementation](#)，这是设计模式中最根本的哲学，注重接口，而不是实现，依赖接口，而不是实现。接口是抽象是稳定的，实现则是多种多样的。在面向对象的S.O.L.I.D原则中会提到我们的依赖倒置原则，就是这个原则的另一种样子。还有一条原则叫 [Composition over inheritance](#) (喜欢组合而不是继承)，这两条是那23个经典设计模式中的设计原则。

- [You Ain't Gonna Need It \(YAGNI\)](#)，这个原则简而言之为——只考虑和设计必须的功能，避免过度设计。只实现目前需要的功能，在以后你需要更多功能时，可以再进行添加。如无必要，勿增复杂性。软件开发是一场trade-off的博弈。

- [Law of Demeter](#)，迪米特法则(Law of Demeter)，又称“最少知识原则”(Principle of Least Knowledge)，其来源于1987年荷兰大学的一个叫做Demeter的项目。克雷格·拉尔曼 (Craig Larman) 把Law of Demeter又称作“不要和陌生人说话”。在《程序员修炼之道》中讲LoD的那一章将其叫作“解耦合与迪米特法则”。

关于迪米特法则有一些很形象的比喻：1) 如果你想让你的狗跑的话，你会对狗狗说还是对四条狗腿说？2) 如果你去店里买东西，你会把钱交给店员，还是会把钱包交给店员让他自己拿？和狗的四肢说话？让店员自己从钱包里拿钱？这听起来有点儿荒唐，不过在我们的代码里这几乎是见怪不怪的事情了。对于LoD，正式的表述如下：

对于对象‘O’中一个方法‘M’，M应该只能访问以下对象中的方法：

1. 对象O；
2. 与O直接相关的Component Object；
3. 由方法M创建或者实例化的对象；
4. 作为方法M的参数的对象。

- [面向对象的S.O.L.I.D 原则](#)">[http://en.wikipedia.org/wiki/Solid_\(object-oriented_design\)](http://en.wikipedia.org/wiki/Solid_(object-oriented_design)):

- **SRP (Single Responsibility Principle) - 职责单一原则。** 关于单一职责原则，其核心的思想是：一个类，只做一件事，并把这件事做好，其只有一个引起它变化的原因。单一职责原则可以看作是低耦合、高内聚在面向对象原则上的引申，将职责定义为引起变化的原因，以提高内聚性来减少引起变化的原因。

职责过多，可能引起它变化的原因就越多，这将导致职责依赖，相互之间就产生影响，从而极大地损伤其内聚性和耦合度。单一职责，通常意味着单一的功能，因此不要为一个模块实现过多的功能点，以保证实体只有一个引起它变化的原因。

- **** OCP (Open/Closed Principle) - 开闭原则**。** 关于开发封闭原则，其核心的思想是：模块是可扩展的，而不可修改的。也就是说，对扩展是开放的，而对修改是封闭的。对扩展开放，意味着有新的需求或变化时，可以对现有代码进行扩展，以适应新的情况。对修改封闭，意味着类一旦设计完成，就可以独立完成其工作，而不要对类进行任何修改。

- **LSP (Liskov substitution principle) - 里氏代换原则。** 软件工程大师罗伯特·马丁 (Robert C. Martin) 把里氏代换原则最终简化为一句话：“Subtypes must be substitutable for their base types”。也就是，子类必须能够替换成它们的基类。即子类应该可以替换任何基类能够出现的地方，并且经过替换以后，代码还能正常工作。另外，不应该在代码中出现if/else之类对子类类型进行判断的条件。里氏替换原则LSP是使代码符合开闭原则的一个重要保证。正是由于子类型的可替换性才使得父类型的模块在无需修改的情况下就可以扩展。

- **ISP (Interface Segregation Principle) - 接口隔离原则。** 接口隔离原则的意思是把功能实现在接口中，而不是类中，使用多个专门的接口比使用单一的总接口要好。举个例子，我们对电脑有不同的使用方式，比如：写作、通讯、看电影、打游戏、上网、编程、计算和数据存储等。

如果我们都把这些功能都声明在电脑的抽象类里面，那么，我们的上网本、PC机、服务器和笔记本的实现类都要实现所有的这些接口，这就显得太复杂了。所以，我们可以把这些功能接口隔离开来，如工作学习接口、编程开发接口、上网娱乐接口、计算和数据服务接口，这样，我们的不同功能的电脑就可以有所选择地继承这些接口。

- **DIP (Dependency Inversion Principle) - 依赖倒置原则。** 高层模块不应该依赖于低层模块的实现，而是依赖于高层抽象。举个例子，墙面的开关不应该依赖于电灯的开关实现，而是应该依赖于一个抽象的开关的标准接口。这样，当我们扩展程序的时候，开关同样可以控制其它不同的灯，甚至不同的电器。也就是说，电灯和其它电器继承并实现我们的标准开关接口，而开关厂商就可以不需要关心其要控制什么样的设备，只需要关心那个标准的开关标准。这就是依赖倒置原则。

- **CCP (Common Closure Principle) - 共同封闭原则**，一个包中所有的类应该对同一种类型的变化关闭。一个变化影响一个包，便影响了包中所有的类。一个更简短的说法是：一起修改的类，应该组合在一起（同一个包里）。如果必须修改应用程序里的代码，那么我们希望所有的修改都发生在一个包里（修改关闭），而不是遍布在很多包里。

CCP原则就是把因为某个同样的原因而需要修改的所有类组合进一个包里。如果两个类从物理上或者从概念上联系得非常紧密，它们通常一起发生改变，那么它们应该属于同一个包。CCP延伸了开闭原则（OCP）的“关闭”概念，当因为某个原因需要修改时，把需要修改的范围限制在一个最小范围内的包里。

- **CRP (Common Reuse Principle) - 共同重用原则**，包的所有类被一起重用。如果你重用了其中的一个类，就重用全部。换个说法是，没有被一起重用的类不应该组合在一起。CRP原则帮助我们决定哪些类应该被放到同一个包里。依赖一个包就是依赖这个包所包含的一切。

当一个包发生了改变，并发布新的版本，使用这个包的所有用户都必须在新的包环境下验证他们的工作，即使被他们使用的部分没有发生任何改变。因为如果包中包含未被使用的类，即使用户不关心该类是否改变，但用户还是不得不升级该包并对原来的功能加以重新测试。CCP则让系统的维护者受益。CCP让包尽可能大（CCP原则加入功能相关的类），CRP则让包尽可能小（CRP原则剔除不使用的类）。它们的出发点不一样，但不相互冲突。

- **好莱坞原则 - Hollywood Principle**，好莱坞原则就是一句话——“don't call us, we'll call you.”。意思是，好莱坞的经纪人不希望你去联系他们，而是他们会在需要的时候来联系你。也就是说，所有的组件都是被动的，所有的组件初始化和调用都由容器负责。

简单来讲，就是由容器控制程序之间的关系，而非传统实现中，由程序代码直接操控。这也就是所谓“控制反转”的概念所在：1) 不创建对象，而是描述创建对象的方式。2) 在代码中，对象与服务没有直接联系，而是容器负责将这些联系在一起。控制权由应用代码中转到了外部容器，控制权的转移，是所谓反转。好莱坞原则就是IoC (Inversion of Control) 或DI (Dependency Injection) 的基础原则。

- **高内聚，低耦合 & - High Cohesion & Low/Loose coupling**，这个原则是UNIX操作系统设计的经典原则，把模块间的耦合降到最低，而努力让一个模块做到精益求精。内聚，指一个模块内各个元素彼此结合的紧密程度；耦合指一个软件结构内不同模块之间互连程度的度量。内聚意味着重用和独立，耦合意味着多米诺效应牵一发动全身。对于面向对象来说，你也可以看看马萨诸塞州戈登学院的面向对象课中的这一节讲义High Cohesion and Low Coupling。

- **CoC (Convention over Configuration) - 惯例优于配置原则**，简单点说，就是将一些公认的配置方式和信息作为内部缺省的规则来使用。例如，Hibernate的映射文件，如果约定字段名和类属性一致的话，基本上就可以不要这个配置文件了。你的应用只需要指定不convention的信息即可，从而减少了大量convention而又不得不花时间和精力啰嗦的东东。

配置文件很多时候相当影响开发效率。Rails 中很少有配置文件（但不是没有，数据库连接就是一个配置文件）。Rails 的fans号称其开发效率是 Java 开发的 10 倍，估计就是这个原因。Maven也使用了CoC原则，当你执行 mvn -compile 命令的时候，不需要指定源文件放在什么地方，而编译以后的class文件放置在什么地方也没有指定，这就是CoC原则。

- **SoC (Separation of Concerns) - 关注点分离**，SoC 是计算机科学中最重要的努力目标之一。这个原则，就是在软件开发中，通过各种手段，将问题的各个关注点分开。如果一个问题能分解为独立且较小的问题，就是相对较易解决的。问题太过于复杂，要解决问题需要关注的点太多，而程序员的能力是有限的，不能同时关注于问题的各个方面。

正如程序员的记忆力相对于计算机知识来说那么有限一样，程序员解决问题的能力相对于要解决的问题的复杂性也是一样的非常有限。在我们分析问题的时候，如果我们把所有的东西混在一起讨论，那么就只会有一个结果——乱。实现关注点分离的方法主要有两种，一种是标准化，另一种是抽象与包装。标准化就是制定一套标准，让使用者都遵守它，将人们的行为统一起来，这样使用标准的人就不用担心别人会有很多种不同的实现，使自己的程序不能和别人的配合。

就像是开发螺丝钉的人只专注于开发螺丝钉就行了，而不用关注螺帽是怎么生产的，反正螺帽和螺丝钉按照标准来就一定能合得上。不断地把程序的某些部分抽象并包装起来，也是实现关注点分离的好方法。一旦一个函数被抽象出来并实现了，那么使用函数的人就不用关心这个函数是如何实现的。同样的，一旦一个类被抽象并实现了，类的使用者也不用再关注于这个类的内部是如何实现的。诸如组件、分层、面向服务等这些概念都是在不同的层次上做抽象和包装，以使得使用者不用关心它的内部实现细节。

- **DbC (Design by Contract) - 契约式设计**，DbC的核心思想是对软件系统中的元素之间相互合作以及“责任”与“义务”的比喻。这种比喻从商业活动中“客户”与“供应商”达成“契约”而得来。如果在程序设计中一个模块提供了某种功能，那么它要：

- 期望所有调用它的客户模块都保证一定的进入条件：这就是模块的先验条件（客户的义务和供应商的权利，这样它就不用去处理不满足先验条件的情况）。
- 保证退出时给出特定的属性：这就是模块的后验条件（供应商的义务，显然也是客户的权利）。
- 在进入时假定，并在退出时保持一些特定的属性：不变式。

- **ADP (Acyclic Dependencies Principle) - 无环依赖原则**，包（或服务）之间的依赖结构必须是一个直接的无环图形，也就是说，在依赖结构中不允许出现环（循环依赖）。如果包的依赖形成了环状结构，怎么样打破这种循环依赖呢？

有两种方法可以打破这种循环依赖关系：第一种方法是创建新的包，如果A、B、C形成环路依赖，那么把这些共同类抽出来放在一个新的包D里。这样就把C依赖A变成了C依赖D以及A依赖D，从而打破了循环依赖关系。第二种方法是使用DIP（依赖倒置原则）和ISP（接口分隔原则）设计原则。无环依赖原则（ADP）为我们解决包之间的关系耦合问题。在设计模块时，不能有循环依赖。

一些软件设计的读物

- **《领域驱动设计》**，本书是领域驱动设计方面的经典之作。全书围绕着设计和开发实践，结合若干真实的项目案例，向读者阐述如何在真实的软件开发中应用领域驱动设计。书中给出了领域驱动设计的系统化方法，并将人们普遍接受的一些实践综合到一起，融入了作者的见解和经验，展现了一些可扩展的设计新实践、已验证过的技术以及便于应对复杂领域的软件项目开发的基本原则。

- **《UNIX编程艺术》**，这本书主要介绍了Unix系统领域中的设计和开发哲学、思想文化体系、原则与经验，由公认的Unix编程大师、开源运动领袖人物之一埃里克·雷蒙德（Eric S. Raymond）倾力多年写作而成。包括Unix设计者在内的多位领域专家也为本书贡献了宝贵的内容。本书内容涉及社群文化、软件开发设计与实现，覆盖面广、内容深邃，完全展现了作者极其深厚的经验积累和领域智慧。

- **《Clean Architecture》**，如果你读过《Clean Code》和《The Clean Coder》这两本书。你就能猜得到这种 Clean 系列一定也是出自“Bob大叔”之手。没错，就是Bob大叔的心血之作。除了这个网站，《Clean Architecture》也是一本书，这是一本很不错的架构类图书。对软件架构的元素、方法等讲得很清楚。示例都比较简单，并带一些软件变化历史的讲述，很开阔视野。

- **The Twelve-Factor App**，如今，软件通常会作为一种服务来交付，它们被称为网络应用程序，或软件即服务（SaaS）。12-Factor 为构建SaaS 应用提供了方法论，这也是架构师必读的文章。（中译版）这篇文章在业内的影响力很大，必读！

- **Avoid Over Engineering**，有时候，我们会过度设计我们的系统，过度设计会把我们带到另外一个复杂度上，所以，我们需要一些工程上的平衡。这篇文章是一篇非常不错地告诉你什么是过度设计的文章。

- **Instagram Engineering's 3 rules to a scalable cloud application architecture**，Instagram 工程的三个黄金法则：1) 使用稳定可靠的技术（迎接新的技术）；2) 不要重新发明轮子；3) Keep it very simple。我觉得这三条很不错。其实，Amazon也有两条工程法则，一个是自动化，一个是简化。

- **How To Design A Good API and Why it Matters - Joshua Bloch**，Google的一个分享，关于如何设计好一个API。

- 关于Restful API的设计，你可以学习并借鉴一下下面这些文章。

- [Best Practices for Designing a Pragmatic RESTful API](#)
- [Ideal REST API design](#)
- [HTTP API Design Guide](#)
- [Microsoft REST API Guidelines](#)
- [IBM Watson REST API Guidelines](#)
- [Zalando RESTful API and Event Schema Guidelines](#)

- **The Problem With Logging**，一篇关于程序打日志的短文，可以让你知道一些可能以前不知道的打日志需要注意的问题。

• [The Problem with Logging](#), 一篇关于程序员打日志的短文，可以让你知道一些可能以任何不知道打了日志而要上热搜的问题。

• [Concurrent Programming for Scalable Web Architectures](#)，这是一本在线的免费书，教你如何架构一个可扩展的高性能的网站。其中谈到了一些不错的[设计方法和知识](#)。

小结

好了，总结一下今天分享的内容。我认为，“品位”不同，是各层次程序员之间最大的区别，这也决定了他们所做出来的软件的质量和价值。因此，我特意撰写了软件设计这一篇章，帮助那些想成长为软件工程师、设计师或架构师的程序员，提高软件设计的品位，进而实现自己的目标。

虽然很多程序员都忽略了对编程范式的学习，但我觉得学习编程范式其实是非常非常重要的事，能够明白编程的本质和各种语言的编程方式。为此，我推荐了好几份学习资料，帮助你系统化地学习和理解。随后我介绍了DRY-避免重复原则、KISS-简单原则、迪米特法则（又称“最少知识原则”）、面向对象的S.O.L.I.D原则等多个经典的软件设计原则。

最后，我精选并推荐了软件设计方面的学习资料，如《领域驱动设计》、《UNIX编程艺术》和《Clean Architecture》等必读好书，以及如何构建SaaS，如何避免过度设计，如何设计API，如何用程序打日志等方面资料。

希望这些内容对你有帮助。从下一篇文章开始，我们将进入《程序员练级攻略（2018）》的第五个篇章——高手成长篇。敬请期待。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 -[.....](#)



这一篇章，是本文最长的一篇，其中包括了如下的内容。

- **系统底层相关。**主要是以Linux系统为主，其中有大量的文章可以让你学习到Linux内核，以及内存、网络、异步I/O模型、Lock-free的无锁编程，还有其它和系统底层相关的东西。注意，系统底层要是深下去是可以完全不见底的。而且内存方面的知识也是比较多的，所以，这里还是主要给出一些非常有价值的基础性的知识和技术。学好这些东西，你会对系统有很深的理解，而且可以把这些知识反哺到架构设计上来。
- **数据库相关。**数据库方面主要是MySQL和各种开源NoSQL的一些相关的有价值的文章和导读，主要是让你对这些数据库的内在有一定的了解，但又不会太深。真正的深入是需要扎入到源代码中的。需要说明的是，这块技术不是我的长项，但又是每个架构师需要知道的，所以，我在这里给的学习资源可能会比较浅，这点还希望你来补充和指正。
- **分布式架构。**这一部分是最长最多的。其中有架构入门、分布式理论中各种非常有价值的经典论文，然后是一些分布式工程设计方面的文章，其中包括设计模式和工程应用，最后还有各大公司的架构供参考。
- **微服务。**有了分布式架构理论和工程的基础，接下来是对微服务的学习。在这部分内容中，我会罗列几个介绍微服务架构非常系统的文章，然后比较一下微服务和SOA的差别，最后则是一些工程实践和最佳实践。
- **容器化和自动化运维。**在容器化和自动化运维中，主要是学习Docker和Kubernetes这两个自动化运维的杀手型技术。而不是Salt、Puppet、Chef和Ansible这样比较传统的工具。原因很简单，因为自动化部署根本不够，还需要对环境和运行时的管理和运维才够，而只有Docker和Kubernetes才是未来。所以，这里重点让你学习这两个技术，其中有很多文章需要一些系统底层的知识。
- **机器学习和人工智能。**机器学习和人工智能，也不是我的长项，我也只是一个入门者。这里，我主要给了一些基础性的知识，其中包括基本原理、图书、课程、文章和相关的算法。你顺着我画的这路走，不能说能成为一个人工智能专家，但成为一个机器学习的高级工程师甚至准专家还是可能的。
- **前端开发。**这里的前端主要是H5的前端了，这一节会带你学习一下前端开发所需要知道的基础知识，尤其是对前端开发语言JavaScript的学习，我花费了相当的篇幅列出了很多很经典的学习资料，必定会让你成为一个JavaScript高手。然后你还需要了解浏览器是怎样工作的，还有相关的网络协议和一些性能优化的技巧。最后则是JavaScript框架的学习，这里我只给了React.js和Vue.js，并通过React.js带出来函数式编程的学习。我虽然不是一个前端程序员，但是，我相信我这个后端程序员给出来的这组前端开发的学习资料和路径会比前端程序员更靠谱一些。
- **信息源。**最后，则是一些信息源，其中包括各大公司的技术Blog，还有相关的论文集散地。

另外，这里需要说明几点。

- 我假设你在前面已经打下了非常扎实的基础，但是要成为一个高手，基础知识只是一个地基，你还需要很多更为具体的技术。对我来说，就是看各种各样的文章、手册、论文、分享……**其实，学习到一定程度，就是要从书本中走出去，到社区里和大家一起学习，而且还需要自己找食吃了。所以，对于这里面的文章，有很多都是在罗列各种文章和资源，只是为你梳理信息源，而不是喂你吃饭。**
- **老实说，我已经为你梳理并过滤掉了很多的信息，这里只留下了30%我觉得最经济也最有价值的信息。**虽然对于不同定位和不同需求的人还可以再对这些信息进行删减，但是觉得我这么一做就会对其他人不公平了。所以，这也是我觉得最小数量集的信息和资源吧。**你也可以把我这里的东西当成一个索引来对待。**
- 这些内容，不能说是隔离开来的，应该说是相辅相成的。也没什么顺序，可以各取所需。虽然看上去内容很多，但你也别害怕，真的不用害怕，你会越学越快，越实践越有感觉，也越有效率。一开始可能会很慢，但是坚持住，积累一段时间后就会越来越快的。而且，我要告诉你，绝大多数人是坚持不下来的。只要你能坚持下来，我保证，你一定会成为各个大公司的抢手货，这点你一定要相信我。**你不需要特别努力，只需要日进一步，3-5年后，你就会发现，绝大多数人都在你身后很远的地方了。**

今天分享的内容为系统底层知识中的Linux系统、内存和网络等方面的相关知识及推荐的学习资料。

Linux系统相关

学习Linux操作系统的原理是通向系统工程师的必经之路。我觉得，Unix/Linux操作系统里的东西并不难学。你千万不要一下子扎到源代码里去，那样没用——你还是要在上层先通过读一些不错的文档来学习。下面我罗列了一些很不错的站点，其中有很多内容供你去钻研和探索。

我在这里默认你前面已经读过并读懂了我推荐的那些和Unix/Linux相关的图书了。所以，我相信你对Unix/Linux下的编程已经是有一些基础了，因此，你继续深挖Linux下的这些知识应该也不是很难的事了。

- [Red Hat Enterprise Linux文档](#)。Red Hat Enterprise Linux (RHEL) 是老牌Linux厂商Red Hat出品的面向商业的Linux发行版。Red Hat网站上的这个文档中有很多很有价值的内容，值得一看。
- [Linux Insides](#)，GitHub上的一个开源电子书，其中讲述了Linux内核是怎样启动、初始化以及进行管理的。
- [LWN's kernel page](#)，上面有很多非常不错的文章来解释Linux内核的一些东西。
- [Learn Linux Kernel from Android Perspective](#)，从Android的角度来学习Linux内核，这个站点上的Blog相对于前面的比较简单易读一些。
- [Linux Kernel Doc](#)，Linux的内核文档也可以浏览一下。
- [Kernel Planet](#)，Linux内核开发者的Blog，有很多很不错的文章和想法。
- [Linux Performance and Tuning Guidelines](#)，这是IBM出的红皮书，虽然有点老了，但还是非常值得一读的。
- [TLK: The Linux Kernel](#)，这是一本相对比较老的书了，Linux内核版本为2.0.33，但了解一下前人的思路，也是很有帮助的。
- [Linux Performance](#)，这个网站上提供了和Linux系统性能相关的各种工具和文章收集，非常不错。
- [Optimizing web servers for high throughput and low latency](#)，這是一篇非常底层的系统调优的文章，来自DropBox，从中你可以学到很多底层的性能调优的经验和知识。

内存相关

计算机内存管理是每一个底层程序员需要了解的非常重要的事儿。当然，这里我们重点还是Linux操作系统相关的内存管理上的知识。

首先，LWN.net上有一系列的“**What every programmer should know about memory**”文章你需要读一下。当然，你可以直接访问一个完整的[PDF文档](#)。下面是这个系列文章的网页版列表。读完这个列表的内容，你基本上就对内存有了一个比较好的知识体系了。

- [Part 1: Introduction](#)，中译版为“[每个程序员都应该了解的内存知识【第一部分】](#)”
- [Part 2: CPU caches](#)
- [Part 3 \(Virtual memory\)](#)
- [Part 4 \(NUMA systems\)](#)
- [Part 5 \(What programmers can do - cache optimization\)](#)
- [Part 6 \(What programmers can do - multi-threaded optimizations\)](#)
- [Part 7 \(Memory performance tools\)](#)
- [Part 8 \(Future technologies\)](#)
- [Part 9 \(Appendices and bibliography\)](#)

然后是几篇和内存相关的论文。下面这三篇论文是我个人觉得能对你非常有帮助的文章，尤其是你要做一些程序的性能优化方面。

- [Memory Barriers: a Hardware View for Software Hackers](#)。内存的读写屏障是线程并发访问共享的内存数据时，从程序本身、编译器到CPU都必须遵循的一个规范。有了这个规范，才能保证访问共享的内存数据时，一个线程对该数据的更新能被另一个线程以正确的顺序感知到。在SMP（对称多处理）这种类型的多处理器系统（包括多核系统）上，这种读写屏障还包含了复杂的缓存一致性策略。这篇文章做了详细解释。
- [A Tutorial Introduction to the ARM and POWER Relaxed Memory Models](#)，对ARM和POWER的宽松内存模型的一个教程式的简介。本篇文章的焦点是ARM和POWER体系结构下多处理器系统内存并发访问一致性的设计思路和使用方法。与支持较强的TSO模型的x86体系结构不同，ARM和POWER这两种体系结构出于对功耗和性能的考虑，使用了一种更为宽松的内存模型。本文详细讨论了ARM和POWER的模型。

- [x86-TSO: A Rigorous and Usable Programmer's Model for x86 Multiprocessors](#), 介绍x86的多处理器内存并发访问的一致性模型TSO。

接下来是开发者最关心的内存管理方面的lib库。通常来说，我们有三种内存分配管理模块。就目前而言，BSD的jemalloc有很大的影响力。后面我们可以看到不同公司的实践性文章。

- [ptmalloc](#) 是glibc的内存分配管理。

- [tcmalloc](#) 是Google的内存分配管理模块，全称是Thread-Caching malloc，基本上来说比glibc的ptmalloc快两倍以上。

- [jemalloc](#) 是BSD提供的内存分配管理。其论文为 [A Scalable Concurrent malloc\(3\) Implementation for FreeBSD](#)，这是一个可以并行处理的内存分配管理器。

关于 C 的这些内存分配器，你可以参看Wikipedia的 "[C Dynamic Memory Allocation](#)"这个词条。

下面是几篇不错的文章，让你感觉一下上面那三种内存分配器的一些比较和工程实践。

- [ptmalloc, tcmalloc和jemalloc内存分配策略研究](#)
- [内存优化总结: ptmalloc, tcmalloc和jemalloc](#)
- [Scalable memory allocation using jemalloc](#)
- [Decreasing RAM Usage by 40% Using jemalloc with Python & Celery](#)

