码农 the code maker

17

大数据时代的统计学思考

在AWS上构建第一台机器

针对PageRank的链接作弊

利用奇异值分解简化数据

机器学习产品开发

R语言可视化初阶

连城：大数据场景下的“骚到痒处”和“戳到痛处”

松本行弘：我为什么要开发新语言Streem

如何成为一位数据科学家

tree by Beresnev

iTuring.cn

目　录

编者的话

专题：如何成为一位数据科学家

1　大数据时代的统计学思考

22　在AWS上构建你的第一台机器

35　针对PageRank的链接作弊方法剖析

44　如何利用奇异值分解简化数据

58　Peter Harrington：如何成为一位数据科学家

64　机器学习产品开发的漫漫长路

71　R语言可视化初阶

人物

106　连城：大数据场景下的“骚到痒处”和“戳到痛处”

鲜阅

121　松本行弘：我为什么要开发新语言Streem

践行

137　老码农的技术理想

九卦

143　为狗狗制作的商业模式画布

动手

154　程序员为什么值得写博客

书榜

161　看看大家都在读什么?
164　电子书榜

妙评

165　Hello Kitty：我的成功你不能复制

成书手记

169　“半个保险丝”之谜 ——《咨询的奥秘》翻译轶事
173　《系统化思维导论》的25周年纪念

编者的话

如何成为一位数据科学家

编者 / 李盼

仅仅在几年前，数据科学家还不是一个正式确定的职业，然而一眨眼的工夫，这个职业就已经被誉为“今后10年IT行业最重要的人才”了。在《数据之美》一书中，对于Facebook的数据科学家，有如下叙述：

“在Facebook，我们发现传统的头衔如商业分析师、统计学家、工程师和研究科学家都不能确切地定义我们团队的角色。该角色的工作是变化多样的：在任意给定的一天，团队的一个成员可以用Python实现一个多阶段的处理管道流、设计假设检验、用工具R在数据样本上执行回归测试、在Hadoop上为数据密集型产品或服务设计和实现算法，或者把我们分析的结果以清晰简洁的方式展示给企业的其他成员。为了掌握完成这多方面任务需要的技术，我们创造了‘数据科学家’这种角色。”

数据科学家是否由Facebook创造，我们尚不可知，然而这确实是一种前所未有，但却无比稀缺的职业。“不懂统计学的码农不是一位好的数据科学家”，纵然这样概括数据科学，却仍显得不够全面。要成为一位数据科学家，需要掌握统计学、线性代数和一些编程技能，也要精通数据预处理、数据再加工、数据建模、编码、可视化和有效沟通。万丈高楼平地起，就让我们从一些简单而有趣的知识开始探索数据科学的旅程吧。

数据科学家每天的工作流程是什么？数据科学的学习路径是什么？在本期《码农》中，你将听到来自数据科学家们的回答。除此之外，我们还将

和你一起探索一些实用的数据加工技术。奇异值分解是一种无论在生物信息学还是在金融学领域内都有广泛应用的数据简化方式，我们将向你介绍基于Python的SVD实现；作为人与数据间的桥梁，数据科学家需要掌握数据可视化方面的必要知识，而即将介绍的R语言则是一种绝妙的工具。另外，我们还将拓展思维，一起回顾一下机器学习的漫漫长路，比起模型设计和软件开发，也许更需要了解的是如何诠释问题。

Google、Amazon、Facebook、Twitter，这些称霸互联网业界的企业，不仅是数据分析的受益者，也是大数据储存和处理技术的推动者。当你需要更多计算资源时，不需要预先购买大量机器，可以直接利用亚马逊Web服务（AWS）来实现；Google的立家之本PageRank是大数据处理领域的伟大创新，就让我们来听听Sergey Brin的老师Ullman教授是如何从垃圾农场架构的角度破解PageRank链接作弊的。

说到大数据领域的创新，Google的三驾马车（GFS，MapReduce，Bigtable）曾经开启了大数据处理时代的序幕，然而技术的更迭创造出了更好的产品。本期“码农人物”连城是Spark核心构建者Databricks的工程师。在做Spark之前，连城从来没有从事过大数据分析方向的工作。如今作为Spark committer的他，对大数据分析逐渐形成了自己的理解，他认为“对工具的选择，既可以解放我们的思想，也可以禁锢我们的思想”。而他自己曾经并不感冒的函数式编程，可能才是更加契合大数据场景的编程方式。

在历史上的任何时期，掌握着先进工具的人也就掌握着未来。在大数据时代，数据科学家无疑就是这个时代点“石”成金的人。■

专题：如何成为一位数据科学家

大数据时代的统计学思考

作者 / Rachel Schutt

美国新闻集团旗下数据科学部门高级副总裁、哥伦比亚大学统计系兼职教授、约翰逊实验室高级研究科学家，同时也是哥伦比亚大学数据科学及工程研究所教育委员会的发起人之一。她曾在谷歌研究院工作数年，负责设计算法原型并通过建模理解用户行为。

“大数据这个词现在时常被人们随意使用，然而其语义十分模糊。简单地说，这个包罗万象的词条一般有三层含义：首先，它指代一揽子的技术；其次，它有可能引发一场度量数据规模的革命；最后，它为人们未来将会、甚或是应该如何制定决策提供了一个新视角，一种新理念。”

—史蒂夫 · 洛尔（Steve Lohr）《纽约时报》

在你打算成为一名数据科学家时，以下技能是必须首先具备的：统计学、线性代数和一些编程技能。同时你还需要发展以下技能：数据预处理、数据再加工、数据建模、编码、可视化和有效沟通，这些技能往往是相辅相成的。万丈高楼平地起，让我们先从统计推断开始探索数据科学的旅程。

我希望想成为数据科学家的人拥有五花八门的背景，你可能是位优秀的软件工程师，有能力搭建数据管道，但对统计学却所知甚少；或者你是一位市场分析专员，一点也不懂如何编写程序；或者你只是一位对数据科学充满好奇的读者，想弄明白数据科学到底是什么。

虽然我们在这里列举了想要成为数据科学家需要具备的一些先决条件，但我们不是查户口的，无法到你家去检查你是否修习过有关统计学的课程，或者是否曾经看过有关统计学的书籍。即使你曾经选修过诸如统计学导论之类的入门课程，但正如我们在鸡尾酒会上经常听到的那样，99% 的

作者 / Cathy O'Neil

约翰逊实验室高级数据科学家、哈佛大学数学博士、麻省理工学院数学系博士后、巴纳德学院教授，曾发表过大量算术代数几何方面的论文。他曾在著名的全球投资管理公司D.E. Shaw担任对冲基金金融师，后加入专门评估银行和对冲基金风险的软件公司RiskMetrics，个人博客：mathbabe.org。

人都十分惧怕统计学，宁愿从来没上过这门课。如此说来，你不大可能从这些课程中真正领略到了统计学的美妙，更不可能深入研究它。

即使你取得了统计学的博士学位，已经对这一领域有精深的研究，你也可以回顾一些基础概念，记起什么是统计推断、什么是统计学的思考方式，这总是有所助益的，特别在“大数据”时代这个全新的语境下，很多传统的统计学方法可能都需要重新审视和修订。

统计推断

我们所处的世界异常复杂，充满了随机性和不确定性，同时，这个世界又是一个巨大的数据生产机器。

我们搭地铁或开车去工作，我们的血液在身体里流动，我们购物、收发电子邮件、因浏览网页或查看股价而耽误正事；我们工作、吃饭、和朋友家人聊天；工厂里生产产品……这些行为都会直接或间接产生数据。

试想花一天时间去观察窗外的人流，记录下每一个经过的人；或者聚集起住在你一公里以内的人，问他们在过去的一年中每天收到多少封电子邮件；或者去你附近的社区医院翻阅血液样本，去发现蕴藏在其中的DNA模式，这些行为乍一听觉得有点变态，让人觉得不安，但事实并非如此。我们的生活本身就是不断产生数据的过程。

我们想用多种方式去描述、揭示这些过程，使人们更好地理解数据的产生。作为科学家，我们想更好地了解这个世界。对过程的了解掌握，也是解决问题的一部分。

数据就是现实世界运转留下的痕迹。而这些痕迹会被如何展示出来，则取决于我们采用什么样的数据收集和样本采集方法。假如你是数据科学家，那么作为一个观察者，你要做的事是将具象的世界转化为抽象的数据，这个过程是绝对主观的，而非人们所想的那么中立客观。

将这一过程从数据收集中剥离出来，就能清晰地看到蕴藏其中的随机性和不确定性的两个源头：一是来自过程本身，二是来自数据采集方法。

从某种程度上说，掌握了数据就掌握了世界，或者世界的运作规律，但是这并不代表着，你拿着一个巨大的Excel文件，或者存有数百万条记录的数据库，手指轻轻一划，就能理解世界及其运行规律。

你需要新的点子，将这些采集到的数据进行简化，使它们更易于理解，能够以一种更简明扼要的方式概述世界运行的规律，能够易于使用数学对其进行建模的数据，这称为统计估计量。

这一套从现实世界到数据，再由数据到现实世界的流程就是**统计推断**的领域。

更准确地说，统计推断这门学科主要关注如何从随机过程产生的数据中提取信息，它是流程、方法和理论的统一。

总体和样本

让我们先来统一一些术语和概念。

在经典统计学理论中，有**总体**和**样本**之分。英语中总体和人口是同一个单词，因此一说这个词，人们就会马上联想到：美国人口总数3亿、全世界人口总数70亿等。但是，在统计推断中，总体并不特指人口，它指的是一组特定的对象或单位，比如推特上发布的消息，照片或者天上的星星等。

如果我们可以度量和提取这些对象的某些特征，就称为对总体的一组**观察数据**，习惯上，使用N表示对总体的观察次数。

假设总体为去年“巨无霸”公司员工发送的所有电子邮件，则对该总体的一组观察数据可以包括以下内容：发信人的姓名、收信人列表、发信日期、邮件内容、邮件字数、邮件中的句子数、邮件中动词的个数、从邮件发出到获得第一次回复中间的时间等。

接下来就要采集样本。所谓**样本**，是指在总体中选取的一个子集，用n来表示。研究者记录下样本的观察数据，根据样本特征推断总体的情况。采样的方法多种多样，有些采样方法会存在偏差，使得样本失真，而不能被视为一个缩小版的总体，去推断总体的特征。当这种情况发生时，基于样本分析所推断出来的结论常常是失真甚或完全错误的。

在上述“巨无霸”公司的例子中，可以**随机**抽取全部员工的1/10，以他们所发的邮件组成样本，或者**随机**选取一天，以该天内所发邮件的1/10作为样本。这两种抽样方式都是合理的，而且样本的数量也是相等的，但是，如果你据此计算每人发送电子邮件的数量，进而估计出“巨无霸”公司员工的发送邮件分布情况的话，应用两种采样方式，你将会得到完全不同的答案。

这样一个简单的问题，由于采样方式的不同，结果都会失真，那么对于那些复杂的算法或模型，如果你没有把获取数据的方式考虑进来，情况又会怎样？

大数据的总体和样本

在大数据时代，我们有能力记录用户的所有行为，我们难道不就可以观察**一切**？那么，此时做总体和样本的区分还有意义吗？如果我们已经拥有了所有的电子邮件，干嘛还要采样？

这些疑问直抵问题的核心，对此，我们有如下几个方面需要加以澄清。

采样可以解决一些工程上的挑战

在时下流行的关于大数据的讨论中，企业都主要采用Hadoop等分布式技术去解决海量数据带来的工程及计算问题，但他们却忽略了采样这种手段也同样有效。事实上，在谷歌，软件工程师、数据科学家和统计学家时刻都在用到采样来处理大数据。

使用多少数据取决于你的目标：比如，做分析或推断，你只需要部分的数据就可以了；但当你试图在用户界面上展示其中一个使用者的信息时，你可能需要搜集该使用者的所有数据。

偏差

即使我们拥有了谷歌、Facebook 或 Twitter 的所有数据，基于这些数据所做出的统计推断并不适用于其他不使用这些服务的人群，即使针对使用这些服务的人群，上述统计结论也不能准确说明他们某一天的活动轨迹。

微软研究院的 Kate Crawford 女士在她的演讲 “Hidden Biases of Big Data” 中提到，如果仅对飓风桑迪到来前后的推特做分析，可能会得出这样的结论：人们在飓风来临前在购物，飓风过后在聚会。然而，这些推特大部分来自纽约人。首先，他们是推特的重度用户，而这时沿海的新泽西人正在担心他们的房子会不会被飓风吹倒，他们哪里还有时间和心情去发推特呢?

换句话说，如果你使用推特数据来分析桑迪飓风的影响，得出的结论可能会是：这场飓风危害不大。事实上，你得出的结论只能说明飓风对推特用户的影响。他们受桑迪飓风的影响很小，还有时间来发推特，但他们无法代表一般意义上的美国大众。

同样在这个例子中，如果你不了解相关的语境或者对桑迪飓风一无所知，你就不可能对数据做出合理的解释。

采样

让我们再来看看总体和样本在各种语境下的含义。

在统计学中，我们通常使用一种基础的数学方法来建模，描述总体和样本的关系。针对背后可能存在的规律、数学结构以及生成数据的过程，

我们会做出一些简单假设。每一次研究，我们对其中一种数据生成过程中所采集到的数据进行观察，这组数据就是所谓的样本。

以“巨无霸”公司的邮件为例，如果我们随机抽取阅读一些邮件，就会产生一个样本。但如果我们再次抽取，又会产生一组完全不同的样本。

由于采样过程不同所带来的不确定性有一个学名：**取样分布**。就像2010年上映的、由莱昂纳多·迪卡普里奥主演的《盗梦空间》一样，这是一个梦中梦。因此，也可以将“巨无霸”公司的电子邮件想象成一个样本，而不是总体。

这些电子邮件是一个更大的超级总体的样本（此刻，我们有点哲学家附体），如果抽样时是用扔硬币来决定的话，硬币若多翻转一次，得出的样本就会完全不同。

在这里，我们使用一组电子邮件作为样本，来推断其中一个数据产生的过程，比如说：“巨无霸”公司员工的电子邮件书写习惯。

新的数据类型

过去，所谓数据是指数字和一些分类变量，但今非昔比。在大数据时代，一个优秀的数据科学家需要多才多艺，要处理的数据种类比过去要多得多，如下所示。

- 传统数据：数字、分类变量和二进制变量。
- 文字：电子邮件、推特、《纽约时报》上的文章。
- 记录：用户数据、带有时间戳的事件记录和JSON格式的日志文件。
- 地理位置信息数据。
- 网络。
- 传感器数据。
- 图片。

这些新的数据类型要求我们在做采样时需更谨慎。

以Facebook用户产生的实时数据流为例，从带时间戳的日志上，可以抓取用户一周内的活动数据，在此基础上进行分析，得出的结论适用于下周或者下一年吗?

在复杂的网络中你又如何采样，使得样本可以反映总体的复杂性?

这些问题都是统计学和计算机科学领域内的开放性研究问题，我们本来就身处科技的前沿。在实际中，数据科学家尽其所能去解决这些问题，在他们的工作中，经常发明出一些创新性的方法。

术语：大数据

我们已经多次提到了“大数据”，但是一直没有好好定义它，我们也觉得不好意思，那就让我们先来看看什么是大数据。

大数据的大是相对的。人为地为大数据限定一个阈值，比如1PB，是没有意义的，这太绝对了。只有当数据的规模大到对现有技术（比如内存、外存、复杂程度、处理速度等）构成挑战时，才配称为“大”。因此，大数据的大是一个相对概念，大数据放在20世纪70年代和现在的意义是完全不同的。

当用一台机器无法处理时，就可以称为“大数据”。不同的人、不同的公司，拥有的计算资源是有差别的，对于数据科学家来说，如果数据大到一台机器处理不了，就可以称其为“大数据”，因为她不得不学习使用一些全新的工具和方法去解决这一问题。

“大数据”是一种文化现象。它描述了数据在人类生活中所占的比重，随着科技的发展，数据所占的比重越来越大。

大数据中的4V原则。这4V是指容量（Volume）、种类（Variety）、速度（Velocity）和价值（Value）。很多人借此来描述大数据的特征，你也可以从中学习借鉴。

大数据意味着大胆的假设

库克耶和迈尔-舍恩伯格的文章“The Rise of Big Data”中，他们提出大数据的革命由以下三方面构成：

- 采集和使用大量的数据，而不是小样本；
- 接受数据中存在杂乱噪声；
- 重视结论，放弃探究产生结果的原因。

他们将这三方面说得冠冕堂皇，他们宣称，数据是如此巨大，没有必要去寻找原因。也不用担心采样出错，因为所有的数据都在这，它们记录了一切事实。之所以这样说，是因为他们声称找到了处理大数据的新方式，那就是让“N = 全部”。

N能代表全部吗？

答案是N永远不能代表全部。我们经常忽略那些我们最应该关心的事实。

以InfoWorld上的一篇帖子为例，互联网监控永远也不可能奏效，我们最想抓住的那些罪犯，恰恰是非常聪明、精通技术的，他们永远领先一步，永远也不会被人抓住。

那篇文章举了一个选举之夜投票的例子，这个例子中本身就存在矛盾之处：即使我们能把所有离开投票站的人都纳入统计，我们也还是遗漏了那些从一开始就不打算投票的人，他们那天晚上压根儿就没来投票站。而这些人或许才是我们需要关注的人群，和他们交谈才能了解我们国家现有投票制度存在的问题。

事实上，我们认为“N = 全部”这个假设是大数据时代人们面临的最大问题。首先，这种假设天然地将一大部分人排除在外，他们可能是因没有时间、精力、渠道去参加那些非正式或者未经宣布的选举投票。

在统计选票时，我们忽略了那些同时打两份工的人，还有那些把时间都花在等公交车上的人，他们因为各种原因没有投票。对你来说，这或许只意味着Netflix的推荐引擎推荐给你的电影不符合你的口味，因为在Netflix上愿意费心去为电影打分的大多是年轻人，他们的口味可能和你的不一样，这些打分行为使得推荐引擎更贴近他们的喜好。但是，上述例子中提出的基本观点在现实生活中很可能会埋下许多潜在隐患。

数据是不客观的

若说“N = 全部”这一假设可以成立，则意味着要承认数据是客观的。但是，相信数据是客观的，或者“让数据说话”，这些观点是错误的，当然，对于那些持完全相反观点的人也应该小心提防。

最近，我们在《纽约时报》上发表了一篇关于运用大数据方法来招聘人员的文章，正是这篇文章使我们意识到上述观点的可怕。文章中引述了一位数据科学家的说法：“让我们把所有东西都放进来，让数据自己说话。”

通读全文后，你会认识到，运用大数据运算法则是为了寻找到有潜质的“璞玉”式人才。这种做法在招聘中值得一试，但是有些事你要深思熟虑。

假设你面前有两位资历相当的求职者，一位是男性，一位是女性。阅读他们的简历你发现，相比之下，这位女性求职者跳槽次数更多，获得提拔的机会较少，而且对以前工作过的地方有更多的负面评价。

当再有这样的情况出现时，你若通过模型做决策，可能会雇用男性而不是女性求职者。你的模型是不会把有些公司歧视女性的情况考虑进去的。

换句话说，忽视因果关系是大数据法则的一种缺陷，而不是特征。忽视因果关系的模型无助于解决现存问题，而只会增加更多问题。数据也不会自己说话，它只能够以一种量化的、无力的方式去描述、再现我们身边的社会事件。

$n = 1$

与“N = 全部”这一极端观点相反的是另一个极端：$n = 1$，意思是说样本的总数为 1。过去，说样本空间的大小为 1 是很荒谬的，没人会通过观察一个个体，就得出对总体的推断。别担心，这种观点现在仍然是荒谬的。不过，在大数据时代，$n = 1$ 有了新的含义，对于一个人，我们可以记录所有关于他的信息，我们可以对所有举动进行采样（比如他打过的电话、他敲击键盘的记录），并对这些行为进行统计推断。这是一种用户级别的建模。

建模

Rachel 曾经给一位朋友打电话，讨论建模交流会的事，几分钟后她意识到，“model”这个单词对他俩来说完全是不同的含义。对方理解成了数据模型（data model），一种存储数据的结构，这属于数据库管理员的研究领域。而 Rachel 的意思是统计学中的模型，更可笑的是，最近 Andrew Gelman 的一篇关于建模的博客被时尚圈的人士发到了推特上，他们可能理解成了模特。

即使你已经使用**统计模型**或**数学模型**这两个术语很多年了，但当你向周围人谈起模型时，你和他们都明确知道这个词的含义吗？什么使一个模型成其为**模型**？当我们问起这些基本问题时，还有一个问题也很重要，统计模型和机器学习算法之间有什么不同？

在深入这些问题之前，让我们先来看看 Chris Anderson 2008 年在《连线》杂志上发表的文章“The End of Theory: The Data Deluge Makes the Scientific Method Obsolete”，这篇引人争议的文章为我们的讨论增加了更多素材。值得一提的是，Chris Anderson 那时候是《连线》杂志的主编。

Anderson认为数据即信息，并且宣称不需要模型，了解相关性就够了。以海量数据为例，“谷歌根本没有必要使用模型”。

是这样吗？我不相信，我觉得你们也不会相信。这种观点类似库克耶和迈尔-舍恩伯格在他们的文章中提出的“N = 全部”，我们刚刚在前面讨论过。现在，你也许已经可以感受到环绕在我们周围的深深的疑惑了吧。

然而我们需要感谢媒体，是他们让公众了解了这些问题，可是，当这些意见领袖们并不是专职从事数据科学工作的人员时，对他们的说法就得打一个问号了。仔细想想你是否同意Anderson的观点，哪些部分你认为是对的，哪些地方你认为是错的，或者你需要获取更多信息才能形成自己的观点。

鉴于公众对数据科学和模型的认知都来自主流媒体这样业余的描述，作为数据科学家，我们应该义不容辞地发出自己的声音，贡献出自己的真知灼见。

在这个大背景之下，当我们谈起**模型**时，其含义究竟是什么？数据科学家是如何使用它们的？

什么是模型？

人类试图用各种方式去描述他们所处的世界。建筑学家用蓝图和三维立体模型来捕捉建筑的属性；分子生物学家用连接氨基酸的三维图像描述蛋白质结构；统计学家和数据科学家则用函数表示产生数据的过程中存在的不确定性和随机性，并以此来形容数据本身的样貌和结构。

模型就好像一个特殊的镜片，我们透过这个镜片去观察和了解现实世界的本质，这个“镜片”可能是建筑学、生物学或数学模型。

模型是人工设计的，用于将无关紧要的细节排除或抽象化。在进行模型分析时，研究者必须关注这些被省略的细节。

以蛋白质为例，一种本身带有侧链的蛋白质骨架却不受规范电子运行轨迹的量子力学理论的约束，最终决定了蛋白质的构架和行为。以统计模型为例，建模时我们可能错误地排除了一些关键变量，而使用了一些与问题无关的变量，或者采用的数学结构偏离了问题本身。

统计建模

在引入数据和开始编写代码前，对于建模的流程有一个大概的了解是有益的。先干什么？谁受谁的影响？什么是因，什么是果？检验结果如何？这些都是我们应该思考的问题。

人们使用的方法各有不同，有些人喜欢用数学去描述这种关系。通用的数学公式里面必须包括参数，但是参数的值是未知的。

按照惯例，数学公式里一般使用希腊字母表示参数，拉丁字母表示数据。比如你有两列数据：x和y，你认为二者之间是线性关系，用公式表达如下：$y = \beta_0 + \beta_1 x$，此时你还不知道 β_0 和 β_1 的具体数值，因此，它们就是参数。

还有些人则喜欢画图。他们先画一张数据流的图，很可能带有箭头，用来描述事物之间是怎么相互影响的，或者在一段时间内发生了些什么。在选择公式表达这种关系之前，这种关系图可以给他们一个大概的描述。

如何构建模型

你怎么知道什么数据该用什么模型？这一半是艺术，一半是科学。这个问题正是打开数据科学大门的钥匙，模型的选择是建模过程中的一环，

你需要对底层结构做出大量假设，应该有一个标准来规范如何选择模型和解释这样选择的理由。但是我们还没有统一的规范，所以只能摸着石头过河，希望经过深思熟虑，能制定这样一套规范。

也许探索性数据分析（EDA）是一个好的开始。它牵扯到绘制图形和从数据集中获取直观的感觉。与试错、反复实验一样，探索性数据分析对问题的解决大有帮助。

实话实说，除非你做过很多次，否则这一过程在你眼里依然是那么神秘。最好是从易到难，先做看起来最傻的事，事后看，或许没有你想象得那么傻。

举例来说，你可以（或者说是应该）先绘制直方图或散点图以对数据产生一个直观的感受，然后试着写点什么，哪怕一开始是错误的（很可能一开始的结论是错误的，没关系）。

比如写出来的是一个线性方程，当你把它写下来，就会强迫自己去思考：这个方程有意义吗？如果没有，为什么没有？那怎样的方程对这个数据集是有意义的？你从最简单的方式开始，逐渐增加复杂度，做出假设并把你的假设写下来。如果你觉得完整的陈述有帮助，就把句子写完整，比如：“我假设将用户自然分成五组，因为销售代表谈起用户的时候，她把用户分成了五类。”然后试着用方程式和代码来表达这一陈述。

记着，从简单处着手永远是个好办法，建模时在简单和准确之间有一个权衡。简单的模型易于理解，很多时候，原始简单的模型帮你完成了90%的任务，而且构建该模型只需要几个小时，采用复杂的模型或许会花上几个月，而且只将这个数值提到了92%。

你将从构建模型开始，它们将组成你的“兵工厂”。在构建模型时会用到很多模块，其中一种就是概率分布。

概率分布

概率分布是统计模型的基础。在讲述线性回归和朴素贝叶斯时，你就会知道我们这么说的原因。概率论是一门需要花费几学期去教授的课程，将这样庞杂的内容压缩讲述，对我们来说实在是个巨大的挑战。

回到还没有发明计算机的年代，科学家观察到了现实生活中的一些现象，经过测量后发现，一些固定的数学模式在重复出现。其中的经典案例莫过于人的身高服从正态分布，正态分布是一种钟形曲线，也叫高斯分布，以数学家高斯的名字命名。

还有其他一些经常出现的曲线，也分别以各自的发现者的名字命名（比如泊松分布、韦伯分布等），另外一些曲线，如伽玛分布、指数分布，则是以描述它们的数学方程式命名。

自然状态下产生的数据，可以用数学函数来描述。通过设定函数中的参数，可使函数曲线接近于实际数据的分布形态。而这些参数可在对数据进行估计的基础上得出。

不是所有过程产生的数据都服从某种已知的分布，但很多都服从。我们可以应用这些分布函数作为模块，来构建最终的模型。我们不打算深入探讨每种分布的细节，但我们提供了图 1 用来展示这些常用的分布，事实上有无穷多种分布，列在这里的这些只是因为有人观察了它们很久，觉得有必要给予命名而已。

概率分布可以理解为对于可能结果的子集指定一个概率，概率分布用与其对应的函数来表示。比如正态分布的函数为：

$$N(x|\mu,\sigma) \sim \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

参数 μ 是平均值或中位数，决定了该分布的位置（正态分布是一种对称的分布）。参数 σ 决定了分布的幅度。这是一般意义上的方程式，在真实世界的各种特定现象中，这些参数的值是固定的，我们可以通过对数据的估计得到这些参数。

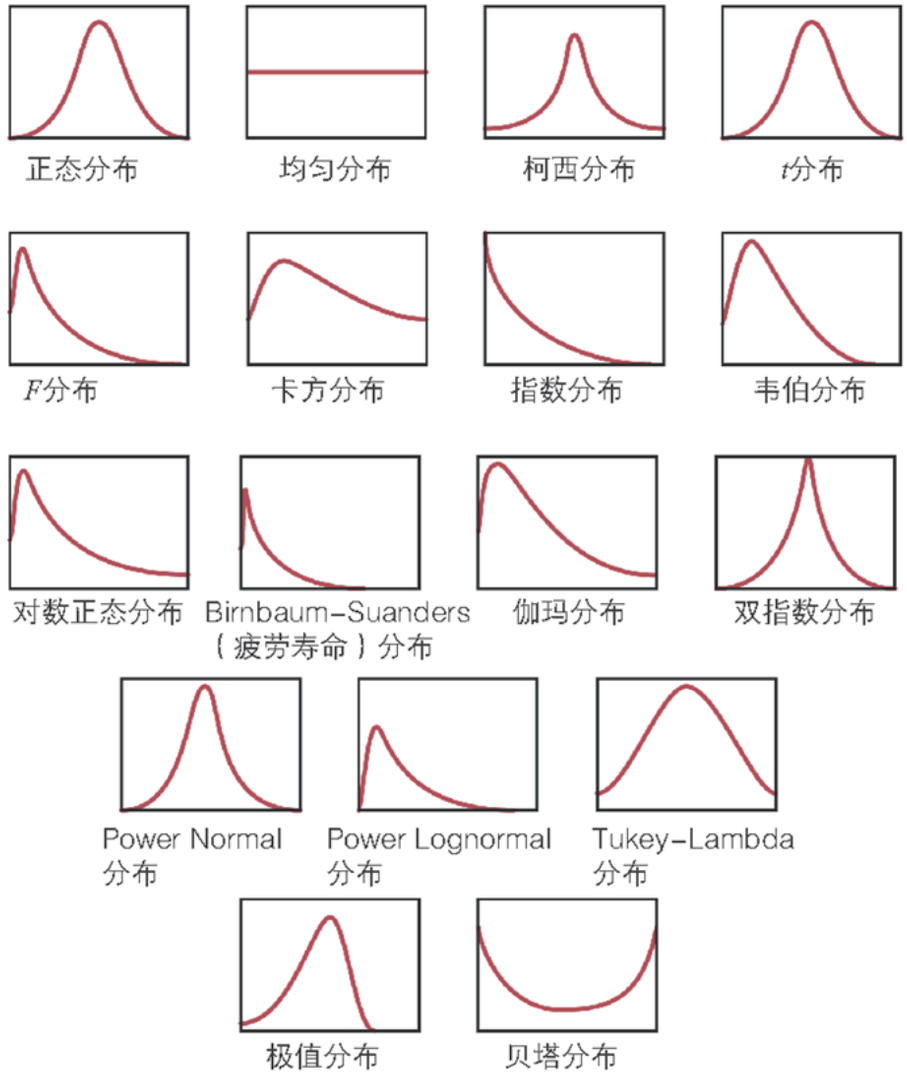


图 1　一组连续的密度函数（也叫概率分布）

假设随机变量x的概率分布为$p(x)$，该函数将x映射为一个实数，要使其成为概率密度函数，需要对其做以下限定：使用积分求曲线覆盖下的面积，则其值必须为 1，这样才能称其为概率。

比如，设x为距离下趟公交车到站的时间（以秒为单位），则x是一个随机变量，因为下趟公交车的到站时间是不定的。

假设我们已知（为了便于讨论）这个等待时间的概率密度函数为 $p(x) = 2e^{-2x}$ 图像说明文字，如果我们想知道下一趟车在等候 12~13 分钟后来的可能性，则只需要对于 12 至 13 之间该概率分布曲线下的区域使用积分图像说明文字 $\int_{12}^{13} 2e^{-2x}$ 求面积即可。

怎么知道该使用哪种概率分布？有两种方法：首先，可以做实验。我们可以随机到达公交车站，测量等候下一趟公交车需要的时间，重复该实验多次。然后将测量得到的数据绘制成散点图，看看与哪种概率分布曲线吻合。或者基于我们对“等待时间”是一种普遍的自然现象这一事实的了解，马上会想到采用指数分布图像说明文字 $p(x) = \lambda e^{-\lambda x}$ 去描述，指数分布就是专门发明用来描述自然界这种普遍现象的。

使用单变量函数可以描述一个随机变量的分布，描述多个随机变量则需要使用多变量函数，这称作**联合分布**。以两个随机变量为例，使用函数 $p(x, y)$ 表示概率分布，输入为平面上的点，输出为一个非负数。为了确保其是一个概率分布函数，在整个平面求二重积分，其值为 1。

还有一种分布叫条件分布$p(x|y)$，其含义是当y给定时x的概率密度函数。

处理数据时，条件意味着一个子集。比如，假设我们有 Amazon.com 网站的一组用户数据，该数据列出了每个用户上月在该网站的消费金额，不论性别，也不管在将第一件商品加入购物车前浏览过多少商品。

设随机变量X表示消费金额，则可以用$p(X)$表示用户的消费金额分布。

从所有用户中选取一个子集，该子集的用户在购买任何商品前至少浏览过5件商品，让我们看看这些用户上月消费金额的概率分布。设随机变量Y表示在购买第一件商品前浏览的商品数量，则$p(X|Y > 5)$表示**条件分布**。条件分布和一般的分布拥有一样的性质：求积分的结果为1，而且永远不会为负数。

当我们观察这些数据点时，如$(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$，我们是在观察随机变量的实例。当我们有一个$n$行$k$列的数据时，我们是在观察$k$个随机变量组成的联合分布的$n$个实例。

如果读者还想了解更多关于概率分布的知识，请参考Sheldon Ross所著的*A First Course in Probability*一书（Pearson）。

拟合模型

拟合模型是指用观察数据估计模型参数的过程。以数据为依据，近似模拟现实中产生数据的数学过程。拟合模型经常要引入各种优化方法和算法，例如最大似然估计等，来确定参数。

事实上，当你估计参数时，参数就成了估计量，他们本身就是数据的函数。当模型拟合成功，就能以数学函数的形式表达，比如$y = 7.2 + 4.5x$，其含义是，基于你对数据间存在线性关系的假设，该函数准确表达了两个变量之间的这种关系。

拟合模型的过程就是开始编写代码的过程：代码将会读入数据，将写在纸上的公式翻译成代码，然后使用R或者Python中内建的优化方法，根据数据，求出尽可能精确的参数值。

等你变得越来越老练，或者这本身就是你的强项时，你可能会去研究这些优化方法。首先得知道这些优化方法的存在，然后弄明白它们是怎么工作的，但是你不必亲自去编写代码实现这些方法，R和Python已经帮你实现好了，直接调用就行。

过拟合

过拟合甚至会成为你的梦魇，过拟合是指使用数据去估计模型的参数时，得到的模型并不能模拟现实情况，在样本以外的数据上效果不好。

在试图用该模型去预测另一组数据（该组数据未用来拟合模型）的标签时，你可能会发现结果不尽如人意，以准确度去衡量，这并不是一个很好的模型。

数据科学的工作流程

综上所述，让我们来看看如何定义数据科学的工作流程。你见的数据科学工作者越多，你就越会发现他们的工作流程符合图2的描述。

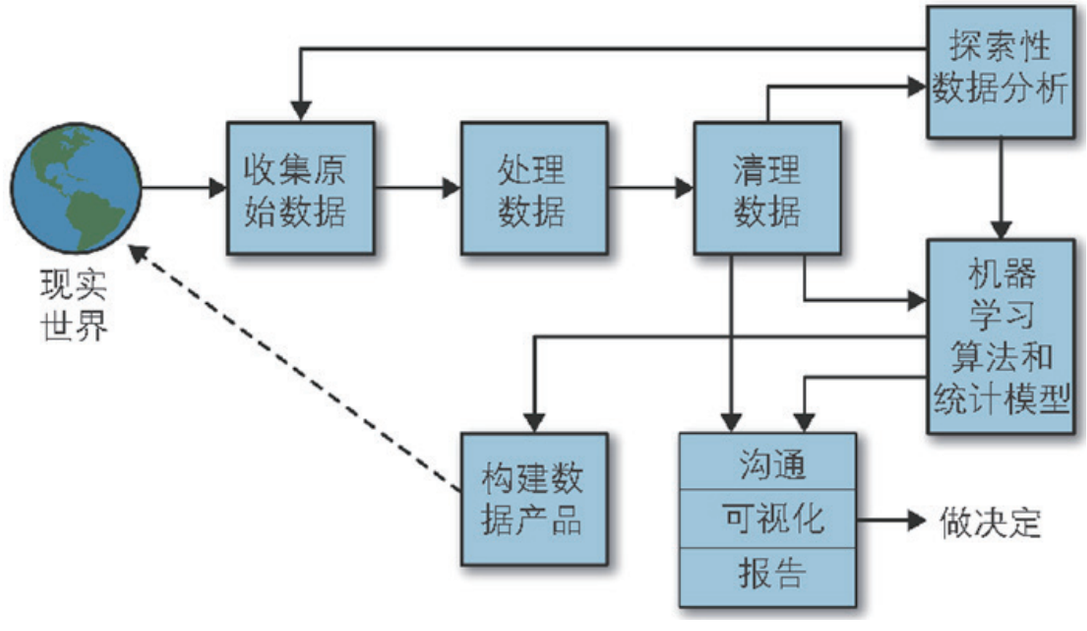


图2　数据科学的工作流程

首先，我们生活在这个世界中。在这个世界上，有很多人在从事各种各样的活动。有些人在使用Google+，另外一些人则在奥运会上一较高下；

有些人在制造、发送垃圾邮件，有些人则在医院里抽血。假设我们拥有其中某项活动的数据。

具体来说，以原始数据为起点，诸如日志、奥运会纪录、安然公司员工的电子邮件、遗传物质记录（需要注意的是，在我们拿到这些原始数据时，这项活动中某些方面的信息已经缺失了）。我们需要处理这些原始数据，使得其便于分析。因此我们创建出管道对数据进行再加工：联合、拼凑、清理，随便你叫它们什么好了，就是要对数据进行再加工。我们可以使用Python、shell脚本、R、SQL完成这件任务。

最终得到格式化好的数据，像下面这种由列构成的数据：

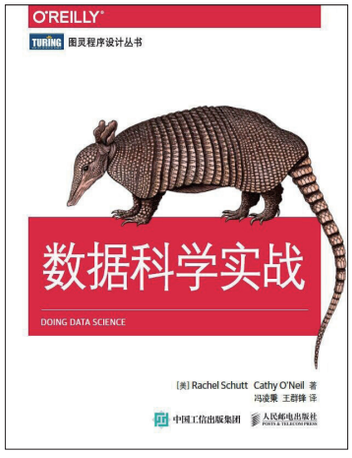
姓名 | 事件 | 年份 | 性别 | 时间

> 在标准的统计学课程中，通常从一份干净有序的数据文件开始，但在现实中，你通常不会有这么好的运气。

在拿到这份干净的数据后，我们应该先做一些探索性数据分析。在这个过程中，我们或许会发现数据并不是那么干净，数据可能含有重复值、缺失值或者荒谬的异常值，有些数据未被记录或被错误地记录。在发现上述现象时，我们不得不回过头采集更多的数据，或者花更多的时间清理数据。

然后，我们使用一些算法，比如k近邻、线性回归、朴素贝叶斯等设计模型。选取何种模型取决于要解决的问题，这可能是一个分类问题、一个预测问题，或者只是一个基本的描述问题。

