

2012/07/05

码农

codeMaker ()

第 1 期

- 本期专题：算法
- 对话归隐的大师：高德纳
- 海量用户积分排名算法探讨
- Matrix67的Aha!Moment
- 终身Coder，可以吗？
- 工程的本质问题是组织

卷首语

- 1 所谓码农

本期专题：算法

- 3 对话归隐的大师——Donald E. Knuth（高德纳）
- 11 关于TAOCP中用集合论对算法进行严格数学定义的理解
- 24 海量用户积分排名算法探讨
- 33 图说归并排序

图灵访谈

- 41 终身Coder，可以吗？
- 47 Matrix67的Aha! Moment
- 59 柳泽大辅谈如何想出好创意

iTran乐译

- 63 移动为什么重要？
- 70 Android 应用该是个啥样子？
- 87 为什么Pinterest是最让人失望的社交网络？
- 91 JavaScript并行运算新机遇——Web Workers的神奇魔法
- 118 别把自己当个超人——给初级程序员的一点小小建议

出版的未来

- 127 创意帮创意——一个关于图书、出版和筹款的故事

当红书榜

153 看看大家都在看什么？

好书妙评

154 《七周七语言》

155 《算法（英文版 第4版）》

157 《罗素的故事》

158 《黑客与画家》

159 《重构：改善既有代码的设计》

TEAP早期试读

160 为什么选择MongoDB？

168 工程的本质问题是组织

编辑手记

177 檐头滴水话求职

社区动态

182 乐译7期

183 专家审读第3期

184 7月听课去

所谓码农



@武卫东

湖南怀化人氏。1991年清华大学毕业后，一直从事计算机图书出版行业。先后在电子工业出版社、西蒙与舒斯特公司、机械工业出版社、华章公司、人民邮电出版社、图灵公司工作。现任图灵公司总经理兼总编辑。

图灵社区ID: [豆他爹](#)，有子名豆，故有此号。

对于码农这个称呼，有些人喜爱有加，有些人不以为然。区别在于对待“农”这个字的感觉。农当然是指农民，这个词寓意很丰富，既可以说它伟大，也可以说它渺小。说它伟大，是因为我们的生存离不开农民，而且中国一直是个农业为本的国家，曾经说是百分之八十的人是农民，这意味着往前翻一两代，你我众人皆是农民出身。以此观之，农可谓大哉。说它渺小，则是因为大家的观念里，农民意味着有很多缺点，冠冕堂皇的说法是劣根性，比如目光短浅、思维陈旧、自私小气，等等，总之是为我们受过教育的人群所看不惯的种种毛病——这些毛病虽然我们自己也有，但是我们看不见——于是乎“农民”成为了骂人时常用的字眼。

其实，把自己的编程生涯与田间地头的锄禾日当午对应起来，确是有那么些相似之处的。你能想象得到，田间整齐栽种的秧苗，与屏幕上显示的错落有致的代码行有几分神似。各种庄稼的种植是有讲究的，正如你要注意编程风格。施肥灌溉，犹如你对代码进行的编译链接。除草除虫，自然是在做着debug。你挑水来我浇园，大概是在小菜园中进行的结对编程。因为靠天吃饭，农民们也要学点云计算，去五道口职业技术学校进修的人也多起来了。收割的季节，活多人少，也常常是要搞外包的，因为deadline很重要，time to market不容错过。

不过坦率地说，码农这个叫法让人体会更多的是滑稽、搞怪、无厘头。毕竟一个是简单的体力劳动，一个是高智商的脑力劳动，不可同日而语。而程序员却偏爱这样的时空错乱的感觉，自嘲（我就是个农民！）的同时却又自命不凡（我是码农我怕谁？！），特立独行极了。

码农的草帽底下，是一颗充满创造力的自由不羁的头脑。他们遵从最佳实践而痛恨陈规教条，他们欣赏天才而不迷信权威，他们喜欢思考而不轻易苟同。他们是技术人，却追求人文理想；他们敢于呐喊，说出自己的观点和主张，也更善于脚踏实地，用自己的点滴工作去改变现状。码农们是勤奋的，加班加点的工作是常有的事情，城市夜间的灯火，有多少是在码农们的办公室和居所点燃？周末四处举办的技术交流和讲座，又活跃着多少码农的身影？线下读书，线上讨论，冥思苦想，动手实践，新技术驱动着码农们的脚步，码农们在改变着我们的生活。

生存离不开农民，生活离不开码农。 ■

对话归隐的大师

——Donald E. Knuth（高德纳）



图灵社区：很多中国程序员都将您视为计算机科学领域的神祇。而这样的情况，又恰恰出现在盛行无神论的中国，很有意思。我们已经清楚您写作[《计算机程序设计艺术》](#)（以下简称 *TAOCP*）的初衷和经历，也知道您关于“信仰与科学的关系”的系列讲座曾大受欢迎，因此对您的写作和信仰之间的关系很感兴趣。能否谈一下，信仰和上帝在 *TAOCP* 的创作过程中，给您带来的是哪些帮助呢？

计算机科学泰斗Donald E. Knuth（高德纳）归隐已近20载，不问世事，潜心修订并继续创作煌煌巨著《计算机程序设计艺术》（*The Art of Computer Programming*）多卷本。

图灵社区藉卷4A影印版出版之机，有幸邀得大师接受关于此书的访谈，并在多位社区成员[留言提问](#)的帮助下，完成了这次珍贵的访谈。

高德纳：计算机科学是既壮观又幽美的，我尝试尽自己所能，以最恰当的方式来解释我所了解的某些片断。很显然，我自己并没有任何超自然能力，但的确很喜欢讲述那些似乎静静地等待着人们去讲出来的故事。写书跟讲故事十分类似。

此外，虽然计算机科学非常美妙，但它也不可能包办一切！我相信，总有一些神秘的东西是超越人类的理解而存在的。

信仰是很私人的东西，它涵盖了一些永远无法证明的概念。因此，本人在信仰问题上的看法，我并不指望每个人都能同意。我认为，上帝希望我能创造某些成果，而这些

东西能够启发其他人去创造其他成果。这就是我的宗教生活和科学生活之间的主要关系。

图灵社区：对您耗费几十年的时间创作的 *TAOCP*，我们代表所有已经和即将从中获益的中国读者，向您致以诚挚的感谢。到现在为止，这部著作已经创作了半个世纪。这样的成书过程，让我们想起歌德的《浮士德》。令人惊讶的是，目前这部作品仍可沿用您最初确立的内容架构。请问这种基础是如何构筑的？在目前的创作过程中，您用了哪些方法来保证自己的进度呢？

高德纳：是啊，我确实是几乎不间断地写计算机程序超过50年了，平均每周完成多于一个程序。譬如，我刚刚查了电脑，统计出我在今年的持续学习和探索中，到目前为止已经写了74个程序。当然，其中某些程序是短小和简单的，但另外那些可都够让我忙上一阵子的。这样的编程过程，很自然地启发了 *TAOCP* 的内容架构，我们能依此建立整个计算机科学的知识体系。1967年，我跟Peter Naur第一次见面时，我们发现各自都独立地对这一领域提出了完全一致的基本框架。

都过了50年啦，照理说我早该写完 *TAOCP* 才对。不过，我还有很多累积下来的材料，需要20年甚至更多的时间，才可以转化成恰当的文字。因此，当看到你问我怎样保持进度时，我都直想发笑。

要说我还是能有那么一点点进度的话，那最主要归功于采用了“批处理”而非“换入换出”的机制：在一个时间段内，我通常只全神贯注地做一件事情。每年我会暂停手上的工作两次，每次用两三周的时间阅读邮寄过来的期刊。我每周都会收到8份左右的期刊，我的秘书会把它们放到盒子里。浏览完它们并了解到技术

动向后，我会在自己的文件中加入备注，提醒自己在将来专注于另外的主题时，应该阅读哪些内容。

目前，我正聚精会神在“可满足性求解器”（SAT solvers）这个令人着迷的领域，最近编写的20个程序都是在这个主题做相关探索的。藉由自己去钻研资料的手段，我可以更好地将核心思想传达给非专家的读者，并将这些思想跟其他应用紧密结合，就仿佛我的一生都在专职研究可满足性的求解问题那么自然。幸运的是，我现在跟顶级的专家们保持着联系，他们自告奋勇帮我检查写作中的错误。

图灵社区：我们听说，您目前还是先写出手稿，再在计算机中编辑。然而，您的TeX实际上颠覆了整个出版行业。那么，请问您不全用计算机写作的原因是什么？*Dr. Dobb's*在评价TAOCP 4A的时候这样说道：“正如前几卷TAOCP那样，最新的这一卷也是浓缩了一个主题的精华内容。基于其高密度的内容和描述，本书是近年来屈指可数的必须在印刷版格式下阅读的计算机图书之一。书中为数庞大的数学注释，将使所有电子格式束手无策，PDF版就像一批处处写满文字的JPEG，难以卒读。”您是否考虑过，未来的电子写作和阅读应该是怎样的呢？

高德纳：我书写的速度跟我思维的速度是匹配的，这么一来，就完全不存在任何“瓶颈”。而我打字的速度就比我思考的速度更快，这样当我试图用键盘创作重要内容时，就会产生同步问题。

（事实上，我也是先用笔写下你这10个问题的答案的。此刻，我正在Mac上输入草稿，并在过程中尽可能修润行文。）

速度通常不会是最重要的标准。科学一般都难以迅速解释或迅速领会。我知道我的书是不容易读的，不过要知道的是，如果不是我精雕细琢地写的话，它们会比现在难读一百倍。

图灵社区：在[《编程人生》](#)中，您讨论到黑盒的问题时，评论道：“程序里有黑盒是不坏，但通常来说，如果可以看到盒子里的东西，弄清楚黑盒内部的机理，那就可以改进它。”我们觉得这里似乎蕴含着黑客的精神。如果是的话，您是否可以具体描述一下您心目中的黑客精神？

高德纳：关于黑客精神，Steven Levy那本了不起的《黑客》中描述得最好。那本书会同时地从众多层面来审视一个问题，并寻找新的形式来组合基本的概念。

图灵社区：您从来都以极客（geek）自诩，并曾在[访谈](#)中提到，论文集第8卷《娱乐和游戏论文集》（*Selected Papers on Fun and Games*）中有一章是“极客艺术品”（geek art）。大部分中国读者都还无缘读到这本书，是否可以简单介绍一下，“极客艺术品”所包含的内容呢？

高德纳：你们应该翻译那本书啊，我说真的！

简单说，能称得上“极客艺术品”的应该是这样的艺术作品：它不仅仅能因其美丽的颜色、质感和形式而打动我，同时也因能其对技术的呈现方式而愉悦我的另一半大脑。

例如，我最珍爱的极客艺术藏品中的一件，正是Bob Sedgewick（即Robert Sedgewick，[《算法》](#)的作者——译注）送给我的，那是1975年他刚完成关于快速排序的博士论文的时

候。那是一件瑰丽的双层编织的纺织品，图案正是他在研究中发现的其中一个数学模型。这个作品是他亲自在提花织机上手工织造的。类似的作品还有我妻子做给我的一张巧夺天工的被子，上面的图案是以爱因斯坦质数的迷人模型为基础的。去年，我自己也利用零碎时间做了一些作品，那是用色彩斑斓的线、樱桃木和黄铜钉交错而成的“凯尔特骑士之旅”。

很多人都认为高德纳是一名非常有趣的人物。他会奖励每一个找出他的著作中任何错误的人2.56美元，因为“256美分刚好是十六进制的一美元”。高德纳可以算是一名标准的黑客，他最喜欢的软件是Emacs，并甚至还向作者理查·史托曼提交修补补丁。

- Wikipedia

我的很多朋友都已经培养出对极客艺术品的品味。我听说，Nathan Myrvoild已经搜罗了几百件这样的作品，其中大部分都是为他的居室专门制作的。

图灵社区：您的TeX系统是开源的，您本人也被认为是开源的重要实践者。在曾经的访谈中，您说“过去的几十年间，开放源代码的成功可能是计算机领域中唯一没使我觉得惊讶的事情”。那么，在后面的几十年，您预想开源运动将会有怎样的发展呢？

高德纳：请别让我预测未来，也不要相信别人在这个问题上的说三道四。

回到开源，怎么说呢，有一件事是我希望发生的（我很奇怪为什么尚未发生）。换言之，我希望人们可以找到一种比较简单的途径，让用户能够定制他们的开源发行版。这么一来，所有人都可以使得系统根据他们自己的计算机进行优化的调适，因为用户是通过编译自己拿到的源代码，而不是仅仅安装（已经编译好的、未根据系统做好编译优化的）二进制包。开源系统有一种尚未开发的潜力，会使它大大好于任何闭源的系统，因为专有的、事先打包的二进制成品必须在可用硬件限制的条件下照顾到最差情况。举例来说，emacs对于我来说已经是又好又快，但我觉得如

果能毫无顾忌地在自己的机器上编译它的话，它运行起来还会快得多。我没空去学习Ubuntu这个发行版的所有底层复杂细节。

（我还真的重新编译过Linux内核——但只有在向导手把手的指引下才得以完成。）

图灵社区：虽然，*TAOCP*代表着您的主要成就，连您目前的头衔都是“计算机程序设计艺术荣休教授”，但也有很多人认为，您花十年时间开发的TeX对世界的影响更大。您对此有何看法？是否可以总结一下算法研究和实际编程之间的联系和各自作用呢？

高德纳：我对于把一项有益的活动排在另一项之前这种事，不十分感冒。例如，生物学家不应该把所有时间都花在攻克癌症和其他重症的疗法上。如果他们中的一些人仅在较轻微的问题上取得了重大进展——比如，消灭了头皮屑——他们也许实际上会带给更多人更持久的快乐。

TeX使得文学编程成为了可能，这件事长远来看也许最终会给更多人的生活带来积极的影响，这一点强过我所做的任何其他工作，因为文学化的程序给它的用户带来的改进是巨大的。

但我们还是别拿苹果去和橘子比较了。我认为生活中的每一个方面都是值得改进的，而我也很高兴能在自己生活的场所和时代中以多种不同的方式做出贡献。

图灵社区：《具体数学：计算机科学基础（第2版）》（*Concrete Mathematics: A Foundation for Computer Science, 2E*）的中文版同样会由图灵出版。是否可以谈谈它的写作初衷，以及它跟*TAOCP*的关系？

高德纳：《具体数学》是一份“纲领”，它的内容是我对于数学诸多方面应该如何教与学的思考。熟练掌握代数公式的基础技能，对我来说始终都是关键所在。这些内容在 *TAOCP* 里都有讨论，但只能是蜻蜓点水；在斯坦福大学的课程中，我得以深入更多的细节，而那些内容都被囊括在这本书中了。

图灵社区：“高德纳”似乎是您仅有的一个外文姓名，这个名字让中国读者很有亲切感。我们只知道这个名字是储枫教授（香港城市大学计算机科学系主任，图灵奖得主姚期智的夫人——译注）在您1977年访华前夕为您取的。给我们谈谈这个名字背后的故事吧。

高德纳：储枫告诉我，之所以选择“高”作为我的中国姓，是因为我个子高，还因为辅音G和K读起来差不多。“德纳”两个字，显而易见，是“Donald”不错的谐音，并且有着体面的意义。她还给我的爱人Jill起了“高精兰”这个名字。

我的两个孩子John和Jen也和我们一起来到了中国，他们当时分别是12和11岁——他们和中国孩子们在城市公园里玩了一些不需要语言交流的游戏。储枫给他们也分别起了“高小强”和“高小珍”的名字（见卷2索引）。

图灵社区：我们已经翻译了[关于您的管风琴的一篇介绍](#)，也读到您在访谈中曾经把写作比喻成演奏管风琴。可以谈谈音乐对您生活和研究的影响吗？

高德纳：音乐是我的主要副业，也是《娱乐和游戏论文集》一书其中四个章节的主要内容。最近闲下来的时间，我在 *TAOCP* 上连续工作几天并需要休息一下时，我开始（尽管只是试验性

[欢迎阅读访谈英文版](#)

地)着手谱写新的管风琴乐曲,也算是终于兑现了一些我在上世纪60年代就拟订了的计划。尽管我知道别人来做这些事的话,可以比我高明得多,但内心却有一个声音在催我歌唱!

图灵社区:最后,送上所有中国读者的最诚挚问候,祝您保持健康,如期完成 *TAOCP* 的下一卷!

高德纳:再次感谢你们富有启发的问题。■

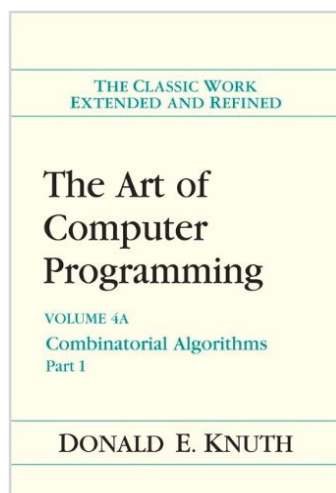
记者 / [@何逸勤](#)

译者 / [@何逸勤](#)

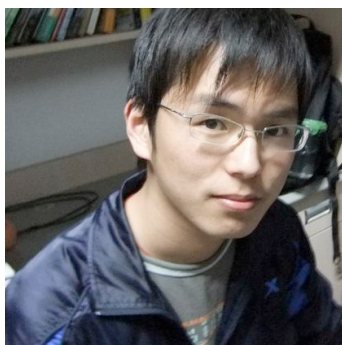
(图灵社区特约编辑)



译者 / [@高博](#)



关于TAOCP中用集合论对算法进行严格数学定义的理解



高德纳（Donald E. Knuth）在其名作*The Art Of Computer Programming*的第一卷*Fundamental Algorithms*中，用集合论对算法进行了严格的数学定义，仅仅用了一页，言简意赅，但是就这一页足以体现出他深厚的数学功底，和驾驭数学的高超能力，把数学语言的简洁与精确体现得淋漓尽致。我在读这一页时想到了很多，受到的启发远不止纸上的这些，现在把这些理解写在这里，希望对大家有所帮助。

作者 / [陆超超](#) @陆超超_lucc

南京大学软件工程专业的大学生，大四即将毕业，研究方向为computer vision和machine learning。喜欢理论也喜欢实践：时而研究数学与算法，时而写写code。闲暇时喜欢书法与绘画，无聊时喜欢登山旅游摄影。热爱生活，执着于梦想！

在介绍算法的数学定义之前，我们要明确算法的五个基本特征：

- 有限性**：描述了一个算法在执行有限步之后必然会终止。
- 确定性**：描述了一个算法的每个步骤都必须精确地定义，可以严格地、无歧义地执行。
- 输入**：描述了一个算法在运行之前赋给它的量，或在运行过程中动态地赋给它的量。
- 输出**：描述了一个算法运行结束时的结果。
- 有效性**：描述了一个算法在运行过程中的所

有运算必须是充分基本的，是可行的，原则上人们可以用笔和纸在有限的时间内精确地完成这些运算。

高教授在用集合论定义算法时是严格按照这五个基本特征来做的，下面我就用常用的集合论术语隆重地介绍算法的定义：

我们定义一个计算方法为一个四元组 (Q, I, Ω, f) ，每个字母的意义为：

Q : 表示计算状态的集合

I : 表示输入的集合

Ω : 表示输出的集合

f : 表示计算规则的集合

满足如下约束：

$$I \subset Q, \Omega \subset Q, f: Q \rightarrow Q$$

$$\forall q \in \Omega, f(q) = q$$

对集合 I 中的每一个输入 x 定义一个序列： $x_0, x_1, x_2, \dots$ ，满足如下规则：

$$x_0 = x,$$

$$x_{k+1} = f(x_k), k \geq 0$$

如果 k 是使 x_k 在 Ω 中为最小的整数，那么就说这个计算序列在 k 步中终止，而且在此种情况下说 x 产生了输出 x_k 。可能有些序列永远不会终止，这只能称为一个计算方法，而不能称之为算法。算法是一种对 I 中所有的 x 在有限步中终止的计算方法。

到目前为止，该定义虽然是用集合论描述的，但是丝毫没有难以理解的地方，就相当于一个有限状态机，每一节点既是上一个节点的输出节点，又是下一个节点的输入节点，最后 f 在输出集合 Ω 中保持逐点不动，很通俗易懂。然而，这个定义还不是很全

面，只包含了算法五个基本特征的前四个，并没有包含第五个有效性特征。在为该定义补充第五个基本特征之前，让我们先来用上述定义描述一个具体的算法，用一个大家都很熟悉的欧几里得算法。

欧几里得算法**E**： m 和 n 是两个正整数，求它们最大的公因子。

E1[求余数]： m 除以 n 并且令 r 为所得的余数。（ $0 \leq r < n$ ）

E2[判断余数是否为零]：如果 $r=0$ ，算法结束， n 就是输出答案。

E3[减少]：置 $m \leftarrow n$ ， $n \leftarrow r$ ，并返回步骤**E1**。

这个算法可以说是众人皆知，最平民化的算法。如何用上述算法定义中的术语来描述这个算法呢？首先在欧几里得算法中，我们涉及到单点（ n ），表示输出；有序数对（ m, n ），表示输入。除此之外，还涉及三个不同的步骤，每一步基本都与 m, n, r 这三个数有关，为了方便区别这三个不同的步骤，我们还要引入一个步骤的序列号，这样我们就可以用三个有序四元组分别表示每一个步骤了。所以欧几里得算法严格的数学描述如下：

欧几里得算法是一个四元组（ Q, I, Ω, f ），每个字母的意义为：

- Q : 所有单点（ n ），所有有序数对（ m, n ）和所有有序四元组（ $m, n, r, 1$ ），（ $m, n, r, 2$ ）以及（ $m, n, p, 3$ ）的集合，其中 m, n, p 是正整数， r 是一个非负整数。
- I : 所有有序数对（ m, n ）的子集。
- Ω : 所有单点（ n ）的子集。
- f 的定义如下：
 1. $f((m, n)) = (m, n, 0, 1)$

2. $f((m, n, r, 1)) = (m, n, m \% n, 2)$
3. 如果 $r=0$, $f((m, n, r, 2)) = (n)$; 否则 $f((m, n, r, 2)) = (m, n, r, 3)$
4. $f((m, n, p, 3)) = (n, p, p, 1)$
5. $f((n)) = (n)$

在这里有必要说明一下，上面的 f 定义了5条规则，第1条规则是得到输入数对之后初始化为四元组，才能参与后面的步骤；第2条对应于**E1**；第3条对应于**E2**；第4条对应于**E3**；第5条是算法结束的终止条件。显而易见，欧几里得算法这样的数学定义与先前的步骤是一一对应的，但是更加严格准确了。

现在，大家应该对算法的集合论定义有了比较清晰的了解。前面已经说过，该集合论的数学定义不尽如人意，还不包括前面的有效性基本特征。例如，当 Q 是仅用纸和笔而无法计算的无穷序列，或者 f 是仅靠人脑也无法实现的计算，那么这时不满足有效性的特征，也就不能称之为算法。所以为了使算法的数学定义更加完善，我们应该给 Q 和 f 加一些约束，使运算的每一步都是可行的、有效的，或更加具体一点说，就是使每一运算都只涉及初等运算。在计算机中，什么样的初等运算最具有代表性呢？当然是字符串的删除、添加、替换等操作啊，这些操作是绝对可行的，而且表示的含义也很广泛，当然你也可以使用其他的初等运算。接下来我们就在 Q 和 f 上加一些约束了：

首先，令 A 是字母的有限集合，并且令 A° 是 A 上所有串的集合，我们用 A° 的串来对计算的状态进行编码，即不同的串代表不同的计算状态。于是我们有：

- Q ：是所有 (σ, j) 的集合，其中 $\sigma \in A^\circ$, $0 \leq j \leq N$, N 为非负整数
- I ：是所有 $j=0$ 时 Q 的子集，即是所有 $(\sigma, 0)$ 的集合

- Ω : 是所有 $j = N$ 时 Q 的子集, 即是所有 (σ, N) 的集合

为了方便描述, 我们还要给出一个定义: 对于 $\theta, \sigma, \alpha, \omega \in A^*$, 如果 $\sigma = \alpha\theta\omega$, 则称 θ 出现在 σ 中。

现在我们可以给出 f 的约束了, 如下:

$$0 \leq j < N, \theta_j, \phi_j, \alpha, \omega \in A^*, a_j, b_j \in \mathbb{N}$$

F1式: 如果 θ_j 不出现在 σ 中, 那么: $f((\sigma, j)) = (\sigma, a_j)$

F2式: 如果 α 是满足 $\sigma = \alpha\theta_j\omega$ 的最短的字符串, 那么:

$$f((\sigma, j)) = (\alpha\phi_j\omega, b_j)$$

F3式: $f((\sigma, N)) = (\sigma, N)$

[注]:

1. **F1式**通俗地说就是: 如果在一个字符串 σ 中如果找不到另一个字符串 θ_j , 就跳转第 a_j 步。
2. **F2式**通俗地说就是: 如果在一个字符串 σ 中找到了另一个字符串 θ_j , 就用字符串 ϕ_j 替换字符串 θ_j , 然后跳转到第 b_j 步。
3. **F3式**是终止条件。

上面对于 f 的约束显然能确保算法的有效性基本特征, 因为上述定义的每一步都对字符串进行了简单的操作。事实上, **F1式**、**F2式**和**F3式**这三个式子足够可以描述我们手头的所有事情, 更具体一点就是, 足够可以描述所有的计算机程序。为什么这样说呢? 我们知道, 所有的计算机算法程序都是用某种编程语言写的, 而所有的编程语言无外乎三种结构: 顺序, 选择和循环。上面的三个式子正好可以描述这三种结构, 下面一一进行解释:

- 顺序结构

这个最简单，这三个式子任意执行就可以了，而且只要指定 j ，下一步就会进行你指定的第 j 个运算，所以不再赘述了。

- 选择结构

F1式和**F2式**结合起来使用便是选择结构：如果 θ_j 不出现在 σ 中，就执行**F1式**，如果 θ_j 出现在 σ 中，就执行**F2式**。这里字符串中的出现与不出现就分别表示了相反的两
种计算状态。

- 循环结构

在**F2式**中，令 $b_j=j$ ，便会进行循环运算，然后用**F1式**判断是否跳出循环，非常容易。

而**F3式**是描述算法执行结束的状态。所以用上面的**F1式**、**F2式**和**F3式**三个式子可以描述所有的算法，而且每一步都是有效的，到这里算法的集合论数学定义才算完备，囊括了算法所有的五个基本特征。

除此之外，编程语言中最基本的数据类型——整型，可以用某个字符的个数来表示，而编程语言中最基本的初等运算——加减法，可以用字符串的替换来表示。所以现在，我可以说基于字符串集合 A° 的**F1式**、**F2式**和**F3式**三个式子就构成了一门编程语言，通过它们可以编写任何你想要的合法程序，和那些高级语言（如C语言等）效果一样。但是这是一门有严格数学定义的编程语言，为了以后叙述的方便，我将其命名为**M语言**（是Math语言的简称，后面均用此简称）。

M语言中仅仅涉及 $\theta_j, \phi_j, a_j, b_j$ 这四个变量（变量的定义和前面一致），所以这四个变量的取值就决定了**M语言**中语句的每一步执

行。换句话说，在**M语言**中每一条语句就是这四个变量的取值，因此每一条语句就可以用这个四元组表示。以后，我们如果说用**M语言**来编写程序，就是说用这四元组来编写程序，程序的每一条语句都是四元组，整个程序就是由这些四元组构成的。

为了熟练使用这种由集合论进行了严格数学定义的算法语言——**M语言**，最后，我将**M语言**应用于一个具体的例子，用来说明其强大的描述能力。当然，我们还是以欧几里得算法为例，考虑到用字符串来描述初等运算，我们使用欧几里得辗转相减法，而不是原先的辗转相除法。

题目表述如下：

通过描述上面的**F1式**、**F2式**和**F3式**三个式子中的 $\theta_j, \phi_j, a_j, b_j$ ，给出计算正整数 m 和 n 的最大公因子的一个“有效性”算法，令输入

为字符串 $\underbrace{aa\cdots a}_m \underbrace{bb\cdots b}_n$ ，输出结果是 $\underbrace{bb\cdots b}_{\gcd(m,n)}$ ，当然你完全可以输出除 b 以外的其他字母，这里只是为了方便而已。希望你尽可能给出简单的解。

我用自己的话重述一下这个题目：就是用**M语言**编写求最大公因子的欧氏辗转相减法。根据前面所述，就是求一系列四元组。首先还是按照前面介绍欧氏辗转相除法的方式来介绍辗转相减法，如下：

欧氏辗转相减法**ERM**： m 和 n 是两个正整数，求它们最大的公因子。

ERM1[求差]： m 减 n 并且令 r 为所得的差的绝对值。（ $0 \leq r < n$ ）

ERM2[判断差是否为零]：如果 $r=0$ ，算法结束， n 就是输出答

案。

ERM3[减少]: 置 $m \leftarrow \min(m, n)$, $n \leftarrow r$, 并返回步骤**ERM1**。

这个算法**ERM**和前面的算法**E**的正确性很容易证明, 限于篇幅, 这里就略去了。下面主要讲解一下我解决这道题目的思路:

题中两个正整数 m 和 n 分别是用 a 和 b 的个数表示的, 所以要求 m 和 n 的差, 只要每次分别去掉一个“ a ”和一个“ b ”, 为了便于字符串的操作, 每次直接去掉“ ab ”, 一直到只剩下“ $a---$ ”序列或“ $b---$ ”序列为止, 这时剩下的“ a ”或“ b ”的个数就是 m 和 n 差的绝对值 r , 这就完成了步骤**ERM1**。

为了方便步骤**ERM3**的赋值操作, 我们需要知道 $\min(m, n)$ 到底是哪一个, 显然 $\min(m, n)$ 就是去掉的“ ab ”的个数, 因此我们需要换一种简单的方式保留“ ab ”的个数, 就用另外一个字母“ c ”替换“ ab ”, 这就是**F2式**中的操作。但是“ c ”会把字符串中的“ a ”序列和“ b ”序列分开, 无法进行继续去掉“ ab ”的操作, 因此我们要把“ c ”移到字符串的最左边或最右边(这就导致了两种解法, 下文我都会给出)。这种移到最左边或最右边的操作一般会是连续重复的, 这就会使用到上文所说的**F2式**中的循环操作。然后继续替换“ ab ”也会使用该循环操作。等替换完所有的“ ab ”后, 这时可以执行步骤**ERM3**, “ c ”的个数就是 $\min(m, n)$, 余下的“ a ”或“ b ”的个数就是 r 。

接下来要分两种情况, 如果把“ c ”移到了最左端, 由于跳到步骤**ERM1**后要继续替换“ ab ”, 所以这时要将所有的“ c ”变成“ a ”, 将所有的“ b ”或“ a ”都变成“ b ”; 如果把“ c ”移到了最右端, 这时要将所有的“ c ”变成“ b ”, 将所有的“ b ”或“ a ”都变成“ a ”, 这些就是执行步骤**ERM3**, 当然这些过程都会用到**F1式**中的跳转操作和**F2式**中的循环操作。执行完步骤**ERM3**后就会重复执行步骤**ERM1**, 以此循

环直到执行步骤**ERM2**，即 $r=0$ ，也就是说字符串中只剩下“ c ”的序列，这时“ c ”的个数就是 m 和 n 的最大公因子（ $\gcd(m, n)$ ）。当然不一定最后以“ c ”的表示输出，可以输出任意其他字符，那么就可以用**F2**式中的循环替换操作。

下面我就给出该题目的一种解答（将“ c ”移到最左边），并将详细解释该**M**语言编写的程序的每一步，该程序如下：

j	θ_j	ϕ_j	a_j	b_j
0	ab	c	1	2
1	ac	ca	1	0
2	a	b	2	3
3	c	a	3	4
4	b	b	0	5
5	a	b	5	5

【上述**M**语言版的辗转相减法程序注解】： j 表示程序的步骤号，每个步骤的具体含义为：

- 第0步：如果找到“ ab ”就用“ c ”替换，然后跳转到第1步；如果找不到“ ab ”，就直接跳转到第2步。（这一步就是用“ c ”替换“ ab ”）
- 第1步：如果找到“ ac ”就用“ ca ”替换，然后循环执行这一步；否则直接跳转到第0步。（这一步就是将“ c ”移到最左边）
- 第2步：如果找到“ a ”就用“ b ”替换，然后循环执行这一步；否则直接跳转到第3步。（这一步就是执行**ERM3**中的 $n \leftarrow r$ 操作）

- 第3步：如果找到“*c*”就用“*a*”替换，然后循环执行这一步；否则直接跳转到第4步。（这一步就是执行ERM3中的 $m \leftarrow \min(m, n)$ 操作）
- 第4步：如果找到“*b*”就用“*b*”替换，然后跳转到第0步；否则直接跳转到第5步。（这一步就是相当于执行ERM2的判断）
- 第5步：如果找到“*a*”就用“*b*”替换，然后循环执行这一步（这一步就是为了满足题目的要求最终以“*b*”序列的形式输出结果）

为了使该M语言版算法的步骤更加清晰，我给出一个实例：
 $m=6$ ， $n=4$ ，此时的输入为 $\sigma=aaaaaabb$ ，执行过程图如下：

$aaaaaabb \rightarrow aaaaacbb \rightarrow aaaacabb \rightarrow aaacaabb \rightarrow aacaaabb$
 $\rightarrow acaaaabb \rightarrow caaaaabb \rightarrow caaaacbb \rightarrow caaacabb \rightarrow caacaabb \rightarrow cacaaabb$
 $\rightarrow ccaaaabb \rightarrow ccaaacb \rightarrow ccaacab \rightarrow ccacaab \rightarrow cccaaab \rightarrow cccaac \rightarrow cccaca$
 $\rightarrow ccccaa \rightarrow ccccba \rightarrow cccbb \rightarrow acccbb \rightarrow aaccbb \rightarrow aaacbb \rightarrow aaaabb \rightarrow aaacb$
 $\rightarrow aacab \rightarrow acaab \rightarrow caaab \rightarrow caac \rightarrow caca \rightarrow ccaa \rightarrow ccba \rightarrow ccbb \rightarrow acbb \rightarrow aabb$
 $\rightarrow acb \rightarrow cab \rightarrow cc \rightarrow ac \rightarrow aa \rightarrow ba \rightarrow bb$

执行结果输出结果 $\sigma=bb$ ，即 $\gcd(6, 4)=2$ 。

同理，我再给出该题目的另外一个解（将“ c ”移到最右边），该程序如下：

j	θ_j	ϕ_j	a_j	b_j
0	ab	c	1	2
1	cb	bc	1	0
2	b	a	2	3
3	c	b	3	4
4	b	b	0	5
5	a	b	5	5

分析同上面的将“ c ”移到最左边的解。

最后为了完整性，同时考虑通用性，我将用伪代码的形式实现由**F1式**、**F2式**和**F3式**所严格定义的算法通用程序，如下：

```
1 while(!terminate condition){
2     find the first  $\theta_j$  in the input_string;
3     if(No Found){
4          $j=a_j$ ;
5     }else{
6         input_string=the sub_string in front of  $\theta_j$ 
7             +  $\phi_j$ 
8             + the sub_string behind  $\theta_j$ ;
9          $j=b_j$ ;
10    }
11 }
```

上面伪代码中选择语句if…else…的第一部分相当于**F1式**，第二部分相当于**F2式**，而terminate condition相当于**F3式**，所以具有

通用性，很容易将上述伪代码用高级语言编程实现，例如依照该伪代码，可以轻松地将上面讲的辗转相减法的**M**语言版用高级语言实现。下面就是我用C++实现的辗转相减法的第一种解（第二种解几乎一样，只要改一下状态表中的变量值即可）：

```
1      #include    <iostream>
2      #include    <string>
3
4      using namespace std;
5
6      int main (){
7          int j=0;
8          string theta[]={"ab","ac","a","c","b","a"};
9          string phi[]={"c","ca","b","a","b","b"};
10         int b[]={1,1,2,3,0,5};
11         int a[]={2,0,3,4,5,5};
12         string input;
13
14         cout<<"Input:";
15         cin>>input;
16         cout<<"Output:"<<endl;
17
18         while(input.find("a",0)!=-1 || input.find("c",0)!
19         ==-1){
20             int location=input.find(theta[j],0);
21             if (location!=-1){
22                 j=a[j];
23             }else{
24                 input=input.substr(0,location)
25                     +phi[j]
26                     +input.substr(location
27                     +theta[j].length());
28                 j=b[j];
29
30                 cout<<input<<"→";
```

```

29         }
30     }
31     cout<<"Success"<<endl;
32     system("pause");
33     return 0;
34 }

```

假设输入aaaaabbbb，则运行该程序的结果为：

```

C:\Users\LuChaochao\Desktop\Algorithm\Debug\Algorithm.exe
Input:aaaaabbbb
Output:
aaaaacbbb→aaaacabbb→aaacaabbb→aacaabbbb→acaaaabbb→caaaaabbb→caaaacbb→caaa
cabb→caacaabb→cacaaabb→ccaaaabb→ccaaacb→ccaacab→ccacaab→cccaaab→cccaac→
cccaca→ccccaa→ccccha→cccbb→acccb→aacbb→aaacbb→aaaabb→aaaabb→aaach→a
acab→acaab→caaab→caac→caca→ccaa→ccba→ccb→acbb→aabb→aabb→ach→cab→cc
→ac→aa→ba→bb→Success
请按任意键继续. . .