计算机网络

网络学习

首先，推荐一本书——[《计算机网络（第五版）》](#)，这本“计算机网络”和前面推荐的那本计算机网络不一样，前面那本偏概论，这本中有很多细节。这本书是国内外使用最广泛、最权威的计算机网络经典教材。全书按照网络协议模型自下而上（物理层、数据链路层、介质访问控制层、网络层、传输层和应用层）有系统地介绍了计算机网络的基本原理，并结合Internet给出了大量的协议实例。

这本书还与时俱进地引入了最新的网络技术，包括无线网络、3G蜂窝网络、RFID与传感器网络、内容分发与P2P网络、流媒体传输与IP语音，以及延迟容忍网络等。另外，本书针对当前网络应用中日益突出的安全问题，用了一整章的篇幅对计算机网络安全进行了深入讨论，而且把相关内容与最新网络技术结合起来阐述。这本书读起来并不枯燥，因为其占有很多小故事和小段子。

然后，有两个网上的教程和讲义也可以让人入门。

- 渥太华大学的一个课程讲义你也可以一看 [Computer Network Design](#)。
- GeeksforGeeks 上也有一个简单的 [Computer Network Tutorials](#)。

网络调优

接下来，你可能需要一些非常实用的可以操作的技术，下面的几篇文章相信可以帮助到你。

- 《Linux的高级路由和流量控制 HowTo》 ([Linux Advanced Routing & Traffic Control HOWTO](#))，这是一个非常容易上手的关于 iproute2、流量整形和一点 netfilter 的指南。
- 关于网络调优，你可以看一下这个文档 [Red Hat Enterprise Linux Network Performance Tuning Guide](#)。
- 还有一些网络工具能够帮上你的大忙，这里有一个网络工具的Awesome列表 [Awesome Pcap Tools](#)，其中罗列了各种网络工具，能够让你更从容地调试网络相关的程序。
- [Making Linux TCP Fast](#)，一篇非常不错的TCP调优的论文。
- 下面是在PackageCloud上的两篇关于Linux网络栈相关的底层文章，非常值得一读。
 - [Monitoring and Tuning the Linux Networking Stack: Receiving Data](#)
 - [Monitoring and Tuning the Linux Networking Stack: Sending Data](#)

网络协议

接下来，想要学习网络协议最好的方式就是学习通讯相关的RFC。所以，在这里我会推荐一系列值得读的RFC给你。读RFC有几个好处，一方面可以学习技术，另一方面，你可以通过RFC学习到一个好的技术文档是怎么写的，还能看到各种解决问题的方案和思路。

对于第2层链路层，你可能需要了解一下ARP：

- [RFC 826 - An Ethernet Address Resolution Protocol](#)

以及Tunnel相关的协议：

- [RFC 1853 - IP in IP Tunneling](#)
- [RFC 2784 - Generic Routing Encapsulation \(GRE\)](#)
- [RFC 2661 - Layer Two Tunneling Protocol "L2TP"](#)
- [RFC 2637 - Point-to-Point Tunneling Protocol \(PPTP\)](#)

对于第4层，你最需要了解的是TCP/IP了。和TCP相关的RFC相当多，这里给一系列经典的RFC。这些RFC我都引用在了我在CoolShell上的《[TCP的那些事儿（上）](#)》和《[TCP的那些事儿（下）](#)》两篇文章中。如果你看不懂RFC，你也可以去看我上述的文章。

- [RFC 793 - Transmission Control Protocol](#) - 最初的TCP标准定义，但不包括TCP相关细节。
- [RFC 813 - Window and Acknowledgement Strategy in TCP](#) - TCP窗口与确认策略，并讨论了在使用该机制时可能遇到的问题及解决方法。
- [RFC 879 - The TCP Maximum Segment Size and Related Topics](#) - 讨论MSS参数对控制TCP分组大小的重要性，以及该参数与IP分段大小的关系等。
- [RFC 896 - Congestion Control in IP/TCP Internetworks](#) - 讨论拥塞问题和TCP如何控制拥塞。
- [RFC 2581 - TCP Congestion Control](#) - 描述用于拥塞控制的四种机制：慢启动、拥塞防御、快重传和快恢复。后面这个RFC被 [RFC 5681](#) 所更新。还有 [RFC 6582 - The NewReno Modification to TCP's Fast Recovery Algorithm](#) 中一个改进的快速恢复算法。
- [RFC 2018 - TCP Selective Acknowledgment Options](#) - TCP的选择确认。
- [RFC 2883 - An Extension to the Selective Acknowledgement \(SACK\) Option for TCP](#) - 对于RFC 2018的改进。
- [RFC 2988 - Computing TCP's Retransmission Timer](#) - 讨论与TCP重传计时器设置相关的话题，重传计时器控制报文在重传前应等待多长时间。也就是经典的TCP Karn/Partridge重传算法。
- [RFC 6298 - Computing TCP's Retransmission Timer](#) - TCP Jacobson/Karels Algorithm重传算法。

我个人觉得TCP最牛的不是不丢包，而是拥塞控制。对此，如果你感兴趣，可以读一下经典论文《[Congestion Avoidance and Control](#)》。

关于Linux下的TCP参数，你需要仔仔细细地读一下[TCP的man page](#)。

对于第7层协议，HTTP协议是重点要学习的。

首先推荐的是《[HTTP权威指南](#)》，这本书有点厚，可以当参考书来看。这本书中没有提到HTTP/2的事，但是可以让你了解到HTTP协议的绝大多数特性。

HTTP 1.1的原始RFC是1999年6月的 [RFC 2616](#)，但其在2014后很快被下面这些RFC给取代了。

- [RFC 7230 - Hypertext Transfer Protocol \(HTTP/1.1\): Message Syntax and Routing](#)
- [RFC 7231 - Hypertext Transfer Protocol \(HTTP/1.1\): Semantics and Content](#)

- [RFC 7232 - Hypertext Transfer Protocol \(HTTP/1.1\): Conditional Requests](#)
- [RFC 7233 - Hypertext Transfer Protocol \(HTTP/1.1\): Range Requests](#)
- [RFC 7234 - Hypertext Transfer Protocol \(HTTP/1.1\): Caching](#)
- [RFC 7235 - Hypertext Transfer Protocol \(HTTP/1.1\): Authentication](#)

关于[HTTP/2](#)，这是HTTP的一个比较新的协议，它于2015年被批准通过，现在基本上所有的主流浏览器都默认启用这个协议。所以，你有必要学习一下这个协议。下面是相关的学习资源。

- [Gitbook - HTTP/2详解](#)
- [http2 explained \(中译版\)](#)
- [HTTP/2 for a Faster Web](#)
- [Nginx HTTP/2 白皮书](#)
- HTTP/2 的两个RFC:
 - [RFC 7540 - Hypertext Transfer Protocol Version 2 \(HTTP/2\)](#)，HTTP/2的协议本身
 - [RFC 7541 - HPACK: Header Compression for HTTP/2](#)，HTTP/2的压缩算法

最后，你可以上Wikipedia的[Internet Protocol Suite](#) 上看看，这是一个很不错的网络协议的词条汇集地。顺着这些协议，你可以找到很多有用的东西。

小结

好了，总结一下今天的内容。这是程序员练级攻略2018版第五篇章——高手成长篇的第一篇文章。前面的内容先介绍了一些这一系列内容的总体构成，及每一部分的学习重点。后面是这一篇章第一个主题系统底层知识中的部分内容，即Linux系统、内存和计算机网络，并给出了相应的学习资料。

我认为，学习到一定程度，就是要从书本中走出去，到社区里和大家一起学习，而且还需要自己找食吃了。所以，这篇文章中，我罗列了各种文章和资源，并给出了简短的推荐语言，就是在为你梳理信息源，而不是喂你吃饭。我更希望看到你自趋地成长。

下篇文章中，我们分享的内容为系统底层知识中的异步I/O模型、Lock-Free编程以及其他一些相关的知识点和学习资源。敬请期待。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络 \(系统底层知识\)](#)
 - [异步I/O模型和Lock-Free编程 \(系统底层知识\)](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门 \(分布式架构\)](#)
 - [分布式架构经典图书和论文 \(分布式架构\)](#)
 -



异步I/O模型

异步I/O是我个人觉得所有程序员都必需要学习的一门技术或是编程方法，这其中的设计模式或是解决方法可以借鉴到分布式架构上来。再说一遍，学习这些模型，是非常非常重要的，你千万要认真学习。

史蒂文斯（Stevens）在《[UNIX网络编程](#)》一书6.2 I/O Models中介绍了五种I/O模型。

- 阻塞I/O
- 非阻塞I/O
- I/O的多路复用（select和poll）
- 信号驱动的I/O（SIGIO）
- 异步I/O（POSIX的aio_functions）

然后，在前面我们也阅读过了 - [C10K Problem](#)。相信你对I/O模型也有了一定的了解。这里，我们需要更为深入地学习I/O模型，尤其是其中的异步I/O模型。

首先，我们看一篇和Java相关的I/O模型的文章来复习一下之前的内容。[Thousands of Threads and Blocking I/O: The Old Way to Write Java Servers Is New Again \(and Way Better\)](#)，这个PPT中不仅回顾和比较了各种I/O模型，而且还有各种比较细节的方案和说明，是一篇非常不错的文章。

然后，你可以看一篇Java相关的PPT - 道格·莱亚（Doug Lea）的 [Scalable IO in Java](#)，这样你会对一些概念有个了解。

接下来，我们需要了解一下各种异步I/O的实现和设计方式。

- [IBM - Boost application performance using asynchronous I/O](#)，这是一篇关于AIO的文章。
- [Lazy Asynchronous I/O For Event-Driven Servers](#)，这篇文章也很不错。
- 另外，异步I/O模型中的 [Windows I/O Completion Ports](#)，你也需要了解一下。如果MSDN上的这个手册不容易读，你可以看看这篇文章 [Inside I/O Completion Ports](#)。另外，关于 Windows，[Windows Internals](#) 这本书你可以仔细读一下，非常不错的。其中有一节I/O Processing也是很不错的，这里我给一个网上免费的链接[I/O Processing](#) 你可以看看Windows是怎么玩的。
- 接下来是Libevent。你可以看一下其主要维护人员尼克·马修森（Nick Mathewson）写的 [libevent 2.0 book](#)。还有一本国人写的电子书《[Libevent深入浅出](#)》。
- 再接下来是Libuv。你可以看一下其官网的 [Libuv Design Overview](#) 了解一下。

我简单总结一下，基本上来说，异步I/O模型的发展技术是：select -> poll -> epoll -> aio -> libevent -> libuv。Unix/Linux用了好几十年走过这些技术的变迁，然而，都不如Windows I/O Completion Port 设计得好（免责声明：这个观点纯属个人观点。相信你仔细研究这些I/O模型后，你会得到你自己的判断）。

看过这些各种异步I/O模式的实现以后，相信你会看到一个编程模式——Reactor模式。下面是这个模式的相关文章（读这三篇就够了）。

- [Understanding Reactor Pattern: Thread-Based and Event-Driven](#)
- [Reactor Pattern](#)
- [The reactor pattern and non-blocking IO](#)

然后是几篇有意思的延伸阅读文章。

- [The Secret To 10 Million Concurrent Connections -The Kernel Is The Problem, Not The Solution](#) - C10M问题来了.....

- 还有几篇可能有争议的文章，让你从不同的角度思考。

- [Select is fundamentally broken](#)
- [Epoll is fundamentally broken 1/2](#)
- [Epoll is fundamentally broken 2/2](#)

Lock-Free编程相关

Lock-Free - 无锁技术越来越被开发人员重视，因为锁对于性能的影响实在是太大了，所以如果想开发出一个高性能的程序，你就非常有必要学习 Lock-Free的编程方式。

关于无锁的数据结构，有几篇教程你可以看一下。

- [Dr.Dobb's: Lock-Free Data Structures](#)
- [Andrei Alexandrescu: Lock-Free Data Structures](#)

然后强烈推荐一本免费的电子书：[Is Parallel Programming Hard, And, If So, What Can You Do About It?](#)，这是大牛 [保罗·麦肯尼（Paul E. McKenney）](#) 写的书。这本书堪称并行编程的经典书，必看。

此时，Wikipedia上有三个词条你要看一下，以此了解并发编程中的一些概念：[Non-blocking algorithm](#)、[Read-copy-update](#) 和 [Seqlock](#)。

接下来，读一下以下两篇论文。

- [Implementing Lock-Free Queues](#)，这也是一篇很不错的论文，我把它介绍在我的网站上，文章为“[无锁队列的实现](#)”。
- [Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms](#)，这篇论文给出了一个无阻塞和阻塞的并发队列算法。

最后，有几个博客你要订阅一下。

- [1024cores](#) - 德米特里·伐由科夫（Dmitry Vyukov）的和 lock-free 编程相关的网站。
- [Paul E. McKenney](#) - 保罗（Paul）的个人网站。
- [Concurrency Freaks](#) - 关于并发算法和相关模式的网站。
- [Preshing on Programming](#) - 加拿大程序员杰夫·普莱辛（Jeff Preshing）的技术博客，主要关注C++和Python两门编程语言。他用C++11实现了类的反射机制，用C++编写了3D小游戏 Hop Out，还为该游戏编写了一个游戏引擎。他还讨论了很多C++的用法，比如C++14推荐的代码写法、新增的某些语言构造等，和Python很相似。阅读这个技术博客上的内容能够深深感受到博主对编程世界的尊敬和痴迷。
- [Sutter's Mill](#) - 赫布·萨特（Herb Sutter）是一位杰出的C++专家，曾担任ISO C++标准委员会秘书和召集人超过10年。他的博客有关于C++语言标准最新进展的信息，其中也有他的演讲视频。博客中还讨论了其他技术和C++的差异，如C#和JavaScript，它们的性能特点、怎样避免引入性能方面的缺陷等。
- [Mechanical Sympathy](#) - 博主是马丁·汤普森（Martin Thompson），他是一名英国的技术极客，探索现代硬件的功能，并提供开发、培训、性能调优和咨询服务。他的博客主题是 Hardware and software working together in harmony，里面探讨了如何设计和编写软件使得它在硬件上能高性能地运行。非常值得一看。

接下来，是一些编程相关的一些C/C++的类库，这样你就不用从头再造轮子了（对于Java的，请参看JDK里的Concurrent开头的一系列的类）。

- [Boost.Lockfree](#) - Boost库中的无锁数据结构。
- [ConcurrencyKit](#) - 并发性编程的原语。
- [Folly](#) - Facebook的开源库（它对MPMC队列做了一个很好的实现）。

- [Junction](#) - C++中的并发数据结构。
- [MPMCQueue](#) - 一个用C++11编写的有边界的“多生产者-多消费者”无锁队列。
- [SPSCQueue](#) - 一个有边界的“单生产者-单消费者”的无等待、无锁的队列。
- [Seqlock](#) - 用C++实现的Seqlock。
- [Userspace RCU](#) - liburcu是一个用户空间的RCU (Read-copy-update, 读-拷贝-更新) 库。
- [libcds](#) - 一个并发数据结构的C++库。
- [liblfd](#) - 一个用C语言编写的可移植、无许可证、无锁的数据结构库。

其它

- 关于64位系统编程，只要去一个地方就行了：[All about 64-bit programming in one place](#)，这是一个关于64位编程相关的收集页面，其中包括相关的文章、28节课程，还有知识库和相关的blog。

- [What Scalable Programs Need from Transactional Memory](#)，事务性内存 (TM) 一直是许多研究的重点，它在诸如IBM Blue Gene/Q和Intel Haswell等处理器中得到了支持。许多研究都使用STAMP基准测试套件来评估其设计。然而，我们所知的所有TM系统上的STAMP基准测试所获得的加速比较有限。

例如，在IBM Blue Gene/Q上有64个线程，我们观察到使用Blue Gene/Q硬件事务内存 (HTM) 的中值加速比为1.4倍，使用软件事务内存 (STM) 的中值加速比为4.1倍。什么限制了这些TM基准的性能？在本文中，作者认为问题在于用于编写它们的编程模型和数据结构上，只要使用合适的模型和数据结构，程序的性能可以有10多倍的提升。

- [Improving OpenSSL Performance](#)，这篇文章除了教你如何提高OpenSSL的执行性能，还讲了一些底层的性能调优知识。

- 关于压缩的内容。为了避免枯燥，主要推荐下面这两篇实践性很强的文章。

- [How eBay's Shopping Cart used compression techniques to solve network I/O bottlenecks](#)，这是一篇很好的文章，讲述了eBay是如何通过压缩数据来提高整体服务性能的，其中有几个比较好的压缩算法。除了可以让你学到相关的技术知识，还可以让你看到一种比较严谨的工程师文化。

- [Linkedin: Boosting Site Speed Using Brotli Compression](#)，Linkedin在2017年早些时候开始使用 [Brotli](#) 来替换 gzip，以此带来更快的访问，这篇文章讲述了什么是Brotli以及与其它压缩程序的比较和所带来的性能提升。

- 这里有两篇关于SSD硬盘性能测试的文章。[Performance Testing with SSDs, Part 1](#) 和 [Performance Testing with SSDs Part 2](#)，这两篇文章介绍了测试SSD硬盘性能以及相关的操作系统调优方法。

- [Secure Programming HOWTO - Creating Secure Software](#)，这是一本电子书，其中有繁体中文的翻译，这本电子书讲了Linux/Unix下的一些安全编程方面的知识。

相关论文

- [Hints for Computer System Design](#)，计算机设计的忠告，这是ACM图灵奖得主 [Butler Lampson](#) 在Xerox PARC工作时的一篇论文。这篇论文简明扼要地总结了他在做系统设计时的一些想法，非常值得一读。（用他的话来说，“Studying the design and implementation of a number of computer has led to some general hints for system design. They are described here and illustrated by many examples, ranging from hardware such as the Alto and the Dorado to application programs such as Bravo and Star”。）

- [The 5 minute rule for trading memory for disc accesses and the 5 byte rule for trading memory for CPU time](#)，根据文章名称也可以看出，5分钟法则是用来衡量内存与磁盘的，而5字节法则则是在内存和CPU之间的权衡。这两个法则是Jim Gray和Franco Putzolu在1986年的文章。

在该论文发表10年后的1997年，Jim Gray和Goetz Graefe 又在 [The Five-Minute Rule Ten Years Later and Other Computer Storage Rules of Thumb](#) 中对该法则进行了重新审视。2007年，也就是该论文发表20年后，这年的1月28日，Jim Gray驾驶一艘40英尺长的船从旧金山港出海，目的是航行到附近的费拉隆岛，在那里撒下母亲的骨灰。出海之后，他就同朋友和亲属失去了联系。为了纪念和向大师致敬，时隔10多年后的2009年Goetz Graefe又发表了 [The Five-Minute Rule 20 Years Later \(and How False Memory Changes the Rules\)](#)。

注明一下，Jim Gray，关系型、数据库领域大师。因在数据库和事务处理研究和实现方面的开创性贡献而获得1998年图灵奖。美国科学院、工程院两院院士，ACM和IEEE两会会士。他25岁成为加州大学伯克利分校计算机科学学院第一位博士。在IBM工作期间参与和主持了IMS、System R、SQL / DS、DB2等项目的开发。后任职于微软研究院，主要关注应用数据库技术来处理各学科的海量信息。

小结

好了，总结一下今天的内容。异步I/O模型是我个人觉得所有程序员都必需要学习的一门技术或是编程方法，这其中的设计模式或是解决方法可以借鉴到分布式架构上来。而且我认为，学习这些模型非常重要，你千万要认真学习。

接下来是Lock-Free方面的内容，由于锁对于性能的影响实在是太大了，所以它越来越被开发人员所重视。如果想开发出一个高性能的程序，你非常有必要学习 Lock-Free的编程方式。随后，我给出系统底层方面的其它一些重要知识，如64位编程、提高OpenSSL的执行性能、压缩、SSD硬盘性能测试等。最后介绍了几篇我认为对学习和巩固这些知识非常有帮助的论文，都很经典，推荐你务必看看。

下篇文章是数据库方面的内容，我们将探讨各种类型的数据库，非常有意思。敬请期待。

下面是《程序员进阶攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 -



一耳听风

洞悉技术的本质
享受科技的乐趣



陈皓

资深技术专家
骨灰级程序员



扫码订阅

前两篇文章分享的是系统底层方面的内容，今天我们进入高手成长篇的第二部分——Java底层知识。

Java 字节码相关

首先，Java最黑科技的玩法就是字节码编程，也就是动态修改或是动态生成Java字节码。Java的字节码相当于汇编，其中的一些细节你可以从下面的这几个教程中学习。

- [Java Zone: Introduction to Java Bytecode](#)，这篇文章图文并茂地向你讲述了Java字节码的一些细节，是一篇很不错的入门文章。
- [IBM DeveloperWorks: Java bytecode](#)，虽然这篇文章很老了，但是这篇文章是一篇非常好的讲Java字节码的文章。
- [Java Bytecode and JVMTI Examples](#)，这是一些使用[JVM Tool Interface](#)操作字节码的比较实用的例子。包括方法调用统计、静态字节码修改、Heap Tagging和Heap Walking。

当然，一般来说，我们不使用JVMTI操作字节码，而是用一些更好用的库。这里有三个库可以帮你比较容易地做这个事。

- [asmtools](#) - 用于生产环境的Java .class文件开发工具。
- [Byte Buddy](#) - 代码生成库：运行时创建Class文件而不需要编译器帮助。
- [JitScript](#) - 和[BiteScript](#)类似的字节码生成库。

就我而言，我更喜欢Byte Buddy，它在2015年还获得了Oracle的“[Duke's Choice](#)”大奖，其中说Byte Buddy极大地发展了Java的技术。

使用字节码编程可以玩出很多高级玩法，最高级的还是在Java程序运行时进行字节码修改和代码注入。听起来是不是一些很黑客，也很黑科技的事？是的，这种方式使用Java这门静态语言在运行时可以进行各种动态的代码修改，而且可以进行无侵入的编程。

比如，我们不需要在代码中埋点做统计或监控，可以使用这种技术把我们的监控代码直接以字节码的方式注入到别人的代码中，从而实现对实际程序运行情况进行统计和监控。如果你看过我的《编程范式游记》，你就知道这种技术的威力了，其可以很魔法地把业务逻辑和代码控制分离开来。

要做到这个事，你还需要学习一个叫Java Agent的技术。Java Agent使用的是“[Java Instrumentation API](#)”，其主要方法是实现一个叫的方法（嗯，一个比 main() 函数还要超前执行的 main 函数），然后把你的代码编译成一个jar文件。

在JVM启动时，使用这样的命令行来引入你的jar文件：`java -javaagent:yourAwesomeAgent.jar -jar App.jar`。更为详细的文章你可以参看：“[Java Code Geeks: Java Agents](#)”，你还可以看一下这个示例项目：[jvm-monitoring-agent](#) 或是 [EntryPointKR/Agent.java](#)。如果想用ByteBuddy来玩，你可以看看这篇文章“[通过使用Byte Buddy，便捷地创建Java Agent](#)”。如果你想学习如何用Java Agent做监控，你可以看一下这个项目 [Stage Monitor](#)。

JVM 相关

接下来讲讲Java底层知识中另一个非常重要的内容——JVM。

说起JVM，你有必要读一下JVM的规格说明书，我在这里放一个Java 8的，[The Java Virtual Machine Specification Java SE 8 Edition](#)。对于规格说明书的阅读，我认为是系统了解JVM规范的最佳文档，这个文档可以让你对于搞不清楚或是诡异的问题恍然大悟。关于中文翻译，有人在GitHub上开了个Repo - “[java-virtual-machine-specification](#)”。

另外，也推荐一下 [JVM Anatomy Park](#) JVM解剖公园，这是一个系列的文章，每篇文章都不长，但是都很精彩，带你一点一点地把JVM中的一些技术解开。

学习Java底层原理还有Java的内存模型，官方文章是 [JSR 133](#)。还有马里兰大学的威廉·皮尤（William Pugh）教授收集的和Java内存模型相关的文献 - [The Java Memory Model](#)，你可以前往浏览。

对于内存方面，道格·利（Doug Lea）有两篇文章也是很有价值的。

- [The JSR-133 Cookbook for Compiler Writers](#)，解释了怎样实现Java内存模型，特别是在考虑到多处理器（或多核）系统的情况下，多线程和读写屏障的实现。
- [Using JDK 9 Memory Order Modes](#)，讲了怎样通过VarHandle来使用plain、opaque、release/acquire和volatile四种共享内存的访问模式，并剖析了底层的原理。

垃圾回收机制也是需要好好学习的，在这里推荐一本书《[The Garbage Collection Handbook](#)》，在豆瓣上的得分居然是9.9（当然，评价人数不多）。这本书非常全面地介绍了垃圾收集的原理、设计和算法。但是这本书也是相当难啃的。中文翻译《[垃圾回收算法手册](#)》翻译得很一般，有人说翻译得很烂。所以，如果可能，还是读英文版的。如果你对从事垃圾回收相关的工作有兴趣，那么你需要好好看一下这本书。

当然，更多的人可能只需要知道怎么调优垃圾回收，那么推荐阅读 [Garbage Collection Tuning Guide](#)，它是Hotspot Java虚拟机的垃圾回收调优指南，对你很有帮助。