这时就可以解释、勾勒、报告或者交流得到的结果。可以将结果报告给老板或同事，或者在学术期刊上发表文章，或者走出去参加一些学术会议，阐述我们的研究成果。



《数据科学实战》脱胎于哥伦比亚大学“数据科学导论”课程的教学讲义，它界定了数据科学的研究范畴，是一本注重人文精神，多角度、全方位、深入介绍数据科学的实用指南，堪称大数据时代的实战宝典。本书旨在让读者能够举一反三地解决重要问题，内容包括：数据科学及工作流程、统计模型与机器学习算法、信息提取与统计变量创建、数据可视化与社交网络、预测模型与因果分析、数据预处理与工程方法。另外，本书还将带领读者展望数据科学未来的发展。本文节选自《数据科学实战》。

如果我们的目标是开发一款数据产品或其产品原型，例如垃圾邮件分类、搜索排名算法、推荐引擎等。数据科学和统计学的不同之处就体现出来了，数据产品最终会融合到日常生活中，用户会和产品产生交互，交互会产生更多的数据，这样形成一个反馈的循环。

这和天气预报大相径庭，在预测天气时，你的模型对于结果没有任何影响。比如，你预测到下星期会下雨，除非你拥有某种超能力，否则不是你**让**天下雨的。但是假如你搭建了一个推荐系统，证明“很多人都喜欢某本书”，那就不一样了，看到这个推荐的人没准觉得大家都喜欢的东西应该不会太差，也喜欢上这本书了，这就形成了反馈。

在做任何分析时，都要将这种反馈考虑在内，以此对模型产生的偏差进行调整。模型不仅预测未来，它还在**影响**未来。

一个可供用户交互的数据产品和天气预报分别处于数据分析的两个极端，无论你面对何种类型的数据和基于该数据的数据产品，不管是基于统计模型的公共政策、医疗保险还是被广泛报道的大选调查，报道本身或许会左右观众的选票，你都要将模型对你所观察和试图理解的现象的影响考虑在内。

数据科学家在数据科学工作流程中的角色

到目前为止，所有这一切仿佛不需要人工干预，奇迹般地发生了。这里说的“人”，是指那些“数据科学家”。总得有人做出决定：该收集哪些数据？为什么要收集这些数据？她还要提出问题，做出假设，制定解决问题的方案。她就是数据科学家，或者她是我们推崇的数据科学团队。

让我们重新修订以前的流程，至少增加一层，来表明数据科学家需要全程参与到这一流程中来，他们不但需要在流程的较高层次上工作，还需要亲手编写程序，如图3所示：

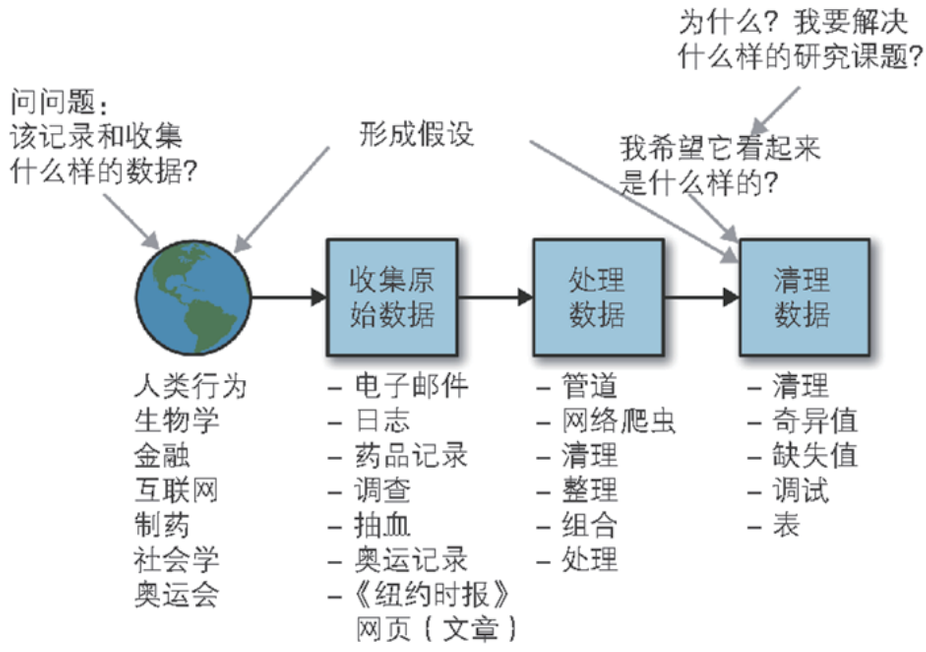


图3　数据科学家需要参与数据科学工作流程的各个环节

数据科学工作流程和其他科学方法的关系

数据科学工作流程，可以看作是其他科学方法的延伸或变体，它的一般步骤为：

- 提出问题；
- 做一些背景研究；
- 构想假设；
- 做实验验证构想的假设；
- 分析数据并得出结论；
- 把你的结果分享给其他人。

在数据科学工作流程和其他科学方法中，不是每个研究问题都需要按部就班地解决，大多数问题都不用严格走完每一步，几个步骤的组合就可能解决问题。比如，如果你的目标是对数据进行可视化（这本身也可以看成是一个数据产品），很可能你不会使用任何机器学习或统计模型，你只需要想方设法得到干净的数据，做一些探索性数据分析，将结果用图表的形式展示出来即可。

专题：如何成为一位数据科学家

在AWS上构建你的第一台机器

作者 / Willi Richer

机器学习和机器人学博士，目前任职于微软Bing搜索核心研发团队。他从事多种机器学习领域的研究，包括主动学习和统计机器翻译。

当你有很多数据、需要进行很多计算的时候，你可能会开始渴求更多的计算资源。亚马逊http://aws.amazon.com允许你按小时租用计算资源。因此，你可以访问到大量的计算资源，而不需要预先购买大量机器（包括管理基础设施的费用）。在这个市场里还有其他的竞争者，但亚马逊是最大的玩家。

亚马逊Web服务（Amazon Web Services，AWS）是一组庞大的服务。我们只关注其中的**Elastic Compute Cluster (EC2)**服务。这个服务提供给你虚拟机和磁盘空间，它们可以很快被分配和释放。

它一共有三种使用模式：保留模式，你可以预先支付，来获得更廉价的按小时的访问；每小时固定率；根据整体计算市场（需求少的时候，费用也低，但需求多的时候，价钱就会提高）提供变化的比率。

如果想测试的话，你可以在免费层级（free tier）中使用一台机器。它允许你试验一下这个系统，熟悉一下接口，等等。然而，这是一台CPU非常慢的机器。所以，我们并不建议用它进行过多的计算。

在整体系统的上层，有几种类型的机器，它们的费用不同；从单核到多核大内存系统，或者甚至是**图形处理单元**（Graphical Processing Unit，GPU）。我们之后会看到，你可以得到几台比较廉价的机器，并构建你自己的虚拟集群。你还可以选择Linux或是Windows服务器，其中

Linux会便宜一点。在本文里，我们是在Linux上运行我们的例子的，但多数东西在Windows机器上也可以使用。

这些资源可以通过Web界面来进行管理。但也可以在程序中通过脚本来分配虚拟机，设置磁盘，以及所有Web界面所能做的操作。事实上，在Web界面频繁变化的同时，编程接口更为稳定。由于服务的引入，整体构架也保持着稳定。

通过传统的用户名/密码就可以访问AWS服务，尽管亚马逊把用户名叫做公钥，把密码叫做私钥。这样做是为了使访问Web接口的用户名和密码分开。事实上，你可以生成很多公/私钥对，并给予它们不同的权限。对于比较大的团队来说（一个能访问所有Web界面的高级用户可以为权限较少的开发者创建密钥），这样做是有益处的。

注意　亚马逊区域

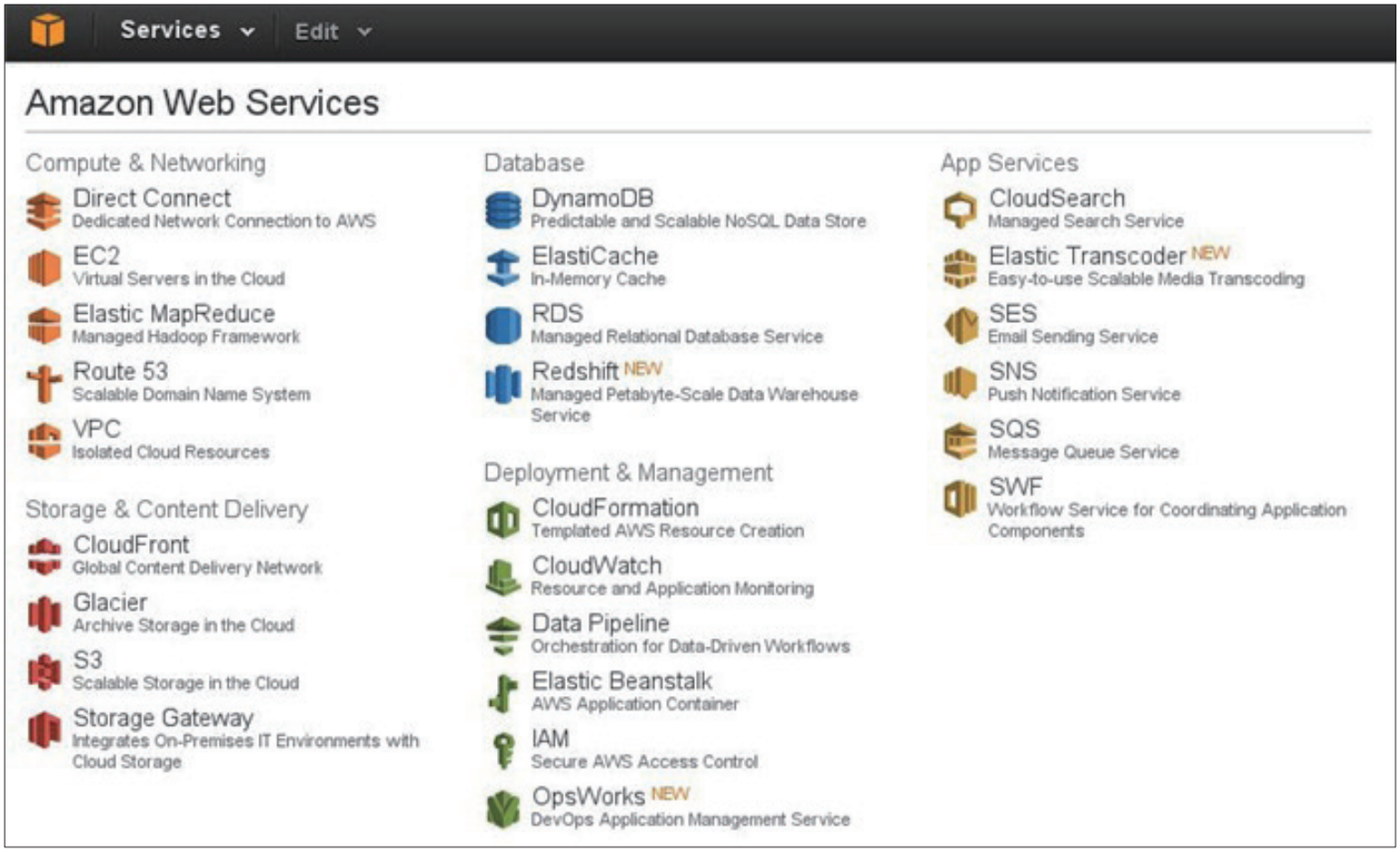
Amazon.com有几个区域。它们与世界的物理区域相对应：美国西海岸、一些亚洲地点、南美区，以及欧洲区。如果你想迁移数据，最好使迁移的出发地和目的地比较靠近。此外，如果你正在处理用户的信息，并且要迁移到另一个管辖区去，那就可能会出现监管问题。在这种情况下，一定要让一个知情律师对欧洲客户数据迁移到美国的影响进行检查；反之亦然。

亚马逊Web服务是一个非常宏大的课题，市面上有各种可以涵盖AWS全部内容的书。我们只是要给你一个整体印象，告诉你AWS有什么，能做什么。基于实践精神，我们先看一些例子，但我们不会穷尽所有可能。

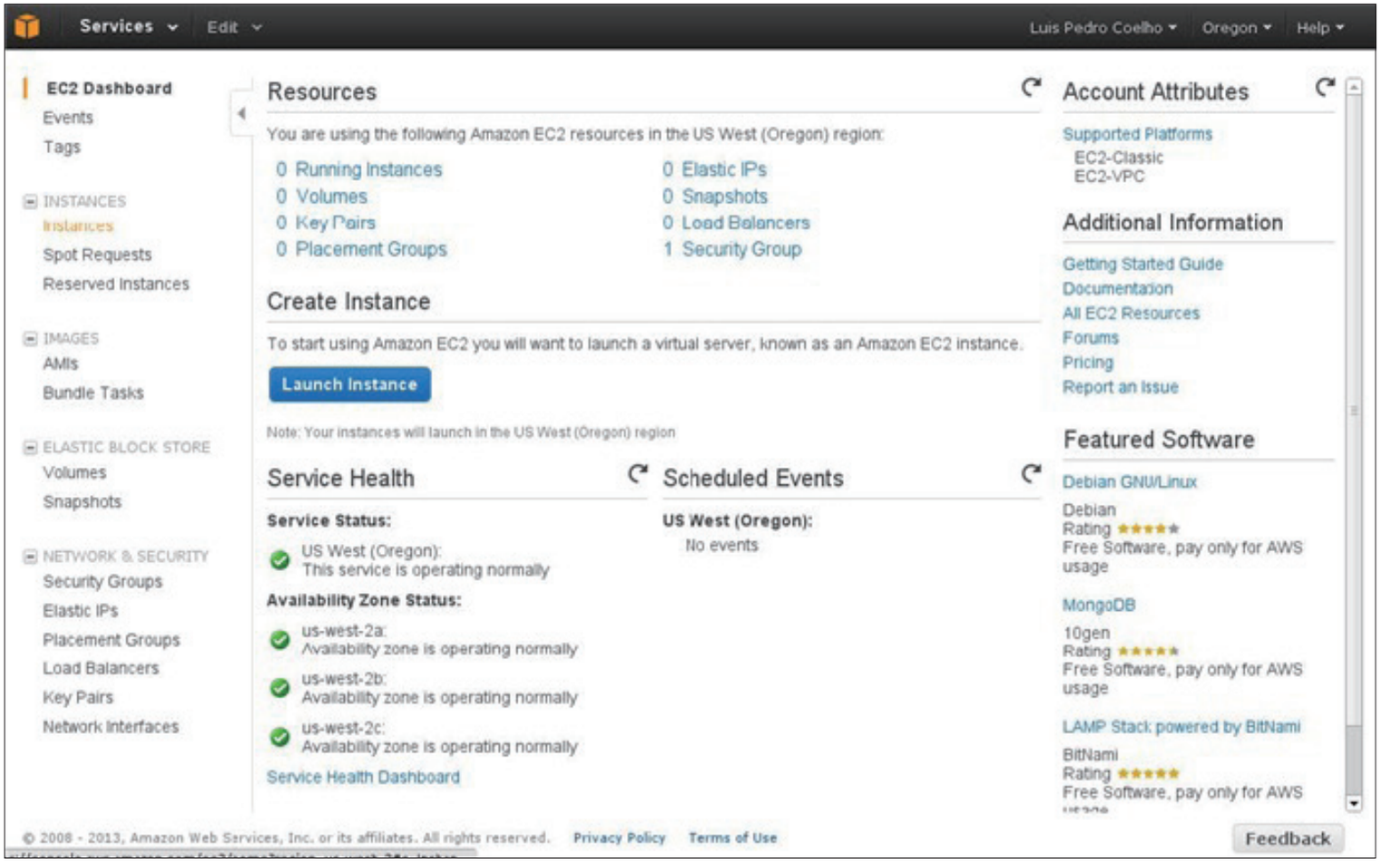
构建你的第一台机器

第一步是到http://aws.amazon.com/创建一个账号，这里的步骤跟其他在线服务类似。如果你想要的是比一台低端机器更好的机器，那你需要一张信用卡。在这个例子里，我们会使用一些机器，如果你想在上面运行程序的话，可能要花费一些钱。如果你还没准备掏出信用卡，那应该先阅读本文，了解一下AWS可以提供什么服务。然后，你可以做出更明智的选择，决定是否要进行注册。

一旦注册了AWS并登录，你会看到它的控制台。在这里，你可以看到AWS提供的多种服务：

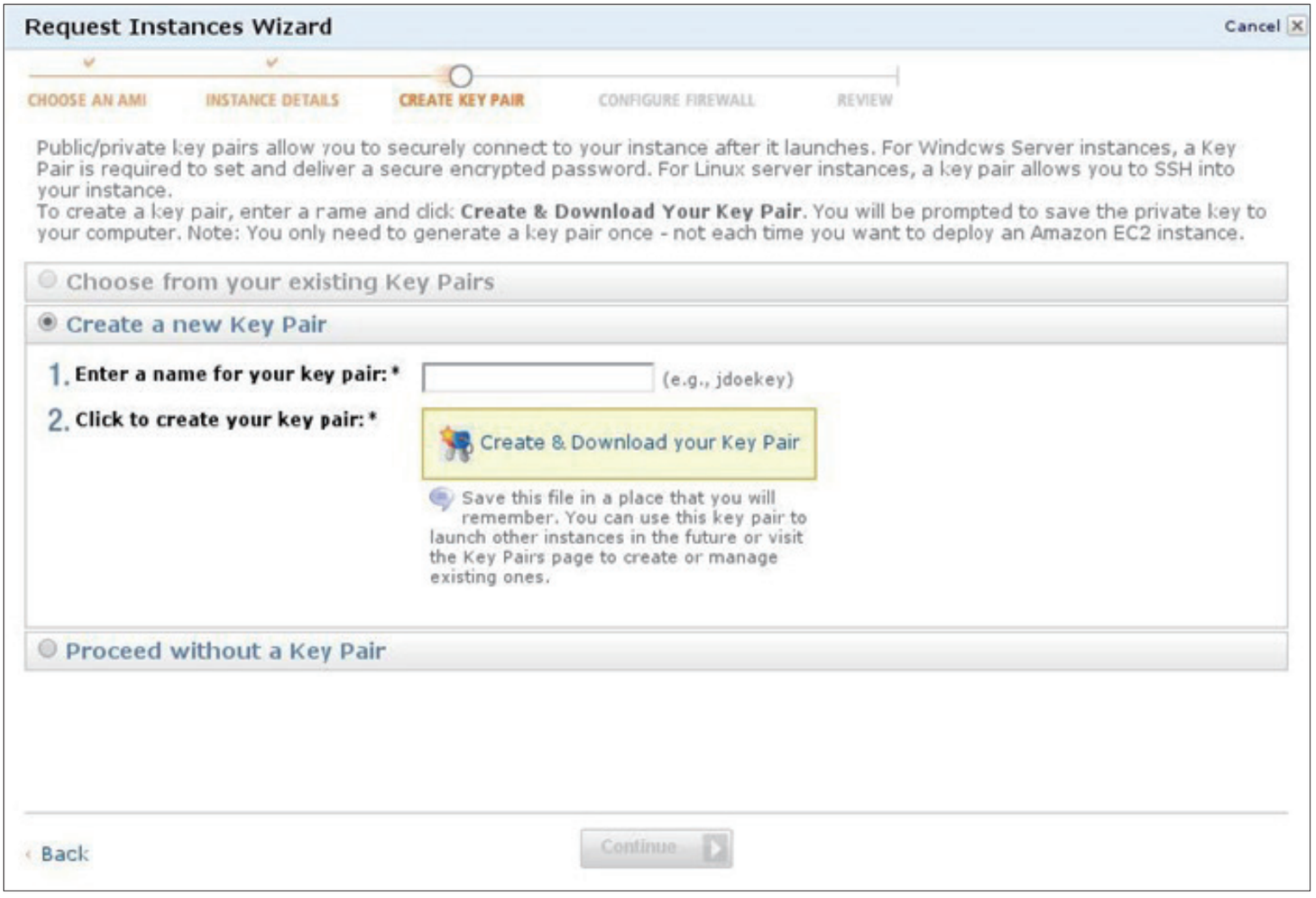


我们选择并点击**EC2**（最左列的第二项，在本文撰写之时，界面是这样显示的；亚马逊经常会进行小修小改，所以你看到的可能有些不同）。我们现在看看EC2的管理控制台，如下图所示：



在右上角，你可以选择你的区域（见美国区域信息框）。注意，你只能看到所选区域的信息。因此，如果选择了错误的区域（或者在不同区域都有机器在运行），这些内容可能不会出现（在同其他程序员的聊天中得知，这似乎是使用EC2 Web管理控制台的一个常见陷阱）。

用EC2的说法，一个正在运行的服务器叫做一个**实例**（instance）。所以现在，我们要选择**Launch Instance**。现在，遵循典型惯例。选择**Amazon Linux**选项（如果你熟悉下列Linux发行版本之一，如Red Hat、SuSe或Ubuntu，可以选择其中之一，但配置将会有些不同）。我们从T1.micro类型的一个实例开始。这是最小的机器，并且是免费的。接受所有默认选项，直到进入下面这个输入密钥对后出现的界面：



我们选择名字awskeys作为密钥对。然后点击**Create & Download your Key Pair**按钮，下载awskeys.pem文件。这是**Secure Sheel (SSH)**，将使你可以登录云机器。要把文件保存在安全的地方！接受完剩下的默认选项，你的实例就启动了。

现在你需要等待一小会儿，等实例出现。最终，这个实例会用绿色显示，状态为“运行中”。右击并选择**Connect**选项，你会知道如何连接。下面是一个标准SSH命令：

```
ssh -i awskeys.pem ec2-user@ec2-54-244-194-143.us-west-2.compute.
amazonaws.com
```

因此，我们会调用ssh命令，并把之前下载的密钥文件传进去，作为身份信息（用-i选项）。我们会以用户ec2-user来登录地址为ec2-54-244-194-143.us-west-2. conompute.amazonaws.com的机器。当然，这个地址跟你的不同。如果在实例中选择另一个版本，用户名也会改变。不管怎样，Web界面会给你正确的信息。

最后，如果是在一个Unix风格的操作系统上（包括Mac OS）运行，你需要通过下面的代码调整它的权限：

```
chmod 600 awskeys.pem
```

这个命令会给当前用户设置读/写权限。否则，SSH会给你一个令人厌恶的警告。

现在，你应该可以登录主机了。如果一切顺利，你会看到下面的标语：

```
       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2013.03-release-notes/
There are 1 security update(s) out of 3 total update(s) available
Run "sudo yum update" to apply all updates.
```

这是一个常规的Linux框，你拥有sudo权限；如果在前面加上sudo，你就可以像高级用户那样运行任何命令。你可以运行更新（update）命令让机器加速。

1. 在亚马逊Linux上安装Python包

如果你要使用另一个分发版本，可以用另一个分发版本的知识来安装Python、NumPy或者其他。在这里，我们采用的是标准亚马逊分发版本。从安装几个基本的Python包开始，如下所示：

```
sudo yum -y install python-devel python-dev python-pip numpy
scipy
python-matplotlib
```

要对mahotas进行编译，还需要一个C++编译器：

```
sudo yum -y install gcc-c++
```

在这个系统里，pip被安装成python-pip。为方便起见，我们用pip来更新它自己。然后我们用pip安装必要的程序包：

```
sudo pip-python install -U pip
sudo pip install scikit-learn jug mahotas
```

在这里，你可以像使用pip那样安装任何其他程序包。

2. 在我们的云主机上运行Jug

现在，我们可以去下载数据和代码了。如下所示：

```
wget FIXME-LET'S-BUILD-A-TAR-GZ-PACKAGE
tar xzf chapter10.tar.gz
cd chapter10
```

然后，进行如下操作：

```
jug execute
```

它工作得还可以，但我们需要等很长时间才能得到结果。我们的免费级机器（`t1.micro`类型）是单核的，不是很快。所以我们要升级机器。

我们回到EC2控制台，在运行实例上右击，会出现一个弹窗。我们需要先停掉这个实例。在虚拟机上停止一个实例的运行就如同把它的电源关掉。你可以在任何时候停止你的机器。这时，你不再为它们承担费用（你仍然在使用磁盘空间，这也是有代价的，会分开计算费用）。

一旦你的机器停止了，**change instance type**选项就会出现。你可以选择一个更强有力的实例，例如，一个`c1.xlarge`实例，它是8核的。这个机器仍然处于关闭状态，你需要重新打开它（这跟重启机器是一样的）。

提示

AWS提供了几种价钱不同的实例类型。由于引入了更强有力的选项，信息经常会不断修正，同时价格也会发生变化，所以在这里我们没法给你很多具体细节（一般来说会变得更便宜）；然而，你可以在亚马逊的网站上找到最新的信息。

我们需要等待实例再次出现。一旦它出来了，右击**Connet**获得连接信息。几乎可以肯定，链接信息已经改变了。在你更改实例类型的时候，你的实例会获得一个新分配的地址。

提示

你可以用亚马逊的Elastic IPs功能给实例分配一个固定IP。你可以在EC2控制台的左边看到这个选项。

通过使用8核机器，你可以同时运行8个jug进程，如下面这些代码所示：

```
jug execute &
jug execute &
(repeat to get 8 jobs going)
```

用`jug status`可以查看到8个作业是否在运行。完成作业之后（现在应该很快就可以实现），你可以停止机器，并再次降级为`t1.micro`实例，以节省花销；微型实例是免费的，而这个特大号的机器的价钱是每小时0.58美元（这是2013年4月的价钱——到AWS网站上可以看到信息）。

用starcluster自动创建集群

正像你所了解到的，我们可以通过Web界面创建机器，但你很快就会发现，这个操作单调枯燥，而且容易出错。幸运的是，亚马逊有一个API。你可以写脚本自动执行之前讨论过的所有操作。更好的是，其他人已经开发了一些工具。你用AWS要进行的很多过程都可以用这些工具自动化执行。

MIT的一个团队正好开发了这样一个工具，叫做starcluster。它恰好是一个Python包，所以你可以用Python工具把它安装上。

```
sudo pip install starcluster
```

你可以在亚马逊主机上，或你的本地机器上运行这个代码。这两种方式都行。

我们需要指定我们的集群是什么样的。通过编辑配置文件即可实现。下面的代码生成了一个模板配置文件：

```
starcluster help
```

然后，我们选择选项在-/.starcluster/config中生成配置文件。做完之后，

我们就可以手动编辑这个文件。

> **提示　不同的密钥**
>
> 在使用AWS的时候，有三种密钥。
>
> - 标准用户名/密码组合，用于登录网站。
> - SSH密钥系统，它是一个用文件实现的公/私钥系统；通过你的公钥，可以登录远程主机。
> - AWS访问钥/密钥系统，这只是某种形式的用户名/密码。它允许你在同一个账户下拥有多个用户（包括为每个用户添加不同权限）。
>
> 要想查看访问/私钥，我们回到AWS控制台，在右上角点击我们的名字；然后选择**Security Credentials**。现在，屏幕的下方应该会出现形如AAKIIT7HHF6IUSN3OCAA的访问钥，本文将以它为例。

现在编辑配置文件。这是一个标准的.ini文件：一个文本文件，每个部分以方括号和它们的名字开始。选项的格式是`name=value`。第一部分是`aws info`，你应该把你的密钥粘贴过来：

```
[aws info]
AWS_ACCESS_KEY_ID = AAKIIT7HHF6IUSN3OCAA
AWS_SECRET_ACCESS_KEY = <your secret key>
```

现在来到一个有趣的部分：定义一个集群。starcluster允许你定义任意多个不同的集群。文件的开始有一个叫做`smallcluster`的集群。它在`cluster smallcluster`部分里定义。把它编辑为：

```
[cluster mycluster]
KEYNAME = mykey
CLUSTER_SIZE = 16
```

它把节点个数从默认的2改为16。我们还可以指定每个节点的实例类型和初始映像（记住，映像是指你使用的是什么操作系统，安装了什么软件）。starcluster有一些预设的映像，但你也可以构建自己的映像。

我们需要创建一个新的ssh密钥，如下所示：

```
starcluster createkey mykey -o .ssh/mykey.rsa
```

现在我们已经配置了一个16节点的集群，并设置了密钥。让我们尝试一下。

```
starcluster start mycluster
```

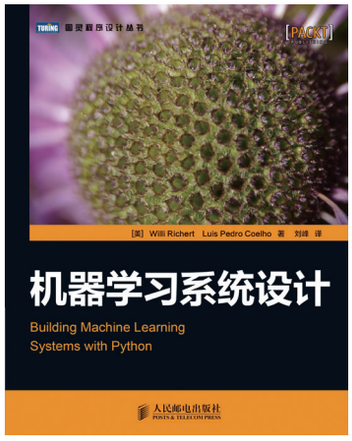
这个操作可能需要几分钟，因为它要分配17台新机器。为什么是17台机器，而我们的集群只有16个节点？这是因为会有一个主节点。所有这些节点都使用同一个文件系统，所以在主节点上创建的任何东西，也可以在从属节点上看到。这也意味着我们可以在这些集群上应用Jug。

这些集群可以按照你想要的方式进行使用，但它们都预装了一个作业队列引擎，这种方式对于批处理作业比较理想。它的使用过程很简单。

1. 登录主节点。
2. 在主节点上准备好你的脚本（更好的方式是，在之前就把脚本准备好）。
3. 把作业提交到队列。一个作业可以是任何一个Unix命令。调度程序会寻找空闲节点来运行你的作业。
4. 等待作业结束。
5. 从主节点上读取结果。你现在还可以杀掉所有从属节点，以节约费用。在任何时候都不要忘记你的系统的运行是长期的。否则，这会花费很大（意思是美元和美分）。

我们可以用一个命令登录到主节点上：

```
starcluster sshmaster mycluster
```



如今，机器学习正在互联网上下掀起热潮，而Python则是非常适合开发机器学习系统的一门优秀语言。作为动态语言，它支持快速探索和实验，并且针对Python的机器学习算法库的数量也与日俱增。《机器学习系统设计》最大的特色，就是结合实例分析教会读者如何通过机器学习解决实际问题。举几个例子，我们会介绍怎么把StackOverflow的回答按质量高低进行分类，怎么知道某个音乐文件是爵士风格，还是重金属摇滚风格。另外，本书还涵盖了主题建模、购物习性分析及云计算等高级内容。总之，通过学习本书，读者可以掌握构建

我们也可以看到我们之前用ssh命令生成的机器地址，但在使用前面那个命令的时候，地址是什么并不重要，因为starcluster已经在背后把这些都处理好了。

正如我们前面所说的，starcluster为集群提供了一个批处理排队系统；你可以写脚本执行你的操作，把作业放在队列里，这些作业就会在空闲节点上运行。

在这一点上，你需要在集群上重复安装必要的程序包。如果这是一个真实项目，我们可以创建一个脚本来执行所有这些初始化工作，但由于这是一个教程，你只需要再初始化一次。

我们可以像以前一样使用相同的jugfile.py系统，但现在，我们会在集群中进行调度，而不是直接在主节点上运行。首先，写一个很简单的脚本：

```
#!/usr/bin/env bash
jug execute jugfile.py
```

用run-jugfile.sh调用它，并用chmod +x run-jugfile.sh赋予它可执行权限：

```
For c in 'seq 16'; do qsub run-jugfile.sh; done
```

这会创建16个任务，每个都会运行run-jugfile.sh脚本调用Jug。你仍然可以使用主节点。需要特别强调的是，你可以在任何时候运行jug status查看计算状态。事实上，Jug就是在这种环境下开发出来的，所以它在这个环境下工作得很好。

最后，计算结束，可以把所有节点都删掉了。但我们一定要把预期的结果保存在某个地方，并执行下述命令：

```
starcluster terminate mycluster
```

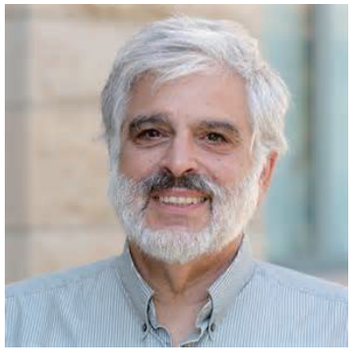
自己所需系统的各方面知识，并且学以致用，解决自己面临的现实问题。本文节选自《机器学习系统设计》。

注意，终止操作将会销毁文件系统中的内容以及所有运行结果。当然，这个默认设置是可以更改的。你可以让集群把结果写在一个不是由starcluster分配并销毁的而你又可以访问的文件系统里。事实上，这些工具的灵活性非常大。不过这些高级操作并不适合在本文里展开。

starcluster有一个优秀的在线文档，见http://star.mit.edu/cluster/，你可以读到关于这个工具的更多信息。我们在这里只看到了一小部分功能，只使用了默认的设置。■

专题：如何成为一位数据科学家

针对PageRank的链接作弊方法剖析



作为 / Jeffrey David Ullman

美国国家工程院院士，计算机科学家，斯坦福大学教授。Ullman早年在贝尔实验室工作，之后任教于普林斯顿大学，十年后加入斯坦福大学直至退休，一生的科研、著书和育人成果卓著。他是ACM会员，曾获SIGMOD贡献奖、Knuth奖等多项科研大奖；他是“龙书”《编译原理》、数据库领域权威指南《数据库系统实

21世纪这十年来人们生活的一个最大改变就是，可以通过谷歌之类的搜索引擎高效准确地进行Web搜索。尽管谷歌并非最早的搜索引擎，但是它却是第一个能够战胜作弊者的搜索引擎，这些作弊者几乎使得搜索毫无意义。此外，谷歌也提供了一个称为PageRank的非凡的创新技术。

当然，发掘Web使用价值的人和别有用心为私利服务的作弊者之间的斗争永不停止。当PageRank作为搜索引擎的核心技术使用时，作弊者也发明了一些人为操纵网页PageRank的方法，这些方法统称为链接作弊（Link Spam）。这使得有人提出了TrustRank以及其他的一些技术来对抗作弊者对PageRank的攻击。

PageRank

PageRank是一种不容易被欺骗的计算Web网页重要性的工具。为了介绍PageRank概念提出的动机，我们首先会简单回顾搜索引擎的部分历史。

早期的搜索引擎及词项作弊

在谷歌之前出现过很多搜索引擎。其中大部分都是利用网络爬虫从Web上抓取数据，然后通过倒排索引方式列出每个页面所包含的词项（term，通常为词或者其他非空格字符组成的字符串）。倒排索引（inverted index）是一种很容易从给定词项找到（指向）它所在的所有网页位置的数据结构。

现》的合著者；麾下多名学生成为了数据库领域的专家，其中最有名的当属谷歌创始人Sergey Brin；本书第一作者也是他的得意弟子。Ullman目前任Gradiance公司CEO。

当提交一个搜索查询（search query，一般指词项列表）时，所有包含这些词项的网页会从倒排索引中抽取出来，并按照能够反映页面内词项作用的某种方式来排序。因此，词项出现在网页头部，会使得该网页的相关性会比词项出现在普通正文中的网页更高。同时，词项的出现次数越多，网页的相关性也会增加。

当人们开始使用搜索引擎在Web上来寻找相关信息时，有些不道德的人发现这是欺骗并利用搜索引擎让人们来访问他们的网页的“良机”。因此，一个在Web上卖T恤的人所关心的只是让人们来访问他的网页，而不管用户在网上找什么。于是，作弊者可以在他的网页上增加一个词项，如“movie”，并将该词项重复几千次，搜索引擎可能认为该网页与“movie”高度相关，因此当用户输入查询“movie”时，搜索引擎就可能把该网页放在第一位。为了掩盖网页上重复出现的“movie”，作弊者还可以将这些词的颜色和背景色设成一致。如果简单地在网页中加入“movie”并不奏效的话，作弊者还可以提交查询“movie”给搜索引擎，然后将第一个结果的内容直接复制到他的网页中，同样也可以采用和背景色一致的做法使得这些内容对浏览者并不可见。

这种欺骗搜索引擎让它们相信一个本来不相关的页面相关的技术称为词项作弊（term spam）。词项作弊者很容易实施作弊行为使得早期的搜索引擎几乎不可用。为了对抗词项作弊，谷歌提出了两项创新。

(1) 使用了PageRank技术来模拟Web冲浪者的行为，这些冲浪者从随机页面出发，每次从当前页面随机选择出链前行，该过程可以迭代多次。最终，这些冲浪者会在页面上汇合。较多冲浪者访问的网页的重要性被认为高于那些较少冲浪者访问的网页。谷歌在决定查询应答顺序时，会将重要的网页排在不重要的网页前面。

(2) 在判断网页内容时，不仅只考虑网页上出现的词项，还考虑指向该网页的链接中或周围所使用的词项。值得注意的是，虽然作弊者很容易在它们控制的网页中增加虚假词项，但是在指向当前网页的网页上添加虚假词项却并不那么容易，除非他们控制了这些网页。

简化的PageRank无法奏效

通过模拟随机冲浪者的方式计算PageRank的时间开销很大。有人可能认为，只是计算每个网页的入链数目来估计冲浪者的结束位置概率会是一个好的近似方法。但是，如果我们只是这样做的话，那么假想的T恤商可以构建一个上百万网页构成的垃圾农场（spam farm），农场中的每个网页都指向T恤销售的网页。于是，该网页将看上去可谓非常重要，从而欺骗搜索引擎。

上述两种技术综合在一起可以使刚才假想的T恤商难以欺骗谷歌。虽然T恤商仍然可以在网页中加入“movie”，但谷歌却认为别的网页对它的评价比自己的评价更重要，因此假词项的加入基本无效。当然，T恤商的明显对策是建立多个网页，并增加这些网页到T恤销售网页的链接，每个链接上增加“movie”字样。但是由于其他网页可能并不会链向这些构造的网页，因此这些网页的PageRank不高。即使T恤商在他构造的网页之间增加很多链接，但是按照PageRank算法，这些网页的PageRank值仍然不高。因此，T恤商的这些行为并不会让谷歌相信他的页面与movie有关。

我们有理由问这样一个问题，为什么随机冲浪者的模拟过程允许我们对刚才提出的网页“重要性”的直观概念进行近似估计？两种相互关联的动机引发了上述做法。

- Web用户会“用脚投票”。他们倾向于链接那些自己认为较好的或有用的网页，而不愿链接那些糟糕或无用的页面。
- 随机冲浪者的行为表明Web用户可能访问哪些网页。用户更可能访问有用而不是无用的网页。

但是，不管原因如何，PageRank方法的有效性已经在实际中得到验证。

PageRank的定义

PageRank是一个函数，它对Web中（或者至少是抓取并发现其中链接关系的一部分Web网页）的每个网页赋予一个实数值。它的意图在于，网页的PageRank越高，那么它就越“重要”。并不存在一个固定的PageRank分配算法，实际上，一些基本方法的变形能够改变任意两个网页的相对PageRank值。接下来我们首先给出最基本也最理想化的PageRank的定义，然后给出面对真实Web结构时对基本PageRank所做的必要修改。

Web可以想象成一个有向图，其中网页是图中节点，如果网页p1到p2之间存在一条或者多条链接，则p1到p2存在一条有向边。图1给出了一个非常小版本的Web图的例子，该图只包括4个网页，页面A到其他三个页面B、C、D都存在链接，页面B只链向A和D，页面C只链向A，而D只链向B和C。