```

海量用户积分排名算法探讨

作者 / [魏大刚](#)

[@weidagang](#)

07年获得四川大学计算机学院数据库与知识工程硕士。擅长算法设计和架构设计，喜欢尝试各种不同的语言，但最欣赏Perl的风格。设计过自然语言搜索引擎框架、高性能期权期货交易中间件，目前致力于基于Android的软硬件整合和云端整合。

问题

某海量用户网站，用户拥有积分，积分可能会在使用过程中随时更新。现在要为该网站设计一种算法，在每次用户登录时显示其当前积分排名。用户最大规模为2亿；积分为非负整数，且小于100万。

PS: 据说这是迅雷的一道面试题，不过问题本身具有很强的真实性，所以本文打算按照真实场景来考虑，而不局限于面试题的理想环境。

存储结构

首先，我们用一张用户积分表`user_score`来保存用户的积分信息。

表结构：

Field	Type	Null	Key	Default
uid	int(11)	NO	PRI	0
score	int(11)	NO		0

示例数据：

uid	score
1	232
2	100289
3	0
4	99
5	878
6	999999
7	298892
8	69891
9	8
10	555

下面的算法会基于这个基本的表结构来进行。

算法1：简单SQL查询

首先，我们很容易想到用一条简单的SQL语句查询出积分大于该用户积分的用户数量：

```
1 select 1 + count(t2.uid) as rank
2 from user_score t1, user_score t2
3 where t1.uid = @uid and t2.score > t1.score
```

对于4号用户我们可以得到下面的结果：

```
mysql> select 1 + count(t2.uid) as rank
-> from user_score t1, user_score t2
-> where t1.uid=4 and t2.score>t1.score;
+-----+
| rank |
+-----+
|    8 |
+-----+
1 row in set (0.00 sec)
```

算法特点

优点：简单，利用了SQL的功能，不需要复杂的查询逻辑，也不引入额外的存储结构，对小规模或性能要求不高的应用不失为一种良好的解决方案。

缺点：需要对user_score表进行全表扫描，还需要考虑到查询的同时若有积分更新会对表造成锁定，在海量数据规模和高并发的应用中，这样做性能是无法接受的。

算法2：均匀分区设计

在许多应用中缓存是解决性能问题的重要途径，我们自然会想：能不能把用户排名用Memcached缓存下来呢？不过再一想发现缓存似乎帮不上什么忙，因为用户排名是一个全局性的统计指标，而并非用户的私有属性，其他用户的积分变化可能会马上影响到本用户的排名。然而，真实的应用中积分的变化其实也是有一定规律的，通常一个用户的积分不会突然暴增暴减，一般用户总是要在低分区混迹很长一段时间才会慢慢升入高分区，也就是说用户积分的分布总体说来是有区段的，我们进一步注意到高分区用户积分的细微变化其实对低分段用户的排名影响不大。于是，我们可以想到按积分区段进行统计的方法，引入一张分区积分表score_range：

表结构：

Field	Type	Null	Key	Default
from_score	int(11)	NO	PRI	0
to_score	int(11)	NO	PRI	0
count	int(11)	NO		0

数据示例：

from_score	to_score	count
0	1000	48892
1000	2000	25329
2000	3000	12568
3000	4000	10207

表示[from_score, to_score)区间有count个用户。若我们按每1000分划分一个区间则有[0, 1000), [1000, 2000), ..., [999 000, 1 000 000)这1000个区间，以后对用户积分的更新要相应地更新score_range表的区间值。在分区积分表的辅助下查询积分为s的用户的排名，可以首先确定其所属区间，把高于s的积分区间的count值累加，然后再查询出该用户在本区间内的排名，二者相加即可获得用户的排名。

乍一看，这个方法貌似通过区间聚合减少了查询计算量，实则不然。最大的问题在于：如何查询用户在本区间内的排名呢？如果是在算法1中的SQL中加上积分条件：

```
1 select 1 + count(t2.uid) as rank
2 from user_score t1, user_score t2
3 where t1.uid = @uid and t2.score > t1.score and
   t2.score < @to_score
```

在理想情况下，由于把t2.score的范围限制在了1000以内，如果对score字段建立索引，我们期望本条SQL语句将通过索引大大减少扫描的user_score表的行数。不过真实情况并非如此，t2.score的范围在1000以内并不意味着该区间内的用户数也是1000，因为这里有积分相同的情况存在！二八定律告诉我们，前20%的低分区往往集中了80%的用户，这就是说对于大量低分区用户进行区间内排名查询的性能远不及对少数高分区用户进行排名查询，所以在一般情况下这种分区方法不会带来实质性的性能提升。

算法特点

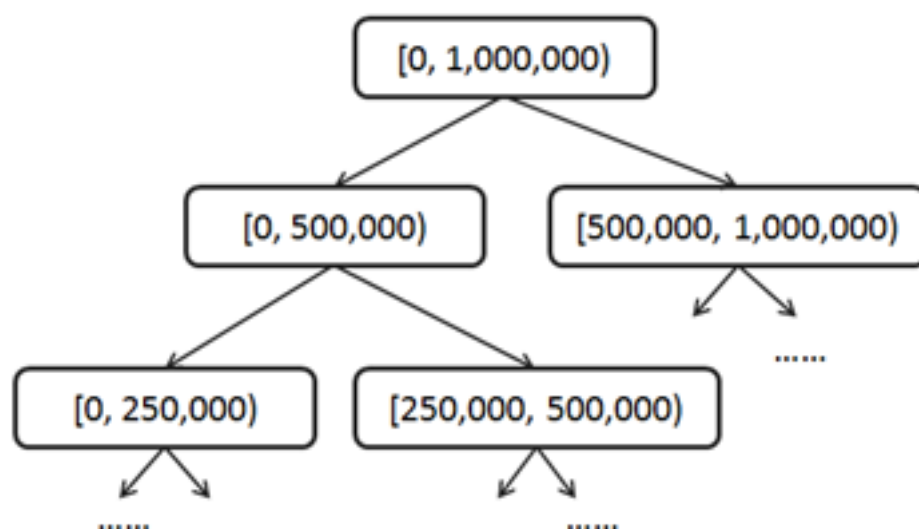
优点：注意到了积分区间的存在，并通过预先聚合消除查询的全表扫描。

缺点：积分非均匀分布的特点使得性能提升并不理想。

算法3：树形分区设计

均匀分区查询算法的失败是由于积分分布的非均匀性，那么我们自然就会想，能不能按二八定律，把score_range表设计为非均匀区间呢？比如，把低分区划密集一点，10分一个区间，然后逐渐变成100分，1000分，10 000分 当然，这不失为一种方法，不过这种分法有一定的随意性，不容易把握好，而且整个系统的积分分布会随着使用而逐渐发生变化，最初的较好的分区方法可能会变得不适应未来的情况了。我们希望找到一种分区方法，既可以适应积分非均匀性，又可以适应系统积分分布的变化，这就是树形分区。

我们可以把[0, 1 000 000)作为一级区间；再把一级区间分为两个2级区间[0, 500 000), [500 000, 1 000 000)，然后把二级区间二分为4个3级区间[0, 250 000), [250 000, 500 000), [500 000, 750 000), [750 000, 1 000 000)，依此类推，最终我们会得到1 000 000个21级区间[0,1), [1,2) ... [999 999, 1 000 000)。这实际上是把区间组织成了一种平衡二叉树结构，根结点代表一级区间，每个非叶子结点有两个子结点，左子结点代表低分区，右子结点代表高分区。树形分区结构需要在更新时保持一种不变量(Invariant)：非叶子结点的count值总是等于其左右子结点的count值之和。



以后，每次用户积分有变化所需要更新的区间数量和积分变化量有关系，积分变化越小更新的区间层次越低。总体上，每次需要更新的区间数量是用户积分变量的 $\log n$ 级别的，也就是说如果用户积分一次变化在百万级，更新区间的数量在二十这个级别。在这种树形分区积分表的辅助下查询积分为 s 的用户排名，实际上是一个在区间树上由上至下、由粗到细一步步明确 s 所在位置的过程。比如，对于积分499 000，我们用一个初值为0的排名变量来做累加；首先，它属于1级区间的左子树 $[0, 500\ 000)$ ，那么该用户排名应该在右子树 $[500\ 000, 1\ 000\ 000)$ 的用户数 $count$ 之后，我们把该 $count$ 值累加到该用户排名变量，进入下一级区间；其次，它属于3级区间的 $[250\ 000, 500\ 000)$ ，这是2级区间的右子树，所以不用累加 $count$ 到排名变量，直接进入下一级区间；再次，它属于4级区间的.....如此往复，直到最后我们把用户积分精确定位在21级区间 $[499\ 000, 499\ 001)$ ，整个累加过程完成，得出排名！

虽然，本算法的更新和查询都涉及若干个操作，但如果我们为区间的`from_score`和`to_score`建立索引，这些操作都是基于键的查询和更新，不会产生表扫描，因此效率更高。另外，本算法并不依赖于关系数据模型和SQL运算，可以轻易地改造为NoSQL等其他存储方式，而基于键的操作也很容易引入缓存机制进一步优化性能。进一步，我们可以估算一下树形区间的数目大约为200 000 000，考虑每个结点的大小，整个结构只占用几十M空间。所以，我们完全可以在内存建立区间树结构，并通过`user_score`表在 $O(n)$ 的时间内初始化区间树，然后排名的查询和更新操作都可以在内存进行。一般来讲，同样的算法，从数据库到内存算法的性能提升常常可以达到 10^5 以上；因此，本算法可以具有非常高的性能。

算法特点

优点：结构稳定，不受积分分布影响；每次查询或更新的复杂度为积分最大值的 $O(\log n)$ 级别，且与用户规模无关，可以应对海量规模；不依赖于SQL，容易改造为NoSQL或内存数据结构。

缺点：算法相对更复杂。

算法4：积分排名数组

算法3虽然性能较高，达到了积分变化的 $O(\log n)$ 的复杂度，但是实现上比较复杂。另外， $O(\log n)$ 的复杂度只在 n 特别大的时候才显出它的优势，而实际应用中积分的变化情况往往不会太大，这时和 $O(n)$ 的算法相比往往没有明显的优势，甚至可能更慢。

考虑到这一情况，仔细观察一下积分变化对排名的具体影响，可以发现某用户的积分从 s 变为 $s+n$ ，积分小于 s 或者大于等于 $s+n$ 的其他用户排名实际上并不会受到影响，只有积分在 $[s, s+n)$ 区间内的用户排名会下降1位。我们可以用一个大小为100 000 000的数组表示积分和排名的对应关系，其中 $\text{rank}[s]$ 表示积分 s 所对应的排名。初始化时， rank 数组可以由 user_score 表在 $O(n)$ 的复杂度内计算而来。用户排名的查询和更新基于这个数组来进行。查询积分 s 所对应的排名直接返回 $\text{rank}[s]$ 即可，复杂度为 $O(1)$ ；当用户积分从 s 变为 $s+n$ ，只需要把 $\text{rank}[s]$ 到 $\text{rank}[s+n-1]$ 这 n 个元素的值增加1即可，复杂度为 $O(n)$ 。

算法特点

优点：积分排名数组比区间树更简单，易于实现；排名查询复杂度为 $O(1)$ ；排名更新复杂度 $O(n)$ ，在积分变化不大的情况下非常高效。

缺点：当 n 比较大时，需要更新大量元素，效率不如算法3。

总结

上面介绍了用户积分排名的几种算法，算法1简单，易于理解和实现，适用于小规模和低并发应用；算法3引入了较复杂的树形分区结构，但是 $O(\log n)$ 的复杂度性能优越，可以应用于海量规模和高并发；算法4采用简单的排名数组，易于实现，在积分变化不大的情况下性能不亚于算法3。本问题是一个开放性的问题，相信一定还有其他优秀的算法和解决方案，欢迎探讨！ ■

图说归并排序

简介

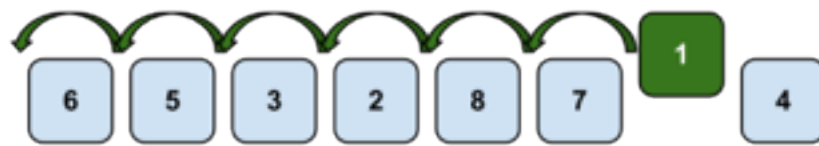
一般来说，排序算法主要被分为两类，即基于比较的算法和基于非比较的算法。我已经发了一些关于前一类算法的帖子。插入排序，冒泡排序和希尔排序是基于比较模型的。这三个算法的问题是它们的复杂度都是 $O(n^2)$ ，所以它们非常慢。

那么有没有比 $O(n^2)$ 更快的排序列表的方法呢？答案是有的，下面就让我们看看这么一个算法。

作者 / Stoimen
[@stoimenpopov](#)

之前提到的三个算法（插入、冒泡和希尔）的共同特点是，我们是从原始列表里把元素两两取出，然后进行比较的。

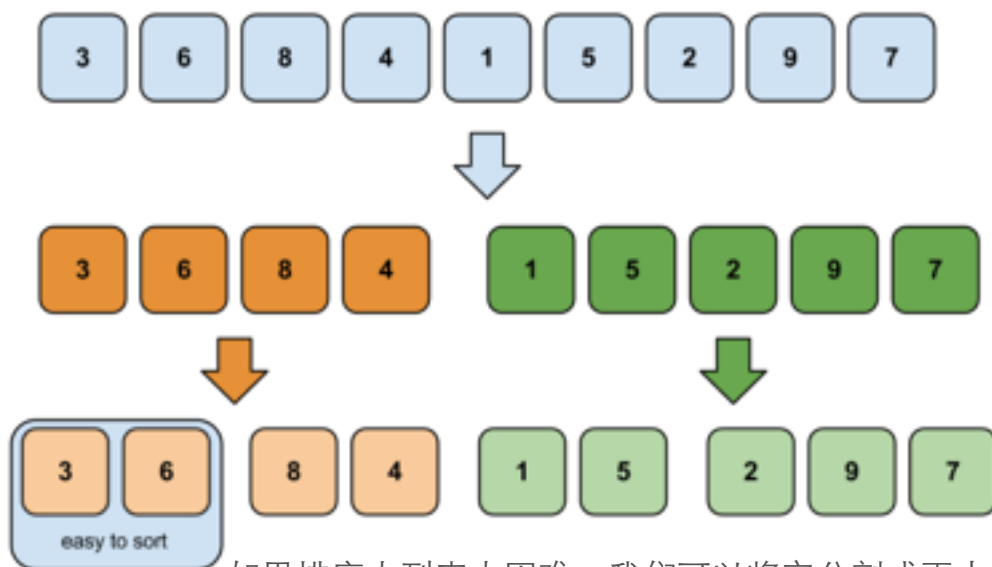
Insertion Sort



插入排序和冒泡排序使用了太多的比较，这也是归并排序需要克服的地方

在原始列表里进行比较不是最好的方式，而且我们也不需要那么做。作为替代做法，我们可以尝试将列表分成更小的子列表然后排序它们。在排序完更小的子列表以后（这么做比直接排序整个原始列表要简单），我们可以尝试将小的子列表合并成一个有序列表。这个技术就是典型的“分治”法（分而治之）。

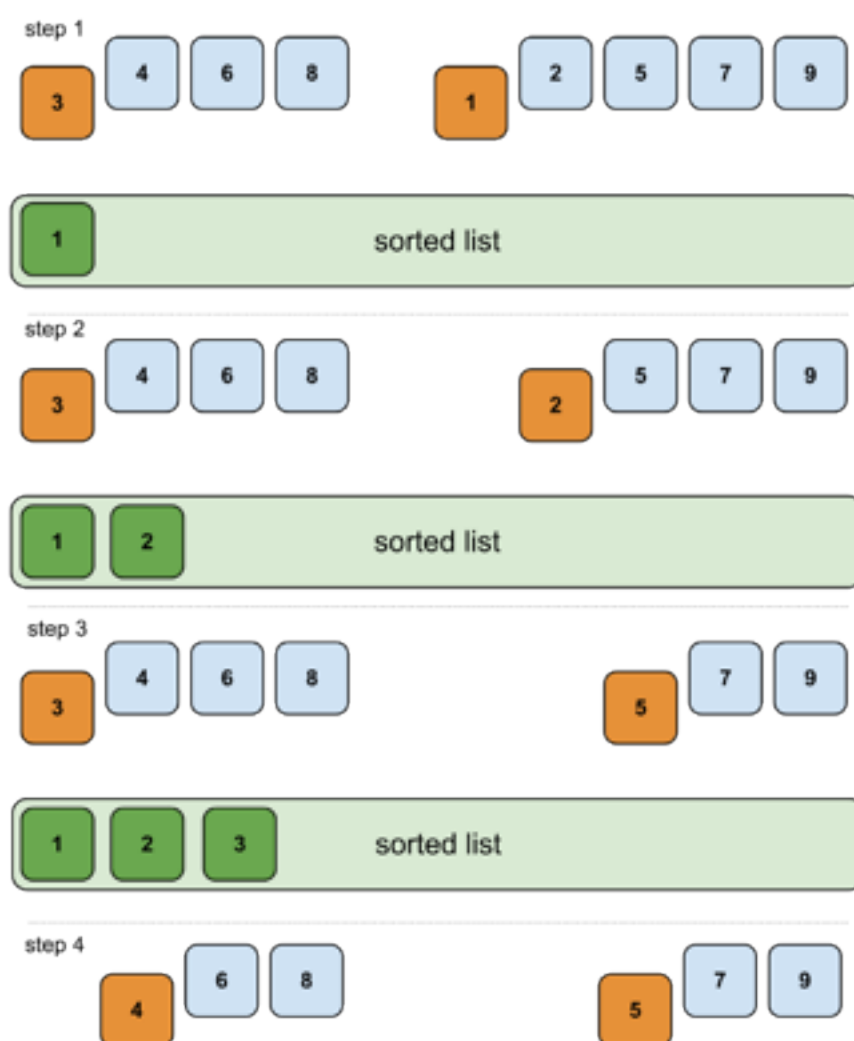
一般来说，如果一个问题太难以至于无从下手，我们可以尝试将它分成较小的子问题，然后尝试解决这些子问题。然后我们可以将子问题的结果合并起来（从而解决原始问题）。



如果排序大列表太困难，我们可以将它分割成更小的子列表，然后排序它们

综述

归并排序是一个基于分治法的比较排序算法。目前看来还不错，当我们有一个非常大的列表需要排序，显然，如果将列表分成两个子列表，然后再排序会更好些。如果等分以后还是太大，我们就继续分割，直到可以很简单地排序为止（参见下图）。



合并排序过的子列表

上图描述了在算法的某些步骤中，我们已经得到两个排序的列表，而巧妙的地方是合并它们。然而，这也不是特别难。我们可以比较两个列表的第一个元素，将其中较小的元素取出放在一个新的列表中（这样得到的列表一定是有序的列表）。

实现

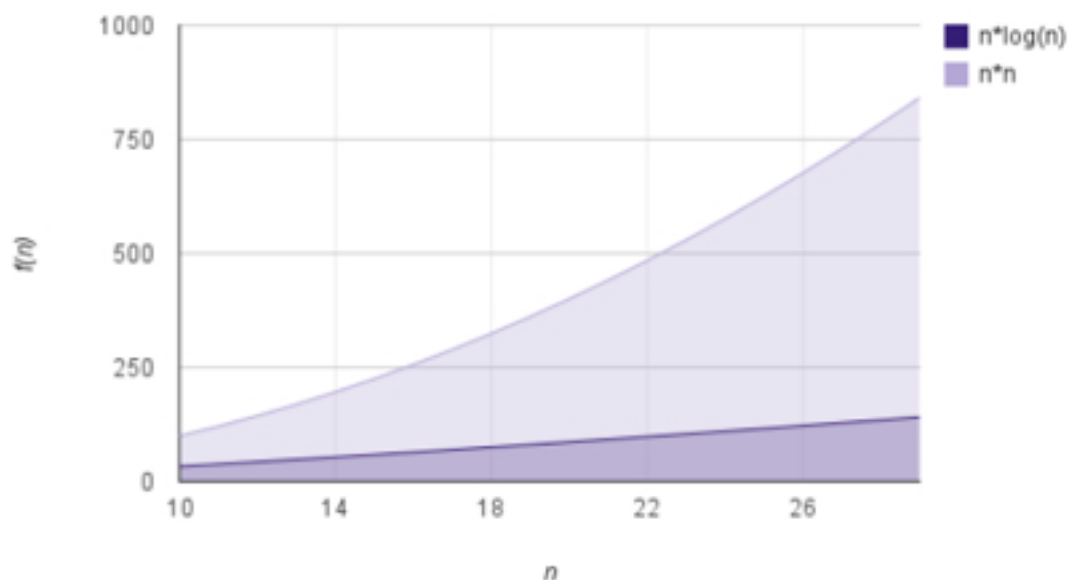
好在这个算法不但快速而且实现起来也不难，站一个开发者角度上，这实在是太美妙了。下面是PHP版的实现。注意：每一个遵从分治法的算法都可以使用递归方案轻易实现。然而，因为递归的性能很糟，所以你可能需要找到一个迭代的方式。一般地，递归可以被花费额外内存空间的迭代方式取代。下面是一个递归版的归并排序。

```
1  $input = array(6, 5, 3, 1, 8, 7, 2, 4);
2
3  function merge_sort($arr)
4  {
5      if (count($arr) <= 1) {
6          return $arr;
7      }
8
9      $left = array_slice($arr, 0, (int)(count($arr)/
10  2));
11  $right = array_slice($arr, (int)(count($arr)/2));
12
13  $left = merge_sort($left);
14  $right = merge_sort($right);
15
16  $output = merge($left, $right);
17
18  return $output;
```

```
18 }
19
20
21 function merge($left, $right)
22 {
23     $result = array();
24
25     while (count($left) > 0 && count($right) > 0) {
26         if ($left[0] <= $right[0]) {
27             array_push($result,
array_shift($left));
28         } else {
29             array_push($result,
array_shift($right));
30         }
31     }
32
33     array_splice($result, count($result), 0, $left);
34     array_splice($result, count($result), 0, $right);
35
36     return $result;
37 }
38
39 // 1, 2, 3, 4, 5, 6, 7, 8
40 $output = merge_sort($input);
```

复杂度

归并排序的复杂度不是那么高，这一点是非常棒的，即使在最差情况下它的复杂度也只是 $O(n \log n)$ 。值得注意的是：即便是快速排序的复杂度也可能在最差情况下达到 $O(n^2)$ 。所以我们可以确定，归并排序无论在什么情况下都非常稳定。



	Best case	Average case	Worst case
Merge sort	$n \log(n)$	$n \log(n)$	$n \log(n)$
Quicksort	$n \log(n)$	$n \log(n)$	n^2
Heapsort	$n \log(n)$	$n \log(n)$	$n \log(n)$
Bubble sort	n	n^2	n^2
Insertion sort	n	n^2	n^2

为什么归并排序如此有用？

1. 快捷和稳定

归并排序成为一个非常棒的排序算法主要是因为它的快捷和稳定。它的复杂度即使在最差情况下都是 $O(n \log n)$ 。而快速排序在最差情况下的复杂度是 $O(n^2)$ ，当 $n=20$ 的时候，它比归并要慢4.6倍。



for $n = 20$ merge sort is about 4.6 times faster than bubble sort!

2. 容易实现

另一个理由是归并排序很容易实现。诚然，大多数开发者认为速度快的算法总是难以实现，但是这条守则并不适用于归并排序的情况。

归并排序不那么实用的三个理由

1. 比非比较排序算法慢

归并算法是基于比较模型的，因此它要比在线性时间里排序数据的非比较算法要慢。当然，这也与输入数据有关，所以我们要仔细对待输入。

2. 对于初学者来说比较难实现

尽管我不认为这是一个不用归并排序的主要理由，但仍有一些人说它对于初学者来说太难实现了，尤其是合并部分的算法。

3. 在数据基本有序的情况下，它比冒泡和插入排序要慢

再一次提醒，要了解输入数据的重要性。如果输入数据基本已经有序，那么插入和冒泡排序会更快。插入和冒泡排序在最好的情况下复杂度是 $O(n)$ ，然而归并排序的最好情况是 $O(n \log n)$ 。

作为总结，我想说在实践中，归并排序是最好的排序算法毋庸置疑，因为它易于实现而且快捷，所以每个开发者都应该牢记它。



查看英文原文：

[Computer Algorithms:
Merge Sort](#)

译者 / [王利霞](#)

从事领域：java web开发、OSGI开发，个人兴趣：编程、上网、看电视、看书，这就是生活。

终身Coder，可以吗？

——老码农如何延续技术生命



梁斌

[@梁斌penny](#)

清华大学计算机系在读博士，THUIRDB和微博寻人的Coder，《走进搜索引擎》、《深入搜索引擎》作者，10年老码农群群主。

《码农》电子刊迎来了第一位受访Coder——梁斌同学。今年3月梁斌同学心血来潮，创建了10年老码农群，意外引起各路大牛关注，很多重量级的Coder纷纷加入，仅仅10天后码农群就开始了第一次Social活动。此外，梁斌也在探索如何延长码农的技术生命，让我们一起来看看他对码农的看法。

图灵社区：为什么您会想建这样一个10年老码农群呢？

梁斌：很多人说，编码这活干到35岁就到头了，继续编码，不从事管理岗，职业的不安全感越来越高。我就思考中国能不能有10年以上的老码农，老码农怎么延长自己的技术生命。建这个群的想法就自然出来了——10年老码农聚在一起，意在抱团取暖。

10年老码农群里有各种各样身份的人，创业的、当老大的、编程的。码农有需要的时候，可以在群里寻求帮助，大家八仙过海，各显神通。举个例子，比如说我，现在从清华毕业，年纪也大了，不好找工作，做的软件也没人买，但群里有人知道你的价值，知道梁斌还能当个仓库保管员（笑），这样机会就来了。也有很多公司，特别是初创公司程序员都是新人，非常需要有经验的老码农来凝聚开发团队。有这种需求的时候，他们会直接来这个群找。类似的信息都会在群里流动，我们算是把肉都揽到自己锅里了……

而且如果有天美帝问，为什么中国没有10年、20年以上编码的人？这个群也算是一个回应，总不能答曰中国人思维活跃、性情奔放，不爱长期做一件事情吧。

图灵社区：为什么很多人毕业选择工作时，对编码还很有热情，但经过多年工作，却不爱这行了？

梁斌：我觉得有很多原因，而且想明白这些原因都很难。

第一，也是最重要的一点，就是虽然你的年纪大了，但公司或者团队对你的期望还是跟年轻人一样——输出大量的代码，但你的精力已经无法匹敌年轻的时候，不能直接跟新码农拼了。但老码农在其他方面是有优势的，经验很丰富、很值钱，就会转而考虑从事别的工作。

第二，这个社会的诱惑特别多，无论是创业的诱惑，还是做管理层赚大钱的诱惑。面对这些诱惑的时候，你还能安心地编程吗？许多跟你一样的人选择其他机会后，价值发挥得比你大得多。你虽然还很热爱编码，但是当老婆揪着耳朵说“你看某某编码水平还没你高，车子买了，游艇买了，国外都去了好几次”，或者在同学聚会看到旧同学各有发展的时候，就会感受到压力，毕竟人不可能永远穷下去。

图灵社区：国外的程序员有这种问题吗？

梁斌：国外多少也有一些，但国外10年以上靠纯编程为生的人，比国内要多得多。因为作为Coder，他们的价值是被社会认可的。管理层和编码人员待遇差距没有那么大。编码的人买得起房，买得起车，可能买不起游艇，但他热爱编码，不在乎这个。

但如果管理层房车都有了，编码的人还要租房子，内心就很难平衡了。就算自己安得其乐，家里人也会有意见。如果社会认可我们的价值，如果编码这件事能带来足够的荣耀，还会有这么多人转行吗？

说远一点，为什么中国互联网界加班这么多，干活那么快？一个原因是中国互联网的钱都是美国人出的。在中国，创业公司想要融资，走银行这条路基本走不通，90%以上都是从美国融资，那么投资方就会对你有很高的要求，比如一年内用户数要增长到多少。中国互联网业界有个共识，代码不用写很好，因为万一企业黄了，代码写得再好都没有用。而请一个老码农的钱可以请好几个新手，在这种情况下，初创阶段的公司为了快，就会请一些一般的人，等公司火起来之后，才会去招牛人。这种浮躁的心理让整个行业认为程序员的价值只在干活“快糙猛”。

图灵社区：作为这个群的群主，你现在是想去做管理还是想去编程，想去大公司还是小公司？

梁斌：我自己摸索出一条路——“个体户技术输出模式”，就是我自己或者带小团队做一份技术，输出给多个客户。可能从每个公司取得的收入不是特别多，但足够养活我和团队，这样我们可以专心做我们想做的技术，让自己的技术生命继续往下走。为了保持独立性，我可能终身都不会去某一家公司，而且我觉得我有责任摸索一条能够延续技术生命的路，让码农们看到，继续编码是有路走的。

图灵社区：这个模式的可行性在哪里？

梁斌：这个模式通过做别人做不到的事来争取客户。因为在中国，通过高质量代码来赚钱是非常难的。举个例子，同样是一千万行乘一百亿列的矩阵运算，如果你我都在两个小时内运算完，那么在客户眼里，我们就是一样的，虽然请个行家来看，可能我的代码质量优于你10倍，但我绝对没可能拿到十倍的钱。只有当你能做一百行乘一千维的小矩阵，而我却能完成更大的矩阵运算时，才会有人买我的帐。

图灵社区：图灵即将发行一些电子书，对此您有什么看法？

梁斌：据我所知，现在很多人看不进书，原因在于书太难。我认识的人中，还没有一位能看完Knuth的《计算机程序设计艺术》，当然这套书很深，但其他书能看懂的人也不多。所以仅仅开放电子书是不够的，这一步只是从0走到了1，你们应该找到更多的方法，帮助读者真正学会。

图灵社区：码农群有一个主题，是“让码农Social起来”，这个想法是怎么来的呢？

梁斌：我认识的很多码农都比较“宅”，跟机器待时间长了，有些人当众发言都很困难，社交能力非常差。想想看，10年码农，社交能力还差，长得又不怎样，怎么办啊！一个方法就是社交起来，让更多人知道你的价值。群里的人都是老码农了，彼此间进行沟通是很容易的，Social起来之后，不同背景的人彼此了解，就能够迸发出更多火花。

码农需要Social，说到底也是出于安全感。我们面临人生、家庭、职业、社会各种困惑，不Social就容易变得愤青、偏激、爱

吵架。这个群里的人有着非常多的共同点，Social起来更容易感受到编码的快乐和荣誉感。

图灵社区：确实，社交起来彼此都能有所收获。《码农》电子刊的部分灵感也是来源于此，好多人说这个名字是自毁长城，您怎么看？

梁斌：为什么我们不愿意管自己叫“程序员”？这个听起来光彩多了，但是社会不认可。他们不觉得程序员有多大的价值，索性我们就自贬一分，叫自己“码农”。如果我们像国外程序员那样被尊重，又何苦这么无聊？现在很多公司要么是产品经理驱动，要么是销售驱动。程序员如果永远出于被驱动的状态，就会逐渐丧失主动性。

其实程序员并不要求多大的权利，只要有正常的发言渠道，能够表达自己的意见就行了。我觉得你们做好一件事就行——就是给那些默默无闻的码农一个发声的渠道，让他们有荣誉感。我们这些人，不是什么经理，也非各种总监，更没有身负xx长的光环，不能忽悠广告，也输出不了什么趋势，顶多是个高级程序员，就是在第一线干活的人，在你什么都不能给他的时候，荣誉感非常重要。

图灵社区：老码农自身是否也存在一些问题，比如排斥新技术，不开放，创新力不足等，使得职业上的不安全感越来越高？

梁斌：确实是，现在让我学一门新技术，我也不太愿意，也因此放弃了很多的机会。因为白纸一张的时候你可以随便选择，但厚厚的油画就再难涂抹。现在技术淘汰速度特别快，有时不懂新技术都不好意思跟人打招呼，所以一些必备的东西还是要学习的。

记者 / @[谢工](#)

@[杨帆](#)

但老码农们肯定会走到这么一天——无法再学习新东西，所以我现在想尝试类似“学徒”这样的制度，老码农虽然学习能力不足，但眼界在，可以让年轻人去学习，我们给予一些指导，通过这种方式把人员架构起来。

图灵社区：谢谢梁斌接受我们的采访，希望有一天，我们能够消灭“码农”这个称号，大家都为自己的职业感到自豪，那时《码农》也可以改名叫《编程人生》，那就再好不过了。 ■

Matrix67的Aha! Moment



Matrix67何许人也？如果说一个人写数学博客写了七年不能让你吃惊，再告诉你他是中文系的，不知道你会不会觉得诡异？这位年轻的数学爱好者即将在图灵出版《思考的乐趣》一书，图灵社区借此机会采访了他，他真是一个特别“好”玩的人，“好”字的两种读法说的都是他。而且跟他接触之后，你会觉得祖国的未来特别有希望。为什么呢？请看下文。

应用语言学是什么？

图灵社区：我们先从你的[博客](#)开始谈吧，现在很多人都在看你的博客。

顾森：这个博客是从2005年7月开始写的，其实本来完全是写给自己看的。我把它当做自己的学习笔记，学了新东西，就放在上面，方便以后查找。后来发现有人开始看我的博客，就多了一份与别人分享的心，开始尝试着尽可能地把东西讲明白，而不是只给自己看了。

博客读者多了，偶尔会遇到一些很神奇的事。记得大一的时候要上《数据结构与算法》，我算法很好，所以完全没有去听过课。最后期末考试时，就两手空空地只拿一支笔去，结果发现所有人都带着书呢，然后才知道原来这门课的期末考试是开卷考试。最神奇的是，老师一进来，我才发现我俩其实本来就认识——他居然是我的网友……于是那个课我得了一百分。

图灵社区：你读中文系是吧，怎么要修这个课呢？IT圈里很多人都知道你，但大家又把你定义成一个北大就读的文科生，像你这么喜欢数学的人怎么会学中文？

顾森：准确地说，其实我学的是应用语言学，是专门做中文信息自动处理的。这是一个理科专业，而且跟数学关系特别大，是语言学、数学、计算机的结合。我们跟着数学学院去学数理逻辑，跟着计算机专业去学数据库、算法，感觉上很不像中文系。但我们又要跟中文系的人去学语言学、古代文学史之类的，后者尤其让我头疼。《古代文学史》是我唯一知道的在北大要学两年的课，四个学期里我的分基本上是逐年下降的。一直到《古代文学史（四）》，我实在不想学了，那个课就一次没去，最后背了两天，考了60分，及格拉倒。我希望把时间花在我真正想学的东西上。

图灵社区：其实这个专业本来就不算是中文系吧。

顾森：对。在中国，中文系的设置一直是有问题的。中文系的学生一方面在学文学，另一方面在学语言学，但在国外，这完全是两回事。我们现在学的就是偏语言学的。

事实上，我还挺喜欢语言学的，我最喜欢举的例子是汉语语言学家陆俭明先生提出来的，他说“看报纸”可以说“把报纸看看”，但是“借报纸”不能说“把报纸借借”；“洗衣服”可以说“把衣服洗洗”，但是“买衣服”不能说“把衣服买买”。这就提出了问题，“看报纸”、“洗衣服”和“借报纸”、“买衣服”这两组动作到底差别在哪儿，以至于一个能换着说，一个不能换着说？一个更诡异的例子则是，“收作业”可以说“把作业收收”，“交作业”却不能说“把作业交交”。这个问题其实现在还没有一个完美的解释。语言学里的

很多内容就是去找规律、建模型、提反例，整个就是一个理科的氛围。我的书里有一节是讲中文分词的，就是从学校里的相关课程中学到的。

现在我在人人网实习，主要做汉语文本分析，也接触算法，还挺对我胃口的。因为我本来也是信息学竞赛保送的北大，对算法特别熟悉。《思考的乐趣》这本书里也写了很多跟计算机相关的东西，比如有一节就是专门讲协议的。这个是最早跟云风聊到的，他先跟我讲了一个关于协议的问题，然后推荐了《应用密码学》那本书，我买了之后疯狂地看，收获特别大。我把从《应用密码学》和从他那儿了解到的东西，加上我自己想到的，整理成一篇文章放进了《思考的乐趣》里，特别好玩。

我常常用我自己的专业知识做一些有趣的事情。上个月，我找来了一份双字词表，然后从中找到了一些两个声母颠倒过来正好也能说的词，然后用它们留空造句，出成一系列题让大家去填，比如：

虽然公司位于一块（ ），但是最后还是（ ）了。

答案：虽然公司位于一块宝地，但是最后还是倒闭了。

宝地和倒闭

魔术师熟练地从（ ）里变出来一只（ ）。

答案：魔术师熟练地从台布里变出来一只白兔。

台布和白兔

他知道好几种（ ）翻新机的（ ）方法。

答案：他知道好几种鉴别翻新机的便捷方法。

鉴别和便捷

为了磨炼意志，他常常赤身睡在（ ）铁钉的（ ）上。

答案：为了磨炼意志，他常常赤身睡在布满铁钉的木板上。

布满和木板。

我特别喜欢干这样的事，这就是所谓的交叉学科吧。

图灵社区：这个学科有哪些实际应用呢？听起来应该有很多机会。

顾森：我在《思考的乐趣》第一章里写到了分词，那其实就是一个最基本的应用。当然，分词之后还有对句法的结构分析、语义的识别，以及用统计的手段去挖掘一些东西，例如从大规模的真实语料里面统计出大家喜欢说什么，不喜欢说什么之类的。

Facebook之前就做过一些很有意思的东西，他们从语言文本中抽出了表示心情好的积极的词，和表示心情不好的消极的词，然后通过两类词出现的比例来看整个美国人口大致的幸福程度，结果表明人们的幸福程度整体呈现上升趋势。这些东西都挺好玩的。

图灵社区：你让我们想起《编程大师访谈录》里面的一个人——Scott Kim，他就是专门在美国的各种杂志或者游戏中设置圈套和谜题的人。

顾森：对啊，我以后挺想做那种工作的，类似游戏关卡设计师。我特别喜欢给人家出题。

好老师是什么样的？

图灵社区：听说你读书期间还休学了一年，为什么？

顾森：不想读书了。虽然我很喜欢这个专业，但有些课真的不喜欢，会浪费我很多时间。比如《古代汉语》，老师规定考试答题写繁体字可以加分，那时候就有很多人练繁体字。但最后发现，这些人最后考试分数都特别低，因为繁体字要想写对是很难的，老师把他们写错的字都当错字扣分了。

虽然很多中文系的必修课我们不用修，比如《古代典籍概要》等背诵内容特别多的课程，但是最经典的文学史课程还是必须要学的。本来我觉得能混过去就混过去，后来实在不想混了，就想出去混了。另外，大三上学期结束后，我确实非常忙，不但在数学培训机构专职工作，还接下了好多约稿，于是就休学了。休学之后没多久，有幸结识了果壳网的一帮人，就在那里工作了大概半年到一年。

我觉得，休学那一年收获特别大。

图灵社区：你在果壳网做了哪些事？

顾森：当时我是做数学分站的编辑。数学分站名字取得挺好，叫“死理性派”。我把竞赛时认识的一群好友都拉过来，跟我一起写文章、发文章，为数学科普做了很多工作。

图灵社区：现在忙什么呢？

顾森：简直像精神分裂一样，周一到周三在北大继续念书，周四周五在人人网实习，周六周日还要去外地讲课，做中学数学培

训。因为北京这边升学压力特别大，进度比课本赶早一年，而且如果家长发现老师讲到课本之外去了，没有老老实实讲题，老师会面临被投诉的危险。所以当我知道外地有分校的时候，就想到外地去看一看。在外地讲课特别累，每天上午、下午、晚上各讲3个小时的课。如果是寒暑假，可能会连续一个多月都是每天9小时的课。

图灵社区：你现在教的学生是什么样的？

顾森：主要是初中生和高中生。这个课的定位是普通的数学培训，帮助学生应对中考、高考或者竞赛。但我走得比这个更远一些。我希望能够把数学之美、思考的乐趣，甚至把生活中有趣的点点滴滴都教给学生。这些对他们可能没有即刻的用途，但对将来是很有帮助的。以后学生可能会发现，这个顾老师讲过，那个顾老师讲过。

图灵社区：学生也能接受这样的教育方法吗？

顾森：对。而且我觉得作为一个教师，一方面是要把学生的思维带活，让他们不要陷到应试教育里面，另一方面最好也能改变学生的性格。他们的压力都比较大，我希望他们至少能在我的课堂上轻松一些，活跃一些。比如，我绝对禁止课前走“同学们好”、“老师好”的形式；也绝对禁止学生举手发言，要发言直接说就行了；当然也绝对禁止站起来跟老师讲话，直接在座位上说就可以了。他们可以喝水，可以吃东西，只要吃的东西味道别太大，别嚼得嘎嘣嘎嘣的，别馋到其他同学就行了。我的课堂就是完全开放的，觉得怎样舒适就怎样学，唯一的要求就是跟着老师的思路走，让思维一直转，保证每一节课里，至少能够体会到一

两次思考的乐趣。后来，这些班里的孩子们思维比较开阔了，性格也比较好了。

我教课教了三年，初一到初三都讲过，所以今后基本不用备课了。但每年我自己都会学到一些新的东西，加进我的教学计划里。而且，我也在不停地积累一些初中教育的方法。我教过一些小孩，初一刚进来的时候，特别不爱思考或者不喜欢数学，性格也不太好，不愿意跟老师、同学交流；初三之后，整个变了。其他老师说我那个班简直就是“流氓班”。但家长没有投诉，家长挺喜欢我的。