[Quick Tips for Fast Code on the JVM](#) 也是一篇很不错的文章，里面有写出更快的Java代码的几个小提示，值得一读。

小结

好了，总结一下今天学到的内容。Java最黑科技的玩法就是字节码编程，也就是动态修改或是动态生成Java字节码。Java的字节码相当于汇编，学习其中的细节很有意思，为此我精心挑选了3篇文章，供你学习。我们一般不使用JVMTI操作字节码，而是用一些更好用的库，如asmtools、Byte Buddy和BiteScript等。使用字节码编程可以玩出很多高级玩法，其中最高级的玩法是在Java程序运行时进行字节码修改和代码注入。同时，我介绍了Java Agent技术，帮助你更好地实现这种高级玩法。

JVM也是学习Java过程中非常重要的一部分内容。我推荐阅读一下JVM的规格说明书，我认为，它是系统了解JVM规范的最佳文档，可以让你对于搞不清楚或是诡异的问题恍然大悟。同时推荐了 [JVM Anatomy Park](#) 系列文章，也非常值得一读。

随后介绍的是Java的内存模型和垃圾回收机制，尤其给出了如何调优垃圾回收方面的资料。这些内容都很底层，但也都很重要。对于想成为高手的你来说，还是有必要花时间来啃一啃的。

下篇文章是数据库方面的内容，我们将探讨各种类型的数据库，非常有意思。敬请期待。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 -

左耳听风

洞悉技术的本质
享受科技的乐趣



陈皓

资深技术专家
骨灰级程序员



扫 码 订 阅

对于数据库方向，重点就是两种数据库，一种是以SQL为代表的关系型数据库，另一种是以非SQL为代表的NoSQL数据库。关系型数据库主要有三个：Oracle、MySQL 和 Postgres。

在这里，我们只讨论越来越主流的MySQL数据库。首先，我们要了解数据库的一些实现原理和内存的一些细节，然后我们要知道数据的高可用和数据复制这些比较重要的话题，了解一下关系型数据库的一些实践和难点。然后，我们会进入到NoSQL数据库的学习。

NoSQL数据库千奇百怪，其主要是解决了关系型数据库中的各种问题。第一个大问题是数据的Schema非常多，用关系型数据库来表示不同的Data Schema是非常笨拙的，所以要有不同的数据库（如时序型、键值对型、搜索型、文档型、图结构型等）。另一个大问题是，关系型数据库的ACID是一件很讨厌的事，这极大地影响了数据库的性能和扩展性，所以NoSQL在这上面做了相应的妥协以解决大规模伸缩的问题。

对于一个程序员，你可能觉得数据库的事都是DBA的事，然而我想告诉你你错了，这些事才真正是程序员的事。因为程序是需要和数据打交道的，所以程序员或架构师不仅需要设计数据模型，还要保证整体系统的稳定性和可用性，数据是整个系统中关键中的关键。所以，作为一个架构师或程序员，你必须了解最重要的数据存储——数据库。

关系型数据库

今天，关系型数据库最主要的两个代表是闭源的Oracle和开源的MySQL。当然，还有很多，比如微软的SQL Server，IBM的DB2等，还有开源的PostgreSQL。关系型数据库的世界中有好多好多产品。当然，还是Oracle和MySQL是比较主流的。所以，这里主要介绍更为开放和主流的MySQL。

如果你要玩Oracle，我这里只推荐一本书《[Oracle Database 9i/10g/11g编程艺术](#)》，无论是开发人员还是DBA，它都是必读的书。这本书的作者是Oracle公司的技术副总裁托马斯·凯特（Thomas Kyte），他也是世界顶级的Oracle专家。

这本书深入分析了Oracle数据库体系结构，包括文件、内存结构以及构成Oracle数据库和实例的底层进程，利用具体示例讨论了一些重要的数据库主题，如锁定、并发控制、事务等。同时分析了数据库中的物理结构，如表、索引和数据类型，并介绍采用哪些技术能最优化地使用这些物理结构。

- 学习MySQL，首先一定是要看[MySQL 官方手册](#)。

- 然后，官方还有几个PPT也要学习一下。

- [How to Analyze and Tune MySQL Queries for Better Performance](#)
- [MySQL Performance Tuning 101](#)
- [MySQL Performance Schema & Sys Schema](#)
- [MySQL Performance: Demystified Tuning & Best Practices](#)
- [MySQL Security Best Practices](#)
- [MySQL Cluster Deployment Best Practices](#)
- [MySQL High Availability with InnoDB Cluster](#)

- 然后推荐《[高性能MySQL](#)》，这本书是MySQL领域的经典之作，拥有广泛的影响力。不但适合数据库管理员（DBA）阅读，也适合开发人员参考学习。不管是数据库新手还是专家，都能从本书中有所收获。

- 如果你对MySQL的内部原理有兴趣的话，可以看一下这本书《[MySQL技术内幕：InnoDB存储引擎](#)》。当然，还有官网的[MySQL Internals Manual](#)。

- 数据库的索引设计和优化也是非常关键的，这里还有一本书《[数据库的索引设计与优化](#)》也是很不错的。虽然不是讲MySQL的，但是原理都是相通的。这也是上面推荐过的《高性能MySQL》在其索引部分推荐的一本好书。

你千万不要觉得只有做数据库你才需要学习这种索引技术。不是的！在系统架构上，在分布式架构中，索引技术也是非常重要的。这本书对于索引性能进行了非常清楚的估算，不像其它书中只是模糊的描述，你一定会收获很多。

下面还有一些不错的和MySQL相关的文章。

- [MySQL索引背后的数据结构及算法原理](#)
- [Some study on database storage internals](#)
- [Sharding Pinterest: How we scaled our MySQL fleet](#)
- [Guide to MySQL High Availability](#)
- [Choosing MySQL High Availability Solutions](#)
- [High availability with MariaDB TX: The definitive guide](#)

最后，还有一个MySQL的资源列表 [Awesome MySQL](#)，这个列表中有很多的工具和开发资源，可以帮助你做很多事。

MySQL有两个比较有名的分支，一个是Percona，另一个是MariaDB，其官网上Resources页面中有很多不错的资源和文档，可以经常看看。[Percona Resources](#)、[MariaDB Resources](#)，以及它们的开发博客中也有很多不错的文章，分别为[Percona Blog](#) 和 [MariaDB Blog](#)。

然后是关于MySQL的一些相关经验型的文章。

- [Booking.com: Evolution of MySQL System Design](#)，Booking.com的MySQL数据库使用的演化，其中有很多不错的经验分享，我相信也是很多公司会遇到的问题。
- [Tracking the Money-Scaling Financial Reporting at Airbnb](#)，Airbnb的数据库扩展的经验分享。
- [Why Uber Engineering Switched from Postgres to MySQL](#)，无意比较两个数据库谁好谁不好，推荐这篇Uber的长文，主要是想让你从中学习到一些经验和技巧，这是一篇很不错的文章。

关于MySQL的集群复制，下面有这些文章供你学习一下，都是很不错的实践性比较强的文章。

- [Monitoring Delayed Replication, With A Focus On MySQL](#)
- [Mitigating replication lag and reducing read load with freno](#)
- 另外，Booking.com给了一系列的文章，你可以看看：
 - [Better Parallel Replication for MySQL](#)
 - [Evaluating MySQL Parallel Replication Part 2: Slave Group Commit](#)
 - [Evaluating MySQL Parallel Replication Part 3: Benchmarks in Production](#)
 - [Evaluating MySQL Parallel Replication Part 4: More Benchmarks in Production](#)
 - [Evaluating MySQL Parallel Replication Part 4, Annex: Under the Hood](#)

对于MySQL的数据分区来说，还有下面几篇文章你可以看看。

- [StackOverflow: MySQL sharding approaches?](#)

- [Why you don't want to shard](#)
- [How to Scale Big Data Applications](#)
- [MySQL Sharding with ProxySQL](#)

然后，再看看各个公司做MySQL Sharding的一些经验分享。

- [MailChimp: Using Shards to Accommodate Millions of Users](#)
- [Uber: Code Migration in Production: Rewriting the Sharding Layer of Uber's Schemaless Datastore](#)
- [Sharding & IDs at Instagram](#)
- [Airbnb: How We Partitioned Airbnb's Main Database in Two Weeks](#)

NoSQL数据库

关于NoSQL数据库，其最初目的就是解决大数据的问题。然而，也有人把其直接用来替换掉关系型数据库。所以在学习这个技术之前，我们需要对这个技术的一些概念和初衷有一定的了解。下面是一些推荐资料。

- Martin Fowler在YouTube上分享的NoSQL介绍 [Introduction To NoSQL](#)，以及他参与编写的 [NoSQL Distilled - NoSQL 精粹](#)，这本书才100多页，是本难得的关于NoSQL的书，很不错，非常易读。
- [NoSQL Databases: a Survey and Decision Guidance](#)，这篇文章可以带你自上而下地从CAP原理到开始了解NoSQL的种种技术，是一篇非常不错的文章。
- [Distribution, Data, Deployment: Software Architecture Convergence in Big Data Systems](#)，这是卡内基·梅隆大学的一篇讲分布式大数据系统的论文。其中主要讨论了在大数据时代下的软件工程中的一些关键点，也说到了NoSQL数据库。
- [No Relation: The Mixed Blessings of Non-Relational Databases](#)，这篇论文虽然有点年代久远。但这篇论文是HBase的基础，你花上一点时间来读读，就可以了解到，对各种非关系型数据存储优缺点的一个很好的比较。
- [NoSQL Data Modeling Techniques](#)，NoSQL建模技术。这篇文章我曾经翻译在了 CoolShell 上，标题为 [NoSQL 数据建模技术](#)，供你参考。
 - [MongoDB - Data Modeling Introduction](#)，虽然这是MongoDB的数据建模介绍，但是其很多观点可以用于其它的NoSQL数据库。
 - [Firebase - Structure Your Database](#)，Google的Firebase数据库使用JSON建模的一些最佳实践。
- 因为CAP原理，所以当你需要选择一个NoSQL数据库的时候，你应该看看这篇文档 [Visual Guide to NoSQL Systems](#)。

选SQL还是NoSQL，这里有两篇文章，值得你看看。

- [SQL vs. NoSQL Databases: What's the Difference?](#)
- [Salesforce: SQL or NoSQL](#)

各种NoSQL数据库

学习使用NoSQL数据库其实并不是一件很难的事，只要你把官方的文档仔细地读一下，是很容易上手的，而且大多数NoSQL数据库都是开源的，所以，也可以通过代码自己解决问题。下面我主要给出一些典型的NoSQL数据库的一些经验型的文章，供你参考。

列数据库Column Database

- Cassandra相关
 - 沃尔玛实验室有两篇文章值得一读。
 - [Avoid Pitfalls in Scaling Cassandra Cluster at Walmart](#)
 - [Storing Images in Cassandra at Walmart](#)
 - [Yelp: How We Scaled Our Ad Analytics with Apache Cassandra](#)，Yelp的这篇博客也有一些相关的经验和教训。
 - [Discord: How Discord Stores Billions of Messages](#)，Discord公司分享的一个如何存储十亿级消息的技术文章。
 - [Cassandra at Instagram](#)，Instagram的一个PPT，其中介绍了Instagram中是怎么使用Cassandra的。
 - [Netflix: Benchmarking Cassandra Scalability on AWS-Over a million writes per second](#)，Netflix公司在AWS上给Cassandra做的一个Benchmark。

- HBase相关
 - [Imgur Notification: From MySQL to HBASE](#)
 - [Pinterest: Improving HBase Backup Efficiency](#)
 - [IBM : Tuning HBase performance](#)
 - [HBase File Locality in HDFS](#)
 - [Apache Hadoop Goes Realtime at Facebook](#)
 - [Storage Infrastructure Behind Facebook Messages: Using HBase at Scale](#)
 - [GitHub: Awesome HBase](#)

对于HBase有两本书你可以考虑一下。

- 首先，先推荐两本书，一本是偏实践的《[HBase实战](#)》，另一本是偏大而全的手册型的《[HBase权威指南](#)》。
- 当然，你也可以看看官方的 [The Apache HBase™ Reference Guide](#)
- 另外两个列数据库：
 - [ClickHouse - Open Source Distributed Column Database at Yandex](#)
 - [Scaling Redshift without Scaling Costs at GIPHY](#)

文档数据库 Document Database - MongoDB, SimpleDB, CouchDB

- [Data Points - What the Heck Are Document Databases?](#)
- [eBay: Building Mission-Critical Multi-Data Center Applications with MongoDB](#)
- [The AWS and MongoDB Infrastructure of Parse: Lessons Learned](#)
- [Migrating Mountains of Mongo Data](#)
- [Couchbase Ecosystem at LinkedIn](#)

- [Couchbase Ecosystem at LinkedIn](#)
- [SimpleDB at Zendesk](#)
- [GitHub: Awesome MongoDB](#)

数据结构数据库 Data structure Database - Redis

- [Learn Redis the hard way \(in production\) at Trivago](#)
- [Twitter: How Twitter Uses Redis To Scale - 105TB RAM, 39MM QPS, 10,000+ Instances](#)
- [Slack: Scaling Slack's Job Queue - Robustly Handling Billions of Tasks in Milliseconds Using Kafka and Redis](#)
- [GitHub: Moving persistent data out of Redis at GitHub](#)
- [Instagram: Storing Hundreds of Millions of Simple Key-Value Pairs in Redis](#)
- [Redis in Chat Architecture of Twitch \(from 27:22\)](#)
- [Deliveroo: Optimizing Session Key Storage in Redis](#)
- [Deliveroo: Optimizing Redis Storage](#)
- [GitHub: Awesome Redis](#)

时序数据库 Time-Series Database

- [What is Time-Series Data & Why We Need a Time-Series Database](#)
- [Time Series Data: Why and How to Use a Relational Database instead of NoSQL](#)
- [Beringei: High-performance Time Series Storage Engine @Facebook](#)
- [Introducing Atlas: Netflix's Primary Telemetry Platform @Netflix](#)
- [Building a Scalable Time Series Database on PostgreSQL](#)
- [Scaling Time Series Data Storage - Part I @Netflix](#)
- [Design of a Cost Efficient Time Series Store for Big Data](#)
- [GitHub: Awesome Time-Series Database](#)

图数据库 - Graph Platform

- 首先是IBM Developerworks 上的两个简介性的PPT。
 - [Intro to graph databases, Part 1, Graph databases and the CRUD operations](#)
 - [Intro to graph databases, Part 2, Building a recommendation engine with a graph database](#)
- 然后是一本免费的电子书《[Graph Database](#)》。
- 接下来是一些图数据库的介绍文章。
 - [Handling Billions of Edges in a Graph Database](#)
 - [Neo4j case studies with Walmart, eBay, AirBnB, NASA, etc](#)
 - [FlockDB: Distributed Graph Database for Storing Adjacency Lists at Twitter](#)
 - [JanusGraph: Scalable Graph Database backed by Google, IBM and Hortonworks](#)
 - [Amazon Neptune](#)

搜索数据库 - ElasticSearch

- [Elasticsearch: The Definitive Guide](#) 这是官网方的ElasticSearch的学习资料，基本上来说，看这个就够了。
- 接下来是4篇和性能调优相关的工程实践。
 - [Elasticsearch Performance Tuning Practice at eBay](#)
 - [Elasticsearch at Kickstarter](#)
 - [9 tips on ElasticSearch configuration for high performance](#)
 - [Elasticsearch In Production-Deployment Best Practices](#)
- 最后是GitHub上的资源列表 [GitHub: Awesome ElasticSearch](#) 。

小结

好了，总结一下今天分享的内容。虽然有人会认为数据库与程序员无关，是DBA的事儿。但我坚信，数据库才真正是程序员的事儿。因为程序是需要和数据打交道的，所以程序员或架构师不仅需要设计数据模型，还要保证整体系统的稳定性和可用性，数据是整个系统中关键中的关键。

对于数据库方向，重点就是两种数据库，一种是以SQL为代表的关系型数据库，另一种是以非SQL为代表的NoSQL数据库。因而，在这篇文章中，我给出了MySQL和各种开源NoSQL的一些相关的有价值的文章和导读，主要是让你对这些数据库的内在有一定的了解，但又不会太深。同时给出了一些知名企业使用数据库的工程实践，这对于了解各种数据库的优劣非常有帮助，值得认真读读。

从下篇文章开始，我们将进入分布式系统架构方面的内容，里面不仅涵盖了大量的理论知识，更有丰富的入门指导和大量的工程实践。敬请期待。

下面是《程序员进阶攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统入门](#)

系统知识

- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 -



学习分布式系统跟学习其它技术非常不一样，分布式系统涵盖的面非常广，具体来说涵盖如下几方面：

- **服务调度**，涉及服务发现、配置管理、弹性伸缩、故障恢复等。
- **资源调度**，涉及对底层资源的调度使用，如计算资源、网络资源和存储资源等。
- **流量调度**，涉及路由、负载均衡、流控、熔断等。
- **数据调度**，涉及数据复本、数据一致性、分布式事务、分库、分表等。
- **容错处理**，涉及隔离、幂等、重试、业务补偿、异步、降级等。
- **自动化运维**，涉及持续集成、持续部署、全栈监控、调用链跟踪等。

所有这些形成了分布式架构的整体复杂度，也造就了分布式系统中的很多很多论文、图书以及很多很多的项目。要学好分布式系统及其架构，我们需要大量的时间和实践才能真正掌握这些技术。

这里有几点需要你注意一下。

- 分布式系统之所以复杂，就是因为其太容易也太经常出错了。这意味着，**你要把处理错误的代码当成正常功能的代码来处理**。
- 开发一个健壮的分布式系统的成本是单体系统的几百倍甚至几万倍。这意味着，**我们要自己开发一个，需要能力很强的开发人员**。
- 非常健壮的开源的分布式系统并不多，或者说基本没有。这意味着，**如果你要用开源的，那么你需要hold得住其源码**。
- 管理或是协调多个服务或机器是非常难的。这意味着，**我们要去读很多很多的分布式系统的论文**。
- 在分布式环境下，出了问题是很难debug的。这意味着，**我们需要非常好的监控和跟踪系统，还需要经常做演练和测试**。
- 在分布式环境下，你需要更科学地分析和统计。这意味着，**我们要用P90这样的统计指标，而不是平均值，我们还需要做容量计划和评估**。
- 在分布式环境下，需要应用服务化。这意味着，**我们需要一个服务开发框架，比如SOA或微服务**。
- 在分布式环境下，故障不可怕，可怕的是影响面过大，时间过长。这意味着，**我们需要花时间来开发我们的自动化运维平台**。

总之，在分布式环境下，一切都变得非常复杂。要进入这个领域，你需要有足够的耐性和足够强的心态来接受各式各样的失败。当拥有丰富的实践和经验后，你才会有所建树。这并不是一日之功，你可能要在这个领域花费数年甚至数十年的时间。

分布式架构入门

学习如何设计可扩展的架构将会有助于你成为一个更好的工程师。系统设计是一个很宽泛的话题。在互联网上，关于架构设计原则的资源也是多如牛毛。所以，你需要知道一些基本概念，对此，这里你先读一下下面两篇文章，都非常不错。

- [Scalable Web Architecture and Distributed Systems](#)，这篇文章会给你一个大概的分布式架构是怎么来解决系统扩展性问题的粗略方法。
- [Scalability, Availability & Stability Patterns](#)，这个PPT能在扩展性、可用性、稳定性等方面给你一个非常大的架构设计视野和思想，可以让你感受一下大概的全景图。

然后，我更强烈推荐GitHub上的一篇文档 - [System Design Primer](#)，这个仓库主要组织收集分布式系统的一些与扩展性相关的资源，它可以帮助你学习如何构建可扩展的架构。

目前这个仓库收集到了好些系统架构和设计的基本方法。其中包括：CAP理论、一致性模型、可用性模式、DNS、CDN、负载均衡、反向代理、应用层的微服务和服务发现、关系型数据库和NoSQL、缓存、异步通讯、安全等。

我认为，上面这几篇文章基本足够可以让你入门了，因为其中基本涵盖了所有与系统架构相关的技术。这些技术，足够这世上90%以上的公司用了，只有超级巨型的公司才有可能使用更高层次的技术。

分布式理论

下面，我们来学习一下分布式方面的理论知识。

首先，你需要看一下 [An introduction to distributed systems](#)。这只是某个教学课程的提纲，我觉得还是很不错的，几乎涵盖了分布式系统方面的所有知识点，而且辅以简洁并切中要害的说明文字，非常适合初学者提纲挈领地了解知识全貌，快速与现有知识结合，形成知识体系。这也是一个分布式系统的知识图谱，可以让你看到分布式系统的整体全貌。你可以根据这个知识图Google下去，然后你会学会所有的东西。

然后，你需要了解一下拜占庭将军问题 ([Byzantine Generals Problem](#))。这个问题是莱斯利·兰波特 (Leslie Lamport) 于1982年提出用来解释一致性问题的一个虚构模型 ([论文地址](#))。拜占庭是古代东罗马帝国的首都，由于地域宽广，守卫边境的多个将军（系统中的多个节点）需要通过信使来传递消息，达成某些一致的决定。但由于将军中可能存在叛徒（系统中节点出错），这些叛徒将努力向不同的将军发送不同的消息，试图会干扰一致性的达成。拜占庭问题即为在此情况下，如何让忠诚的将军们能达成行动的一致。

对于拜占庭问题来说，假如节点总数为 N，叛变将军数为 F，则当 $N \geq 3F + 1$ 时，问题才有解，即拜占庭容错 (Byzantine Fault Tolerant, BFT) 算法。拜占庭容错算法解决的是，网络通信可靠但节点可能故障情况下一致性该如何达成的问题。

最早由卡斯特罗 (Castro) 和利斯科夫 (Liskov) 在1999年提出的实用拜占庭容错 (Practical Byzantine Fault Tolerant, PBFT) 算法，是第一个得到广泛应用的BFT算法。只要系统中有 $2/3$ 的节点是正常工作的，则可以保证一致性。PBFT算法包括三个阶段来达成共识：预准备 (Pre-Prepare)、准备 (Prepare) 和提交 (Commit)。

这里有几篇和这个问题相关的文章，推荐阅读。

- [Dr.Dobb's - The Byzantine Generals Problem](#)
- [The Byzantine Generals Problem](#)
- [Practicle Byzantine Fault Tolerance](#)

拜占庭容错系统研究中有三个重要理论：CAP、FLP和DLS。

- **CAP定理**，CAP理论相信你应该听说过不下N次了。CAP定理是分布式系统设计中最基础也是最为关键的理论。CAP定理指出，分布式数据存储不可能同时满足以下三个条件：一致性 (Consistency)、可用性 (Availability) 和分区容忍 (Partition tolerance)。“在网络发生阻断 (partition) 时，你只能选择数据的一致性 (consistency) 或可用性 (availability)，无法两者兼得”。

论点比较直观：如果网络因阻断而分隔为二，在其中一边我送出一笔交易：“将我的十元给A”；在另一边我送出另一笔交易：“将我的十元给B”。此时系统要不是，a) 无可用性，即这两笔交易至少会有一笔交易不会被接受；要不就是，b) 无一致性，一半看到的是A多了十元而另一半则看到B多了十元。要注意的是，CAP理论和扩展性 (scalability) 是无关的，在分片 (sharded) 或非分片的系统皆适用。

- **FLP impossibility**-在异步环境中，如果节点间的网络延迟没有上限，只要有一个恶意的节点存在，就没有算法能在有限的时间内达成共识。但值得注意的是，“[Las Vegas” algorithms](#)（这个算法又叫撞大运算法，其保证结果正确，只是在运算时所用资源上进行赌博，一个简单的例子是随机快速排序，它的pivot是随机选的，但排序结果永远一致）在每一轮皆有一定机率达成共识，随着时间增加，机率会越趋近于1。而这也是许多成功的共识算法会采用的解决问题的办法。

- 容错的上限-由 [DLS论文](#)，我们可以得到以下结论。

- 在部分同步 (partially synchronous) 的网络环境中 (即网络延迟有一定的上限，但我们无法事先知道上限是多少)，协议可以容忍最多 $1/3$ 的拜占庭故障 (Byzantine fault)。
- 在异步 (asynchronous) 的网络环境中，具有确定性质的协议无法容忍任何错误，但这篇论文并没有提及 [randomized algorithms](#)，在这种情况下可以容忍最多 $1/3$ 的拜占庭故障。

• 在同步(synchronous)网络环境中(即网络延迟有上限且上限是已知的)，协议可以容忍100%的拜占庭故障，但当超过 $1/2$ 的节点为恶意节点时，会有一些限制条件。要注意的是，我们考虑的是“具有认证特性的拜占庭模型(authenticated Byzantine)”，而不是“一般的拜占庭模型”；具有认证特性指的是将如今已经过大量研究且成本低廉的公私钥加密机制应用在我们的算法中。

当然，还有一个著名的“8条荒谬的分布式假设 ([Fallacies of Distributed Computing](#))”。

1. 网络是稳定的。
2. 网络传输的延迟是零。
3. 网络的带宽是无穷大。
4. 网络是安全的。
5. 网络的拓扑不会改变。
6. 只有一个系统管理员。
7. 传输数据的成本为零。
8. 整个网络是同构的。