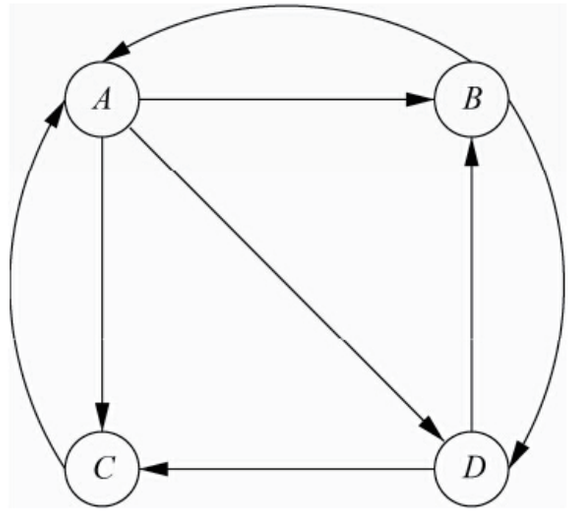


图1　一个假想的Web的例子

假定一个随机冲浪者从图1的页面A出发，由于A链向B、C和D，所以他会以各1/3的概率分别访问B、C和D，但是下一步继续访问A的概率为0。同样，到达B点的随机冲浪者下一步会分别以1/2的概率访问A和D，而到B或C的概率为0。

一般地，可以定义一个Web转移矩阵（transition matrix）来描述随机冲浪者的下一步访问行为。如果网页数目为n，则该矩阵M是一个n行n列的方阵。如果网页j有k条出链，那么对每一个出边链向的网页i，矩阵第i行第j列的矩阵元素m_{ij}值为1/k。而其他网页i的$m_{ij} = 0$。

垃圾农场的架构

为了提高某个或者某些特定网页的PageRank的目的而构建的一个网页集合称为垃圾农场（spam farm）。图2给出了垃圾农场的最简单形式。按照作弊者的观点，整个Web分成三部分。

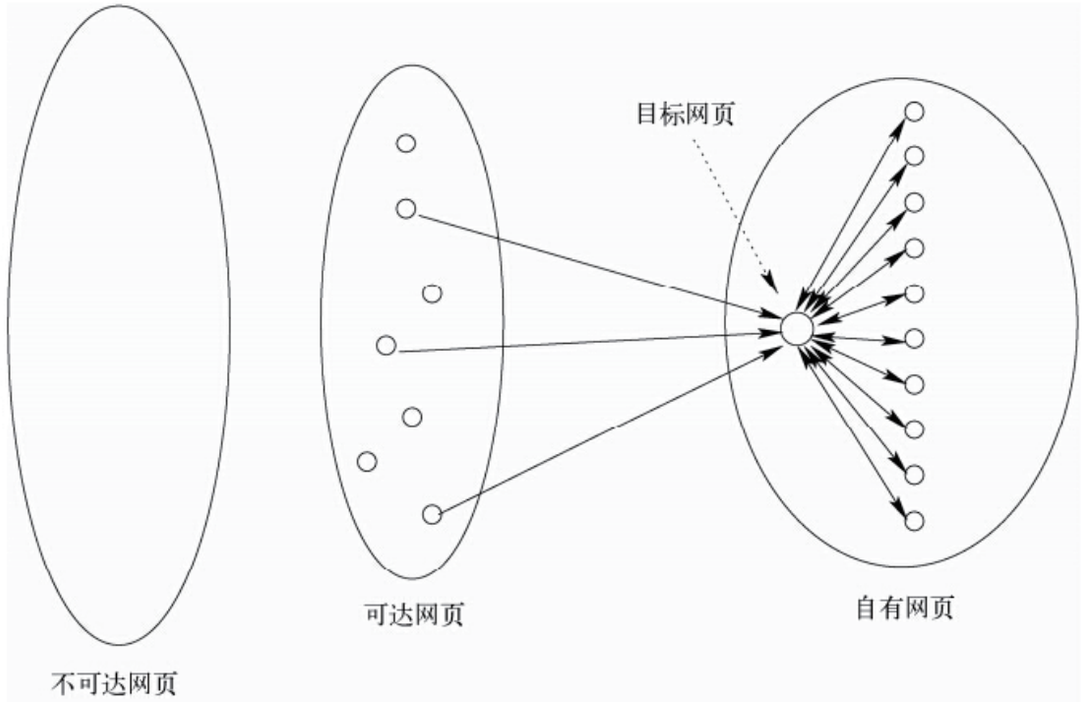


图2　链接作弊者眼中的Web组成结构

(1) 不可达网页或不可达页 (inaccessible page) 即作弊者无法影响的网页。Web 中的大部分网页都属于这一类。
(2) 可达网页或可达页 (accessible page) 这些网页虽然不受作弊者控制，但是作弊者可以影响它们。
(3) 自有网页或自有页 (own page) 作弊者拥有并完全控制的网页。

作弊者的自有网页按照某种特定方式来组织 (参考图 2 右部)，而垃圾农场由作弊者的自有网页和从一些可达网页指向它们的链接共同组成。如果没有外部指入的链接，垃圾农场就不可能被一般搜索引擎的采集器所采集，因而毫无价值。

至于可达网页，看起来可能有些奇怪，因为在不归作弊者所有的情况下它们却受影响。但是，现在有很多网站，如博客或者报纸等，它们都邀请用户在网站上发表评论。为了从外部流入尽可能多的 PageRank 到自有网页，作弊者会发表很多诸如“我同意。请访问链接 www.mySpamFarm.com 来阅读我的文章”之类的评论。

垃圾农场中有一个页面 t 称为目标网页或目标页 (target page)，作弊者试图尽可能将更多的 PageRank 放在该页面上。存在数目为 m 的大量支持网页或支持页 (supporting page) 用于积聚平均分发给所有网页的 PageRank (即 PageRank 的 $1-\beta$ 部分，它代表冲浪者到达一个随机页面)。由于每一轮迭代都有随机跳转的可能，因此支持页面同时也可以尽量避免页面 t 的 PageRank 被流失。需要注意的是，t 到每个支持页面都有链接，而每个支持页面同时都只链向 t。

垃圾农场的分析

假定 PageRank 计算时的参数为 β，通常取 0.85 左右。也就是说，该参数 β 代表的是当前网页的 PageRank 在下一轮迭代被分发给其链向网页的比例。假设整个 Web 中有 n 个网页，其中部分网页构成图 2 所示的垃

圾农场，其中包含一个目标网页t和m个支持网页。令x为所有可达网页为垃圾农场所提供的PageRank总量，也就是说，x是所有指向t的可达网页p的PageRank分量之和，即每个p的PageRank乘以β然后除以p的出度数之后进行累加求和。最后，令y为t的未知的PageRank，接下来要求出y。

首先，每个支持网页的PageRank为

$$\beta y/m + (1-\beta)/n$$

上式的第一项代表t的贡献，t的PageRank y会被“抽税”，因此只有βy会分发给t的链向网页。该PageRank会平均分给m个支持网页。第二项来自整个Web中所有网页所平均分到的那一部分。接下来我们计算目标网页t的PageRank y，其来自三个信息源：

- (1) 刚才假设的外部贡献x；
- (2) β 乘以每个支持网页的PageRank，即 $\beta(\beta y/m + (1-\beta)/n)$；
- (3) $(1-\beta)/n$，同样这一部分来自Web中所有网页平均分到的那一部分。这部分值很小，为简化分析，将在后面忽略。

因此，基于上述(1)和(2)，有

$$y = x + \beta m\left(\frac{\beta y}{m} + \frac{1-\beta}{n}\right) = x + \beta^2 y + \frac{\beta(1-\beta)m}{n}$$

解上述方程可得

$$y = \frac{x}{1-\beta^2} + c\frac{m}{n}$$

其中$c = \beta(1-\beta)/(1-\beta^2) = \beta/(1+\beta)$。

例 1　如果选择$\beta = 0.85$，那么$1/(1-\beta 2) = 3.6$，$c = \beta/(1+\beta) = 0.46$。也就是说，上述结构能够把外部的PageRank贡献放大到360%，并且还可以获得垃圾农场网页数目和总的网页数目的比值m/n的46%。

与链接作弊的斗争

对于搜索引擎来说，检测并消除链接作弊已经变得至关重要，其必要性就如同上个十年对付词项作弊一样。对付链接作弊有两种方法。第一，查找如图2给出的垃圾农场结构，即其中某个网页链向大量网页，而这些网页又都回指这种结构。搜索引擎肯定能够找出满足这些结构的网页并将它们从索引中消除。但是这样一来，作弊者又提出新的不同的结构，这些结构本质上同样可以达到提高某个或某些目标网页的PageRank的效果。也就是说，从本质上而言，图2给出的结构有无数变形，所以作弊者和搜索引擎之间的斗争可能将会存在很长一段时间。然而，有另外一种做法可以在不依赖链接作弊定位的同时去掉这些链接。更确切地说，搜索引擎可以修改PageRank算法的定义来自动降低链接作弊网页的重要度。下面将考虑两个不同的计算公式：

- (1) TrustRank面向主题的PageRank的一种变形，设计为可以降低垃圾网页的得分；
- (2) 垃圾质量（spam mass）能够识别可能为垃圾的网页并允许搜索引擎去掉这些网页或者大力降低这些网页的PageRank的计算。

TrustRank

TrustRank是面向主题的PageRank，其中这里所说的“主题”指的是一个值得信赖的可靠网页集合（非垃圾网页）。其基本的理论为，虽然垃圾网页可能很容易链向可靠网页，但是可靠网页却不太可能会链向一个垃圾网页。而其中的边缘地带就是诸如博客或其他的作弊者能够创建链接的网站。这些网站的网页不能认为是可靠网页，尽管其本身的内容相当可靠。一个允许读者发表评论的知名报纸网站也是这样的一个例子。

《大数据：互联网大规模数据挖掘与分布式处理》由斯坦福大学的“Web挖掘”课程的内容总结而成，主要关注极大规模数据的挖掘。主要内容包括分布式文件系统、相似性搜索、搜索引擎技术、频繁项集挖掘、聚类算法、广告管理及推荐系统。其中相关章节有对应的习题，以巩固所讲解的内容。读者更可以从网上获取相关拓展材料。本文节选自《大数据：互联网大规模数据挖掘与分布式处理》。

为了实现TrustRank，我们必须要构建一个可靠网页组成的合适的随机跳转集合。前人尝试了以下两种做法。

(1) 人工检查一系列网页，判断哪些十分可靠。比如，我们可以选择具有最高PageRank的那些网页来考察。其依据的理论在于，尽管链接作弊可以将某个网页的PageRank从最低提高到中等，但是基本上不可能将某个垃圾网页的PageRank提到前几名。

(2) 选择一个成员受限的域名。其依据的假设是，作弊者很难将网页放到这些域名下。比如，我们可以选择.edu域名，而大学的网页不太可能会是垃圾农场。同样我们可以选择.mil、.gov等域名。但是，这种做法的问题是这些网页大都是来自美国的网站。为了使可靠网页的分布更好，我们应该包括来自外国的同类网站，如ac.il或edu.sg等。

当前的搜索引擎很可能在常规中使用上述中的第二种实现策略，因此我们想象的PageRank实际上是TrustRank的一种形式。

垃圾质量

垃圾质量的思想是度量来自垃圾的PageRank的比例。在计算普通的PageRank的同时，也计算基于某个可靠的随机跳转网页集合的TrustRank。假设页面p的PageRank是r，TrustRank是t。那么p的垃圾质量（spam mass）是(r-t)/r。如果该值为负或者一个很小的正数意味着p可能不是一个垃圾网页，而如果该值接近1则表明p可能是垃圾网页。这样一来，就可能在Web搜索引擎的索引当中去掉这些具有较高垃圾质量值的网页，从而可以在不用识别垃圾农场所使用的特定结构的情况下，剔除大部分作弊垃圾信息。■

如何利用奇异值分解简化数据

作者 / Peter Harrington

拥有电气工程学士和硕士学位，他曾经在美国加州和中国的英特尔公司工作 7 年。Peter 拥有 5 项美国专利，在三种学术期刊上发表过文章。他现在是 Zillabyte 公司的首席科学家，在加入该公司之前，他曾担任 2 年的机器学习软件顾问。Peter 在业余时间还参加编程竞赛和建造 3D 打印机。

餐馆可划分为很多类别，比如美式、中式、日式、牛排馆、素食店，等等。你是否想过这些类别够用吗？或许人们喜欢这些的混合类别，或者类似中式素食店那样的子类别。如何才能知道到底有多少类餐馆呢？我们也许可以问问专家？但是倘若某个专家说应该按照调料分类，而另一个专家则认为应该按照配料分类，那该怎么办呢？忘了专家，我们还是从数据着手吧。我们可以对记录用户关于餐馆观点的数据进行处理，并且从中提取出其背后的因素。

这些因素可能会与餐馆的类别、烹饪时所用的某个特定配料，或其他任意对象一致。然后，我们就可以利用这些因素来估计人们对没有去过的餐馆的看法。

提取这些信息的方法称为**奇异值分解**（Singular Value Decomposition，SVD）。从生物信息学到金融学等在内的很多应用中，SVD 都是提取信息的强大工具。

我们将介绍 SVD 的概念及其能够进行数据约简的原因。然后，我们将会介绍基于 Python 的 SVD 实现以及将数据映射到低维空间的过程。再接下来，我们就将学习推荐引擎的概念和它们的实际运行过程。为了提高 SVD 的精度，我们将会把其应用到推荐系统中去，该推荐系统将会帮助人们寻找到合适的餐馆。最后，我们讲述一个 SVD 在图像压缩中的应用例子。

SVD的应用

奇异值分解

优点：简化数据，去除噪声，提高算法的结果。

缺点：数据的转换可能难以理解。

适用数据类型：数值型数据。

利用SVD实现，我们能够用小得多的数据集来表示原始数据集。这样做，实际上是去除了噪声和冗余信息。当我们试图节省空间时，去除噪声和冗余信息就是很崇高的目标了，但是在这里我们则是从数据中抽取信息。基于这个视角，我们就可以把SVD看成是从有噪声的数据中抽取相关特征。如果这一点听来奇怪，也不必担心，我们后面会给出若干SVD应用的场景和方法，解释它的威力。

首先，我们会介绍SVD是如何通过隐性语义索引应用于搜索和信息检索领域的。然后，我们再介绍SVD在推荐系统中的应用。

隐性语义索引

SVD的历史已经超过上百个年头，但是最近几十年随着计算机的使用，我们发现了其更多的使用价值。最早的SVD应用之一就是信息检索。我们称利用SVD的方法为**隐性语义索引**（Latent Semantic Indexing，LSI）或**隐性语义分析**（Latent Semantic Analysis，LSA）。

在LSI中，一个矩阵是由文档和词语组成的。当我们在该矩阵上应用SVD时，就会构建出多个奇异值。这些奇异值代表了文档中的概念或主题，这一特点可以用于更高效的文档搜索。在词语拼写错误时，只基于词语

存在与否的简单搜索方法会遇到问题。简单搜索的另一个问题就是同义词的使用。这就是说，当我们查找一个词时，其同义词所在的文档可能并不会匹配上。如果我们从上千篇相似的文档中抽取出概念，那么同义词就会映射为同一概念。

推荐系统

SVD 的另一个应用就是推荐系统。简单版本的推荐系统能够计算项或者人之间的相似度。更先进的方法则先利用 SVD 从数据中构建一个主题空间，然后再在该空间下计算其相似度。考虑图 14-1 中给出的矩阵，它是由餐馆的菜和品菜师对这些菜的意见构成的。品菜师可以采用 1 到 5 之间的任意一个整数来对菜评级。如果品菜师没有尝过某道菜，则评级为 0。

	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	手撕猪肉
Ed	0	0	0	2	2
Peter	0	0	0	3	3
Tracy	0	0	0	1	1
Fan	1	1	1	0	0
Ming	2	2	2	0	0
Pachi	5	5	5	0	0
Jocelyn	1	1	1	0	0

	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	手撕猪肉
Ed	0	0	0	2	2
Peter	0	0	0	3	3
Tracy	0	0	0	1	1
Fan	1	1	1	0	0
Ming	2	2	2	0	0
Pachi	5	5	5	0	0
Jocelyn	1	1	1	0	0

图 1　餐馆的菜及其评级的数据。对此矩阵进行 SVD 处理则可以将数据压缩到若干概念中去。在右边的矩阵当中，标出了一个概念。

我们对上述矩阵进行 SVD 处理，会得到两个奇异值（读者如果不信可以自己试试）。因此，就会仿佛有两个概念或主题与此数据集相关联。我们

看看能否通过观察图中的0来找到这个矩阵的具体概念。观察一下右图的阴影部分，看起来Ed、Peter和Tracy对“烤牛肉”和“手撕猪肉”进行了评级，同时这三人未对其他菜评级。烤牛肉和手撕猪肉都是美式烧烤餐馆才有的菜，其他菜则在日式餐馆才有。

我们可以把奇异值想象成一个新空间。与图14-1中的矩阵给出的五维或者七维不同，我们最终的矩阵只有二维。那么这二维分别是什么呢？它们能告诉我们数据的什么信息？这二维分别对应图中给出的两个组，右图中已经标示出了其中的一个组。我们可以基于每个组的共同特征来命名这二维，比如我们得到的美式BBQ和日式食品这二维。

如何才能将原始数据变换到上述新空间中呢？下面我们将会进一步详细地介绍SVD，届时将会了解到SVD是如何得到U和VT两个矩阵的。VT矩阵会将用户映射到BBQ/日式食品空间去。类似地，U矩阵会将餐馆的菜映射到BBQ/日式食品空间去。真实的数据通常不会像图1中的矩阵那样稠密或整齐，这里如此只是为了便于说明问题。

推荐引擎中可能会有噪声数据，比如某个人对某些菜的评级就可能存在噪声，并且推荐系统也可以将数据抽取为这些基本主题。基于这些主题，推荐系统就能取得比原始数据更好的推荐效果。在2006年末，电影公司Netflix曾经举办了一个奖金为100万美元的大赛，这笔奖金会颁给比当时最好系统还要好10%的推荐系统的参赛者。最后的获奖者就使用了SVD。

下面将介绍SVD的一些背景材料，接着给出利用Python的NumPy实现SVD的过程。然后，我们将进一步深入讨论推荐引擎。当对推荐引擎有相当的了解之后，我们就会利用SVD构建一个推荐系统。

SVD是矩阵分解的一种类型，而矩阵分解是将数据矩阵分解为多个独立部分的过程。接下来我们首先介绍矩阵分解。

矩阵分解

在很多情况下，数据中的一小段携带了数据集中的大部分信息，其他信息则要么是噪声，要么就是毫不相关的信息。在线性代数中还有很多矩阵分解技术。矩阵分解可以将原始矩阵表示成新的易于处理的形式，这种新形式是两个或多个矩阵的乘积。我们可以将这种分解过程想象成代数中的因子分解。如何将12分解成两个数的乘积？(1,12)、(2,6)和(3,4)都是合理的答案。

不同的矩阵分解技术具有不同的性质，其中有些更适合于某个应用，有些则更适合于其他应用。最常见的一种矩阵分解技术就是SVD。SVD将原始的数据集矩阵Data分解成三个矩阵U、Σ和V^T。如果原始矩阵Data是m行n列，那么U、Σ和V^T就分别是m行m列、m行n列和n行n列。为了清晰起见，上述过程可以写成如下一行（下标为矩阵维数）：

$$Data_{m\times n} = U_{m\times m} \sum_{m\times n} V^T_{n\times n}$$

上述分解中会构建出一个矩阵Σ，该矩阵只有对角元素，其他元素均为0。另一个惯例就是，Σ的对角元素是从大到小排列的。这些对角元素称为奇异值（Singular Value），它们对应了原始数据集矩阵Data的奇异值。使用PCA，我们会得到矩阵的特征值，它们告诉我们数据集中的重要特征。Σ中的奇异值也是如此。奇异值和特征值是有关系的。这里的奇异值就是矩阵Data * DataT特征值的平方根。

前面提到过，矩阵 Σ 只有从大到小排列的对角元素。在科学和工程中，一直存在这样一个普遍事实：在某个奇异值的数目（r个）之后，其他的奇异值都置为0。这就意味着数据集中仅有r个重要特征，而其余特征则都是噪声或冗余特征。接下来，我们将看到一个可靠的案例。

我们不必担心该如何进行矩阵分解。在NumPy线性代数库中有一个实现SVD的方法。如果读者对SVD的编程实现感兴趣的话，请阅读*Numerical Linear Algebra*。

利用Python实现SVD

如果SVD确实那么好，那么该如何实现它呢？SVD实现了相关的线性代数，但这并不在我们的讨论范围之内。其实，有很多软件包可以实现SVD。NumPy有一个称为linalg的线性代数工具箱。接下来，我们了解一下如何利用该工具箱实现如下矩阵的SVD处理：

$$\begin{bmatrix} 1 & 1 \\ 7 & 7 \end{bmatrix}$$

要在Python上实现该矩阵的SVD处理，请键入如下命令：

```
>>> from numpy import *
>>> U,Sigma,VT=linalg.svd([[1, 1],[7, 7]])
```

接下来就可以在如下多个矩阵上进行尝试：

```
>>> U
array([[-0.14142136, -0.98994949],
[-0.98994949, 0.14142136]])
>>> Sigma
array([ 10., 0.])
>>> VT
array([[-0.70710678, -0.70710678],
[-0.70710678, 0.70710678]])
```

我们注意到，矩阵Sigma以行向量array([10., 0.])返回，而非如下矩阵：

```
array([[ 10., 0.],
       [ 0., 0.]]).
```

由于矩阵除了对角元素其他均为0，因此这种仅返回对角元素的方式能够节省空间，这就是由NumPy的内部机制产生的。我们所要记住的是，一旦看到Sigma就要知道它是一个矩阵。好了，接下来我们将在一个更大的数据集上进行更多的分解。

建立一个新文件svdRec.py并加入如下代码:

```
def loadExData():
return[[1, 1, 1, 0, 0],
       [2, 2, 2, 0, 0],
       [1, 1, 1, 0, 0],
       [5, 5, 5, 0, 0],
       [1, 1, 0, 2, 2],
       [0, 0, 0, 3, 3],
       [0, 0, 0, 1, 1]]
```

接下来我们对该矩阵进行SVD分解。在保存好文件svdRec.py之后，我们在Python提示符下输入:

```
>>> import svdRec
>>> Data=svdRec.loadExData()
>>> U,Sigma,VT=linalg.svd(Data)
>>> Sigma
array([  9.72140007e+00,  5.29397912e+00,
6.84226362e-01,7.16251492e-16, 4.85169600e-32])
```

前3个数值比其他的值大了很多（如果你的最后两个值的结果与这里的结果稍有不同，也不必担心。它们太小了，所以在不同机器上产生的结果就可能会稍有不同，但是数量级应该和这里的结果差不多）。于是，我们就可以将最后两个值去掉了。

接下来，我们的原始数据集就可以用如下结果来近似:

$$Data_{m\times n} \approx U_{m\times 3} \sum_{3\times 3} V^{T}_{3\times n}$$

图2就是上述近似计算的一个示意图。

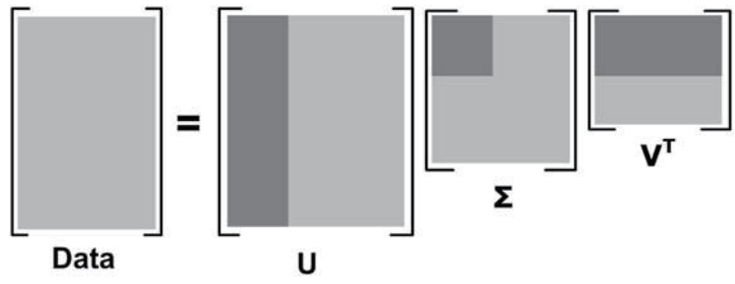


图2　SVD的示意图。矩阵Data被分解。浅灰色区域是原始数据，深灰色区域是矩阵近似计算仅需要的数据

我们试图重构原始矩阵。首先构建一个3×3的矩阵Sig3：

```
>>> Sig3=mat([[Sigma[0], 0, 0],[0, Sigma[1], 0], [0, 0,
Sigma[2]]])
```

接下来我们重构原始矩阵的近似矩阵。由于Sig3仅为3x3矩阵，我们只需使用矩阵U的前3列和VT的前3行。在Python中实现这一点，输入命令：

```
>>> U[:,:3]*Sig3*VT[:3,:]
array([[ 1.,   1.,   1.,   0.,   0.  ],
       [ 2.,   2.,   2.,   -0.,  -0. ],
       [ 1.,   1.,   1.,   -0.,  -0. ],
       [ 5.,   5.,   5.,   0.,    0. ],
       [ 1.,   1.,   -0.,   2.,   2. ],
       [ 0.,   0.,   -0.,   3.,   3. ],
            [ 0.,   0.,   -0.,   1.,   1.  ]])
```

我们是如何知道仅需保留前3个奇异值的呢？确定要保留的奇异值的数目有很多启发式的策略，其中一个典型的做法就是保留矩阵中90%的能量信息。为了计算总能量信息，我们将所有的奇异值求其平方和。于是可以将奇异值的平方和累加到总值的90%为止。另一个启发式策略就是，当矩阵上有上万的奇异值时，那么就保留前面的2000或3000个。尽管

后一种方法不太优雅，但是在实际中更容易实施。之所以说它不够优雅，就是因为在任何数据集上都不能保证前3000个奇异值就能够包含90%的能量信息。但在通常情况下，使用者往往都对数据有足够的了解，从而就能够做出类似的假设了。

现在我们已经通过三个矩阵对原始矩阵进行了近似。我们可以用一个小很多的矩阵来表示一个大矩阵。有很多应用可以通过SVD来提升性能。下面我们将讨论一个比较流行的SVD应用的例子——推荐引擎。

基于协同过滤的推荐引擎

近十年来，推荐引擎对因特网用户而言已经不是什么新鲜事物了。Amazon会根据顾客的购买历史向他们推荐物品，Netflix会向其用户推荐电影，新闻网站会对用户推荐新闻报道，这样的例子还有很多很多。当然，有很多方法可以实现推荐功能，这里我们只使用一种称为**协同过滤**（collaborative filtering）的方法。协同过滤是通过将用户和其他用户的数据进行对比来实现推荐的。

这里的数据是从概念上组织成了类似图2所给出的矩阵形式。当数据采用这种方式进行组织时，我们就可以比较用户或物品之间的相似度了。这两种做法都会使用我们很快就介绍到的相似度的概念。当知道了两个用户或两个物品之间的相似度，我们就可以利用已有的数据来预测未知的用户喜好。例如，我们试图对某个用户喜欢的电影进行预测，推荐引擎会发现有一部电影该用户还没看过。然后，它就会计算该电影和用户看过的电影之间的相似度，如果其相似度很高，推荐算法就会认为用户喜欢这部电影。

在上述场景下，唯一所需要的数学方法就是相似度的计算，这并不是很难。接下来，我们首先讨论物品之间的相似度计算，然后讨论在基于物品和基于用户的相似度计算之间的折中。最后，我们介绍推荐引擎成功的度量方法。

相似度计算

我们希望拥有一些物品之间相似度的定量方法。那么如何找出这些方法呢？倘若我们面对的是食品销售网站，该如何处理？或许可以根据食品的配料、热量、某个烹调类型的定义或者其他类似的信息进行相似度的计算。现在，假设该网站想把业务拓展到餐具行业，那么会用热量来描述一个叉子吗？问题的关键就在于用于描述食品的属性和描述餐具的属性有所不同。倘若我们使用另外一种比较物品的方法会怎样呢？我们不利用专家所给出的重要属性来描述物品从而计算它们之间的相似度，而是利用用户对它们的意见来计算相似度。这就是协同过滤中所使用的方法。它并不关心物品的描述属性，而是严格地按照许多用户的观点来计算相似度。图3给出了由一些用户及其对前面给出的部分菜肴的评级信息所组成的矩阵。

	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	手撕猪肉
Jim	2	0	0	4	4
John	5	5	5	3	3
Sally	2	4	2	1	2

图3　用于展示相似度计算的简单矩阵

我们计算一下手撕猪肉和烤牛肉之间的相似度。一开始我们使用欧氏距离来计算。手撕猪肉和烤牛肉的欧氏距离为：

$$\sqrt{(4-4)^2+(3-3)^2+(2-1)^2}=1$$

而手撕猪肉和鳗鱼饭的欧氏距离为：

$$\sqrt{(4-2)^2+(3-5)^2+(2-2)^2}=2.83$$

在该数据中，由于手撕猪肉和烤牛肉的距离小于手撕猪肉和鳗鱼饭的距离，因此手撕猪肉与烤牛肉比与鳗鱼饭更为相似。我们希望，相似度值在0到1之间变化，并且物品对越相似，它们的相似度值也就越大。我们可以用相似度=1/(1+距离)这样的算式来计算相似度。当距离为0时，相似度为1.0。如果距离真的非常大时，相似度也就趋近于0。

第二种计算距离的方法是**皮尔逊相关系数**（Pearson correlation）。它度量的是两个向量之间的相似度。该方法相对于欧氏距离的一个优势在于，它对用户评级的量级并不敏感。比如某个狂躁者对所有物品的评分都是5分，而另一个忧郁者对所有物品的评分都是1分，皮尔逊相关系数会认为这两个向量是相等的。在NumPy中，皮尔逊相关系数的计算是由函数`corrcoef()`进行的，后面我们很快就会用到它了。皮尔逊相关系数的取值范围从-1到+1，我们通过`0.5 + 0.5*corrcoef()`这个函数计算，并且把其取值范围归一化到0到1之间。

另一个常用的距离计算方法就是**余弦相似度**（cosine similarity），其计算的是两个向量夹角的余弦值。如果夹角为90度，则相似度为0；如果两个向量的方向相同，则相似度为1.0。同皮尔逊相关系数一样，余弦相似度的取值范围也在-1到+1之间，因此我们也将它归一化到0到1之间。计算余弦相似度值，我们采用的两个向量A和B夹角的余弦相似度的定义如下：

$$\cos\theta = \frac{A \cdot B}{||A||\,||B||}$$

其中，||A||、||B||表示向量A、B的2范数，你可以定义向量的任一范数，但是如果不指定范数阶数，则都假设为2范数。向量[4,2,2]的2范数为：

$$\sqrt{4^2 + 3^2 + 2^2}$$

同样，NumPy的线性代数工具箱中提供了范数的计算方法`linalg.norm()`。

接下来我们将上述各种相似度的计算方法写成Python中的函数。打开svdRec.py文件并加入下列代码。

程序清单1　相似度计算

```
from numpy import *
from numpy import linalg as la

def ecludSim(inA,inB):
    return 1.0/(1.0 + la.norm(inA - inB))

def pearsSim(inA,inB):
    if len(inA) < 3 : return 1.0
    return 0.5+0.5*corrcoef(inA, inB, rowvar = 0)[0][1]

def cosSim(inA,inB):
    num = float(inA.T*inB)
    denom = la.norm(inA)*la.norm(inB)
    return 0.5+0.5*(num/denom)
```

程序中的3个函数就是上面提到的几种相似度的计算方法。为了便于理解，NumPy的线性代数工具箱linalg被作为la导入，函数中假定inA和inB都是列向量。perasSim()函数会检查是否存在3个或更多的点。如果不存在，该函数返回1.0，这是因为此时两个向量完全相关。

下面我们对上述函数进行尝试。在保存好文件svdRec.py之后，在Python提示符下输入如下命令：

```
>>> reload(svdRec)
<module 'svdRec' from 'svdRec.pyc'>
>>> myMat=mat(svdRec.loadExData())
>>> svdRec.ecludSim(myMat[:,0],myMat[:,4])
0.12973190755680383
>>> svdRec.ecludSim(myMat[:,0],myMat[:,0])
1.0
```

欧氏距离看上去还行，那么接下来试试余弦相似度：

```
>>> svdRec.cosSim(myMat[:,0],myMat[:,4])
0.5
>>> svdRec.cosSim(myMat[:,0],myMat[:,0])
1.0000000000000002
```

余弦相似度似乎也行，就再试试皮尔逊相关系数：

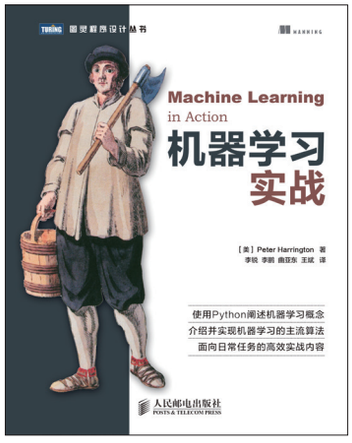
```
>>> svdRec.pearsSim(myMat[:,0],myMat[:,4])
0.20596538173840329>>> svdRec.pearsSim(myMat[:,0],myMat[:,0])
1.0
```

上面的相似度计算都是假设数据采用了列向量方式进行表示。如果利用上述函数来计算两个行向量的相似度就会遇到问题（我们很容易对上述函数进行修改以计算行向量之间的相似度）。这里采用列向量的表示方法，暗示着我们将利用基于物品的相似度计算方法。后面我们会阐述其中的原因。

基于物品的相似度还是基于用户的相似度？

我们计算了两个餐馆菜肴之间的距离，这称为**基于物品**（item-based）的相似度。另一种计算用户距离的方法则称为**基于用户**（user-based）的相似度。回到图3，行与行之间比较的是基于用户的相似度，列与列之间比较的则是基于物品的相似度。到底使用哪一种相似度呢？这取决于用户或物品的数目。基于物品相似度计算的时间会随物品数量的增加而增加，基于用户的相似度计算的时间则会随用户数量的增加而增加。如果我们有一个商店，那么最多会有几千件商品。在撰写本文之际，最大的商店大概有100000件商品。而在Netflix大赛中，则会有480000个用户和17700部电影。如果用户的数目很多，那么我们可能倾向于使用基于物品相似度的计算方法。

对于大部分产品导向的推荐引擎而言，用户的数量往往大于物品的数量，即购买商品的用户数会多于出售的商品种类。



《机器学习实战》通过精心编排的实例，切入日常工作任务，摒弃学术化语言，利用高效的可复用Python代码来阐释如何处理统计数据，进行数据分析及可视化。通过各种实例，读者可从中学会机器学习的核心算法，并能将其运用于一些策略性任务中，如分类、预测、推荐。另外，还可用它们来实现一些更高级的功能，如汇总和简化等。本文节选自《机器学习实战》。

推荐引擎的评价

如何对推荐引擎进行评价呢？此时，我们既没有预测的目标值，也没有用户来调查他们对预测的满意程度。这里我们就可以采用前面多次使用的交叉测试的方法。具体的做法就是，我们将某些已知的评分值去掉，然后对它们进行预测，最后计算预测值和真实值之间的差异。

通常用于推荐引擎评价的指标是称为**最小均方根误差**（Root Mean Squared Error，RMSE）的指标，它首先计算均方误差的平均值然后取其平方根。如果评级在1星到5星这个范围内，而我们得到的RMSE为1.0，那么就意味着我们的预测值和用户给出的真实评价相差了一个星级。

小结

SVD是一种强大的降维工具，我们可以利用SVD来逼近矩阵并从中提取重要特征。通过保留矩阵80%～90%的能量，就可以得到重要的特征并去掉噪声。SVD已经运用到了多个应用中，其中一个成功的应用案例就是推荐引擎。

推荐引擎将物品推荐给用户，协同过滤则是一种基于用户喜好或行为数据的推荐的实现方法。协同过滤的核心是相似度计算方法，有很多相似度计算方法都可以用于计算物品或用户之间的相似度。通过在低维空间下计算相似度，SVD提高了推荐系引擎的效果。

在大规模数据集上，SVD的计算和推荐可能是一个很困难的工程问题。通过离线方式来进行SVD分解和相似度计算，是一种减少冗余计算和推荐所需时间的办法。■

专访：《机器学习实战》作者Peter Harrington

如何成为一位数据科学家

Peter Harrington，拥有电气工程学士和硕士学位，他曾经在美国加州和中国的英特尔公司工作7年。Peter拥有5项美国专利，在三种学术期刊上发表过文章。他现任HG Data首席科学家。如果说LinedIn跟踪的是人和人之间的商务往来，HG Data则是致力于挖掘公司间的商业往来。他曾是Zillabyte公司的创始人和首席科学家，在此之前，他曾担任2年的机器学习软件顾问。Peter在业余时间还参加编程竞赛和建造3D打印机。

问：机器学习似乎比其他计算机科学学科都要难，特别是对于数学不太好的程序员而言。你对这些程序员有什么样的建议呢？

我建议应该先自学基本的概率、统计，以及线性代数。你不需要学一个学期那么长的课，这些基础知识就会让你有很大收获。有很多在线资源，比如Kahn academy视频。（我在56.com和Kahn academy找了一下有很多英文的，也有一些中文的。）也有一些比较容易起步的书，我比较熟悉有美国英文版的“teach yourself”（自学）系列，“statistics for dummies”（傻瓜统计），“probability refresher”（概率补习），“statistics demystified”（统计解惑）等等。

我其实认为这里面其实很有商机。Kahn academy视频很不错，因为它们都很短，但遗憾的是这些视频都是英文的。我看见的中文线性代数视频都很长。如果你能做出像Kahn academy那样的中文视频，我觉得是会非常受欢迎的。

问：如何进阶学习机器学习？对于初学者是否有一个类似于路线图的东西？你有什么推荐书单吗？

我会读Witten和Frank所著的《数据挖掘：实用机器学习工具与技术》，这里面涉及的数学很少，但是又对普通算法做了很好的介绍。我觉得紧接着就该读Tan, Steinbach，以及Kumar的《数据挖掘导论》。

当然，这些书都很厚，如果你想马上就搞明白一些东西，估计就不想读这些大部头了。如果要把某个算法弄明白，我会在网上找很多教程。比如Adboost算法，我认为多读一些不同的教程比只读一个，深入钻研要好很多。

最后我觉得应该多动手玩玩实例。问问你自己：如果我改变这个数据，结果会是怎么样的呢？

问：在真实案例中，数据预处理可能要比算法还要重要，你要不要考虑在新版《机器学习实战》加入数据预处理技巧和实例？

我完全同意，我的大部分时间都是用来做数据预处理。我会在未来加入数据预处理的内容。我不知道这里面会不会有什么神奇的捷径，有时候我面对的就是一堆苦活儿。我还要说：你一定要把能自动化的都自动化，这样就会节省很多你未来的工作量。

问：对于有些人来说“算法”才是机器学习真正有趣的地方，但是机器学习里面总有一些苦活累活不那么有意思，比如数据预处理。你是怎么完成这些“不那么有趣”的工作的？

当然，肯定有无聊的工作，所以你一定要把这部分工作自动化，这样你就不需要重复做这些无聊的工作了。这样做也会让你变成一个更好的软件开发者。

问：能向我们介绍一些机器学习方面的开源项目吗？

我现在能想到最好的就是Scikit-learn (http://scikit-learn.org/stable/)了。这是用Python写的项目，用到了Scipy和Numpy。