图灵社区：你的学生应该也很喜欢你。

顾森：对。上课时经常扯淡，学生们爱听。家长也挺高兴，因为扯淡也扯得跟数学相关。老师讲正课，学生没有几个愿意听的，最好的方法就是扯淡，扯一些表面上和课本没有关系的东西，大家都愿意听，然后把数学的知识放到里面去。学生能够听到好玩的故事，还能把该学的东西学下来，自己比较感兴趣的，还能跟别人去讲，这就最好了。我常讲的，要么是《思考的乐趣》第一章那种比较好玩的故事和应用，要么是后面那些好玩的证明，用我的话来说，就是“可以骗小女生用”。真有一些同学一拍大腿说：“这个题太妙了，骗小女生太管用了。”把小孩教活，我觉得非常重要。

要做周游全国的Geek

图灵社区：讲了这么久的课，收获如何？你明年就毕业了，有什么职业规划吗？

顾森：虽然累，但是收获挺大。再讲个三年，把一届初一学生带到初中毕业，到时候，攒的东西又够写一本书了。

而且工作的时间集中在周六周日两天的话，其他的时间就多了。等明年毕业之后，从周一到周五我都闲下来了。我想到处去实习。在人人网玩一圈之后，我可能会去豆瓣、百度等等，每一个单位可能都去实习一下，甚至到我不熟悉的行业去实习，这种感觉挺好的。

图灵社区：有什么长远打算吗？

顾森：没有长远打算，我实习的目的就是想去接触各种各样的东西，可能IT这个圈子混完了之后，会混其他的一些圈子。我有一些特别远的理想，也可以说是不靠谱的理想吧。从北京出发，去全国各个地方，二线城市、三线城市都跑一圈，用一个理科生的眼光，或者说一个Geek的眼光去看，去寻找，去发现。大概花个几年、十几年，把整个中国游一圈，然后再把沿途所见所闻所想汇总起来，如果能够写成一本书，就太棒了。

我的双休日，就是别人的工作日。每周一、二、三休息，这样到哪儿去都没有人跟我抢，可以到处去玩。

图灵社区：那你跟别人不是没有交集了？以后跟家人在一起，时间都岔开了。

顾森：如果说另一半，我希望能够找一个跟我一样的人。我特别讨厌有个固定的工作，我更希望每一个地方都去混一下。可能在每个地方都是做最初级的工作，但却能够体会一下公司文化，体会一下这个行业的新鲜事，目的也就达到了。

图灵社区：你想过创业吗？

顾森：没有想过创业，因为创业其实事儿也挺多的，还不如看看东西，写写东西。

图灵社区：在科普方面还有什么打算？

顾森：首先是要提高自己。我觉得我读的书太少了，尤其是考虑到我不算是一个真正学数学的人。所以毕业之后，我最想做的就是用自己的方法重新去学一次数学，然后用自己的方法再讲一次数学。

图灵社区：面对什么样的对象去讲呢？

顾森：现在可能是面对一些数学爱好者，甚至是一些本来不喜欢数学的人。但是以后，当我能够系统地去讲一些真正的数学思维、数学方法的时候，有可能是针对一些数学专业的学生。因为我很想改变中国的数学教育现状。

举个例子，现在的课本也好，教辅也好，考试也好，从来就是让你证明一个命题，但从来没有让你推翻一个命题，也没有一道题是让你先猜猜命题是对的还是错的，然后给你认为正确的结论找到一个证明方法，给你认为错误的结论找出一个能推翻它的反例。在数学研究中，你首先要有一个直觉，能够去判断一个问题值得不值得研究，一个命题到底是对还是错，然后才是去证明或者去找反例。如果有一天我能够改变数学教育，我肯定会培养学生做这类问题。现在的数学教育就纯粹是让学生做题，很少有学生能够领会到数学研究的乐趣。

Aha! Moment

图灵社区：写这本书的想法是怎么产生的呢？

顾森：我曾经在网上看到热心的网友，把我的博客整个扒下来，按时间从前到后整理起来。一方面我觉得很高兴，另一方面又觉得这样帮助其实不是特别大。因为博客上的文章很零散，尤其是最早2006年、2007年写的东西，其实非常不成熟。偶尔看原来写的文章的时候，我就觉得，原来写的东西还可以再总结、提高一下。

我还曾收到很多网友的来信，他们真的把所有的文章全部看了一遍。当时我就想，应该给更多新看到这个博客的人提供一个途径，让他们可以一下子读完这个博客中所有值得看的东西。认识你们之后，想到可以写成一本书。我就想，一定要找到一条线索，把这几年学到的东西有机地串起来，像说话一样，能够很自然地从一个话题跳到另外一个话题，前后都是相关的，一气呵成。

图灵社区：这本书跟你的博客有什么不一样？

顾森：书里添加了好多东西。在写这个博客之前，我就对很多数学话题感兴趣。虽然后来有些文章可能会提到，但并没有专门把它们写下来。比如关于“无穷集合的势”的话题，博客里面很多文章就是建立在这个话题之上的。我一直在想，既然经常提到这些，就应该找一个时间把它们系统地写下来。这次借写书的机会，我把最想写的几个比较大的话题都写出来了。

另外，这本书里我最喜欢的部分是几个“小合集”。它们其实是几年来分散在各个时间、各个地方的内容，现在用统一的语言把它们串在了一起。它们真的都是最好玩的东西。

回顾目录，我觉得特别满意，因为我真的把这几年来最想和大家分享的东​​西，写在这一本书里了。

图灵社区：所以短时间内也没有第二本书可以出来了？

顾森：要看我能不能找到其他的话题。这本书面太广了。如果一个人想来看我的博客，这本书就是我最想让他看的文章，并且以最想让他看的方式来呈现，里面还包括我个人的一些经历，比如我是在什么情况下学到这些的。

有一些话题确实太大了，一两篇文章是讲不完的。要是有机会，我想专注在其中的一两个点展开来讲，就像数学花园的一角。我挺喜欢你们那本 [《图灵的秘密》](#)，那本书就是这样做的。它从图灵的角度，把很大的一个话题从头到尾串起来。我之前就对图灵特别感兴趣，这本书里的内容正好属于我不知道，但稍微一看就能看明白的那种。这本书真的特别好，它应该是我这段时间看过的最好的科普书。

图灵社区：为什么你这本书不叫《数学的乐趣》，而叫《思考的乐趣》？

顾森：数学的话，我觉得还挺大的，会不会有人感觉比较畏惧？而“思考的乐趣”就比较浅了。而且这本书里很多内容也不算是和数字相关的，而是和思维相关的。比如说第一章里讲到统计，很多时候统计数据上显示两个东西相关，但不一定有因果关系。我

顾森，重庆人，数学狂，喜欢教学生数学魔术骗小女生，喜欢混各种圈子，讨厌固定工作。

常举的例子就是：去救火的消防员越多，火灾损失越严重。大家估计会想，这怎么可能？其实这里的因果关系是颠倒的。正因为损失越严重的，所以去救火的人才越多。这些其实和数学没有什么关系，而是一种思维方式。看到这个，你能恍然大悟，然后会心一笑，这就算是体会到思考的乐趣了。

图灵社区：你觉得什么样的人 would 喜欢《思考的乐趣》？

顾森：我觉得对数学感兴趣的人，不管是学数学的，还是普通的数学爱好者，都会喜欢。这本书里会有一些比较冷门的东西，可能数学专业的人也很难接触到。本来我以为像我这样纯粹因为兴趣而去学习数学的人很少，后来发现其实挺多的。从很多网友给我写的邮件里，我都发现他们是真的喜欢数学。我觉得特别感动。我应该给这些读者送上一份这样的礼物。我希望读者朋友们在每天晚上睡觉之前，或是在其他空闲的时候，不带任何功利的目的来看这本书，如果真的能有“啊哈，灵机一动”的感觉，目的就达到了。 ■

记者 /
@[杨海玲](#)
@[杨帆](#)
@[何健辉](#)

柳泽大辅谈如何想出好创意



柳泽大辅

DAISUKE YANASAWA

1974年出生于香港。毕业于庆应义塾大学信息学院，后就职于索尼音乐公司。1998年与3位朋友共同创立了KAYAC公司，24岁就任总裁。该公司是一家极具创新能力的网络互动公司，每年推出100多种新服务，销售规模连年增长，在日本受到了广泛关注。著有《飞翔思考法》、《创意不是想出来的》、《Web百发百中》、《这样的公司制度很有用》。

图灵社区：[《创意不是想出来的》](#)这本书里面有许多观点相当有趣，我第一次看的时候总是忍不住频频点头。比如说“其实只要站在人类历史的角度俯瞰下来，就会发现搞砸工作丝毫算不上什么大事”、“愉快工作的人，其实要比‘听话’的人更具有工作上的主动性”。请问一下您当时为什么会想写这样一本书？

柳泽大辅：15年前，我与大学同学共同创立了KAYAC公司。虽然我们共事多年，但难免有意见相左的时候。这个时候我们都遵循一个原则：不采用任何一方的意见，双方重新考虑方案，直到互相认可。这种工作方式使我们都形成了一个好习惯：与其去批判对方，不如想出更精彩的点子去获得大家的认可。

图灵社区：您在本书中提到，出点子其实很简单，有量的积累才能有质的飞跃。热爱工作、保持积极乐观的生活态度也不难。但这和我们实际的感受却不太一样，比如有些人并不喜欢

自己的工作，或者有些人即使刚开始喜欢自己的工作，也无法一直保持对工作的热情。对此您是怎么看的？

柳泽大辅：我个人认为工作是一件有趣的事，但肯定也会有人认为“工作是一种煎熬”。在这里我给大家出个主意。你可以这么想：“如果不工作靠什么生存呢？”从这个角度去考虑，你就会发现你会有许多意想不到的选择。

图灵社区：您有一个观点：无论是谁只要掌握了方法就能提出大量的创意（暂且不论创意的好坏）。这里的方法有比如结果逆推法、曼陀罗思考法、创意公式法，还有贵公司独创的Lonely Idea等等。但从我个人的观察来看，似乎出点子还跟个人的性格有一定关系。比如有些性格外向的人，在头脑风暴的时候可能就会比较活跃。对此您怎么看？

柳泽大辅：有研究表明，如果脑部的某处受损，原本开朗的人会变得忧郁。一个人的创造力也许是与生俱来的。不过如果说“出点子完全是由性格决定的”，那就有点太悲观了。我个人相信，无论是谁都能想出许多创意来。

图灵社区：贵公司有一项“闪光弹”服务，替别人出点子，比如起名字、如何进行演讲、生意怎么做等等，1个点子100日元。当时为什么会想到开展这项业务？

柳泽大辅：我想通过这项服务向大家传达这样的理念：点子多了世界就会变得充满活力。同时，这项业务也是为了我们自己进行思维锻炼之用，并不是正经生意。1个点子100日元只能是倒贴钱.....

图灵社区：在这项业务中，所有问题都能得到解决吗？如果客户觉得不满意，贵公司有什么措施吗？

柳泽大辅：对于这项业务，我们本着一个原则：全力以赴。至于是否真的能想出好点子来，我们自己也不太清楚。不过对于每个单子，我们最少提供11个点子。只要能够以此拓宽客户的视野，我们的努力就没有白费。

图灵社区：在进行头脑风暴时，除了做好分工（推动者、丢人者、积极乐观的点子王）以及遵循三项基本原则（保证创意数量、不要否定别人、听取别人意见）以外，在开始之前还应该做一些准备工作，比如说查阅资料等等。那么是不是需要看很多资料呀？如果是24小时之内就需要的点子，怎么来得及看资料呢？

柳泽大辅：在进行头脑风暴前，并不需要看很多资料。在头脑风暴中迸发出的点子，是比较原始的，或者说是一些“种子”。这些“种子”能否生根发芽（即在现实生活中是否可行）？是否有重复？是否有需求？这些问题都是需要在事后进行详细调查的。不过在进行头脑风暴时可以带PC，对于不断迸发的想法，可以实时地用搜索引擎查询相关资料。

图灵社区：那些负责“闪光弹”服务的员工（目前是11名吧），每天给别人出点子，会不会很累呀？个人觉得头脑风暴是一件非常消耗体力的事情，而且客户提出的要求也都不太一样。

柳泽大辅：正如刚才所说的，这项业务是为锻炼思维之用，并不是正经生意。虽然如此，这项工作也是很辛苦的。有的时候1个单子需要100多个点子，让你连哭的时间都没有。不过，经过这样的锻炼，你的创造力就增强不少。



欢迎阅读[日文版](#)

记者 / @[乐馨](#)

图灵社区：这样的创意产业的前景应该非常乐观。对于此项服务，您未来有什么规划吗？比如说如何让它的销售额或者利润高速增长。

柳泽大辅：目前我们的模式是：客户提出要求，由公司来回应。将来我想让一般的用户也参与进来，让更多人感受到这种创新能力的力量。

图灵社区：我看到贵公司每年都面向中国大学生进行招聘。25日还要召开一个5国语言的企业招聘宣讲会。很少有公司会这么做的。请问贵公司是由于什么考虑？将来是否考虑进军中国市场？

柳泽大辅：近几年，考虑到公司的全球化发展，我们公司开始面向日本国外进行招聘。我个人认为，企业应该是超越国界的一种全球化集团。我希望能将我们公司的理念推广到全世界。在我们公司，有不少员工曾在中国受过教育，而且我们每年都会面向中国进行招聘。这些员工对进军中国市场特别有兴趣，希望能够共同努力。 ■

移动为什么重要？

三年之前，我第一次提出“移动优先”（Mobile First）的思想，引来了一堆怀疑者。如今，现实情形已经让更多的人相信，认真对待移动互联网是多么重要。不过，考虑到仍会有人不太相信，下面我就用一种非常生动的方式来解释一下现今的情形。

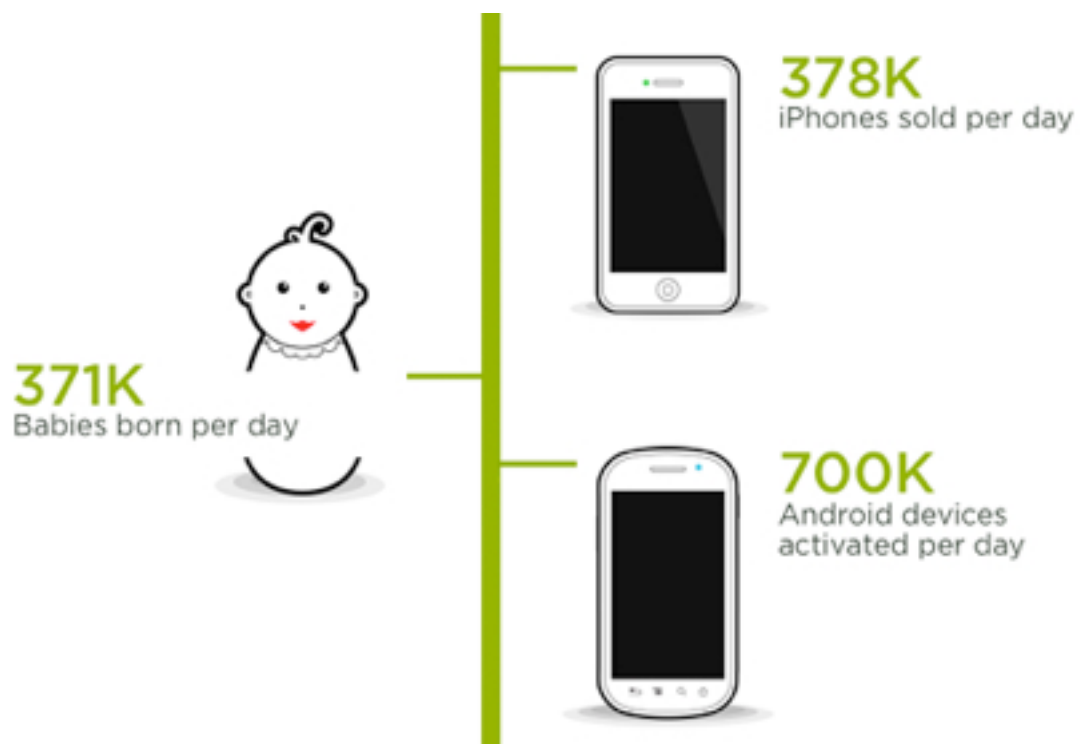
移动设备的数量



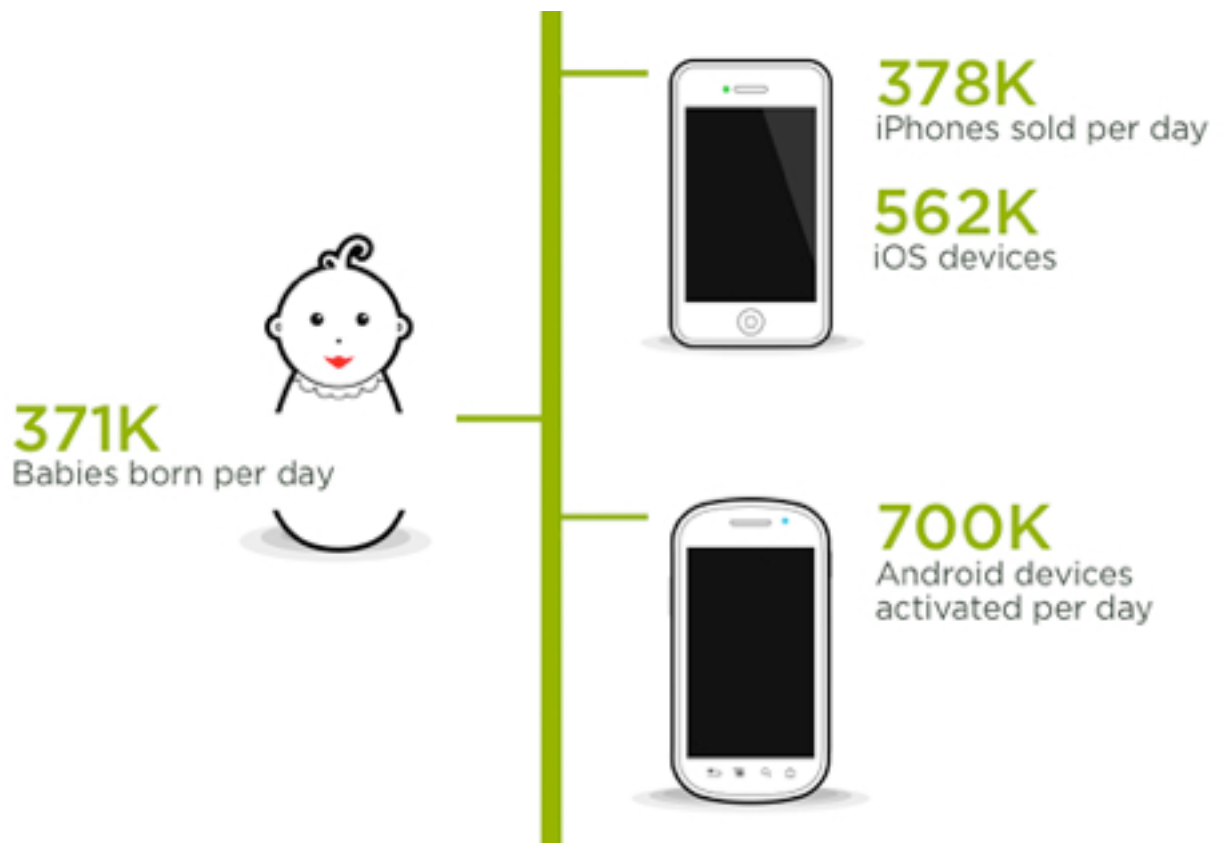
每天，全世界有[371,124](#)名婴儿降生。



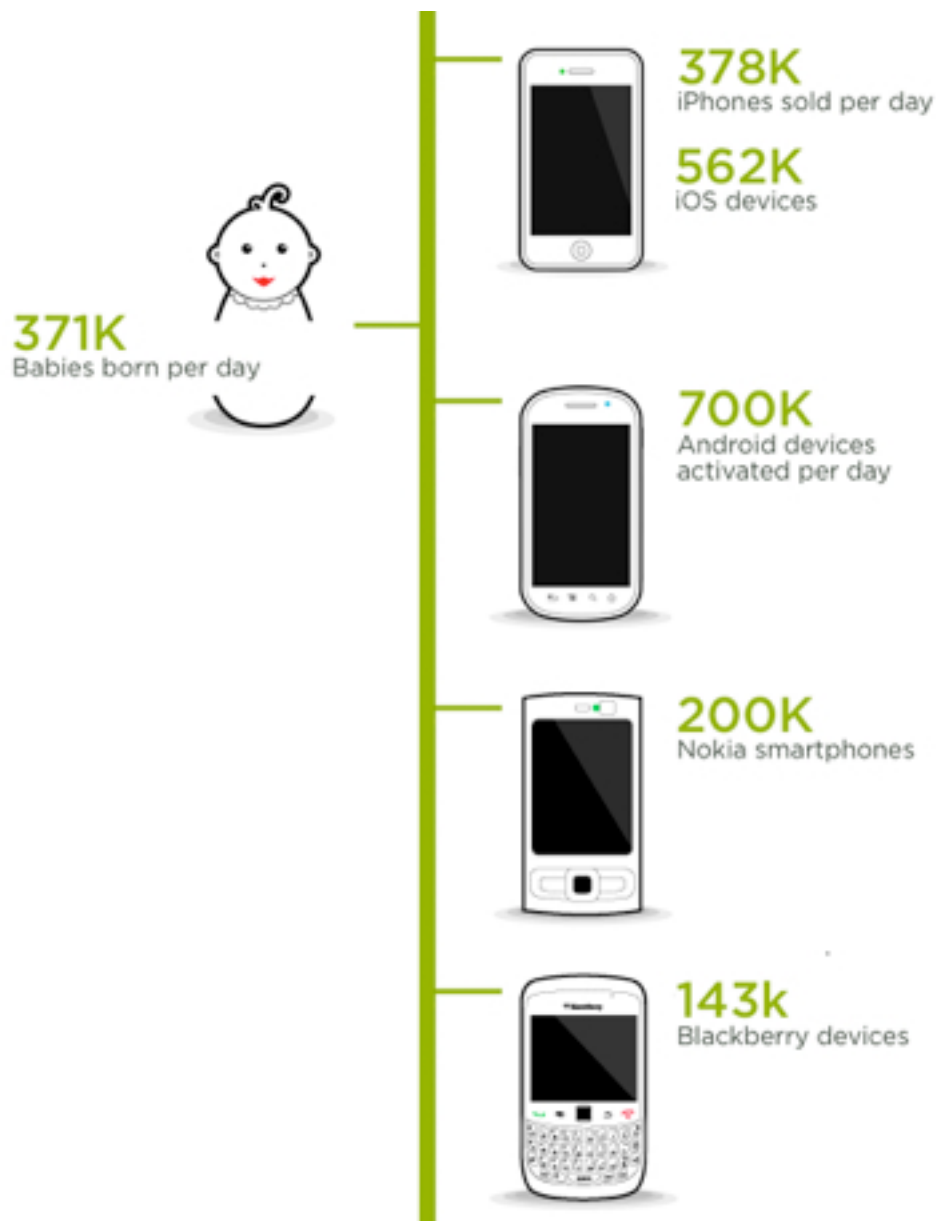
每天，全世界有[377,900](#)台iPhone售出。



每天，全世界有[700,000](#)台Android设备激活。



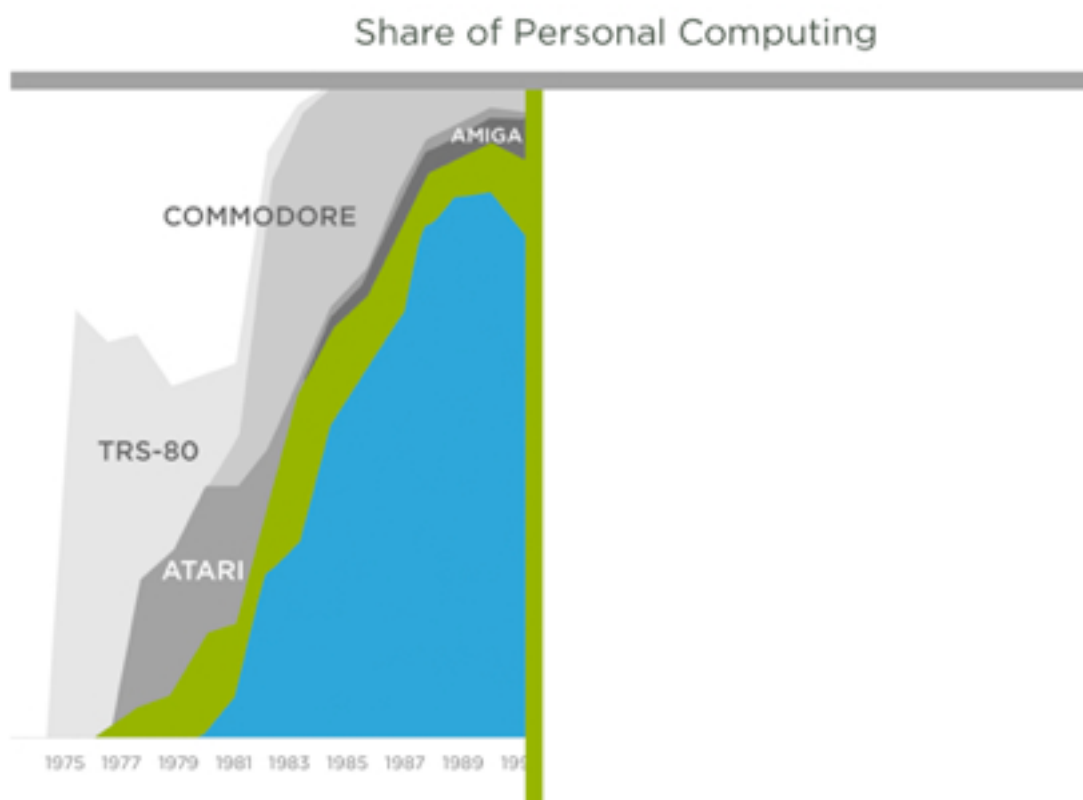
如果看iOS设备的总量（iPhone加iPad和iPod Touch），每天售出的Apple移动设备总数就有562,000台。再加上Android设备，那就是每天售出或激活127万台移动设备，比比看，每天出生的宝宝才371,124名。



然而还不止这些。Nokia每天售出[200,000多台智能手机](#)。到2011年底，RIM每天售出[143,000台黑莓手机](#)。如此算来，每天进入这个世界的智能手机总数就是**145万台**了，再比比看，每天出生的宝宝是371,124（原文错为317,124）名。

个性化计算市场的份额

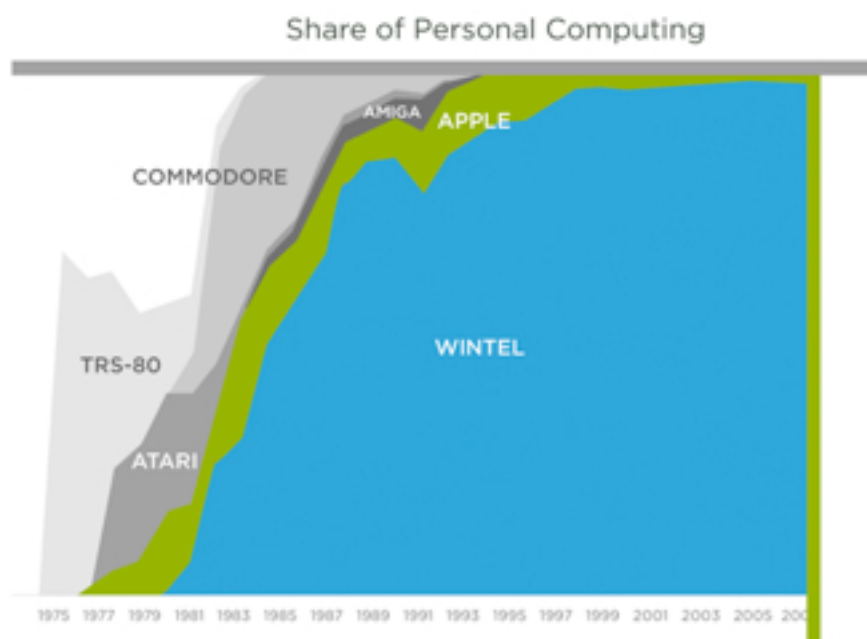
显然，大量移动设备不断来到这个世界。这正给个性化计算市场带来巨大的影响。看一看[来自Asymco的数据](#)，个性化计算最开始的15年里，只有少数生产商（Amiga、Atari、Apple）在探路。



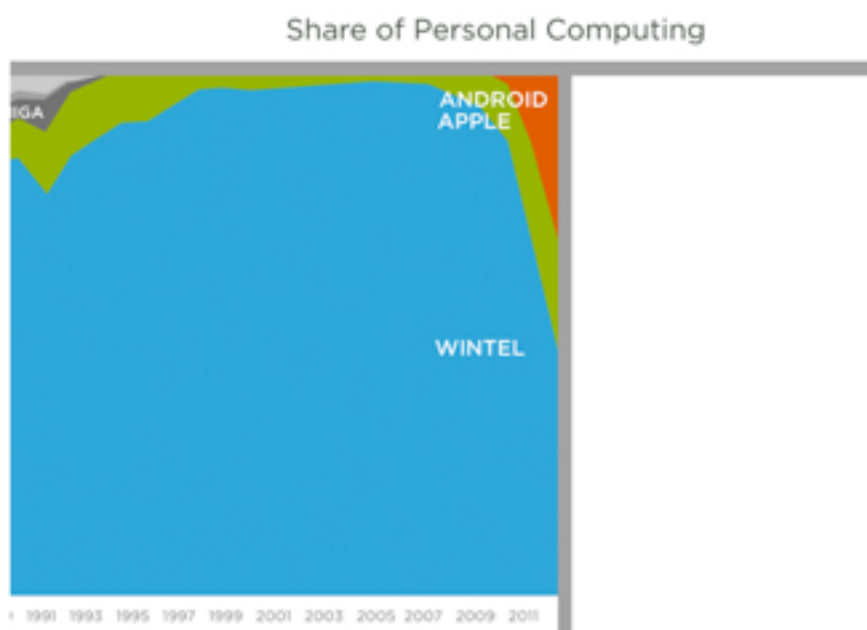
接下来的15年完全被微软的WinTel平台主宰了，除了Apple勉强坚持。



作者 / Luke Wroblewski
[@lukew](#)
[Bagcheck](#)联合创始人及
CPO。Bagcheck在公开
成立九个月后被Twitter
收购。此前，曾是
[Benchmark Capital](#) 的驻
创家（EIR），及Yahoo
首席设计师（Chief
Design Architect
（VP））。



很快到了之前这三年，你可以看到市场完全重新洗牌，至今仍未结束。Apple和Android大量蚕食着个性化计算市场。这是因为今天的移动设备已不只是电话，它是我们所拥有的最具个性化的电脑：随时陪伴我们，一直保持连接，且具备高性能。



移动设备接管了个性化计算，同时为软件公司和软件服务造就了大量机会。想想PayPal的移动支付业务。2009年，移动支付总额为1.41亿美元，到2011年底，这一数字已[增涨至40亿美元](#)。你没有看错，就是在三年时间里从1.41亿涨到40亿。

PayPal™

MOBILE PAYMENTS

\$141M 2009

\$750M 2010

\$4B 2011

eBay的趋势差不多。2011年eBay的成交总量（GMV）达到了[50亿](#)，比2010年20亿的GMV翻了一倍还多。2009年只有[6亿](#)。

eBay

MOBILE GMV

\$600M 2009

\$2B 2010

\$5B 2011

查看英文原文: [Why Mobile Matters](#)

译者 / [晨星](#)
[@steedhorse](#)

单车歌手，浴室票友；红酒葫芦，绿茶水泵；有限版主，无量书虫；职业码农，业余译工（Java语言精粹，软件开发路线路线，敏捷开发的艺术，C#3.0设计模式，代码之美）。

希望这组数据能帮助一些移动怀疑论者看到我们正面临的机遇。如果你还需要更多信心方肯释疑，可以看看我正在推出的数据海报(data post)系列，从而更深入地了解移动市场，以及其它内容。 ■

Android 应用该是个啥样子？



作者 / Juhani Lehtimaeki
[@lehtimaeki](#)
[Snapp TV](#)公司（一家位于英国的创业公司）Google TV和Android开发主管。

对于应用该是个啥样子以及如何运作，Android 平台并没有硬性的准则。谷歌一开始就很明确，他们没有规定什么是可接受的，什么不是。这里有一系列[《UI准则》](#)，但多数只是关注像图标（icons）、部件（widget）和菜单（menu）这样的小玩艺。

Android 平台面世以来，曾出现过成千上万套不同应用 UI 方案，应用的观感非常不统一。现在，随着平台的成熟和应用数量的激增，Android UI 的交汇点正在出现。某些 UI 特性已经成为通用标准，其中的一些甚至已经通过某种方式成为了 SDK 库的一部分。不久，用户可以期待应用以更加统一的方式运作。某些控制和交互模型将成为 Android 整体体验的一部分。

在本文中，我想从某个较高的层次来总结 Android UI 的常见功能。之前我曾写过一些关于 UI 模式的文章，但从未从宏观的角

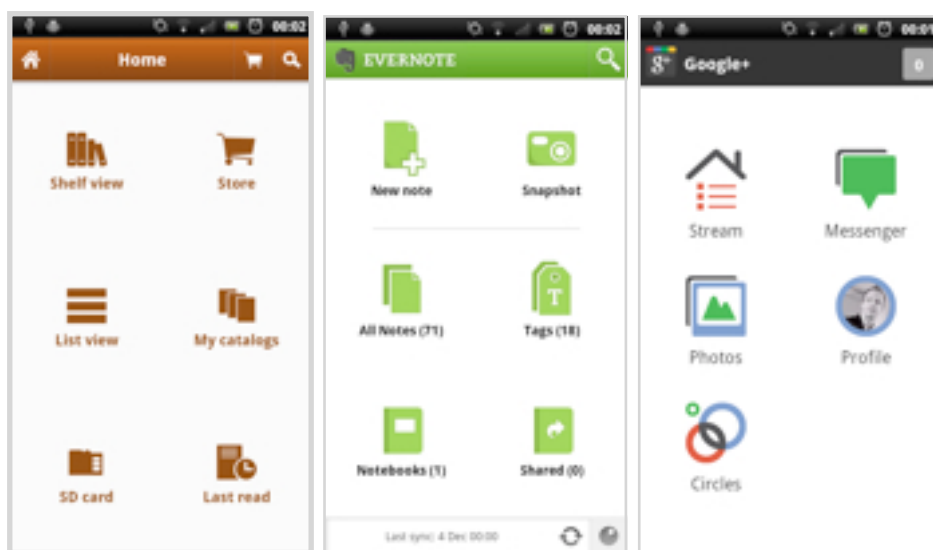
度来看待过这个问题。现在，我想将它们归拢起来，以展现我对 Android 应用看起来应该是啥样子的观点。

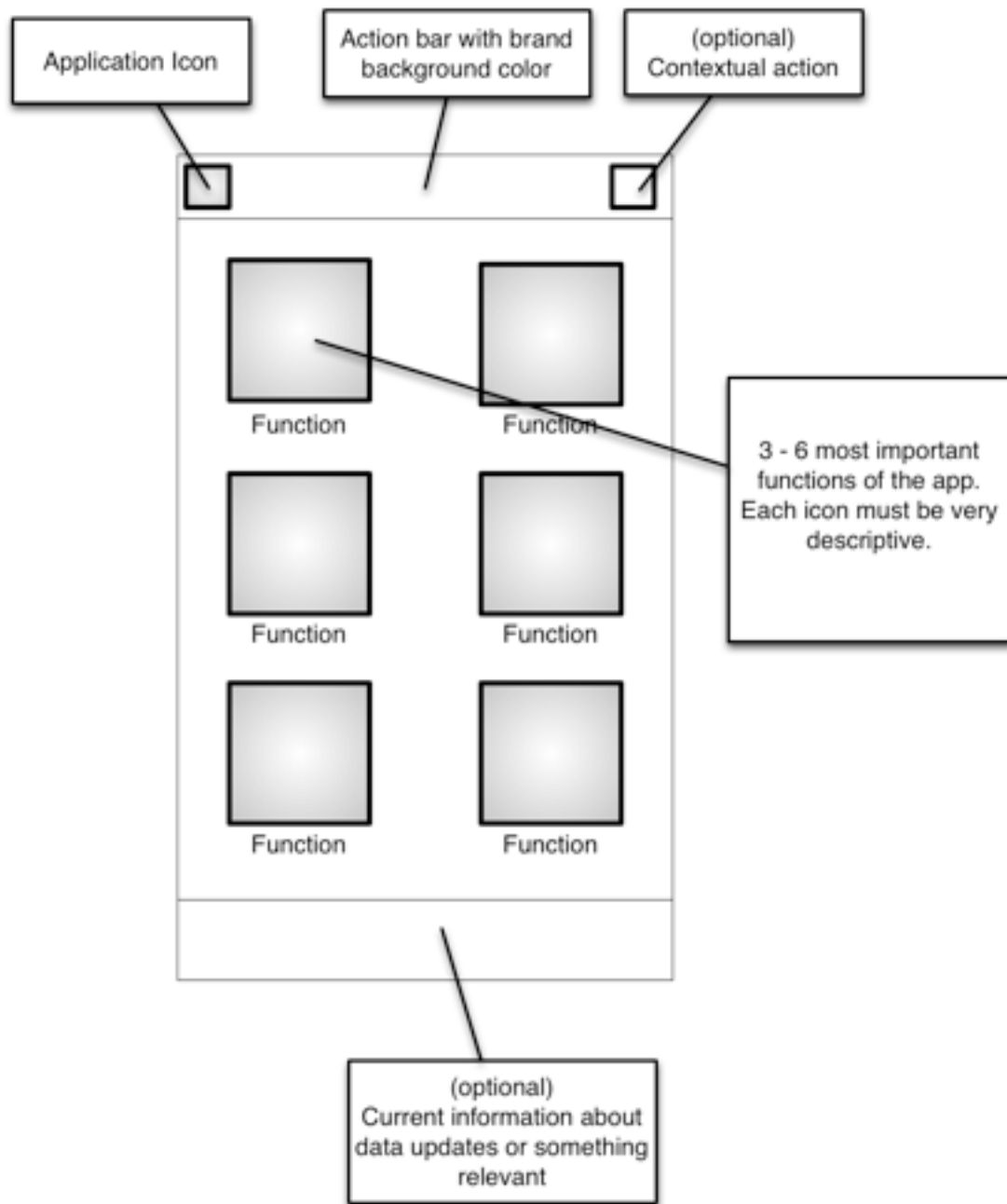
冰淇淋三明治

就在不久之前，最新的 Android 版本（4.0）发布了。该版本中出现了有“史”以来数量最多的用户体验改进。很自然，这些改进将会影响将来 Android 应用的外观。一些改进可以向前移植到旧版本，但是并不是所有的都可以。本文主要讨论当下 Android 应用的外观。我们可以期待很快就看到 ICS UI 的变革，但事实是：市场上有差不多 200,000,000 部 Android 设备运行从 2.1 到 2.3 之间的版本。

应用登录画面

很多应用都使用了仪表盘（Dashboard）UI 模式。如果应用的主功能超过一项，此界面将会是个非常不错的起点。仪表盘展现了应用最重要的功能，并提供了简便的快捷访问方式。