阿尔农·罗特姆-盖尔-奥兹 (Arnon Rotem-Gal-Oz) 写了一篇长文 [Fallacies of Distributed Computing Explained](#) 来解释为什么这些观点是错误的。另外，[加勒思·威尔逊 \(Gareth Wilson\)](#) 的文章则用日常生活中的例子，对这些点做了通俗的解释。为什么我们深刻地认识到这8个错误？是因为，这要我们清楚地认识到——在分布式系统中错误是不可能避免的，我们在分布式系统中，能做的不是避免错误，而是要把错误的处理当功能写在代码中。

下面分享几篇一致性方面的论文。

- 当然，关于经典的CAP理论，也存在一些误导的地方，这个问题在2012年有一篇论文 [CAP Twelve Years Later: How the Rules Have Changed \(中译版\)](#) 中做了一些讨论，主要是说，在CAP中最大的问题就是分区，也就是P，在P发生的情况下，非常难以保证C和A。然而，这是强一致性的。

其实，在很多时候，我们并不需要强一致性的系统，所以后来，人们争论关于数据一致性和可用性时，主要是集中在强一致性的ACID或最终一致性的BASE。当时，BASE还不怎么为世人所接受，主要是大家都觉得ACID是最完美的模型，大家很难接受不完美的BASE。在CAP理论中，大家总是觉得需要“三选二”，也就是说，P是必选项，那“三选二”的选择题不就变成数据一致性(consistency)、服务可用性(availability)间的“二选一”？

然而，现实却是，P很少遇到，而C和A这两个事，工程实践中一致性有不同程度，可用性也有不同等级，在保证分区容错性的前提下，放宽约束后可以兼顾一致性和可用性，两者不是非此即彼。其实，在一个时间可能允许的范围内是可以取舍并交替选择的。

- [Harvest, Yield, and Scalable Tolerant Systems](#)，这篇论文是基于上面那篇“CAP 12年后”的论文写的，它主要提出了Harvest和Yield概念，并把上面那篇论文中所讨论的东西讲得更为仔细了一些。
- [Base: An Acid Alternative \(中译版\)](#)，本文是eBay的架构师在2008年发表给ACM的文章，是一篇解释BASE原则，或者说最终一致性的经典文章。文中讨论了BASE与ACID原则的基本差异，以及如何设计大型网站以满足不断增长的可伸缩性需求，其中有如何对业务做调整和折中，以及一些具体的折中技术的介绍。一个比较经典的话是——“在对数据库进行分区后，为了可用性(Availability)牺牲部分一致性(Consistency)可以显著地提升系统的可伸缩性(Scalability)”。
- [Eventually Consistent](#)，这篇文章是AWS的CTO维尔纳·沃格尔 (Werner Vogels) 在2008年发布在ACM Queue上的一篇数据库方面的重要文章，阐述了NoSQL数据库的理论基石——最终一致性，对传统的关系型数据库 (ACID, Transaction) 做了较好的补充。

小结

好了，总结一下今天分享的内容。文章的开头，我给出了学习分布式架构需要注意的几个关键点，然后列出了入门学习的资源，基本涵盖了所有与系统架构相关的技术。随后讲述了拜占庭容错系统研究中有三个重要理论：CAP、FLP和DLS，以及8条荒谬的分布式假设，从理论和认知等角度让你更为清楚地理解分布式系统。最后分享了几篇一致性相关的论文，很实用很经典，推荐阅读。

下篇文章中，我将推荐一些分布式架构的经典图书和论文，并给出了导读文字，几乎涵盖了分布式系统架构方面的所有关键的理论知识。敬请期待。

下面是《程序员练级攻略 (2018)》系列文章的目录 (持续更新中)。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络 \(系统底层知识\)](#)
 - [异步I/O模型和Lock-Free编程 \(系统底层知识\)](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门 \(分布式架构\)](#)
 - [分布式架构经典图书和论文 \(分布式架构\)](#)
 -

The promotional image features a portrait of Chen Hao, a middle-aged man with short dark hair, wearing a dark blue button-down shirt. He is seated at a desk, looking slightly to his left with a thoughtful expression. To his left, large vertical Chinese characters '左耳听风' are written in a bold, white, artistic font. To his right, the 'Geke Time' logo is displayed, consisting of a stylized orange 'Q' icon followed by the text '极客时间' and the tagline '重拾极客精神·提升技术认知'. Below the logo, Chen Hao's name '陈皓' is written in a large, bold, black font. Underneath his name, the text '资深技术专家' and '骨灰级程序员' is displayed in a smaller black font. At the bottom right, there is a QR code with the text '扫码订阅' next to it.

经典图书

首先，我推荐几本分布式架构方面的经典图书。

- [Distributed Systems for fun and profit](#)，这是一本免费的电子书。作者撰写此书的目的是希望以一种更易于理解的方式，讲述以亚马逊的Dynamo、谷歌的Bigtable和MapReduce等为代表的分布式系统背后的核心思想。
 - [Designing Data Intensive Applications](#)，这本书是一本非常好的书，我们知道，在分布式的世界里，数据结点的扩展是一件非常麻烦的事。这本书深入浅出地用很多的工程案例讲解了如何让数据结点做扩展。作者马丁·科勒普曼（Martin Kleppmann）在分布式数据系统领域有着很深的功底，并在这本书中完整地梳理各类纷繁复杂设计背后的技术逻辑，不同架构之间的妥协与超越，很值得开发人员与架构设计者阅读。
- 这本书深入到B-Tree、SSTables、LSM这类数据存储结构中，并且从外部的视角来审视这些数据结构对NoSQL和关系型数据库的影响。这本书可以让你很清楚地了解到真正世界的大数据架构中的数据分区、数据复制的一些坑，并提供了很好的解决方案。最赞的是，作者将各种各样的技术的本质非常好地关联在一起，令你触类旁通。
- 而且，这本书完全就是抽丝剥茧，循循善诱，从“提出问题”到“解决问题”、“解决方案”、“优化方案”和“对比不同的方案”，一点一点地把非常晦涩的技术和知识展开。本书的引用相当多，每章后面都有几百个Reference，通过这些Reference你可以看到更为广阔、更为精彩的世界。

- [Distributed Systems: Principles and Paradigms](#)，本书是由计算机科学家安德鲁·斯图尔特·塔能鲍姆（Andrew S. Tanenbaum）和其同事马丁·范·斯蒂恩（Martin van Steen）合力撰写的，是分布式系统方面的经典教材。

语言简洁，内容通俗易懂，介绍了分布式系统的七大核心原理，并给出了大量的例子；系统讲述了分布式系统的概念和技术，包括通信、进程、命名、同步化、一致性和复制、容错以及安全等；讨论了分布式应用的开发方法（即范型）。但本书不是一本指导“如何做”的手册，仅适合系统性地学习基础知识，了解编写分布式系统的基本原则和逻辑。中文翻译版为《[分布式系统原理与范型](#)》（第二版）。

- [Scalable Web Architecture and Distributed Systems](#)，

这是一本免费的在线小册子，其中文翻译版[可扩展的Web架构和分布式系统](#)。本书主要针对面向互联网（公网）的分布式系统，但其中的原理或许也可以应用于其他分布式系统的设计中。作者的观点是，通过了解大型网站的分布式架构原理，小型网站的构建也能从中受益。本书从大型互联网系统的常见特性，如高可用、高性能、高可靠、易管理等出发，引出了一个类似于Flickr的典型的大型图片网站的例子。

- [Principles of Distributed Systems](#)，本书是苏黎世联邦理工学院的教材。它讲述了多种分布式系统中会用到的算法。虽然分布式系统的不同场景会用到不同算法，但并不表示这些算法都会被用到。不过，作为学生来说，掌握了算法设计的精髓也就能举一反三地设计出解决其他问题的算法，从而得到分布式系统架构设计中所需的算法。

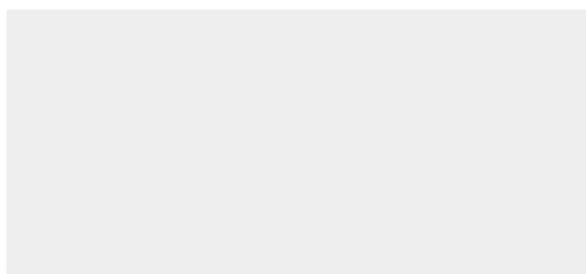
经典论文

分布式事务

想了解分布式模型中最难的“分布式事务”，你需要看看Google App Engine联合创始人瑞恩·巴雷特（Ryan Barrett）在2009年的Google I/O大会上的演讲《[Transaction Across DataCenter](#)》（[YouTube视频](#)）。

在这个演讲中，巴雷特讲述了各种经典的解决方案如何在一致性、事务、性能和错误上做平衡。而最后得到为什么分布式系统的事务只有Paxos算法是最好的。

下面这个图是这个算法中的结论。



你也可以移步看一下我在Coolshell上写的这篇文章《[分布式系统的事务处理](#)》。

Paxos一致性算法

Paxos算法，是莱斯利·兰伯特（Leslie Lamport）于1990年提出出来的一种基于消息传递且具有高度容错特性的一致性算法。但是这个算法太过于晦涩，所以一直以来都属于理论上的论文性质的东西。其真正进入工程圈，主要是来源于Google的Chubby lock——一个分布式的锁服务，用在了Bigtable中。直到Google发布了下面这两篇论文，Paxos才进入到工程界的视野中来。

- [Bigtable: A Distributed Storage System for Structured Data](#)
- [The Chubby lock service for loosely-coupled distributed systems](#)

Google与Bigtable相齐名的还有另外两篇论文。

- [The Google File System](#)
- [MapReduce: Simplified Data Processing on Large Clusters](#)

不过，这几篇文章中并没有讲太多的Paxos算法上的细节，反而是《[Paxos Made Live - An Engineering Perspective](#)》这篇论文中提到了很多工程实现的细节。这篇论文详细解释了Google实现Paxos时遇到的各种问题和解决方案，讲述了从理论到实际应用二者之间巨大的鸿沟。

Paxos算法的原版论文比较晦涩，也不易懂。这里推荐一篇比较容易读的——《[Neat Algorithms - Paxos](#)》。这篇文章中还有一些小动画帮助你读懂。还有一篇可以帮你理解的文章是《[Paxos by Examples](#)》。

Raft一致性算法

因为Paxos算法太过于晦涩，而且在实际的实现上有太多的坑，并不太容易写对。所以，有人搞出了另外一个一致性的算法，叫Raft。其原始论文是《[In search of an Understandable Consensus Algorithm \(Extended Version\)](#)》，寻找一种易于理解的Raft算法。这篇论文的译文在InfoQ上，题为《[Raft一致性算法论文译文](#)》，推荐你读一读。

这里推荐几个不错的Raft算法的动画演示。

- [Raft - The Secret Lives of Data](#)
- [Raft Consensus Algorithm](#)
- [Raft Distributed Consensus Algorithm Visualization](#)

Gossip一致性算法

后面，业内又搞出来一些工程上的东西，比如Amazon的DynamoDB，其论文《[Dynamo: Amazon's Highly Available Key Value Store](#)》的影响力非常大。这篇论文中讲述了Amazon的DynamoDB是如何满足系统的高可用、高扩展和高可靠的。其中展示了系统架构是如何做到数据分布以及数据一致性的。GFS采用的是查表式的数据分布，而DynamoDB采用的是计算式的，也是一个改进版的通过虚拟结点减少增加结点带来数据迁移的一致性哈希。

这篇文章中有几个关键的概念，一个是Vector Clock，另一个是Gossip协议。

- [Time, Clocks and the Ordering of Events in a Distributed System](#)，这篇文章是莱斯利·兰伯特（Leslie Lamport）于1978年发表的，并在2007年被选入SOSP的名人堂，被誉为第一篇真正的“分布式系统”论文，该论文曾一度成为计算机科学史上被引用最多的文章。分布式系统中的时钟同步是一个非常难的问题，因为分布式系统中是使用消息进行通信的，若使用物理时钟来进行同步，一方面是不同的process的时钟有差异，另一方面是时间的计算也有一定的误差，这样若有两个时间相同的事件，则无法区分它们谁前谁后了。这篇文章主要解决分布式系统中的时钟同步问题。

- [马萨诸塞大学课程Distributed Operating System](#) 中第10节 [Clock Synchronization](#), 这篇讲议讲述了时钟同步的问题。

- 关于Vector Clock, 你可以看一下 [Why Vector Clocks are Easy](#) 和 [Why Vector Clocks are Hard](#) 这两篇文章。

用来做数据同步的Gossip协议的原始论文是 [Efficient Reconciliation and Flow Control for Anti-Entropy Protocols](#)。Gossip算法也是Cassandra使用的数据复制协议。这个协议就像八卦和谣言传播一样, 可以“一传十、十传百”传播开来。但是这个协议看似简单, 细节上却非常麻烦。

Gossip协议也是NoSQL数据库Cassandra中使用到的数据协议, 你可以上YouTube上看一下这个视频介绍: [Understanding Gossip \(Cassandra Internals\)](#)。

关于Gossip的一些图示化的东西, 你可以看一下动画 [Gossip Visualization](#)。

分布式存储和数据库

除了前面的Google的BigTable和Google File System那两篇论文, 还有Amazon的DynamoDB的论文, 下面也有几篇也是要读一下的。

- 一篇是AWS Aurora的论文 [Amazon Aurora: Design Considerations for High Throughput Cloud -Native Relation Databases](#)。
- 另一篇是比较有代表的论文是Google的 [Spanner: Google's Globally-Distributed Database](#)。其2017年的新版论文: [Spanner, TrueTime & The CAP Theorem](#)。
- [F1 - The Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business](#)。
- [Cassandra: A Decentralized Structured Storage System](#)。
- [CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data](#), 这里提到的算法被应用在了Ceph分布式文件系统中, 其架构可以读一下 [RADOS - A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters](#) 以及 [Ceph的架构文档](#)。

分布式消息系统

- 分布式消息系统, 你一定要读一下Kafka的这篇论文 [Kafka: a Distributed Messaging System for Log Processing](#)。
- [Wormhole: Reliable Pub-Sub to Support Geo-replicated Internet Services](#), Wormhole是Facebook内部使用的一个Pub-Sub系统, 目前还没有开源。它和Kafka之类的消息中间件很类似。但是它又不像其它的Pub-Sub系统, Wormhole没有自己的存储来保存消息, 它也不需要数据源在原有的更新路径上去插入一个操作来发送消息, 是非侵入式的。其直接部署在数据源的机器上并直接扫描数据源的transaction logs, 这样还带来一个好处, Wormhole本身不需要做任何地域复制(geo-replication)策略, 只需要依赖于数据源的geo-replication策略即可。
- [All Aboard the Databus! LinkedIn's Scalable Consistent Change Data Capture Platform](#), 在LinkedIn投稿SOCC 2012的这篇论文中, 指出支持对不同数据源的抽取, 允许不同数据源抽取器的开发和接入, 只需该抽取器遵循设计规范即可。该规范的一个重要方面就是每个数据变化都必须被一个单调递增的数字标注(SCN), 用于同步。这其中的一些方法完全可以用做异地双活的系统架构中。(和这篇论文相关的几个链接如下: [PDF论文](#)、[PPT分享](#)。)

日志和数据

- [The Log: What every software engineer should know about real-time data's unifying abstraction](#), 这篇文章好长, 不过这是一篇非常好非常好的文章, 这是每个工程师都应知道的事, 必看啊。你可以看中译版 [日志: 每个软件工程师都应该知道的有关实时数据的统一概念](#)。
- [The Log-Structured Merge-Tree \(LSM-Tree\)](#), N多年前, 谷歌发表了“Bigtable”的论文, 论文中很多很酷的方面, 其一就是它所使用的文件组织方式, 这个方法更一般的名字叫Log Structured-Merge Tree。LSM是当前被用在许多产品的文件结构策略: HBase、Cassandra、LevelDB、SQLite, 甚至在MongoDB 3.0中也带了一个可选的LSM引擎(Wired Tiger实现的)。LSM有趣的地方是它抛弃了大多数数据库所使用的传统文件组织方法。实际上, 当你第一次看它时是违反直觉的。这篇论文可以让你明白这个技术。(如果读起来有些费解的话, 你可以看看中文社区里的这几篇文章: [文章一](#)、[文章二](#)。)
- [Immutability Changes Everything](#), 这篇论文是现任Salesforce软件架构师帕特·赫兰德(Pat Helland)在CIDR 2015大会上发表的([相关视频演讲](#))。
- [Tango: Distributed Data Structures over a Shared Log](#)。这个论文非常经典, 其中说明了不可变性(immutability)架构设计的优点。随着为海量数据集存储和计算而设计的以数据为中心的新型抽象技术的出现, 分布式系统比以往任何时候都更容易构建。但是, 对于元数据的存储和访问不存在类似的抽象。

为了填补这一空白, Tango为开发人员提供了一个由共享日志支持的内存复制数据结构(例如地图或树)的抽象。Tango对象易于构建和使用, 通过共享日志上简单的追加和读取操作来复制状态, 而不是复杂的分布式协议。在这个过程中, 它们从共享日志中获得诸如线性化、持久性和高可用性等属性。Tango还利用共享日志支持跨不同对象的快速事务处理, 允许应用程序跨机器进行状态划分, 并在不牺牲一致性的情况下扩展到底层日志的上限。

分布式监控和跟踪

- Google的分布式跟踪监控论文 - [Dapper, a Large-Scale Distributed Systems Tracing Infrastructure](#), 其开源实现有三个 [Zipkin](#)、[Pinpoint](#) 和 [HTrace](#)。我个人更喜欢Zipkin。

数据分析

- [The Unified Logging Infrastructure for Data Analytics at Twitter](#), Twitter公司的一篇关于日志架构和数据分析的论文。
- [Scaling Big Data Mining Infrastructure: The Twitter Experience](#), 讲Twitter公司的数据分析平台在数据量越来越大, 架构越来越复杂, 业务需求越来越多的情况下, 数据分析从头到底是怎么做的。
- [Dremel: Interactive Analysis of Web-Scale Datasets](#), Google公司的Dremel, 是一个针对临时查询提供服务的系统, 它处理的是只读的多层数据。本篇文章介绍了它的架构与实现, 以及它与MapReduce是如何互补的。
- [Resident Distributed Datasets: a Fault-Tolerant Abstraction for In-Memory Cluster Computing](#), 这篇论文提出了弹性分布式数据集(Resilient Distributed Dataset, RDD)的概念, 它是一个分布式存储抽象, 使得程序员可以在大型集群上以容错的方式执行内存计算; 解释了其出现原因: 解决之前计算框架在迭代算法和交互式数据挖掘工具两种应用场景下处理效率低下问题, 并指出将数据保存在内存中, 可以将性能提高一个数量级; 同时阐述了其实现原理及应用场景区分等内容。很有趣儿, 推荐阅读。

与编程相关的论文

- [Distributed Programming Model](#)
- [PSync: a partially synchronous language for fault-tolerant distributed algorithms](#)
- [Programming Models for Distributed Computing](#)
- [Logic and Lattices for Distributed Programming](#)

其它的分布式论文阅读列表

除了上面的那些我觉得不错的论文, 下面还有三个我觉得不错的分布式系统论文的阅读列表, 你可以浏览一下。

- [Services Engineering Reading List](#)
- [Readings in Distributed Systems](#)
- [Google Research - Distributed Systems and Parallel Computing](#)

小结

今天分享的内容是分布式架构方面的经典图书和论文, 并给出了导读文字, 几乎涵盖了分布式系统架构方面的所有关键的理论知识。这些内容非常重要, 是学好分布式架构的基石, 请一定要认真学习。

下篇文章中, 我们将讲述分布式架构工程设计方面的内容, 包括设计原则、设计模式以及工程实践等方面的内容。敬请期待。

下面是《程序员进阶攻略(2018)》系列文章的目录(持续更新中)。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇

- 修养篇
 - 程序员修养
- 专业基础篇
 - 编程语言
 - 理论学科
 - 系统知识
- 软件设计篇
 - 软件设计
- 高手成长篇
 - Linux系统、内存和网络 (系统底层知识)
 - 异步I/O模型和Lock-Free编程 (系统底层知识)
 - Java底层知识
 - 数据库
 - 分布式架构入门 (分布式架构)
 - 分布式架构经典图书和论文 (分布式架构)
 -

左耳朵耗子
全年独家专栏《左耳听风》

陈皓
骨灰级技术程序员专家

拼团价 **¥199** / 3人成团
原价: 299

扫码拼团

A promotional banner for Chen Hao's column 'Left Ear Hair'. The banner features a portrait of Chen Hao, a middle-aged man with short dark hair, wearing a dark blue button-down shirt. He is seated at a desk, looking slightly to his right with a neutral expression. To the left of the portrait, the text '左耳朵耗子' is displayed in large, bold, white font, with '全年独家专栏《左耳听风》' below it in a smaller white font. To the right of the portrait, vertical text reads '陈皓' (Chen Hao) and '骨灰级技术程序员专家' (Expert in Technical Programming). On the far left, there is promotional text: '拼团价 **¥199** / 3人成团' (Group buy price ¥199 for 3 people) and '原价: 299' (Original price: 299). In the bottom right corner of the banner, there is an orange rounded rectangle containing a QR code and the text '扫码拼团' (Scan to group buy).

程序员练级攻略（2018）：分布式架构工程设计

2018-07-17 陈皓，杨爽

要学好分布式架构，你首先需要学习一些架构指导性的文章和方法论，即分布式架构设计原则。下面是几篇很不错的文章，值得一读。

- [Designs, Lessons and Advice from Building Large Distributed Systems](#), Google 杰夫·迪恩 (Jeff Dean) 2009年一次演讲的PPT。2010年，斯坦福大学请杰夫·迪恩到大学里给他们讲了一节课，你可以在YouTube上看一下，[Building Software Systems At Google and Lessons Learned](#)，其回顾了Google发展的历史。
- [The Twelve-Factor App](#)，如今，软件通常会作为一种服务来交付，它们被称为网络应用程序，或软件即服务（SaaS）。12-Factor 为构建SaaS应用提供了方法论，是架构师必读的文章。[（中译版）](#)这篇文章在业内的影响力很大很大，必读！
- [Notes on Distributed Systems for Young Bloods](#)，给准备进入分布式系统领域的人的一些忠告。
- [On Designing and Deploying Internet-Scale Services](#) [（中译版）](#)，微软Windows Live服务平台的一些经验性的总结文章，很值得一读。
- [4 Things to Keep in Mind When Building a Platform for the Enterprise](#)，Box平台VP海蒂·威廉姆斯 (Heidi Williams) 撰写的一篇文章，阐述了为企业构建平台时需要牢记的四件关于软件设计方面的事：1. Design Broadly, Build Narrowly; 2. Platforms Are Powerful and Flexible. Choose wisely what to expose when!; 3. Build Incrementally, Get Feedback, and Iterate; 4. Create a Platform-first Mentality。文章中有详细的解读，推荐看看。
- [Principles of Chaos Engineering](#)，我们知道，Netflix公司有一个叫Chaos Monkey的东西，这个东西会到分布式系统里瞎搞，以此来测试系统的健壮和稳定性。这个视频中，Netflix分享了一些软件架构的经验和原则，值得一看。
- [Building Fast & Resilient Web Applications](#)，伊利亚·格里高利克 (Ilya Grigorik) 在Google I/O 2016上的一次关于如何通过弹性设计来实现快速和可容错的网站架构的演讲，其中有好些经验分享。
- [Design for Resiliency](#)，这篇文章带我们全面认识“弹性（Resiliency）”，以及弹性对于系统的重要性，并详细阐述了如何设计和实现系统的弹性。
- 微软的Azure网站上有一系列的[Design Principle](#)的文章，你可以看看这几篇：[Design for Self-healing](#)、[Design for Scaling Out](#) 和 [Design for Evolution](#)。
- [Eventually Consistent](#)，AWS CTO维尔纳·沃格尔斯 (Werner Vogels) 发布在自己Blog上的一篇关于最终一致性的好文。
- [Writing Code that Scales](#)，Rackspace的一篇很不错的博文，告诉我们一些很不错的写出高扩展和高性能代码的工程原则。
- [Automate and Abstract: Lessons from Facebook on Engineering for Scale](#)，软件自动化和软件抽象，这是软件工程中最重要的两件事了。通过这篇文章，我们可以看到Facebook的关于这方面的一些经验教训。

设计模式

有了方法论后，你还需要学习一些比较细节的落地的技术。最好的方式就是学习被前人总结出来的设计模式，虽然设计模式也要分场景，但是设计模式可以让你知道一些套路，这些套路对于我们设计的分布式系统有非常大的帮助，不但可以让我们少走一些弯路，而且还能让我们更为系统和健壮地设计我们的架构。

下面是一些分布式架构设计模式的网站。

首先，需要重点推荐的是微软云平台 Azure 上的设计模式。[Cloud Design Patterns](#)，这个网站上罗列了分布式设计的各种设计模式，可以说是非常全面和完整。对于每一个模式都有详细的说明，并有对其优缺点的讨论，以及适用场景和不适用场景的说明，实在是一个非常好的学习分布式设计模式的地方。其中有如下分类。

- [设计模式：可用性；](#)
- [设计模式：数据管理；](#)
- [设计模式：设计和实现；](#)
- [设计模式：消息；](#)
- [设计模式：管理和监控；](#)
- [设计模式：性能和扩展；](#)
- [设计模式：系统弹力；](#)
- [设计模式：安全。](#)