问：数据科学家被评为世界上最火的工作之一，你认同吗？您本人作为一个数据科学家，有什么可以和我们分享的经验吗？要成为一个数据科学家需要有什么条件？

我认为数据科学家现在确实很好找工作。什么是数据科学家呢？我认为数据科学家是介于统计学家和软件工程师中间的一种工作。公司、个人、NPO，甚至运动队都需要根据数据来做决策。他们需要可以分析数据的人。这需要我之前提过的两种条件。人们不需要单纯的统计学家，这些人可能对于争论自己到底用不用贝叶斯定理更感兴趣，人们需要的是真正能做**实事**的人。

所以我也建议大家多动手**做**一些东西。这是什么意思呢？创造一些项目，收集数据，预处理数据，然后做一些数据分析，展示数据，最后向公众展示这些数据。如果你做了很多这样的事情，那么你就有一个可以用来向你未来老板或者其他人展示的档案夹。几乎我书里的每个例子都可以用来做成一个网站或者智能电话app，这些都是你可以示人的资本。

问：人工智能的发展到了瓶颈期，而机器学习似乎是可以打破这个僵局的领域。你认为是什么原因造成了机器学习这样的发展步伐？

相比于物理学或者电气工程这样的学科，人工智能可能是很年轻的。一个年轻的学科中的很多课题和原则都是被不断发现和精炼的。很多时候，研究项目被当做事实一样摆出来，我认为这就是“人工智能承诺得太多，实现得太少”的真正原因。

我觉得这里面一个很好的例子就是很多学者想要用神经网络再造哺乳动物大脑。这让我想起来早些时候人们试图通过造出外形很像鸟翅膀的翼来制造飞机，其结果只能是飞起来把自己的骨头砸碎了。我不是要批判任何在做神经网络方面工作的人：这就是个试验，有一些有用的应用，但是这些解决不了我们的问题也没法造出有感知的机器。问题是这些试验被当做了事实放在教科书里、电影里，以及新闻里，但它们还仅仅是试验。

回到那个飞机的例子。当人类第一次知道动力飞行时，他们是因为要解决一个小任务而做出来的，而不是要建造什么机器鸟。我觉得同样的方法也促成了人工智能上的一些成功。2010-2011年的大突破：IBM的Watson计算机、Google的自动驾驶汽车，以及iPhone的Siri语音识别，甚至还有一个公司成功地用人工智能写出了新闻报道。这些都不是试验，这些都是生产线上的商品，被无数的人所使用。人工智能纯化论者会认为这些只是被用来完成明确任务的工具，而不是智能机器。

回到我们的问题，我认为机器学习是很实用的工具，可以用来解决很具体的问题，但是人工智能是一个高高在上的目标，很难达到。这也就是人工智总让人感到失望，而机器学习总会为我们带来惊喜的原因。

问：很多大（数据）公司，比如Google, Facebook和Baidu都投入很多精力在深度学习上。你认为深度学习会在未来取代“人工特性+机器学习”的方法吗？

不，我不认为深度学习会取代人工特性+机器学习。有很多领域，深度学习确实很擅长，比如识别图片。但是仍然有很多领域现存算法的表现更胜一筹。

问：在深度学习之后，机器学习的下一个热点是什么？

我不知道，也许你可以基于学术或者技术会议的论文提交来创造一个预测模型来告诉我下一个与研究热点。

问：很多人认为语言会是大数据和机器学习的未来主要功用。让我们举一个具体的例子，如果要预测一个公司的收入，你会用什么模型？

这点说得很对。我知道大的零售商会有一整个团队来做销售的预测。如果他们真能准确预测销量，那他们就会省下一大笔钱。如果要预测一家公司的收入，我会首先用回归+逻辑回归。逻辑回归让我们可以随时打开或关闭操作，这对于相关事情发生以及金钱入账这样的事来说都是一个很好的模型。

问：请问7.3节的著名的45问题到底是什么？

不好意思，我应该在书中说明地更清楚来着，这也来自于一个英文论坛上的问题。

45问题指的就是数据都在一条呈45°角的线上，或者以$y = x$的形式存在。这是关于如何为这类数据制造一个简单分类器的问题。

这为什么会是一个问题呢？如果我们有一个类：在$y = x$这条线上的1，我们还有第二个类：在$y = x + 6$这条线上的0。

那么现在在X轴（垂直轴）上选择一个值，这个值可以让所有属于1类的数值在其一边，而所有属于0类的数值在其另一边。再试着在Y轴（水平线）上找一个值。你无法找到一个简单的X&Y组合把点分成两类，这就是45问题。

一个支撑向量机，或者逻辑回归对于这样的数据不会有什么问题。你也可以用一个数据转换，和一个决策残根来轻松应对这个数据。

问：你打算想让《机器学习实战》变得更加有趣吗？比方说，可以在每一章中加入一个日常生活中的例子。

这听起来是个不错的主意。 ■

访谈英文版

专题：如何成为一位数据科学家

机器学习产品开发的漫漫长路

作者 / Aria Haghighi

Aria Haghighi 是 Prismatic 的联合创始人。他曾数次夺得“统计自然语言处理”学者奖，他也是前微软研究院的院士。

译者 / 若萱

优秀的机器学习产品来自有丰富经验和渊博知识的开发小组

机器学习（Machine learning），搭着“大数据（big data）”的顺风车，正成为风靡一时的话题。同其他被炒作的技术方向一样，宣传的热度远远超过实际实现的产品。可以说，自从上世纪90年代至千禧年间谷歌的革命性创新之后，再没有哪一项算法技术的创新能催生出一个可以如此深刻影响大众文化的产品了。并不是说那之后机器学习就没有任何成就，只是没有哪个能有如此深远的影响力，也没有哪个是真正算法级的改进。

Netflix也许在使用推荐技术，但是用与不用，Netflix还是Netflix，并不影响什么。然而，如果没有Page和Brin等人发明利用网络的图形结构和链接文字（anchor text）提高搜索效率的算法，就没有今天的谷歌。

那么，为什么会这样呢？

缺乏尝试？不是的。回想一下，有多少创业者志在将自然语言处理应用于大众却在产品真正推出后逐渐被遗忘？

构建优秀的机器学习产品的挑战性不仅仅在于对基础理论的理解，更需要的是对这个领域和课题有足够了解，这样才能根据想法设计出可操作

的模型。好的课题不会有现成的解决方案。机器学习的主要应用领域（例如自然语言处理NLP）的发展主要源于对这些课题的独到见解而不是泛泛的机器学习理论。一般来说，对一个课题的独到见解加上细致的模型设计就决定了一个产品／系统的成败。

本文的目的并不是为了吓退开发者，对开发以机器学习为核心的美妙产品敬而远之，而是要明确其困难所在。

机器学习的发展

在过去的十年了，机器学习经历了一个漫长的发展过程。

在我开始读研之前，对诸如支持向量机（SVM）一类的大型边缘分类器（large-margin classifier）的训练都还是利用John Platt的序列最小优化算法（SMO algorithm）来完成。那时，训练时间与训练数据量几乎不成比例，编写算法本身不但要具有二次规划（quadratic programming）的相关背景还要添加各种启发训练以选择主动约束以及令人费解的参数调整。

现在，（相对）简单的同步算法（simple online algorithm [PDF]）可以在线性时间内完成对性能相当的大型边缘分类器的训练。（基于概率的）图形模型也取得了类似的进展：马氏蒙特卡洛方法（Markov-chain Monte Carlo，MCMC）和变分法（variational methods）极大地便利了对任意复杂度的图形模型的推断[1]。

[1] 尽管MCMC并不是什么新的统计技术，但直到近期，它才得以广泛应用于大规模机器学习的应用中。

说句题外话，如果你看过这8年来计算机语言学协会（ACL）期刊上发表的论文，你会发现2011年的顶级论文比2003年的技术难度提升了无数倍。

在教育前沿，我们同样经历了很大的变更。

21 世纪 00 年代，当时我还是斯坦福的本科生，我选修了 Angrew Ng 的机器学习课程和 Daphne Koller 的概率图模型课程，那是我在斯坦福选修过的最好的两门课，尤其是 Koller 的，我从中还学到了很多教学的方法。在那个年代，那两门课每年只能容纳 100 名左右的学生，而现在，任何人都可以从网上课堂选修这些课程。

作为机器学习（尤其是自然语言处理方面）的应用者，这些发展极大地便利了科研的各个方面。然而，最核心的决策并不是我该选择哪个抽象机器学习算法，哪个损失函数，或者哪个学习目标，而是哪些功能哪些结构可以解决我的问题。而这一技能只能来源于实践的积累。因此，越来越多的人了解认识机器学习这一领域固然很好，但这并不能解决开发构建智能系统的关键难点。

好的课题没有现成的解决方案

一个让你感兴趣并且真的想动手去做的课题的建模过程要远远复杂于抽象化那些经典的机器学习命题。

举个例子：机器翻译（machine translation，MT）。

乍一看去，机器翻译类似于统计的分类问题（statistical classification problem）——根据输入的非英语的短句预测出其对应的英文短句。问题是，英语的各种可能组合太多了，很难简单的把机器翻译看待为黑盒分类问题（black-box classification problem）。跟其他机器学习的应用类似，机器翻译有大量的结构，而优秀的研究者的部分工作就是分解问题，把复杂问题分解成可定性学习和编码的小问题。我认为，对于这一类的复杂命题，其下一步的发展主要在于如何分解和结构化这些问题的解空间，而不是所需的机器学习技术。

在过去的十年里，机器翻译取得了飞跃性进步。我觉得这些进步很大程

度上（当然，不完全）取决于针对这一课题的深刻理解，而不是机器学习领域的普遍发展。

当代的统计机器翻译起源于一篇很牛的论文——《统计机器翻译中的数学》[PDF]，这篇文章提出的容噪架构（noisy-channel architecture）将成为未来机器翻译系统的基础。

简单来说，这个模型的工作原理是这样的[2]：你可以把它理解为一个基于概率的词典。每个非英语的单词对应几个候选的英语短语，包括那些没有英语对应词汇的无意义空字，这些候选短语重新排序生成一个貌似合理的英语翻译。这里忽略了很多错综复杂的细节，比如如何有效的从各种排列组合中得出候选的英语翻译，比如选用哪个学习模型能系统地根据不同语言对短语进行重组，比如如何评定各候选英语翻译的合理性（见语言模型language model）。

❷ 模型是生成的，所以这里是从演绎的角度进行描述的。模型的生成过程其实是相反的。

机器翻译最重要的改进就是对这个模型的改进。

现在的机器翻译不再纠结于学习单个单词的翻译概率，转而研究将非英文短语翻译为英文短语的模型。例如，德语单词“abends”被意译为英文的介词短语“in the evening”。

在采用短语翻译（phrase-based translation [PDF]）前，逐字翻译的模型只能将其翻译为一个英文单词，很难达到正确的英文翻译[3]。而短语翻译通常能更准确的翻译出流利的惯用的英文释义。当然，增加短语释义在一定程度上增加了复杂度，包括对于没遇到过的短语片段如何估计短语结果——谁也不知道“in the evening”能对应其他语言中的哪个短语。最出乎意料的是，并不是机器学习领域的发展促成了这些改进，恰恰是这种针对特定问题的模型设计造就了这些进步。在机器翻译系统中的很多地方，人们可以并且已经使用了更精妙的机器学习技术，而这些也确实实现了一些改善，但远不及一个良好的针对特定问题的研究结果产生的影响。

❸ IBM模型3号引入了派生（fertility）的概念，允许从一个单词生成多个独立的目标单词。尽管这样有可能产生正确的翻译，但是这种可能性非常低。

短语翻译相关论文的原作者之一，Franz Och，后来受聘于谷歌并成为这个搜索公司翻译部门的核心人物。尽管谷歌的体系知识基础可回溯到Och还是信息科学院（Information Sciences Institute）的一名科研人员的时候（甚至更早，当他还是一名研究生的时候），除了短语翻译的发现（以及最小错误率训练，Och的另一创新），Och更多的贡献在于将这些想法应用于互联网而进行的大量软件工作，这些软件工作将重要的科研发现引入大规模语言模型以及自然语言处理的其他方面。值得一提的是，Och不仅是一位世界一流的科学家，更是一位杰出的黑客和开发者。正是他的这种难得的专业技能组合使得这些创意能够从实验室的一个项目演变为今天谷歌翻译。

问题的阐述

我觉得还有一个比精巧的模型设计和软件开发技术更大的障碍，那就是如何诠释我们要解决的问题。

以机器翻译和语音识别为例，我们能很直观且清晰地阐述我们在做的课题。然而，很多可能为下一个十年带来革命性产品的自然语言处理的新技术却都前途模糊。我们应如何，确切地说，能否将优秀的科研创新（如结构化的主题模型structured topic models，话语处理discourse processing，主观态度分析sentiment analysis等等）转化为一个大众化的产品呢？

比如自动化摘要（summarization）。

我们都希望在某种程度上可以对内容进行总结和结构化。然而，出于计算和科研的原因，你需要对问题的范围进行某些限定从而能够建模、设计算法并最终对其进行评估。

例如，在摘要法的著作中，多文档摘要 (multi-document summarization) 这一问题通常被定义为从文档中选取小部分语句并对其进行排序。这真的就是我们要解决的问题么？从一篇文字中提取少量整句就是最好的摘要法么？即使这样的摘要是精确的，这种生拼硬凑的句子结构会不会让用户仿佛面对一个科学怪人 (弗兰克斯坦) 呢？

再比如主观态度分析。

人们会满足于用一个粗略的“赞”或“踩”去评价一个东西或一件事么？他们会不会需要更丰富的情绪表达方式，需要针对一个东西的某些细节进行评价呢 (如，喜欢这里的食物，讨厌这里的装修风格，等等)？人们是更需要明确单个评论者的态度，还是更在意一份综合评价分析？

通常，产品设计者下达这些决策再交由科研人员和工程师去实现。这一过程的问题在于，以机器学习为核心的产品很大程度上受制于当时的技术可实现性和算法可实现性。根据我的个人经验，对机器学习及其相关领域的技术有一定了解的人更容易想出新颖的产品创意，而对于没有这些技术背景的人则几乎没有可能。举一个泛泛的类比，就像建筑一样，搭建一座桥梁需要各种材料资源和物理知识，没有相关背景的人怎么可能能设计出一座桥呢？

说了这么多，主要就是一句话：如果想开发一个优秀的机器学习产品，你需要一个优秀的产品＋设计＋科研＋工程师团队来解决各方面的细节问题：从机器学习理论到系统构建到专业领域知识到宏观产品思路到技术交流到图形设计等等。最好他们各自在某一领域都是世界一流专家并对其他几个领域也熟知一二。

拥有这些技术的小型智囊团可以更灵活的面对产品远景及其模型设计过程中的不确定性。相比之下，大型公司里研发人员和产品设计人员如果

不能紧密合作反而更难解决这类难题。因此，未来的机器学习的优秀产品将出自一些拥有这些资源的小型创始团队，也就是众所周知的“车库公司”。 ■

阅读原文：机器学习产品开发的漫漫长路

阅读英文原文：What it takes to build great machine learning products

R 语言可视化初阶

作者 / Robert I. Kabacoff

R 语言社区著名学习网站 Quick-R 的幕后维护者，现为全球化开发与咨询公司 Management 研究集团研发副总裁。此前，Kabacoff 博士是佛罗里达诺瓦东南大学的教授，讲授定量方法和统计编程的研究生课程。Kabacoff 还是临床心理学博士、统计顾问，擅长数据分析，在健康、金融服务、制造业、行为科学、政府和学术界有 20 余年的研究和统计咨询经验。

我曾经多次向客户展示以数字和文字表示的、精心整理的统计分析结果，得到的只是客户呆滞的眼神，尴尬得房间里只能听到鸟语虫鸣。然而，当我使用图形向相同的用户展示相同的信息时，他们往往会兴致盎然，甚至豁然开朗。还有很多次，我都是通过看图才得以发现了数据中的模式，或是检查出了数据中的异常值——这些模式和异常都是在我进行更为正式的统计分析时彻底遗漏的。

人类非常善于从视觉呈现中洞察关系。一幅精心绘制的图形能够帮助你在数以千计的零散信息中做出有意义的比较，提炼出使用其他方法时不那么容易发现的模式。这也是统计图形领域的进展能够对数据分析产生重大影响的原因之一。数据分析师需要观察他们的数据，而 R 在该领域表现出众。

我们将在本文中讨论处理图形的一般方法。我们首先探讨如何创建和保存图形，然后关注如何修改那些存在于所有图形中的特征，包括图形的标题、坐标轴、标签、颜色、线条、符号和文本标注。我们的焦点是那些可以应用于所有图形的通用方法。最后，我们将研究组合多幅图形为单幅图形的各种方法。

使用图形

R 是一个惊艳的图形构建平台。这里我特意使用了“构建”一词。在通常的交互式会话中，你可以通过逐条输入语句构建图形，逐渐完善图形特征，直至得到想要的效果。

考虑以下五行代码:

```
attach(mtcars)
plot(wt, mpg)
abline(lm(mpg~wt))
title("Regression of MPG on Weight")
detach(mtcars)
```

首句绑定了数据框mtcars。第二条语句打开了一个图形窗口并生成了一幅散点图，横轴表示车身重量，纵轴为每加仑汽油行驶的英里数。第三句向图形添加了一条最优拟合曲线。第四句添加了标题。最后一句为数据框解除了绑定。在R中，图形通常都是以这种交互式的风格绘制的（参见图1）。

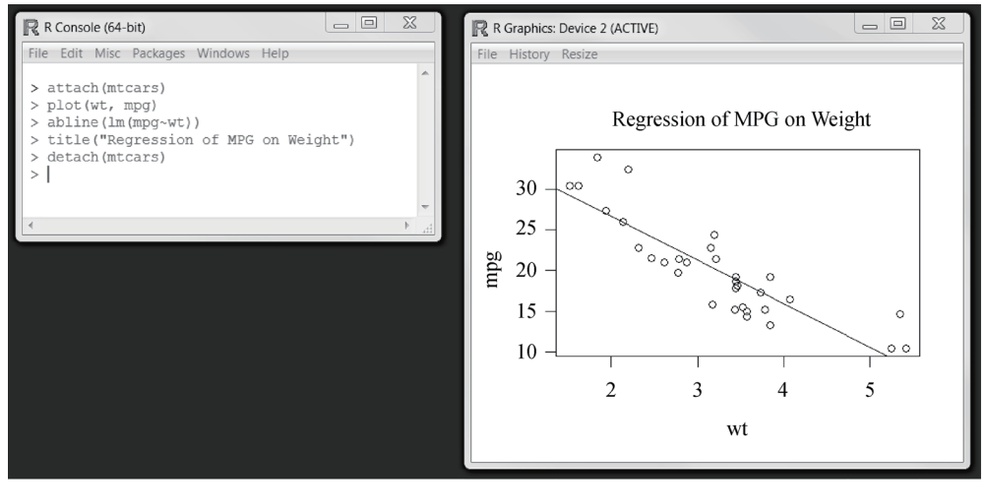


图1　创建图形

可以通过代码或图形用户界面来保存图形。要通过代码保存图形，将绘图语句夹在开启目标图形设备的语句和关闭目标图形设备的语句之间即可。例如，以下代码会将图形保存到当前工作目录中名为mygraph.pdf的PDF文件中:

```
pdf("mygraph.pdf")
 attach(mtcars)
 plot(wt, mpg)
 abline(lm(mpg~wt))
 title("Regression of MPG on Weight")
 detach(mtcars)
dev.off()
```

除了pdf()，还可以使用函数win.metafile()、png()、jpeg()、bmp()、tiff()、xfig()和postscript()将图形保存为其他格式。（注意，Windows图元文件格式仅在Windows系统中可用。）

通过图形用户界面保存图形的方法因系统而异。对于Windows，在图形窗口中选择“文件”→“另存为”，然后在弹出的对话框中选择想要的格式和保存位置即可。在Mac上，当Quartz图形窗口处于高亮状态时，点选菜单栏中的“文件”→“另存为”即可。其提供的输出格式仅有PDF。在UNIX系统中，图形必须使用代码来保存。在附录A中，我们将考虑每个系统中可用的备选图形用户界面，这将给予你更多选择。

通过执行如plot()、hist()（绘制直方图）或boxplot()这样的高级绘图命令来创建一幅新图形时，通常会覆盖掉先前的图形。如何才能创建多个图形并随时查看每一个呢？方法有若干。

第一种方法，你可以在创建一幅新图形之前打开一个新的图形窗口：

```
dev.new()
 statements to create graph 1
dev.new()
 statements to create a graph 2
etc.
```

每一幅新图形将出现在最近一次打开的窗口中。

第二种方法，你可以通过图形用户界面来查看多个图形。在Mac上，你可以使用Quartz菜单中的“后退”(Back)和“前进”(Forward)来逐个浏览图形。在Windows上，这个过程分为两步。在打开第一个图形窗口以后，勾选“历史”(History)→“记录”(Recording)。然后使用菜单中的“上一个”(Previous)和“下一个”(Next)来逐个查看已经绘制的图形。

第三种也是最后一种方法，你可以使用函数`dev.new()`、`dev.next()`、`dev.prev()`、`dev.set()`和`dev.off()`同时打开多个图形窗口，并选择将哪个输出发送到哪个窗口中。这种方法全平台适用。关于这种方法的更多细节，请参考help(dev.cur)。

R将在保证用户输入最小化的前提下创建尽可能美观的图形。不过你依然可以使用图形参数来指定字体、颜色、线条类型、坐标轴、参考线和标注。其灵活度足以让我们实现对图形的高度定制。

我们将以一个简单的图形作为开始，接着进一步探索按需修改和强化图形的方式。然后，我们将着眼于一些更复杂的示例，以阐明其他的图形定制方法。我们关注的焦点是那些可以应用于多种R图形的技术。

一个简单的例子

让我们从表1中给出的假想数据集开始。它描述了病人对两种药物五个剂量水平上的响应情况。

表1　病人对两种药物五个剂量水平上的响应情况

剂　　量	对药物A的响应	对药物B的响应
20	16	15
30	20	18
40	27	25
45	40	31
60	60	40

可以使用以下代码输入数据：

```
dose  <- c(20, 30, 40, 45, 60)
drugA <- c(16, 20, 27, 40, 60)
drugB <- c(15, 18, 25, 31, 40)
```

使用以下代码可以创建一幅描述药物A的剂量和响应关系的图形：

```
plot(dose, drugA, type="b")
```

plot()是R中为对象作图的一个泛型函数（它的输出将根据所绘制对象类型的不同而变化）。本例中，plot(x，y，type="b")将x置于横轴，将y置于纵轴，绘制点集(x, y)，然后使用线段将其连接。选项type="b"表示同时绘制点和线。使用help(plot)可以查看其他选项。结果如图2所示。

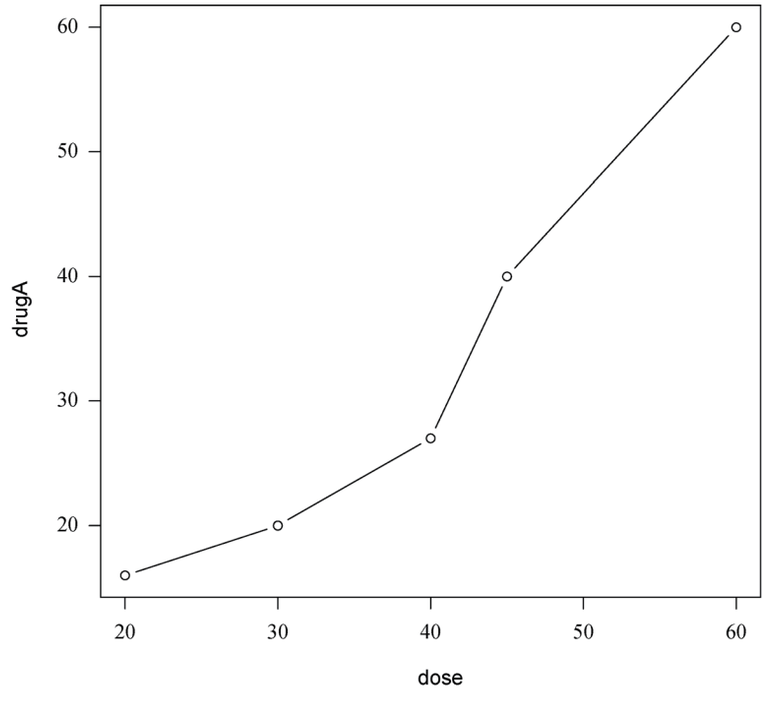


图2　药物A剂量和响应的折线图

现在我们来修改此图的外观。

图形参数

我们可以通过修改称为**图形参数**的选项来自定义一幅图形的多个特征（字体、颜色、坐标轴、标题）。

（修改图形参数的）一种方法是通过函数par()来指定这些选项。以这种方式设定的参数值除非被再次修改，否则将在会话结束前一直有效。其调用格式为par(*optionname=value*, *optionname=name*,…)。不加参数地执行par()将生成一个含有当前图形参数设置的列表。添加参数no.readonly=TRUE可以生成一个可以修改的当前图形参数列表。

继续我们的例子，假设你想使用实心三角而不是空心圆圈作为点的符号，并且想用虚线代替实线连接这些点。你可以使用以下代码完成修改：

```
opar <- par(no.readonly=TRUE)
par(lty=2, pch=17)
plot(dose, drugA, type="b")
par(opar)
```

结果如图3所示。

首个语句复制了一份当前的图形参数设置。第二句将默认的线条类型修改为虚线（lty=2）并将默认的点符号改为了实心三角（pch=17）。然后我们绘制了图形并还原了原始设置。

你可以随心所欲地多次使用par()函数，即par(lty=2, pch=17)也可以写成：

```
par(lty=2)
par(pch=17)
```

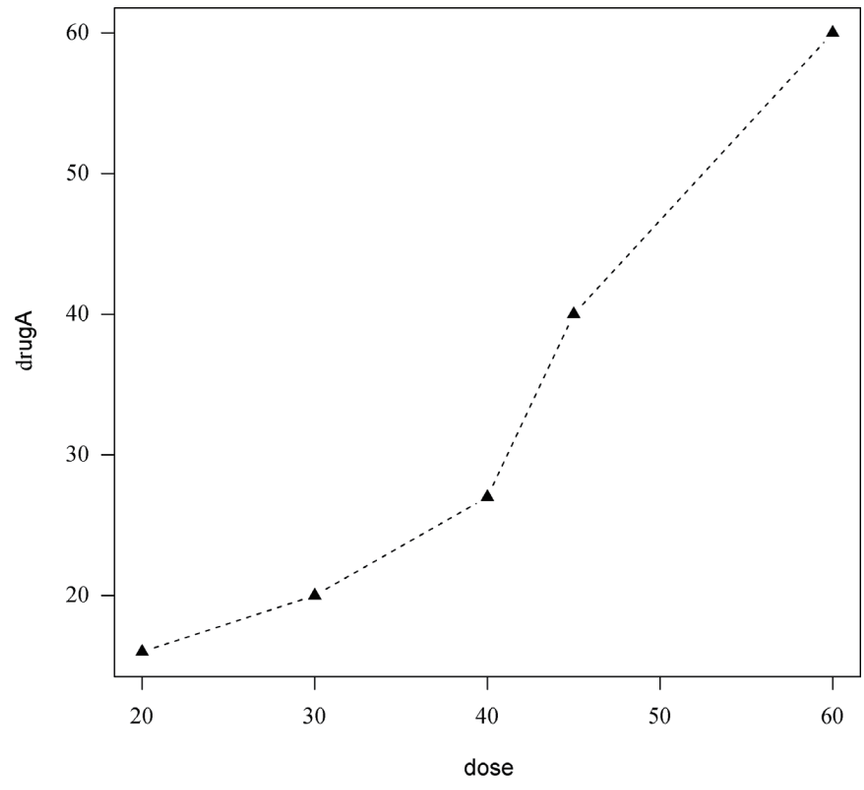


图3　药物A剂量和响应的折线图。修改了线条类型和点的符号

指定图形参数的第二种方法是为高级绘图函数直接提供optionname=value的键值对。这种情况下，指定的选项仅对这幅图形本身有效。你可以通过代码：

```
plot(dose, drugA, type="b", lty=2, pch=17)
```

来生成与上图相同的图形。

并不是所有的高级绘图函数都允许指定全部可能的图形参数。你需要参考每个特定绘图函数的帮助（如?plot、?hist或?boxplot）以确定哪些参数可以以这种方式设置。下面介绍可以设定的许多重要图形参数。

符号和线条

如你所见，可以使用图形参数来指定绘图时使用的符号和线条类型。相关参数如表2所示。

表2　用于指定符号和线条类型的参数

参　数	描　　述
pch	指定绘制点时使用的符号（见图4）
cex	指定符号的大小。cex是一个数值，表示绘图符号相对于默认大小的缩放倍数。默认大小为1，1.5表示放大为默认值的1.5倍，0.5表示缩小为默认值的50%，等等
lty	指定线条类型（参见图5）
lwd	指定线条宽度。lwd是以默认值的相对大小来表示的（默认值为1）。例如，lwd=2将生成一条两倍于默认宽度的线条

选项pch=用于指定绘制点时使用的符号。可能的值如图4所示。

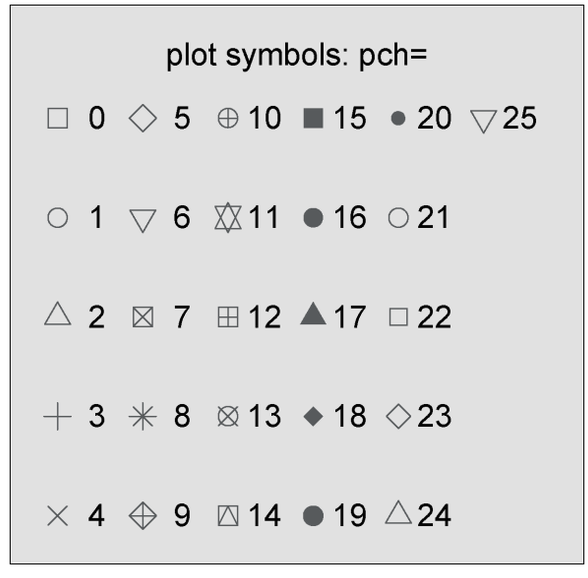


图4　参数pch可指定的绘图符号

对于符号21~25，你还可以指定边界颜色（col=）和填充色（bg=）。

选项lty=用于指定想要的线条类型。可用的值如图5所示。

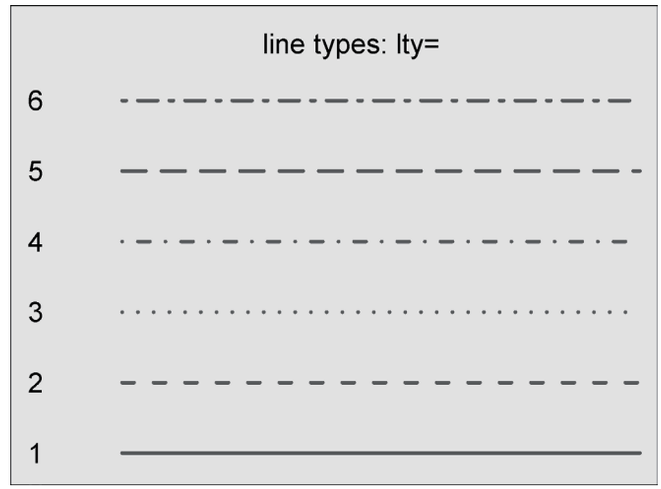


图5　参数lty可指定的线条类型

综合以上选项，以下代码：

```
plot(dose, drugA, type="b", lty=3, lwd=3, pch=15, cex=2)
```

将绘制一幅图形，其线条类型为点线，宽度为默认宽度的3倍，点的符号为实心正方形，大小为默认符号大小的2倍。结果如图6所示。

接下来我们将讨论颜色的指定方法。

颜色

R中有若干和颜色相关的参数。表3列出了一些常用参数。

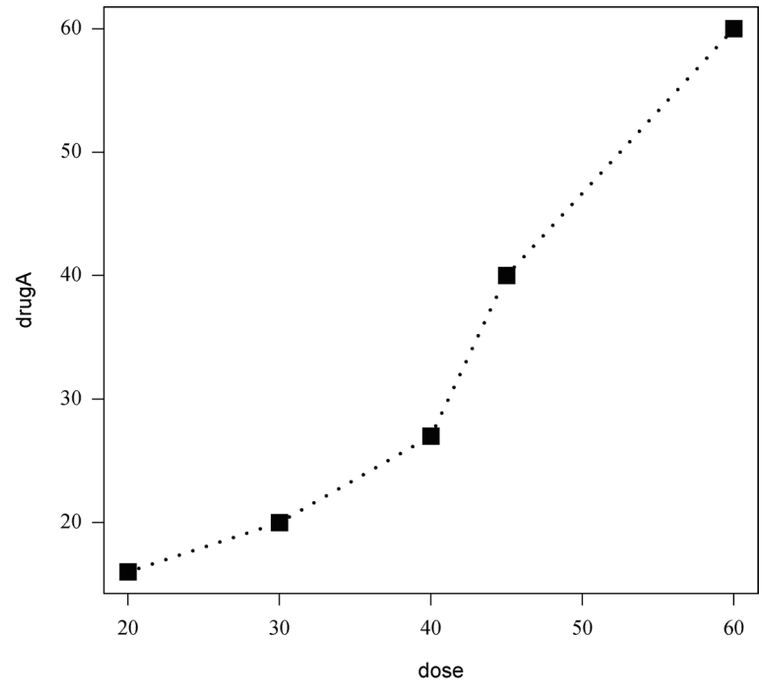


图6　药物A剂量和响应的折线图。修改了线条类型、线条宽度、点的符号和符号大小

表3　用于指定颜色的参数

参　数	描　　述
col	默认的绘图颜色。某些函数（如lines和pie）可以接受一个含有颜色值的向量并自动循环使用。例如，如果设定col=c("red", "blue")并需要绘制三条线，则第一条线将为红色，第二条线为蓝色，第三条线又将为红色
col.axis	坐标轴刻度文字的颜色
col.lab	坐标轴标签（名称）的颜色
col.main	标题颜色
col.sub	副标题颜色
fg	图形的前景色
bg	图形的背景色

在R中，可以通过颜色下标、颜色名称、十六进制的颜色值、RGB值或HSV值来指定颜色。举例来说，`col=1`、`col="white"`、`col="#FFFFFF"`、`col=rgb(1,1,1)`和`col=hsv(0,0,1)`都是表示白色的等价方式。函数

rgb() 可基于红—绿—蓝三色值生成颜色，而hsv() 则基于色相—饱和度—亮度值来生成颜色。请参考这些函数的帮助以了解更多细节。

函数colors() 可以返回所有可用颜色的名称。Earl F. Glynn为R中的色彩创建了一个优秀的在线图表，参见http://research.stowers-institute.org/efg/R/Color/Chart。R中也有多种用于创建连续型颜色向量的函数，包括rainbow()、heat.colors()、terrain.colors()、topo.colors() 以及cm.colors()。举例来说，rainbow(10) 可以生成10种连续的“彩虹型”颜色。多阶灰度色可使用gray()函数生成。这时要通过一个元素值为0和1之间的向量来指定各颜色的灰度。gray(0:10/10) 将生成10阶灰度色。试着使用以下代码:

```
n <- 10
mycolors <- rainbow(n)
pie(rep(1, n), labels=mycolors, col=mycolors)
mygrays <- gray(0:n/n)
pie(rep(1, n), labels=mygrays, col=mygrays)
```

来观察这些函数的工作方式。我们始终会有使用颜色参数的示例。

文本属性

图形参数同样可以用来指定字号、字体和字样。表4阐释了用于控制文本大小的参数。字体族和字样可以通过字体选项进行控制（见表5）。

表4　用于指定文本大小的参数

参　数	描　　述
cex	表示相对于默认大小缩放倍数的数值。默认大小为1，1.5表示放大为默认值的1.5倍，0.5表示缩小为默认值的50%，等等
cex.axis	坐标轴刻度文字的缩放倍数。类似于cex
cex.lab	坐标轴标签（名称）的缩放倍数。类似于cex
cex.main	标题的缩放倍数。类似于cex
cex.sub	副标题的缩放倍数。类似于cex

表5　用于指定字体族、字号和字样的参数

参　数	描　　述
font	整数。用于指定绘图使用的字体样式。1=常规，2=粗体，3=斜体，4=粗斜体，5=符号字体（以Adobe符号编码表示）
font.axis	坐标轴刻度文字的字体样式
font.lab	坐标轴标签（名称）的字体样式
font.main	标题的字体样式
font.sub	副标题的字体样式
ps	字体磅值（1磅约为1/72英寸）。文本的最终大小为 ps*cex
family	绘制文本时使用的字体族。标准的取值为serif（衬线）、sans（无衬线）和mono（等宽）

举例来说，在执行语句：

```
par(font.lab=3, cex.lab=1.5, font.main=4, cex.main=2)
```

之后创建的所有图形都将拥有斜体、1.5倍于默认文本大小的坐标轴标签（名称），以及粗斜体、2倍于默认文本大小的标题。

我们可以轻松设置字号和字体样式，然而字体族的设置却稍显复杂。这是因为衬线、无衬线和等宽字体的具体映射是与图形设备相关的。举例来说，在Windows系统中，等宽字体映射为TT Courier New，衬线字体映射为TT Times New Roman，无衬线字体则映射为TT Arial（TT代表True Type）。如果你对以上映射表示满意，就可以使用类似于`family="serif"`这样的参数获得想要的结果。如果不满意，则需要创建新的映射。在Windows中，可以通过函数`windowsFont()`来创建这类映射。例如，在执行语句：

```
windowsFonts(
  A=windowsFont("Arial Black"),
  B=windowsFont("Bookman Old Style"),
  C=windowsFont("Comic Sans MS")
)
```

之后，即可使用A、B和C作为family的取值。在本例的情境下，par(family="A")将指定Arial Black作为绘图字体。请注意，函数windowsFont()仅在Windows中有效。在Mac上，请改用quartzFonts()。

如果以PDF或PostScript格式输出图形，则修改字体族会相对简单一些。[1]对于PDF格式，可以使用names(pdfFonts())找出你的系统中有哪些字体是可用的，然后使用pdf(file="*myplot.pdf*", family="*fontname*")来生成图形。对于以PostScript格式输出的图形，则可以对应地使用names(postscriptFonts())和postscript(file="*myplot.ps*", family= "*fontname*")。请参阅在线帮助以了解更多信息。

[1] PDF中文字体的使用比较麻烦，同时在Linux系统中可能会遇到中文字体无法嵌入的问题。Windows上的解决方案之一是使用Cairo包中的CairoPDF()函数。此话题的详细讨论请参考http://cos.name/cn/topic/101521。——译者注