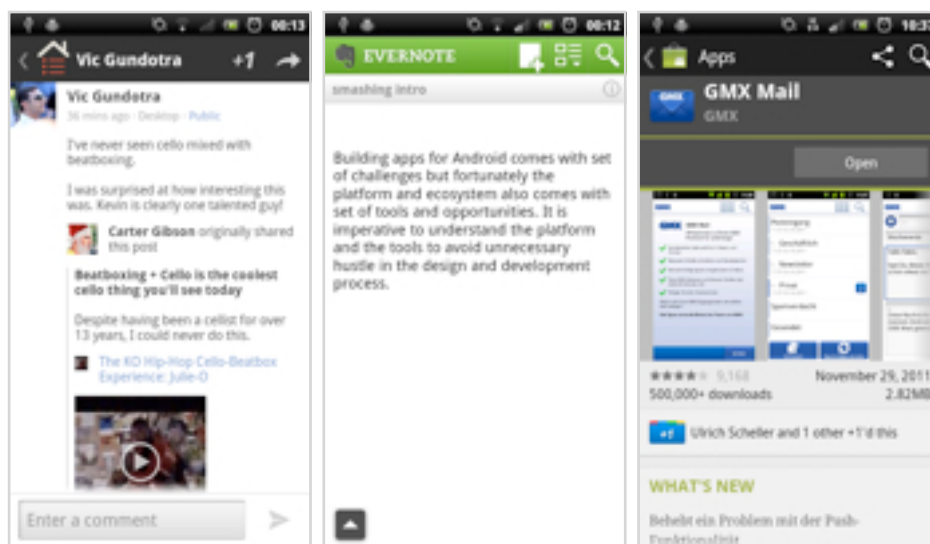


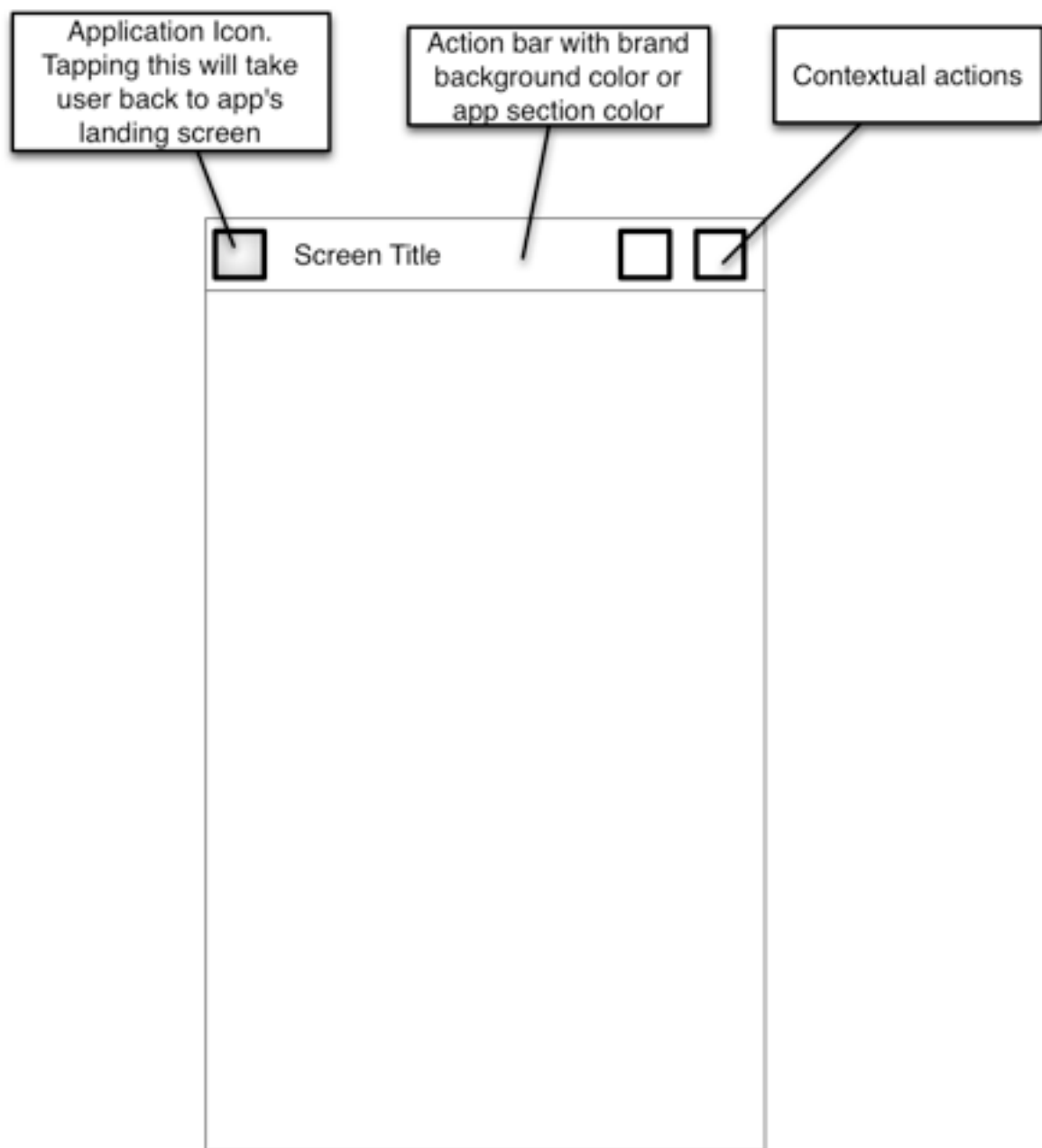
Android 用户都非常熟悉仪表盘。如果运用得当，该方法能够让用户在第一眼看到应用时产生“宾至如归”的感觉。

通用应用程序画面

实际的窗体画面总是以多种形式出现，但只有少数特性变得非常通用，并被用户学习、掌握和期待。屏幕顶端的操作条（Action Bar）非常通用而且易于把握概念。

- 左上角扮演应用图标或者Home图标的角色。点击操作即可把用户带到应用的首页。新的操作条将用户带到上一级画面，而不是首页，这种做法是毫无意义的。
- 操作条的中间将会扮演提供画面标题的角色，同时显示应用的品牌颜色，或当前应用子功能的颜色。
- 屏幕的右上角会放置代表当前画面的最重要的、可执行功能的图标。这部分应当只有与当前画面内容相关的动作。但搜索功能已成为一个常见的例外。





请查阅 Jake Wharton 的 [ActionBarSherlock 库项目](#) 了解操作条的实现。

列表画面

列表是 Android UI 中最常见的部件之一。在展示数据时，如果不知道数据量将会有多大，它将会很有用。

当然，列表也有自己的弊端。为较好地展示列表内容的全貌，每个条目应相对较小。但另一方面，如果把大量信息塞到狭小的区域内，可能会导致用户难以使用列表及查阅要交互的条目。

关于如何使用列表，如果有些准则将会是件好事。用户已经习惯于某些特定的元素和功能，如果列表遵循了某个类似的方法，用户的学习曲线将会更为平缓。

列表画面的操作条

列表画面可使用操作条来展示整个列表所指向的目标操作。请注意，列表画面的操作条不应当是用户对某个（或多个但不是全部）列表条目执行操作的位置。

列表项和勾选框

通常，列表项自身仅包含很少的文本和图片元素。常见的情况是，每个列表项都有一个勾选框，用于选择一个或多个列表项来执行操作。

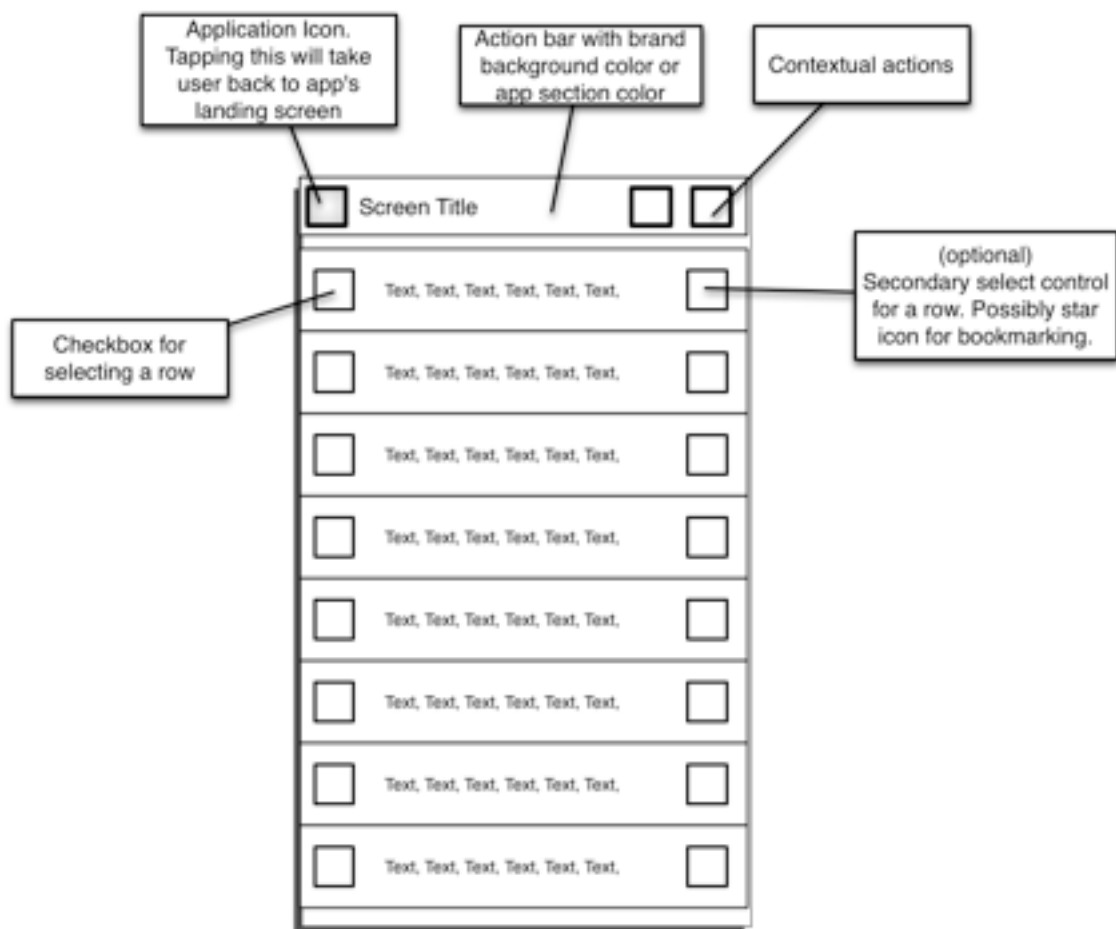
将勾选框放在列表项的最左端有以下几个好处：

1. 我们习惯于在所选项目的左边看到勾选框。无论网页、桌面程序用户界面，还是在移动设备的某个地方，都是这样。

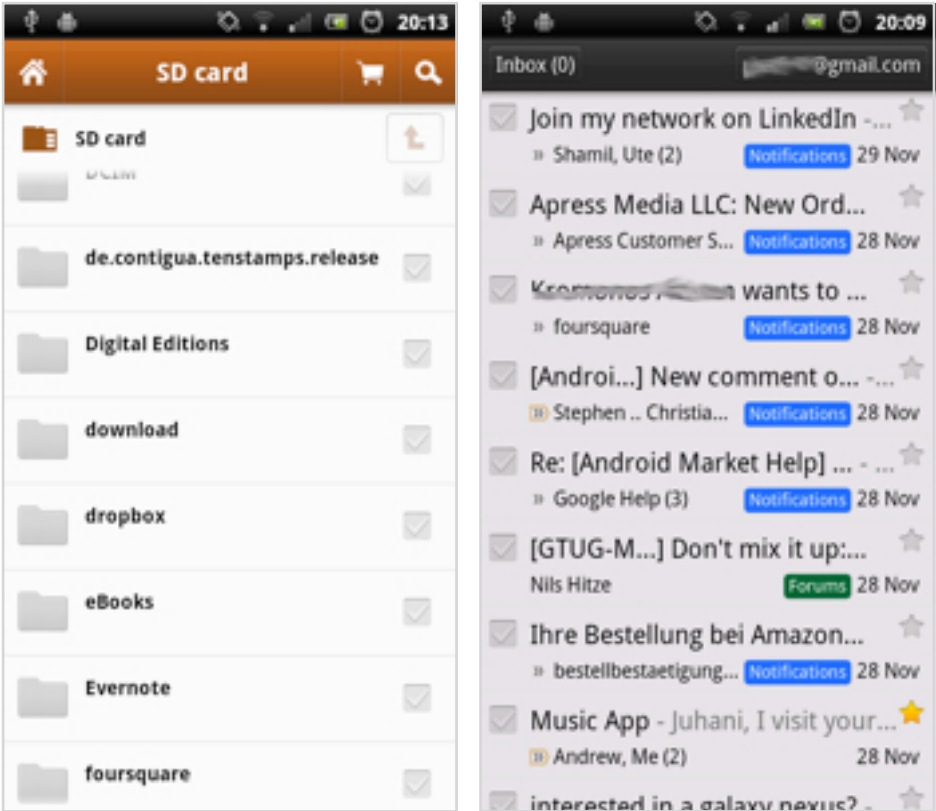
2. 在列表项的边上放置一个勾选项，允许我们给勾选框建立更大的点击区域，由此能让用户更轻松地区分是对某个列表条目进行点击还是选择。
3. 条目左边的图形部件创建了便捷的可视化提示，它会告诉用户某个条目结束，而另一个条目开始，使用户对列表的快速扫描更加轻松。

次要条目控制

一些条目需要更多的交互可能性，而不仅仅是选择（勾选框）或者导航（点击）。这种控制方式最常见的例子是对行条目加注星号或者添加书签。次要控制的唯一正常位置是条目的最右端。放在其它任何位置都会有点击区域方面的问题。

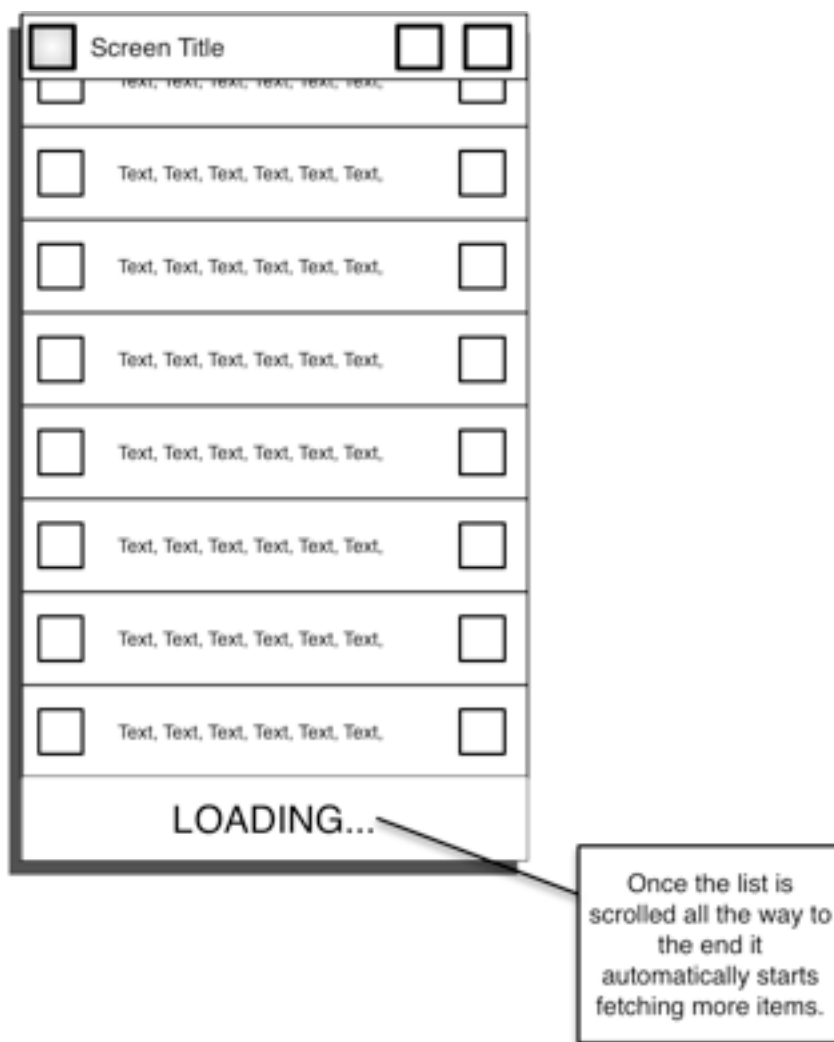


Aldiko 和 GMail 是恰当运用列表的出色范例。Aldiko 选择将勾选框放在右边，因为它们有可见的文件夹图标，如果再在其后放置勾选框，将使UI失去平衡。



永无止境的列表

很多列表中包含的项目都要通过网络来加载。加载过程可能会花上一些时间，生成新的列表项的速度可能会赶不上用户滚动操作的速度。当用户滚动到列表的底部时，应用应当自动开始获取更多列表项。指示器（Indicator）则告诉用户，有更多正在加载的条目将会出现在列表的尾部。其中，使用进度条这样的加载动画是个不错的想法。用户会把动画和正在执行中的任务关联起来。



到达列表的底部时，Android Market 和 Twitter 都会自动加载更多列表项。



单行操作——长按——快速操作

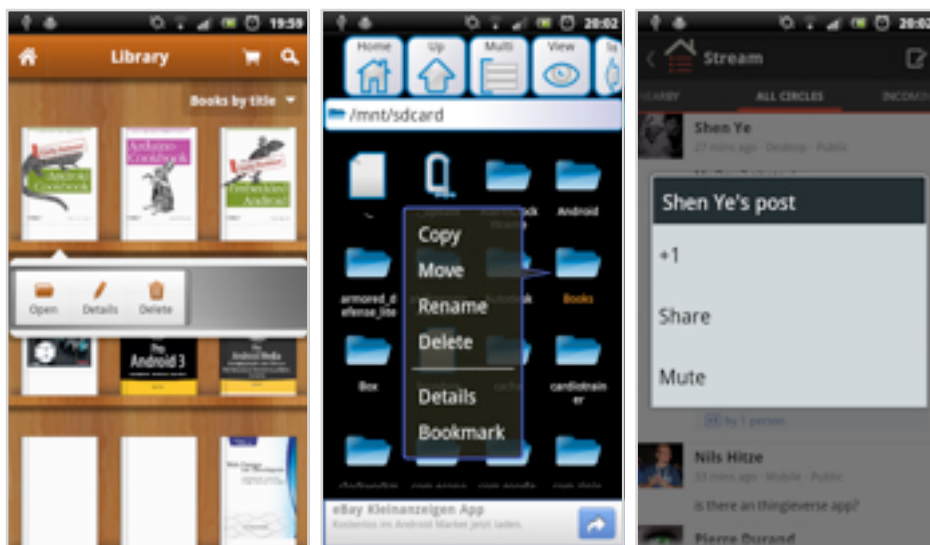
给用户提供一种无须先进入条目画面，即可对单行条目执行操作的方法。

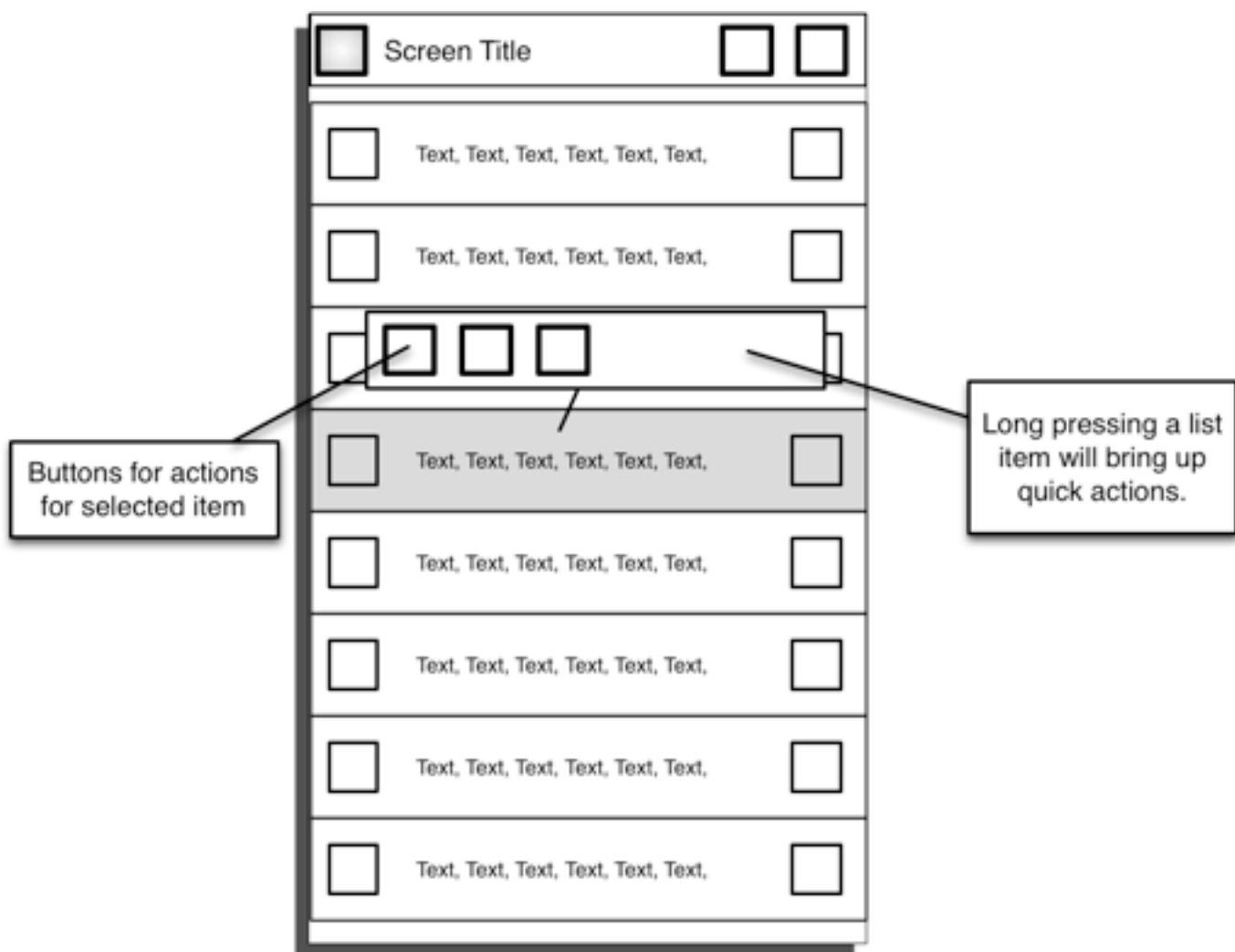
由于电话和平板设备没有鼠标右键（实际上也没有左键），触摸屏特定的“右键点击”由此产生。通过长按某个条目，用户表明他们想要对该特定条目执行某个操作。

有种“快速操作”用户界面模式，可用于显示某个条目对应的操作。最初所使用的图形方法，现在看来几乎都已经绝迹，但其概念犹存。通常是某种形式的覆盖菜单，显示一份非常简单的操作列表，一般只有 3 到 5 项。无论快速操作是如何实现的，有几点必须谨记：

- 不要覆盖选中的条目！特别是在删除操作时。如果能够一直看到要操作的目标条目，用户的信心将会倍增。
- 只显示简单的操作。需要大量交互的操作应当在单独的条目画面中处理，而不是在快速操作当中。

Aldiko和Astro都是良好的设计范例，但Google+用了一个简单的弹出框，从而违背了“覆盖目标条目”这一规则。我希望他们能够在将来的版本中修正这一可用性缺陷。



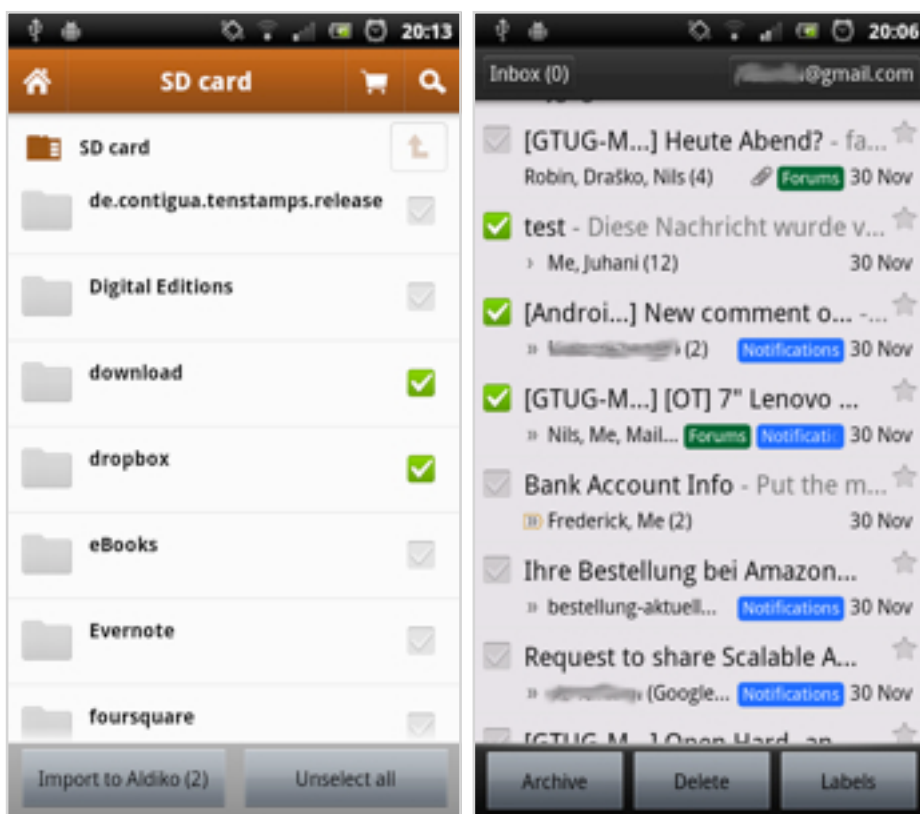


Aldiko、Astro文件管理器和Google+在快速操作中使用了多种不同的视觉风格。无论使用哪种风格，它们都通过长按画面中某个条目来调用。

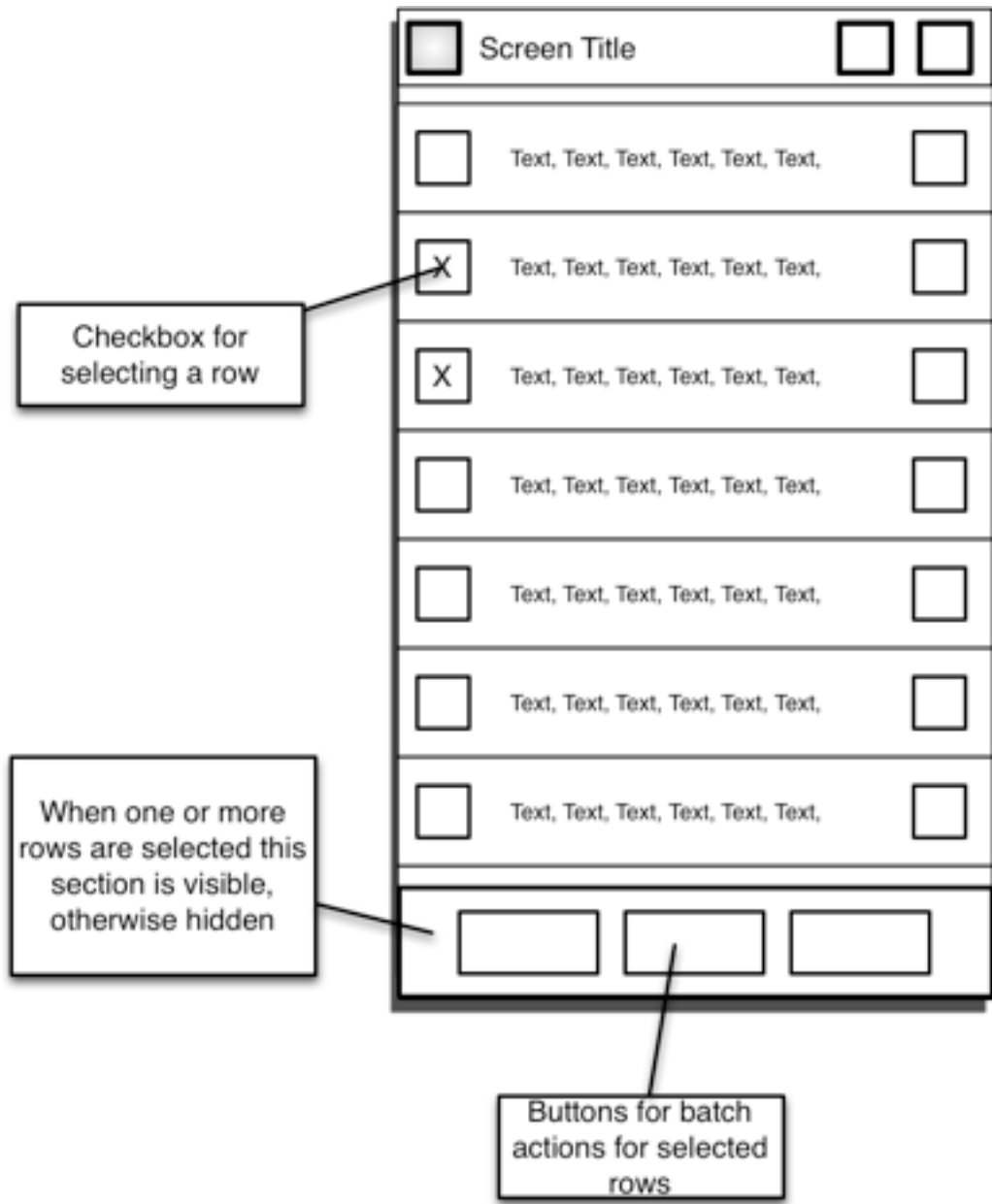
多条目操作

如果列表中使用了前面谈到的勾选框控件，就可以允许用户选择多个条目。通过选择多个条目，用户表达了他们要对所有选中条目执行某个操作的意图。

处理多个条目操作的常见方法，是当用户勾选后，画面底部增加一个面板，提供多个可用操作的按钮。一段漂亮的滑动动画将会给 UI 添加不少平滑和精致的感觉。当最后一个选项被选中或者操作执行后，该面板将会自动消失。



在多条目操作方面，Aldiko 和 GMail 都处理得不错。底部面板出现时，两个应用都会播放一段精美的滑动动画。Aldiko 还在 import 按钮当中添加一个数字来告诉用户共选中了多少个条目。这是个非常不错额外功能，但并不适用于所有的情况。



关于列表的更多信息

要了解关于列表的更多技术细节信息，请阅读以下两个系列的优秀文章：

Styling Android 的 Mark Allison：

- [ListView – Part 1](#)
- [ListView – Part 2](#)
- [ListView – Part 3](#)
- [ListView – Part 4](#)

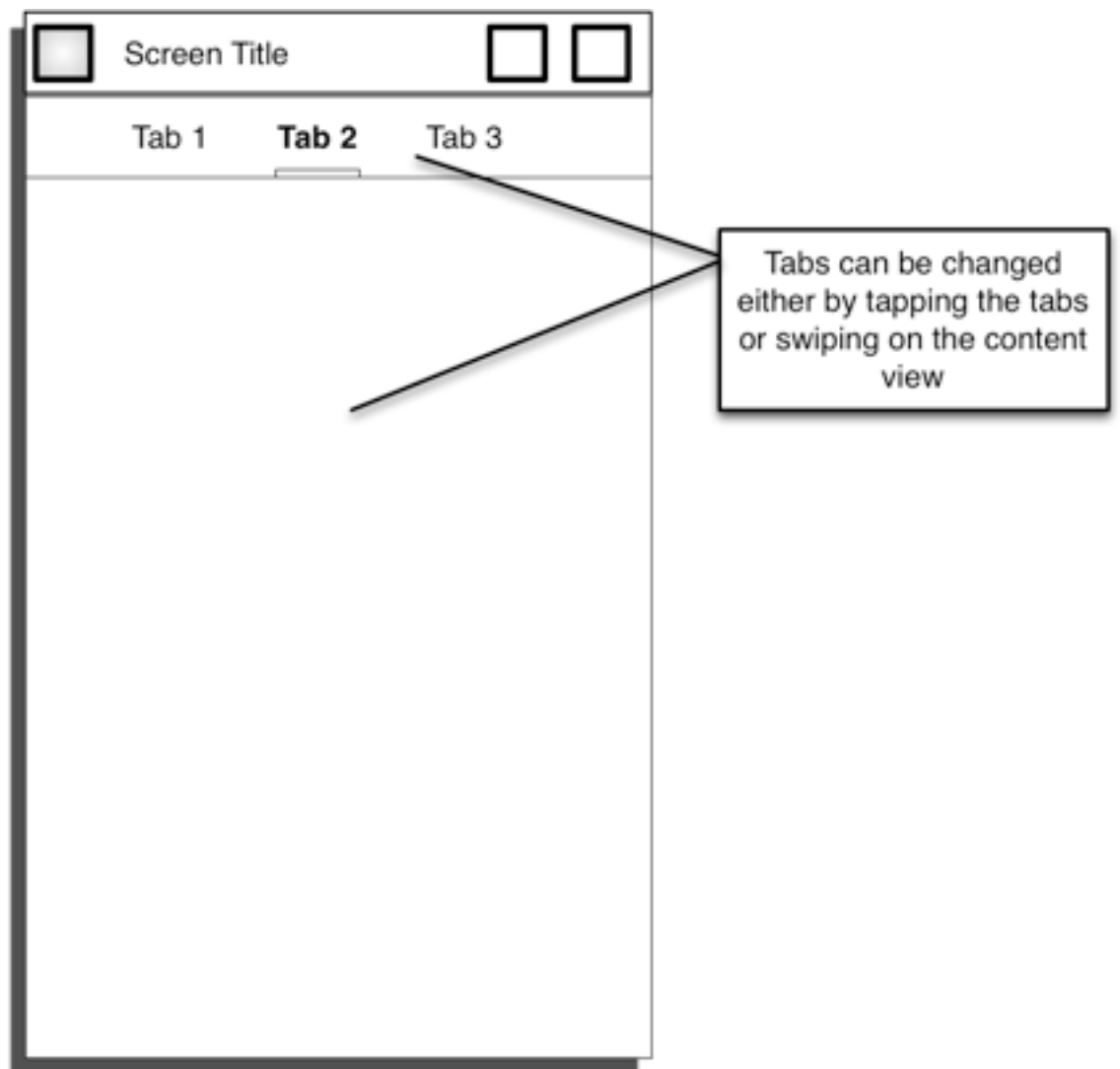
AndroidDevBlog 的 Cyril Mottier：

- [ListView Tips & Tricks #1: Handle emptiness](#)
- [ListView Tips & Tricks #2: Section your ListView](#)
- [ListView Tips & Tricks #3: Create fancy ListViews](#)
- [ListView Tips & Tricks #4: Add several clickable areas](#)

标签页

很多应用都通过某种形式使用标签页，来帮助用户在多个页面中操作。Android 版本 Honeycomb (3.0) 和 Ice Cream Sandwich (4.0) 对标签页的工作模式及外观做了大幅修改。我认为，我们应该尝试在所有应用中加入这些修改，无论是运行在哪个版本中。

我在[这篇博文](#)中写过关于 ICS 的内容，在此我不打算重复该内容。长话短说，是标签页间进行导航有了新的方法。用户的期待是：如果应用中有标签页，他们可以通过拖动（Swiping）的方式来切换。



查看英文原文：[What Android Apps Should Look like?](#)

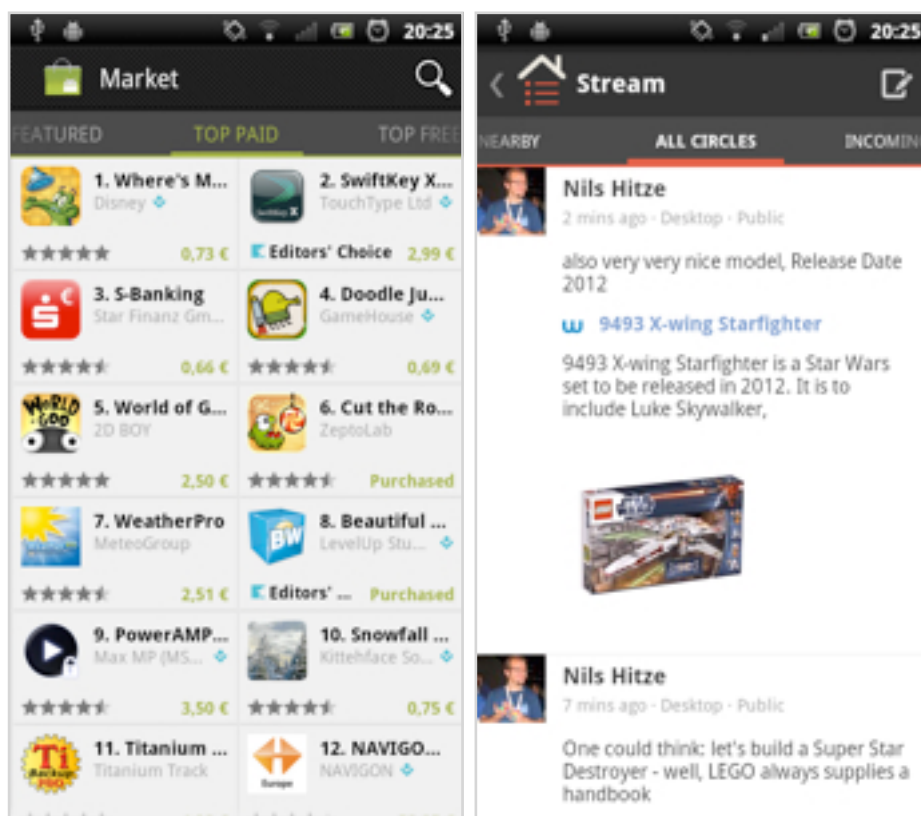
译者 / [惑者](#)

Python、Django 等基于开源社区项目的IT技术狂热分子，业余级别的中翻英译者，小型 Android +J2EE 团队项目协调者（自定义）。

结论

Android 正在迅速发育成熟成一个可靠且稳定的平台。应用在观感方面开始具备全面的稳定性，而用户则开始期待使用确定的方式来交互。尽管没有官方的准则，但对出色的应用进行深入研究，我们可以更好地理解自己应该如何去做。

Android Market 和 Google+ 是通过拖动手势在多个标签页中进行导航操作的不错范例。



Mark Allison 也曾撰写了关于该技术实现的多篇出色文章：

- [ViewPager – Part 1](#)
- [ViewPager – Part 2](#)
- [ViewPager – Part 3](#)

如果需要实现细节方面的帮助，可以看看Jake Wharton 的项目：[ViewPagerIndicator at GitHub](#)

为什么Pinterest是最让人失望的社交网络？



作者 / Reggie Ugwu

[@ocugwu](#)

Complex 网站记者

我得说：我讨厌**Pinterest**。

哦，这么说也不准确，因为我对它可不单单是厌恶。我憎恶Pinterest，而且希望它死得难看点儿；这不仅是为我自己考虑，也是为了全人类着想。

吐槽两句之后，感觉心情好多了，唉……好吧，在你们开始喷我之前，容我先辩解一下。

背景信息

对外行来说，[Pinterest](#)是人们分享和浏览图片的地方，简单来讲就是这样的。图片几乎可以来自网络上的任何地方（至少包括所

有的非成人网站），Pinterest让人们只用一次点击（即所谓的“Pinning”）即可分享这些图片，大大简化了操作。

每个用户都可以将照片（视频也可以，不过不常见）整理到各种“pinboards”（别针板）上。这样，如果我在网上看到了喜欢的小狗的照片，我就可以把它pin到名为“Dogs”的pinboard上。这不仅是种社交行为，也是种自利行为。因为发布到Pinterest上的东西默认都是公开的，人们可以表示喜欢或进行评论；而这些图片同时也保存在了你的空间里，以后需要它们的时候，可以随时调用。下次我去宠物店里买小狗，就可以将“Dogs”板给店家看，告诉他们我需要哪种小狗。