除此之外，还有其它的一些关于分布式系统设计模式的网站和相关资料。

- [AWS Cloud Pattern](#)，这里收集了AWS云平台的一些设计模式。
- [Design patterns for container-based distributed systems](#)，这是Google给的一篇论文，其中描述了容器化下的分布式架构的设计模式。
- [Patterns for distributed systems](#)，这是一个PPT，其中讲了一些分布式系统的架构模式，你可以顺着到Google里去搜索。

我个人觉得微服务也好，SOA也好，都是分布式系统的一部分，这里有两个网站罗列了各种各样的服务架构模式。

- [A Pattern Language for Micro-Services](#)；
- [SOA Patterns](#)。

当然，还有我在极客时间上写的那些分布式的总结。

- [弹性设计篇](#)，内容包括：认识故障和弹性设计、隔离设计、异步通讯设计、幂等性设计、服务的状态、补偿事务、重试设计、熔断设计、限流设计、降级设计、弹性设计总结。
- [管理设计篇](#)，内容包括：分布式锁、配置中心、边车模式、服务网格、网关模式、部署升级策略等。
- [性能设计篇](#)，内容包括：缓存、异步处理、数据库扩展、秒杀、边缘计算等。

设计与工程实践

分布式系统的故障测试

- [FIT: Failure Injection Testing](#)，Netflix公司的一篇关于做故障注入测试的文章。
- [Automated Failure Testing](#)，同样来自Netflix公司的自动化故障测试的一篇博文。
- [Automating Failure Testing Research at Internet Scale](#)，Netflix公司伙同圣克鲁斯加利福尼亚大学和Gremlin游戏公司一同撰写的一篇论文。

弹性伸缩

- [4 Architecture Issues When Scaling Web Applications: Bottlenecks, Database, CPU, IO](#)，本文讲解了后端程序的主要性能指标，即响应时间和可伸缩性这两者如何能提高的解决方案，讨论了包括纵向和横向扩展，可伸缩架构、负载均衡、数据库的伸缩、CPU密集型和I/O密集型程序的考量等。
- [Scaling Stateful Objects](#)，这是一本叫《Development&Deployment of Multiplayer Online Games》书中一章内容的节选，讨论了有状态和无状态的节点如何伸缩的问题。虽然还没有写完，但是可以给你一些很不错的基本概念和想法。
- [Scale Up vs Scale Out: Hidden Costs](#)，Coding Horror上的一篇有趣的文章，详细分析了可伸缩性架构的不同扩展方案（横向扩展或纵向扩展）所带来的成本差异，帮助你更好地选择合理的扩展方案，可以看看。
- [Best Practices for Scaling Out](#)，OpenShift的一篇讨论Scale out最佳实践的文章。
- [Scalability Worst Practices](#)，这篇文章讨论了一些最差实践，你需要小心避免。
- [Reddit: Lessons Learned From Mistakes Made Scaling To 1 Billion Pageviews A Month](#)，Reddit分享的一些关于系统扩展的经验教训。

- 下面是几篇关于自动化弹性伸缩的文章。

- [Autoscaling Pinterest](#);
- [Square: Autoscaling Based on Request Queuing](#);
- [PayPal: Autoscaling Applications](#);
- [Trivago: Your Definite Guide For Autoscaling Jenkins](#);
- [Scryer: Netflix's Predictive Auto Scaling Engine](#)。

一致性哈希

- [Consistent Hashing](#)，这是一个一致性哈希的简单教程，其中还有代码示例。
- [Consistent Hashing: Algorithmic Tradeoffs](#)，这篇文章讲述了一致性哈希的一些缺陷和坑，以及各种哈希算法的性能比较，最后还给了一组代码仓库，其中有各种哈希算法的实现。
- [Distributing Content to Open Connect](#)，Netflix的一个对一致性哈希的实践，提出了Uniform Consistent Hashing，是挺有意思的一篇文章。
- [Consistent Hashing in Cassandra](#)，这是Cassandra中使用到的一致性哈希的相关设计。

数据库分布式

- [Life Beyond Distributed Transactions](#)，该文是Salesforce的软件架构师帕特·赫兰德 (Pat Helland) 于2016年12月发表的针对其在2007年CIDR (创新数据库研究会议) 上首次发表的同名文章的更新和缩写版本。业界谈到分布式事务通常指两段提交2PC事务 (Spring/JEE中JTA等)或者Paxos与Raft，这些事务都有明显缺点和局限性。

而赫兰德在本文讨论的是另外一种基于本地事务情况下的事务机制，它是基于实体和活动 (Activity) 的概念，其实类似DDD聚合根和领域事件的概念，这种工作流类型事务虽然需要程序员介入，依靠消息系统实现，但可以实现接近无限扩展的大型系统。赫兰德文中提出了重要的观点：“如果你不能使用分布式事务，那么你就只能使用工作流。”

- [How Sharding Works](#)，这是一篇很不错的探讨数据Sharding的文章。基本上来说，数据Sharding可能的问题都在这篇文章里谈到了。
- [Why you don't want to shard](#)，这是Percona的一篇文章，其中表达了，不到万不得已不要做数据库分片。是的，最好还是先按业务来拆分，先把做成微服务的架构，然后把数据集变简单，然后再做Sharding会更好。
- [How to Scale Big Data Applications](#)，这也是Percona给出的一篇关于怎样给大数据应用做架构扩展的文章。值得一读。
- [MySQL Sharding with ProxySQL](#)，用ProxySQL来支撑MySQL数据分片的一篇实践文章。

缓存

- [缓存更新的套路](#)，这是我在CoolShell上写的缓存更新的几个设计模式，包括Cache Aside、Read/Write Through、Write Behind Caching。
- [Design Of A Modern Cache](#)，设计一个现代化的缓存系统需要注意到的东西。
- [Netflix: Caching for a Global Netflix](#)，Netflix公司的全局缓存架构实践。
- [Facebook: An analysis of Facebook photo caching](#)，Facebook公司的图片缓存使用分析，这篇文章挺有意思的，用数据来调优不同的缓存大小和算法。
- [How trivago Reduced Memcached Memory Usage by 50%](#)，Trivago公司一篇分享自己是如何把Memcached的内存使用率降了一半的实践性文章。很有意思，可以让你学到很多东西。
- [Caching Internal Service Calls at Yelp](#)，Yelp公司的缓存系统架构。

消息队列

- [Understanding When to use RabbitMQ or Apache Kafka](#)，什么时候使用RabbitMQ，什么时候使用Kafka，通过这篇文章可以让你明白如何做技术决策。
- [Trello: Why We Chose Kafka For The Trello Socket Architecture](#)，Trello的Kafka架构分享。
- [LinkedIn: Running Kafka At Scale](#)，Linkedin公司的Kafka架构扩展实践。
- [Should You Put Several Event Types in the Same Kafka Topic?](#)，这个问题可能经常困扰你，这篇文章可以为你找到答案。
- [Billions of Messages a Day - Yelp's Real-time Data Pipeline](#)，Yelp公司每天十亿级实时消息的架构。
- [Uber: Building Reliable Reprocessing and Dead Letter Queues with Kafka](#)，Uber公司的Kafka应用。
- [Uber: Introducing Chaperone: How Uber Engineering Audits Kafka End-to-End](#)，Uber公司对Kafka消息的端到端审计。
- [Publishing with Apache Kafka at The New York Times](#)，纽约时报的Kafka工程实践。
- [Kafka Streams on Heroku](#)，Heroku公司的Kafka Streams实践。
- [Salesforce: How Apache Kafka Inspired Our Platform Events Architecture](#)，Salesforce的Kafka工程实践。
- [Exactly-once Semantics are Possible: Here's How Kafka Does it](#)，怎样用Kafka让只发送一次的语义变为可能。这是业界中一个很难的工程问题。
- [Delivering billions of messages exactly once](#) 同上，这也是一篇挑战消息只发送一次这个技术难题的文章。
- [Benchmarking Streaming Computation Engines at Yahoo!](#)，Yahoo!的Storm团队在为他们的流式计算做技术选型时，发现市面上缺乏针对不同计算平台的性能基准测试。于是，他们研究并设计了一种方案来做基准测试，测试了Apache Flink、Apache Storm和Apache Spark这三种平台。文中给出了结论和具体的测试方案。（如果原文链接不可用，请尝试搜索引擎对该网页的快照。）

关于日志方面

- [Using Logs to Build a Solid Data Infrastructure - Martin Kleppmann](#)，设计基于log结构应用架构的一篇不错的文章。
- [Building DistributedLog: High-performance replicated log service](#)，Distributed是Twitter 2016年5月份开源的一个分布式日志系统。在Twitter内部已经使用2年多。其主页在[distributedlog.io](#)。这篇文章讲述了这个高性能日志系统的一些技术细节。另外，其技术负责人是个中国人，其在微信公众号中也分享过这个系统[Twitter高性能分布式日志系统架构解析](#)。
- [LogDevice: a distributed data store for logs](#)，Facebook分布式日志系统方面的一些工程分享。

关于性能方面

- [Understand Latency](#)，这篇文章收集并整理了一些和系统响应时间相关的文章，可以让你全面了解和Latency有关的系统架构和设计经验方面的知识。
- [Common Bottlenecks](#)，文中讲述了20个常见的系统瓶颈。
- [Performance is a Feature](#)，Coding Horror上的一篇让你关注性能的文章。
- [Make Performance Part of Your Workflow](#)，这篇文章是图书《Designing for Performance》中的节选（国内没有卖的），其中给出来了一些和性能有关的设计上的平衡和美学。
- [CloudFlare: How we built rate limiting capable of scaling to millions of domains](#)，讲述了CloudFlare公司是怎样实现他们的限流功能的。从最简单的每客户IP限流开始分析，进一步讲到anycast，在这种情况下PoP的分布式限流是怎样实现的，并详细解释了具体的算法。

关于搜索方面

- [Instagram: Search Architecture](#)
- [eBay: The Architecture of eBay Search](#)
- [eBay: Improving Search Engine Efficiency by over 25%](#)
- [LinkedIn: Introducing LinkedIn's new search architecture](#)

- [LinkedIn: Search Federation Architecture at LinkedIn](#)
- [Slack: Search at Slack](#)
- [DoorDash: Search and Recommendations at DoorDash](#)
- [Twitter: Search Service at Twitter \(2014\)](#)
- [Pinterest: Manas: High Performing Customized Search System](#)
- [Sherlock: Near Real Time Search Indexing at Flipkart](#)
- [Airbnb: Nebula: Storage Platform to Build Search Backends](#)

各公司的架构实践

[High Scalability](#) , 这个网站会定期分享一些大规模系统架构是怎样构建的，下面是迄今为止各个公司的架构说明。

- [YouTube Architecture](#)
- [Scaling Pinterest](#)
- [Google Architecture](#)
- [Scaling Twitter](#)
- [The WhatsApp Architecture](#)
- [Flickr Architecture](#)
- [Amazon Architecture](#)
- [Stack Overflow Architecture](#)
- [Pinterest Architecture](#)
- [Tumblr Architecture](#)
- [Instagram Architecture](#)
- [TripAdvisor Architecture](#)
- [Scaling Mailbox](#)
- [Salesforce Architecture](#)
- [ESPN Architecture](#)
- [Uber Architecture](#)
- [DropBox Design](#)
- [Splunk Architecture](#)

小结

今天我们分享的内容是高手成长篇分布式架构部分的最后一篇——分布式架构工程设计，讲述了设计原则、设计模式等方面的内容，尤其整理和推荐了国内外知名企业的设计思路和工程实践，十分具有借鉴意义。

下篇文章中，我们将分享微服务架构方面的内容。敬请期待。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 - [分布式架构工程设计（分布式架构）](#)
 - [微服务](#)
 -

程序员练级攻略（2018）：微服务

2018-07-19 陈皓

微服务是分布式系统中最近比较流行的架构模型，也是SOA架构的一个进化。微服务架构并不是银弹，所以，也不要寄希望于微服务架构能够解决所有的问题。微服务架构主要解决的是如何快速地开发和部署我们的服务，这对于一个能够适应快速开发和成长的公司是非常必要的。同时我也觉得，微服务中有很多很不错的想法和理念，所以学习微服务是每一个技术人员迈向卓越的架构师的必经之路。

首先，你需要看一下，Martin Fowler的这篇关于微服务架构的文档 - [Microservice Architecture](#)（中译版），这篇文章说明了微服务的架构与传统架构的不同之处在于，微服务的每个服务与其数据库都是独立的，可以无依赖地进行部署。你也可以看看Martin Fowler老人家现身说法的[视频](#)。

另外，你还可以简单地浏览一下，各家对微服务的理解。

- [AWS的理解 - What are Microservices?](#)
- [Microsoft的理解 - Microservices architecture style](#)
- [Pivotal的理解 - Microservices](#)

微服务架构

接下来，你可以看一下 [IBM红皮书: Microservices Best Practices for Java](#)，这本书非常好，不但有通过把Spring Boot和Dropwizard来架建Java的微服务，而且还谈到了一些标准的架构模型，如服务注册、服务发现、API网关、服务通讯、数据处理、应用安全、测试、部署、运维等，是相当不错的一本书。

当然，有一本书你也可以读一下——[微服务设计](#)。这本书全面介绍了微服务的建模、集成、测试、部署和监控，通过一个虚构的公司讲解了如何建立微服务架构。主要内容包括认识微服务在保证系统设计与组织目标统一上的重要性，学会把服务集成到已有系统中，采用递增手段拆分单块大型应用，通过持续集成部署微服务，等等。

与此相似的，也有其它的一系列文章，值得一读。

下面是Nginx上的一组微服务架构的系列文章。

- [Introduction to Microservices](#)
- [Building Microservices: Using an API Gateway](#)
- [Building Microservices: Inter-Process Communication in a Microservices Architecture](#)
- [Service Discovery in a Microservices Architecture](#)
- [Event-Driven Data Management for Microservices](#)
- [Choosing a Microservices Deployment Strategy](#)
- [Refactoring a Monolith into Microservices](#)

下面这是 [Auto0 Blog](#) 上一系列的微服务的介绍，有代码演示。

- [An Introduction to Microservices, Part 1](#)
- [API Gateway: An Introduction to Microservices, Part 2](#)
- [An Introduction to Microservices, Part 3: The Service Registry](#)
- [Intro to Microservices, Part 4: Dependencies and Data Sharing](#)
- [API Gateway: the Microservices Superglue](#)

还有DZone的这个Spring boot的教程。

- [Microservices With Spring Boot - Part 1 - Getting Started](#)
- [Microservices With Spring Boot - Part 2 - Creating a Forex Microservice](#)
- [Microservices With Spring Boot - Part 3 - Creating Currency Conversion Microservice](#)
- [Microservices With Spring Boot - Part 4 - Using Ribbon for Load Balancing](#)
- [Microservices With Spring Boot - Part 5 - Using Eureka Naming Server](#)

当然，如果你要玩得时髦一些的话，我推荐你使用下面的这套架构。

- 前端：[React.js](#) 或 [Vue.js](#)。
- 后端：[Go语言](#) + 微服务工具集 [Go kit](#)，因为是微服务了，所以，每个服务的代码就简单了。既然简单了，也就可以用任何语言了，所以，我推荐Go语言。
- 通讯：[gRPC](#)，这是Google远程调用的一个框架，它比Restful的调用要快20倍到50倍的样子。
- API：[Swagger](#)，Swagger是一种Restful API的简单但强大的表示方式，标准的，语言无关，这种表示方式不但人可读，而且机器可读。可以作为Restful API的交互式文档，也可以作为Restful API形式化的接口描述，生成客户端和服务端的代码。今天，所有的API应该都通过Swagger来完成。
- 网关：[Envoy](#)，它包含了服务发现、负载均衡和熔断等这些特性，也是一个很有潜力的网关。当然，Kubernetes也是很好的，而且它也是高扩展的，所以，完全可以把Envoy通过Ingress集成进Kubernetes。这里有一个开源项目就是干这个事的 - [contour](#)。
- 日志监控：[fluentd](#) + [ELK](#)。
- 指标监控：[Prometheus](#)。
- 调用跟踪：[Jaeger](#) 或是 [Zipkin](#)，当然，后者比较传统一些，前者比较时髦，最重要的是，其可以和Prometheus和Envoy集成。
- 自动化运维：[Docker](#) + [Kubernetes](#)。

微服务和SOA

在对微服务有了一定的认识以后，一定有很多同学分不清楚微服务和SOA架构，对此，你可以看一下这本电子书 - [《Microservices vs. Service-Oriented Architecture》](#)。通过这本书，你可以学到，服务化架构的一些事实，还有基础的SOA和微服务的架构知识，以及两种架构的不同。这本书的作者马克·理查兹（Mark Richards）同学拥有十年以上的SOA和微服务架构的设计和实现的经验。

另外，还有几篇其它对比SOA和微服务的文章你也可以看看。

- [DZone: Microservices vs. SOA](#)
- [DZone: Microservices vs. SOA - Is There Any Difference at All?](#)
- [Microservices, SOA, and APIs: Friends or enemies?](#)

除此之外，我们还需要知道微服务和其它架构的一些不同和比较，这样我们就可以了解微服务架构的优缺点。下面几篇文章将帮助获得这些知识。

- [PaaS vs. IaaS for Microservices Architectures: Top 6 Differences](#)
- [Microservices vs. Monolithic Architectures: Pros, Cons, and How Cloud Foundry \(PaaS\) Can Help](#)
- [Microservices - Not A Free Lunch!](#)
- [The Hidden Costs Of Microservices](#)

设计模式和最佳实践

然后，你可以看一下微服务的一些设计模式。

- [Microservice Patterns](#)，微服务架构的设计模式和最佳实践。
- [Microservice Antipatterns and Pitfalls](#)，微服务架构的一些已知的反模式和陷阱。

• [Microservice Architecture: All The Best Practices You Need To Know](#), 这是一篇长文，里面讲诉了什么是微服务、微服务架构的优缺点、微服务最大的挑战和解决方案是什么、如何避免出错，以及构建微服务架构的最佳实践等多方面的内容。推荐阅读。

• [Best Practices for Building a Microservice Architecture](#), 这篇文章分享了构建微服务架构的最佳实践。

• [Simplicity by Distributing Complexity](#), 这是一篇讲如何使用事件驱动构建微服务架构的文章，其中有很多不错的设计上的基本原则。

相关资源

• [Microservices Resource Guide](#), 这个网页上是Martin Fowler为我们挑选的和微服务相关的文章、视频、书或是podcast。

• [Awesome Microservices](#), 一个各种微服务资源和相关项目的集中地。

小结

好了，总结一下今天的内容。我认为，微服务中有很多很不错的想法和理念，所以学习微服务是每一个技术人员迈向卓越的架构师的必经之路。在这篇文章中，我先给出了AWS、Microsoft和Pivotal对微服务的理解；然后给出了好几个系列的教程，帮你全面学习和理解微服务架构；然后通过一系列文章帮你来区分何为微服务，何为SOA；最后给出了微服务架构的设计模式和最佳实践，以及相关资源。相信通过这一系列内容的学习，你一定会对微服务有全面、透彻的理解。

下篇文章，我们将讲述的容器化和自动化运维方面的内容。敬请期待。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 - [分布式架构工程设计（分布式架构）](#)
 - [微服务](#)
 -[.....](#)



程序员练级攻略（2018）：容器化和自动化运维

2018-07-24 陈皓

这篇文章重点学习 Docker 和 Kubernetes，它们已经是分布式架构和自动化运维的必需品了，也是你必需要学习的。对于这两个东西，你千万不要害怕，因为技术方面都不算复杂，只是它们的玩法和传统运维不一样，所以你不用担心，只要你花上一点时间，一定会学好的。

Docker

- 你可以先看一下Docker的官方介绍 [Docker Overview](#)。
- 然后再去一个Web在线的Playground上体验一下，[Katacoda Docker Playground](#) 或者是 [Play With Docker](#)。
- 接下来，跟着 [Learn Docker](#) 这个文档中的教程自己安装一个Docker的环境，实操一把。
- 然后跟着 [Docker Curriculum](#) 这个超详细的教程玩一下Docker。

有了上述的一些感性体会之后，你就可以阅读Docker官方文档 [Docker Documentation](#) 了，这是学习Docker最好的方式。

如果你想了解一下Docker的底层技术细节，你可以参看我的文章。

- [Docker 基础技术: Linux Namespace \(上\)](#)
- [Docker 基础技术: Linux Namespace \(下\)](#)
- [Docker 基础技术: Cgroup](#)
- [Docker 基础技术: AUFS](#)
- [Docker 基础技术: DeviceMapper](#)

还有一些不错的与Docker网络有关的文章你需要阅读及实践一下。

- [A container networking overview](#)
- [Docker networking 101 - User defined networks](#)
- [Understanding CNI \(Container Networking Interface\)](#)
- [Using CNI with Docker](#)

Docker有下面几种网络解决方案：[Calico](#)、[Flannel](#) 和 [Weave](#)，你需要学习一下。另外，还需要学习一下 [netshoot](#)。这是一个很不错的用来诊断Docker网络问题的工具集。

关于这几个容器网络解决方案的性能对比，你可以看一下下面这几篇文章或报告。

- [Battlefield: Calico, Flannel, Weave and Docker Overlay Network](#)
- [Comparison of Networking Solutions for Kubernetes](#)
- [Docker Overlay Networks: Performance analysis in high-latency environments](#)

如果你对Docker的性能有什么问题的话，你可以看一下下面这些文章。

- [IBM Research Report: An Updated Performance Comparison of Virtual Machines and Linux Containers](#)
- [An Introduction to Docker and Analysis of its Performance](#)

下面是一些和存储相关的文章。

- [Storage Concepts in Docker: Network and Cloud Storage](#)
- [Storage Concepts in Docker: Persistent Storage](#)
- [Storage Concepts in Docker: Shared Storage and the VOLUME directive](#)

然后是跟运维相关的文章。

- [Docker Monitoring with the ELK Stack: A Step-by-Step Guide](#)

最后，推荐看看 [Valuable Docker Links](#)，其中收集并罗列了一系列非常不错的 Docker 文章。

最佳实践

下面分享一些与Docker相关的最佳实践。

- [Best Practices for Dockerfile](#)，Docker官方文档里的Dockerfile的最佳实践。
- [Docker Best Practices](#)，这里收集汇总了存在于各个地方的使用Docker的建议和实践。
- [Container Best Practices](#)，来自Atomic项目，是一个介绍容器化应用程序的架构、创建和管理的协作型文档项目。
- [Eight Docker Development Patterns](#)，八个Docker的开发模式：共享基础容器、共享同一个卷的多个开发容器、开发工具专用容器、测试环境容器、编译构建容器、防手误的安装容器、默认服务容器、胶黏容器。

Kubernetes

Kubernetes 是Google开源的容器集群管理系统，是Google多年大规模容器管理技术Borg的开源版本，也是CNCF最重要的项目之一，主要功能包括：

- 基于容器的应用部署、维护和滚动升级；
- 负载均衡和服务发现；
- 跨机器和跨地区的集群调度；
- 自动伸缩；
- 无状态服务和有状态服务；
- 广泛的Volume支持；
- 插件机制保证扩展性。

Kubernetes发展非常迅速，已经成为容器编排领域的领导者。

首先，我推荐你阅读Kubernetes前世今生的一篇论文。

- [Borg, Omega, and Kubernetes](#)，看看Google这十几年来从这三个容器管理系统中得到的经验教训。

学习kubernetes，有两个免费的开源电子书。

- [《Kubernetes Handbook》](#)，这本书记录了作者从零开始学习和使用Kubernetes的心路历程，着重于经验分享和总结，同时也会有相关的概念解析。希望能够帮助你少踩坑，少走弯路，还会指引你关注kubernetes生态周边，如微服务构建、DevOps、大数据应用、Service Mesh、Cloud Native等领域。
- [《Kubernetes指南》](#)，这本书旨在整理平时在开发和使用Kubernetes时的参考指南和实践总结，形成一个系统化的参考指南以方便查阅。

这两本电子书都不错，前者更像是一本学习教程，而且面明显广一些，还包括Cloud Natvie、Service Mesh以及微服务相关的东西。而后者聚焦于Kubernetes本身，更像一本参考书。

但是也别忘了Kubernetes的官方网站：[Kubernetes.io](#)，上面不但有[全面的文档](#)，也包括一个很不错的[官方教程](#)。

此外，还有一些交互式教程，帮助你理解掌握，以及一些很不错的文章推荐你阅读。

一些交互式教程

- [Katacoda](#)
- [Kubernetes Bootcamp](#)

一些文章

这里还有一些不错的文档，你应该去读一下。

- [Kubernetes tips & tricks](#)
- [Achieving CI/CD with Kubernetes](#)
- [How to Set Up Scalable Jenkins on Top of a Kubernetes Cluster](#)
- 10 Most Common Reasons Kubernetes Deployments Fail [Part I](#) 和 [Part II](#)
- [How to Monitor Kubernetes](#)，一共有4个篇章
- [Logging in Kubernetes with Fluentd and Elasticsearch](#)
- [Kubernetes Monitoring: Best Practices, Methods, and Existing Solutions](#)