图形尺寸与边界尺寸

最后，可以使用表6列出的参数来控制图形尺寸和边界大小。

表6　用于控制图形尺寸和边界大小的参数

参　数	描　　述
pin	以英寸表示的图形尺寸（宽和高）
mai	以数值向量表示的边界大小，顺序为“下、左、上、右”，单位为英寸
mar	以数值向量表示的边界大小，顺序为“下、左、上、右”，单位为英分*。默认值为c(5, 4, 4, 2) + 0.1

*一英分等于十二分之一英寸。——译者注

代码：

```
par(pin=c(4,3), mai=c(1,.5, 1, .2))
```

可生成一幅4英寸宽、3英寸高、上下边界为1英寸、左边界为0.5英寸、右边界为0.2英寸的图形。关于边界参数的完整指南，不妨参阅

Earl F. Glynn编写的一份全面的在线教程（http://research.stowers-institute.org/efg/R/Graphics/Basics/mar-oma/）。

让我们使用最近学到的选项来强化之前的简单图形示例。代码清单1中的代码生成的图形如图7所示。

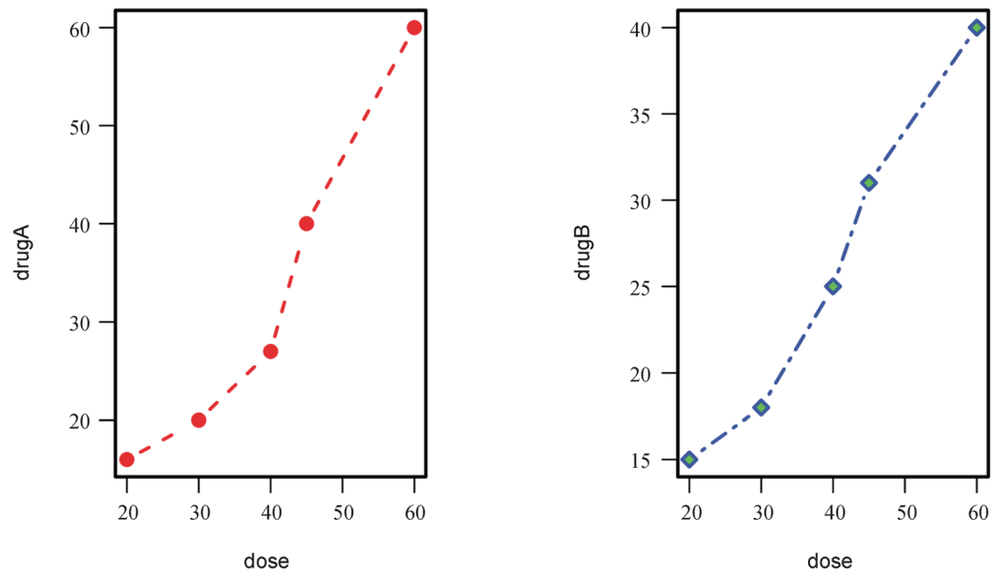


图7　药物A和药物B剂量与响应的折线图

代码清单1　使用图形参数控制图形外观

```
dose  <- c(20, 30, 40, 45, 60)
drugA <- c(16, 20, 27, 40, 60)
drugB <- c(15, 18, 25, 31, 40)
opar <- par(no.readonly=TRUE)
par(pin=c(2, 3))
par(lwd=2, cex=1.5)
par(cex.axis=.75, font.axis=3)
plot(dose, drugA, type="b", pch=19, lty=2, col="red")
plot(dose, drugB, type="b", pch=23, lty=6, col="blue", bg="green")
par(opar)
```

首先，你以向量的形式输入了数据，然后保存了当前的图形参数设置（这样就可以在稍后恢复设置）。接着，你修改了默认的图形参数，这样，得到的图形将为2英寸宽、3英寸高。除此之外，线条的宽度将为默认宽度的两倍，符号将为默认大小的1.5倍。坐标轴刻度文本被设置为斜体、缩小为默认大小的75%。之后，我们使用红色实心圆圈和虚线创建了第一幅图形，并使用绿色填充的绿色菱形加蓝色边框和蓝色虚线创建了第二幅图形。最后，我们还原了初始的图形参数设置。

值得注意的是，通过par()设定的参数对两幅图都有效，而在绘图函数中指定的参数仅对那个特定图形有效。观察图7可以发现，图形的呈现上还有一定缺陷。这两幅图都缺少标题，并且纵轴的刻度单位不同，这无疑限制了我们直接比较两种药物的能力。同时，坐标轴的标签（名称）也应当提供更多的信息。

接下来，我们将转而探讨如何自定义文本标注（如标题和标签）和坐标轴。要了解可用图形参数的更多信息，请参阅help(par)。

添加文本、自定义坐标轴和图例

除了图形参数，许多高级绘图函数（例如plot、hist、boxplot）也允许自行设定坐标轴和文本标注选项。举例来说，以下代码在图形上添加了标题（main）、副标题（sub）、坐标轴标签（xlab、ylab）并指定了坐标轴范围（xlim、ylim）。结果如图3-8所示。

```
plot(dose, drugA, type="b",
     col="red", lty=2, pch=2, lwd=2,
     main="Clinical Trials for Drug A",
     sub="This is hypothetical data",
     xlab="Dosage", ylab="Drug Response",
     xlim=c(0, 60), ylim=c(0, 70))
```

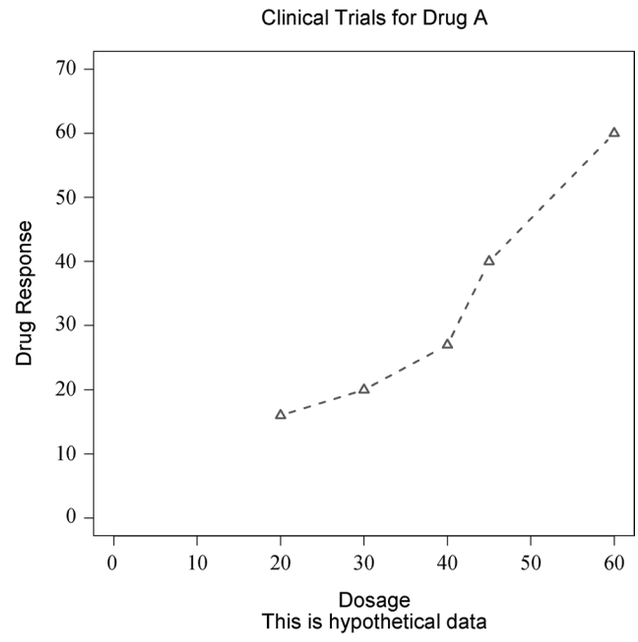


图8　药物A剂量和响应的折线图。添加了标题、副标题和自定义的坐标轴

再次提醒，并非所有函数都支持这些选项。请参考相应函数的帮助以了解其可以接受哪些选项。从更精细的控制和模块化的角度考虑，你可以使用余下部分描述的函数来控制标题、坐标轴、图例和文本标注的外观。

注意

某些高级绘图函数已经包含了默认的标题和标签。你可以通过在plot()语句或单独的par()语句中添加ann=FALSE来移除它们。

标题

可以使用`title()`函数为图形添加标题和坐标轴标签。调用格式为：

```
title(main="main title", sub="sub-title",
      xlab="x-axis label", ylab="y-axis label")
```

函数title()中亦可指定其他图形参数（如文本大小、字体、旋转角度和颜色）。举例来说，以下代码将生成红色的标题和蓝色的副标题，以及较默认大小小25%的绿色x轴、y轴标签：

```
title(main="My Title", col.main="red",
      sub="My Sub-title", col.sub="blue",
      xlab="My X label", ylab="My Y label",
      col.lab="green", cex.lab=0.75)
```

坐标轴

你可以使用函数axis()来创建自定义的坐标轴，而非使用R中的默认坐标轴。其格式为：

```
axis(side, at=, labels=, pos=, lty=, col=, las=, tck=, ...)
```

各参数已详述于表7中。

表7　坐标轴选项

选　项	描　　述
side	一个整数，表示在图形的哪边绘制坐标轴（1=下，2=左，3=上，4=右）
at	一个数值型向量，表示需要绘制刻度线的位置
labels	一个字符型向量，表示置于刻度线旁边的文字标签（如果为NULL，则将直接使用at中的值）
pos	坐标轴线绘制位置的坐标（即与另一条坐标轴相交位置的值）
lty	线条类型
col	线条和刻度线颜色
las	标签是否平行于（=0）或垂直于（=2）坐标轴
tck	刻度线的长度，以相对于绘图区域大小的分数表示（负值表示在图形外侧，正值表示在图形内侧，0表示禁用刻度，1表示绘制网格线）；默认值为-0.01
(…)	其他图形参数

创建自定义坐标轴时，你应当禁用高级绘图函数自动生成的坐标轴。参数axes=FALSE将禁用全部坐标轴（包括坐标轴框架线，除非你添加了参数frame.plot=TRUE）。参数xaxt="n"和yaxt="n"将分别禁用X轴或Y轴（会留下框架线，只是去除了刻度）。代码清单2中是一个稍显笨拙和夸张的例子，它演示了我们到目前为止讨论过的各种图形特征。结果如图9所示。

代码清单2　自定义坐标轴的示例

```
x <- c(1:10)      # 生成数据
y <- x
z <- 10/x

opar <- par(no.readonly=TRUE)

par(mar=c(5, 4, 4, 8) + 0.1)       # 增加边界大小

plot(x, y, type="b",         # 绘制x对y的图形
     pch=21, col="red",
     yaxt="n", lty=3, ann=FALSE)

lines(x, z, type="b", pch=22, col="blue", lty=2)  # 添加x对1/x的直线

axis(2, at=x, labels=x, col.axis="red", las=2)    # 绘制你自己的坐标轴

axis(4, at=z, labels=round(z, digits=2),
     col.axis="blue", las=2, cex.axis=0.7, tck=-.01)

mtext("y=1/x", side=4, line=3, cex.lab=1, las=2, col="blue")  # 添加
标题和文本

title("An Example of Creative Axes",
      xlab="X values",
      ylab="Y=X")

par(opar)
```

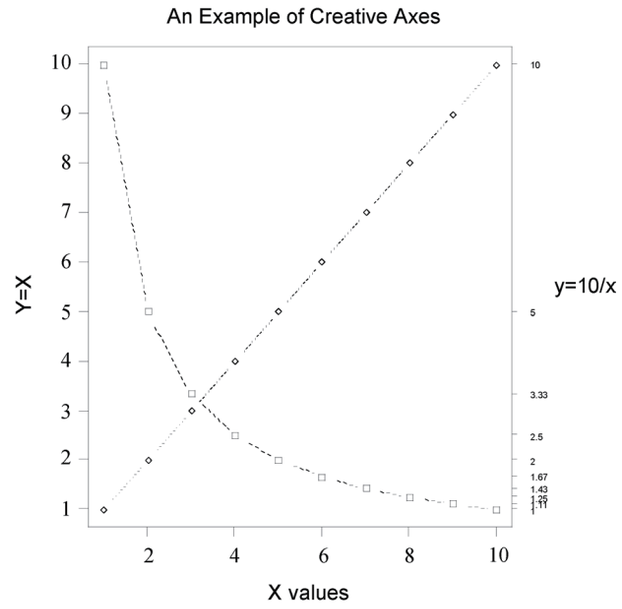


图9　各种坐标轴选项的演示

到目前为止，我们已经讨论过代码清单2中除`lines()`和`mtext()`以外的所有函数。使用`plot()`语句可以新建一幅图形。而使用`lines()`语句，你可以为一幅**现有**图形添加新的图形元素。函数`mtext()`用于在图形的边界添加文本。

次要刻度线

注意，我们最近创建的图形都只拥有主刻度线，却没有次要刻度线。要创建次要刻度线，你需要使用`Hmisc`包中的`minor.tick()`函数。如果你尚未安装`Hmisc`包，请先安装它。你可以使用代码：

```
library(Hmisc)
minor.tick(nx=n, ny=n, tick.ratio=n)
```

来添加次要刻度线。其中nx和ny分别指定了X轴和Y轴每两条主刻度线之间通过次要刻度线划分得到的区间个数。tick.ratio表示次要刻度线相对于主刻度线的大小比例。当前的主刻度线长度可以使用par("tck")获取。举例来说，下列语句将在X轴的每两条主刻度线之间添加1条次要刻度线，并在Y轴的每两条主刻度线之间添加2条次要刻度线：

```
minor.tick(nx=2, ny=3, tick.ratio=0.5)
```

次要刻度线的长度将是主刻度线的一半。

参考线

函数abline()可以用来为图形添加参考线。其使用格式为：

```
abline(h=yvalues, v=xvalues)
```

函数abline()中也可以指定其他图形参数（如线条类型、颜色和宽度）。举例来说：

```
abline(h=c(1,5,7))
```

在y为1、5、7的位置添加了水平实线，而代码：

```
abline(v=seq(1, 10, 2), lty=2, col="blue")
```

则在x为1、3、5、7、9的位置添加了垂直的蓝色虚线。代码清单3为我们的药物效果图在$y = 30$的位置创建了一条参考线。结果如图10所示。

图例

当图形中包含的数据不止一组时，图例可以帮助你辨别出每个条形、扇

形区域或折线各代表哪一类数据。我们可以使用函数legend()来添加图例（果然不出所料）。其使用格式为：

```
legend(location, title, legend, ...)
```

常用选项详述于表8中。

表8　图例选项

选　项	描　　述
location	有许多方式可以指定图例的位置。你可以直接给定图例左上角的x、y坐标，也可以执行locator(1)，然后通过鼠标单击给出图例的位置，还可以使用关键字bottom、bottomleft、left、topleft、top、topright、right、bottomright或center放置图例。如果你使用了以上某个关键字，那么可以同时使用参数inset=指定图例向图形内侧移动的大小（以绘图区域大小的分数表示）
title	图例标题的字符串（可选）
legend	图例标签组成的字符型向量
…	其他选项。如果图例标示的是颜色不同的线条，需要指定col=加上颜色值组成的向量。如果图例标示的是符号不同的点，则需指定pch=加上符号的代码组成的向量。如果图例标示的是不同的线条宽度或线条类型，请使用lwd=或lty=加上宽度值或类型值组成的向量。要为图例创建颜色填充的盒形（常见于条形图、箱线图或饼图），需要使用参数fill=加上颜色值组成的向量

其他常用的图例选项包括用于指定盒子样式的bty、指定背景色的bg、指定大小的cex，以及指定文本颜色的text.col。指定horiz=TRUE将会水平放置图例，而不是垂直放置。关于图例的更多细节，请参考help(legend)。这份帮助中给出的示例都特别有用。

让我们看看对药物数据作图的一个例子（代码清单3）。你将再次使用我们目前为止讲到的许多图形功能。结果如图10所示。

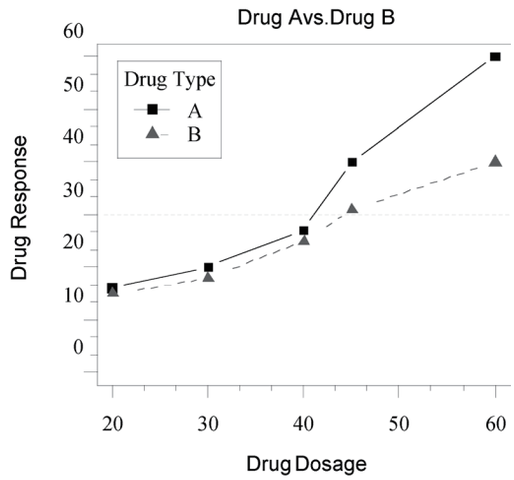


图10　进行标注后的图形，对比了药物A和药物B的效果

代码清单3　依剂量对比药物A和药物B的响应情况

```
dose  <- c(20, 30, 40, 45, 60)
drugA <- c(16, 20, 27, 40, 60)
drugB <- c(15, 18, 25, 31, 40)

opar <- par(no.readonly=TRUE)

par(lwd=2, cex=1.5, font.lab=2)        # 增加线条、文本、符号、标签的宽
度或大小

plot(dose, drugA, type="b",
     pch=15, lty=1, col="red", ylim=c(0, 60),       # 绘制图形
     main="Drug A vs. Drug B",
     xlab="Drug Dosage", ylab="Drug Response")

lines(dose, drugB, type="b",
      pch=17, lty=2, col="blue")

abline(h=c(30), lwd=1.5, lty=2, col="gray")

library(Hmisc)          # 添加次要刻度线
minor.tick(nx=3, ny=3, tick.ratio=0.5)
```

```
legend("topleft", inset=.05, title="Drug Type", c("A","B"),
# 添加图例
       lty=c(1, 2), pch=c(15, 17), col=c("red", "blue"))

par(opar)
```

图10的几乎所有外观元素都可以使用我们讨论过的选项进行修改。除此之外，还有很多其他方式可以指定想要的选项。

文本标注

我们可以通过函数text()和mtext()将文本添加到图形上。text()可向绘图区域内部添加文本，而mtext()则向图形的四个边界之一添加文本。使用格式分别为：

```
text(location, "text to place", pos, ...)
mtext("text to place", side, line=n, ...)
```

常用选项列于表9中。

表9　函数text()和mtext()的选项

选　项	描　　述
location	文本的位置参数。可为一对x,y坐标，也可通过指定location为locator(1)使用鼠标交互式地确定摆放位置
pos	文本相对于位置参数的方位。1=下，2=左，3=上，4=右。如果指定了pos，就可以同时指定参数offset=作为偏移量，以相对于单个字符宽度的比例表示
side	指定用来放置文本的边。1=下，2=左，3=上，4=右。你可以指定参数line=来内移或外移文本，随着值的增加，文本将外移。也可使用adj=0将文本向左下对齐，或使用adj=1右上对齐

其他常用的选项有cex、col和font（分别用来调整字号、颜色和字体样式）。

除了用来添加文本标注以外，text()函数也通常用来标示图形中的点。我们只需指定一系列的x, y坐标作为位置参数，同时以向量的形式指定要放置的文本。x、y和文本标签向量的长度应当相同。下面给出了一个示例，结果如图11所示。

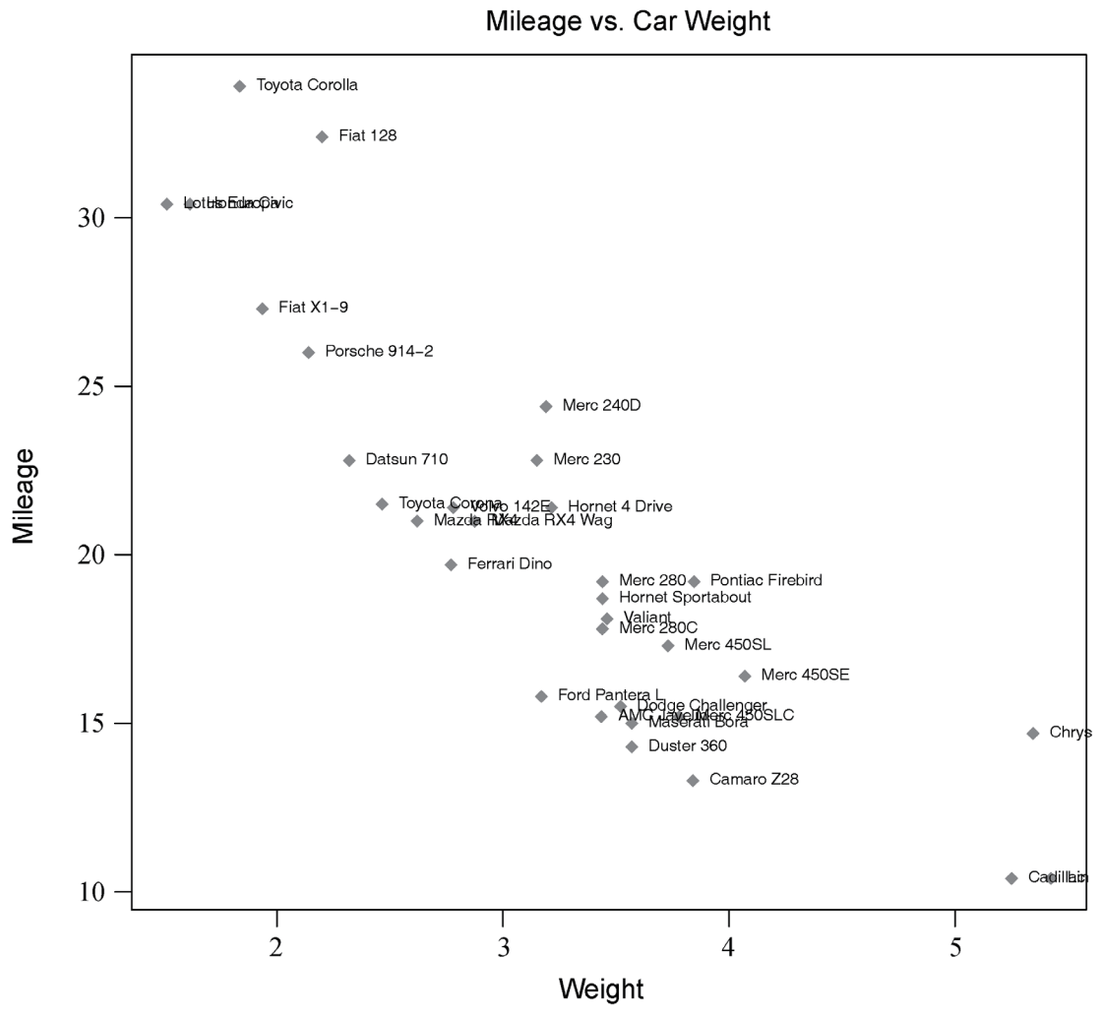


图11　一幅散点图（车重与每加仑汽油行驶英里数）的示例，各点均添加了标签（车型）

```
attach(mtcars)
plot(wt, mpg,
     main="Mileage vs. Car Weight",
     xlab="Weight", ylab="Mileage",
     pch=18, col="blue")
text(wt, mpg,
```

```
     row.names(mtcars),
     cex=0.6, pos=4, col="red")
detach(mtcars)
```

这里，我们针对数据框mtcars提供的32种车型的车重和每加仑汽油行驶英里数绘制了散点图。函数text()被用来在各个数据点右侧添加车辆型号。各点的标签大小被缩小了40%，颜色为红色。

作为第二个示例，以下是一段展示不同字体族的代码：

```
opar <- par(no.readonly=TRUE)
par(cex=1.5)
plot(1:7,1:7,type="n")
text(3,3,"Example of default text")
text(4,4,family="mono","Example of mono-spaced text")
text(5,5,family="serif","Example of serif text")
par(opar)
```

在Windows系统中输出的结果如图12所示。这里为了获得更好的显示效果，我们使用par()函数增大了字号。

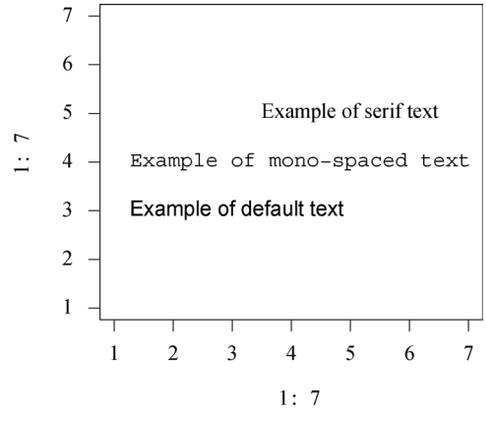


图12　Windows中不同字体族的示例

本例所得结果因平台而异，因为不同系统中映射的常规字体、等宽字体和有衬线字体有所不同。在你的系统上，结果看起来如何呢？

数学标注

最后，你可以使用类似于TeX中的写法为图形添加数学符号和公式。请参阅help(plotmath)以获得更多细节和示例。要即时看效果，可以尝试执行`demo(plotmath)`。部分运行结果如图13所示。函数`plotmath()`可以为图形主体或边界上的标题、坐标轴名称或文本标注添加数学符号。

Arithmetic Operators		Radicals	
x + y	x + y	sqrt(x)	$\sqrt{x}$
x − y	x − y	sqrt(x, y)	$\sqrt[y]{x}$
x * y	xy	Relations	
x/y	x/y	x == y	x = y
x %+−% y	x ± y	x != y	x ↑ y
x%/%y	x √ y	x < y	x < y
x %*% y	x × y	x <= y	x ″ y
x %.% y	x · y	x > y	x > y
−x	−x	x >= y	x ⩾ y
+x	+x	x %~~% y	x ⊕ y
Sub/Superscripts		x %=~% y	x ≅ y
x[i]	x_i	x %==% y	x ≡ y
x^2	x^2	x %prop% y	x ∝ y
Juxtaposition		Typeface	
x * y	xy	plain(x)	x
paste(x,y,z)	xyz	italic(x)	*x*
Lists		bold(x)	**x**
list(x,y,z)	x, y, z	bolditalic(x)	***x***
		underline(x)	x

图13　demo(plotmath)的部分结果

同时比较多幅图形，我们通常可以更好地洞察数据的性质。所以，作为本文的结尾，下面讨论将多幅图形组合为一幅图形的方法。

图形的组合

在R中使用函数par()或layout()可以容易地组合多幅图形为一幅总括图形。此时请不要担心所要组合图形的具体类型，这里我们只关注组合它们的一般方法。

你可以在par()函数中使用图形参数mfrow=c(*nrows*, *ncols*)来创建按行填充的、行数为*nrows*、列数为*ncols*的图形矩阵。另外，可以使用nfcol=c(*nrows*, *ncols*)按列填充矩阵。

举例来说，以下代码创建了四幅图形并将其排布在两行两列中：

```
attach(mtcars)
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(wt,mpg, main="Scatterplot of wt vs. mpg")
plot(wt,disp, main="Scatterplot of wt vs disp")
hist(wt, main="Histogram of wt")
boxplot(wt, main="Boxplot of wt")
par(opar)
detach(mtcars)
```

结果如图14所示。

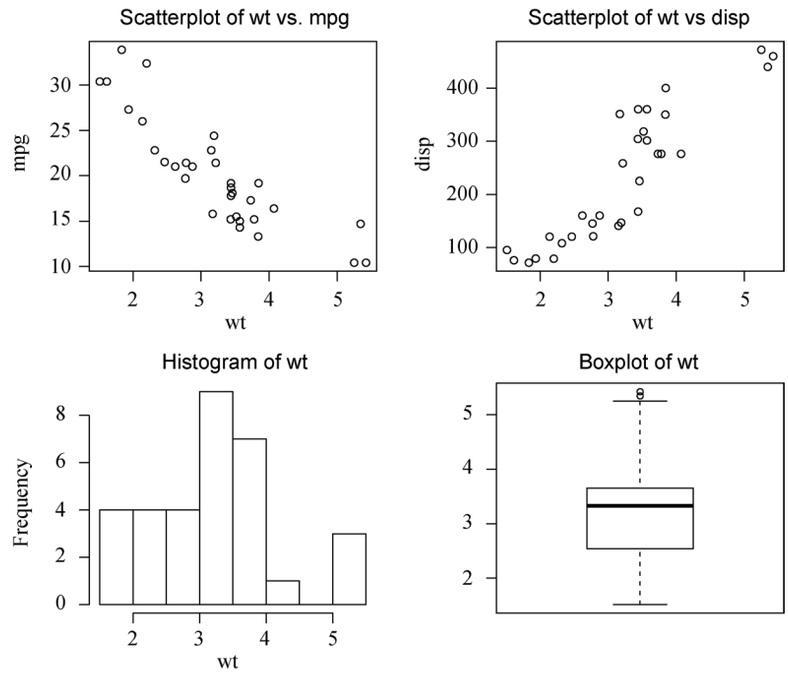


图14　通过par(mfrow=c(2,2))组合的四幅图形

作为第二个示例，让我们依3行1列排布3幅图形。代码如下：

```
attach(mtcars)
opar <- par(no.readonly=TRUE)
par(mfrow=c(3,1))
hist(wt)
hist(mpg)
hist(disp)
par(opar)
detach(mtcars)
```

所得图形如图15所示。请注意，高级绘图函数hist()包含了一个默认的标题（使用main=""可以禁用它，抑或使用ann=FALSE来禁用所有标题和标签）。

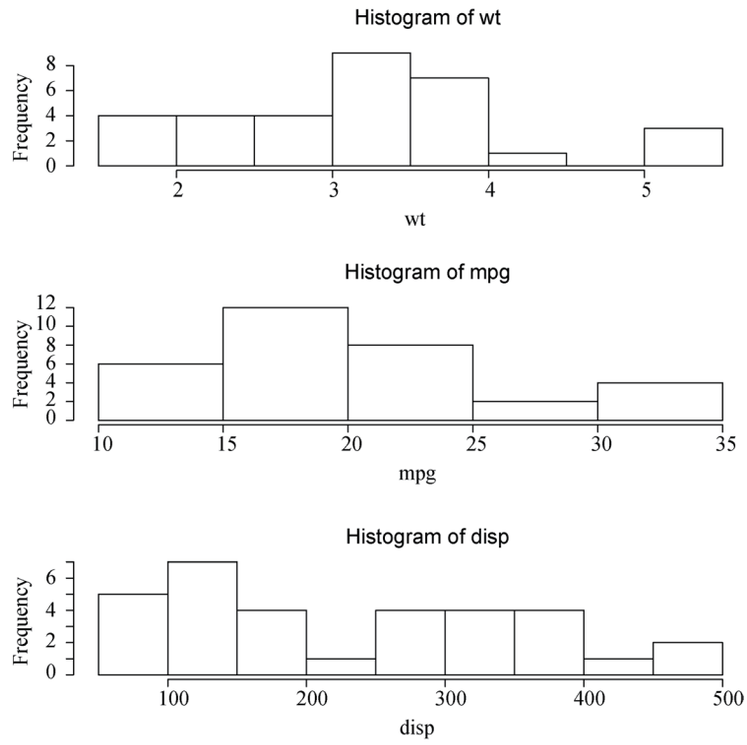


图15　通过par(mfrow=c(3,1))组合的三幅图形

函数layout()的调用形式为layout(*mat*)，其中的mat是一个矩阵，它指定了所要组合的多个图形的所在位置。在以下代码中，一幅图被置于第1行，另两幅图则被置于第2行：

```
attach(mtcars)
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
hist(wt)
hist(mpg)
hist(disp)
detach(mtcars)
```

结果如图16所示。

为了更精确地控制每幅图形的大小，可以有选择地在layout()函数中使用widths=和heights=两个参数。其形式为：

widths = 各列宽度值组成的一个向量

heights = 各行高度值组成的一个向量

相对宽度可以直接通过数值指定，绝对宽度（以厘米为单位）可以通过函数lcm()来指定。

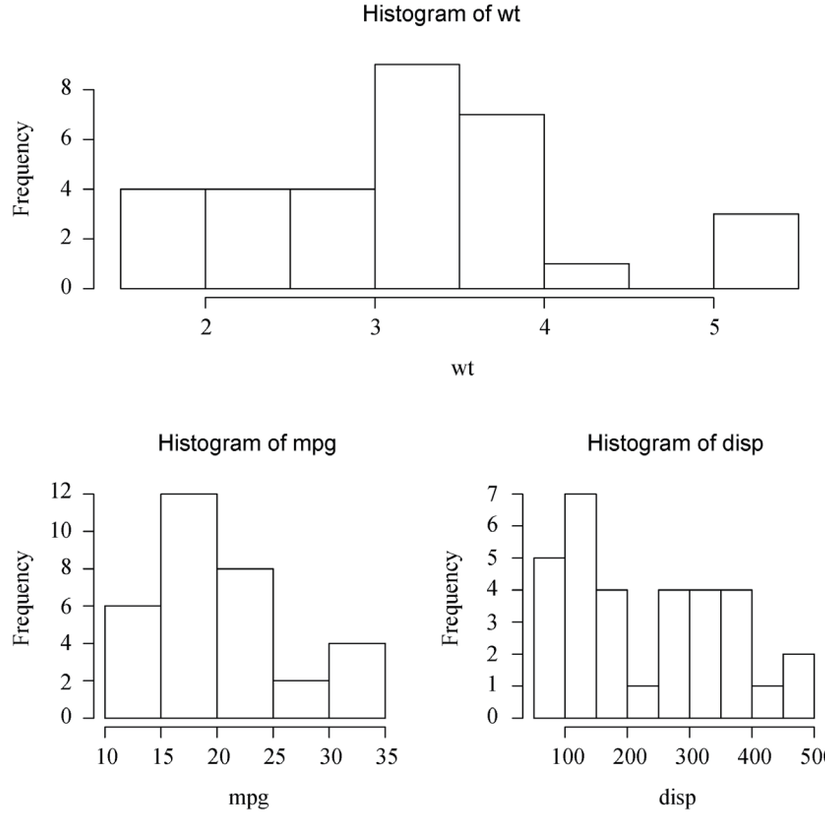


图16　使用函数layout()组合的三幅图形，各列宽度为默认值

在以下代码中，我们再次将一幅图形置于第1行，两幅图形置于第2行。但第1行中图形的高度是第2行中图形高度的三分之一。除此之外，右下角图形的宽度是左下角图形宽度的四分之一：

```
attach(mtcars)
layout(matrix(c(1, 1, 2, 3), 2, 2, byrow = TRUE),
       widths=c(3, 1), heights=c(1, 2))
hist(wt)
hist(mpg)
hist(disp)
detach(mtcars)
```

所得图形如图17所示。

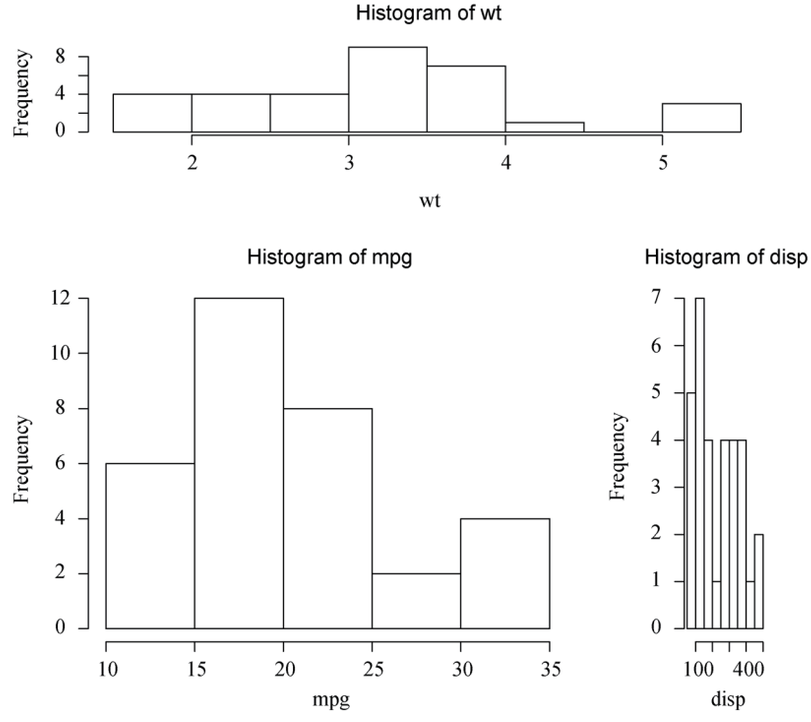


图17　使用函数layout()组合的三幅图形，各列宽度为指定值

如你所见，layout()函数能够让我们轻松地控制最终图形中的子图数量和摆放方式，以及这些子图的相对大小。请参考help(layout)以了解更多细节。

图形布局的精细控制

可能有很多时候，你想通过排布或叠加若干图形来创建单幅的、有意义的图形，这需要有对图形布局的精细控制能力。你可以使用图形参数fig=完成这个任务。代码清单4通过在散点图上添加两幅箱线图，创建了单幅的增强型图形。结果如图18所示。

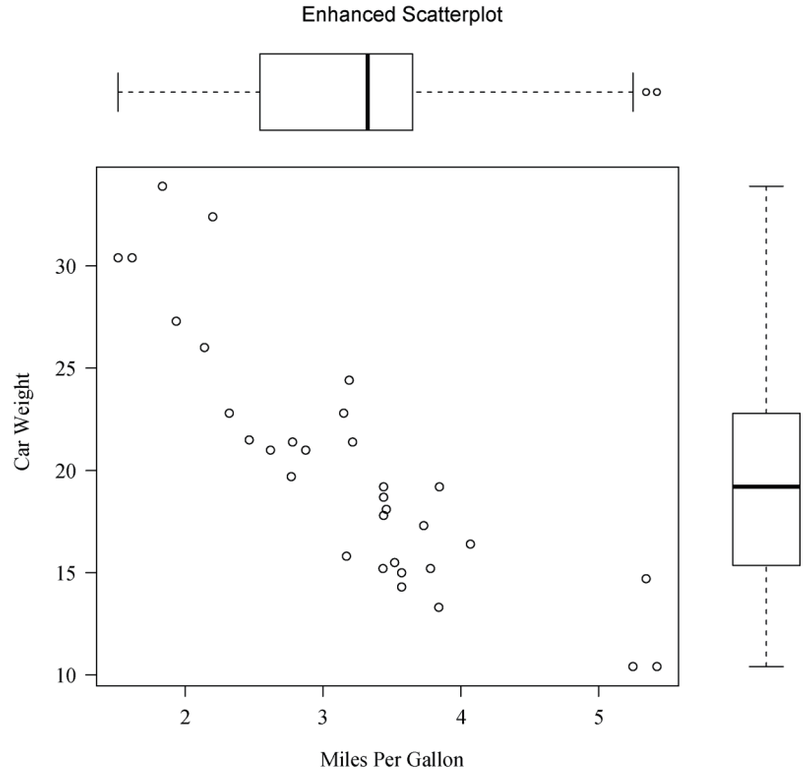
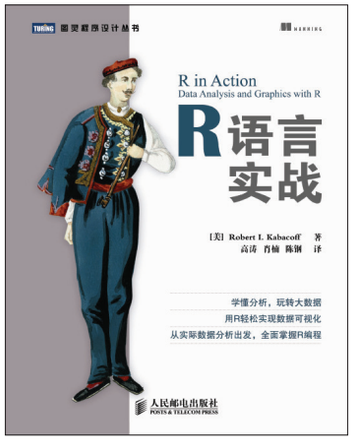


图18　边界上添加了两幅箱线图的散点图



《R语言实战》从解决实际问题入手，尽量跳脱统计学的理论阐述来讨论R语言及其应用，讲解清晰透澈，极具实用性。作者不仅高度概括了R语言的强大功能、展示了各种实用的统计示例，而且对于难以用传统方法分析的凌乱、不完整和非正态的数据也给出了完备的处理方法。通读本书，你将全面掌握使用R语言进行数据分析、数据挖掘的技巧，并领略大量探索和展示数据的图形功能，从而更加高效地进行分析与沟通。本文节选自《R语言实战》。