Pinterest大约在两年前上线，之后很长一段时间里，这个网站都比较封闭（尽管现在仍然需要被邀请才能注册，但已经容易多了），所受关注不多。然而，在去年年底的时候，它开始急速成长，而且出人意料地[先在女性中流行起来](#)。Pinterest就是从那会儿开始爆发的。

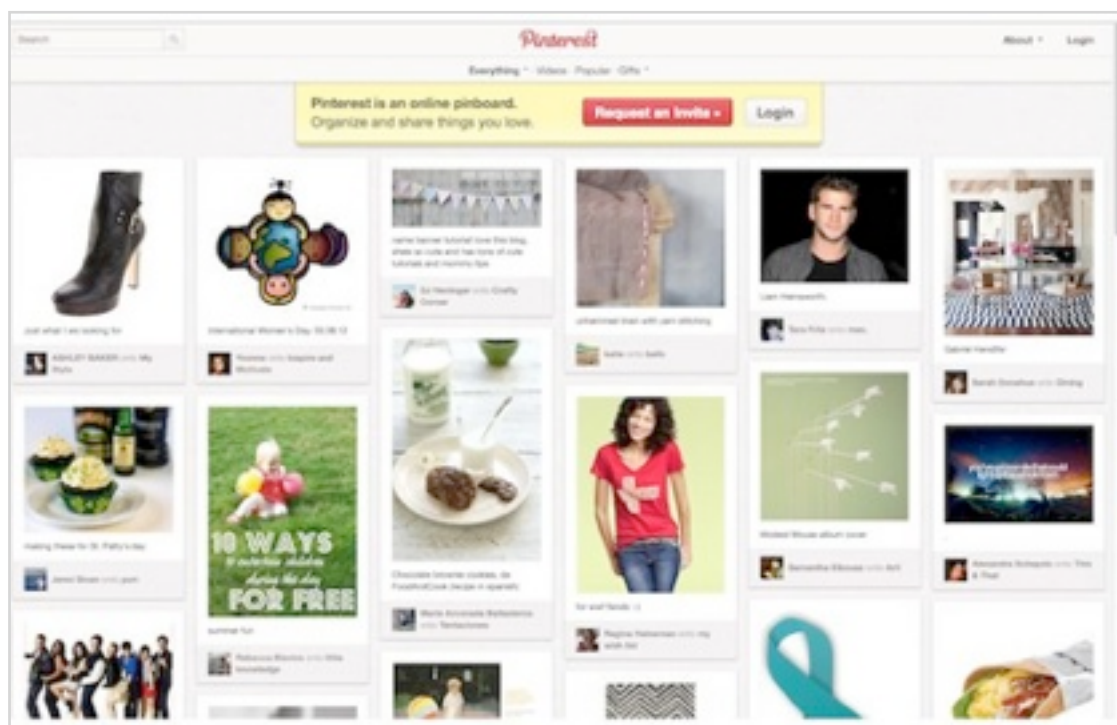
今年1月份，[comScore的报道](#)称，Pinterest在之前一个月里积累了1170万的独立访客（UV）——这个数字对于成立不到两年的网站来说是史无前例的。然后，关注这类产品的人们（[包括我们](#)）就宣称，Pinterest是下一个革命性产品（The Next Big Thing）。

为什么Pinterest最糟糕

这是我的理由：这个网站现在的形式，就是一个金玉其外的污水坑，散发着其封闭的梦想。它引导甚至鼓励社交网络上最令人失

望的一些动机——它从其他富有创意的网站上抽取内容，但却不回馈任何东西。

根据我的Gmail记录，我从2010年7月7号开始就是Pinterest的用户了，或多或少见证了这把火是怎么烧起来的。你不用在Pinterest上花多少时间，就能感受到这个网站深层的、无处不在的浅薄，而这种浅薄已经成为它赖以生存的要素。这些天来，主页上充满了大量萌物照、高跟鞋照片，以及颇有艺术感的甜点照片。



Pinterest的这个样子，并不是因为它是所谓“女性主导”的，而是因为它让人们不再创造，而去分享。即使以社交网络的标准来看，在鼓励人类进行有价值的活动方面，Pinterest依然代表了前

译者 / 张重骐

[@candyhorse](#)

北邮土著，本科学习设计，研究生转攻技术。接触Python一年多，喜欢折腾web应用开发。

热爱互联网，之前在百度、创新工场做过16个月的产品经理实习生，对LBS产品、图片应用有一定了解和研究。现在果壳网做实习Python工程师。在奔向文青与极客的道路上艰难前行。

所未有的低水准。它是缺少有力沟通工具和社区建设工具的**Facebook**，是没有地道的自我表现平台的**Tumblr**，是不需要拍摄或编辑照片的**Instagram**。

在**Pinterest**上，人们挑选并分享那些据说是“启发”了他们的图片，乍一看好像是一种很慷慨的举动。但我们在这儿关注的除了被分享的东西外，还有分享者自身。**Pinterest**的用户竭力追求“粉丝数”，所以他们要做的就是发布最漂亮、最幽默或是最鼓舞人的照片、图片以及产品——然而这些东西中的绝大多数都是其他人创作或拥有的，而且没有授权给你使用。**Pinterest**奉行一种宗旨：我喜欢这样做，所以我就这样做；在这一点上，它比同时代的其他产品要严重得多。

Pinterest原则上鼓励用户标明作品的原始出处，但可以料想，其用户大都不注意这些。目前看来，它会让你觉得像是身处**Tumblr**，而作品却被贴在**Pinterest**，这一切怎么可能呢？上月末的时候，它被迫发布了一个HTML代码“nopin”，让其他网站保护其内容，以免在未经允许的情况下被分享到**Pinterest**上，像**Flickr**这样的网站几乎立即[加上了这个代码](#)。

此时此刻，要求我们的社交媒体不再追求“平均停留时间”的增长，可能有些过分。网上的事物越来越注重是否能改善我们的生活体验，即使扭曲或者重新定义了我们的生活，然而这已经是大势所趋。不过，我们期盼的是更好的产品，而不是像**Pinterest**这样的现代互联网蛀虫。这个“虚拟的别针板”所承诺的世界，就像针头一样空洞。 ■

JavaScript并行运算新机遇

——Web Workers的神奇魔法



作者 / David Rousset

[@davrous](#)

Windows 8/HTML5布道师

很明显HTML5的应用是用JavaScript写的，但是跟其他的开发环境相比（例如一些原生的），JavaScript一直有个很严重的局限性：它的所有执行进程都在同一个线程里。

对于如今像i5/i7这种动辄就8个CPU的多核处理器来说，这就有些麻烦了。最新的ARM手机处理器也都是双核或者4核。顺利的话，我们有望看到HTML5为Web开发提供一个应对这些又新又强劲的处理器的方法，让我们可以拥抱一个Web应用开发的新时代。

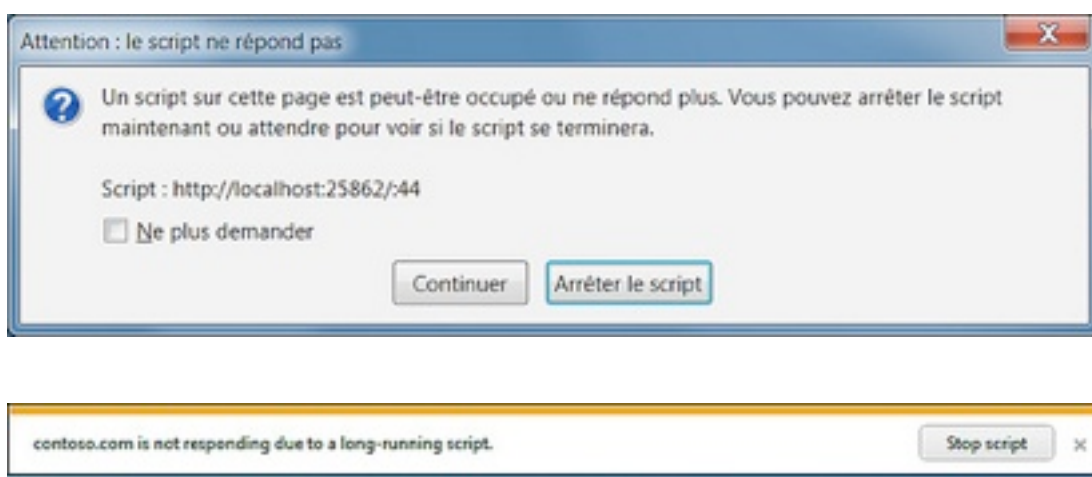
在没有 Workers 之前

这个JavaScript的局限性意味着一个长时间运行的进程会冻结主窗口。我们常说我们被“UI 线程”阻塞，这是因为主线程在处理所有的可视化元素及其相关任务：绘制，刷新，动画，用户输入事件等等。

我们都知道这个线程过载的严重后果：页面冻结，并且用户不能再与应用进行交互了，这时的用户体验相当差。用户可能关掉这个Tab或者整个浏览器，你当然不希望看到这发生在你的app上。

为了避免这种情况，浏览器引入了一种保护机制：当一个脚本有可能长时间运行时，会对用户进行提示。

悲剧的是，这种机制并不能正确分辨究竟是脚本编写得有问题，还是它确实需要更多的时间来完成它的工作。尽管如此，因为它阻塞了UI线程，所以还是提示现在可能发生错误了比较好。下面是一些提示消息的例子（从Firefox 5 和 IE9 获得）：



迄今为止，由于以下两个原因，那些问题已经很少发生了：

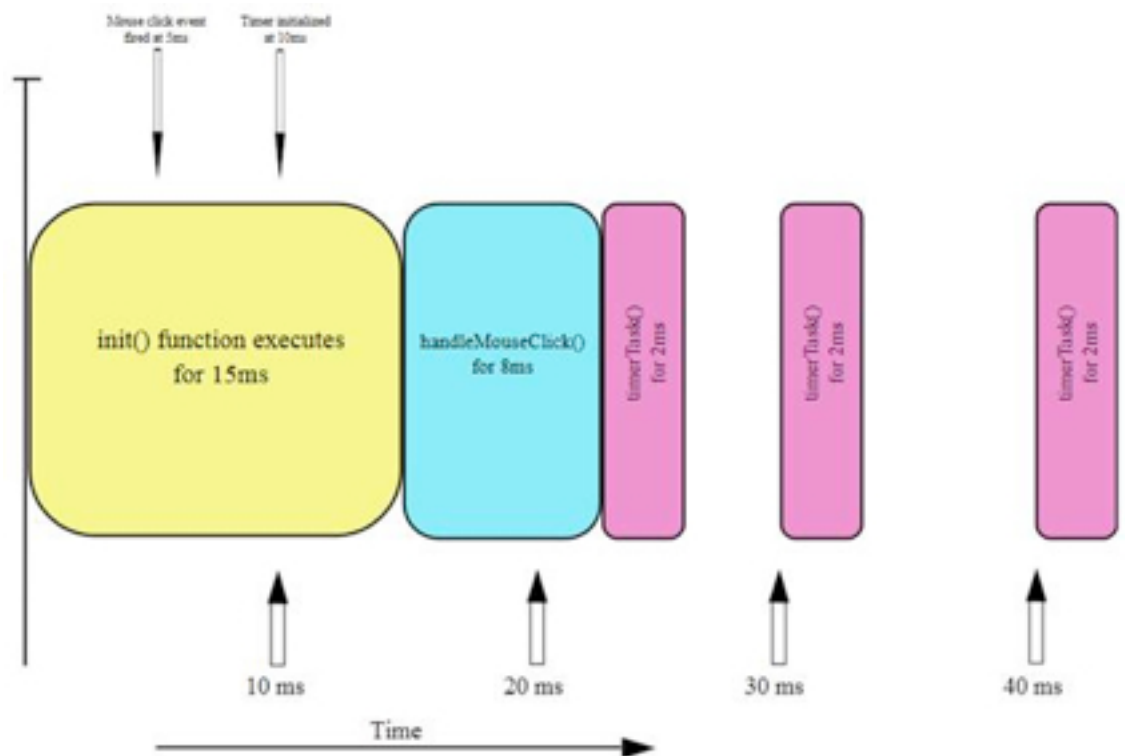
1. 因为其他的技术可以完成多线程的任务，HTML和JavaScript的使用方式和使用目的已经与不同于以往。与本地应用相比，网站提供的体验更简单了。
2. 总有一些其他的办法可以或多或少地解决这些并发问题。

那些办法都是Web开发者所熟知的。例如，通过 `setTimeout()` 和 `setInterval()` 方法，我们可以尝试模拟并行任务。通过 `XMLHttpRequest` 对象，也可以异步地处理HTTP请求，避免从远程服务器载入资源时冻结UI。最后，应用DOM事件写出的应用程序能够给人一种错觉，让人误以为几个事件是在同时发生。真的是错觉吗？是的！

为了更好的理解其原理，让我们来看一段伪代码，并且看看在浏览器内部究竟发生了什么：

```
1 <script type="text/javascript">
2   function init(){
3       { piece of code taking 5ms to be executed }
4       A mouseClickEvent is raised
5       { piece of code taking 5ms to be executed }
6       setInterval(timerTask,"10");
7       { piece of code taking 5ms to be executed }
8   }
9
10  function handleMouseClick(){
11      piece of code taking 8ms to be executed
12  }
13
14  function timerTask(){
15      piece of code taking 2ms to be executed
16  }
17 </script>
```

让我们为这段代码建立一个模型。这个图表展示了在一个时间段内，浏览器内部发生了什么：



这个图表很好地诠释了任务的非并行本质。5毫秒后，用户产生一个鼠标点击事件。然而，由于`init()`方法仍然在执行，并且独占了主线程，所以这个事件不能被立即处理。点击事件将被保存并且延迟处理。

- 从5毫秒到10毫秒之间：`init()`方法在这5毫秒中仍然执行，然后在10毫秒时请求调用 `timerTask()` 方法。这个方法理论上应该在20毫秒的时间点执行。

- 从10毫秒到15毫秒之间：init()方法仍然需要5毫秒来完成运行。这与15毫秒时的黄色区块相对应。由于我们冻结了主线程，所以浏览器现在才可以继续进行刚才保存的请求。
- 从15毫秒到23毫秒之间：浏览器开始执行handleMouseClicked()方法，该方法执行了8毫秒（蓝色区块）。
- 从23毫秒到25毫秒之间：作为一个副作用，在20毫秒时间点就应该执行的timerTask（）方法被稍微平移了3毫秒。而其他的时间点（例如30ms、40ms），被当作没有代码占用CPU。

说明：这个例子和上面的图表（通过特征监测机制判断使用SVG或者PNG）是受到这篇文章的启发：[HTML5 Web Workers Multithreading in JavaScript](#)

所有这些提示并没有解决我们最初的问题：所有东西都在主UI线程里执行。

此外，即使JavaScript还没有被用来开发像其他“高级语言”一样的应用，它仍然在随着HTML5和其相关技术所提供的新的可能而改变。因此给JavaScript赋予更多新的能力，使之能够建立新一代的能够处理并行任务的应用，就变得更加重要了。这就是为什么我们有了Web Workers。

Web Workers或如何在UI线程外执行代码

[Web Workers APIs](#)定义了一个在后台运行脚本的方法。你可以在主页面之外的线程执行一些任务，而不影响页面的绘制性能。然而，如同不是所有的算法都能并行执行，也不是所有的JavaScript代码都能从Workers中受益。Ok，唠叨得够多了，我们看看这些著名的Workers。

我的第一个Web Worker

由于Web Workers将在一个独立的线程里执行，你必须把代码从主页面中分离出来，放到独立的文件中。完成这些后，需要实例化一个Worker对象来调用它们：

```
var myHelloWorker = new Worker('helloworkers.js');
```

然后就可以给它发送一条信息来开启Worker（由此也开启了一个窗口之外的线程）：

```
myHelloWorker.postMessage();
```

是的，Web Workers和主页面通过消息进行通信。这些消息可以是一般的字符串或者JSON对象。为了演示简单的消息发送，我们来review一个非常基础的例子。这个例子会发送一个字符串给worker，将其与worker联系起来。首先，将下面代码放到“helloworker.js”文件中：

```
1 function messageHandler(event) {
```

```

2      // Accessing to the message data sent by the main
page
3      var messageSent = event.data;
4      // Preparing the message that we will send back
5      var messageReturned = "Hello " + messageSent + "
from a separate thread!";
6      // Posting back the message to the main page
7      this.postMessage(messageReturned);
8  }
9
10 // Defining the callback function raised when the main
page will call us
11 this.addEventListener('message', messageHandler,
false);

```

我们只在“helloworkers.js”中定义了一小段将在另一个线程执行的代码。它可以从主页面接收消息，执行任务，并向主页面返回一个消息。然后我们需要在主页面编写一个接收者。下面是处理消息的页面：

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Hello Web Workers</title>
5  </head>
6  <body>
7      <div id="output"></div>
8
9      <script type="text/javascript">
10         // Instantiating the Worker
11         var myHelloWorker = new
Worker('helloworkers.js');

```

```
12         // Getting ready to handle the message sent
back
13         // by the worker
14         myHelloWorker.addEventListener("message",
function (event) {
15             document.getElementById("output").textContent
= event.data;
16             }, false);
17
18         // Starting the worker by sending a first
19 message
        myHelloWorker.postMessage("David");
20
21         // Stopping the worker via the terminate()
22 command
        myHelloWorker.terminate();
23     </script>
24 </body>
25 </html>
```

运行的结果将是: "Hello David from a separate thread!", 你被打动了, 有木有?

你要注意worker会一直存活, 直到它被终止。

既然没有自动垃圾收集, 那么控制它们的状态就全靠你自己了。要记住, 初始化一个worker会消耗一定的内存.....而且也不要忽略冷启动时间。要想停止一个worker, 有两种可能的方式:

1. 从主调用页面调用terminate()命令。
2. 在worker内部通过调用close()命令。

演示：你可以在浏览器中测试这个稍微增强了一点的例子：

[http://david.blob.core.windows.net/html5/
HelloWebWorkers_EN.htm](http://david.blob.core.windows.net/html5/HelloWebWorkers_EN.htm)

通过JSON发送消息

当然，大多数时候我们会发送更加结构化的数据给Workers。

（顺便说一下，Web Workers也可以通过[Message channels](#)进行通讯）。

但是使用JSON格式是唯一可以给worker发送结构化消息的方法。幸运的是，浏览器现在支持worker的程度已经与原生支持JSON的程度一样好了。它们真是太好了！

让我们拿出之前的代码例子。我们打算增加一个WorkerMessage类型的对象。该类型将被用于向Web Workers发送一些带参数的命令。

我们使用下面这个简化版的HelloWebWorkersJSON_EN.htm页面：

```
1      <!DOCTYPE html>
2      <html>
3      <head>
4          <title>Hello Web Workers</title>
5      </head>
6      <body>
7          <div id="output"></div>
8
9          <script type="text/javascript">
10             // Instantiating the Worker
```

```

11         var myHelloWorker = new
Worker('helloworldworkers.js');
12         // Getting ready to handle the message sent
back
13         // by the worker
14         myHelloWorker.addEventListener("message",
function (event) {
15             document.getElementById("output").textContent
= event.data;
16             }, false);
17
18         // Starting the worker by sending a first
message
19         myHelloWorker.postMessage("David");
20
21         // Stopping the worker via the terminate()
command
22         myHelloWorker.terminate();
23     </script>
24 </body>
25 </html>

```

我们使用一种非侵入式的JavaScript方法来分离表现层和逻辑层。然后绑定的逻辑就存在于HelloWebWorkersJSON_EN.js文件中：

```

1 // HelloWebWorkersJSON_EN.js associated to
HelloWebWorkersJSON_EN.htm
2
3 // Our WorkerMessage object will be automatically
4 // serialized and de-serialized by the native JSON
parser
5 function WorkerMessage(cmd, parameter) {
6     this.cmd = cmd;

```

```
7         this.parameter = parameter;
8     }
9
10    // Output div where the messages sent back by the
    worker will be displayed
11    var _output = document.getElementById("output");
12
13    /* Checking if Web Workers are supported by the browser
    */
14    if (window.Worker) {
15        // Getting references to the 3 other HTML elements
16        var _btnSubmit =
    document.getElementById("btnSubmit");
17        var _inputForWorker =
    document.getElementById("inputForWorker");
18        var _killWorker =
    document.getElementById("killWorker");
19
20        // Instantiating the Worker
21        var myHelloWorker = new
    Worker('helloworldworkersJSON_EN.js');
22        // Getting ready to handle the message sent back
23        // by the worker
24        myHelloWorker.addEventListener("message", function
    (event) {
25            _output.textContent = event.data;
26        }, false);
27
28        // Starting the worker by sending it the 'init'
    command
29        myHelloWorker.postMessage(new WorkerMessage('init',
    null));
30
31        // Adding the OnClick event to the Submit button
32        // which will send some messages to the worker
```

```

33     _btnSubmit.addEventListener("click", function
(event) {
34         // We're now sending messages via the 'hello'
command
35         myHelloWorker.postMessage(new
WorkerMessage('hello', _inputForWorker.value));
36     }, false);
37
38     // Adding the OnClick event to the Kill button
39     // which will stop the worker. It won't be usable
anymore after that.
40     _killWorker.addEventListener("click", function
(event) {
41         // Stopping the worker via the terminate()
command
42         myHelloWorker.terminate();
43         _output.textContent = "The worker has been
stopped.";
44     }, false);
45 }
46 else {
47     _output.innerHTML = "Web Workers are not supported
by your browser. Try with IE10: <a href=\"http://
ie.microsoft.com/testdrive\">download the latest IE10
Platform Preview</a>";
48 }

```

再次说明，这个例子是非常基础的。但是，它可以帮助你理解背后的逻辑。当然，没人能阻止你发送一些可以被人工智能或者物理引擎处理的游戏元素。

演示：可以在这儿测试JSON的例子：

http://david.blob.core.windows.net/html5/HelloWebWorkersJSON_EN.htm

浏览器支持

Web Workers刚刚出现在了IE10平台预览版上。Firefox（3.6以上）、Safari（4.0以上）、Chrome和Opera11也都支持它。然而，这些浏览器的手机版并不支持。如果你想获得更详尽的浏览器支持列表，可以看看这里：

<http://caniuse.com/#search=worker>

要实时地了解代码的支持情况，请使用特性监测机制。（你不应该使用神马用户代理嗅探！）

为了帮助你实现，这里有两个可用的解决方案。第一个是用这样一小段代码，自己简单地测试特性：

```
1  /* Checking if Web Workers are supported by the browser
   */
2  if (window.Worker) {
3      // Code using the Web Workers
4  }
```

第二个是使用著名的[Modernizr](#)库（现在已经原生的移到了ASP.NET的MVC3项目模版中）。然后，只要用下面这样一段代码即可：

```
1  <script type="text/javascript">
2      var divWebWorker =
document.getElementById("webWorkers");
3      if (Modernizr.webworkers) {
4          divWebWorker.innerHTML = "Web Workers ARE
supported";
```

```
5         }
6         else {
7             divWebWorker.innerHTML = "Web Workers ARE NOT
supported";
8         }
9     </script>
```

译者注：原文页面对当前浏览器支持情况进行监测，并将结果展示在这里。

例如，这就是你的浏览器支持情况：**Web Workers are not supported inside your browser.**

这将使你的应用产生两个版本。如果**Web Workers**不被支持，就正常执行你的**avaScript**代码。在大多数现代浏览器中，**Web Workers**是被支持的，这种情况下就可以推送一些**JavaScript**代码给**workers**，以便提高应用的性能。这样就不必中断任何事，或为最新的浏览器单独建立一个版本了。在全部浏览器中，它都能运行，只是性能稍有差别。

Worker不能访问的元素

——Worker不能干什么

与其看看你用**Workers**不能干什么，不如让我们了解一下你只能用**worker**干点儿什么：

Method	Description
<code>void close();</code>	Terminates the worker thread.
<code>void importScripts(urls);</code>	A comma-separated list of additional JavaScript files.
<code>void postMessage(data);</code>	Sends a message to or from the worker thread.

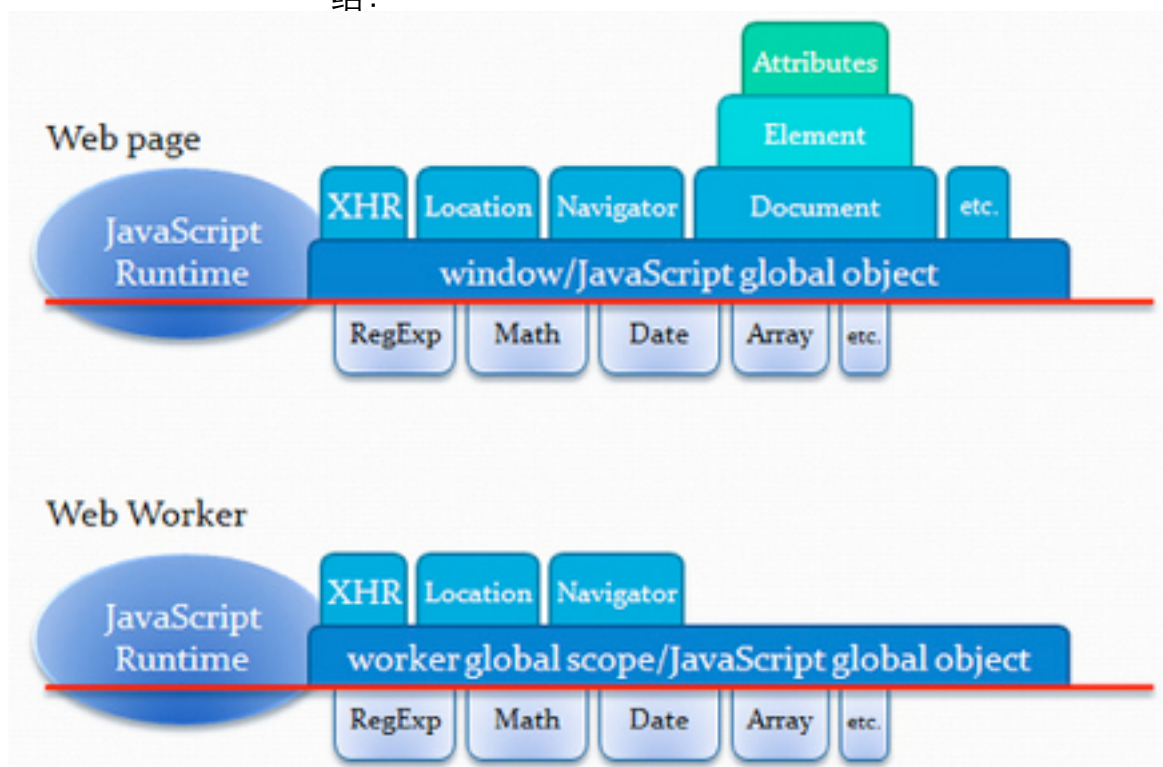
Attributes	Type	Description
<code>location</code>	WorkerLocation	Represents an absolute URL, including protocol, host, port, hostname, pathname, search, and hash components.
<code>navigator</code>	WorkerNavigator	Represents the identity and onLine state of the user agent client.
<code>self</code>	WorkerGlobalScope	The worker scope, which includes the WorkerLocation and WorkerNavigator objects.

Event	Description
<code>onerror</code>	A runtime error occurred.
<code>onmessage</code>	Message data received.

Method	Description
<code>void clearInterval(handle);</code>	Cancels a timeout identified by handle.
<code>void clearTimeout(handle);</code>	Cancels a timeout identified by handle.
<code>long setInterval(handler, timeout value, arguments);</code>	Schedules a timeout to be run repeatedly after the specified number of milliseconds. Note that you can now pass additional arguments directly to the handler. If handler is a DOMString, it is compiled as JavaScript. Returns a handle to the timeout. Clear with <code>clearInterval</code> .
<code>long setTimeout(handler, timeout value, arguments);</code>	Schedules a timeout to run after the specified number of milliseconds. Note that you can now pass additional arguments directly to the handler. If handler is a DOMString, it is compiled as JavaScript. Returns a handle to the timeout. Clear with <code>clearTimeout</code> .

说明：这些表格是从MSDN文档中引用的：[HTML5 Web Worker](#)

总之，你没有操作**DOM**的权限。这有一个非常好的图表作为总结：



举个例子，既然在worker中无法操作window对象，因此就无法访问本地存储（**Local Storage**，反正看起来也不像线程安全的）。对于在其他环境中习惯了多线程操作的开发者来说，这些限制或许看起来过于严格了。然而，它带来的最大的优点，就是我们会不再陷入那些经常遇到的问题：死锁、竞争条件等。在**Web Workers**中，完全不用考虑对于这些。这样，当一些特定的场景需要增强性能时，**Web Workers**是非常好用的。

错误处理与调试

处理Web Workers的错误非常容易，用与注册OnMessage事件同样的方法，注册一个OnError事件即可：

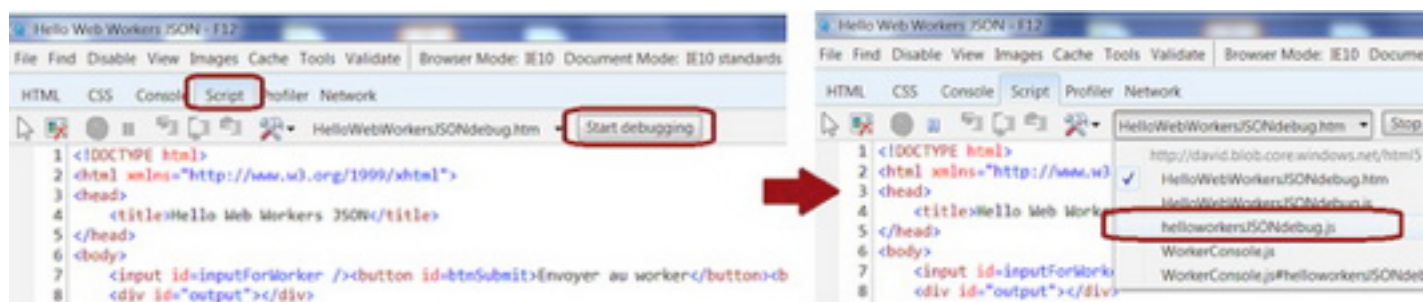
```
1 myWorker.addEventListener("error", function (event) {  
2     _output.textContent = event.data;  
    }, false)
```

这已经是为了调试代码，Web Worker所提供的的最好的原生支持了.....非常有限，对吧？

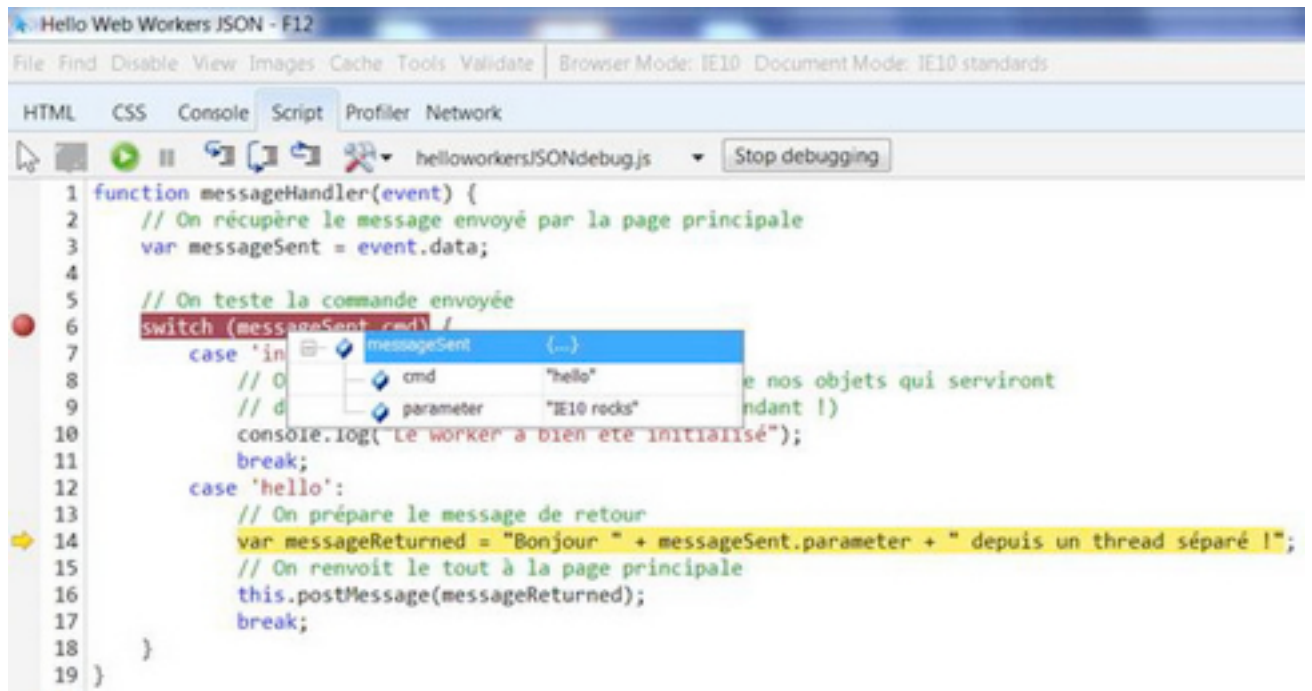
通过F12开发工具获得更好的调试体验

为了突破这些局限，IE10在它的脚本调试器中提供了一个直接调试Web Workers代码的功能，就像调试其他脚本一样。

对此，你需要通过F12键调出开发者工具栏，并且打开“脚本”标签。现在，应该还看不到与worker相关的JS文件。但一旦点击“开始调试”按钮，它就应该神奇地出现了：



下一步就是像调试以往的JavaScript代码一样，调试worker了！



IE10是目前唯一支持这样调试的浏览器。如果想了解更多关于这个特性的细节，可以读一下这篇文章：[Debugging Web Workers in IE10](#)

一个用来模拟console.log()的有趣方法

最后，你要知道，在worker中是不能使用console对象的。通过.log()方法来跟踪worker内部发生了什么没有用的，因为console对象没有定义。幸好，我找到一个有趣的方法，即通过MessageChannel：[console.log\(\) for Web Workers](#)，它可以模拟console.log()行为。该方法在IE10、Chrome和Opera中运行良好，但是在Firefox中还不行，因为Firefox还不支持MessageChannel。

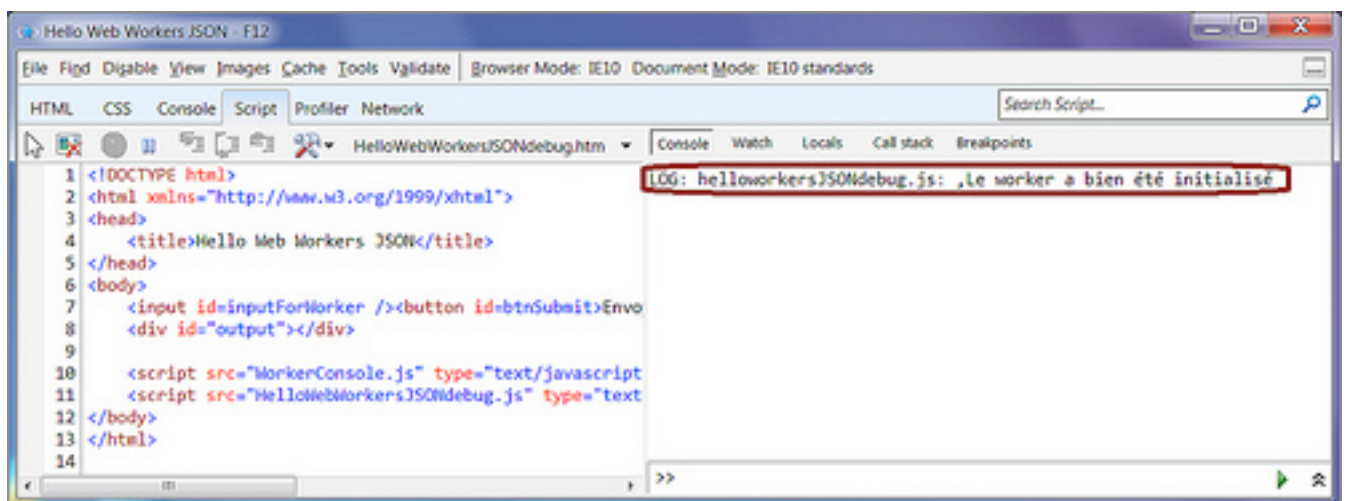
说明：为了使链接中的例子能在IE10下能运行，需要把下面这行代码：

```
console.log.apply(console,  
args); // Pass the args to the real log
```

修改成：