网络相关的文章

要学习Kubernetes，你只需要读一下，下面这个Kubernetes 101系列的文章。

- [Kubernetes 101 - Networking](#)
- [Kubernetes networking 101 - Pods](#)
- [Kubernetes networking 101 - Services](#)
- [Kubernetes networking 101 - \(Basic\) External access into the cluster](#)
- [Kubernetes Networking 101 - Ingress resources](#)
- [Getting started with Calico on Kubernetes](#)

CI/CD相关的文章

- [Automated Image Builds with Jenkins, Packer, and Kubernetes](#)
- [Jenkins setups for Kubernetes and Docker Workflow](#)
- [Lab: Build a Continuous Deployment Pipeline with Jenkins and Kubernetes](#)

最佳实践

- [Kubernetes Best Practices](#) by [Sachin Arote](#)，AWS工程师总结的最佳实践。
- [Kubernetes Best Practices](#) by [Sandeep Dinesh](#)，Google云平台工程师总结的最佳实践。

Docker和Kubernetes资源汇总

下面是 Github 上和 Docker & Kubernetes相关的Awesome系列。

- [Awesome Docker](#)。
- [Awesome Kubernetes](#)。

虽然上面的这些系列非常全的罗列了很多资源，但是我觉得很不系统。对于系统的说明Docker和Kubernetes生态圈，我非常推荐大家看一下 The New Stack 为Kubernetes出的一系列的电子书或报告。

- [The New Stack eBook Series](#)，非常完整和详实的 Docker 和 Kubernetes 生态圈的所有东西。

- Book 01: [The Docker Container Ecosystem](#)
- Book 02: [Applications & Microservices with Docker & Containers](#)
- Book 03: [Automation & Orchestration with Docker & Containers](#)
- Book 04: [Network, Security & Storage with Docker & Containers](#)
- Book 05: [Monitoring & Management with Docker & Containers](#)
- Book 06: [Use Cases for Kubernetes](#)
- Book 07: [State of the Kubernetes Ecosystem](#)
- Book 08: [Kubernetes Deployment & Security Patterns](#)
- Book 09: [CI/CD with Kubernetes](#)
- Book 10: [Kubernetes solutions Directory](#)
- Book 11: [Guid to Cloud-Native Microservices](#)

小结

总结一下今天的内容。Docker 和 Kubernetes已经成为分布式架构和自动化运维方面的不可或缺的两大基本构成，是你必需要学习的。虽然它们的玩法跟传统运维不一样，但技术方面并不算复杂，只要你花上一点时间，一定学会好的。

在这篇文章中，我推荐了Docker和Kubernetes基础技术方面的学习资料，并给出了存储、运维、网络、CI/CD等多方面的资料，同时列出了与之相关的最佳实践。相信认真学习和消化这些知识，你一定可以掌握Docker和Kubernetes两大利器。

下篇文章，我们将学习机器学习和人工智能方面的内容。敬请期待。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)

- 分布式架构经典图书和论文 (分布式架构)
- 分布式架构工程设计(分布式架构)
- 微服务
- 分布式架构工程设计
- 容器化和自动化运维
-

左耳听风

洞悉技术的本质
享受科技的乐趣



陈皓

资深技术专家
骨灰级程序员



扫 码 订 阅

我之前写过一篇机器学习的入门文章，因为我也是在入门和在学习的人，所以，那篇文章和这篇机器学习和人工智能方向的文章可能都会有点太肤浅。如果你有更好的学习方式或资料，欢迎补充。

基本原理简介

我们先来介绍一下机器学习的基本原理。

机器学习主要有两种方式，一种是监督式学习（Supervised Learning），另一种是非监督式学习（Unsupervised Learning）。下面简单地说一下这两者的不同。

- **监督式学习（Supervised Learning）**。所谓监督式学习，也就是说，我们需要提供一组学习样本，包括相关的特征数据和相应的标签。我们的程序可以通过这组样本来学习相关的规律或是模式，然后通过得到的规律或模式来判断没有被打过标签的数据是什么样的数据。

举个例子，假设需要识别一些手写的数字，我们要找到尽可能多的手写体的数字的图像样本，然后人工或是通过某种算法来明确地标注上什么是这些手写体的图片，谁是1，谁是2，谁是3……。这组数据叫样本数据，又叫训练数据（training data）。然后通过机器学习的算法，找到每个数字在不同手写体下的特征，找到规律和模式。通过得到的规律或模式来识别那些没有被打过标签的手写数据，以此完成识别手写体数字的目的。

- **非监督式学习（Unsupervised Learning）**。对于非监督式学习，也就是说，数据是没有被标注过的，所以相关的机器学习算法需要找到这些数据中的共性。因为大量的数据是没被标识过的，所以这种学习方式可以让大量的未标识的数据能够更有价值。而且，非监督式学习，可以为我们找到人类很难发现的数据里的规律或模型，所以也有人称这种学习为“特征点学习”，其可以让我们自动地为数据进行分类，并找到分类的模型。

一般来说，非监督式学习会应用在一些交易型的数据中。比如，你有一堆堆的用户购买数据，但是对于人类来说，我们很难找到用户属性和购买商品类型之间的关系。所以，非监督式学习算法可以帮助我们找到它们之间的关系。比如，一个在某个年龄段的女性购买了某种肥皂，有可能说明这个女性在怀孕期，或是某人购买儿童用品，有可能说明这个人的关系链中有孩子，等等。于是，这些信息会被用作一些所谓的精准市场营销活动，从而可以增加商品销量。

我们这么来说吧，监督式学习是在被告知过了正确的答案后的学习，而非监督式学习是在没有被告知正确答案时的学习。所以，非监督式学习是在大量的非常乱的数据中找寻一些潜在的关系，这个成本也比较高。非监督式学习经常被用来检测一些不正常的事情发生，比如信用卡的诈骗或是盗刷。也被用在推荐系统，比如买了这个商品的人又买了别的什么商品，或是如果某个人喜欢某篇文章、某个音乐、某个餐馆，那么他可能会喜欢某个车、某个明星或某个地方。

在监督式学习算法下，我们可以用一组“狗”的照片来确定某个照片中的物体是不是狗。而在非监督式学习算法下，我们可以通过一个照片来找到其中有与其相似的事物的照片。这两种学习方式都有些有用的场景。

关于机器学习，你可以读一读 [Machine Learning is Fun!](#)，这篇文章（[中文翻译版](#)）恐怕是全世界最简单的入门资料了。

- [Data Science Simplified Part 1: Principles and Process](#)
- [Data Science Simplified Part 2: Key Concepts of Statistical Learning](#)
- [Data Science Simplified Part 3: Hypothesis Testing](#)
- [Data Science Simplified Part 4: Simple Linear Regression Models](#)
- [Data Science Simplified Part 5: Multivariate Regression Models](#)
- [Data Science Simplified Part 6: Model Selection Methods](#)
- [Data Science Simplified Part 7: Log-Log Regression Models](#)
- [Data Science Simplified Part 8: Qualitative Variables in Regression Models](#)
- [Data Science Simplified Part 9: Interactions and Limitations of Regression Models](#)
- [Data Science Simplified Part 10: An Introduction to Classification Models](#)
- [Data Science Simplified Part 11: Logistic Regression](#)

相关课程

接下来，我们需要比较专业地学习一下机器学习了。

在学习机器学习之前，我们需要学习数据分析，所以，我们得先学一些大数据相关的东西，也就是Data Science相关的内容。下面是两个不错的和数据科学相关的教程以及一个资源列表。

- [UC Berkeley's Data 8: The Foundations of Data Science](#) 和电子书 [Computational and Inferential Thinking](#) 会讲述数据科学方面非常关键的概念，会教会你在数据中找到数据的关联、预测和相关的推断。
- [Learn Data Science](#)，这是GitHub上的一本电子书，主要是一些数据挖掘的算法，比如线性回归、逻辑回归、随机森林、K-Means聚类的数据分析。然后，[donnemartin/data-science-ipython-notebooks](#) 这个代码仓库中用TensorFlow、scikit-learn、Pandas、NumPy、Spark等把这些经典的例子实现了个遍。
- [Data Science Resources List](#)，这个网站上有一个非常长的和数据科学相关的资源列表，你可以从中得到很多你想要的东西。

之后，有下面几门不错的在线机器学习的课程供你入门，也是非常不错。

- 吴恩达教授（Andrew Ng）在 [Coursera 上的免费机器学习课程](#) 非常棒。我强烈建议从此入手。对于任何拥有计算机或科学学位的人，或是还能记住一点点数学知识的人来说，都应该非常容易入门。这个斯坦福大学的课程请尽量拿满分。可以在 [网易公开课](#) 中找到这一课程。除此之外，吴恩达教授还有一组新的和深度学习相关的课程，现在可以在网易公开课上免费学习——[Deep Learning Specialization](#)。
- [Deep Learning by Google](#)，Google的一个关于深度学习的在线免费课程，其支持中英文。这门课会教授你如何训练和优化基本神经网络、卷积神经网络和长短期记忆网络。你将通过项目和任务接触完整的机器学习系统TensorFlow。
- 卡内基梅隆大学汤姆·米切尔（Tom Mitchell）的机器学习 [英文原版视频与课件PDF](#)。
- 2013年加利福尼亚理工学院亚瑟·阿布-穆斯塔法（Yaser Abu-Mostafa）的Learning from Data [课程视频及课件PDF](#)，内容更适合进阶。
- 关于神经网络方面，YouTube上有一个非常火的课程视频，由宾夕法尼亚大学的雨果·拉罗歇尔（Hugo Larochelle）的教学课程 - [Neural networks class - Université de Sherbrooke](#)。

除此之外，还有很多的在线大学课程你可以学习。比如：

- 斯坦福大学的《[统计学学习](#)》、《[机器学习](#)》、《[卷积神经网络](#)》、《[深度学习之自然语言处理](#)》等。
- 麻省理工学院的《[神经网络介绍](#)》、《[机器学习](#)》、《[预测](#)》等。

更多的列表，请参看——[Awesome Machine Learning Courses](#)。

相关图书

- [Pattern Recognition and Machine Learning](#)，这本书是机器学习领域的圣经之作。该书也是众多高校机器学习研究生课程的教科书，Google上有[PDF版的下载](#)。这本书很经典，但并不适合入门来看。GitHub上有这本中的[Matlab 实现](#)。
- 下面这两本电子书也是比较经典的，其中讲了很多机器学习的知识，可以当做手册或字典。
 - 《[Understanding Machine Learning: From Theory to Algorithms](#)》。
 - 《[The Elements of Statistical Learning - Second Edition](#)》。
- [Deep Learning: Adaptive Computation and Machine Learning series](#) 中文翻译为《深度学习》，又叫“花书”。这本书由全球知名的三位专家伊恩·古德费洛（Ian Goodfellow）、肖华·本吉奥（Yoshua Bengio）和亚伦·考维尔（Aaron Courville）撰写，是深度学习领域奠基性的经典教材。

全书内容包括3部分：第1部分介绍基本的数学工具和机器学习的概念，它们是深度学习的预备知识；第2部分系统深入地讲解现今已成熟的深度学习方法和技术；第3部分讨论某些具有前瞻性的方向和想法，它们被公认为是深度学习未来的研究重点。这本书的官网为“[deeplearningbook.org](#)”，在GitHub上也有中文翻译 - [《Deep Learning 中文翻译》](#)。

- [《Neural Networks and Deep Learning》（中文翻译版）](#)，这是一本非常不错的神经网络的入门书，在豆瓣上评分9.5分。从理论讲到了代码。虽然有很多数学公式，但是有代码相助，就

不难理解了。其中讲了很多如激活函数、代价函数、随机梯度下降、反向传播、过度拟合和规范化、权重初始化、超参数优化、卷积网络的局部感受野、混合层、特征映射的东西。

- 《[Introduction to Machine Learning with Python](#)》，算是本不错的入门书，也是本比较易读的英文书。其是以Scikit-Learn框架来讲述的。如果你用过Scikit这个框架，那么你学这本书还是很不错的。
- 《[Hands-On Machine Learning with Scikit-Learn and TensorFlow](#)》，这是一门以TensorFlow为工具的入门书，其用丰富的例子从实践的角度来让你学习。这本书对于无基础的人也是很适合的，对于小白来说虽然略难但是受益匪浅。

相关文章

除了上述的那些课程和图书外，下面这些文章也很不错。

- YouTube 上的 Google Developers 的 [Machine Learning Recipes with Josh Gordon](#)，这9集视频，每集不到10分钟，从Hello World讲到如何使用TensorFlow，非常值得一看。
- 还有 [Practical Machine Learning Tutorial with Python Introduction](#) 上面一系列的用Python带着你玩Machine Learning的教程。
- Medium上的 [Machine Learning - 101](#)，讲述了好些我们上面提到过的经典算法。
- Medium上的 [Machine Learning for Humans](#)。
- [Dr. Jason Brownlee 的博客](#)，也非常值得一读，其中好多的“How-To”，会让你有很多的收获。
- [Rules of Machine Learning: Best Practices for ML Engineering](#)，一些机器学习相关的最佳实践。
- [i am task](#)，也是一个很不错的博客。
- 关于Deep Learning中的神经网络，YouTube上有介绍视频 [Neural Networks](#)。
- 麻省理工学院的电子书 [Deep Learning](#)。
- 用Python做自然语言处理[Natural Language Processing with Python](#)。
- 最后一个是Machine Learning和Deep Learning的相关教程列表，[Machine Learning & Deep Learning Tutorials](#)。

下面是一些和神经网络相关的不错的文章。

- [The Unreasonable Effectiveness of Recurrent Neural Networks](#)，这是一篇必读的文章，告诉你为什么要学RNN，以及展示了最简单的NLP形式。
- [Neural Networks, Manifolds, and Topology](#)，这篇文章可以帮助你理解神经网络的一些概念。
- [Understanding LSTM Networks](#)，解释了什么是LSTM的内在工作原理。
- [Attention and Augmented Recurrent Neural Networks](#)，用了好多图来说明了RNN的attention机制。
- [Recommending music on Spotify with deep learning](#)，一个在Spotify的实习生分享的音乐聚类的文章。

相关算法

下面是10个非常经典的机器学习的算法。

- 对于监督式学习，有如下经典算法。
 1. [决策树 \(Decision Tree\)](#)，比如自动化放贷、风控。
 2. [朴素贝叶斯分类器 \(Naive Bayesian classifier\)](#)，可以用于判断垃圾邮件、对新闻的类别进行分类，比如科技、政治、运动、判断文本表达的感情是积极的还是消极的、人脸识别等。
 3. [最小二乘法 \(Ordinary Least Squares Regression\)](#)，是一种线性回归。
 4. [逻辑回归 \(Logistic Regression\)](#)，一种强大的统计学方法，可以用一个或多个变量来表示一个二项式结果。可以用于信用评分，计算营销活动的成功率，预测某个产品的收入。
 5. [支持向量机 \(Support Vector Machine, SVM\)](#)，可以用于基于图像的性别检测、图像分类等。
 6. [集成方法 \(Ensemble methods\)](#)，通过构建一组分类器，然后通过它们的预测结果进行加权投票来对新的数据点进行分类。原始的集成方法是贝叶斯平均，但最近的算法包括纠错输出编码、Bagging和Boosting。
- 对于无监督式的学习，有如下经典算法。
 1. [聚类算法 \(Clustering Algorithms\)](#)。聚类算法有很多，目标是给数据分类。有5个比较著名的聚类算法你必需要知道：[K-Means](#)、[Mean-Shift](#)、[DBSCAN](#)、[EM/GMM](#)、和[Agglomerative Hierarchical](#)。
 2. [主成分分析 \(Principal Component Analysis, PCA\)](#)。PCA的一些应用包括压缩、简化数据便于学习、可视化等。
 3. [奇异值分解 \(Singular Value Decomposition, SVD\)](#)。实际上，PCA是SVD的一个简单应用。在计算机视觉中，第一个人脸识别算法使用PCA和SVD来将面部表示为“特征面”的线性组合，进行降维，然后通过简单的方法将面部匹配到身份。虽然现代方法更复杂，但很多方面仍然依赖于类似的技术。
 4. [独立成分分析 \(Independent Component Analysis, ICA\)](#)。ICA是一种统计技术，主要用于揭示随机变量、测量值或信号集中的隐藏因素。

如果你想了解更全的机器学习的算法列表，你可以看一下Wikipedia上的 [List of Machine Learning Algorithms](#)。

在 [A Tour of Machine Learning Algorithms](#)，这篇文章带你概览了一些机器学习算法，其中还有一个“脑图”可以下载，并还有一些How-To的文章供你参考。

对于这些算法，[SciKit-Learn](#)有一些文档供你学习。

- [1. Supervised learning](#)
- [2.3 Clustering](#)
- [2.5. Decomposing signals in components \(matrix factorization problems\)](#)
- [3. Model selection and evaluation](#)
- [4.3. Preprocessing data](#)

相关资源

- 对于初学者来说，动手是非常非常重要的，不然，你会在理论的知识里迷失掉自己，这里有篇文章“[8 Fun Machine Learning Projects for Beginners](#)”，其中为初学者准备了8个很有趣的项目，你可以跟着练习。
- 学习机器学习或是人工智能你需要数据，这里有一个非常足的列表给你足够多的公共数据 – [《Awesome Public Datasets》](#)，其中包括农业、生物、天气、计算机网络、地球科学、经济、教育、金融、能源、政府、健康、自然语言、体育等。
- GitHub上的一些Awesome资源列表。
 - [Awesome Deep Learning](#)
 - [Awesome - Most Cited Deep Learning Papers](#)
 - [Awesome Deep learning papers and other resources](#)

小结

总结一下今天的内容。我首先介绍了机器学习的基本原理：监督式学习和非监督式学习，然后给出了全世界最简单的入门资料 [Machine Learning is Fun!](#)。随后给出了与机器学习密切相关的数据仓库方面的文章和资料，然后推荐了几门学习机器学习知识的在线课程。最后一部分是关于深度学习的资料，最后给出了机器学习的入门书籍以及相关的学习资料。

据分析方面的内容和资料，然后推荐了深入学习机器学习知识的在线课程、图书和文章等，尤其列举了神经网络方面的学习资料。最后描述了机器学习的十大经典算法及相关的学习资料。

在机器学习和人工智能领域，我也在学习，也处于入门阶段，所以本文中推荐的内容，可能在你看来会有些浅。如果你有更好的信息和资料，欢迎补充。目前文章中给出来的是，我在学习过程中认为很不错的内容，我从中受益良多，所以希望它们也能为你的学习提供帮助。

从下篇文章开始，我们将进入前端知识的学习，包括基础和底层原理、性能优化、前端框架、UI/UX设计等内容。敬请期待。

下面是《程序员进阶攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 - [分布式架构工程设计（分布式架构）](#)
 - [微服务](#)
 - [分布式架构工程设计](#)
 - [容器化和自动化运维](#)
 - [机器学习和人工智能](#)
 -



对于前端的学习和提高，我的基本思路是这样的。首先，前端的三个最基本的东西HTML5、CSS3和JavaScript（ES6）是必需要学好的。这其中有很多很多的技术，比如，CSS3引申出来的Canvas（位图）、SVG（矢量图）和WebGL（3D图），以及CSS的各种图形变换可以让你做出非常丰富的渲染效果和动画效果。

ES6简直就是把JavaScript带到了一个新的台阶，JavaScript语言的强大，大大释放了前端开发人员的生产力，让前端得以开发更为复杂的代码和程序，于是像React和Vue这样的框架开始成为前端编程的不二之选。

我一直认为学习任何知识都要从基础出发，所以我会有很大的篇幅在讲各种技术的基础知识和基本原理，尤其是如下的这些知识，都是前端程序员需要一块一块啃掉的硬骨头。

- **JavaScript的核心原理**。这里我会给出好些网上很不错的讲JavaScript的原理的文章或图书，你一定要学好语言的特性和其中的各种坑。
- **浏览器的工作原理**。这也是一块硬骨头，我觉得这是前端程序员需要了解和明白的东西，不然，你将无法深入下去。
- **网络协议HTTP**。也是要着重了解的，尤其是HTTP/2，还有HTTP的几种请求方式：短连接、长连接、Stream连接、WebSocket连接。
- **前端性能调优**。有了以上的这些基础后，你就可以进入前端性能调优的主题了，我相信你可以很容易上手各种性能调优技术的。
- **框架学习**。我只给了React和Vue两个框架。就这两个框架来说，Virtual DOM技术是其底层技术，组件化是其思想，管理组件的状态是其重点。而对于React来说，函数式编程又是其编程思想，所以，这些基础技术都是你需要好好研究和学习的。
- **UI设计**。设计也是前端需要做的事情，比如像Google的Material UI，或是比较流行的Atomic Design等应该是前端工程师需要学习的。

而对于工具类的东西，这里我基本没怎么涉及，因为本文主要还是从原理和基础入手。那些工具我觉得都很简单，就像学习Java我没有让你去学习Maven一样，因为只要你去动手了，这种知识你自然就会获得，我们还是把精力重点放在更重要的地方。

下面我们从前端基础和底层原理开始讲起。先来讲讲HTML5相关的内容。

HTML5

HTML5主要有以下几本书推荐。

- [HTML5权威指南](#)，本书面向初学者和中等水平Web开发人员，是牢固掌握HTML5、CSS3和JavaScript的必读之作。书看起来比较厚，是因为里面的代码很多。
- [HTML5 Canvas核心技术](#)，如果你要做HTML5游戏的话，这本书必读。

对于SVG、Canvas和WebGL这三个对应于矢量图、位图和3D图的渲染来说，给前端开发带来了重武器，很多HTML5小游戏也因此蓬勃发展。所以，你可以学习一下。

学习这三个技术，我个人觉得最好的地方是MDN。

- [SVG: Scalable Vector Graphics](#)
- [Canvas API](#)
- [The WebGL API: 2D and 3D graphics for the web](#)

最后是几个资源列表。

- [Awesome HTML5](#)。GitHub上的Awesome HTML5，其中有大量的资源和技术文章。
- [Awesome SVG](#)
- [Awesome Canvas](#)
- [Awesome WebGL](#)

CSS

在《程序员练级攻略（2018）》系列文章最开始，我们就推荐过CSS的在线学习文档，这里再推荐一下[MDN Web Doc - CSS](#)。我个人觉得只要你仔细读一下文档，CSS并不难学。绝大多数觉得难的，一方面是文档没读透，另一方面是浏览器支持的标准不一致。所以，学好CSS最关键的是要仔细地读文档。

之后，在写CSS的时候，你会发现，你的CSS中有很多看起来相似的东西。你的DRY - Don't Repeat Yourself告诫告诉你，这是不对的。所以，你需要学会使用[LESS](#)和[Sass](#)这两个CSS预处理器，其可以帮你提高很多效率。

然后，你需要学习一下CSS的书写规范，前面的《程序员修养》一文中提到过一些，这里再补充几个。

- [Principles of writing consistent, idiomatic CSS](#)
- [Opinionated CSS styleguide for scalable applications](#)
- [Google HTML/CSS Style Guide](#)

如果你需要更有效率，那么你还需要使用一些CSS Framework，其中最著名的就是Twitter公司的[Bootstrap](#)，其有很多不错的UI组件，页面布局方案，可以让你非常方便也非常快速地开发页面。除此之外，还有，主打清新UI的[Semantic UI](#)、主要响应式界面的[Foundation](#)和基于Flexbox的[Bulma](#)。

当然，在使用CSS之前，你需要把你浏览器中的一些HTML标签给标准化掉。所以，推荐几个Reset或标准化的CSS库：[Normalize](#)、[MiniRest.css](#)、[sanitize.css](#)和[unstyle.css](#)。

关于更多的CSS框架，你可以参看[Awesome CSS Frameworks](#)上的列表。

接下来，是几个公司的CSS相关实践，供你参考。

- [CodePen's CSS](#)
- [Github 的 CSS](#)
- [Medium's CSS is actually pretty f***ing good](#)
- [CSS at BBC Sport](#)
- [Refining The Way We Structure Our CSS At Trello](#)

最后是一个可以写出可扩展的CSS的阅读列表[A Scalable CSS Reading List](#)。

JavaScript

下面是学习JavaScript的一些图书和文章。

- [JavaScript: The Good Parts](#)，中文翻译版为《JavaScript语言精粹》。这是一本介绍JavaScript语言本质的权威图书，值得任何正在或准备从事JavaScript开发的人阅读，并且需要反复阅读。学习、理解、实践大师的思想，我们才可能站在巨人的肩上，才有机会超越大师，这本书就是开始。
- [Secrets of the JavaScript Ninja](#)，中文翻译版为《JavaScript忍者秘籍》，本书是jQuery库创始人编写的一本深入剖析JavaScript语言的书。适合具备一定JavaScript基础知识的读者阅读，也适合从事程序设计工作并想要深入探索JavaScript语言的读者阅读。这本书有很多晦涩难懂的地方，需要仔细阅读，反复琢磨。
- [Effective JavaScript](#)，Ecma的JavaScript标准化委员会著名专家撰写，作者凭借多年标准化委员会工作和实践经验，深刻辨析JavaScript的内部运作机制、特性、陷阱和编程最佳实践，将它们高度浓缩为极具实践指导意义的68条精华建议。
- 接下来是ES6的学习，这里给三个学习手册源。
 - [ES6 in Depth](#)，InfoQ上有相关的中文版 - [ES6 深入浅出](#)。还可以看看[A simple interactive ES6 Feature list](#)，或是看一下[阮一峰翻译的ES6的教程](#)。
 - [ECMAScript 6 Tools](#)，这是一堆ES6工具的列表，可以帮助你提高开发效率。
 - [Modern JS Cheatsheet](#)，这个Cheatsheet在GitHub上有1万6千颗星，你就可见其影响力了。

- 然后，还有一组很不错的《[You Don't Know JS 系列](#)》的书。

- [You Don't Know JS: "Up & Going"](#)
- [You Don't Know JS: "Scope & Closures"](#)
- [You Don't Know JS: "this & Object Prototypes"](#)
- [You Don't Know JS: "Types & Grammar"](#)
- [You Don't Know JS: "Async & Performance"](#)
- [You Don't Know JS: "ES6 & Beyond"](#)

- 接下来是一些和编程范式相关的文章。

- [Glossary of Modern JavaScript Concepts: Part 1](#)，首先推荐这篇文章，其中收集了一些编程范式方面的内容，比如纯函数、状态、可变性和不可变性、指令型语言和声明式语言、函数式编程、响应式编程、函数式响应编程。
- [Glossary of Modern JavaScript Concepts: Part 2](#)，在第二部分中主要讨论了作用域和闭包，数据流，变更检测，组件化.....