代码清单4　多幅图形布局的精细控制

```
opar <- par(no.readonly=TRUE)
par(fig=c(0, 0.8, 0, 0.8))                    # 设置散点图
plot(mtcars$wt, mtcars$mpg,
     xlab="Miles Per Gallon",
     ylab="Car Weight")

par(fig=c(0, 0.8, 0.55, 1), new=TRUE)         # 在上方添加箱线图
boxplot(mtcars$wt, horizontal=TRUE, axes=FALSE)
par(fig=c(0.65, 1, 0, 0.8), new=TRUE)         # 在右侧添加箱线图
boxplot(mtcars$mpg, axes=FALSE)

mtext("Enhanced Scatterplot", side=3, outer=TRUE, line=-3)
par(opar)
```

要理解这幅图的绘制原理，请试想完整的绘图区域：左下角坐标为(0,0)，而右上角坐标为(1,1)。图19是一幅示意图。参数fig=的取值是一个形如c(x1, x2, y1, y2)的数值向量。

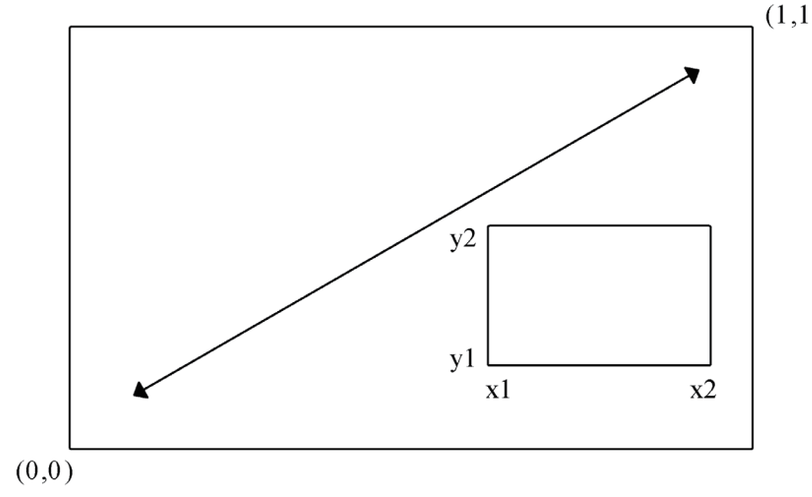


图19　使用图形参数fig=指定位置

第一个 fig= 将散点图设定为占据横向范围0~0.8，纵向范围0~0.8。上方的箱线图横向占据0~0.8，纵向0.55~1。右侧的箱线图横向占据0.65~1，纵向0~0.8。fig= 默认会新建一幅图形，所以在添加一幅图到一幅现有图形上时，请设定参数new=TRUE。

我将参数选择为0.55而不是0.8，这样上方的图形就不会和散点图拉得太远。类似地，我选择了参数0.65以拉近右侧箱线图和散点图的距离。你需要不断尝试找到合适的位置参数。

注意

各独立子图所需空间的大小可能与设备相关。如果你遇到了“Error in plot.new(): figure margins too large”这样的错误，请尝试在整个图形的范围内修改各个子图占据的区域位置和大小。

你可以使用图形参数 fig= 将若干图形以任意排布方式组合到单幅图形中。稍加练习，你就可以通过这种方法极其灵活地创建复杂的视觉呈现。 ■

读《码农》吐吐槽 还能赚银子！

《码农》电子刊如今慢慢悠悠已经出版了17期，从一开始的好评如潮，到现在的“怎么这么抽象啊”“赶脚内容没有以前好了？”“一下就翻完了，没什么内容啊”……小编表示压力很大，现在的码农读者太难伺候了，明明是本免费杂志，￥%#%&*%@#￥*@￥#@！

但是，一看到好评，盼盼姐还是会热泪盈眶，热血沸腾，各种正能量啊“这么好质量的杂志还免费，天上掉馅饼呀”“希望一直出！每一版都读了，很值得推荐！！”……

所以《码农》还是会一直做下去，但是，是好是坏，您得给个话儿啊！

活动规则：在吐槽贴中留言，选出任何一期中你最喜欢的文章和最不喜欢的文章，即可获得图灵社区银子2两！凡吐槽吐得掷地有声者，加赠银子3两（共5两）！（怎样使用银子兑换图书）

活动时间：本活动长期有效。

连城：大数据场景下的“骚到痒处”和“戳到痛处”

连城，Databricks工程师，Apache Spark committer。《Erlang/OTP并发编程实战》与《Erlang并发编程(第一篇)》译者。目前从事Apache Spark中结构化数据分析组件Spark SQL的开发。

在做Spark之前，连城从来没有做过大数据分析方向的工作。为了理解函数式编程，他用Scheme写了两年的side project；为了学习分布式存储，他把1974年以后的分布式文献都过了一遍；为了参与Spark相关的项目，他在Coursera上自学了Scala。如今作为Spark committer的他，对大数据分析逐渐形成了自己的理解，他认为“对工具的选择，既可以解放我们的思想，也可以禁锢我们的思想”。而他自己曾经并不感冒的函数式编程，才是更加契合大数据场景的编程方式。

爱上函数式编程

> 因为从小到大走的都是C/C++/Java这类命令式语言的路线，函数式语言对于当时的我来说实在是很不合胃口。

问：你是从什么时候开始编程的？

初一的时候因为我爸爸工作需要，家里买了台电脑，我就拿着电脑画画、玩扫雷。我爸大概是觉得我扫雷太上瘾了，就不知从哪儿拣了一本少年儿童BASIC编程之类的书。现在唯一有印象的是那本书插图特别地粗糙，讲的基本概念也非常模糊。就i=i+1这么一个概念，花了我好几个礼拜才搞明白。因为那本书根本没有讲到赋值跟相等判断的区别，而BASIC里面赋值跟相等判断都是一个等号。不管怎么样，正是这本书让我知道了计算机编程这个事物的存在。

问：大学学的是什么？

大学学的计算机。这是从初中开始就打定了主意的。初二时班级重组，恰好发现坐我前面的男生下课时间在草稿纸上涂写BASIC程序，一问才知道他从小学就开始写程序，于是拜师学艺。后来成了莫逆之交。现在他也在湾区工作。那时候其实也写不出什么有技术含量的东西。但是无知无畏啊，觉得写程序的时候有造物主的感觉，于是就下定决心走这条路了。

问：为什么对函数式编程和分布式系统感兴趣？

二者都是从工作需要出发，逐渐探索出来的兴趣。虽然从初中就开始写程序，但是其实在本科之前从来没有接受过正规的计算机教育。毕业的时候各种基础都一般，也没有确定到底要做哪个细分方向。本科毕业后第一份工作是在网易杭州研究院做即时通讯，也就是老一辈网民熟悉的网易泡泡。当时我也不知道自己对什么感兴趣，于是采取了一个简单粗暴的策略：来者不拒，能干的我都干，再把不爱干的筛掉。

在网易泡泡，我先后做过新旧版本的Windows客户端、FreeBSD/Linux服务端以及旧版线上集群运维等等。后来这个项目中需要用到一些分布式存储的东西，我觉得很有意思。坦白说在此之前除了本科毕业的时候写毕设翻过一些文献材料，我还没有看过什么论文。于是我从Google Bigtable开始，顺着参考文献列表一路往下遍历，然后就一发不可收拾，包括后来去了百度也还在接着看。就这样看了两三年，大概1974年以后的分布式文献都过了一遍。

函数式编程其实也是在网易做即时通讯的时候接触到的。即时通讯有一个开放的协议叫XMPP，当时我想找一个开源的分布式XMPP服务器实现。找来找去，唯一一个比较靠谱的就是ejabberd。ejabberd是用Erlang写的，而Erlang是一个函数式特性很强的语言。但是当时并没有啃下去，

因为从小到大走的都是C/C++/Java这类命令式语言的路线，函数式语言对于当时的我来说实在是很不合胃口。

后来到了百度，2009年我发现Facebook Chat的服务端就是一套定制版的ejabberd。大概也就是从那时候开始，Erlang开始进入互联网行业，我想Facebook的这一动作应该算是个标志。Erlang诞生于80年代，早就已经成熟应用在电信行业里，最擅长处理的就是高并发、高可靠、分布式的场景。而早年互联网行业内这样的场景还并不广泛，于是直到现在大众才开始关注Erlang。那时候我正好也还在研究分布式系统，并且需要一套能够快速完成原型验证的工具。用C/C++的话，光是底层网络通讯等基础设施就已经够复杂了。于是我又把Erlang捡起来了。此后，从Erlang出发，业余又开始对函数式语言做了些研究。所以说，函数式编程和分布式的学习背景其实都是因为工作需要。

问：你是如何学习函数式编程的？

玩Erlang的时候，我开始对Erlang的很多特性感到好奇，包括匿名函数、GC、尾递归调用等等，这些特性在Python、Ruby等一些动态语言中也同样存在，但是我只知其然，而不知其所以然。我觉得好奇的是，第一、函数式语言跟普通的命令式语言的本质相比有什么优势；第二、这些特性背后的运行时机制到底是什么。那个时候我在百度已经开始做管理，基本脱离了一线开发，说实话少了挺多乐趣。我就决定用业余时间把函数式编程搞清楚。

我自己的习惯是每年都会做一个side project，我觉得要把函数式搞明白，最简单的办法就是自己做一个实验性的函数式语言实现，比如一个最简单的解释器什么的，也不求它能够多么高效、实用，只求把个中原理搞明白。既然是实验，目标语言当然越简单越好，于是选中了Scheme。Scheme的整个R5RS标准连目录带附录总共才50页，特别精简。

因为都是业余时间做，这个小项目断断续续地做了两年，期间用不同方法重写了若干遍。项目整个做完了之后，确实把动态类型函数式语言最基本的东西，从运行时模型到理论上的一些概念和原理都弄明白了。实际上就跟我之前研究的分布式系统一样，也是把主要文献都粗略扫了一遍。

问：听说你在百度有一段时间在做管理工作，为什么？

这个说来有趣，最早入行的时候，对管理有误解，本来是下定决心坚决不走管理路线的，因为当时觉得做经理这个事情特别无趣。百度内部各个部门有各自的TC（技术委员会），由部门内高级工程师组成，负责本部门的技术方向把控、技术评审、职称评定等工作。我当时在客户端软件部，起初部门很小，高级工程师也不多。由于当时处在扩张期，招了很多新人，TC的工作重点之一便是培训和技术氛围建设。我当时对这些事情比较活跃一些，发现新的、好玩的东西我比较喜欢拿出来跟大家讲，因为我觉得如果好东西光我自己会用是用不起来的，必须大家一起用，这个东西才能推得动。大概因为这个原因，我入职不到一年的时候，TC主席破格把我调到TC。由于TC站在一个相对全局的视角，并且经常需要跟其他经理打交道，相对于其他工程师，我得以更早地接触到一些管理相关的工作，逐渐体会到了这些工作的重要性和难度。

后来，客户端软件部内部整合，需要建立一个后端团队统一负责各产品的后端服务。由于一时没有合适的经理人选，TC主席对我说：“要不你先把这个队伍建起来，过个半年我们招到经理了就换你下来。”我心说，试试管理岗位也不错，就答应了。结果这一干就干了两年没换下来。这支队伍我从零建起来，到离职时已经有25人，并且保持了部门内最低的离职率。这两年里，我从我的团队和合作伙伴们那里学到了大量的非技术知识和技能，并完全改变了我以前对管理的幼稚看法。但尽管如此，我个人仍然对管理工作提不起兴趣。因此直到离职为止，也一直没有正式转到管理线上去。

问：后来找工作好像有些趣事？

是的，2012年下半年从百度离职，打算出国。准备了半年后来去Google、Amazon和Facebook面试，没有成功，所以写了那篇《加州求职记》。那篇文章两三天被刷上万，大概是因为很少有人写面试失败的面经，所以大家觉得特别亲切吧……因为面试过程不顺，刷题又很无聊，于是跟着Coursera上Martin Odersky开设的Functional Programming Principles in Scala课程学了Scala。巧的是，13年六月份意外得知Intel中国研究院在招会Scala的实习生做Spark。那时候Scala还在国内还没有多少动静，不要说会，听说过Scala的学生都不多。当时我对Spark还一无所知，简单看了一下，发现这个项目很好地结合了分布式系统和函数式语言这两个我最为感兴趣的技术方向。耐不住在家赋闲刷题的无聊，我最终以contractor身份加入了Intel，并和Intel中国研究院吴甘沙和杨栋带领的团队一起做了大半年Hadoop和Spark相关的研究。

在做Spark之前，我从来没有做过大数据分析相关的方向。以前研究分布式系统，主要集中在分布式存储方面，和大数据分析差别还是蛮大的。在Intel期间，能在大数据分析方面快速入门，要特别感谢甘沙。他技术嗅觉极其敏锐，精力极其充沛，而且善于表达。在短时间内就将这一领域的历史、现状、机遇高屋建瓴地描绘给了整个团队。更妙的是，我们团队成员的能力非常互补，既有做操作系统的，又有做机器学习的，也有我这样工作多年有实际系统经验的。这半年多，算是我近几年学习速度最快，长进最大的一段时间。

问：你是如何加入Databricks的？

刚开始做Spark相关的工作时，Databricks还没有成立。当时一方面是对Spark感兴趣，一方面也是为14年初在湾区找工作再打些基础。（我在加入Intel之初就明确说过打算出国，也正因此一直是contractor。）

在Intel期间，我先后向Spark贡献了一些补丁，对整体系统比较熟悉之后，就挑了一件相对完整的工作。我基于两年多前Ankur Dave（后来GraphX的作者）的Arthur项目开发了用于分析和调试分布式Spark作业的Spark Replay Debugger（可惜后来并没有被合并到主线）。这个时候，Databricks成立的消息公布，对我产生了巨大的吸引力。正好公司创始人之一辛湜博士回国参加China Hadoop Summit。以Spark Replay Debugger为契机，在甘沙的引荐下，辛湜为我安排了面试。面试过程种种略去不表，最终总算幸运通过。

问：现在你在Databricks工作，但是还没有去美国那边，你现在是怎么和团队合作的？

因为Apache Spark本身是个开源项目，大家直接通过GitHub和邮件列表就可以比较好地交互。同时我们每周有一到两次远程会议。还好时差不算太严重，北京早上8点的时候是加州那边下午4点。一般事务主要通过邮件，急事则通过Google Hangout或Skype联系。由于开源项目的公开透明性质，沟通问题并不太严重。

大数据处理的所以然

> JVM的设计使得在JVM之上实现函数式变成一种戴着枷锁跳舞的艺术，Clojure、Scala都因为JVM的限制而不得不在语言层面作出了一些丑陋的妥协。

问：Databricks上面有一个对Spark的介绍，其中提到大数据时代的计算平台必须要开源，你能稍微解释一下吗？

我自己的理解是这样的，大数据分析复杂性相对来说是很高的，从用户的角度来看，复杂性突出体现在两个方面：

第一、很多情况下，采用大数据系统的组织内部原本就存在一些重要的遗留系统，难免面临对内外系统进行扩展和/或改造，这时也会产生很多复杂的技术性和非技术性问题。采用开源系统时，我们可以有效地对系统进行定制。

第二、大数据系统一般来说代码量比较大，动辄十数万、数十万行代码，难免会存在各式各样的缺陷；分析、调试、修复这些系统中的缺陷同样是一个复杂的工作。大数据场景下，很多问题必须在足够的体量下才能够重现，难以构造最小的问题重现场景，这使得现场调试定位显得尤为重要。而开源软件大大降低了现场调试定位的难度。

问：有人说MapReduce是一个巨大的倒退，说它倒退回了现代数据管理系统发明以前的时代，你怎么看？

这个说法是数据库David DeWitt和Michael Stonebraker在2008年在发表的MapReduce: A major step backwards一文中提出的。这里的“现代数据管理系统”指的主要是关系数据库。文章将MapReduce与关系数据库做了一个详细的比较，得出了MapReduce在功能、性能、易用性等方面相对于关系数据库都是巨大倒退的结论，并且抨击MapReduce背后的想法“毫无创新可言”，甚至还质疑了MapReduce的可伸缩性（scalability）。这篇文章本身发表之后饱受争议。评论中有大量针锋相对的辩论。我最为赞同的两点相反意见包括：

1. MapReduce和数据库面临的场景是很不一样的。传统数据库处理的是规整的结构化数据集，所有数据都符合预先设定的规则（schema）；而Google最初MapReduce处理的是不规则的半结构化数据——整个互联网。

2. MapReduce的可伸缩性是铁板钉钉的不争事实。MapReduce论文中提到，Google利用MapReduce每天处理20PB的数据，在当时这时任何传统关系型数据库也做不到的记录。

MapReduce提供的抽象固然十分简单，缺少高层抽象和更加易用的接口，但在当时它确实有效地解决了最迫切的问题——可伸缩性。这就好比乘坐火车、汽车、轮船时我们可以用手机，但坐飞机的时候就不行，我们能说飞机相对于火车等传统交通工具是倒退吗？当技术发展还不足够成熟时，为了达到某一重要目标，往往不得不舍弃另外一些东西。

但从现在来看，MapReduce确实已经落伍了。DeWitt和Stonebraker提出的MapReduce的诸多缺点也逐渐被后来人所弥补。自Hadoop提供了开源的MapReduce之后，整个大数据产业蓬勃发展。缺少schema、逻辑层和执行层耦合紧密的问题，已经由各种SQL on Hadoop系统解决；做复杂计算时MapReduce涉及的shuffle过多，磁盘IO密集的问题由Tez等DAG计算引擎缓解；MapReduce所不擅长的流式数据处理，也由MillWheel、Storm等系统挑起了大梁。而Spark作为MapReduce的超集，更是可以在单一软件栈上搭建一体化的大数据流水线，同时完成批处理、流处理、关系查询、迭代计算、图计算等多种计算范式而无须维护多套系统。

所以我认为，说MapReduce相对于现代数据管理系统是历史倒退是不公平的，因为它在当时提供了关系数据库无法提供的价值——足以处理整个互联网的超高可伸缩性。但在MapReduce论文发表十多年后的今天，它也确实该退休了。

问：JVM能在大数据领域里面流行，其背后的原因是什么？

JVM在大数据领域的流行，与Hadoop脱不开干系。Hadoop本身的成功，与Java的低入门门槛、高开发效率（相对于C++而已）应该有相当大的关系。在HDFS、Hadoop MapReduce流行之后，为了能与Hadoop无缝互操作，后续的一些大数据系统自然而然地也选择了Java。近年来，虽然Java在语言层面发展缓慢，越来越被诟病，但Clojure、Scala等JVM上的新语言却层出不穷，这又进一步激发了人们继续以JVM为平台

搭建新兴大数据系统的热情。Hadoop生态圈越做越大，而试图加入这个生态圈的新系统若想无缝利用现有的遗产，就只能选择JVM。于是雪球越滚越大，进而令JVM几乎垄断了整个大数据行业。

然而我想说的是，除了Hadoop的原因之外，还有更多人的因素掺合在内。JVM之所以能够流行，另一个原因是JVM之上最初的语言Java，对于几十年来被命令式语言和OOP所统治的工业界来说非常直观和简单。这种直观和简单最直接的体现就是初学者的心智负担很低，让人觉得“理所当然”，这使得人们自然而然地更倾向于使用这些这类语言，促使其流行。然而，直观的事物却未必是正确，反之正确的事物也未必就直观——现实世界中客观存在的波粒二象性、量子纠缠、黑洞等事物都是典型的反直觉的存在。

在这个问题上，我禁不住想多说两句题外话。越来越多的实践表明，函数式编程更加契合大数据场景。对不变性的强调使得分布式一致性、容错、并行/并发都变得更为简单和高效。函数式语言的各种基本算子天生适合数据并行处理。GFS/HDFS对数据不变性的强调以及MapReduce的成功，本身就是函数式编程范式适合大数据处理的绝佳证据。实际上不仅仅是大数据领域，在很多方面函数式语言都有着优异的表现。虽然JVM上Clojure和Scala逐渐壮大，渣打银行也打出了Haskell程序员的招聘帖，但为什么函数式语言却仍然如此小众呢？更进一步，我发现历史上以及当前很多流行的工具，其实未必是该领域内最优的选项。这是为什么呢？

这两年时不时会思考这个问题，得出了一些结论：

1. 对于大部分人来说，甚至包括很多非常聪明的人，他们并不乐于去改变自己的思维方式——对自己的大脑重新“编程”实在太难了。

2. 流行的工具大致可以分为两类。第一类可以帮助人们以熟悉、直观、符合大众思维

的方式便捷地解决眼前的问题，骚到痒处。第二类要么能够将不可能变为可能，要么在应用效果上远远超越同类竞争者，戳到痛处。

然而长远来看，第一类解决方案却未必正确，甚至往往牺牲了其他更为重要的系统属性——例如一些动态语言中的monkey patching，看似方便，却打破了对象运行时布局不变的假设，结果既牺牲了运行时性能，又破坏了系统的静态分析潜力，使得大型项目重构困难重重。这类技术和工具，即便会带来一定的心智负担，要求大众转变思维方式，大家也心甘情愿。例如当年横空出世时的MapReduce，以及现在的Spark。这类技术和工具，更有益于健康的思维方式和方法论的推广。正如Dijkstra在1975年时所说：

> The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities.

对工具的选择，既可以解放我们的思想，也可以禁锢我们的思想。

从这个意义上将，作为一个为命令式OOP语言设计的JVM，在大数据领域已经开始显得格格不入了。JVM的设计使得在JVM之上实现函数式变成一种戴着枷锁跳舞的艺术，Clojure、Scala都因为JVM的限制而不得不在语言层面作出了一些丑陋的妥协。Erlang、Haskell等强调不变性的单次赋值语言所特有的高效GC和并行优势在JVM上也不可能实现。Full GC带来的停顿恐怕是每个与JVM打交道的工程师都饱尝过的噩梦。

也正是因为这些限制，JVM上目前还没有一个特别令我满意的语言实现，Scala算是JVM上最为接近的一个。脱离JVM，Rust也是我特别感兴趣的一门语言。个人特别期待能出现一门既带有静态强类型纯函数式语言特质，又带有与Erlang类似的运行时级别的软实时、高并发支持，同时还能尽量接近native执行性能的语言。

问：如果你的想法这么具体的话，是不是可以自己去写了？

确实时不时会有这种想法。但是实际上去做一个语言的话，要做的并不仅仅是一个编译器，还有很多和整个生态发展相关的因素需要考虑，其工作量是巨大的，并且我自认现在也远没有相应的能力，耽于空想罢了。当然，我非常期望今后有精力的时候，能够去做这样的研究。哪怕不是做一个实用的语言，仅仅试做一个原型，去做一些探索。

进击的Spark

> 他们试用过Spark SQL之后普遍反馈：“再也不想回到Shark了。”

问：Spark及其类似产品取代MapReduce是不是一个必然？

我觉得现在已经有比较明显的趋势了。当然，Spark跟Hadoop生态之间还是一个补充关系，但是Spark作为一个计算引擎，确实已经是在革MapReduce的命了，实际上它也在革很多基于MapReduce搭建的其他计算引擎的命。这也是为什么现在Mahout、Hive、Pig等一些原先基于Hadoop MapReduce的系统都正在逐渐迁移到Spark上来。

和Spark类似的好像还有一些模型，比如你刚才提到的Tez，还有一个就是在2014 Daytona GraySort大赛中和Spark并列第一的Themis，能不能比较一下？

Themis系统实际上是一个特化系统，而不是一个通用的计算引擎。这也是Spark在排序比赛当中获奖特别令人振奋的一点，因为Spark并没有专门为这样的一个排序竞赛去做特殊的优化。Spark做的所有的优化实

际上都是对整体执行效率以及稳定性的提升，搭建在Spark core之上的Spark Streaming、Spark SQL、MLlib、GraphX等所有组件都可以直接获益。Tez虽然也是DAG计算引擎，从目前来看整体的执行效率还在Spark之下。Hive 0.14集成Tez引擎后，单从执行效率上讲，跟Spark SQL相比并没有优势。

问：Databricks提供技术支持的服务吗？

虽然去年公司人数翻了一倍多，但Databricks目前总共也只有不到50人。直接面向终端用户提供Spark技术支持非常消耗人力，这对于Databricks目前的规模来说是不现实的，我们目前更希望将人力投入到研发和Spark的推广上。目前Databricks和包括Coursera、Hortonworks、Pivotal、MapR在内的所有Hadoop发行商都建立和合作关系，终端用户可以从这些厂商处获得专业的技术支持服务。

问：Spark现在在国内和国外的应用情况如何？国内现在有腾讯、百度这样的企业用起来了，TDW应该算是国内Spark比较典型的成功案例，会不会影响大环境？

肯定会带来积极的影响。大公司的规模效应可以起到一个定心丸的作用，加强了很多观望中的中小公司的信心。据我所知，国内的很多小公司尤其很多创业公司也都在用Spark。实际上，相对于大公司来讲，Spark对于这些创业公司来说更加有吸引力。原因很简单，对于大公司来讲，它们在引入Spark之前就已经有了自己的专有系统或者其他开源系统。引入Spark之后，还有对接、替换、改造、扩展的成本。

此外，Spark相对于其他大数据系统的一个巨大优势在于可以让开发者在一个系统内实现多种计算范式，从而无缝地在一套系统上搭建一个完整的一体化流水线。对于创业公司来说，无需部署多套系统，只需要一个stack，就可以完成各种类型的大数据分析需求，这样就节省了大量

的学习成本、开发成本、部署成本和运维成本。对于大公司来讲，由于现有遗留系统的存在，Spark一体化流水线的优势反而不太容易发挥出来，他们只会用Spark来做现有系统做不到或做不好的事情。比如腾讯和阿里更多地是用Spark去做机器学习，而不太可能将从数据清洗开始的一整条数据流水线都迁移到Spark上来。百度之所以前两年在Spark社区里声音不多，也是因为他们在做内部系统的整合和消化。现在百度自己的BMR服务已经出来了，说明内部的整合和消化已经基本完毕了。

问：你现在的工作重点还是在Spark SQL上？你说过希望外部数据源API设计能有更多进展，现在怎么样了？

我的工作重点目前的确还在Spark SQL上。在Spark1.2中，外部数据源API最重要的一部分已经发布出来了。目前开发者已经可以基于这个API去开发各种各样的connector，然后把外部的数据源对接到Spark上。但是当前的API中我们只提供了查询入口，还没有办法把我们处理完毕的数据写回到外部系统当中去。在1.3和1.4中会提供相应的支持。

另外在外部数据源的集成上，我们目前的重点主要集中在高度可扩展的API上，同时也会实现一些最为常用的数据源，例如JSON、Parquet、JDBC。此外社区中也已经出现了一些第三方数据源的实现。

问：Shark现在已经没有在开发了，把Shark用户迁移到Spark SQL上，这个过程有没有困难？

我曾和国内的一些用户聊过这个问题，包括一些曾经在Shark上用了蛮久的用户。他们试用过Spark SQL之后普遍反馈："再也不想回到Shark了。"

这里边有好几个原因。

第一、Shark是直接从Hive改造过来的，所以很大程度上受到了Hive的钳制。Shark中从查询语句解析成抽象语法树，到抽象语法树翻译成逻辑执行计划，再到逻辑执行计划优化，都是利用Hive完成的，而Hive的查询优化器是针对MapReduce设计的，这使我们受到很大的限制。而在Spark SQL中，我们拿到抽象语法树之后，除了通过Hive的Metastore获取表的元信息以外，后续的执行路径跟Hive就没有什么关系了，查询计划的生成和优化全部都由我们自己接管，这样就可以更加充分地发挥Spark的优势，进一步提升执行效率。

第二、Spark SQL不仅仅提供了关系查询，而且还可以借助SchemaRDD（1.3以后升级为DataFrame）这一统一抽象实现与其他Spark组件的无缝集成，并可以融合多种数据源完成更为丰富的计算。

第三、抛开上述的各种相对于Shark的优势，Spark SQL本身就是一个十分令人兴奋的项目。Spark SQL的核心，实际上是一个用一种函数式语言（Scala）实现的另一种函数式语言（SQL）的编译器。Spark SQL中原创的Catalyst框架大量应用了Scala的函数式特性，令开发者得以以极为简洁明了的声明式代码实现各种查询优化策略，从而大大降低了查询优化器这一数据库中的硬骨头的开发难度和维护成本。

所有这些因素加在一起，使得Spark SQL成为一个显著优于Shark的方案，因此大家还是相对愉快地和Shark告别了。

现在整个Spark SQL社区里面有非常多中国的用户，也有非常多中国开发者。我曾经做过一个统计，发现Spark SQL的贡献者中有一半左右是华人。

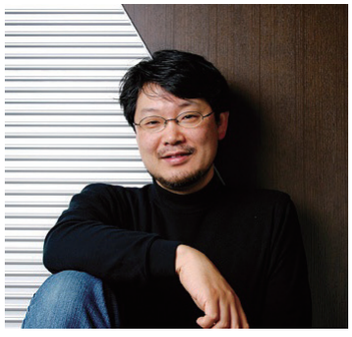
问：你觉得有这么多中国开发者的原因是什么？

据我了解到的情况，国内的很多企业，尤其是电信行业的企业，过往大

量依赖Oracle和DB2，业务紧密依赖SQL。而在近几年的去IOE潮流中，又偏偏缺乏高效的能够处理大数据的SQL执行引擎。这个时候，Shark和Spark SQL的出现给大家带来了较好的选择。以此为契机，大量的开发者被吸引到了Spark SQL社区。此外，Shark的作者辛湜博士本人就是中国人，这点不知道是不是也有关系 :-)

松本行弘：

我为什么要开发新语言 Streem



作者 / 松本行弘

Ruby 语言发明者，亦是亚洲首屈一指的编程语言发明者。现兼任网络应用通信研究所（NaCl）研究员、乐天技术研究所研究员、Heroku 首席架构师等。昵称“Matz”。讨厌东京，喜欢温泉。

随着多核 CPU 的普及，shell 脚本的（一部分）价值也在逐渐被我们重新认识。shell 脚本的基本计算模型是基于管道来连接多个进程。如果操作系统支持多核的话，则各进程会被分配到不同的 CPU 上去执行，这样就可以充分发挥多核 CPU 的优势。同时这也证明了一点，那就是只要选择合适的计算模型，就能非常容易地实现并发执行。

在实际的业务系统中，我也听说有人采用 shell 脚本来进行处理。虽说是用 shell 脚本进行信息的筛选和加工，但是和传统的软件开发模式相比，它有着成本低、灵活性高等优点。

shell 脚本已经有些力不从心

但也并不能说 shell 脚本有多么理想，实际上它也有它的局限性。

比如，创建 OS 进程的成本非常高，如果需要使用 shell 脚本创建大量轻量进程的话，那么在性能上将会非常不利。

还有另外一种成本，由于连接进程的管道只能发送字节数组的数据，所以发送方需要先将数据转换为字节数组，接收方则需要将字节数组还原。比如很多时候我们都会使用以逗号分隔的 CSV（Comma Separated Values）格式或表示 JavaScript 对象的 JSON（JavaScript Object

译/刘斌

原文刊载于《日经Linux》2015/01号中（©日经BP社2015）。原文题为《跟Matz边做边学编程语言：21世纪的并发编程语言》。

Notation）格式，将数据从这些格式转换为字节数组，或者对字节数组进行解析并还原，这样做的成本是非常高的。

在进行大数据处理、高性能计算等时，我们多会选择使用多核CPU。因此，数据转换或创建进程所花费的成本是不可忽视的。这可以说是shell脚本的一个缺陷。

更进一步来说，构成管道的进程（process）所执行的命令（command），可能并不是由同一个开发者所开发的，这些命令的参数设置方法等往往并不统一，因此要想熟练使用这些命令，难度会有所增加。

21世纪的shell脚本

这样说来，如果能将shell脚本的优点，和通用编程语言的优点结合起来的话，应该就可以创造出一门非常强大的语言。

首先我们来看看这门强大的语言都需要满足哪些必要条件。

第1个条件是可以进行轻量的并发。由于不管是OS级别的进程还是线程，创建成本都很高，因此我们应该尽量避免去使用它们。比较现实的方式是在一个OS的进程中，预先生成与CPU的核数（+α）相同个数的线程，让它们轮番去执行各种操作请求。采用这种实现方式的典型语言包括Erlang和Go。在本文中，我们将相当于Erlang中的“process”、Go中的“goroutine”的概念称为“任务”（task）。

第2个条件就是解决并发执行时的竞争条件。具体来说就是“状态”的排除。也就是说，如果变量或者属性的值发生变化，就会产生一个新的状态，这也带来了因执行时机（timing）不同而产生问题的危险。所以需要将所有数据都设为不可变（immutable），这样就可以避免因执行时机而出现的缺陷。

第3个条件是计算模型。线程模型虽然应用领域非常广泛，但自由程度也很高，因此程序可能会变得难以掌控。于是我们可以参考shell的执行模型，引入一个抽象度非常高的并发计算模型。抽象度高了，反过来表现的自由度就会降低，所以在编写代码的时候就要下一番功夫。而另一方面，我们的程序也会变得非常容易调试。

新语言Streem

于是我就开始设计一门满足上述条件的新语言。由于是以流（Stream）为计算模型的语言，因此我们就将它命名为“Streem”。

首先我们来看看它的语法。由于它是基于shell的，因此也没有什么特别的语法。它的基本语法如下所示。

```
表达式1 | 表达式2 | ...
```

若要采用这种语法来实现一个从标准输入读取数据，然后输出到标准输出，功能类似于cat命令的程序，只需编写如下代码即可。

```
STDIN | STOUT
```

就是如此简单。STDIN和STDOUT是用常量表示标准输入输出的对象。Streem程序中的STDIN会从标准输入中一行一行地读取（字符串）数据，并将这一行行的数据传递给其他表达式，就像“流”一样。像这样用来表示数据的流动的对象就称为“流”（对象）。STDOUT则正相反，它是一个接收字符串参数并输出到外部（标准输出）的流。如果想读取指定文件名的文件内容的话，可以使用如下代码。

```
read(path)
```

如果是写入到指定文件，则可以使用如下代码。

```
write(path)
```

这两个方法都会返回用来读取或者写入的流对象。

表达式

Streem的表达式包括常量、变量引用、函数调用、数组表达式、Map表达式、函数表达式、运算符表达式以及if表达式。它们的语法如表1所示。如果你有其他语言的编程经验的话，这些内容应该都很容易理解。

表1　Streem的表达式

类　型	语　法	示　例
字符串常量（字面值）	"字符串"	"foobar"
数值常量	数值表现形式	123
符号（Symbol）常量	:标识符	:Foo
变量引用	标识符	FooBar
函数调用	标识符(参数...)	square(2)
方法调用	表达式.标识符(参数...)	ary.push(2)
数组表达式	[表达式, ...]	[1,2,3]
Map表达式	[表达式:表达式, ...]	[1:"2", 3:"4"], [:]（空Map）
函数表达式	{| 变量.. | 语句 }	{|x| x + 1}
运算符表达式	表达式 运算符 表达式	1 + 1
if表达式	if 表达式 {语句 ...} else { 语句 ...}	if true {1} else {2}

赋值

Streem中的赋值有两种方式。第一种是和其他语言类似的使用“=”的方式。

```
ONE = 1
```

还有一种就是使用“->”的反方向赋值的方法。如果把上面采用等号的赋值语句改为使用“->”的话，代码将如下所示。

```
1 -> ONE
```

这种赋值方式非常适合在将管道执行结果存放到变量时使用，这样代码的顺序就能和程序的执行流程保持一致，非常方便。

不管采取上面哪种赋值方式，为了避免变量状态的改变，都需要遵循如下规则。

- 规则 1：不能给同一个变量进行多次赋值。对一个变量的赋值在其作用域之内只能进行一次。

- 规则 2：仅在交互执行环境的顶层作用域（top level）下，才允许对一个变量进行重复赋值。不过，你可以将这看作是对变量名相同的不同变量进行的赋值。

语句的组合

Streem 支持将多条表达式语句并列编写。语句之间用分号（;）或者换行来分割。也可以认为这些语句会按照编码的顺序来执行。如果这些语句之间没有依赖关系的话，在实际执行的时候也可能会并发执行。

Streem 程序示例

接下来让我们来看一些 Streem 程序的具体例子。

前面我们看到了一个实现类似于 cat 命令的例子，下面我们来看一个稍微有点不同的例子。这里我们将使用 Streem 来实现经常被拿来举例的 FizzBuzz 游戏（图 1）。这个游戏要求玩家从 1 开始输入数字，当这个

数字能被3整除的时候，会输出“Fizz”；当能被5整除的时候，会输出“Buzz”；当能同时被3和5整除的时候，则会输出“FizzBuzz”。

```
seq(100) | { |x|
  if x % 15 == 0 {
    "FizzBuzz"
  }
  else if x % 3 == 0 {
    "Fizz"
  }
  else if x % 5 == 0 {
    "Buzz"
  }
  else {
    x
  }
} | STOUT
```

图1　Streem版FizzBuzz

seq函数用来生成一个从1到指定参数的整数数列。如果将该数列连接到管道上的话，则该数列会将各元素的值按顺序传递给管道。STDOUT则将接收到的数列的值进行输出。

从上面的例子我们可以看出，Streem的管道表述方式直接体现了要干什么，是不是更为直截了当呢?