```
console.log.apply(console, args); // Pass the args to  
the real log
```

然后，应该可以得到这样的结果：



例子：如果想使用这个console.log()模拟，请到这里：

[http://david.blob.core.windows.net/html5/
HelloWebWorkersJSONdebug.htm](http://david.blob.core.windows.net/html5/HelloWebWorkersJSONdebug.htm)

用例分析与可以取代的场景

在哪些场景中使用Web Workers?

在网上查找Web Workers的例子，总会找到这一类：强化的数学/科学计算。然后是一些JavaScript光线跟踪、分形、素数之类的东西。它们虽然是理解Workers运行方式的好例子，但无法就如何在“真实的世界“的应用中使用它们给出具体建议。

确实，上面展示的这些Web Workers的缺点缩小了限制了使用Web Workers的有趣场景。尽管如此，花点儿时间仔细想想，就会发现一些新的有趣用途：

- **图像处理** 通过使用<canvas>或者<video>元素中获取的数据，可以把图像分割成几个不同的区域，并把它们推送给并行的不同Workers，这样就能在新一代的多核处理器中受益。受益越多，运行得就越快。
- **大量数据** 检索调用 XMLHttpRequest后要处理的大量数据。如果处理这些数据所需的时间长短非常重要，最好在后台Web Worker中来做，以免冻结UI线程。这样可以保持一个可交互的应用。
- **背景数据分析**：因为使用Web Workers时我们有了更多潜在的CPU可用时间，现在可以考虑JavaScript中的新应用场景。例如，可以想像在不影响UI体验的情况下，实时处理用户输入。有了这样一种可能，就可以想像一个像Word（Office Web Apps 套装）一样的应用：用户打字时，后台在词典中进行查找，帮助用户自动纠错等等。

- 针对本地数据的并发请求。IndexDB 可以提供本地存储（Local Storage）不具备的特性：一个针对Web Workers的线程安全的存储环境。

此外，如果想转到视频游戏的世界，可以考虑推送人工智能或者物理引擎的数据到Web Workers。例如，我做了这样一个小实验：[On Web Workers, GWT, and a New Physics Demo](#)，该实验使用[Box2D physic engine](#)和Workers。对人工智能引擎来说，这意味着可以使用同样的时间帧来处理更多的数据（例如在棋类游戏中预测更多的步数）。

我的一些同事或许会说唯一的限制就是想象力！

但是一般来说，只要你不需要DOM，任何可能影响用户体验的耗时的JavaScript代码，都可以用Web Workers来代替。然而，使用Workers时还需要注意以下三点：

1. worker的初始化和通讯时间不应该比自身的处理时间长。
2. 使用多个Workers时的内存消耗。
3. 代码块之间的依赖关系，可能需要一些同步的逻辑。并行没那么简单！

我们最近发布了一个演示，叫做[Web Workers Fountains](#):



这个例子展示了一些颗粒效果（喷泉），并且对每个喷泉使用一个Web Worker来尽可能快地计算这些粒子。每个Worker的结果汇总后显示在<canvas>元素中。Web Workers也可以在通过Message Channels 相互交换信息。在本例中，这被用来询问每个Workers何时改变喷泉的颜色。之后循环这组颜色数组：红色，橙色，黄色，绿色，蓝色，紫色和粉色，感谢Message Channels！如果对细节感兴趣，请跳到 Demo3.js 文件中的 LightManager() 函数部分。

欢迎大家在 Internet Exploer 10 中运行这个例子，非常好玩！

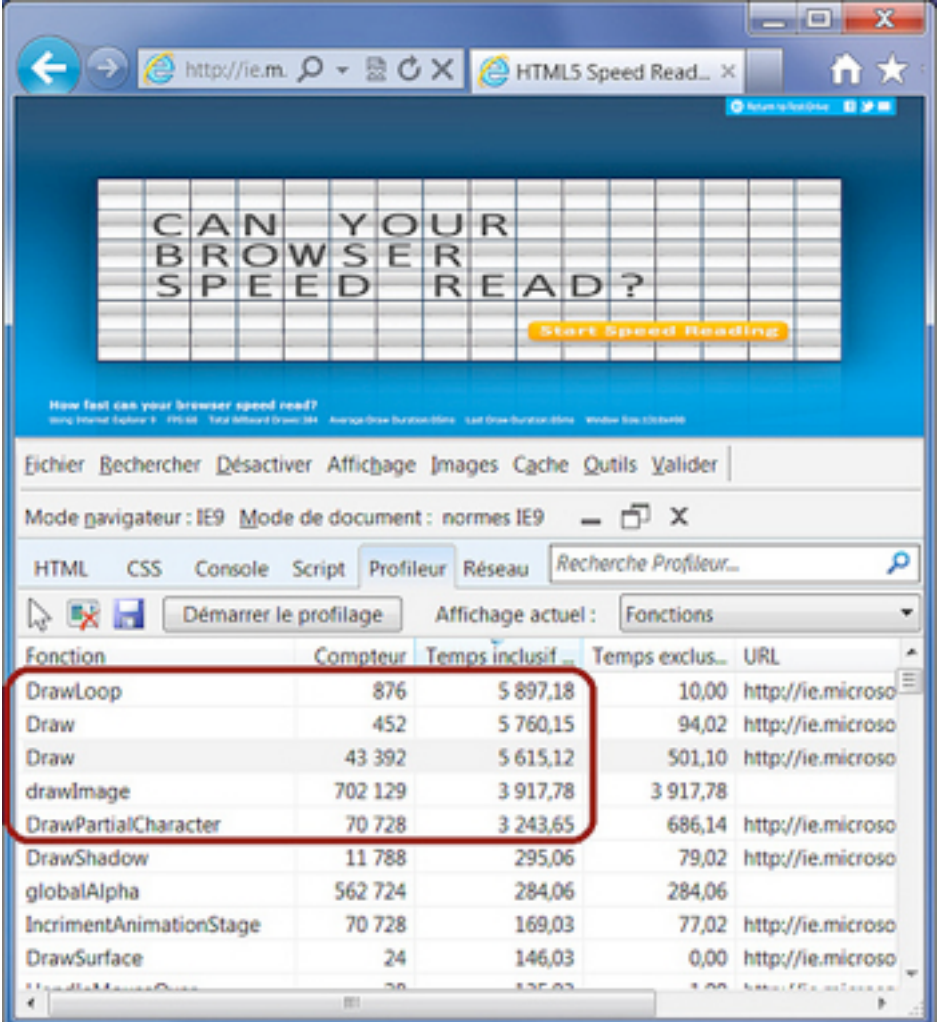
如何识别代码中的热点

为了追踪代码的瓶颈，并识别代码中的哪些部分可以发送给Web Workers，我们可以使用IE9/10中提供的F12工具栏中的脚本探查器。它可以帮助你识别代码中的热点。然而，识别一个热点并不意味着已经找到一个适合Web Workers的场景。为了更好地理解这些，我们来review两个有趣的案例。

案例1：<canvas>中的速读动画演示

这个演示是从 [IE Test Drive](#) 获取的，可以直接在这儿找到：[Speed Reading](#)。该例试图使用<canvas>来尽可能快地显示字符。目的是强调浏览器执行硬件加速层的质量。但是除此之外，把一些操作分割成线程能获得更好的性能吗？我们需要做一些分析来验证。

如果在IE9/10中运行这个例子，可以同时在这几秒之内打开探查器。下面是得到的结果：



Fonction	Compteur	Temps inclusif...	Temps exclus...	URL
DrawLoop	876	5 897,18	10,00	http://ie.microso
Draw	452	5 760,15	94,02	http://ie.microso
Draw	43 392	5 615,12	501,10	http://ie.microso
drawImage	702 129	3 917,78	3 917,78	
DrawPartialCharacter	70 728	3 243,65	686,14	http://ie.microso
DrawShadow	11 788	295,06	79,02	http://ie.microso
globalAlpha	562 724	284,06	284,06	
IncrimentAnimationStage	70 728	169,03	77,02	http://ie.microso
DrawSurface	24	146,03	0,00	http://ie.microso

如果降序排列那些比较耗时的方法，会清楚地看到那些最先出现的方法：`DrawLoop()`，`Draw()`和`drawImage()`。如果双击`Draw`这一行，就会跳到这个方法对应的代码。你会看到几个这种类型的调用：