- 下面三篇文章是德米特里·索什尼科夫 (Dmitry Soshnikov) 个人网站上三篇讲JavaScript内的文章。

- [JavaScript. The Core: 2nd Edition](#)
- [JavaScript. The Core \(older ES3 version\)](#)
- [JS scope: static, dynamic, and runtime-augmented](#)

- “[How JavaScript Works](#)”是一组非常不错的文章（可能还没有写完），强烈推荐。这一系列的文章是SessionStake的CEO写的，现在有13篇，我感觉可能还没有写完。这个叫[亚历山大·兹拉特科夫 \(Alexander Zlatkov\)](#) 的CEO太猛了。

- [An overview of the engine, the runtime, and the call stack](#)
- [Inside the V8 engine + 5 tips on how to write optimized code](#)，了解V8引擎。这里，也推荐 [Understanding V8's Bytecode](#) 这篇文章可以让你了解V8引擎的底层字节码。
- [Memory management + how to handle 4 common memory leaks](#)，内存管理和4种常见的内存泄露问题。
- [Event loop and the rise of Async programming + 5 ways to better coding with async/await](#)，Event Loop和异步编程。
- [Deep dive into WebSockets and HTTP/2 with SSE + how to pick the right path](#)，WebSocket和HTTP/2。
- [A comparison with WebAssembly + why in certain cases it's better to use it over JavaScript](#)，JavaScript内在原理。
- [The building blocks of Web Workers + 5 cases when you should use them](#)，Web Workers技术。
- [Service Workers, their lifecycle and use cases](#)，Service Worker技术。
- [The mechanics of Web Push Notifications](#)，Web端Push通知技术。
- [Tracking changes in the DOM using MutationObserver](#)，Mutation Observer技术。
- [The rendering engine and tips to optimize its performance](#)，渲染引擎和性能优化。
- [Inside the Networking Layer + How to Optimize Its Performance and Security](#)，网络性能和安全相关。
- [Under the hood of CSS and JS animations + how to optimize their performance](#)，CSS和JavaScript动画性能优化。

- 接下来是Google Chrome工程师 [阿迪·奥斯马尼 \(Addy Osmani\)](#) 的几篇JavaScript性能相关的文章，也是非常好的。

- [The Cost Of JavaScript](#)
- [JavaScript Start-up Performance](#)

- 其它与JavaScript相关的资源。

- [JavaScript has Unicode Problem](#)，这是一篇很有价值的JavaScript处理Unicode的文章。
- [JavaScript Algorithms](#)，用JavaScript实现的各种基础算法库。
- [JavaScript 30 秒代码](#)，一堆你可以在30秒内看懂各种有用的JavaScript的代码，在GitHub上有2万颗星了。
- [What the f*ck JavaScript](#)，一堆JavaScript搞笑和比较tricky的样例。
- [Airbnb JavaScript Style Guide](#)，Airbnb的JavaScript的代码规范，GitHub上有7万多颗星。
- [JavaScript Patterns for 2017](#)，YouTube上的一个JavaScript模式分享，值得一看。

浏览器原理

你需要了解一下浏览器是怎么工作的，所以，你必需要看《[How browsers work](#)》。这篇文章受众之大，后来被人重新整理并发布为《[How Browsers Work: Behind the scenes of modern web browsers](#)》，其中还包括中文版。这篇文章非常非常长，所以，你要有耐心看完。如果你想看个精简版的，可以看我在Coolshell上发的《[浏览器的渲染原理简介](#)》或是看一下[这个幻灯片](#)。

然后，是对Virtual DOM的学习。Virtual DOM是React的一个非常核心的技术细节，它也是前端渲染和性能的关键技术。所以，你有必要好好学习一下这个技术的实现原理和算法。当然，前提条件是你需要学习过前面我所推荐过的浏览器的工作原理。下面是一些不错的文章可以帮你学习这一技术。

- [How to write your own Virtual DOM](#)
- [Write your Virtual DOM 2: Props & Events](#)
- [How Virtual-DOM and diffing works in React](#)
- [The Inner Workings Of Virtual DOM](#)
- [深度剖析：如何实现一个 Virtual DOM 算法](#)
- 以及两个Virtual-DOM实现供你参考：
 - [Matt-Esch/Virtual-DOM](#)
 - [Maquette](#)

网络协议

- [High Performance Browser Networking](#)，本书是谷歌公司高性能团队核心成员的权威之作，堪称实战经验与规范解读完美结合的产物。本书目标是涵盖Web开发者技术体系中应该掌握的所有网络及性能优化知识。

全书以性能优化为主线，从TCP、UDP 和TLS协议讲起，解释了如何针对这几种协议和基础设施来优化应用。然后深入探讨了无线和移动网络的工作机制。最后，揭示了HTTP协议的底层细节，同时详细介绍了HTTP 2.0、XHR、SSE、WebSocket、WebRTC和DataChannel等现代浏览器新增的能力。

- 另外，[HTTP/2](#)也是HTTP的一个新的协议，于2015年被批准通过，现在基本上所有的主流浏览器都默认启用这个协议。所以，你有必要学习一下这个协议。下面相关的学习资源。

- [Gitbook - HTTP/2详解](#)
- [http2 explained \(中译版\)](#)
- [HTTP/2 for a Faster Web](#)
- [Nginx HTTP/2 白皮书](#)
- HTTP/2的两个RFC：
 - [RFC 7540 - Hypertext Transfer Protocol Version 2 \(HTTP/2\)](#)，HTTP/2的协议本身。

- 新的HTML5支持 [WebSocket](#), 所以, 这也是你要学的一个重要协议。

◦ [HTML5 WebSocket: A Quantum Leap in Scalability for the Web](#), 这篇文章比较了HTTP的几种链接方式, Polling、Long Polling和Streaming, 并引入了终极解决方案WebSocket。你知道的, 了解一个技术的缘由是非常重要的。

◦ [StackOverflow: My Understanding of HTTP Polling, Long Polling, HTTP Streaming and WebSockets](#), 这是StackOverflow上的一个HTTP各种链接方式的比较, 也可以让你有所认识。

◦ [An introduction to Websockets](#), 一个WebSocket的简单教程。

◦ [Awesome Websockets](#), GitHub的Awesome资源列表。

◦ 一些和WebSocket相关的想法, 可以开阔你的思路:

◦ [Introducing WebSockets: Bringing Sockets to the Web](#)

◦ [Websockets 101](#)

◦ [Real-Time Web by Paul Banks](#)

◦ [Are WebSockets the future?](#)

小结

总结一下今天的内容。我一直认为学习任何知识都要从基础出发, 所以今天我主要讲述了HTML5、CSS3和JavaScript (ES6) 这三大基础核心, 给出了大量的图书、文章以及其他一些相关的学习资源。之后, 我建议你学习浏览器的工作原理和网络协议相关的内容。我认为, 掌握这些原理也是学好前端知识的前提和基础。值得花时间, 好好学习消化。

下篇文章中, 我们将讲讲如何做前端性能优化, 并推荐一些好用的前端框架。敬请期待。

下面是《程序员练级攻略 (2018)》系列文章的目录 (持续更新中)。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络 \(系统底层知识\)](#)
 - [异步I/O模型和Lock-Free编程 \(系统底层知识\)](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门 \(分布式架构\)](#)
 - [分布式架构经典图书和论文 \(分布式架构\)](#)
 - [分布式架构工程设计\(分布式架构\)](#)
 - [微服务](#)
 - [分布式架构工程设计](#)
 - [容器化和自动化运维](#)
 - [机器学习和人工智能](#)
 - [前端基础和底层原理 \(前端方向\)](#)
 - [前端性能优化和框架](#)
 - [UI/UX设计](#)
 -

The promotional image features a portrait of Chen Hao, a middle-aged man with short grey hair, wearing a dark blue long-sleeved shirt. He is seated, looking slightly to his right with a gentle smile. To his left, large vertical Chinese characters '左耳听风' are written in a light blue gradient. To his right, the text '陈皓' is displayed in large white characters, with '资深技术专家' and '骨灰级程序员' in smaller white text below it. In the top right corner, there is a logo for '极客时间' (Geek Time) with the tagline '重拾极客精神·提升技术认知'. In the bottom right corner, there is a QR code with the text '扫码订阅' underneath.

前端性能优化

首先是推荐几本前端性能优化方面的图书。

- [Web Performance in Action](#)，这本书目前国内没有卖的。你可以看电子版本，我觉得是一本很不错的书，其中有CSS、图片、字体、JavaScript性能调优等。
 - [Designing for Performance](#)，这本在线的电子书很不错，其中讲了很多网页优化的技术和相关的工具，可以让你对整体网页性能优化有所了解。
 - [High Performance JavaScript](#)，这本书在国内可以买到，能让你了解如何提升各方面的性能，包括代码的加载、运行、DOM交互、页面生存周期等。雅虎的前端工程师尼古拉斯·扎卡斯（Nicholas C. Zakas）和其他五位JavaScript专家介绍了页面代码加载的最佳方法和编程技巧，来帮助你编写更为高效和快速的代码。你还会了解到构建和部署文件到生产环境的最佳实践，以及有助于定位线上问题的工具。
 - [High Performance Web Sites: Essential Knowledge for Front-End Engineers](#)，这本书国内也有卖，翻译版为《高性能网站建设指南：前端工程师技能精粹》。作者给出了14条具体的优化原则，每一条原则都配以范例佐证，并提供了在线支持。
- 全书内容丰富，主要包括减少HTTP请求、Edge Computing技术、Expires Header技术、gzip组件、CSS和JavaScript最佳实践、主页内联、Domain最小化、JavaScript优化、避免重定向的技巧、删除重复JavaScript的技巧、关闭ETags的技巧、Ajax缓存技术和最小化技术等。
- 除了上面这几本书之外，Google的[Web Fundamentals](#)里的[Performance](#)这一章节也有很多非常不错的知识和经验。

接下来是一些最佳实践性的文档。

- [Browser Diet](#)，前端权威性能指南（中文版）。这是一群为大型站点工作的专家们建立的一份前端性能的工作指南。
- [PageSpeed Insights Rules](#)，谷歌给的一份性能指南和最佳实践。
- [Best Practices for Speeding Up Your Web Site](#)，雅虎公司给的一份7个分类共35个最佳实践的文档。

接下来，重点推荐一个性能优化的案例学习网站[WPO Stats](#)。WPO是Web Performance Optimization的缩写，这个网站上有很多很不错的性能优化的案例分享，一定可以帮助你很多。

然后是一些文章和案例。

- [A Simple Performance Comparison of HTTPS, SPDY and HTTP/2](#)，这是一篇比较浏览器的HTTPS、SPDY和HTTP/2性能的文章，除了比较之外，还可以让你了解一些技术细节。
- [7 Tips for Faster HTTP/2 Performance](#)，对于HTTP/2来说，Nginx公司给出的7个增加其性能的小提示。
- [Reducing Slack's memory footprint](#)，Slack团队减少内存使用量的实践。
- [Pinterest: Driving user growth with performance improvements](#)，Pinterest关于性能调优的一些分享，其中包括了前后端的一些性能调优实践。其实也是一些比较通用的玩法，这篇文章主要是想让前端的同学了解一下如何做整体的性能调优。
- [10 JavaScript Performance Boosting Tips](#)，10个提高JavaScript运行效率的小提示，挺有用的。
- [17 Statistics to Sell Web Performance Optimization](#)，这个网页上收集了好些公司的Web性能优化的工程分享，都是非常有价值的。
- [Getting started with the Picture Element](#)，这篇文章讲述了Responsive布局所带来的一些负面的问题。主要是图像适配的问题，其中引出了一篇文章[Native Responsive Images](#)，值得一读。
- [Improve Page Load Times With DNS Prefetching](#)，这篇文章教了你一个如何降低DNS解析时间的小技术——DNS prefetching。
- [Jank Busting for Better Rendering Performance](#)，这是一篇Google I/O上的分享，关于前端动画渲染性能提升。
- [JavaScript Memory Profiling](#)，这是一篇谷歌官方教你如何使用Chrome的开发工具来分析JavaScript内存问题的文章。

接下来是一些性能工具。在线性能测试分析工具太多，这里只推荐比较权威的。

- [PageSpeed](#)，谷歌有一组PageSpeed工具来帮助你分析和优化网站的性能。Google出品的，质量相当有保证。
- [YSlow](#)，雅虎的一个网页分析工具。
- [GTmetrix](#)，是一个将PageSpeed和YSlow合并起来的一个网页分析工具，并且加上一些Page load或是其它的一些分析。也是一个很不错的分析工具。
- [Awesome WPO](#)，在GitHub上的这个Awesome中，你可以找到更多的性能优化工具和资源。

另外，中国的网络有各种问题（你懂的），所以，你不能使用Google共享的JavaScript链接来提速，你得用中国自己的。你可以到这里看看中国的共享库资源，[Forget Google and Use These Hosted JavaScript Libraries in China](#)。

前端框架

接下来，要学习的是Web前端的几大框架。目前而言，前端社区有三大框架Angular.js、React.js和Vue.js。我认为，React和Vue更为强劲一些，所以，我这里只写和React和Vue相关的攻略。关于两者的比较，网上有好多文章。我这里推荐几篇我觉得还不错的，供你参考。

- [Angular vs. React vs. Vue: A 2017 comparison](#)
- [React or Vue: Which JavaScript UI Library Should You Be Using?](#)
- [ReactJS vs Angular5 vs Vue.js-What to choose in 2018?](#)

其实，比较这些框架的优缺点还有利弊并不是要比出个输赢，而是让你了解一下不同框架的优缺点。我觉得，这些框架都是可以学习的。而在我们生活工作中具体要用哪个框架，最好还是要有一些出发点，比如，你是为了找份好的工作，为了快速地搭一个网站，为了改造一个大规模的前端系统，还是纯粹地为了学习……

不同的目的会导致不同的决定。我并不希望上述的这些比较会让你进入“二选一”或是“三选一”的境地。我只是想通过这些文章让你知道这些框架的设计思路和实现原理，这些才是让你受益一辈子的事。

React.js框架

下面先来学习一下React.js框架。

入门

React学起来并不复杂，就看[React 官方教程](#)和其文档就好了（[React 的中文教程](#)）。

然后，下面的文章会带你了解一下React.js的基本原理。

- [All the fundamental React.js concepts](#)，这篇文章讲了所有的React.js的基本原理。
- [Learn React Fundamentals and Advanced Patterns](#)，这篇文章中有几个短视频，每个视频不超过5分钟，是学习React的一个很不错的地方。
- [Thinking in React](#)，这篇文章将引导你完成使用React构建可搜索产品数据表的思考过程。

提高

学习一个技术最重要的是要学到其中的思想和方法。下面是一些我觉得学习React中最重要的东西。

- 状态，对于富客户端来说是非常麻烦也是坑最多的地方，这里有几篇文章你可以一读。

- [Common React.js mistakes: Unneeded state](#)，React.js编程的常见错误——不必要的状态。
- [State is an Anti-Pattern](#)，关于如何做一个不错的组件的思考，很有帮助。
- [Why Local Component State is a Trap](#)，一些关于“Single state tree”的想法。
- [Thinking Statefully](#)，几个很不错的例子让你对声明式的有状态的技术有更好的理解。
- 传统上，解决React的状态问题一般用Redux。在这里推荐 [Tips to learn React + Redux in 2018](#)。Redux是一个状态粘合组件，一般来说，我们会用Redux来做一些数据状态和其上层Component上的同步。这篇教程很不错。
- 最后是 “State Architecture Patterns in React” 系列文章，非常值得一读。

- [Part 1: A Review](#)
- [Part 2: The Top-Heavy Architecture, Flux and Performance](#)
- [Part 3: Articulation Points, zine and An Overall Strategy](#)
- [Part 4: Purity, Flux-duality and Dataflow](#)

- 函数式编程。从jQuery过来的同学一定非常不习惯React，而从Java等后端过来的程序员就会很习惯了。所以，我觉得React就是后端人员开发的，或者说是做函数式编程的人开发的。对此，你需要学习一下JavaScript函数式编程的东西。

这里推荐一本免费的电子书 [《Professor Frisby's Mostly Adequate Guide to Functional Programming》](#)，其中译版为《JS函数式编程指南中文版》。

下面有几篇文章非常不错。前两篇和函数式编程有关的文章非常值得一读。后三篇是一些比较实用的函数式编程和React结合的文章。

- [Master the JavaScript Interview: What is Functional Programming?](#)
- [The Rise and Fall and Rise of Functional Programming \(Composing Software\)](#)
- [Functional UI and Components as Higher Order Functions](#)
- [Functional JavaScript: Reverse-Engineering the Hype](#)
- [Some Thoughts on Function Components in React](#)

- 设计相关。接下来是学习一些React的设计模式。[React Pattern](#) 是一个不错的学习React模式的地方。除此之外，还有如下的一些不错的文章也会对你很有帮助的。

- [React Higher Order Components in depth](#)
- [Presentational and Container Components](#)
- [Controlled and uncontrolled form inputs in React don't have to be complicated](#)
- [Function as Child Components](#)
- [Writing Scalable React Apps with the Component Folder Pattern](#)
- [Reusable Web Application Strategies](#)
- [Characteristics of an Ideal React Architecture](#)

• 实践和经验

还有一些不错的实践和经验。

- [9 things every React.js beginner should know](#)
- [Best practices for building large React applications](#)
- [Clean Code vs. Dirty Code: React Best Practices](#)
- [How to become a more productive React Developer](#)
- [8 Key React Component Decisions](#)

资源列表

最后就是React的资源列表。

- [Awesome React](#)，这是一些React相关资源的列表，很大很全。
- [React/Redux Links](#)，这也是React相关的资源列表，与上面不一样的是，这个列表主要收集了大量的文章，其中讲述了很多React知识和技术，比上面的列表好很多。
- [React Rocks](#)，这个网站主要收集各种React的组件示例，可以让你大开眼界。

Vue.js框架

Vue可能是一个更符合前端工程师习惯的框架。不像React.js那样使用函数式编程方式，是后端程序员的思路。

- 通过文章 [“Why 43% of Front-End Developers want to learn Vue.js”](#)，你可以看出其编程方式和React是大相径庭的，符合传统的前端开发的思维方式。
- 通过文章 [Replacing jQuery With Vue.js: No Build Step Necessary](#)，我们可以看到，从jQuery是可以平滑过度到Vue的。
- 另外，我们可以通过 [“10 things I love about Vue”](#)，了解Vue的一些比较优秀的特性。

最令人高兴的是，Vue的作者是我的好朋友尤雨溪 (Evan You)，最近一次对他的采访 [“Vue on 2018-Interview with Evan You”](#) 当中有很多故事以及对Vue的展望。（注意：Vue是完全由其支持者和用户资助的，这意味着它更接近社区而不受大公司的控制。）

要学习Vue并不难，我认为上官网看文档 ([Vue 官方文档 \(中文版\)](#))，照着搞一搞就可以很快上手了。[Vue.js screencasts](#) 是一个很不错的英文视频教程。

另外，推荐 [新手向：Vue 2.0的建议学习顺序](#)，这是Vue作者写的，所以有特殊意义。

Vue的确比较简单，有Web开发经验的人上手也比较快，所以这里也不会像React那样给出很多的资料。下面是一些我觉得还不错的内容，推荐给你。

- [How not to Vue](#)，任何技术都有坑，了解Vue的短板，你就能扬长避短，就能用得更好。
- [Vue.js Component Communication Patterns](#)
- [4 AJAX Patterns For Vue.js Apps](#)
- [How To \(Safely\) Use A jQuery Plugin With Vue.js](#)
- [7 Ways To Define A Component Template in Vue.js](#)
- [Use Any Javascript Library With Vue.js](#)
- [Dynamic and async components made easy with Vue.js](#)

当然，最后一定还有 [Awesome Vue](#)，Vue.js里最为巨大最为优秀的资源列表。

小结

总结一下今天的内容。我先介绍的是前端性能优化方面的内容，推荐了图书、最佳实践性的文档、案例，以及一些在线性能测试分析工具。随后重点讲述了React和Vue两大前端框架，给出了大量的文章、教程和相关资源列表。我认为，React.js使用函数式编程方式，更加符合后端程序员的思路，而Vue是更符合前端工程师习惯的框架。因此，两者比较起来，Vue会更容易上手一些。

下篇文章，我们将讲述前端工程师的一个基本功——UI/UX设计。敬请期待。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 - [分布式架构工程设计（分布式架构）](#)
 - [微服务](#)
 - [分布式架构工程设计](#)
 - [容器化和自动化运维](#)
 - [机器学习和人工智能](#)
 - [前端基础和底层原理（前端方向）](#)
 - [前端性能优化和框架（前端方向）](#)
 - [UI/UX设计（前端方向）](#)
 -[.....](#)

左耳听风，分享有赏

8月1日 - 8日，老用户邀请好友订阅
可享受 **¥50** 现金返现

多邀多的，上不封顶，立即提现



上面的技术都讲完了，前端还有一个很重要的事就是设计。作为前端人员，我们有必要了解现在的一些知名且流行的设计语言或是一些设计规范或是设计方法，学习它们的设计思想和方法，有助于我们拓宽眼界、与时俱进。我并不觉得这些内容是设计师要学习的，如果你要成为一个前端程序员，那么学习这些设计上的东西可以让你有更好的成长空间。

对于学习设计的新手来说，推荐看看 [7 steps to become a UI/UX designer](#)，这是一篇很不错的让新手入门的文章，非常具有指导性。首先，你得开始学习设计的一些原则和套路，如配色、平衡、排版、一致性等。还有用户体验的4D步骤——Discover、Define、Develop 和 Delivery。然后，开始到一些网站上找灵感。接下来，是到不同的网站上读各种文章和资源，开始学习使用设计工具，最后是找人拜师。此外，其中还链接了其它一些不错的文章、网站、博客和工具。我认为，这篇文章是一篇很不错的设计师从入门到精通的练级攻略。

虽然有这么一个速成的教程，但我觉得还是应该系统地学习一下，所以有了下面这些推荐。

图书和文章推荐

先推荐几本书。

- [Don't Make Me Think](#)，这是我看的第一本和设计相关的书。这本书对我的影响也比较深远。这本书践行了自己的理论，整本书短小精悍，语言轻松诙谐，书中穿插大量色彩丰富的屏幕截图、趣味丛生的卡通插图以及包含大量信息的图表，使枯燥的设计原理变得平易近人。
- [Simple and Usable Web, Mobile, and Interaction Design](#)，中文版译名为《简约至上》。本书作者贾尔斯（Giles）有20多年交互式设计的探索与实践。提出了合理删除、分层组织、适时隐藏和巧妙转移这四个达成简约至上的终极策略，讲述了为什么应该站在主流用户一边，以及如何从他们的真实需求和期望出发，简化设计，提升易用性。
- [Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules](#)，中文版译名为《认知与设计：理解UI设计准则》。这本书语言清晰明了，将设计准则与其核心的认知学和感知科学高度统一起来，使得设计准则更容易在具体环境中得到应用。涵盖了交互计算机系统设计的方方面面，为交互系统设计提供了支持工程方法。不仅如此，这也是一本人类行为原理的入门书。
- [Designing Interfaces: Patterns for Effective Interaction Design](#)，中文版译名为《界面设计模式》。这本书开篇即总结了“与人有关”的各类问题，为读者提供了界面设计总体思路上的指引，帮助读者举一反三。然后，收集并分析了很多常用的界面设计模式，帮助读者理解在实现级别的各种常用解决方案，将它们灵活地运用到自己的设计中。

除了上面的这几本书，还有下面的这几篇文章也是很不错的，推荐一读。

- [The Psychology Principles Every UI/UX Designer Needs to Know](#)，这篇文章讲述了6大用户界面用户体验设计的心理学原则。
- [18 designers predict UI/UX trends for 2018](#)，我倒不觉得这篇文章中所说的UI/UX是在2018年的趋势，我反而觉得，这18条原则是指导性的思想。
- [The Evolution of UI/UX Designers Into Product Designers](#)，这篇文章是Adobe公司的一篇博客，其在回顾整个产品设计的演化过程中有一些不错的思考和想法，并提供了一些方法论。