1对1、1对n、n对m

通过图1的例子，我们已经知道了使用Streem可以非常简单地完成诸如对数值序列进行处理并输出的程序。然而现实中的程序并不完全都是对这种1对1关系的数据进行处理。比如类似于grep（单词搜索）这样“查找所有满足指定条件”的类型，以及类似于wc（统计单词数量）这样对数据进行聚合计算的类型。

Streem也支持这种应用场景，并提供了一些关键字来进行这类操作。

在一次执行需要返回多个值的时候，可以使用emit。如果给它传递多个（参数）值的话，那么它也会返回多个值。也就是说，

```
emit 1, 2
```

就相当于下面这行代码。

```
emit 1; emit 2
```

此外，如果在数组前面加上“*”的话，就表示要返回这个数组的所有元素。比如，

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

就相当于如下代码。

```
emit 1; emit 2; emit 3
```

图2是一个使用emit的例子。这个程序会将从1到100之间的整数每个都打印两次。

```
# 将从1到100的整数分别打印两次
seq(100) | {|x| emit x, x} | STDOUT
```

图2　使用emit的例子

return用来终止函数的执行并返回值。return可以返回多个值，这时候它就相当于对多个值进行了emit操作。有一点前面我们没有提到，那就是如果一个函数主体只有一个表达式的话，那么即使不使用return，这个表达式的执行结果也会作为函数的返回值。

使用emit和return的话，就可以产生比输入值个数更多的返回值。与之相反，如果我们想生成少于输入值个数的返回值的话，则可以使用skip函数。skip用来终止当前函数的执行，但是并不产生任何返回值。图3是一个使用skip的例子，该程序用来筛选出1到100之间的偶数。

```
#  skip奇数，选择偶数
seq(100) | {|x| if x % 2 == 1 {skip}; x} | STDOUT
```

图3　使用skip的例子

不可变性（immutable）

前面我们已经说过，在Streem中，为了避免竞争条件的出现，所有的数据结构都是不可变的。数组和Map（类似于Ruby中的Hash）类型的变量也是不可变的。向这些结构的数据添加新元素的时候，并不是直接修改已有的数据，而是在原数据的基础上添加新元素来创建新的数据（图4）。

```
a = [1, 2, 3, 4] # a是一个拥有4个元素的数组
b = a.push(5)    # b是在a之后添加了5的数组
```

图4　修改immutable数据

在一般的面向对象编程语言中，对象的属性（实例的变量）都是可以修改的，而在Streem中，这种操作是被禁止的，这需要注意一下。从这一点上来说，Streem非常像函数式编程语言。

统计单词出现次数

接着我们再看另一个Streem程序的例子。这里我们选择了在介绍MapReduce时经常使用的一个例子——统计单词出现次数。下面我们用Streem来实现一下（图5）。

```
STDIN | { |line|
    return *line.split
} | reduce([:]) { |map, word|   # [:]是一个空Map
    map.set(word, map.get(word,0) + 1)
} | STDOUT
```

图5　使用Streem统计单词出现次数

首先我们对图5的程序中新出现的语法进行说明。在调用reduce函数的地方，我们看到了类似于Ruby中的Block的语句。这是Streem语言中的一个语法糖，如果函数的参数列表后面是一个函数表达式的话，那么这个函数表达式就会被视为该函数的参数列表的最后一个元素。也就是说表达式

```
reduce(0) {|x, y| x + y }
```

是下面的表达式的另一种写法。

```
reduce(0, {|x, y| x + y })
```

这也是Streem为了能在普通函数调用中将类似于Ruby中的Block变量作为参数而做出的努力，而不必使用&block的方式。

如果我们看一下图5中程序的实际执行情况，就会看到具体流程是首先从STDIN中一行行地读取数据，用split进行单词分割，再通过reduce函数来统计各个单词出现的次数，并将结果存放到Map中去。如果作为key的单词不存在的话，map.get就会返回第二个参数作为默认值（这里是0），这样就可以通过map.get得到该单词的出现次数。map.set用来更新单词出现的次数，并创建一个新的Map。因为每次更新单词出现次数时都会创建一个新的Map，所以看上去有点浪费系统资源，但实际上我们无需为此担心，完全可以将这些问题交给垃圾收集器或者系统运行时环境的内部实现。实际上Clojure及Haskell等很多函数式编程语言也都采用了相同的策略。

最后，程序将生成的Map和STDIN通过管道连接起来，将Map中的键值对打印出来，显示各个单词及其出现次数。这个例子中我们并没有做其他的额外处理，需要的话你可以增加一个管道以在输出结果之前对单词进行排序等工作。

Socket编程

Unix的Socket也是基于流而设计的，Streem当然也支持Socket操作。图6的程序是一个最简单的使用了Socket的网络Echo服务器（将接收到的数据原封不动地返回给客户端）。

```
# 在8007端口提供服务
tcp_server(8007) | { |s|
  s | s     # 直接将输入数据作为输出返回给客户端
}
```

图6　Echo服务器程序

代码是不是非常简单？如果程序的应用场景非常匹配流模型，那么采用Streem语言的话，编码工作将会非常简单。

我们还是来解析一下这段代码吧。tcp_server会在参数指定的端口上打开一个服务器端Socket进行监听，等待客户端的连接。在Streem中，服务器端Socket是客户端Socket的流对象。

客户端Socket是客户端的输入和输出的流，所以如下代码

```
s | s
```

的实际功能就是“原封不动地将客户端输入直接返回给客户端”。如果需要对输入内容进行加工处理的话，只需要在管道之间加入一个进行数据处理的流就可以了。

管道业务

目前为止我们看到的管道的组成都是如下方式。

```
表达式1 | 表达式2 ... | 表达式n
```

表达式1是一个产生值的流（产生器，generator），表达式2及后续表达式都是对值进行变换、处理的流（过滤器，filter），管道最后的表达式n则可以认为是输出目的地（消费者，consumer）。

产生器有很多种，比如像STDIN这样的从外部获取输入的流，以及像seq()这样的通过计算产生值序列的函数。如果将产生器替换为一个函数表达式的话，那么这个函数表达式就成为了一个通过return或emit来产生值的产生器。

过滤器在大多数情况下都是一个函数，通过参数接收前面的流传过来的值，再通过emit或return将值传递给下一个流。

最后的消费者只会接收值，是一个不会emit值的流。

Streem程序的基本结构就是像这样将流通过管道串联起来，从产生器开始对数据进行流式处理。也许我们也可以称之为“管道业务”。虽说这种计算模型并不是万能的，但是它具有抽象程度高、容易理解、支持并发编程等优点。有时候我们并不需要做到100%的功能，而是专注于那重要的80%就可以了。

但是，并不是说所有程序中的数据流都只有一种（即一条管线），因此完全放弃这样的程序的做法也有点过头。我们需要更加复杂的管线配置。具体来说，我们还需要将多个流合并（merge）为一个流，以及从一个流派生出多条通知（广播）这两种类型的结构。

更进一步来说，在将流进行连接的时候，如果有一个能指定缓冲区大小的方法的话，是不是更好呢？

合并管道

到这里为止我们看到的例子中数据流都只有一条管线，这在简单的应用场景下倒没什么问题，但是这种方式并不能解决现实中的所有问题。

有时候我们可能需要将多个管道合并为一个，或者对一条管道进行分割操作。管道的合并可以使用“&”操作符。

```
管道1 & 管道2
```

通过使用“&”操作符，就能将管道1和管道2的值合并成一个数组，并创建一个新的管道。合并后的新管道在任意一个原管道（这里为管道1和管道2）终止的时候都会同时终止。比如本文前面的cat的例子，我们如果想像cat -n一样同时输出行号的话，可以使用图9中的代码。

```
seq() & STDIN | STDOUT
```

图7　cat -n的实现代码

由于“&”运算符的优先级高于“|”，所以下面的代码

```
a & b | c
```

会被解释为

```
(a & b) | c
```

当省略seq()的参数的时候，该函数会从1开始进行无限循环。由于STDIN是从标准输入一行一行地读取数据并写入到管道中的，因此管道合并的结果如下所示。

```
[行号，行内容]
```

将这个流合并后得到的新数组写入到STDOUT（标准输出），就实现了带行号的cat。从实用角度来讲，也许我们还需要对行号进行显示位数的格式化等工作，不过这也只需要你在STDOUT之前加入一个用来格式化的管道操作就可以了[1]。

[1] 由于Streem还在开发之中，因此还没有格式化相关的规范。

通道缓冲（channel buffering）

如果管道中最后一个流不是消费者的话，则会返回一个被称为“通道”（channel）的对象。比如下面的代码。

```
seq() & STDIN -> sequence
```

这里的sequence就是一个用来表现合并了seq()产生的数列和从STDIN读取的输入内容的通道。我们可以将管道理解为使用通道将进行流处理的task串联起来的结构。

当然各个流中对数据进行处理的速度都有所不同。如果前面的流中数据产生速度太快的话，就会将数据堆积到通道中，进而导致占用大量内存。反过来说，如果通道中没有任何缓存数据的话，则会增加前面处理的等待时间，从而降低整体效率。

所以，Streem会将适当数量（当然这个数量既不多也不少最理想了）的通道放到缓冲区中。但是真正合适的缓冲区大小则是由程序来决定的，我们不能进行准确的预测。从性能的角度来讲，有时需要根据实际情况来手动设置这个缓冲区的大小。这时候我们可以使用chan()这个非常方便的函数。

chan()函数用来显式地创建通道对象。管道运算符“|”的右边如果是通道对象的话，则该通道就会直接作为输出目的地。另外你也可以为chan()指定一个整数型的参数，来设置缓冲区的大小。也就是说，如果我们想在图9的程序中将缓冲区大小显式地设置为3的话，代码就会变为图10那样。

```
seq() & STDIN | chan(3) | STDOUT
```

图8　指定了缓冲区大小的cat -n

如果将缓冲区大小设置为0的话，那么在一个通道对象被创建之后，直到其被消费掉之前，流会进行等待，这样管道就会以前后交互的方式来运行。这在单核CPU环境下也许会非常实用。

广播

在聊天类的应用程序中，一个人发送的消息要被广播给所有参与聊天的成员。通道也可以应用在这种场景下。如果将通过chan()创建的通道连接到多个流的话，那么作为输入发送给该通道的值就会被广播给所有和其连接的流。

如果我们将图6中的Echo服务器修改为聊天服务器，将接收到的消息发送给所有参与者，则代码如图9所示。

```
broadcast = chan()
# 打开8008端口上的服务
tcp_server(8008) | { |s|
  broadcast | s     # 返回参与者的消息
  s | broadcast     # 将消息发送给所有参与者
}
```

图9　Chat服务器

聪明的你也许已经发现了，广播通道是具有状态的。也就是说，连接到broadcast的流作为消息接收方，是会被保存到broadcast中的。另外，作为输出目标的流如果关闭了的话，或者通过disconnect方法被显式地断开连接的话，则该流就不再是输出目标了。immutable是基本的Streem，但是为了编写容易理解的程序，有时候我们需要牺牲一点纯粹性。当然，由于broadcast的状态变化在Streem内部实现了互斥操作，因此即使在并行环境下运行也不会有问题。

总结

我们围绕管道计算模型设计了的新语言Streem。如果是非常适合流处理的程序的话，写起来将简单得让人吃惊。

实际上Streem语言刚开始设计没多久，在达到实用的程度之前，还有许多需要考虑的东西。比如如何进行异常处理、如何支持用户自定义流、类似于对象的概念该如何定义等问题。随着软件规模变得越来越大，编程语言不得不考虑的问题也会越来越多。

“这种语言不能用来编写大型软件项目”，这是编程语言设计者经常使用

的“借口”。但是，只要这种语言还不是一无是处，还没有什么证据能表明这种借口会有什么实际作用。

下次我们将会对 Streem 的设计进行更深入的讲解，同时也会涉及一些具体的实现细节。 ■

阅读原文：松本行弘：我为什么要开发新语言 Streem

践　行

老码农的技术理想

作者 / 徐飞

苏宁云商前端架构师，长期致力于前端的高效开发，熟悉富因特网应用的各种业务场景，了解各类前端MV*框架的实现原理和优缺点，尤其对AngularJS有较深入的研究。微博@民工精髓V，图灵社区ID: 民工精髓

小时候，老师问我，你的理想是什么？我不假思索说是工程师，于是长大之后果然成了工程师。

工作这么多年，一直在思考工程师这三个字的意义，终于有一天恍然大悟，原来就是：用技术手段改进世界。

那么，在软件方面，目前的世界有哪些问题需要解决呢？有这么一些问题可以思考：

- 现在整个世界的信息化程度是偏高还是偏低？
- 程序员的人数够用吗？
- 软件行业的生产力是偏高还是偏低？
- 大部分软件系统都可靠吗？

我想说说自己对这几个问题的理解。

虽然现在我们的生活与十年前相比，已经发生了巨大变化，比如智能手持设备已经非常普及，可穿戴设备也在蓬勃发展。十年前我们用手机收发短信或者邮件，浏览非常简单而老土的wap页面，但现在，绝大部分人的手机已经取代了电脑，成为日常生活中不可缺少的工具。

我们用手机交流，购物，欣赏影视，阅读书籍，玩各类游戏，尤其是飞

速发展的移动购物和支付体系，使得我们能在任意场合购买心仪的物品，订购旅游服务和宾馆，叫快餐，打车等等，生活非常美好，那么，整个世界的信息化程度处于什么级别呢？

我觉得，才刚刚相当于小学二年级，整个世界的信息化程度仍然严重偏低。从现在算起，往前10年，往后10年，这20年时间中，面向个人的信息化服务处于高速发展期，这个领域非常吸引眼球，因为它与每个人的生活息息相关。可是，另外有一些领域，却非常需要发展，那就是传统行业的信息化。

之前有不少传统行业，进行了一定程度的信息化，但这个信息化仅仅能满足自身运作的基本要求，当它与整个社会的潮流相对接的时候，就显得非常落后，迟缓。比如说在网购这个大体系中，普通用户所能看到的是商品展示，比价，下单的过程，但背后的核心环节却是配货与物流。

我还在上学的时候，有老师这么说过，现在计算机行业非常火热，很可能要饱和了，你们不一定非要从事这方面的工作。现在回头看这句话，觉得很有趣，人真的很难有眼光看到未来。去年我入职苏宁培训的时候，孙为民副总讲了当年一个决策失误的例子。90年代末，公司统计发现全国空调的年销售量达到数百万台，觉得很可怕，这个行业可能要饱和，估计要再想办法拓展别的商品经营了，但现在，全国空调的保有量为七亿台，即使完全没有新增，十年换一轮，每年也卖得出去七千万台，当年凭什么说这就饱和了？

所以我现在看程序员的状况，仍然是供不应求，尤其是高端程序员，十分抢手。这个问题的背景就是全社会的信息化进程在加速，之前的程序员人数远远跟不上需求量。

那么，如何解决这个问题呢？一方面是继续培训，促使更多新人来到这个行业，并且认真做下去，另外还有一些别的手段需要考虑。

我想追问一个问题：世界上懂业务的人多，还是懂技术的人多？很明显，懂业务的人要多很多，什么叫业务？其实就是行业常识，生活经验。

比如说，一个有经验的仓库保管员，可能文化程度不高，理解不了软件的运行原理之类，但一定对产品出库入库的流程非常熟悉，包括各种审批过程和异常状况，但这些，程序员是不懂的。那如果要促进这个领域的信息化，必然要在两者之间寻找一个结合点，程序员可以学业务，业务人员也可以尝试参与软件研发过程，目前来说，都是前者比较多，因为程序员相对来说还是比较年轻，学东西快些。但从整体社会效益来说，这其实是不利的，因为程序员是更稀缺资源，而传统业务人员非常多。

之前见过一个问题：如何让业务人员更好地参与软件研发过程。这个问题的根本解决方法是DSL（Domain Specific Language），核心解决方案是二次开发平台。

什么是DSL和二次开发平台呢，这两个词听上去很高端，但其实大家有很常用的东西就属于这个范畴，比如Excel，它提供了各种各样的公式，还有VBA，使用这些东西的人绝大部分不是软件行业的，Excel就是一种很成功的二次开发平台，公式和VBA就可以算DSL了。

很多时候这些东西还不够直观，我们可以看到一些图形化的编程语言，比如Scratch，现在很多小学生的兴趣班就会学，这些东西相对学起来就比较容易了，我们也可以做一些类似的抽象，以图形化的方式让业务人员能够参与，比如流程配置等等。图形化的东西，是最适合非技术人员理解的。

所以，要促进社会的信息化程度，最好是能够想办法把各行业的业务人员都拖进来一起搞。具体的分工大致是：技术人员和业务人员一起定义DSL，技术人员负责DSL的底层平台实现，业务人员负责使用它来构建业务模型和业务流程，甚至业务界面。

那么，软件行业的生产力是偏高还是偏低呢？我认为严重偏低。什么叫严重偏低？如果以机械力量的变革来对比，软件行业目前的生产力水平处于蒸汽机发明之前。也就是说，生产力远远没有被解放，大家做的大部分东西将来是会被机械化的，不再需要这么多人来做这么重复的劳动。可能很多人会对这段话不满，怎么就重复劳动了，你说说我做的什么是可以被机器替代的？

换个角度看，为什么几乎所有外行都觉得软件贵呢？因为人力成本太高了，他们觉得，做出这么多东西，应该是不需要这么多时间。为什么双方的反差这么大呢？

我觉得其中的关键点在于绝大部分工作的抽象程度严重不足，另外有很大一部分效率损失在编程平台或编程语言的不完善，比如Web前端。

从第一代到第四代编程语言，每一代都是损失一定运行效率，而大幅提升编写效率。随着硬件技术的发展，软件编程必然越来越粗放，大的趋势是不特别重视细节效率，只要没有数量级的性能损耗。

所以我们可以预期，会有越来越多的人使用一些运行效率相对不怎么高的语言或框架，只是为了提高单位时间的生产力。从老板们角度想，也会明白，提升运行机器的性能，要比多雇几个程序员便宜多了。因此，从整体趋势看，追求细节性能的程序员们恐怕会离自己的理想越来越远了，除非是在某些特定领域。

那么，绝大部分软件系统都可靠吗？我换一句话来问：各位程序员朋友，如果你们住的房子质量跟你们正在做的软件一样，你敢住吗？感觉大家都在笑，笑是什么意思，我们都懂的。

那为什么软件系统的质量不容易高呢？我觉得主要原因是流程不完善。那为什么不完善？需求容易变。为什么容易变？是因为不论程序员自己，还是需求方，其实潜意识都认为自己做的东西是变更成本较低的。

试想一下，为什么没人在盖高楼盖一半变更需求？为什么没人修大桥修一半变更需求？甚至做衣服做一半的时候变更需求，理发到一半变更需求，都会被人认为是不讲理。但是在软件领域，好像这倒成了普遍现象。

因为整个软件系统的实现，都是虚拟的，看不见摸不着，并不消耗什么物料，所以从这个角度想，变起来当然是容易的。但软件系统的架构，其实也跟实体的没本质区别，变更时候要考虑很多关联因素，并不是就那么孤立的看一小块地方，当然，也会有一些不影响全局的变更。打个比方说，如果你在盖房子盖到一半，那变更外墙颜色肯定是要比变更窗户大小容易的。要是想变得太多，估计只好拆了重来。

我见过不少公司是通过加强测试的方式来试图控制质量，但个人觉得这种方式不划算，而且收效不高。要想很好地应对需求变更，很重要的一点就是不要有这个软件一定不会改的想法，然后，从架构上做拆分，隔离，组件化等等，力争做到即使要改，也只改某一块的内部，不影响别的地方。

很多软件公司，一方面不注重架构的设计与宣贯，导致变更的时候问题多多，程序员也不能很好领会架构意图，一方面忽视整个过程中对架构的管控，认为架构只是最初那张静态图。

任何一种架构方案，都需要一个良好的管控机制。没有哪个盖大楼的只认真管设计图纸，不控制施工过程。架构其实是跟施工过程严格相关的，架构并不是一张扁平的图，而是一个立体的东西，作为整个系统工程的骨架。如果能在开发的时候看到这个骨架逐渐建立，血肉充盈的过程，对整个系统的成功把握一定会大得多，这也就是开发过程中架构管控的理念，具体实现要依赖于不同场景。

所以，将来的软件开发方案，一定是会朝着几个方向发展：

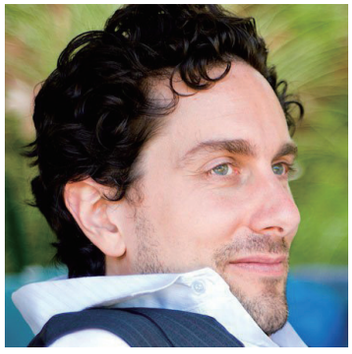
- 高生产力，单位时间生产效率更高，普通人员也可以参与
- 高可控性，整个生产过程更加完备可靠

有时候看现在的小孩子，会觉得他们很幸福，因为等他们这代长大，就不需要像我们现在这样编写程序了，那时候，编程已经成了一种令人习以为常的通用技能，就像现在的人用Office软件一样，所谓的编程，很可能已经不需要敲代码了，而是图形化，设置几个参数就完事了。■

阅读原文：老码农的技术理想

九　卦

为狗狗制作的商业模式画布



作者 / Tristan Kromer

精益创业教练，创业者社区 Lean Startup Circle 成员。

2012年4月1日，TaskRabbit[1]推出了一种全新商业模式：狗狗租赁服务（Puppies-as-a-service）。

尽管他们没有选择在这一方向继续坚持下去，但我觉得这个创意非常棒，而且非常适合拿来给我们的新款商业模式画布（Business Model Canvas）做例子。

注意

为那些不熟悉美国传统的读者说明一下，4月1日又称愚人节，总能看到很多愚蠢的行为。

更新：事实可能比小说更奇异，其实“狗狗租赁服务”这一想法并非原创。苏珊妮·麦克尔威说，纽约真的有宠物租赁服务。

狗狗租赁服务

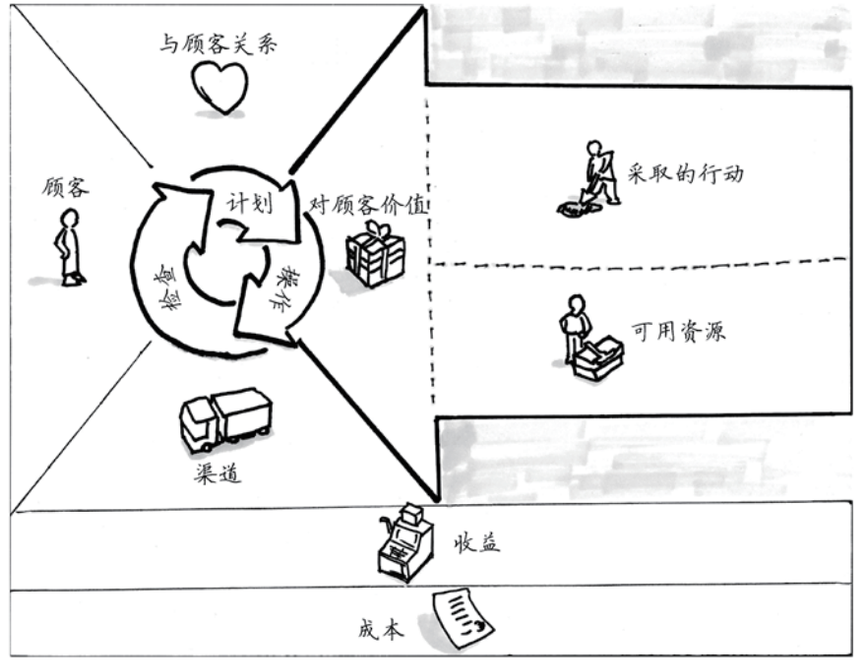
关于如何改变世界，TaskRabbit的创立者和首席执行官莉亚·布斯克在一篇文章中描述了这样一种令人印象深刻的想法：

[1] TaskRabbit，“跑腿兔”网站，可以在网站上发布或认领有偿跑腿任务，网站会抽取一定佣金。

狗狗非常合群，憨态可掬，总让人乐不可支，七大洲的人们都能看到它的身影。

我对此非常赞同。虽然我很想养只狗，但养狗要花不少精力，我在旧金山的狭小公寓里也没有多余的空间。如果我可以只在雨天里租一只小狗，大概就不至于像个不负责任的混蛋，世界也会变得更美好吧。

可是该怎样实现这个想法呢？让我们先把它画在商业模式画布上吧。



注意

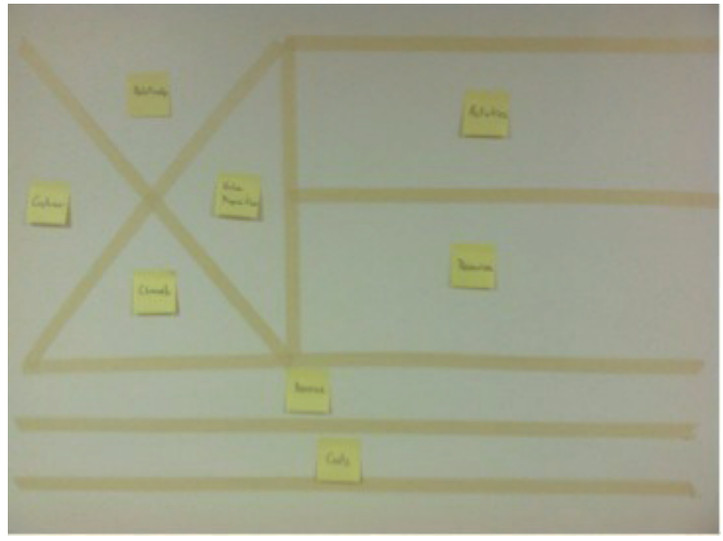
这里使用的商业模式画布是我自己改造的版本，在听取了埃里希·冯·豪斯克的批评意见之后，我还进行了修改。不过由亚历山大·奥斯特瓦德设计的原始版本商业模式画布在本例中也适用，所以你可以根据喜好任意选择。

找到一面墙

我需要做的第一件事是准备好工作场所。

我身高6英尺4英寸[2]，很重视用户体验，所以，用户体验型的人怎么做，我就怎么做。我撕开胶带，拆开便利贴，这样就能借助信息发射源（information radiator）[3]，把工作情况清晰地展示出来。我的目标是，任何一位初来乍到、想让所有人享受到狗狗的好处的新雇员，只要走进我的办公室，一眼就能看明白这种商业模式。

所以，我选了一大块墙面，用胶带把商业模式画布的结构勾了出来：



注意

我都是用便利贴直接在墙上工作的。不过鉴于我拍照的技术非常糟糕，接下来所有的注释插图均为数码技术制图。

[2] 约1.93米。——译者注

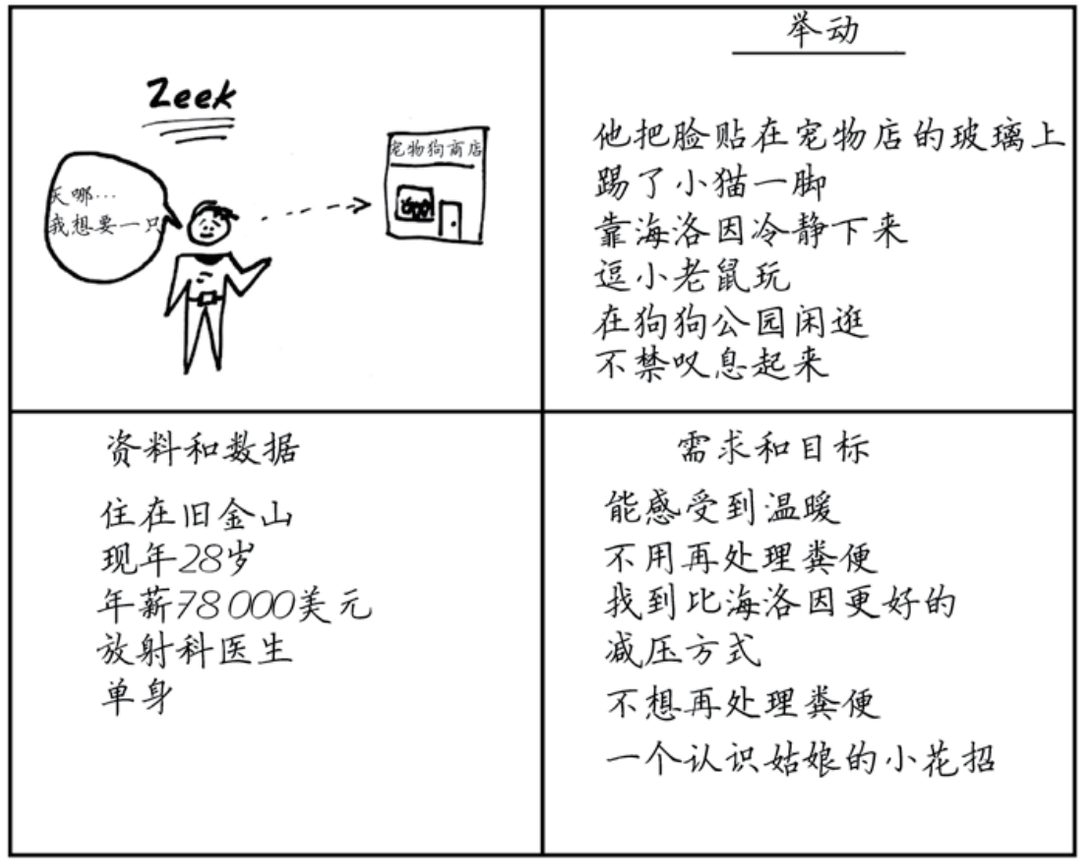
[3] 信息发射源，多由软件开发团队采用，形式多种多样，用来直观记录并展示、说明工作进程，尽可能减少工作被打断的情况。——译者注

架子支好了，可以把远大的愿景记下来了。从左到右看看画布的内容，我记起来得从这里开始……

顾客

显然天底下每个人都想租一只狗狗，但这一点对我获得第一位顾客毫无帮助。我需要关注顾客角色（customer persona）。具体说来就是我想知道，“为了能排在第一个租到狗狗，谁愿意露宿一夜？”

排在第一个的人既是早期使用者，也是坚定的拥护者。我画的图表如下：



我从珍妮丝·弗雷泽和凯特·鲁特那里学的这招。这就是一个LUXr[4]模式下可以用来应急的顾客角色，大概15分钟就能完成。

如果你之前没见过这种图表，需要一个制作方法（以及为什么要制作）的详细说明，可以在推特上联系我@TriKro。

有几件事要说明：

- 第一，Zeek有名字！这是口头上的一个代号。这样，我和团队成员讨论的时候就会讨论Zeek，而不是“一位用户”，而我们每个人说的“用户”都有不一样的意思；
- 第二，Zeek有一张脸。图片可以用作助记符，这样，每个团队成员都能记住Zeek是谁；
- 第三，Zeek不是一个伟人。他靠吸食海洛因来放松，还会踢小猫。

你可能会觉得对于狗狗租赁服务来说，这不是一个理想的早期使用者形象。不过我觉得，如果Zeek能借这个机会抚平自己的心魔，而且为了养狗戒掉海洛因，世界就能变得更美好。因此，我仍然会选择将他看作早期使用者。

毕竟，顾客角色只是我们对于顾客所作的一种假设，而不是一个一成不变、毫无争议的形象。

接下来，我想了解为什么Zeek会选择狗狗租赁服务。所以我需要……

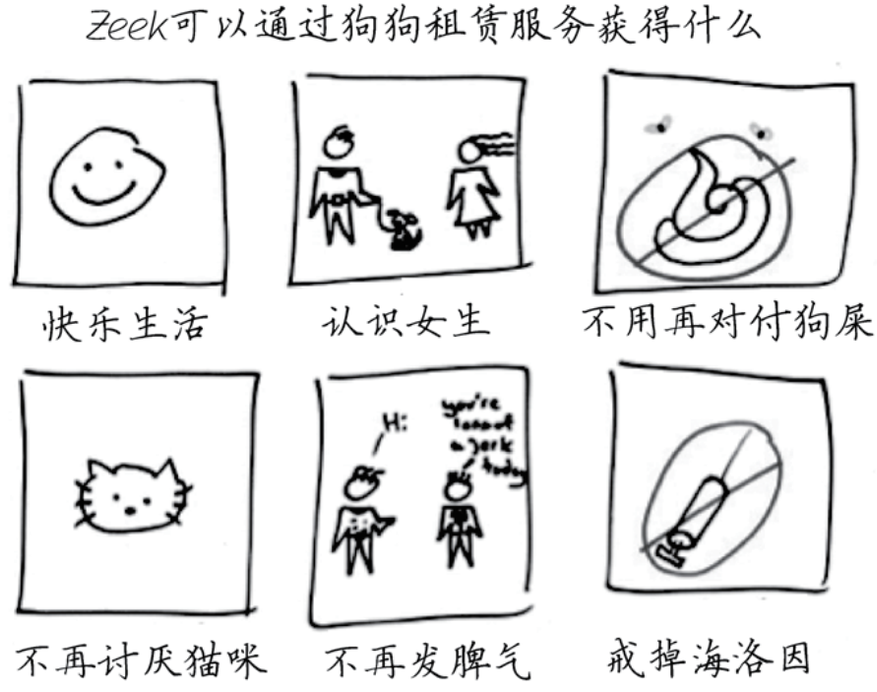
价值主张

在这里，我直接回到了精益用户体验训练的内容上，为“Zeek可以从狗狗租赁服务中获得什么？”这一问题绘制了六格图。这花了五分钟，我现在了解了Zeek可能从狗狗租赁服务中得到的六项收获。

注意

如果需要六格图更加具体的说明，请在推特上@我。

[4] LUXr，精益用户体验（Lean User Experience）。——译者注



我的团队成员（自我，本我和超我）迅速在投票中达成一致。我决定只关注Zeek想实现的三件事。

我列出的两件大任务很相似：“快乐生活”和“不再发脾气”。尽管引发的情感共鸣稍有区别，但这两项都说得太宽泛，我决定把它们删掉。

一项更为具体的价值主张是Zeek确确实实需要摆脱海洛因，这是重中之重。毕竟我做狗狗租赁生意是为了帮助他人、改变世界啊！

如果对顾客不够尊重、缺乏同情，那我活该失败。

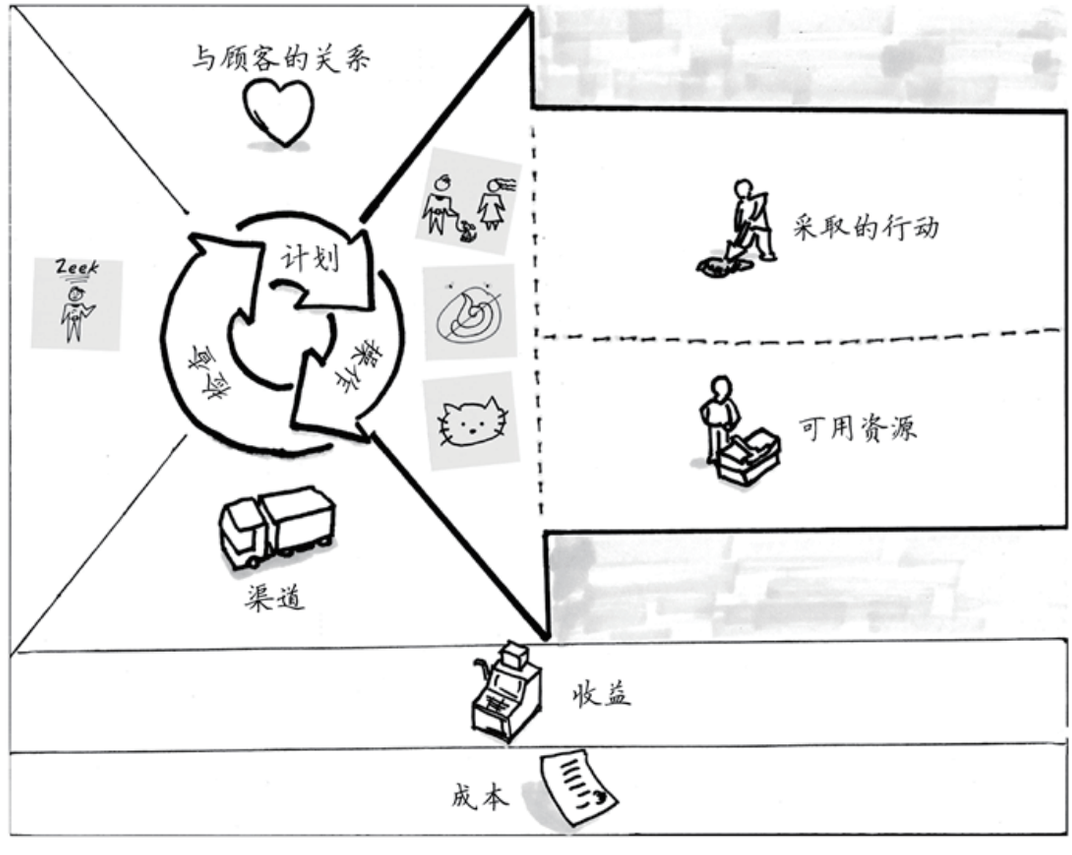
不过，由于Zeek是位瘾君子，他并不是真的想戒除海洛因。也就是说，虽然我认为Zeek的问题很严重，但就他本人而言，我认为的问题并不存在。相反，Zeek倒是的确有很强的愿望想要约会，所以“认识女生”可以作为“作用一”。

如果Zeek能意识到他的问题，与这项价值主张有关的交流就会容易很多，相关市场战略的操作也会更简单。

接下来，我选择把“不要狗屎！”算上，因为Zeek真的受不了狗屎（我同意）。

最后，我会留下“不再讨厌猫咪”这项，因为Zeek本来想靠租来的狗狗吸引女性，但他踢小猫的习惯会把她们中很多人赶跑。

现在，终于有些东西可以填到商业模式画布上去了。我们将表示“顾客角色”和三项“作用”的助记符画在便利贴上，贴到正确的位置上。



然后，我该好好想想怎样从顾客那里获得第一份反馈了，所以我需要填上……

与顾客的关系

如果有小偷来偷狗狗怎么办？如果狗狗随地撒尿怎么办？养狗的Zeek该向谁寻求帮助？

要做的第一件事是30秒的头脑风暴，把想到的所有东西都写下来，用上我最喜欢的UX工具：便利贴！关于如何处理与顾客的关系、提供顾客支持，我已经想出了四个主意。

Osterwalder提出，我可以为顾客提供自助服务，这样在长期中能大幅降低成本。就这次的商业模式而言，我更偏好采用自助服务，不过在这里没必要把整个想法表述出来。

计划-操作-检查（也叫创建-评估-学习，如果你更喜欢这个说法的话）循环（以及从左到右的自然阅读顺序）提醒了我，与顾客之间的关系代表了顾客为我的公司提供的价值。

我非常需要来自顾客的反馈信息。

然而，就算我相信世界上每个人都会想租一只狗狗，我的生意会发展到比Facebook更大的规模，但还是应该保持怀疑的态度。我应该对自己的想法有信心，但每迈出一步时都应该反躬自问，我采用的战术和策略是不是恰当。

有了来自顾客的反馈，可以确认我的价值主张有没有取得成功，检验我设定的顾客角色是不是足够准确。因此，我决定暂时舍弃自助服务这一选择。建立顾客论坛好像要从零开始，选在宠物商店的话成本太高（还拉开了我和顾客之间的距离）。我准备在咨询台设立一部直拨电话，为顾客提供即时的“狗狗支持”。