```
surface.drawImage(imgTile, 0, 0, 70, 100, this.left,
this.top, this.width, this.height);
```

这里`surface`对象引用了一个`<canvas>`元素。

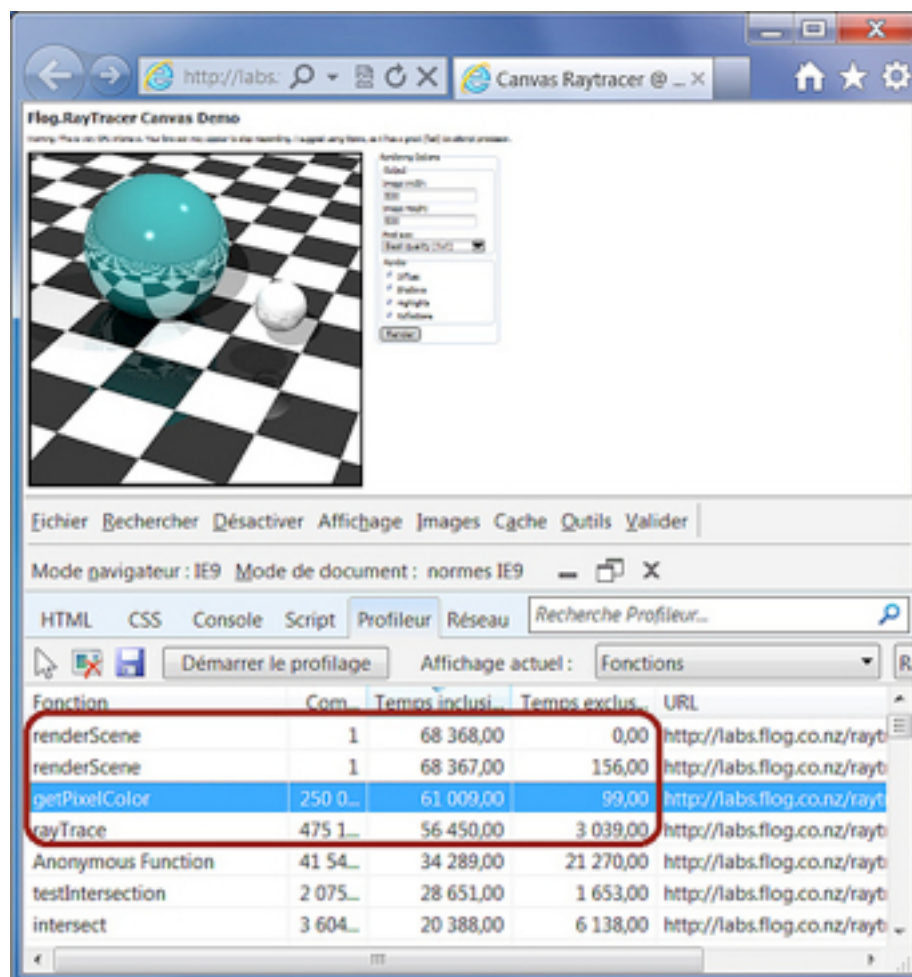
通过简短的分析，我们可以得到一个初步的结论，通过`drawImage()`方法在`Canvas`内部绘图用掉了大多数时间。由于`Web Worker`无法获取`<canvas>`元素，我们无法将这个耗时的任务分离到其他的线程中（可以想像一些并发处理`<canvas>`元素的方法）。这个例子就不是一个很好的用`Web Workers`处理并行的场景。

但是它很好地说明了你应该采取的操作过程。如果经过一些探查工作后，你发现耗时的脚本的主要部分是与`DOM`对象紧密耦合的，就没法用`Web Workers`来增强`Web app`的性能了。

案例2：<canvas>元素中的光线追踪

我们再举一个简单的例子帮助理解，以下是一个光线追踪的例子：[Flog.RayTracer Canvas Demo](#)。光线追踪使用一些CPU密集型的数学计算，据此来模拟光线的路径。主要用来模拟一些诸如反射，折射，材质等效果。

运行脚本探索器时，应该能得到类似这样的结果：



如果降序排列这些方法，有2个方法明显地占用了大多数时间：
`renderScene()`和`getPixelColor()`。

`getPixelColor()`方法的目的是计算当前的像素。光线追踪是一个像素一个像素的渲染场景。`getPixelColor()`方法之后，再调用`rayTrace()`方法接管渲染阴影，环境光等等操作，这是应用的核心部分。如果这个review一下`rayTrace()`方法的代码，你会发现它是100%原汁原味的JavaScript。这些代码没有任何DOM依

查看英文原文：

[Introduction to HTML5](#)

[Web Workers: The](#)

[JavaScript Multi-](#)

[Threading Approach](#)

译者 / 姬光

[@姬小光](#)

男，前端开发一枚，目前就职于腾讯ISUX拍拍设计中心前端开发组，爱好桌球、篮球、KTV等，更爱结交好友。

博客：Hi,laser! (www.44ux.com)

赖。好吧，我相信你懂的：这个例子非常适合并行处理。此外，由于每个像素的计算没有同步进行的必要，我们可以很容易地将图像渲染拆分到几个线程中（因此，也可能在几个CPU中）。由于本例中没有用到抗锯齿，所以每个像素的操作都是独立的。

这样一来，当我们看到用Web Workers实现光线追踪的例子就不会感到奇怪，比如这个：<http://nerget.com/rayjs-mt/rayjs.html>

使用IE10探查这个光线追踪例子，我们可以看出不使用Worker和使用4个Worker的显著区别：

结论

对于使用Web Workers来review或构建JavaScript代码，使之可以并行执行，并没有什么神奇或者新的概念。需要做的就是将代码中要加强的部分独立出来。它需要和页面中的其他逻辑相对独立，避免等待同步执行的任务。最重要的是：代码不能跟DOM有耦合。如果所有这些条件都具备了，那就考虑一下Web Workers。它们绝对可以提高Web app的总体性能！ ■

别把自己当个超人

——给初级程序员的一点小小建议



作者 / Ted Neward

[@tedneward](#)

Ted Neward是一名Java和.NET开发者、顾问、作家和演说家，他的著作包括*Effective Enterprise Java*、*Server-Based Java Programming*、*C# in a Nutshell*、*SSCLI Essentials*等技术图书。

有一天我和朋友Simone一起喝咖啡，期间我们聊起一些工作上的事情。我们俩都管理着一些员工，为了说明给初级职员分派任务时出现的问题，她打了一个绝妙的比方。

这就像你让他们挂一幅画，但他们从来没有干过这样的活。你明白你要做什么——只要让他们这么做就行了。事实上，你认为有些东西不用解释，因为它们太简单了。所以，当一些新手为你工作时，你说，“把这幅画挂在那里，做完了告诉我”，很好理解，对不对？但他知道应该怎样钉钉子吗？实际上，有很多细节他不清楚，他需要学习才能把画挂好。另外，还有很多事情是你容易忽略的。

首先，是怎么做。他需要什么工具？你知道工具箱里有锤子和钉子。但他不知道，他认为他有合适的工具完成任务。于是他在办公桌上找到了一个订书机和胶带座。

现在他有两种方法完成任务。他可以做很多小的胶带圈，这样两面都能粘，然后把它们贴在画的背面。这种方法看上去可能没事，而只有画掉下来的时候，你才知道他做错了。另一种方法是，他把一大条胶带缠在画上，并且把它用订书机牢牢钉在墙上。用这种方法的结果可能更糟，虽然和你的要求差不多——画挂好了，但是画面被遮住了。只要有足够的订书钉，画就能挂住。但是这样太难看了，而且也不是你想要的挂法。如果你不及时制止他，他可能会继续用这种方法挂画。

还有一种可能的情况，特别是对那种急性子的初级程序员来说。当你的老板来问你射钉枪的采购订单时，你才发现他这样做了。于是你叫来你的下属了解情况，发现他上周一直在google上搜索，阅读参考书，并向讨论组求助。他知道你想把画挂在墙上的钉子上，并且认为钉钉子的工具是高级气动射钉枪。如果他能接受你的意见，你得指出挂图和装修用的钉子是不一样的，并且告诉他工具箱里的锤子才是真正适合做这件事的工具。要是他还固执己见，就可能会有像下面这样的争吵了：

“为什么不能买射钉枪？”

“因为没有足够预算。”

“难道就没法做好一件事情吗？”

“你可以用锤子钉钉子。”

“可是我们不是应该更快更好地完成工作吗？难道我们用锤子的原因只是因为用习惯了它？眼光应该放远点。”

“我们没有足够的时间证明买射钉枪是对的。明白吗？”

最后双方不欢而散。

现在，你觉得你把工具的问题说得够清楚了。他也拿到了锤子和钉子，开始了他的工作。问题是，他还应该知道如何有效地使用它们。对你来说，熟练使用锤子是轻而易举的事情。我们会用手拿着钉子，然后大力钉下去。但对于从未见过锤子的人来说，好像用比较大且平的侧面来敲一些小东西更简单。当然，用锤柄末端也行。用楔形部分夹住钉子，然后把它钉进墙里看起来也可以。

从木工的角度看，这似乎有点低级，但它反映了使用软件工具时的实际问题。一个软件往往提供很多参考文档，但范例和习惯用法却不多。你可以买一本一千多页的参考书，它告诉你用一个软件可以做些什么，却没有用哪怕仅仅5页内容，来说明实际情况下该怎么用它。就算你有一个实例，它们也不告诉你为什么要用这种方式运行。读完本文之后，你就不会再纠结钉钉子是用锤子还是射钉枪了。

我刚开始使用XML时就遇到了这种情况。我读过的帮助文档里这样说，“用SAX解析器读取XML文件，不要用DOM解析器。DOM解析器运行速度很慢，并且占用内存过多”。后来我问过其他人，“为什么不行呢？难道DOM解析器执行效果很差吗？”

他说：“并非如此，但如果你只想获得作者和标题信息，就没必要加载一个10M的文件。”

“啊，是这样，我想把一个20K的文件内容发布为一个网页。”

“那你还是用DOM解析器吧。”

此外，还可能会出现数据交互问题。现在你的下属知道怎么钉钉子了，他做的第一件事情是在画框上钉一个钉子。

天哪！！

“不不不，你没看到画框背后的绳子吗？你应该把钉子钉在墙上，然后把绳子挂上去。”

“哦，我不知道它有什么作用。不过你只钉一个钉子？多钉几个不是更安全吗？比方说钉6个。”

“用一个就足够了，钉子多了反而不好调整。”

“为什么要调整呢？”

“你得把画挂正吧。”

“哦？要挂正吗？”

唉，又没说清楚。

现在我们开始讨论更高层次的设计问题。画应该挂在哪里？应该挂多高？他没法决定。如上文所说，它没有你想得那么简单。

你明白画不能挂在门边，因为开门时会挡住它。它也不能挂在那边，因为你要把新书柜放在那。或许你的天花板有14英尺高，画只是用来让这个大房间显得不那么空荡。也有可能这是你和“猫王”的合影，有人坐在办公桌前的时候，你可以显摆显摆。如果它是一张老照片，你必须确保它不会受阳光直射。这些都是“业务规则”。虽然你挂画的方式大同小异，但你必须考虑它们。

也有些业务规则会影响你的决策。如果画比较贵重，你得想办法把它保护好，比方说把它挂在比较难够着的地方。如果它价值连城，你得用两英寸厚的玻璃来保证它的安全，周围还得装上警报系统。要是你打算用来挂画的那堵墙非常结实，你得用钻头才行。如果墙本身比较值钱，你还是暂时打消挂画的想法吧。

这些规则可能有些道理，但它们并非那么显而易见。在某种情况下正确的方案，在其它情况下不一定是对的。你只能根据挂画的经验来了解它们。另外，你还得考虑哪些规则可能改变。确定要把画挂在这吗？这幅画会被移到别处吗？它会不会被换成另外一幅画？新的画和原来那幅还是一样大吗？

别指望新手会考虑这些，你可以适当指点他一下。你的任务是尽可能多地告诉他工作的细节。如果他聪明、好奇，他会提问，并了解这样做的来龙去脉。不过这需要时间。

如果你没有给他足够的信息，他会试着猜测。前面提到的急性子程序员这时可能就不顾规则了。你告诉他，把你的宠物狗照片挂起来，一周后他回来了，问你要不要再考虑考虑他关于石膏锯的建议。

“你为什么会想到石膏锯呢？”

“办公室的工具箱里只有木锯，不太适合锯石膏板。”

“什么？你认为只有你想锯石膏板吗？你可以在Home Depot上买一个锯子。”

“那么，好吧，我去买一把。”

“等等，你怎么会想到锯石膏板？”

“是这样，我不知道挂画最好的方式是什么，所以我上网找了讨论组里的画廊设计师。他们说，最好的方法是锯穿墙壁，做出一个框架。把画从后面放进去，这样能够保证玻璃的安全，因为你不必动它。而且这种方法比钉钉子更加美观。”

“...”

这个比喻可能有些不太切题，不过请相信我，它非常有参考价值。如果你的职业生涯中还没有遇到此类事情，你可以先看看它。

关键是，从细节技术层面到整体效果层面，有很多东西你必须知道。不能让一个新手随意猜测，不管他有多聪明。而且这和聪不聪明没关系，一切都要根据实际情况决定。可能你和他们一起工作太久了，你忘了你也有一段时间不知道这些。但是你必须详细地描述你的要求，方便他们提问。

仅仅是这样吗？

这让我想起了《帝国反击战》中我最喜欢的场景之一：Luke Skywalker遭受了失去亲人的打击，他一直在寻找自己隐秘的身世。后来他跟随绝地大师Yoda学习如何成为绝地武士。但是

Luke想学的东西Yoda却没有教他。实际上，观众们认为，Luke似乎并不了解绝地武士应该是怎样的，毕竟除了和Ben Knobi一起学习的时间，他几乎没有任何这方面的经验。而他之前并不知道Kenobi是绝地武士，直到他亲眼目睹Kenobi在战斗中牺牲。

这是这个场景中令我印象最深的画面：

LUKE：师父，移动石头很难，它和我以前学的完全不一样。

YODA：不，没有什么不同。只是你认为它们不同罢了。你必须忘掉你所学的。

LUKE：（静静地集中精神）好吧，我试试看。

YODA：不，别老想着只“试试”。要么做，要么就别做。

而《帝国反击战》和上文挂画的例子有什么联系呢？

初级程序员们应该“忘掉”他们觉得自己已经知道的东西，然后重新学习他们需要的东西。

刚走出校园的程序员有两种类型：要么干劲十足，随时准备改变世界；要么胆小如鼠，不敢抓住机会或者尝试有风险的东西，生怕被炒鱿鱼。

第一种是我非常担心的一种。他们自认为知道要做什么，且在google和互联网上搜索他们需要的资料，他们会为了挂画把墙锯开，或者会因为射钉枪和你争吵，因为他们觉得那是对的：射钉枪钉钉子的效率更高。实际上他们错了，因为射钉枪没法控制钉钉子的力量（它会把钉子整个钉进墙里），并不适合挂画。

另外一种就比较不幸了，他们没法被用人单位接纳。因为缺乏工作的主动性，没法真正学到什么东西，只能做简单的拼写检查或是类似于行政助理的工作，甚至可能一辈子都耗在HTML网页上。他们会抱怨、厌倦这份工作，最后跳槽到邮政、市场销售行业（也许更糟，做个推销员）。不管怎样，这对他们没有好处。

这不得不让我深思：初级程序员应该做些什么呢？如何让一个程序员从入门到精通？初级开发人员应该怎样避免向这两个极端发展呢？

结论就是：初级程序员们必须学会“问”。问了，就必定会有收获。

看看急性子的初级程序员的例子：如果他敢于发问，“老板，我从来没挂过画，我该怎么做？”，就不会有射钉枪、胶带、石膏锯这样的问题了。作为一个过来人，你应该知道他没有你那么经验丰富。

老手们没法了解新手们不会什么，但是帮助他们认识到“哪些不会”是非常重要的。这种关系就好比之前提到的绝地大师和绝地学徒之间的关系，也可以说是西斯领主和西斯学徒的关系，或者是千百年来人类历史上的各种师徒关系。

最关键的一点呢？

从某个角度来说，我们都是初级程序员。就算你有四十年各个平台和嵌入式系统的C++开发经验，但当涉及关系数据库、Nosql或是Java和JVM，或是C#和CLR的时候，你依然是一个初级程序员。就像涉及原力，或者如何像父亲一样成为绝地武士并拯救

查看英文原文: [Unlearn, young programmer](#)

译者 / 万琦 ([五指琴魔](#))
刚步入社会，处于迷茫期中的90后一员。计算机专业毕业生一枚，不堪打杂之苦毅然踏上考研不归路。天性爱折腾，任何东西都想花点时间研究。正以“成为一名合格的程序员”为目标奋斗。”

宇宙的时候（包括两集之后才发现那个漂亮女孩是你妹妹），你也依然是个新手。

现在你知道该怎么做了。

为你自己找一位老师（也许现在更准确的说法是“导师”），向他们学习或提问是非常有用的。然后，让你的同事或是朋友中的初级程序员也这么做。他们不一定会接受你的建议，但仔细想想，你在这个年纪的时候，不也希望有一位大师来教你吗？

你是想成为爱抱怨的Luke Skywalker，还是想成为绝地武士Luke Skywalker呢？Luke失去了一只手之后才明白Yoda远远比他聪明，应该多请教他，而不是告诉他“你做错了”。

你还想败掉多少项目呢？还是老老实实承认自己不是无所不能吧。 ■

出版的未来

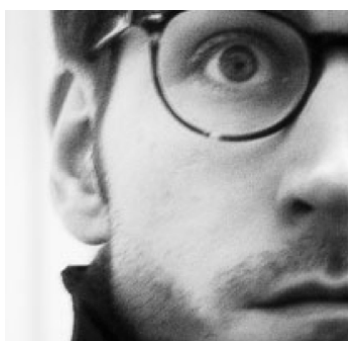
30天，24,000美元

60天，一屋子的书

90天，一种新的出版方式

创意帮创意

——一个关于图书、出版和筹款的故事



作者 / Craig Mod

[@craigmod](#)

身兼作家、出版人、设计师及开发者数职，关注图书、出版和叙事的未来。前Flipboard成员，亦是Art Space Tokyo一书联合作者及设计师。

出版业的明天在哪里？

我们都有kindle或者iPad，iPhone或者android手机，按需打印很方便快捷，每个人都在amazon一类的电子商务网站买东西，所以，在现在这个时代里，很多书只需要轻点鼠标就可以购买和阅读。

现代出版业有非常多的问题急需重新探索，比如，什么样的才算一本好书？纸质的还是电子的？生产所需的资源（时间，资金和精力）是多少？读者怎么才能知道并购买它？

之所以会有这么多问题，是因为在这个时代，现代出版业有很多种可能。图书印刷、推广和发行成本上的减少，都极大降低了潜在图书发行者的门槛。

我们可以选择媒体。

我们可以建立读者社区。

并且现在，我们还可以用Kickstarter来筹集资金。

我在这里讲的这个故事跟图书，出版、筹款和种子资金相关，但我希望它能改变人们对这些话题的传统概念。当然，我也希望它成为一个可供参考的模版。

2010年4月份的时候，我和Ashley Rawlings利用社区筹款找到了差不多24,000美元，并使用这些钱给我们的书《东京艺术空间》注入了新的生命。我现在要讲讲我们做了些什么，并且为什么要这么做，希望它可以鼓励其他有想法、有热情并且有执行力的人做同样的事情，把好的、精心考虑过的事情带到这个世界上来。



这个故事开始于日本标准时间2010年3月29号，22点18分，坐在布鲁克林公寓里的一个女人推倒了多米诺骨牌的第一块，她的65美元的认捐，结束了为期一个月的筹款，这笔钱后来变成了一屋子的精装书，一个出版的智囊团，和一种iPad时代图书出版实验方式。哦，还有一篇这样的文章。

东京艺术空间项目

从一本书开始。

《东京艺术空间》是一本介绍东京这个城市里隐藏的画廊和博物馆的书，由我跟编辑、合著者Ashley Rawlings 在2008年合作完成。这本书在一年内卖光，以后一直没有重印。

这本书对我们两个很重要，因为在出版发行的过程中我们花了很多的心血。同时这本书对我个人又特别重要，因为在这个项目中，我表达了自己关于出版、印刷以及将图书对象化的理念。

写“[iPad时代的图书](#)”的时候，我就一直想起《东京艺术空间》，它是一个完全成型的文学对象，它是精心设计的，在编辑、设计和物理工艺之间达成的一种平衡。虽然它并不完美，但是当我完成时我非常引以为傲，直到现在还是。

2009年我花了很大一部分时间思考出版业中数字和模拟印刷怎样融合，而今年（2010年），我开始为这个主题公开演讲并撰写文章。我在独立出版方向进行了试水，然后我想，也许我可以鼓励其他有相似想法的人，反思一下我们陷入的思维模式——我们已经用大众市场的出版商而不是别的什么人的脑子在思考了。

2010年的春天，斗转星移，我买回了《东京艺术空间》的出版权。这是非常令人激动的事情，我把当成是我关于出版理论的实践机会：一本实体书既能发挥实体印刷的长处，又能发挥电子书在多样性和确定内容上（访谈、地图、随笔、插图、表格数据等等）的优势。

当我获得出版权以后，剩下的唯一问题就是，钱从哪里来。



你的筹款伙伴：Kickstarter.com

我决定用Kickstarter。

2009年我第一次听说它的时候就想到了，具体原因我也很难说明白。但这并不重要，这只是我获得《东京艺术空间》版权过程中的一个小环节。

Kickstarter.com是一个筹款网站。你创建一个帐号，说明你的项目，然后就可以设定在多久的时间内想得到多少钱，这个世界上任何一个有amazon帐号的人都可以捐钱资助你的项目。认捐是分等级的，不同的等级有不同的奖励。如果你的项目在设定的时间期限（不可变）内没有达到预先设定（不可变）的捐助，那这个项目就失败了，捐助人不用付钱。如果你在规定时间之前就完成了筹款目标，那你还可以在剩下的时间内继续筹款。这个系统有些很有趣的规矩。

捐助者一般不会亏，如果你没有完成项目，他们不用付钱。如果你完成了项目，他们既可以得到所属等级的奖励，又可以得到资助别人从无到有完成一个项目的成就感。

Kickstarter的筹款流程可以完美运作，有赖于Kickstarter使用amazon作为支付系统，在这个过程中，amazon扮演可信的第三方，如果达到筹款目标，一到规定时间，捐助就会自动生效。

创建项目的人，在设定筹款目标的时候，一般需要从以下三点来考虑：

- 准确确定你要做的事情的最小目标
- 计算一下要完成这个最小目标需要多少钱
- 最后，考虑一下社交网络的力量，也就是说，你觉得，实际上你能找到多少钱？

微种子资金

设定Kickstarter项目目标的时候，我的想法是发行足够多的书来产生丰厚的回报，然后利用这些回报扩大这本书的规模，或者用于其他相似的出版物。

我从来不是只想卖掉几本书，如果这个项目只是完成了《东京艺术空间》新一版的印刷和销售，那它远远没有达到我的期望。相反，我希望它是个起点，可以引出和《东京艺术空间》有相似精神的更多项目，并且能够成为一种发掘数字图书、寻找资金并启动出版的方式。

也就是说，我把它视为微种子资金。从这个层面上讲，如果不是在Kickstarter筹款，我们就会和网络上所有可能的机会失之交臂。

在Kickstarter上，人们是在预购你的想法。当然，他们也会买有形的商品，比如CD、电影、书等等，但预购想法更有意义的地方在于，他们基于对你的信任才给你钱。如果你发现之前的想法过于简单，事实上可以做得更好，完全可以按照新的想法去做项目，超出之前的目标设定，不用放弃任何好的想法。

这种不用放弃所有权的微种子资金，就是Kickstarter资金的潜力所在。

确定认捐等级

确定认捐分级的第一步是分析其他成功的项目。我收集了2010年3月份收入前20到30个项目的数据，然后计算出每个等级认捐

的数目，然后试图在每个等级认捐的数目和该等级所有认捐资金总和在所有收入中所占比例之间找到平衡，下面就是这些数据，资金总和所占比例排序前五的等级被黄色高亮标出：

等级金额	认捐数目	总金额 \$	所占比例 %
\$10,000	2	\$20,000	5.58%
\$7,500	1	\$7,500	2.09%
\$2,500	6	\$15,000.00	4.19%
\$2,000	2	\$4,000.00	1.12%
\$1,500	1	\$1,500.00	0.42%
\$1,000	12	\$12,000.00	3.35%
\$750	9	\$6,750.00	1.88%
\$500	60	\$30,000.00	8.37%
\$300	8	\$2,400.00	0.67%
\$250	92	\$23,000.00	6.42%
\$200	25	\$5,000.00	1.40%
\$150	142	\$21,300.00	5.94%
\$125	14	\$1,750.00	0.49%
\$100	586	\$58,600.00	16.35%
\$80	20	\$1,600.00	0.45%
\$75	2	\$150.00	0.04%
\$60	40	\$2,400.00	0.67%
\$50	1,699	\$84,950.00	23.71%
\$35	14	\$490.00	0.14%
\$30	489	\$14,670.00	4.09%
\$25	1,253	\$31,325.00	8.74%
\$20	134	\$2,680.00	0.75%
\$17	94	\$1,598.00	0.45%
\$15	248	\$3,720.00	1.04%
\$12	37	\$444.00	0.12%
\$10	352	\$3,520.00	0.98%
\$5	391	\$1,955.00	0.55%
Totals:	5,733	\$358,302.00	

当然，这个数据也并不是完美的（比如，并不是我看到的每个项目都用了同样的分级），但它至少很好地说明了，人们喜欢什么样的价格范围。

50美元等级的认捐最多，在所有收入中占到了将近四分之一。令人吃惊的是，100美元的等级以不大的差距位居第二，占到了将近16%。25美元等级的表现也不错，但从这个数据中得出的压倒性的结论是，人们不介意为他们喜欢的项目支付**50美元或更多的钱**。

同时值得深思的是，超过100美元的250美元和500美元两个等级，和其他较高的等级比，也有着较高的认捐比例。

低端的等级，少于25美元的，在统计意义上是不太重要的，在所有的认捐中勉强只占到5%，**我建议忽略它们**。当然，这也取决于你的项目，也许你有非常好的理由为5美元分个等级。总之，最重要的是，这些数据表明了人们更愿意支付25美元。

划分太多的等级有可能会把你的支持者们弄晕，我看到一些项目有十几个等级，不过请别这么做，别让你的支持者陷入左右为难的选择困境。搞得简单点，实际操作中，设置的等级不要超过5个。

平均认捐金额大概是62.5美元，这个数值和我估计的《东京艺术空间》的生产加运输成本非常接近，看来把每本书的价格设在65美元是个很正确的决定。

综合以上数据分析，下面是我在“东京艺术空间”这个项目中最终确定的分级及其奖励：

等级金额	认捐数目	总金额 \$	比例 %	奖励描述
\$25	28	\$700	3%	《东京艺术空间》 电子版PDF文档
\$65	155	\$10,075	42%	以上+ 实体书
\$100	64	\$6,400	27%	以上+ 名字出现在书的鸣谢列表中
\$250	11	\$2,750	12%	以上 + 签名书 + <i>tenugui</i> 限量版
\$850	4	\$3,400	14%	以上 + Nobumasa Takahashi原作
\$2,500	0	\$0	0%	以上 + 东京一日游

这是一种很好的传播方式。25美元的分级虽然在统计数字上处在弱势，但它使以前购买过书（第一版）的人继续支持这个项目。从某种意义上说，它主要是用来加强支持者社区，而不仅仅是单纯提供资金。这点很重要，因为你不仅是在筹款，也是在筹款的过程中组建支持者的社区。

毫无意外，65美元轻松地占据了统治地位，我们的数据早就揭示了这一点。100美元提供了一个好的“升级”等级（我怀疑几乎所有捐了100美元的支持者都宁愿选择65美元，而不是250美元）如果我早点解释一下*tenugui*的奖励，250美元等级的表现应该会更好点。850美元的等级，是给那些Takahashi-san作品爱好者的一个不顾一切的支持方式，或者为那些有能力给一个项目如此丰厚支持的人准备的。

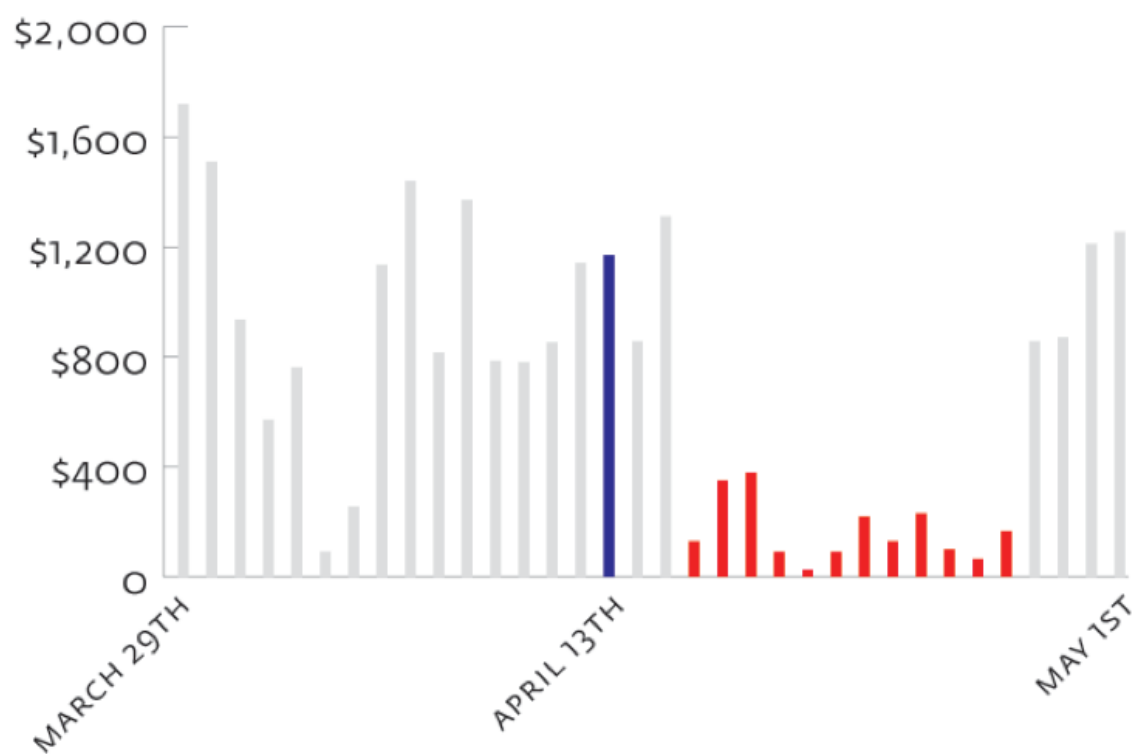
我那些让人兴奋的自行车、咖啡、博物馆以及建筑、烹饪东京一日游，就是疯狂的2500美元等级，一个也没卖出去。不过，它出现在那，只是为了让850美元等级的takahashi的作品看起来更合算，我不是真的想要当导游。

认捐分析

人们总是在在一件事情刚刚开始或者马上就要结束时，参与最为踊跃，在中间阶段往往会失去兴趣。考虑到这个因素，我试图把“东京艺术空间”的认捐周期设成只有三个星期。第一个星期是初始推动，第二个星期是缓冲期，然后第三个是最后的推动。第二个星期的缓冲时间只是我们高声叫喊“开业酬宾啦”和“跳楼大甩卖”之间的一个心理停顿。

但是最终，我们把认捐周期设置成 5 个星期，不过现在看来，四个星期就足够了。当然，这个多出的时间让我们有余地来寻找博客和网络杂志（发表一篇帖子往往需要几天甚至几个星期），所以，有一个12天的死亡时段（用红色标出）。如果还有机会，我在那个星期的努力方向，将会是覆盖更多网络媒体。

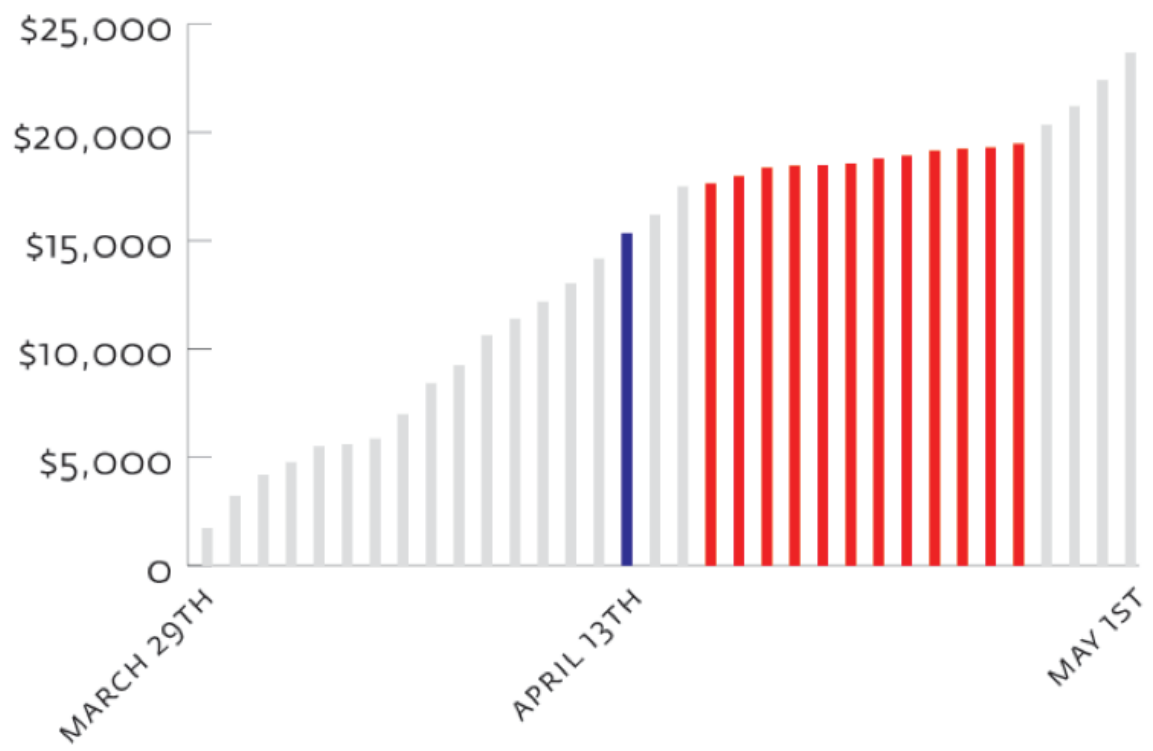
下面的图表显示了活动期间每天的认捐数目。



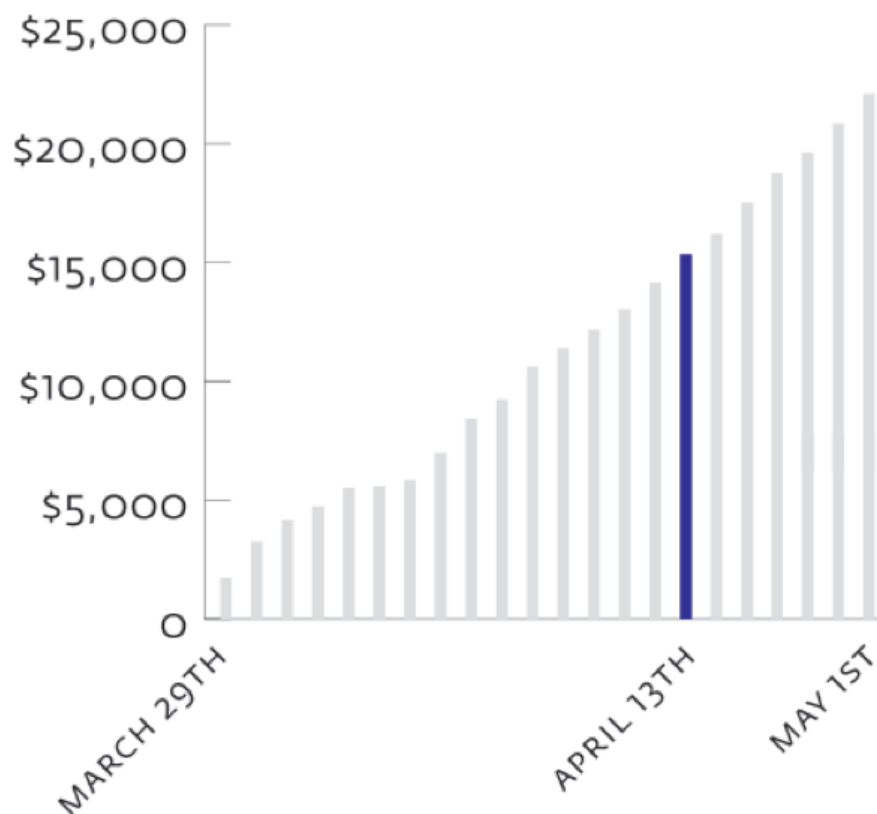
收入最高的是第一天，3月29号1720美元。最差的一天是4月20号，25美元。整体来看平均每天的认捐金额是695.88美元。

我们在4月13号达到了融资目标，这时活动开始仅16天，然后就跌进了无聊的既不新鲜也不近结束的阶段，导致活动结束后出现了12天的捐款死亡区间。

从图表中可以看出，到达筹款目标里程碑前，我们的收入有稳定持续的增长：



实际上如果剪掉死亡区间，重新调整总数，可以得到如下图表：



在死亡区间的12天里。我们只收到了1976美元的捐赠，平均每天165美元。这个数字远远低于接近700美元的日平均收入（如果没有死亡区间的话，应该是几乎1000美元每天）。

为什么这么在意这12天呢？那2000美元不是2000美元吗？当时是，但拿不到钱就没办法把项目进行下去，所以效率也是非常重要的。活动效率越高，就越容易让人们保持强烈的兴趣。项目徘徊不前的时间太长，会有支持者们热情衰退的风险。所以，那2000美金如果能压缩到一个更短的时间中，效率就会越高。

推广的策略和行动

我和Ashley Rawlings（《东京艺术空间》的编辑和合著者）主要用了下面三种资源来推广这个Kickstarter项目：

- Twitter和Facebook
- 大量的个人邮件列表，里面有很多设计界和艺术界的朋友
- 在线媒体：一些顶级的艺术和设计博客和网络杂志

推广战略大概是这样的：

- 持续在Twitter和Facebook上更新信息。
- 在筹款初期和结尾，用邮件列表里集中宣传。
- 中间部分，在邮件列表里进行持续渐进的状态更新，这是个非常好的在媒体上宣传的机会。
- 寻找相关的在线媒体，发布项目的情况，从始至终贯穿整个项目周期。

Twitter

在Kickstarter上发起这个项目的时机真是千载难逢。全世界的爱好者突然很想听听东京的那些人对书籍的未来说些什么，我的方向上（主要是在Twitter）出现了很多这样的新耳朵，于是我有了新的被吸引过来的听众，他们很可能对“东京艺术空间”的项目感兴趣。跟项目有关的tweets尽量保证少但贴近主题。这么做主要是为了不让我的twitter看上去全是“东京艺术空间”项目的更新

（如果实际上是这样的话，我很抱歉），但是可以温和地提醒人们，项目保持着很好的势头并且依然在寻找认捐。

通过Topsy，可以回顾整个活动中的tweet/retweet的时间表。

日期	活动区间	前一天的认捐 \$	当天的认捐 \$	后一天的认捐 \$
4/ 6	刚刚开始! [a]	\$1,135.00	\$1,440.00	\$815.00
4/8	刚刚开始! [b]	\$815.00	\$1,370.00	\$785.00
4/15	中间更新	\$855.00	\$1,310.00	\$130.00
4/ 30	还剩三天	\$870.00	\$1,210.00	\$1,255.00

邮件列表

过去六年我们都在设计和艺术领域工作，我们利用了这期间建立的大量联系人列表。下面这个表列出了发出邮件当天、前一天、后一天收到的认捐数目，以及当时所属的活动区间。

可以很清楚地看到，每发送一封邮件（4月6号1000个收件人，4月8号1000个收件人，15号2000个收件人，30号2000个收件人）会让销售显著增长。好奇我们发了什么样的邮件吗？下面是发送的邮件：

- 4月6号、8号：[刚刚开始[a/b](#)]
- 4月15号： [中期更新](#)
- 4月30号： [还剩三天](#)

博客/在线媒体

宣传项目的策略很简单。寻找那些与主题相关的、有重要有影响力的媒体，然后分别给这些博客、网络杂志或者报纸写邮件，着重描述他们可能感兴趣的方面。通常情况下，我会给我读了很多年的博客写邮件，所以对于我来讲，参考他们以前的文章然后分别定制这些邮件内容是件既琐碎又有趣的事情。

不管你做的是什​​么，都不要漫无目的地给媒体发邮件。要深思熟虑，你的目标是打动那些编辑和社区众生，让他们对你的工作产生兴趣。不要给所有大牌博客群发邮件。对的博客就算发一篇帖子，也比在高流量但跑题的博客上发十篇帖子有用100倍。你要的是参与的用户，而不是眼球。

为了Kickstarter的“东京艺术空间”项目，我和Ashley在超过12家相关的媒体上做了宣传，包括以下列出的：

时间	媒体	社区
3/30	37Signals, SvN	创业者
3/31	Spoon & Tamago	设计, 日本
4/1	Kickstarter Tweet	Kickstarter
4/1	Hypebeast	设计
4/5	Viewers Like You	设计
4/9	Superfuture	设计
4/9	Complex	设计
4/9	Street Giant	设计
4/10	Notcot	设计, 艺术
4/11	Subtraction	设计
4/13	We Jet Set	设计, 旅行
4/13	Limited Hype	设计
4/14	Jean Snow	东京, 设计, 艺术
4/16	PSFK	艺术, 设计
4/17	Nonaca	艺术, 东京

可以看出，我们的活动临近结束时的宣传有些仓促，显得头重脚轻。如果能在4月17号之后覆盖更多的媒体，就更好了。现在看来，我们应该早点开始筹备媒体的宣传（当时几乎是和Kickstarter活动同时开始的），并且试图和媒体商量一下帖子发表的日期。

不同的做法

我们最大的错误就是把筹款目标设得太低。如果一个Kickstarter项目还未达到时间期限，就已经达到了筹款目标（可以看前面关于资金的章节），就会变得不那么激动人心，并且失掉一些“赌一把”的成分。对所有Kickstarter项目来说，定一个在分配的时间内可以刚好达到的筹款目标是比较理想的。

刚开始的时候，我们觉得筹到15000美元都很困难，结果每天有500美元(相当于九本书)的认捐。不需要再多但是也不能更少，15000美元是最低目标，出书的数量值得我们去付出。我们的想法是，多的每一美元，都要用来扩大项目的规模。如果筹集到24000美元，就可以谨慎地增加印刷量，现在我可以很开心地说，这个项目已经超越了之前设定的目标。

如果我说，你应该把筹款目标应该设得高一些，希望这听起来不会显得太贪婪。记住，最大化筹款目标也就意味着最大化社区参与度。你不只是在筹钱，同时也在建立一个更强大的社区。

24000美元能够干什么呢？我们设法使精装丝印的《东京艺术空间》的印刷量接近翻了一翻，更新了书里的地图、餐馆和咖啡馆推荐，做了充分的重新编辑，启动了智囊团“PRE/POST”，在东京管理订单和运输，运出了超过300本书，还在东家一家非常重

要的艺术书店举办了启动仪式，并且写了这篇文章。所有这些都在三个月内完成，我认为做得实在是太好了。

这篇文章为该实验的第一章划上了句点。下一步，我们会全力创建这本书的数字版本。当然，也一定会分享这一路上我们会学到的东西。

瞄准1000000美元

总结一下：

- 24000美元一个月
- Kickstarter = 简单而又有吸引力的第三方托管
- 以微种子资金的方式去思考，不要认为是一次性的钱
- 等级越少越好
- 人们更愿意捐赠25美元及以上
- 做好媒体宣传的计划，避免空白区域
- 聪明地推广，关注高质量社区
- 勇敢一点，把筹款目标定到可能的最上限

就算在五年以前，我们也还无法想象可以用社交媒体为一本书的重新出版召集24000美元。但今天，我们做到的不仅是这样，我们能持续地为一本书定价，发起出版智囊团，直接把书卖给我们的读者，冲击传统的发行渠道。毫无疑问，我们正处在一个为未来的出版业塑形的时代——它会怎么开始，由谁开始，以及以什么样的形式开始。

我希望这篇文章至少可以帮助另外五十个创作者完成类似的项目。五十个创造者做到像我们一样浅层的成功，就意味着，超过1,000,000美元的资金流向了那些有创造性的、并对社会有意义的项目。所有Kickstarter上的成功项目都无可争辩地证明，实现这个目标是有可能的。让我们知道你创造了什么吧，我们渴望听到这样的消息。

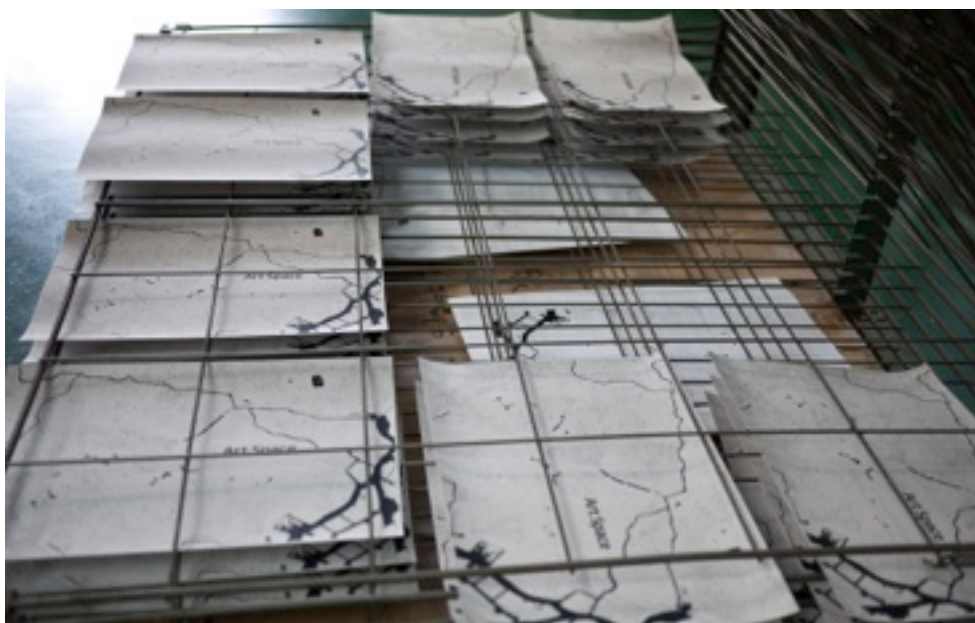
番外故事

丝印



在这幅图中，你第一眼看到的应该是中村先生善良的眼睛，然后是他的祥和，他是丝印之神。他和他的助手总是非常幽默。每当我到访并研究他们的工具，他们给我一个半小时，我就会非常受宠若惊。他们开心地回答了我们所有的问题，并骄傲地展示了他们特有的工艺流程。

这里有让你印象深刻的设备，他们可以处理各种各样的任务，小到“东京艺术空间”这样的项目，大到高速铁路的广告牌。他们有飞机机库大小的暗房，曝光灯泡象个微型的太阳。他们的产品是极其丰富的。



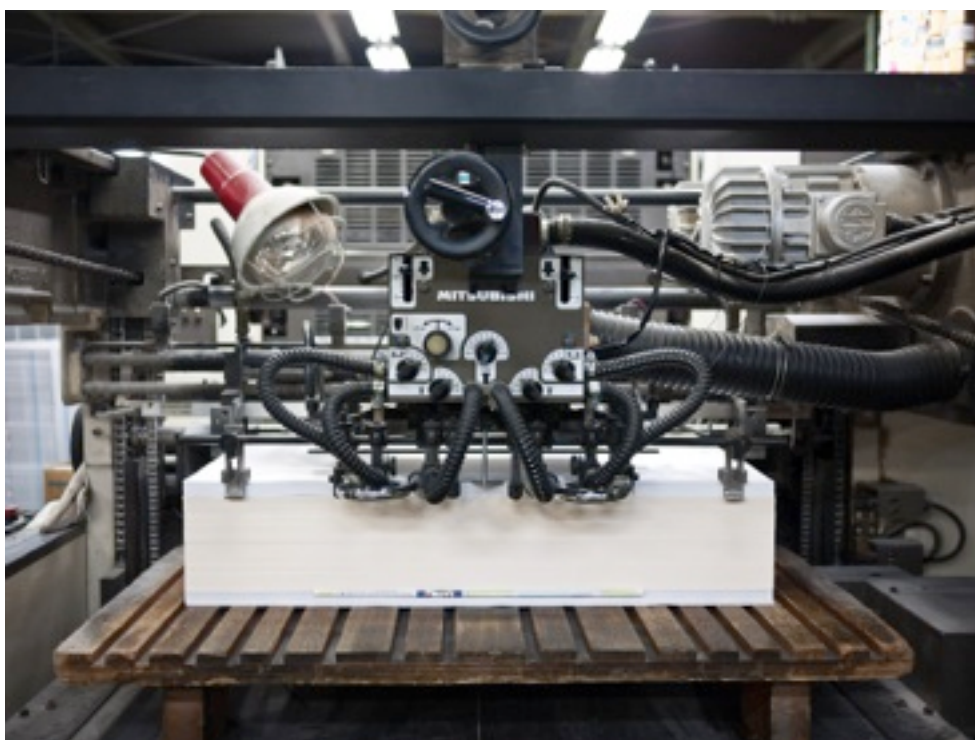
非常遗憾的是，他们坚决不同意我给这些设备拍照。尽管如此，我还是偷偷的拍了唯一的一张——在烘干中的《东京艺术空间》的封面。为了使每个封面都有相同的效果，封面是非常仔细地手工印刷的。看着这个柔弱的步骤，我更加感慨这是个奇迹，每本书里都蕴藏着的奇迹。

中间的时候，我问中村先生，他们是否感觉到出版业的不景气而导致的压力。“没有”中村先生说。“为什么？”“因为尽管出版业形势动荡，设计师们却比以往任何时候都更关注封面和海报的特殊印刷工艺，也就是我们的经营专业。实际上，我们的产量在最近几年有所增长。”中关先生回答。

无论这答案是真是假，跟我都没有太大的关系。这回答让我觉得开心，是因为我们可以继续使用中美先生的设备。大众市场的出版理念是不生产丝印和布面的书，但Kickstarter允许我们逆势而动，忽略大众市场的价格要求，生产小众产品，生产我们认为应该存在的书。所以，我现在坐在了伟大的、屡屡获奖的本地印刷者面前，因为我们支持了他们的工作，作为回报，对于我们的项目，他们也表现出了十分的殷勤和深深的赞许。

他只穿着内衣

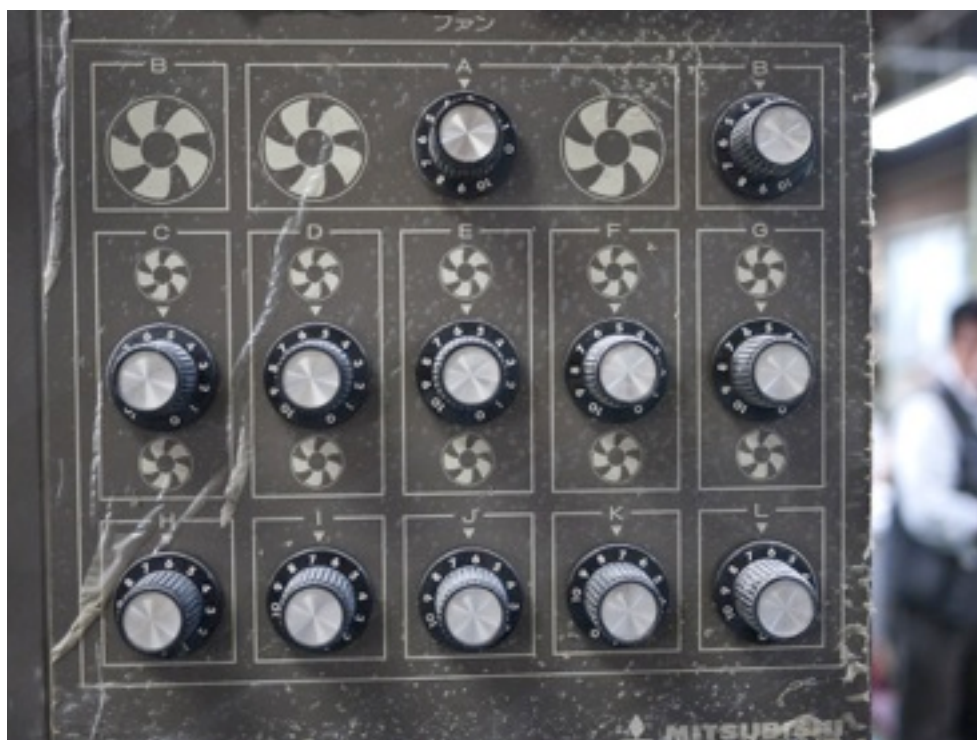
他差不多有80岁，只穿着内衣站在那儿，一根香烟在嘴唇上翻滚。我挥手致意，他报以微笑，慵懒地操作着宽大的打印机上的表盘和推杆。



八小时结束了，这是我们今天的最后一站。这个小店藏在西新宿西边的某个不知名的街道的某个不知名的公寓大楼里。

生产这本书的设备主要包括：四台印刷机，一台装订机，一个储备和实施组。我今天看到了所有打印机和这个——在这，非常不协调，在烟雾弥漫的复式公寓的第一层——是obi（和服上的宽腰带），或者说是腰封印刷机。它们会在便宜的纸上印一种颜色，然后这张纸会被竖着裹在封底上。

日本印刷界中总是弥漫着手工艺的粉尘。这些设备很脏（用得很多），我刚刚还在想，这些书的诞生过程竟然如此原始。但是，这就是事实。



一个染着黄头发、戴着厚眼镜的小伙子负责机器的输出，大概一分钟可以产出一百个腰封。他把粘着墨水的手伸进打印机，拔出一个板子检查墨水扩散的情况。看来状况不错，他比较满意，然后转动了几个表盘，眨了眨眼，挥手向内衣先生发出了“加快输出”的信号。

正文部分

墨水浓度是我们今天的主要问题。我不知道那个技术员的名字，但是他正好负责正文文字和插图的墨水浓度。Kohiyama-san，印刷协调员，就在我的身旁，我头一次意识到他非常明白我们想要的那种效果。“背面看起来不错，但是你需要把前面多打开一点”，他在吵闹的印刷机器后面喊。我同意，就没有说话。技术员调整着大面板上的墨水饱和度等级，然后通过系统发出了新一波的印刷测试品。



一个窍门就是减少黑色。插图家Nobumasa Takahashi的作品非吸引我的原因有很多。我喜欢厚重的、对比强烈的水墨画，而这正是他的强项。当然，我热爱他在创作过程中表现出的坚持与坚韧。他花了非常多的时间调配墨水，然后用正确的方式晾干。他用水磨墨，调到刚刚好的浓度，然后在刚刚好的温度下，以刚刚好的速度让它们变干。他能发现那些别的插画家忽视的细微之处，他可以创作出金属微粒的观感，渲染上也带有此类风格的插画一般没有的深度。

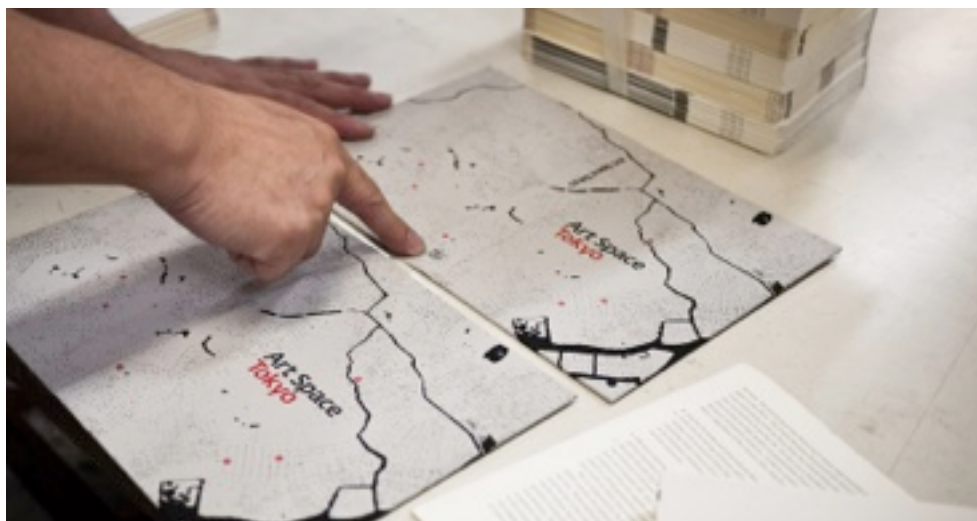
我们想要保护这种深度。

我们开始调试。新的印刷品会跟前一次进行对比，然后再跟第一版进行对比。我们小心翼翼地每次只调整微小的增量，如果插图的细节层次过于丰富，就会降低正文文字和地图的黑度。这样的印刷要在美学和实用主义之间掌握平衡，幸运的是，这个我们很在行。



装订业务

每本书都会经历从排版软件里的设计稿到真正印刷成书的诞生过程，而装订就是这个神奇过程的最后一步。我们的装订地在本乡，就在皇宫的西南边，有一间旧印刷厂房。这一区有很多夹在住宅和公寓中的小仓库，但现在的规模比以前缩水了不少。空白的告示牌和仓库纷纷转成了与印刷无关的用途，看起来像个工业太平间。很明显，这一区已经有点过气了。



现在是五月份，今天真正开始热起来。Kohiyama-san和我沿着弯弯曲曲的小巷走过几所房子，我们不停地出汗，循着“克朗-克朗-克朗”的机器有节奏的切割和装订的声音，来到了目的地。

我们从炽热的阳光下，走到凉爽的黑暗中。这里几乎每个机器都配有两个人，Kohiyama-san——我的长期搭档，也是无所不能的解说员，凑到我身边感慨这有多难得，他小声嘟囔说“你太幸运了，在这种地方可以找到一个能同时操作两台机器的人”。

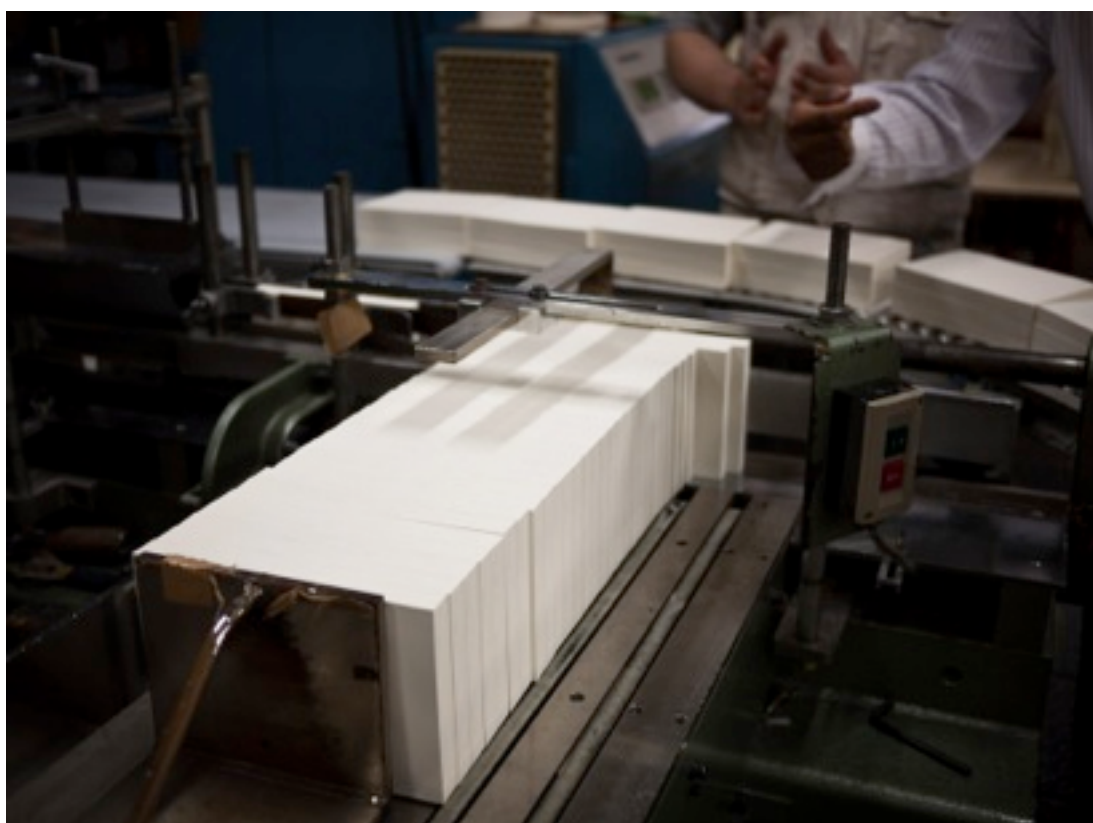
所有的打印店都会为完美的作品而感到骄傲，日本有着十分精湛的铅印工艺，这些印刷工和装订工也都是老师傅。他们对像“东京艺术空间”这样的小项目都很认真和上心，这点让我很感动。

每本书的每个步骤都经过反复核对。整个过程几乎完全自动化，只有一点需要人工的干预，流程编辑（如果你愿意这么称呼他们）会轻轻地按一下，引导半成品的书安全进入下一个生产阶段。



我站在生产线的末端，看到书被堆向一个大的木质托盘，每次10本。即使是这样一个往托盘上堆书的环节，都被精心设计和认真对待，每两本书都是反向摆放的，这样堆高了以后，就不会因为装订的重量压在一侧而失去平衡。

书的数量不断增加。托盘现在已经装满了四分之三。我走到每个工作人员面前，感谢他们，拍了最后一张照片，心里默默感谢kickstarter和日本印刷社区的支持，没有他们，我不可能做到这些，然后走回到了酷热的阳光中。 ■



译者 / 吴瑜

[@带围脖的enya](#)

先后供职于法国电信、惠普等公司，从事手机平台和移动服务及应用的开发和产品、项目管理工作。热爱新鲜事物、方便美好人们生活的优秀产品，关注所有软件开发相关、移动产品相关、敏捷相关。创新拥护者，谨慎创业者。

当红书榜

致力呈现公正、客观的数据

大家都在看什么？图灵
2012年6月图书销售榜
让你一窥真相！

1	推荐系统实践	新书
2	你不可不知的50个建筑学知识	新书
3	七周七语言：理解多种编程范式	
4	远大前程：从软件新手到行业大牛	新书
5	他山之石：TechStars孵化器中的创业真经	新书
6	高扩展性网站的50条原则	新书
7	写给大家看的设计书（第3版）	
8	C++ Primer中文版(第4版)	
9	Go语言·云动力	新书
10	你不可不知的50个物理知识	
11	黑客与画家：硅谷创业之父Paul Graham文集	
12	Android 平板电脑编程基础教程	新书
13	CCNA学习指南（640-802）（第7版）	
14	金领简历：敲开苹果、微软、谷歌的大门	新书
15	简约至上：交互式设计四策略	
16	Unity 3D游戏开发	新书
17	软件管理沉思录：SEI的项目管理、人际沟通和团队协作	新书
18	JavaScript高级程序设计（第3版）	
19	HTML5程序设计（第2版）	
20	演讲的艺术：演说大师教你如何打动听众	

好书妙评

致力呈现公正、精彩的评价



《[七周七语言：理解多种编程范型](#)》

在众多"架构师"层出不穷的今天，好多程序员以架构师为目标，尤其是C#、java程序员，眼里只有面向对象、架构，仿佛程序员中的世界就那么几种颜色。大师的眼中也是这样么？Bruce Tate的回答是不。他通过七门语言让你感受程序世界的多姿多彩，通过和电影角色结合，让电脑中的字符鲜活了起来，和程序有关的事不再有机械的质感，平添了生活的气息。你得学会和8位仙女沟通，他们具有不同的性格仪态，Ruby是贴心御姐，Io是调皮萝莉，Prolog是聪明木讷的管家婆，Scala是具有多重性格的倔强女孩，Erlang是精灵古怪的双胞胎，Clojure是想把中华武术在现代发扬光大的武者，Haskell是冷艳的知性美女，和她们经过了精彩的七周之后，我想你一定会精通第八门语言...

——今夜不关心架构，只谈语言 [更多专家审读意见](#)

你可以在周末舒服地阅读本书，这就像是品酒，将眼前的一切缓缓放下。花些时间学习这些语言吧。使用七个星期，或至少花几个星期。深入体味这些语言，并决定你要继续深入了解哪些语言。这将是一个有趣的旅程，当然值得你花时间。

—— 编程语言爱好者的第七天堂 [更多国外读者评论](#)



[算法（英文版·第4版）](#)

译者 / 谢路云

[@一盆花](#)

Robert Sedgwick和Kevin Wayne的《算法》（第4版，由Addison-Wesley于2011年3月出版）是我读过的最棒的计算机科学方面的书籍之一。它应该是所有程序员以及计算机科学专业的学生的必读书籍——它的目标是覆盖“每个程序员都应该了解的50个算法”。下面我就来说说为什么我觉得这本书是如此地优秀。

《算法》包含了具体的源代码（基于一个Java的子集），这和它的主要对手——由Cormen、Leiserson、Rivest和Stein (CLRS)完成的《算法导论》(*An introduction to algorithms*)——非常不同。这一点非常重要，因为它意味着学生们可以使用这些代码去解决许多真实的问题。这些算法产生了从网络搜索到基因组学的许多有趣和令人激动的应用，这些应用也贯穿了全书。

（这本书的网站上提供了所有的源代码和数据。）

对真实代码的一种很自然的担心是，它们可能会影响对概念的理解。但在这本书中，作者通过精心定义的抽象数据类型（队列、背包、哈希表、树、有向无环图等类）赏心悦目地创造了许多既有可读性又非常准确的实现。

使用真实代码的另一个好处是迫使你解决一些容易被忽略却十分重要实现细节。例如，大家都知道并归排序需要辅助性内存空间。在CLRS的伪代码中，作者在他们的并归函数内部分配了临时存储空间。但在实践中，仅分配一次临时存储空间并将它作为一个指针传递给并归函数（或是将它作为并归排序类的一个私有成员）将会高效得多。这种重要的技巧你又可以从哪里学到呢？

除了代码的展示之外，本书也用清晰的语言解释了这些方法。本书非常与众不同的一点就是包含许多详细的例子来说明这些算法

在处理真实数据时的行为和表现。（除非你真的实现了这些算法，否则是很难得到这些数据的！）

这本书的另一个优点是它严格遵守了软件工程的最佳实践：先写API，再写单元测试或是实现一个使用该数据结构或算法的应用（用例），最后才考虑应该如何实现这个API。另外，本书在许多时候还讨论了同一API的多种实现，它们在简洁性、速度和内存使用上的折中都各有不同。

对于数据结构，使用“类”是很自然的，但对于算法作者也采用了这种描述方式，特别是图算法。这使得算法能够进行预处理并保存一些内部状态，然后再为使用者提供服务。这种方式比传统的无状态的函数式的算法更加通用。

这本书的每一节都有大量的练习，分为“简单”、“提高”和“实验”三种，它的网站提供了部分练习的解答。

这本书的一个特别之处是除了理论之外，它还含有许多经验性的算法题目。它展示了算法的不同实现对于不同规模问题的实际运行时间，并用这些数据作为传统理论分析的补充。

全书的组织非常好，前面介绍过的内容（和代码）会在后面得到多次应用（例如：堆→优先队列→最小生成树的Prim算法）。书中的话题也会越来越高级。因此读者最好顺序地阅读本书，每一页。

Kevin P. Murphy（加拿大不列颠哥伦比亚大学计算机系教授）

[阅读更多好书妙评之《算法》内容...](#)



《罗素的故事》

在读这本《罗素的故事》之前，如果要说说罗素，我没法说出超过100字的内容。这本围绕这罗素讲述数学、哲学历史的漫画小说，让我捧起就难放下。

刚开始看时打4星，扣分在未知原版是否全彩，封面作者介绍有绘图Alec Papadatos和着色Annie Di Donna两位艺术家，封底介绍后者的作品包括卡通片《巴巴（Babar）》和《丁丁历险记（Tintin）》，很期待这个“着色”。

看到尾声，果断改成5星。完成这样一本书的难度很大，数学、逻辑、哲学，这些伟大的人，他们那时候思考些什么、纠结些什么，缺少相关历史知识和训练的我们读这样的内容是十分艰难的。晦涩的内容用漫画小说的方式展现得如此简单和精彩，很难想象两位作者有着怎样的才华和知识积累。哥伦比亚大学毕业的Apostolos Doxiadis写过一本畅销书，一本将数学写成迷人小说的书，而加州大学伯克利分校计算机科学系的讲习教授Christos H. Papadimitriou是*Turing: A Novel about Computation*一书的作者。他们和两位艺术家的合作，让这本漫画内容丰富，引人入胜。

张伸

[@loveisbug](#)

不合格码农。

爱咖啡，爱葡萄酒。爱听布鲁斯，也爱吃巧克力。

不是小资，不是文青，只是喜欢收藏图书和CD。

慢生活，每天听相声入睡。

译文也靠谱，准确、流畅，据说图灵的傅老师翻译了一年，致敬。很高兴秒杀到的这本书有傅老师的签名。

读完这本书，我想，各大学的哲学系应该开设数学分析这门课，而数学系和计算机系应该开设西方哲学史，还得是必修。

——[精彩的漫画小说](#) 读者[张伸](#)



《黑客与画家》

王海鹏

@王海鹏Seal

1994年毕业于华东师范大学。独立咨询顾问、培训讲师、译者和软件开发人员。已翻译二十余本软件开发书籍。拥有二十多年编程经验，目前主要的研究领域是软件架构和方法学，致力于提高软件开发的品质和效率（一回事）。

艾森豪威尔曾说：“计划并不重要，重要的是围绕着计划的那些过程。”什么人参与制定计划？这些人如何制定出这个计划？他们如何执行计划？投入多少人力和物力？当计划与实际情况不符时，他们采取怎样的行动？

书也是一样。什么人写了这本书？《黑客与画家》的作者格雷厄姆既是黑客，也是画家。（好吧，作为画家，他不太有名，只能算个业余画家。）同时他还是天使投资人，创业导师。不得不承认，有些人掌握了一些窍门，导致他们不断地取得成功，从胜利走向胜利。另一些人，在不断失败后，终于找到成功的窍门，从此无往而不胜。成功的喜悦留在了记忆和书里，失败的纠结和痛苦被下意识地淡化了，这称为“保护性失忆”。作者通过《黑客与画家》这本书，分享了他的成功心得。

精通一门技艺很难，如果有人精通两门以上的技艺，那肯定是值得关注的。孔子精通六艺。《学习的艺术》一书的作者精通国际象棋和太极拳。《黑客与画家》的作者精通写程序和画画。《黑客与画家》的译者阮一峰在金融学院教书，却精于IT图书的翻译。大学之道，在于培养某种悟性，能够迅速把握事物的本质。

作为一名C++和Java程序员，我喜欢书中讨论编程语言的部分。读完之后有一种去学习Lisp语言的冲动。一个不懂Lisp的Java程序员，不是一个好的C++程序员。但你不必懂Lisp语言，也可以读这本书。

喜欢写程序的人都是聪明人。聪明是好的，但如果聪明到发现聪明并不是唯一重要的事情，那就更好了。亚马逊的杰夫·贝索斯是这样的人。这本书的作者格雷厄姆也是这样的人，在书中他告诉我们除了聪明之外的那些重要的事情……

——书并不重要，重要的是围绕着书的那些故事 读者王海鹏



《重构：改善既有代码的设计》

特约编辑 高翌翔

[@高翌翔](#)

与其说喜爱编程，不如说是在享受思考的过程。将“诚实、求知、协作、分享”作为个人成长及开发团队建设的核心价值观，并坚持实践、推行。从2005年至今一直从事基于ASP.NET的Web应用程序开发。长期关注各种设计、开发、测试方法及最佳实践！

InfoQ中文站翻译编辑团队一员。

作为程序员，对于《重构》一定并不陌生。尽管许多人把它奉为经典，例如有“改善代码的济世良方”、“隐藏的宝藏”等等好评如潮，但也不乏对它言词犀利的差评，有些甚至尖酸刻薄，例如“内容虽好但是写得太烂”、“肤浅、冗长、且好高骛远”等等。

为何读者的评论会出现如此巨大反差？真相到底是什么？[点击阅读国外读者好差评](#)

对于第一个问题其实并不难回答，“众口难调”四字足矣。除此之外，在差评中多位读者提到此书过于浅显，不适于经验丰富的程序员，更适于刚入门的菜鸟。其实仔细想想就能明白，两位主要作者Martin Fowler和Kent Beck都是敏捷社区的领军人物，总不能算是菜鸟吧？他们劳神费力写出的必定是自认为有价值的内容！二位牛人况且能一步一步地认真重构，可是为什么所谓“经验丰富的程序员”却做不到呢？答案很简单，他们骄傲了！（若您也有此苗头，不妨待会儿读读[《禅修程序员十诫》](#)。）

人的问题讲完了，回过头来认真分析一下此书到底有没有问题？

通过阅读评论可知，此书虽然多次提及《设计模式》，但是与之联系并不密切，正如有位差评读者所写“此书不是为《设计模式》（堪称杰作）读者所写的”。此书的重点在于苦练编码基本功，消除代码中“坏味道”。而对于志在修炼更高阶武功的读者，不妨读一读[《重构与模式》](#)，该书开创性地深入揭示了重构与模式这两种软件开发关键技术之间的联系，说明了通过重构实现模式改善既有的设计，往往优于在新的设计早期使用模式…… ■

为什么选择MongoDB?

TEAP是Turingbook Early Access Program的简称，即图灵早期试读，它公布的是图灵在途新书未经编辑的内容。一本书的翻译或写作周期约为3到6个月，如果作译者能早早地与读者进行沟通和交流，对成书是非常有帮助的。通过TEAP，读者可以提前阅读将来才能出版的内容，作译者也能收获宝贵的反馈意见。

为什么MongoDB对您的项目来说是一个好的选择？我想我已经提供了不少理由了。本节中，我会更明白地进行说明，首先考虑MongoDB项目的总体设计目标。根据其作者的观点，MongoDB的设计是要结合键值存储和关系型数据库最好的特性。键值存储，因为非常简单，所以速度极快而且相对容易伸缩。关系型数据库更难伸缩，至少水平伸缩很难，但拥有丰富的数据模型和强大的查询语言。如果MongoDB能介于两者之间，就能成为一款易于伸缩、能存储丰富数据结构、提供复杂查询机制的数据库。

在使用场景方面，MongoDB非常适合用做以下应用程序的主要数据存储——Web应用程序、分析与记录应用程序，以及任何要求有中等级别缓存的应用程序。此外，由于能方便地存储无Schema数据，MongoDB还很适合保存事先无法知晓其数据结构的数据。



[《MongoDB实战》](#)

之前所说的内容还不太足以让人信服，为了证实它们，我们大致了解一下目前市面上的众多数据库，并和MongoDB做个对比。接下来，我将讨论一些特殊的MongoDB使用场景，提供一些生产环境中的例子。最后，我还会讨论一些MongoDB实际使用中的注意事项。

MongoDB与其他数据库的对比

市面上的数据库数量成爆炸式增长，要在它们之间进行权衡是很困难的。幸运的是，它们之中的大多数数据库都能归在几个分类里。本节中，我会描述简单及复杂的键值存储、关系型数据库和文档数据库，并对它们做一个比较。

简单键值存储

简单键值存储正如其名，它们基于给定的键对值做索引。常见的场景是缓存。举例来说，假设您需要缓存一个由应用程序呈现的HTML页面，此处的键可能是页面的URL，值是HTML本身。请注意，对键值存储而言，值就是一个不透明的字节数组。不用强加关系型数据库中的Schema，也没有任何数据类型的概念。这自然限制了键值存储的操作：可以放入一个新值，然后通过键将其取出或删除。拥有如此简单性的系统通常很快，而且具有可伸缩性。

最著名的简单键值存储是memcached（发音是mem-cach-dee）。memcached仅在内存里存储数据，用持久性来换取速度。它也是分布式的，跨多台服务器的memcached节点能像单个数据存储来使用，这消除了维护跨服务器缓存状态的复杂性。

与MongoDB相比，memcached这样的简单键值存储通常读写会更快。但与MongoDB不同的是，这些系统很少能充当主要数据存储。简单键值存储的最佳用途是附加存储，既可以作为传统数据库之上的缓存层，也可以作为任务队列之类的短暂服务的简单持久层。

复杂键值存储

可以改进简单键值模型来处理复杂的读写Schema或提供更丰富的数据模型。如此一来，就有了复杂键值存储。广为流传的论文 *Dynamo: Amazon's Highly Available Key-value Store* 中描述的 Amazon Dynamo 就是这样一个例子。Dynamo 旨在成为一个健壮的数据库，在网络故障、数据中心停转及类似情况下仍能工作。这要求系统总是能够进行读和写，本质上就是要求数据能自动跨多个节点进行复制。如果一个节点发生故障，系统的用户，也许这里是一个使用 Amazon 购物车的顾客，不会察觉到服务中断。当系统允许同一份数据被写到多个节点时，发生冲突的情况是不可避免的，Dynamo 提供了一些解决冲突的方法。与此同时，Dynamo 也很容易伸缩。因为没有主节点——所有节点都是对等的，很容易从整体上理解系统，能方便地添加节点。尽管 Dynamo 是一个私有系统，但其构建理念启发了很多 NoSQL 系统，包括 Cassandra、Project Voldemort 和 Riak。

看看是谁开发了这些复杂键值存储，看看实践中它们的使用情况如何，您就能知道它们的优点了。以 Cassandra 为例，它实现了很多 Dynamo 的伸缩属性，同时还提供了与 Google BigTable 类似的面向列的数据模型。Cassandra 是一款开源的数据存储，是 Facebook 为其收件箱搜索功能而开发的。该系统可以水平扩

展，索引60TB以上的收件箱数据，允许在收件箱中对关键字和收件人做检索。数据是根据用户ID做索引的，每条记录由一个用于关键字检索的搜索项数组和一个用于收件人检索的收件人ID数组构成。

这些复杂键值存储是由Amazon、Google和Facebook这样的大型互联网公司开发的，用来管理系统的多个部分，拥有非常大的数据量。换言之，复杂键值存储管理了一个相对自包含的域，它对有效存储和可用性有一定要求。由于采用了无主节点的架构，这些系统能轻松地通过添加节点进行扩展。它们都选择了最终一致性，也就是说读请求不必返回最后一次写的内容。用户用较弱的一致性所换得的是在某一节点失效时仍能写入的能力。

这与MongoDB正好相反，它提供了强一致性、（每个分片）一个主节点、更丰富的数据模型，还有二级索引，最后两项特性总是一起出现的。如果一个系统允许建模多个领域对象，例如，构建完整Web应用程序时就会有此要求，那么查询就需要跨整个数据模型，这时就要用到二级索引了。

因为有丰富的数据模型，可以考虑把MongoDB作为更通用的大型、可伸缩Web应用程序的解决方案。MongoDB的伸缩架构有时也会受到争议，因为它并非源自Dynamo。但MongoDB针对不同领域有不同的伸缩解决方案。MongoDB的自动分片受到了Yahoo! PNUTS数据存储和Google BigTable的启发。读过这些数据存储的白皮书的人会发现，MongoDB实现伸缩的方法已经被实现了，而且还很成功。

关系型数据库

本章中已经介绍了不少关系型数据库的内容，简单起见，我只讨论RDBMS与MongoDB的相同点和不同点。尽管MySQL使用固定Schema的数据表，MongoDB使用无Schema的文档，但两者都能表示丰富的数据模型。MySQL和MongoDB都支持B树索引，那些适用于MySQL索引的经验也同样适用于MongoDB。MySQL支持关联和事务，因此，如果您必须使用SQL或者要求有事务，那么只能选择MySQL或其他RDBMS。在不需要事务的情况下，MongoDB的文档模型通常也足够表示对象了。MongoDB中对单独文档的更新也是原子的，这提供了传统事务的一个子集。MongoDB和MySQL都支持复制。就可伸缩性而言，MongoDB设计成能水平扩展的，能自动分片并处理故障转移。MySQL上的分片都需要手动管理，有一定的复杂性，更常见的是垂直扩展的MySQL系统。

文档数据库

自称为文档数据库的产品还不多，在本书编写时，除了MongoDB之外，唯一的著名文档型数据库就是Apache CouchDB。尽管CouchDB的数据是使用JSON格式的纯文本存储的，而MongoDB使用的是BSON二进制格式，但两者的文档模型是相似的。与MongoDB一样，CouchDB也支持二级索引，不同之处是CouchDB中的索引是通过编写MapReduce函数来定义的，这比MySQL和MongoDB使用的声明式语法更复杂一些。两者伸缩的方式也有所不同，CouchDB不会把数据分散到多台服务器上，每个CouchDB节点都是其他节点的完整副本。

使用场景和生产部署

老实说，您不会仅根据数据库的特性就做出选择，您需要知道使用它的真实成功案例。这里，我提供一些广义上的MongoDB使用场景，以及一些生产环境中的范例。

Web应用程序

MongoDB很适合作为Web应用程序的主要数据存储。就算是一个简单的Web应用程序也会有很多数据模型，用来管理用户、会话、应用特定的数据、上传和权限，更不用说完整领域了。正如它们能和关系型数据库的扁平式方法对齐一样，它们也能获益于MongoDB的集合与文档模型。因为文档能表示丰富的数据结构，建模相同数据所需的集合数量通常会比使用完全正规化关系型模型的数据表数量要少。此外，动态查询和二级索引能让您轻松地实现SQL开发者所熟悉的大多数查询。最后，作为一个成长中的Web应用程序，MongoDB提供了清晰的扩展路线。

在生产环境中，MongoDB已经证明它能管理应用的方方面面，从主要数据领域到附加数据存储，比如日志和实时分析。这里的案例来自 *The Business Insider* (TBE)，它从2008年1月起使用MongoDB作为主要数据存储。虽然TBE是一个新闻网站，但它流量很大，每天有超过一百万独立页面访问 (page view)。这个案例中有意思的是除了处理站点的主要内容 (文章、评论、用户等等)，MongoDB还处理并存储实时分析数据。这些分析被TBE用于生成动态热点地图，标明不同新闻故事的点击率。该站目前还没有太多的数据需要分片，但它有使用副本集来保证自动故障转移。

敏捷开发

无论如何看待敏捷开发运动，您都很难否认大家对于快速构建应用程序的渴望。不少开发团队，包括Shutterfly和纽约时代的团队，都部分选择了MongoDB，因为相比关系型数据库，使用MongoDB他们能更快地开发应用程序。一个明显的原因是MongoDB没有固定的Schema，所有花在提交、沟通和实施Schema变更的时间都省下来了。

除此之外，不再需要花时间把数据的关系型表述硬塞进面向对象的数据模型里去了，也不用处理ORM生成的SQL的奇怪行为，或者对它做优化了。如此一来，MongoDB为项目带来了更短的开发周期和敏捷的、中型大小的团队。

分析和日志

我之前已经暗示过MongoDB适用于分析和日志，将MongoDB用于这些方面的应用程序数量增长得越来越快。通常，发展成熟的公司都会选择用于分析的特殊应用作为切入点，进入MongoDB的世界。这些公司包括GitHub、Disqus、Justin.tv和Gilt Groupe，还有其他公司就不再列举了。

MongoDB与分析的关联源自于它的速度和两个关键特性：目标原子更新和固定集合（capped collection）。原子更新让客户端能高效地增加计数器，将值放入数组。固定集合，常被用于日志，特点是分配的大小是固定的，能实现自动过期。相比文件系统，将日志数据保存在数据库里更易组织，而且能提供更强大的查询能力。现在，抛开grep或自定义日志检索工具，用户可以使用他们熟悉并喜欢的MongoDB查询语言来查看日志输出。

缓存

这是一种数据模型，它能更完整地表示对象，结合了更快的平均查询速度，经常让MongoDB介于传统的MySQL与memcached之间。例如之前提到的TBE，它不使用memcached，直接通过MongoDB来响应页面请求。

可变Schema

看看这段代码范例：