原子设计（Atomic Design）

在2013年网页设计师布拉德·弗罗斯特（Brad Frost）从化学中受到启发：原子（Atoms）结合在一起，形成分子（Molecules），进一步结合形成的生物体（Organisms）。布拉德将这个概念应用在界面设计中，我们的界面就是由一些基本的元素组成的。

乔希·杜克（Josh Duck）的“HTML元素周期表”完美阐述了我们所有的网站、App、企业内部网、[hoobadyboops](#)等是如何由相同的HTML元素组成的。通过在大层面（页）和小层面（原子）同时思考界面，布拉德认为，可以利用原子设计建立一个适应组件的动态系统。

为什么要玩原子设计，我认为，这对程序员来说是非常好理解的，因为这就是代码模块化重用化的体现。于是，你就是要像搭积木一样开发和设计网页，当你把其模块化组件化了，也更容易规范整体的风格，而且容易维护……这些都意味着你可以更容易地维护你的代码。所以，这个方法论导致了Web组件化的玩法。这是设计中非常重要的方法论。

关于这个设计方法论，你可以阅读一下下面这几篇文章。

- [Atomic Design原子设计|构建科学规范的设计系统](#)
- [网页设计：Atomic Design简介及工作实例](#)

但是，真正权威的地方还是布拉德·弗罗斯特的电子书、博客和实验室，可以从中获取更多的信息。

- [电子书：Atomic Design by Brad Frost](#) 是布拉德·弗罗斯特写的一本书。
- [博客：Atomic Design](#) 是布拉德·弗罗斯特的博客。
- [实验室：Pattern lab](#) 是布拉德·弗罗斯特依照这个设计系统所建立的一套工具，可以前往Pattern Lab的 [GitHub](#) 来试试Atomic design。

接下来是关于这个设计方法和React.js框架的几篇文章。

- [Atomic Design with React](#)
- [Atomic Components: Managing Dynamic React Components using Atomic Design](#)

设计语言和设计系统

下面来介绍一下设计语言和设计系统。

Fluent Design System

Fluent Design System 中文翻译为流畅设计体系，是微软于2017年开发的设计语言。流畅设计是Microsoft Design Language 2的改版，其中包含为所有面向Windows 10设备和平台设计的软件中的设计和交互的指导原则。

该体系基于五个关键元素：光感、深度、动效、材质和缩放。新的设计语言包括更多对动效、深度及半透明效果的使用。过渡到流畅设计体系是一个长期项目，没有具体的完成目标，但是从创作者更新以来，新设计语言的元素已被融入到个别应用程序中。它将在未来的Windows 10秋季创作者更新中更广泛地使用，但微软也表示，该设计体系不会在秋季创作者更新内完成。

微软于2017年5月11日的Microsoft Build 2017开发者大会上公开了该设计体系。

- [What's new and coming for Windows UI: XAML and composition](#)，从概念上讲了一下Fluent Design System的各个部分。
- [Introducing Fluent Design](#)，介绍了Fluent Design System的各个部分。

还有Build 2018上的一些微软的YouTube分享。

- [Fluent Design: Evolving our Design System : Build 2018](#)
- [Microsoft Build 2018 - Fluent Design System Demo](#)
- [Microsoft Build 2018 - Fluent Design System Evolution](#)
- [Fluent Design System inside of Microsoft: Office : Build 2018](#)

Material Design

Material Design 中文翻译为质感设计，或是材质设计、材料设计。这是由Google开发的设计语言。扩展于 [Google Now](#) 的“卡片”设计，Material Design基于网格的布局、响应动画与过渡、填充、深度效果（如光线和阴影）。设计师马蒂亚斯·杜阿尔特（Matías Duarte）解释说：“与真正的纸张不同，我们的数字材质可以智能地扩大和变形。材质具有实体的表面和边缘。接缝和阴影表明组件的含义。”Google指出他们的新设计语言基于纸张和油墨。

Material Design于2014年的Google I/O大会上发布（参看 [Google I/O 2014 - Material witness: How Android material applications work](#)）。其可借助v7 appcompat库用于Android 2.1及以上版本，几乎支持所有2009年以后制造的Android设备。随后，Material Design扩展到Google的网络和移动产品阵列，提供一致的跨平台和应用程序体验。Google还为第三方开发人员发布了API，开发人员可将质感设计应用到他们的应用程序中。

除了到 [官网](#) 学习 Material Design，你还可以访问 [Material Design 中文版](#) 来学习。

另外，Wikipedia 上有一张 [Material Design 实现的比较表](#)，供你参考。

下面是几个可供你使用的Material UI的工程实现。

- [Material Design Lite](#)，这是Google官方的框架，简单易用。
- [Materialize](#)，一组类似于Bootstrap的前端UI框架。
- [Material-UI](#)是基于Google Material Design的React组件实现。
- [MUI](#)是一个轻量级的CSS框架，遵循Google的Material Design设计方针。

其它公司

接下来再来推荐其它几家公司的设计语言。

- [苹果公司的设计指南](#)，在这个网站有苹果的各种设备的设计规范和指导，一方面可以让你的App能和苹果的UI融合在一起，另一方面，你也可以从中看到苹果的审美和思维方式。
- [IBM公司的设计语言](#)，我们总觉得IBM公司是一家比较传统的没有新意的公司，但是并不是这样的。IBM公司的这个设计语言的确比较出众。所以，在这里推荐一下。
- [Salesforce公司的Lightning Design System](#)，是在Salesforce生态系统中用于创建统一UI的设计模式、组件和指南的集合，是一个企业级的产品。
- [Facebook Design - What's on our mind?](#)，Facebook的设计师们收集的一系列的文章、视频和资源。很不错哦。

动画效果设计

我认为，要了解Web动画效果设计的第一步，最好的地方是[CodePen](#)。这个网站不只是让人分享HTML、CSS和JavaScript代码的网站。其中也有很多分享样例都和动画效果有关。这个网站可以让你对动画效果有一些感性认识，当然还有代码供你参考。

接下来，我们要了解动画效果设计的一些方法。基本上来说，动画设计都会受“[动画的12项基本法则](#)”的影响，这个方法论源自于迪士尼动画师奥利·约翰斯顿（Ollie Johnston）和弗兰克·托马斯（Frank Thomas）在1981年所出的《The Illusion of Life: Disney Animation》一书。这些法则已被普遍采用，至今仍与制作3D动画法则有关联。这里还有一篇文章“[Understand the 12 principles of animation](#)”是对这个法则的解读和理解。

除此之外，还有几个动画设计指南和相关文章供你参考和学习。

- [6 Animation Guidelines for UX Design](#)。这是Prototypy公司的一个指南，其中主要指出，动画效果不是为了炫酷，而是能让你的UI/UX能活起来，自然，不消耗时间，并且是生动故事型的动画效果。其中还推荐了如下几篇很不错的文章。
 - [Transitional Interfaces](#)
 - [UI Animation and UX: A Not-So-Secret Friendship](#)
 - [Invisible animation](#)
 - [Creating Usability with Motion: The UX in Motion Manifesto](#)
- [Designing Interface Animation](#)，这篇文章同样说明，任何一个动画都是要讲一个微故事的，而且这些微故事会和你的品牌和产品理念相融合。动画会给人更深的印象，让人们更容易记住你。这篇文章主要是讲品牌动画。
- [Animation principles in motion design](#)，这篇文章有点像设计模式，给了一些动画效果的套路和演示。
- [Creating Usability with Motion: The UX in Motion Manifesto](#)
- [Integrating Animation into a Design System](#)
- Great UI/UX Animations是设计师丹尼尔（Daniel）收集的一些很不错的动画，可以给你一些灵感。
 - [Great UI/UX Animations 第一组](#)
 - [Great UI/UX Animations 第二组](#)

相关资源

下面分享一下UI/UX设计的相关资源。文章资源主要有以下这些。

文章来源

- [Web Designer News](#)，一个文章聚合的网站。除此之外，还有两个文章聚合网站，你也可以订阅。一个是[Designer News](#)，另一个是[Reddit Web Design](#)。
- [Marvel Blog](#)，Marvel团队的博客。
- [The Next Web](#)，内容主要涵盖国际技术新闻、商业和文化等多个方面。
- [Medium - Design](#)，Medium现在已经成为一个好文章的集散地了，这个地方必去。
- [Smashing Magazine](#)，这个地方是给专业的Web设计师和程序员的。不但有设计还有HTML、CSS和JavaScript等各种资源。
- [Sitepoint](#)，这个网站上也有很多不错的给Web前端程序员和设计师看的文章（当然，给程序员看的有点简单了，我觉得更像是让设计师来学写程序的网站）。

设计收集

接下来推荐一些优秀设计的聚集地。

- [Awwwards](#)，这个网站给一些设计得不错的网站的评分，在这里你可以看到很多设计不错的网站。
- [One Page Love](#)，就是一个单页的网页设计的收集。
- [Inspired UI](#)，移动App的设计模式。
- [Behance](#)，这个平台有很多很不错的很有创意的作品。
- [Dribbble](#)，这应该是设计师都知道也都爱去的网站。除了你可以看到一些很不错的作品外，你还可以在这里看到很多不错的设计师。
- [UI Movement](#)，也是个设计的收集网站，上面有很多很不错的UI设计，大量的动画。虽说会像抖音一样，让你不知不觉就看了好几个小时，但是它比抖音让你的收获大多了。

小结

总结一下今天的内容。我并不认为UI/UX设计这些内容只是设计师要学习的，如果你要成为一个前端程序员，那么学习这些设计上的东西可以让你有更好的成长空间。首先，我推荐了一些图书和文章，让你更好地了解经典的设计原则和指导思想。

然后介绍了原子设计，以及深入学习和理解这一设计方法论的图书、文章和其他相关资源。最后分享了当下主流和知名公司中在用的设计语言和设计系统，并给出了大量的学习资源，推荐了一些优秀设计的聚集地。相信通过学习这些内容，你在UI/UX设计方面不仅能收获方法，还能获得非常多的灵感。

下篇文章是程序员练级攻略2018版高手成长篇的最后一篇，我将推荐大量有价值的技术资源，这些内容将会为你后续的学习和成长提供很大的助力。敬请期待。

下面是《程序员练级攻略（2018）》系列文章的目录（持续更新中）。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)

- 专业基础篇
 - 编程语言
 - 理论学科
 - 系统知识
- 软件设计篇
 - 软件设计
- 高手成长篇
 - [Linux系统、内存和网络 \(系统底层知识\)](#)
 - [异步I/O模型和Lock-Free编程 \(系统底层知识\)](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门 \(分布式架构\)](#)
 - [分布式架构经典图书和论文 \(分布式架构\)](#)
 - [分布式架构工程设计\(分布式架构\)](#)
 - [微服务](#)
 - [分布式架构工程设计](#)
 - [容器化和自动化运维](#)
 - [机器学习和人工智能](#)
 - [前端基础和底层原理 \(前端方向\)](#)
 - [前端性能优化和框架 \(前端方向\)](#)
 - [UI/UX设计 \(前端方向\)](#)
 - 技术资源集散地

左耳听风，分享有赏

8月1日 - 8日，老用户邀请好友订阅

可享受 **¥50** 现金返现

多邀多的，上不封顶，立即提现



个人技术博客

首先，我先推荐一些不错的个人技术博客。

- [Coding Horror](#)，这是杰夫·阿特伍德 (Jeff Atwood) 于2004年创办的博客，记录其在软件开发经历中的所思所想、点点滴滴。时至今日，该博客每天都有近10万人次的访问量，读者纷纷参与评论，各种观点与智慧在这里不断地激情碰撞。其博文选集在中国被翻译成《[高效能程序员的修炼](#)》，在豆瓣上有8.3的高分。2008年，他和Joel Spolsky联合创办了[StackOverflow](#) 问答网站，为程序员在开发软件时节省了非常多的时间，并开启了"StackOverflow Copy + Paste 式编程"。
- [Joel on Software](#)，Joel Spolsky的这个博客在全世界都有很多的读者和粉丝，其博文选集在中国被翻译成《[软件随想录](#)》在豆瓣上有8.7的高分。这是一本关于软件技术、人才、创业和企业管理的随想文集，作者以诙谐幽默的笔触将自己在软件行业的亲身感悟娓娓道来，观点新颖独特，简洁实用。
- [Clean Coder Blog](#)，这是编程大师是"Bob 大叔"的博客，其真名叫Robert C. Martin，世界级软件开发大师，设计模式和敏捷开发先驱，敏捷联盟首任主席，C++ Report前主编，被后辈程序员尊称为"Bob大叔"。其博文选集在中国被翻译成《[程序员的职业素养](#)》，在豆瓣上有8.8的高分。
- [Martin Fowler](#)，这是另外一个程序员大师，Martin主要专注于面向对象分析与设计、统一建模语言、领域建模，以及敏捷软件开发方法，包括极限编程。他的《[重构](#)》、《[分析模式](#)》、《[企业应用架构模式](#)》、《[领域特定语言](#)》和《[NoSQL精粹](#)》都是非常不错的书。在他的博客上有很多很多的编程和架构模式方法可以学习。
- [Paul Graham Essays](#)，美国著名程序员、风险投资家、博客和技术作家。《[黑客与画家](#)》是他的著作之一。2005年他与人共同创建了著名的创业投资公司Y Combinator，是初创公司最想被投的。他有几篇创业方面的文章都很经典，如果你想创业，可以读一读这几篇：《[How to Get Startup Ideas](#)》、《[Do Things that Don't Scale](#)》、《[Startup = Growth](#)》。Paul Graham的文章以清新自然，思想深刻见长。不仅可以跟Paul Graham学创业，学思考，学技术，更可以学习写作。
- [Steve Yegge](#)，Steve Yegge这个人算是一个知名的程序员了，在Amazon呆过，现在在Google，他的文章都是长篇大论，最知名的文章就是对Amazon和Google平台的吐槽，这篇文章引发了大家的讨论和议论。
- [Bruce Eckel's Programming Blog](#)，《Thinking in Java》作者的博客，他之前的博客在artima - [Computing Thoughts](#)。
- [Herb Sutter](#)，C++大拿，C++标准委员会专家，微软软件架构师。《Exceptional C++》、《More Exceptional C++》、《Exceptional C++ Style》作者。
- [Eli Bendersky's website](#)，这位老哥从2003年就一直写博客到今天，其中的文章都非常不错，原理型的，主要是C、C++和Python相关的。里面有很多干货。
- [Peter Krumins' blog](#)，这位老哥从2007年开始写博客，他博客里好玩的东西太多了。
- [Brendan D. Gregg](#)，Brendan是Netflix的工程师，他的博客里有大量的非常非常不错的文章，基本上都是和Linux性能分析相关的，这是一个如果你要玩底层性能分析一定不能错过的博客。
- [Evan Klitzke](#)，主要讨论Linux和C++相关的内容。
- [Julia Evans](#)，主要讨论Linux debug工具和网络相关的内容。
- [null program](#)，和C/C++相关的一个博客。其中关于Linux系统调用、GPU、无锁编程、JIT编译的一些文章非常不错。
- [Fluent {C++}](#)，博主是Murex的首席工程师，主要玩C++，在这个博客里有很多很不错的C++相关的文章。
- [Preshing on Programming](#)，这也是一个和C/C++相关的博客，其中有很多的干货。
- [Programming is Terrible](#)，这个博客有很多强观点的文章，主要是软件开发中的一些教训。
- [Accidentally Quadratic](#)，姑且翻译成事故二次方，这里有好些非常有趣的文章。
- [Hacker Noon](#)，这是一个一堆人在写的博客，里面有很多质量很高的文章。

其实还有很多不错的博客，不过，现在国外不错的博客都在一个叫 [Medium](#) 的网站，我也发现我Google很多东西时都会到这个网站上。这个网站上的内容不只有技术的，还有很多其他方面的内容，比如文化、艺术、科学等等。这个网站就是一个博客发布系统，其是由Twitter联合创始人埃文·克拉克·威廉姆斯 (Evan Clark Williams) 和克里斯多福·艾萨克·比兹·斯通 (Christopher Isaac Biz Stone) 创办的，这两个人觉得Twitter上全是垃圾没有营养的信息。所以，创办了Medium，这个平台上有关专业和非专业的贡献者，亦有受雇的编者。

我已经感觉到，未来高质量的文章都会在Medium这个平台上出现，因为有一些公司的技术博客也在这个平台上发布了，比如Netflix的。所以，你有必要上到这个平台上 follow 一些作者、专栏和主题。

YouTube 技术频道

下面是我订阅的一些我认为还不错的和编程相关的频道，推荐给你。

- [Devoxx](#)，Devoxx的频道，其中有各种很不错的技术分享。
- [Coding Tech](#)，也是个非常不错的编程频道，涵盖各种技术。
- [Amazon Web Services](#)
- [Facebook Developers](#)
- [Google Developer](#)，Google公司的官方频道，其中包括Google I/O 大会、教程、新闻、最佳实践、技巧分享.....
- [Spring Developer](#)，Spring的官方频道。
- [Microsoft Research](#)
- [MIT 公开课](#)
- [Stanford Online](#)
- [Prof. Dr. Jens Dittrich](#)，一个德国教授开的一个关于数据库相关的频道，里面有很不错的数据库内在原理的内容。
- [Red Hat Summit](#)，RedHat峰会频道，其中有很多和Linux相关的技术新闻和分享。
- [Open Networking Summit](#)，这是一个网络相关的频道。
- [Dan Van Boxel](#)，这是一个机器学习工程师折腾各种事的视频，挺有意思的。
- [The New Boston](#)，这个频道应该是前端开发工程师必去的地方，可能也是我所知道的最好的关于前端技术的YouTube频道。
- [Derek Banas](#)是一个教程型的频道，其中包括编程语言、游戏开发、Web开发.....我个人觉得是一个可以用来练英文听力的频道。
- [Java](#)，Java相关的各种分享。
- [CppCon](#)，C++大会的一些视频，可以让你了解很多C++最新功能和相关的动态。
- [Computerphile](#)，这个频道是布雷迪·哈伦 (Brady Haran) 运作的几个频道中的一个，在这个频道里你可以看到很多很有趣的技术方面的科普教程、资讯、见闻等，说得都非常地简单易懂，所以有大量的订阅用户。布雷迪是个对任何技术都很有热情的人，这个频道是关于计算机技术的。除此之外，他还运作 [Numberphile](#) (数学)、[Periodic Videos](#) (化学)、[Sixty Symbols](#) (物理)、[Deep Sky Videos](#) (天文) 等有众多阅人数的频道。如果你喜欢，你都可以——订阅，感觉就是一个个人版的Discovery。
- 关于安全，有如下四个频道你可以订阅一下：

- [DEFCONConference](#), defcon.org的官方频道。
- [CCCen](#), Chaos Computer Club。
- [RSA Conference](#), RSA Conference。
- [Black Hat](#) - Black Hat Conference。

各大公司技术博客

细心的你一定会发现这份攻略中的很多推荐文章都来自于各个公司的技术团队的博客。是的，跟随这些公司的博客，你不但可以看到这些公司的工程技术，还能掌握到一些技术方向和趋势。

下面是Airbnb、AWS、Cloudera、Dropbox、Facebook、Google等各个公司的技术博客列表。

- [Airbnb Engineering](#)
- AWS 相关
 - [All Things Distributed](#)
 - [AWS Architecture Blog](#)
 - [On Efficiency, Reliability, Scaling - James Hamilton, VP at AWS](#)
- [Bandcamp Tech](#)
- [BankSimple Simple Blog](#)
- [Bitly Engineering Blog](#)
- [Cloudera Developer Blog](#)
- [Dropbox Tech Blog](#)
- [Etsy Code as Craft](#)
- [Facebook Engineering](#)
- [Flickr Code](#)
- [Foursquare Engineering Blog](#)
- [Google Research Blog](#)
- [Groupn Engineering Blog](#)
- [High Scalability](#)
- [Instagram Engineering](#)
- [LinkedIn Engineering](#)
- [Oyster Tech Blog](#)
- [Pinterest Engineering Blog](#)
- [Quora Engineering](#)
- [Songkick Technology Blog](#)
- [SoundCloud Backstage Blog](#)
- [Square The Corner](#)
- [The Reddit Blog](#)
- [The GitHub Blog](#)
- [The Netflix Tech Blog](#)
- [Twilio Engineering Blog](#)
- [Twitter Engineering](#)
- [WebEngage Engineering Blog](#)
- [Yammer Engineering](#)
- [Yelp Engineering Blog](#)
- [Smarkets Blog](#)

论文

要想将技术研究得精深，论文是必不可少的。那要如何读论文呢？

如何读论文

下面有几篇文章，教你一些读论文的方法，非常不错。

- [How to read an academic article](#)
- [Advice on reading academic papers](#)
- [How to read and understand a scientific paper](#)
- [Should I Read Papers?](#)
- [The Refreshingly Rewarding Realm of Research Papers](#)

论文集散地

要成长为一个高手，论文是你一定要读的。下面是一些非常不错的计算机方面的论文集散地。

- [2 Minute Papers](#), 这是一个YouTube的频道，其会给出一些非常不错的和计算机相关的论文介绍，让你了解目前最有意思的一些科学突破，每次两分钟左右。
- [Best Paper Awards in Computer Science](#), 从1996年以来，获奖的计算机科学方面的论文收集。
- [Google Scholar](#), Google学术搜索（英语：Google Scholar）是一个可以免费搜索学术文章的网络搜索引擎，由计算机专家阿努拉格·阿查里雅（Anurag Acharya）开发。2004年11月，Google第一次发布了Google学术搜索的试用版。该项索引包括了世界上绝大部分出版的学术期刊。
- [Facebook](#), Facebook公司的论文。
- [Research at Google](#), Google发布一些论文。
- [Microsoft Research](#), 微软发布的论文。
- [MIT's Artificial Intelligence Lab Publications](#), MIT和人工智能相关的论文。
- [MIT's Distributed System's Reading Group](#), MIT和分布式系统相关的论文。
- [arXiv Paper Repository](#), arXiv是一个收集物理学、数学、计算机科学与生物学的论文预印本的网站，始于1991年8月14日。截至2008年10月，arXiv.org已收集超过50万篇预印本。至2014年底，藏量达到1百万篇。

在2014年时，约以每月8000篇的速度增加。arXiv的存在是造就科学出版业中所谓开放获取运动的因素之一。[现今的一些数学家及科学家习惯先将其论文上传至arXiv.org](#)，再提交予专业的学术期刊。这个趋势对传统学术期刊的经营模式造成了可观的冲击。

- [SciRate](#), arXiv上的论文太多，所以，SciRate索引了arXiv上的一些好评的论文，并供大家评论和打分。（[开源代码](#)。）
- [cat-v.org](#), 这个网站，不只有论文，还有技术手册或是一些有意思的文章，包括一些历史资料什么的。
- [Usenix: Best Papers](#), Usenix上推荐的最佳论文。
- [The Morning Paper](#), 该博客会每天推送一篇论文，特别棒。
- [Lobste.rs tagged as PDF](#), Lobsters是一个聚焦于技术的社区，主要是链接聚合和对话题进行讨论。其中的PDF分类可以认为也是一个论文的集散地。
- [Papers We Love](#), GitHub上的一个近3万颗星的计算机科学方面的论文社区。

小结

以上就是关于技术学习的一些攻略，希望对你有所帮助。当然，这只是冰山一角，还有很多其他的内容没有提到。

总结一下今天的内容。这篇文章我主要跟你分享了一些好的学习资源，帮你开拓眼界，为后续学习夯实基础。

首先，我推荐了Coding Horror、Joel on Software、Clean Coder Blog、Martin Fowler、Paul Graham Essays等多个知名的个人技术博客。然后分享了一些我订阅的我认为还不错的和编程相关的YouTube频道，比如Coding Tech、Amazon Web Services、Facebook Developers、Google Developer等。

随后是Airbnb、AWS、Cloudera、Dropbox、Facebook、Google等各个公司的技术博客，跟随这些公司的博客，你不但可以看到这些公司的工程技术，还能掌握到一些技术方向和趋势。最后，想成长为一个高手，论文是一定要读的。所以，我给出了一个非常不错的计算机方面的论文集散地，并推荐了一些学习资源来教你如何读这些论文。

我一直认为，学习需要自我驱动，要学会自己“找食物”，而不是“等着喂”。程序员练级攻略2018版到今天就全部更新完成了，但我认为，这其实只是技术练级的起点，还有很多知识和技术，需要我们不断地去探索和发现。加油，我能做到的，你一定也可以做到。

下面是《程序员练级攻略（2018）》系列文章的目录。

- [开篇词](#)
- 入门篇
 - [零基础启蒙](#)
 - [正式入门](#)
- 修养篇
 - [程序员修养](#)
- 专业基础篇
 - [编程语言](#)
 - [理论学科](#)
 - [系统知识](#)
- 软件设计篇
 - [软件设计](#)
- 高手成长篇
 - [Linux系统、内存和网络（系统底层知识）](#)
 - [异步I/O模型和Lock-Free编程（系统底层知识）](#)
 - [Java底层知识](#)
 - [数据库](#)
 - [分布式架构入门（分布式架构）](#)
 - [分布式架构经典图书和论文（分布式架构）](#)
 - [分布式架构工程设计（分布式架构）](#)
 - [微服务](#)
 - [容器化和自动化运维](#)
 - [机器学习和人工智能](#)
 - [前端基础和底层原理（前端方向）](#)
 - [前端性能优化和框架（前端方向）](#)
 - [UI/UX设计（前端方向）](#)
 - [技术资源集散地](#)