```
curl https://stream.twitter.com/1/statuses/sample.json -umongodb:secret | mongoimport -c tweets
```

这里您从Twitter的流上拉下一小段范例，并用管道将其直接导入MongoDB集合。因为流生成的是JSON文档，在把它发给数据库前就不需要预先处理数据了。mongoimport工具能直接将数据转换成BSON。这意味着每条Tweet都能保持其结构，原封不动地存储为集合中的单个文档。无论您是想查询、索引或执行MapReduce聚合，都能立刻操作数据。而且，不需要事先声明数据的结构。

如果您的应用程序需要调用JSON API，那么拥有这样一个能轻松转换JSON的系统就太棒了。如果在存储之前无法预先了解数据的结构，MongoDB没有Schema约束的特性能大大简化数据模型。 ■

作者 / [Kyle Banker](#)
10gen 团队成员，
MongoDB开发者。

译者 / 丁雪丰
[@digitalsonic](#)
支付宝Java攻城师一枚，
InfoQ中文站小编，常年混迹于各种社区，业余时间写作翻译、汉化软件。
《Spring 攻略》、
《RESTful WebServices Cookbook中文版》等图书的译者。

工程的本质问题是组织



[《大道至易：实践者的思想》](#)

工程不是“做”的，是“组织”的。这个“组织”，既有名词的含义，也有动词的含义。除了这个层面上的意思，他还意味着一旦没有确定的组织模式，那么一个具体的工程也就难以落足——即使是个体工程，也有一个确定的组织模式。

组织的源起及其发展的过程，是一个核心问题。

组织的背景

然而在实践中，并没有人注意到工程的组织问题，而只看到了工程实施的结果。这些结果包括RUP、UML、XP以及未知的种种模型、框架和方法论。所有这些看起来可以一统天下的，或致力于一统天下的，都最终走出了它们原本栖身的岩洞，走入莽莽苍

苍的草原或森林；而它们还在认为可以通过敲击岩壁去听发出的声音，并以之判断树洞、沙穴或草坑的居住年限。

它们的共同问题是：没有考虑到具体方法实施时的组织背景。

组织是一个项目的原始背景，组织的构成是人，是人群。工程中最终要解决的，不是具体方法的问题，而是将不同性格、性质的人组织成群体并实质推动的问题。从这个角度上来说，《集体行动的逻辑》所带来的思考远比《人件》、《人月神话》来更加深刻和直接。

无论是以人为工件，还是以“人/月”为度量衡，根本上还是把人作为个体来思考的。即使这些直指个体的思考是必要的，但也无助于组织与该组织下的集体活动。

契约社会与人情关系之间的区别

中国的传统社会模型是基于人情关系的，这与儒家的社会阶层化有着密不可分的关系。儒家从根本上来确定了天地人伦的关系之后，人们的社会行为、族群关系以及家庭婚姻等等都是在这一背景下的自然发展。因此，我们讨论的同学、朋友、党派、宗族以及更简单的门当户对等等，都是人情关系的基础单元。并且这样的基础单元深深地植根于我们的民族文化中，是文化内涵的一部分，也是构成我们的民族化的人性的一部分。

人情作为社会成本（如果将规模缩小到一个组织，则是组织成本）的一部分，是我们这个社会的现实状况。这个成本在不同的领域与不同的事务之间，并不是均匀分布的。有些领域，例如产品生产的人情成本会少一些，而另一些领域，例如组织构建则会

多一些。粗略地分析这一分布的模式，大体上（并且与后面我们要讨论的契约社会比较来看）可以认为：所需要决策的事物对人与人之间的信任的依赖程度，决定了人情关系的比重。

我们将人情社会中的人情成本，转嫁为信任成本的源由在于，很大程度上来说，人情是构成社会信任（以及组织信任）的要素。对于任何一个事物的决策，信任所带来的收益往往是别的努力所无法达到的，例如开上100小时的会议抵不上老总的一句口头传达。这看起来是政治与权力，事实上也可以看着对权威的普遍信任。

我们讨论的“组织”，既是社会背景下的一个局部群体，也是社会行为模式下的一个普遍模式。简单地说，我们的大多数组织是人情组织，而这样的组织下也是人情成本转嫁为信任成本的。正如某天，一个开发人员给我说他想转职去做管理，我问他的第一个问题是：你认为去做管理，会有多少人会听你的？同样的问题是，“你去做产品，会有多少人听你的”，或“你要去创业，又能拉上几个兄弟的人马”？

当提出这样的问题时，我们的大多数开发人员都哑口无言。因为我们多数的、工匠思维下的工程师是关注于做事，而不关注于人情的。在我们的社会与组织背景下，事做得好不好，只是整件事情中的一个比重较大，但不十分关键的因素。同样地，作为一种“集体行动”的组织方法，与“在做的事情”关系也并不大。我常常举例问道：“为什么外行可以当领导”？当然，这一设问本身也有逻辑问题，我的意思是说：领导作为一门技术，显然也是有其方法的。所以，一个反过来的问题是，一个“很有领导能力”的领导，为什么不能做（这件事的）领导？然而，这样的逻辑——至

少在我们的背景下，掩盖了基于人情与信任的那些逻辑，亦即是我们的组织基础以及由此进化而来的行事逻辑。

然而我并不是要将这样的组织与逻辑引入到我们的“工程”中来。正好与此相反，我是要将这些因素从工程中推出去。我的意思是想彻底地问清楚，工程到底是什么呢？在我们的讨论语境下，所谓工程，指的是一种行事方法。如果我们不能清醒地认识到“方法是依赖背景的”，则我们不会转而去看一看这些背景，也就不会把这些背景摒弃在我们的工程之外，或者善而用之，变成有益于工程方法的一部分。然而这仍然存有巨大的风险：工程究其底里仍旧是事，它必然有着方法的背景。那么如果我们把“这样的组织与逻辑”推出去了，又能把什么样的东西拿进来呢？

我的答案，并不是“契约”。

只有基于契约的利益得失会严重影响到社会成本时，“契约”才能够作为组织手段实施。这是基于“自利”这一思维模式的、必然的、必须的组织结论。换个简单的说法，就是如果毁约代价大，则大家都守约；否则，大家该人情还人情，该无耻还无耻。就如同没有极端代价的、公义的法律约束，大家就会犯法一样——这看起来很是好笑，但我们的社会在底子里就存在着这样的一个模式（注1）。

有没有可能换一个视角来看这个问题呢？比如信仰，又比如文明。但如果没有与法律这样一种“人与社会的契约”类似东西的话，又何以判断行为“是否文明”，或是否是“信仰所确定的一部分”呢？看起来，“道德”可能是一个很好的解，也就是“讲道德”大概是文明的，或符合某种信仰的规义的。但“道德”又是什么呢？

注1至于这一模式诸多的内在与外在的驱动力，还可以有相当多的讨论，但并不是我们这里的话题。

注2《社会契约论》是从自由、平等开始讨论的，因此契约本身的成本即是社会关系的成本。基于此，维护契约及其平等性才有必要，因为这就是社会信任的全部出处。然而如果我们以“既存的人情”与“基于人情的组织”来讨论这一问题，那么人情关系在信任的背景中所占的比重就相当地重要了，而“维护契约及其平等性”的必要性与这一比重正好相反。

注3做好一职，做好一事，是我对事的求解，而非对组织的求解，亦非对软件工程的求解（它可以视为从组织视角出发，在具体工程下的一个可能的解）。需要明确的是，所谓“一诺”是基于道德的，而非基于契约的，只是在形式上用到了契约的封皮罢了。

我一直说我们的思维模式是理性而又逻辑的，因此我们必须先说清楚“是什么”。然而在我们的实践中，是不是要这样才没有“错”呢？看起来，我们一方面在花极大的时间，去讨论这样的“正确与否的问题”与“问题的正确与否”，却在一些现实上要去解决的“事”上寸步不前。

换言之，我们是不是可以把大家普遍看起来“道德一些”的东西拿来就用，而暂且无视它学术的、逻辑的或严谨的定义呢？

回到契约的本质上来。即使我们承认一个人的选择是自利的，承认这个选择的背景是阶层化而非自主平等的（**注2**），但为什么不能将选择的结果视为契约的呢？“诺”于一职，“诺”于一事，于“人情社会”真的就是挑战么？我认为不是的。所以在我的看法中，工程在具体上于我们的解，便在于大家对事的负责与不负责，而不在于对组织的“是否契约”的讨论。若变了法子要去改变组织，或要去理解组织的全体，既无益于我们的一个具体的工程，亦无益于我们自己的选择——如果是后者，那你的选择应该不是“做一事”，而是组织的革命（**注3**）。

对此，承认还是漠视？

学术的讨论通常会将契约与人情二元对立起来，试图非黑即白地讨论这二者。大多数情况下，一个论者必须表明他在两个观点之间的明确态度，并且义正辞严地批判另一种。出于学术的讨论与真相的挖掘，这样极端的思维——强调：在一时之间——是必要的，因为若不如此，我们便难于看到一面或另一面的真相。因为有了这种学术讨论的氛围，我们在其中要说“契约社会不错，人情社会也很好”，那么就会被斥为和稀泥，变成两种观点同时讨伐

的对象。大家争来伐去，把第三者打倒了，又开始二人间的互殴，这是市井间打架的惯常路数。

改革家也会跳出来。他们往往关注于旧事物的败亡，以及新事物全面争取阵地的那一刻。这种情况下的局面通常是：旧的东西没有了，即使有一点点可能存在的痕迹也要用黑布挡起来；新的东西看来已经存在，至少每个人的口号中都这样地呼喊。然而这时，这些“改革家”（例如那些来向你宣传的布道者们）立即就消失了踪迹，因为他们也在瞬间失去了主意，不知道应该怎么办才好了。——他们的义务与职责，原本只是改革来着，无对象可革了，这样的角色也就不复存了。

所以真正有意义的改革是革新，而不是革命。革新是容旧渐新的，革命是推翻了事的。然而既然如此，我们便会推知一个重要的事实：旧的，必须与新的共存着非常长的时间。这意味着，这一过程中旧的必须渐渐地“懂得”新的，而新的也必须渐渐地“容得”旧的。于是旧的不再旧，而新的也必然不再是最初设定的那个“新”。因为，“新”的自身也必然是在发展着的，如我一再说的，即使那些“（看起来更正确的）新东西”是真理，也只是一时暂存的“真理的某个表象”，我们必是在追求真理的过程中揭示它的各个方面，而最终——以人类的终极来说，得到一个所谓的真理的。

看起来我是把这个问题变成了一个哲学的讨论。但问题在于，对于“承认还是漠视”这个设问来说，观点无非是“承认”、“漠视”以及“一边承认着，又一边漠视”。这样的观点在二元论的、一元论的思维体系中，总是要去打倒一个两个的。于是我们大家就变成

了一群聒噪的村妇，为着地头上的几株桑树占了谁家的地盘而厮打不休。

然而又过了些年，无论吴、楚都归作了秦，这些厮打也就全然失去了价值。

学术与实践的区别就在于此。能用就用是践行的第一原则，小平同志的话更简单直接：不管黑猫白猫，能抓老鼠的就是好猫。而在我们的工程实践中，我们的背景是人情关系，是阶层管理，是不完善的规章制度……我们若是无视于这些而去实施某种“标准化的”工程，那是有革命的胆量却没有改革的眼光，勇而无能；若是受了这些的牵绊，便又陷入了人事的、章程的困局，因循守旧故而难有破局。

然而我的问题一直不在于这样的——一个“看似合理而又毫无意义”的结论。亦即是说，偏左亦死偏右亦亡，是通常学术所谓“骑墙派”的观点。——这些骑墙看戏的，通常置身事外，将看戏视作一种乐趣亦或是一种必需。尽管这样的“结论”无助于任何的实践，然而这一“过程”——亦即是

“停下来，看看你们在干什么，在什么样的背景下干”，

却是实践所必需的。

所以我认为，工程的问题，尤其是一个具体工程的问题，还是要回到工程的本体——亦即是这个组织与系统的背景上，重新地审视才对。无论那些争吵不休的学术家，或是那些埋头苦干的践行者，都应该仔细地看看我们的工程环境，看看那些“正在做”的做

法，看看那些在组织中被记着“人头数”的工程师，以及被记着“大头数”的管理者们。

我想若是停了下来，哪怕一刻，也是有益于我们前行的。

组织与系统

如果“你”已经是组织角色中的一员——例如某个架构师、项目经理或程序员，那么当跳到系统之外去思考时，你会提出的有关组织的问题是：那个人是谁、在哪里以及在做什么？人总是能看清别人而无法认识自己，因此当你看到“那个人”其是就是“你”的时候，你就已经看到了整个体系、整个体系中的每一个人与每一个“我”——当看到“你”的时候，你已经游离于自我之外。

于是，你的问题是不是在另一个体系上，或在你当前的体系上，只是你的视角问题。例如当你辨别出了所谓的“另一个体系”，以及“当前的体系”，那么只是因为你观察的时候站在了两个体系之间或者前后。如果，你能够站在由二者构成的“整个的体系”之上，那么你的问题就被投放到了这“整个的体系”之中，变成了一个确定的问题。

而所谓的“整个的体系”，就是整体。

你看得到“整体”吗？

所谓“架构师要有大局观”，其实就是指架构师对系统全局的思维能力。例如，当把需求方纳入你的思考范畴之后，你就来到了更为复杂的工程背景设定之中。这时你需要考虑的问题，将不单单包括一个项目的具体实施过程，也包括一个项目如何产生与交

付。影响产生或交付一个项目产品的因素极其复杂，从社会的产能，到各国首脑的多方会谈，甚至到跟你握手的那个客户经理能否在楼下顺利地找到停车位……这所有的一切，构成了一个项目过程的外部因素。而你的组织，只是飘摇在这些外因之下的、微不足道的一把木片而已。

只有站在组织之外，才能看到组织的整体。无论那把木片中的某一块（例如架构师这个角色）的材质是何等地优良，也不能决定自己以何种形式、能否足够优美地飘摇。因为局部不能决定系统，它只能影响后者，或受后者影响。最简单的实例是，你的项目过程再完美，但老板说客户已经破产了，于是你的项目就终结了。在这个例子中，“公司+客户公司”之间的关系，决定了项目的成败，与团队、组织、过程这些因素，全然无关。

无论如何，组织问题以及工程实施问题仍是我们讨论的“系统”的具体视角，并不是其全部。但我们不可能要求一个或某几个角色去关注所有的系统问题。本书也不是“系统论”，因此并不以讨论系统的整体、关联、等级、平衡、时序等等这类问题为主旨。本书希望通过“领域角色的关注”这样的设问，来发现各角色所在领域中最应当关注，以及对系统影响最大的问题集。因此，本书的后续部分将讨论这些问题集以及不同角色的思考，至于这些思考过程与问题的解决方案是否是某个工程方法的实施要件，是一个应用话题，而非本书的焦点。 ■

作者 / 周爱民

[@Aimingoo](#)

资深软件工程师，技术作家。十余年的软件开发、项目管理、团队建设的经验，豫能 - 佳讯 - 盛大 - 支付宝 - ？著有《Delphi源代码分析》、《大道至简》和《JavaScript语言精髓与编程实践》等。

檐头滴水话求职

——《金领简历》编辑手记



《金领简历：敲开苹果、
微软、谷歌的大门》

你写过简历吗？你参加过面试吗？

如果我郑重其事地问你这俩问题，估计你都懒得理我吧？

那么，你知道怎么写简历吗？你了解面试官通常关注哪些方面吗？你知道如何在几个offer之间取舍，如何就录用条件进行谈判吗？你了解如何处理简历面试被拒的情况？知道辞职前该怎么办吗？

如果我再问以上这些问题，你是不是就会抬起头来，满脸期待地等着我说下去呢？

这就是我编辑《金领简历》这本书的亲身感受。十几天的编辑过程中，我脑海里无数次浮现出这样的场景：紧张而期待的面试者、正襟危坐的面试官、整洁的大会议室、泛着蓝光的投影仪，

面试官的桌子旁边，或许还有一摞厚厚的简历；外面的等候室里，或许还坐着一排紧张地等待面试的人……

我个人差不多十年的职场生涯中，也投递过不少简历，参加过不少次面试。从初出校园的羞涩与茫然，到职业转型时的兴奋与期待，再到发展瓶颈期的矛盾与求索，希望再上一个台阶时的诸多期待与困惑……颇不平坦的十年，撞过“大运”，走过弯路，曾遇到一生难忘的伯乐，得到无数丝毫不图回报的指点与栽培，也曾在变态的主管手下战战兢兢度日，在大公司令人窒息的文化氛围里抓墙挠壁、寝食难安。自认是一个满怀理想的热血小青年，愿意披肝沥胆“为知己者死”，但当个人的发展规划难与公司的方向契合时，也会生出太多的迷茫与感伤。

种种经历，尚不足以刻碑立传、警戒后来人，但于我而言，确是一部五彩斑斓之“职场血泪史”，足可在年老发白之日，细细回忆；甚或写成文字，聊慰余年。

但当我仔细阅读《金领简历》时，我觉得自己那些或平坦或崎岖的经历，那些激情与汗水、教训与感伤，都幻化成了书中的两个字：认识。是的，认识职场，就是知道需要掌握哪些技能，以及该如何从学业和专业的角度出发为自己的事业做好准备；认识行业，就是知道该怎样引起目标公司的注意，明白哪些要素会使得他们选中你的简历——或是将其丢在一边；认识面试官，就是明白在面试中不要只顾着自我展现，还要注意倾听，弄清对方问话的真实意图；认识自己，就是知道该在哪些方面付出努力，弥补自己的短处，补充该学会的技能，甚至当对方问起关于你的问题时，你不至于在自认为最熟悉的方面张口结舌；认识未来，就是不会在被拒时心灰意冷，或者怒火中烧；认识机会，就是知道

如何卓有成效地处理工作中的事务，从而使你的职业生涯更上一层楼。总之，有太多的认识，也会有太多的误区，太多我们习以为常的做法，太多大多数人都会犯的错误。说这本书是职场百科，可能有人会撇嘴摇头说“哥已经不信忽悠”；但小编的亲身感受是：本书作者牛X至极的职场经历（至于有多牛X，参见译者序及正文第一章）以及令人难以置信的好运气（同样参见译者序及第一章）绝非偶然，这背后有系统的职场理论所带来的强大支撑，也有作者多年的经历练就的一身硬功夫。

随作者的款款介绍步入缤纷职场，内中的真理与技巧会让你一生受益！

跟我一起走进书中的世界吧，绝对让你不枉此行！

限于篇幅，编辑当止住心里面太多的感慨和惊叹，再来说一下编辑过程中的有趣经历。

本书译者漆犇，曾翻译《[Linux/Unix设计思想](#)》一书，自己发明了修正稿件的迭代流程，愿意一遍遍地修正译稿。本书译稿尚在小编手里时，她就迫不及待询问何老师能否等编辑结束后再给他时间修修订订，然后，最后终稿之前再让她做一次迭代审阅。如此态度，值得鼓励！倘若译者都有此责任心，书稿出来的效果自然不会差啊！

而小编自己在编辑过程中也是边学习边挑错，其中趣味多多，收获多多。试举挑错的有趣一例，若改得不妥，还请各位不吝赐教。

原译中有这样几段内容：



作者 / 图灵编辑 李瑛 [hitliying](#)

毕业于哈工大，偏痴迷文字，得好书则沉醉其中；以编书为乐，遇高手常欲击节而赞。

主攻经管、人文类书的编辑，编过《互联网创业启示录》、《编程大师访谈录》、《你就是极客！》等作品。目前在攻科普书《图说人体系统》。终日纠缠于字词斟酌间，仍不忘时而跳出“编辑加工”之圈，细品器官细胞诸图，赞叹于人体之复杂精妙，造物主之匠心独运。遇绝妙处，常释卷而叹，恨不能起立而歌！痴人独痴，不亦乐乎，不亦乐乎。

我的大学日常事务还包括每周与我的教授马克斯·明茨(Max Mintz)博士一起喝咖啡，他教授的课程是如此地引人入胜，以至于做成了《纽约时报》的一个专题。我们通常在巴克斯县咖啡店(Buck's County Coffee Co.)碰头，然后他会点一份大杯冰咖啡，星巴克没有任何咖啡符合他的口味，他喜欢那种低热量、咖啡因减半且超多奶沫的咖啡。在冰咖啡卖光了的时候（这种情况可比你预想的要多），新手咖啡师便能够体会到他那干巴巴的幽默感，就像新入学的大一新生在他的课上体会过的：

“你有冰块吗？”

“有的。”

“那你有咖啡吗？”

“有的。”

“那不就有冰咖啡么。”

“是的，先生。”

读至此处，小编在为教授的“幽默感”莞尔之余，也不禁心生疑问：马克斯博士只想要杯冰咖啡而已（从下文他要店员用冰和咖啡配咖啡就可以看出），那么星巴克为什么没有任何咖啡符合他的口味呢？据我所知，星巴克的咖啡口味相当全啊，实乃怪哉！于是小编开始查阅资料。网上有关巴克斯县咖啡店(Buck's

County Coffee Co.) 的资料很少，但星巴克的介绍还是蛮多的，功夫不负有心人，找到下面两段，大意是下面这样的（经过了小编的压缩与加工）：

星巴克的咖啡文化就是讲究喝咖啡要喝得PROFESSIONAL。

星巴克咖啡店的一个特色就是，每个顾客都可以点他/她自己的咖啡，想多有个性就有多个性。通过一大堆的术语，你可以点世界上只有你会喝的咖啡，这个叫customization。这也是个趋势：星巴克的每个咖啡师都可以向你保证，只要你说得出来，我们就能做得出来，如果你去其他的咖啡店，你能点你要想的温度吗？星巴克可以。星巴克可以做任何温度的咖啡，只要你说到，我们就会给你做出来。在星巴克点咖啡还有不少专用术语.....

上面的文字也印证了之前的印象，于是小编据此推测，原文绝不是说“他喜欢那种低热量、咖啡因减半且超多奶沫的咖啡”，这些对咖啡的要求很可能只有星巴克才有。再看原文：none of that crazy Starbucks venti-skinny-halfcaf-extra-foam lingo for him。也就是说，星巴克咖啡店里那些“超大杯、调入脱脂牛奶、低咖、多奶沫”之类疯狂的术语没有一个适合他。

可不是咋的？咱家教授喝咖啡没那么多讲究，就要大杯的、冰的就可以啦！于是半截笔头一挥，改之！（本想说“大笔一挥”的，可惜因为俺们公司的红笔芯和笔壳不配套，笔芯被俺剪得过多，只好用纸再垫上一截，故曰“半截笔头”。） ■



好文章要给更多人看，用你的笔传递最新IT好文。阅读量前5名的译人可任选图灵出版图书一本。

[iTTran乐译7期](#)

恭喜高翌翔、hasse、旁观者、迷茫、五指琴魔和周庆成6位同学在本期拔得头筹

- [本期算法：图解暴力字符串匹配（含JS实现示例）](#)
- [大数据与拓扑学](#)
- [Light Table ——一种新的IDE概念](#)
- [剪贴板的秘密（一）](#)
[剪贴板的秘密（二）](#)
- [故事样设计——怎样像用户一样思考？](#)
- [世界的呈现 \(演讲记录稿\)](#)
——来自Mapbox，创新的地图模式
- [禅修程序员十诫](#)
- [Paul Graham博客：专利的誓言](#)
- [创意帮创意——一个关于图书、出版和筹款的故事，能且只能出现在这个时代](#)



“专家审读”是图灵编辑出版流程的重要环节，即邀请图灵社区会员以专业读者身份阅读全稿，并修正之前环节未查出的问题。事实证明，这个环节能有效地保证书籍质量。

专家审读第3期

本期共有11位会员加入图灵技术专家审读小组，在我们的新书付梓前，以专业读者身份审读了以下三本书的纸稿，并给予了积极的反馈和评价，他们是（社区ID）：

旁观者、zangxt、kraft、veldts、侯伯薇、刘晓日、Jovi、常新居士、William_Xu、于东兴、小谢

为了表达我们的谢意，在新书出版后，除了审读图书，各位专业读者可再联系我们，选一本您喜欢的书（图灵出版）。



[七周七语言](#)
2010年Jolt效率奖。

Amazon ★★★★★
(26人评价)



[演讲的艺术](#)
全球知名演讲培训机构的教程精华。

Amazon ★★★★★
(37人评价)



[他山之石: TechStars孵化器中的创业真经](#)

Amazon ★★★★★
(58人评价)



图灵每个月都会举办或参与一些技术会议，“跟着图灵听课去”让您跟随图灵的脚步提前了解这些会议的亮点，在现场和图灵互动，并在会后得到第一手的会议报到和相关资源。

七月，跟着图灵听课去！

7日 [淘宝开放平台-金牌合作伙伴交流会：和小二面对面](#)

由淘宝开放平台和天猫社区电子商务部的各路小二，为你带来产品、技术的精彩分享。

7日-8日 [阿里技术嘉年华：上天入地的技术盛宴](#)

从前端到后端、从测试到运维、从交互体验到大数据。本次嘉年华分为七个分会，每个主题都有深入的技术分享和讨论。来自新浪、百度、腾讯、土豆的优秀工程师和设计师为大家奉上一场技术盛宴。

重在参与的workshop。本次大会特别推出Tech Loft版块，组织现场工程师们“近距离”交流，技术“深层次”探讨。大会挑选出的若干精彩专题，由来自阿里集团的技术发烧友发起并组织交流。

12日 [美国计算机图书市场分析交流会：和出版人聊出版](#)

什么编程语言在去年独领风骚？电子书和数字出版的前路如何？哪家计算机出版商在2011年拔得头筹？来自美国的数据对于我们又有什么启示？也许，我们无法得出一个确定的答案，但是在这充满变革的时代，我们愿意尝试读懂数据给我们的暗示。

14日 [我们的开源项目技术聚会：低调的精彩](#)

真正的开源开发者，其实都是非常务实和低调的。但.....有些事，你不说，别人怎么能知道呢？在这次技术聚会上，各路开源开发者都会聊聊自己做过的、正在做的、打算做的开源项目。中国的开源项目数量也许并不多，于是，这次聚会就更显得弥足珍贵。

[更多最新活动，请来社区活动版](#)



多看阅读 for iOS
诚意上架

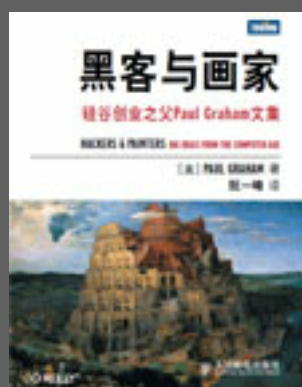


现在，您可以在iOS设备上阅读图灵电子书了！

图灵与多看合作的 [多看阅读](#)，已陆续上线多本电子书，敬请关注：



番茄工作法图解：简单易行的时间管理方法



黑客与画家：硅谷创业之父Paul Graham文集



布道之道：引领团队拥抱技术创新



门后的秘密：卓越管理的故事

图灵社区 出品

出版人：武卫东

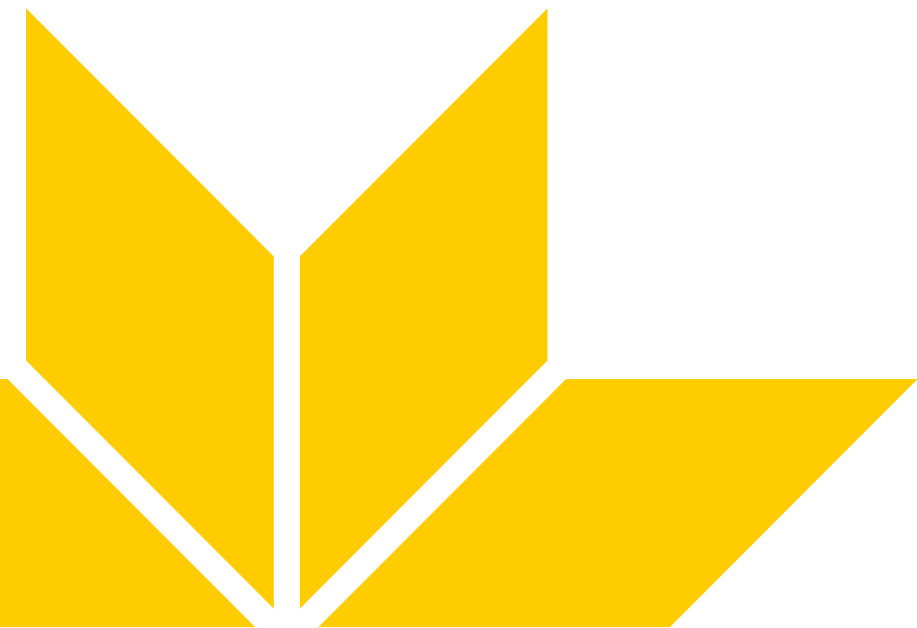
执行主编：杨帆

顾问：何健辉、谢工、胡晓东

设计：大胖

本刊只用于行业交流，免费赠阅。

署名文章及插图版权归原作者所有。



地址：北京市朝阳区北苑路13号院领地OFFICE C座603室

电话：010-51095181

微博：weibo.com/ituring

Email: yangf@turingbook.com