途径

下一个问题，我该怎样把狗狗租赁到Zeek手中？再来一次30秒头脑风暴。

我又列出了六种可能性，其中的胜者很明显：宠物商店！

为什么是宠物商店呢？在顾客角色的描述中，Zeek“把脸贴在宠物商店的玻璃上”。

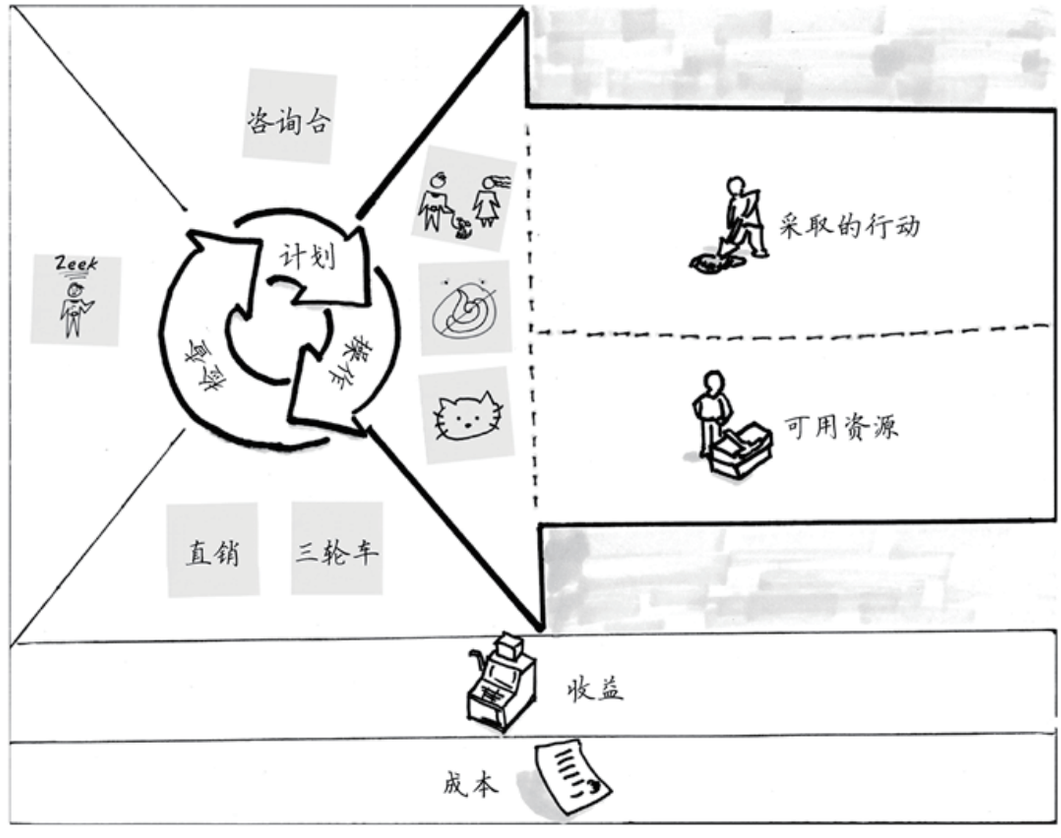
所以宠物商店很可能是一个不错的销售渠道。如果我看到Zeek（或者一个和他足够像的人）把鼻子贴在玻璃上，就可以试着租一只狗狗给他。

不幸的是，在通过宠物商店迅速发展了一些顾客之后，宠物商店的店主认为狗狗租赁会抢走他们卖宠物狗的生意，所以这条路走不通了。幸好，莉亚写的那篇关于TaskRabbit的文章提供了重要启示：

TaskRabbit新组建的三轮车队…改造之后被用来运送狗狗。

太棒了！我就让我的三轮车队在宠物商店外面晃悠，直接把狗狗租到Zeek手中。这样，如果宠物商店的店主拿着棒球棒冲出来，因为三轮车速度够快，我和销售人员可以迅速撤退。

将产品/市场的主要匹配方式填入相应扇形区域之后，画布是这样的：



《创业的智慧》涵盖13个重要领域，为你衡量创业的成败设定了标尺，讨论了精益硬件原型技术，分析了会毁掉公司的错误外包决策，展示了获取客户的实战技巧，给出了塑造品牌的策略指南，解释了各类天使投资人所扮演的角色……这些只是其中一小部分内容，相信它一定能给你带来启发和灵感，助你创业获得成功！本文节选自《创业的智慧》。

最简可行产品

画布还没有完工，但我已经很清楚最简可行产品该做什么了，主要让价值主张中的三项“作用”来驱动我造那堆东西。

显然，我要帮Zeek弄到一只狗狗，让他能在外面什么地方遇上女孩子，这是功能之一。开始时只需要一只狗狗，如果运气好，我可以借到一只小狗，把生意做下去。最糟糕的情况下，得去流浪狗收留所。

功能二是提供某种管理粪便的服务。也许我可以用一下门房测试，如果Zeek真的不会处理，我就举着粪勺追着他跑。

最后一项作用不会衍生出更多的功能，因为根据我的猜测/假设，如果Zeek带着租来的狗狗去公园，根本不用担心有流浪猫需要他轻轻给上一脚。猫很少去公园玩，再说，我希望Zeek能被周围所有陌生女郎的目光勾走了神。所以作用三就作为赠品吧。

不过这还没完，找不到顾客渠道的最简可行产品是对时间的最大浪费。

另外，我还需要为顾客提供支持，完成计划-操作-检查（或创建-评估-学习）的循环。所以我会租一辆三轮车来做第一笔直接销售，并把手机号留在狗狗的项圈上作为咨询电话，以便Zeek可以打电话让我坐着三轮车冲过去做“紧急清扫”。■

程序员为什么值得写博客

作者 / 黄峰达

1991 年出生于闽南，双子座。毕业于西安文理学院，现就职于 Thoughtworks（西安）有限公司。最喜欢折腾与计算机相关的东西：SEO、前端开发、硬件相关（如：Raspberry Pi，51，Arduino，OpenWRT，uCOS...）。在编程的路上已经走了较长一段，但始终相信前路更长更加广阔。

Hire Great Writers

仿佛这是写给自己看的，不过这在其中也有着相当有趣的意义 。虽然自己算是一个能写的人，或许这算是一种不算才华的才华，写博文的意义通常不会在于去描述自己怎样，怎样。通常在某些时候对自己来说，可以回顾自己学习的过程，呈现其中的一些思路，并其学习的过程分享出来。

原文的意义在于说明，一个优秀的写手，其优点并不仅仅在于写作，文法清晰代表思路明晰。优秀的写手都懂得如何与人沟通，他们使事情变得易于理解，他们善于换位思考，懂得抓重点、砍枝节，这些都是合格的应聘者身上应具备的特点。

想想自己算是后面说的那一部分，至于前面那部分——一个优秀的写手，就有待商榷。写的人越来越想，阅读的人越来越多的这个信息冗余的年代，会写就代表会思考？？（转载保留：程序员为什么值得写博客）

为什么要写博文

写一篇博文意味着要花一定的时间，有时候可能是一个小时，有时候可能会更多，于是人们开始去 copy。在这个 Ctrl+C 越来越盛行的年代，我们还是输了，于是乎在我们的国度里，我们的计算机书算是输给国外

的精英了。我们也有优秀的程序员，有优秀的工程师，只是在其中能与大众沟通的又有多少。我们的最终用户可从来不会管你用的是什么技术，他只懂得什么是体验，什么是速度等等。至于你说的东西，他不知道，也不会在乎。

这也就是为什么大师可以成为大师的原因，而菜鸟却还是菜鸟，大师在心里写博文的时候学会了总结，比如，定义算法的集合，将各算法封装，使它们能够交换。利用Strategy模式，算法和利用这些算法的客户程序可以分别独立进行修改而不互相影响。这个就是你需要的方法，于是大师就和你说了，“你需要Strategy模式”。这就是你要的答案，GoF分享了他的东西给了你，我们就有了一本《Head First设计模式》或者是《设计模式解析》。

我们开始走上了成为大师的西天取经，为什么是西天呢，这个领域一直是西方比东方分享得多。《西游记》就这样成为了《西游记》，写下这个过程的到处是吴承恩，还是师徒五人？师徒五人从一个地痞无赖直至成仙成佛。（PS：一直觉得自己写的东西，比较像不是散文的散文，中心似乎一直很明确，只是看懂的仿佛不到。）简单点来说，就是他们写下了自己的那些点点滴滴，我们就知道怎么去“西天”（我的意思不是那个意思，我想你懂的。）

这个过程就是一个个为什么你会看到那么多本优秀的计算机书的原因，大师分享了他们的心得告诉我们如何去成为大师，不过我还不是。只是如果你要成为大师，就要去分享你的过程。至于为什么？简单的说几点：

- 技艺的掌握在于重复。技术和游玩的相同之处在于技术玩得越多，也就越熟悉，当你试着去写一篇博文的时候，你也回顾了过程。游玩的回味可以再次欢乐，博文的书写可以再次熟悉。
- 你的过程正是别人所需要的。不要以为你手上的那点点关于编码的小知识不是别人所需要的，有时候人们就需要像《七周七语言 理解多种编程范型》这种书。

- 你的作品有可能因此重构。至于你对于重构是害怕还是享受，我就不得而知的，但是你写出来的时候，也许你会有更好的思路涌现出来。不好的一点是你还需要对这篇文章进行重构，不是么？
- 别人的评价。别人的评价有时候是打击，不过我想更多的时候是一种建议，比如Linus在刚写Linux的第一个版本的时候，他也遇到了这样的问题。至于宏内核好还是微内核好，这个问题有点类似于先有鸡还是先有蛋，不过我想后者可能科学家会给出答案。至于前者，不同的领域可能是不同的，Python好还是C好？相同的领域也可能是不同的，Ruby强大，还是Python强大？
- 最后一点就是，你想成为大师，不是么？如果你还甘愿……，我就不说了。

成为笔杆子

Copy与盗版

当我开始越来越频繁写博客的时候，同学开始复制，于是有一天他的排名对于我来说，已经遥不可及了，于是远远地排在了CSDN的前面。一步步的前进着，开始懂得怎样去试着推销自己的博客，这时候渐渐有趣了。又去鼓励另外一个同学去写博客，就如他所说的，“就算是你，写一篇博客也要一个小时”吧，或者对于我的打字速度来说，不算什么，半个小时可以达到三千，五笔加上机械键盘好的手感。

我们总会说别人写的说怎么怎么的烂，但是如果一本书上不是Copy过来的，那么他就是不错的，在版权的地位比代课老师还低的天朝。我想你就可以骂这本书烂，因为他是复制的，因为到了最后你没有找到出处。换到博文来说，你搜索到的结果一个个都是一样的，你找不到原版的文章，去问作者一些问题。

一开始的时候我试着去反抗那些复制，你花一个小时写的东西，可能在发布的瞬间就被抓取过去了。有趣的是，渐渐我发现这有利于我们去传播我们的思想。换句话说，这是一个信息时代，你写的东西有可能在一瞬间到了Obama的眼前。至于优缺点嘛，补充一句可以借此SEO。

天朝一直都有天朝特色，无论从哪些方面来说，计算机也是如此，中国特色的免费。至于付费，我想这就是为什么我不会考虑去做收费软件的原因了。 程序员害了程序员，自己害了自己有什么好说的。于是转战到了openSUSE，都挺好的fcitx的五笔很给力，bug也没有原来多，还有WPS For Linux下的此文，因为网络原因。

Copy对于读者来说，看到的都是千篇一律的东西，只会写的人失去兴趣。盗版对于用户来说，看到的都是免费的东西，只会让开发商失去动力。用户便看到了越来越多的广告，读者便只看一个门户的新闻。

如何去写博文

标题——必须重要，类名

对于写博文的人来说，重点的是如何清楚的去表达他们的想法，标题算是其中之一，这个也就是为什么标题党成为了标题党，而《设计模式》成为了经典。刚开始学编程的时候，更吸引你注意力的可能是《72小时学会Javascript》，而不是《Javascript 权威指南》，兴许让你买前者的原因是因为你能看懂前者，而后者不仅看不懂，而且价格更贵。只是一年以后，《72小时学会Javascript》被你扔到了垃圾箱，而《Javascript 权威指南》却放在了原来放那本书的位置上。你定义的类难道仅仅应该是class class1么?

小标题——地图，method

小标题有点类似于sitemap.xml，只是他就是站点地图，一点就到了相应的地方。他应该直接了解的说这是开始菜单，标题栏，菜单栏，而不应该是简简单单的第一章，如果你真是那样写的话，你写的函数想必是

```
def fun1
end
```

如果你写的是get*data那么，我想你的函数名应该和你的文章一样，告诉人们，你要的是get*data。所以不要吝啬你鼠标的一下，它可以承受上百万字的点击。如果因为那样坏了，你可以告诉我，我可以帮你免费换一个欧姆龙的微动，前提是你的鼠标可以换。如果是HTML那么应该是h2 h3，markdown也就是用得比较多的github上的README.md的##或者是###

内容——函数体

这里可不是让你用一个让人难以理解的Magic Number，你写得越复杂，别人看的时间就越久，通俗易懂，就是一个很好的开始。你可以把一个个复杂的方法分解出来，或者提炼函数，或者重命名。当你相信你看不懂你的文章，正如你看不懂你写的hello,world我想你是时候去重构你的函数了。

复杂的部分，就用段落来解决，一个函数如同一个段落只应该表达一个思想，太长了就如同这篇文章一样没有多少人会认真去看。你需要给你写的一个精美的代码加一个注释，所以你也需要给你复杂的地方加上个（PS）。

引言——README

我想都会去看的，无论是在破解软件的时候，还是github上面的项目。简单的说说，这篇文章是干什么的，这个程序是干什么的。大家都会，不是么？

没有什么好写的？

说说你是怎么开始编程吧，然后写在你的博客上，你会发现你会爱不释手的。

小提醒

- 代码，代码有时候会更清晰的表达你思路，太长的代码可能会影响阅读，通常不超过一屏就不算太长
- 图表，耐心的画个UML图，或者程序框图也是不错的，很清晰的表达你的思想。
- 美观，要知道C上是有indent，如果我看到别人让我帮他看的代码是一坨。。。WPS也有段落，如果你没有学好WORD，找本《72小时精通Office》吧，顺便找PPT、EXLS也学了。
- 格式，记得好好用好手上的工具，如果你用的是CSDN用的editor，试着一个个探索，CODE应该要有CODE的格式，LIST应该要有LIST的格式

最后，耐心

在CSDN上的博文的话，可以按长尾理论来分析，这里说的通常是指——你的东西是原创的，写博文有些时间。SEO上，以谷歌为例，谷歌对其抓取是比较及时的，同时谷歌会排除掉部分专业的复制网站——就是拉到重复的搜索结果里。文章刚发表的时候的流量有可能会很低，但是有些文章时间一长就显示出来了，比如我写的东西中的《Android上使用GCC》算是一个很好的示例。一开始的时候我们写的东西访问量不会很大，特别是我们刚起步的时候，这时候就要一步步慢慢来。只要你写的东西是别人需要的话，那么就会一步步慢慢来。如果你写的刚好是热门的话题、技术的话，那就是好莱坞大篇《速度与激情1》《指环王2》《黑客帝国3》《纳尼亚传奇4》。。。直到《哈利波特7上》，《哈利波特7下》。于是作为迪斯尼的你，又推出了下一部分电影……

总结

现在的我们更多的技术是直接来源于Google、百度、CSDN或者其他，搜索得来的，我们并不去考虑别人在其中花费的时间和经历，有时候我

们要试着去想我们是不是也应该分享给别人。这算是自己开始写博客的原因，受益于开源社区，我们自然而然的也要回顾给这个社区，只有分享才会使未来更美好。

我们都希望看到有一篇博文够清楚的对我们当前所遇到的那个问题进行好好的解析，问题是也许你解决过的那个问题正是别人所需要的，但是你并没有将它分享出来，仿佛是一个循环一样

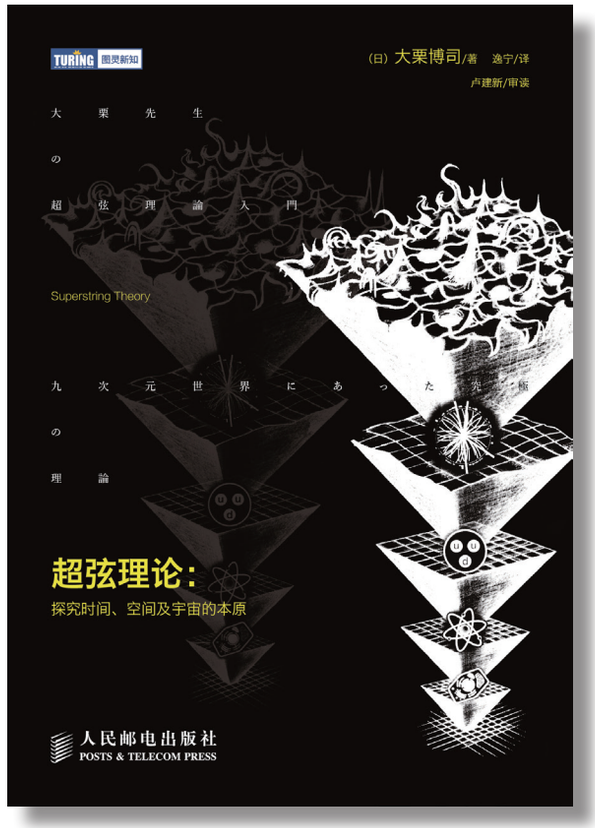
```
for(;;){
}
```

于是我们又回到了一个起点，人都是自私的。我们都希望自己能更快的学习好一项技术，一门语言，别人也需要你手上的那项技术，那个语言。都你开始意识到别人需要你手上的东西的时候，你算懂得换位思考了。

写篇分享，写篇心得就是一个好的开始，或许我们已经被高中的语文所吓怕了。但是，是时候从新开始。如果你被C的指针吓坏了，被C++的模板吓到了，被Javascript的简陋吓到了，而你又需要拾起它，我想是时候重新开始了。■

阅读原文：程序员为什么值得写博客

书　榜



超弦理论：探究时间、空间及宇宙的本原

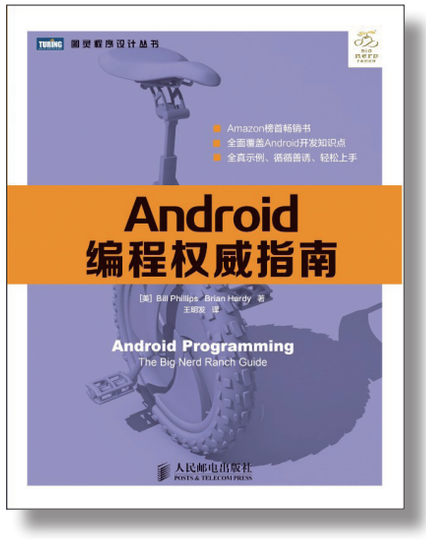
作者：大栗博司

译者：逸宁

书号：978-7-115-37386-1

图灵社区推荐：5

“超弦理论”是继牛顿力学、爱因斯坦相对论之后，时空概念的“第三次革命”。“超弦理论”统一了引力理论与量子力学的矛盾，超越了“弦理论”的局限，解释“标准模型”中“费米子”与包括“上帝粒子”的“玻色子”的振动形态。本书中，大栗教授以通俗、风趣的语言讲解了量子物理基础、“弦理论”到“超弦理论”的最新发展、“超弦理论”的理论原理及证明，并在“超弦理论”下重新思考与探究了时空概念。



Android编程权威指南

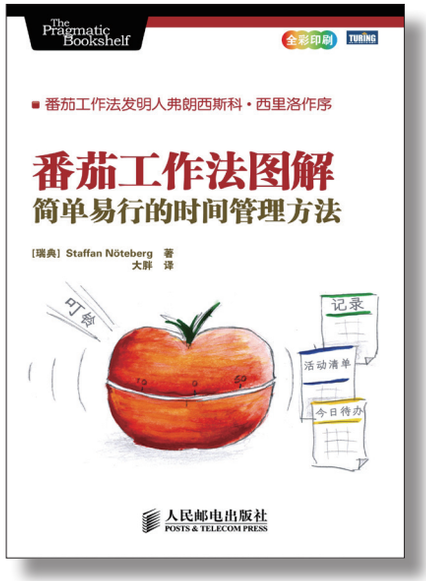
作者：Brian Hardy，Bill Phillips

译者：王明发

书号：978-7-115-34643-8

图灵社区推荐：23

本书根据美国大名鼎鼎的Big Nerd Ranch训练营的Android培训讲义编写而成，已经为微软、谷歌、Facebook等行业巨头培养了众多专业人才。



番茄工作法图解：简单易行的时间管理方法

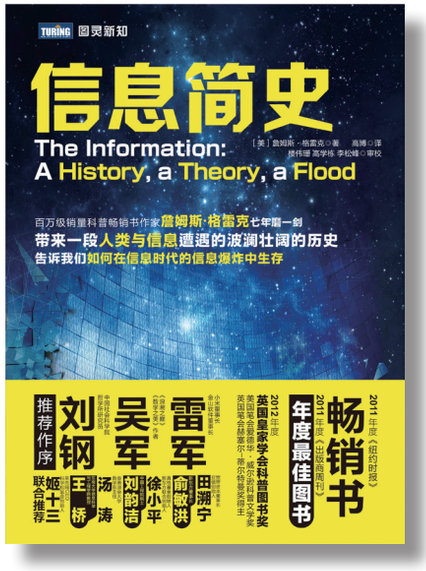
作者：Staffan Noteberg

译者：大胖

书号：978-7-115-24669-1

图灵社区推荐：14

作者根据亲身运用番茄工作法的经历，以生动的语言，传神的图画，将番茄工作法的具体理论和实践呈现在读者面前。番茄工作法简约而不简单，本书亦然。



信息简史

作者：詹姆斯·格雷克

译者：高博

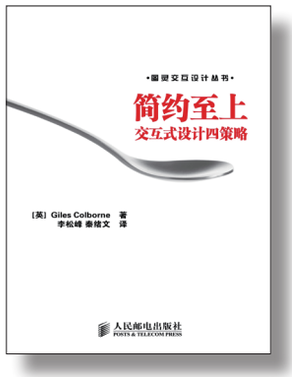
书号：978-7-115-33180-9

图灵社区推荐：31

在信息“碎片化”时代，这是一本值得收藏并反复阅读的书。生动的语言，哲学式的思考，超群的洞察力，为我们描绘信息史。

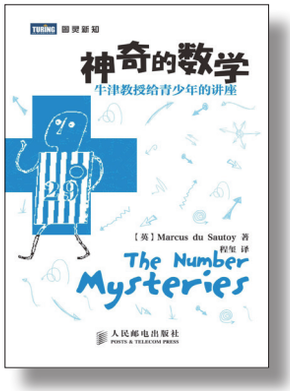
5 24小时365天不间断服务：服务器/基础设施核心技术

作者：伊藤直也，胜见祐己，田中慎司，广濑正明，安井真伸，横川和哉
译者：张毅
书号：978-7-115-38024-1
图灵社区推荐：10



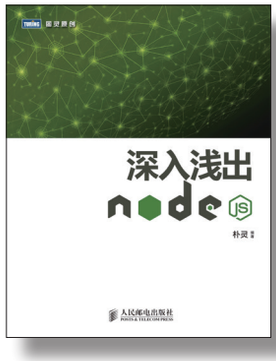
6 简约至上：交互式设计四策略

作者：Giles Colborne
译者：李松峰　秦绪文
书号：978-7-115-24324-9
图灵社区推荐：10



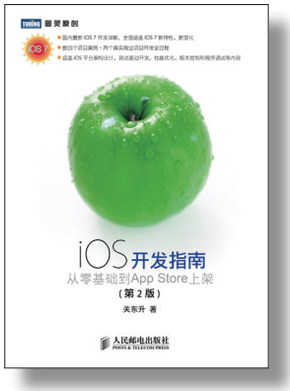
7 神奇的数学：牛津教授给青少年的讲座

作者：Marcus Du Sautoy
译者：程玺
书号：978-7-115-30241-0
图灵社区推荐：12



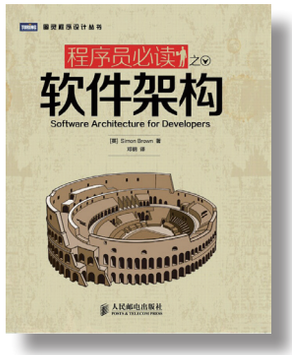
8 深入浅出Node.js

作者：朴灵
书号：978-7-115-33550-0
图灵社区推荐：17



9 iOS开发指南：从零基础到App Store上架（第2版）

作者：关东升
书号：978-7-115-34802-9
图灵社区推荐：8



10 程序员必读之软件架构

作者：Simon Brown
译者：邓钢
书号：978-7-115-37107-2
图灵社区推荐：9

电子书榜

1. JavaScript编程全解
   ——无论是前端还是后端的JavaScript开发者都可以在本书中找到自己需要的内容。

2. Web性能权威指南
   ——世界顶尖的Web性能工程师讲解并演示了针对TCP、UDP和TLS协议的性能优化最佳实践。

3. Node与Express开发
   ——熟悉JavaScript的前端和后端工程师会在书中发现一种新的Web开发视角。

4. HTML5数据推送应用开发
   ——一本简明的数据推送技术指南。

5. Python网络编程攻略
   ——这本书非常有趣，你可以随意挑选任一攻略进行阅读。

6. 图解HTTP
   ——从基础知识到最新动向，一本书掌握HTTP协议。

7. Linux Shell脚本攻略（第2版）
   ——精选极具实用价值的技巧，让你的日常工作更加轻松。

8. 自制编程语言
   ——手把手地教你用C语言制作两种编程语言：crowbar与Diksam。

9. 系统化思维导论（25周年纪念版）
   ——历久弥新的经典著作，开启心灵，磨砺思维。

10. MongoDB权威指南（第2版）
    ——数据库开发人员的工作指南，系统管理人员的进阶指导。

妙　评

Hello Kitty：我的成功你不能复制

作为一个姑娘，从来就没有喜欢过Hello Kitty，但这不妨碍我读完《Hello Kitty的秘密》这本书。事实上，我非常感兴趣这只连嘴都没有的傻猫为什么能红遍全球40年甚至更长。

可是，书里没有答案。连Hello Kitty的创作者清水侑子和三丽鸥的创始人，被誉为“Hello Kitty”之父的辻信太郎都说不清这只无嘴猫走红的原因。

《Hello Kitty 的秘密》是对Hello Kitty的诞生成长和三丽鸥的经营策略的全景式分析记录。作者以Hello Kitty的成长进程为主线，通过对三丽鸥创始人、Hello Kitty之父辻信太郎，Hello Kitty第三代设计师山口裕子，三丽鸥欧美、日本相关运营负责人的独家访谈以及相关资料，替这只影响力遍及世界的可爱“无嘴猫”述说她的秘密，并深入剖释了三丽鸥的全球品牌经营策略，以及Kitty作为日本“卡哇伊”文化符号后背深切的经济与文化价值。

1974年11月1日，Hello Kitty诞生。她被定位成出生于伦敦，体重等于三只苹果，顶着一只红色蝴蝶结，喜爱在森林里玩耍，练习弹钢琴和烤饼干，只有坐姿的猫。这个卡通形象在刚被创造出来时，辻信太郎也不过觉得还行，并没有预判出日后的疯狂情形。他甚至有点怀疑Hello Kitty能否形成风尚：“说实在的，当初我刚看到那张图的时候只觉得不太坏而已。而且我也压根没有想到这会是三丽鸥最红的明星。”

70年代的日本，正是动物卡通形象大行其道的时候，孩子们都喜欢动物明星，史努比正努力掏干净孩子们的口袋，大量的美国电视剧里也日复一日地向孩子们灌输动物明星：灵犬莱西、友善的海豚Flipper和温顺的大熊Gentle Ben。与此同时，日本国民的生活品质却遭到挑战。原本温

作者 / 一阐提人

孕晚期准妈妈一枚。电商运营出身，半吊子产品，半吊子项目管理。先后就职于58同城、蜘蛛网、杉德集团等多家互联网公司。喜好以用户体验为中心，用数据说话。不爱争论，只爱埋头阅读。

馨的家庭和住宅区正在被冷酷的摩天大楼和冷漠的公寓楼房所替代，不得不让大人们与冷酷的社会划上等号。女人们逐渐沦为丈夫的用人，在疏离的公寓楼里期望着与人沟通，获得安慰的时候，柔软温暖的Hello Kitty出现了。

这大概就叫运气。

不可否认，Hello Kitty能持续走红，自然有后期三丽鸥的运营、营销策略之功。比如没有在一开始就让Hello Kitty的形象铺天盖地——过于轻易能得到的必将不被珍惜；比如至始至终让Kitty远离毒品、酒精、性这些大人们认为孩子不应当接触到的领域——孩子喜欢可不够，父母才是买单的人；比如让Kitty像一个正常姑娘一样穿衣打扮——20-30岁没有孩子的姑娘也是需要一个安抚心灵的玩偶形象的。

但这一切都是后话，即这一切都发生在Hello Kitty走红之后。这真的不能证明，辻信太郎有多英明。

1960年，辻信太郎开了山梨丝绸公司，卖的是丝绸产品。62年就开始招募设计师为自己的公司设计卡通形象，当年也出现过小红一时的“草莓”形象，但很快就被市场遗忘。随后，他开始做礼品生意，他进口了霍尔马克卡片公司的卡片，可很快发现日本人其实不爱送卡片，这个业务其实是失败的，同样失败的是他获得了芭比娃娃的进口权，可是日本人对这个形象也不买账，这使得辻信太郎最终损失了7亿日元。

同样失败的还有辻信太郎1974年在洛杉矶开设的“三丽鸥传播公司”附属机构，专门拍摄电影并在北美发行三丽鸥产品。十几年内制作了二三十部影片，但大部分都没赚到钱。

最糟糕的也许是1990年他在日本多摩市打造的三丽鸥彩虹乐园。除了被指责模仿迪士尼乐园之外，这个乐园拖累了三丽鸥的资产负债表，至今依然受其影响尚未恢复。

种种迹象表明，三丽鸥不过是凭借着Hello Kitty这一形象为大众所了解，并不代表着它的每一条业务线都盛满了成功的香槟酒。辻信太郎或许有前瞻的商业眼光，但依然只是一个普通的人，非卡哇伊之神。

当今这个社会，我们常会奉某些商业精英为神，比如电商界的马云、马化腾，并对他们的发家史津津乐道，以此来鼓励自己。仿佛今天吞下的泡面明天就会变成金条。可谁敢说他们的成功不是运气？马化腾当年若真的找到买家，以100万的价格将QQ出售，今天还会有牛逼的腾讯吗？马云当年不过就是想模仿下ebay，他真的能高瞻远瞩想到今时今日被誉为电商教父？

Kitty的成功不过是证实了专家的话：突破总是在你最不经意的时候来临，要发展一个成功的商品，你得有10%的技巧与90%的好运。这其中有太多的变数，相信他的产品一定会大卖的人头脑绝对有问题。■

转自微信公众号：LovelsBug

阅读原文：Hello Kitty：我的成功你不能复制

“半个保险丝”之谜
——《咨询的奥秘》翻译轶事

作者 / 劳佳

硕士毕业于上海交通大学，现在SAP美国任高级软件支持顾问。业余爱好语言、数学、设计，近年翻译出版了《咨询的奥秘》、《卓越程序员密码》、《周末读完英国史》、《加州大学伯克利分校人文建筑之旅》等书。

在被我拖了好几个月的稿之后，在新年来临之际，《咨询的奥秘（续）：咨询师的百宝箱》的新译本终于上市了。作为译者，能够翻译温伯格先生的两本经典咨询作品，实在是一大幸事。有兴趣的读者，可以在图灵社区或豆瓣试读来阅读本书部分章节。

在翻译过程中，我多次就文字中遇到的问题向温伯格先生本人咨询，先生均一一作答。唯有一个问题未解，是关于第六章中的一个故事（本节亦可见于豆瓣试读）：

> 1969 年，暴风雪。雪堆已经攀上了窗台。景色固然壮美，可我们和六位客人整个圣诞节都要困在屋里了。不过既然宾·克劳斯贝能渡过难关，我们决定也要尽力而为。出门的车道被封住了，不过在房子下面的车库里有一捆木柴，冰柜里有一扇上好的牛肉，地下室里还有一柜子好酒。我们没觉得这是灾难，而是尽情享受节日聚会，可这时候水泵坏了。
>
> 实际上它也没全坏，更准确地说是坏了一半。不知道是什么原因，水泵转啊转啊，可抽上来的只有涓涓细流。我们还可以化雪水来喝，所以活命倒不是问题。哦，可能还是有问题的——没水冲马桶的话，在这个只有一个洗手间的房子里，八个人可是够受的。

《咨询的奥秘（续）：咨询师的百宝箱》以生动的语言将咨询人员应注重培养的个人能力归纳为“咨询师百宝箱”中的一件件法宝，其中包括：智慧盒（正确的判断是非的能力）；金钥匙（不断探索新的学习和实践领域的能力）；勇气棒（尝试新方法并承担风险的勇气）等，列举了咨询人员应该注重培养的能力、方法和生活态度。

幸运的是，我们的六位客人里有三个电气工程师：两个博士，另一个是机械工程和电气工程双硕士。一位博士艰难地下到地窖里去“瞧一眼”，我们剩下的几个人就安心地松了口气。

在一趟趟上楼下楼、一项项检查、一次次讨论、一套套理论之后，水泵一直转啊转啊转啊，水还是只有一点点。工程师们似乎江郎才尽了——看起来就是这样。

这时，一位不是工程师的客人发话了。这位金发女陶艺师提议说：“可能是保险丝的问题。”在不到一微秒的时间里，她的想法就遭到了你能想到的最尖酸刻薄的奚落。不过她却不为所动，坚持说：“为什么不可能是保险丝的问题？”

然后就有了三套完整的解释，理论层次虽有不同，不过最后都归结为一件事：根本不可能是保险丝嘛，因为泵还在转呢，虽然只是部分在转。

“那可能是半个保险丝嘛。”她说。工程师们开始居高临下地和她解释保险丝没有半个一说：“保险丝要么是好的，要么是坏的，没有中间态啊。”

“我家就有一半正常工作的保险丝。”陶艺师维护着自己的说法。话说到这一步，工程师们就转去讨论别的事情了。这样的说法根本不值得一驳。

几分钟之后，陶艺师不见了。我有点担心工程师敷衍的态度会让她不高兴，于是去找她，以尽主人之谊。后来我发现根本用不着操这个心。

她从地下室回来的时候正好碰上我。她不发一言，走到厨房水槽边拧开了水龙头。看着涌出的水，她带着胜利的笑容宣布：“我早就告诉他们是半个保险丝的问题。”

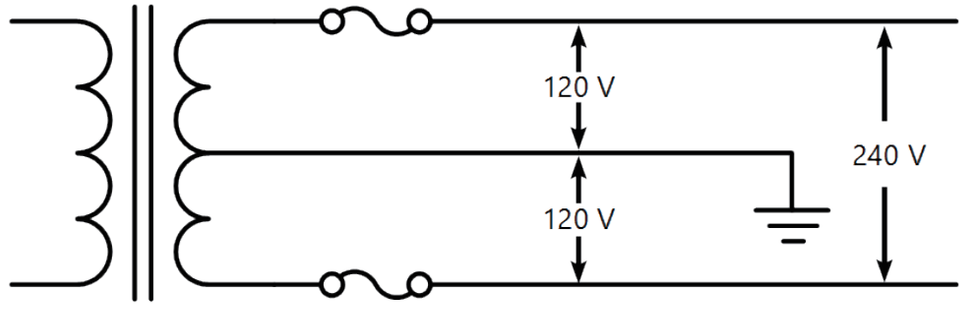
> 确实，还真是半个保险丝。更具体来说，把 120 伏转成水泵用的 240 伏所需的两根保险丝中的一根。一根保险丝烧断了以后，可怜的水泵就只能以额定电压的一半徒劳地在那儿抽水。泵是在转，可就是没什么作用。

故事就是这样了，可我还一下子没有反应过来。为什么烧断了一根保险丝之后就只剩下 120 伏了呢？难道不应该是整个断开了吗？看来学电的都掉进一样的圈里了……

我给温伯格先生写信，可惜先生自己也记不清楚细节了。他说这房子修于 1920 年前后，当时的电气标准和今日也未必相同。如今他也进不去当时的房子，无从查看线路是怎么接的了。

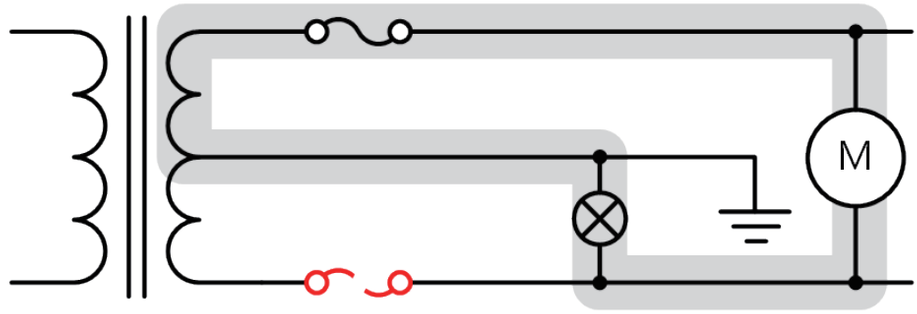
不过幸好（或是不幸），过了些时候，我家的电路也出了点问题。电力公司把后院的水泥凿开，挖出地下比手指还粗的铝电缆，一直修到半夜。就在这时，我忽然明白这半个保险丝是怎么回事了。

话还得从美国的电路接法说起。它用的是一种 180° 分相的接法，如下图所示。从变压器下来是三根线，两根火线之间是 240 伏，中心头接地。两根火线各负责家里的一半（比如卧室和厨房），平常小电器和照明用的都是相电压也就是 120 伏。但是遇到水泵、烘干机、电炉这种大功率的电器，假如用 120 伏那电流就太大了，得需要很粗的铜线。那么这种大功率的东西就是接在两根火线上，用 240 伏供电。



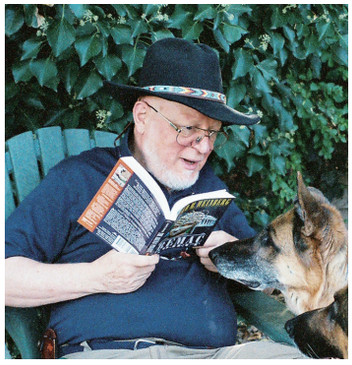
好了。每根火线上都有一个保险丝（当然现在都是多路空气开关了），如果有一边烧断了的话，那这一侧就没有电了。

那用 240 伏的呢？跨在两根火线之间似乎也就不通了呀……但如果有个灯开着呢？



原来如此！通过一盏比平常要暗的灯，水泵就又开始用还不到原来四分之一的功率吱呀呀转起来了。■

《系统化思维导论》的25周年纪念



作者/杰拉尔德·温伯格

数据处理领域最负盛名的专家之一，国际著名的演讲家和顾问，美国计算机名人堂代表人物，也是Weinberg & Weinberg顾问公司的负责人。他撰写过《你的灯亮着吗?》《成为技术领导者》《系统化思维导论》《程序开发心理学》等30多本广受欢迎的著作，在西方乃至全球都拥有庞大的读者群。这些

“要解决我们面对的重要问题，不能停留在当初制造它们的思维层面上。”

——爱因斯坦

从记事起，我就一直对思维很感兴趣。我从1961年开始编写这本关于思维的书，在它上面整整花了14年的时间，本书在1975年得以出版。从那时起，我收到了几百封来信和关于这本书的评论。大多数读者表示本书帮助他们改进了思维，这让我很高兴。但是，因为编写本书也帮助我改进了思维，所以对此我并不吃惊。

我不是喜欢收藏的人。我找不全25年前本书刚面市时收到的那些“漂亮”的书评，也找不全那些信件。所以，我有点困惑，不知怎样写这篇前言。

好吧，大多数思维，甚至是一般系统思维，有时候都需要一点运气。我花了点时间下载了自己的电子邮件，并很幸运地找到了这样一封赞扬信，部分摘录如下：

我是Wayne Johnson，是一名兽医，在中国南部提供技术咨询服务……大约10年前，我很偶然地发现了这本书，也可以说很意外，当时我正在寻找一些基本信息以支撑我的增长模型项目。我要告诉你，这是我读过的对我影响最大的书之一。那本书我最后不得不还给了大学图书馆，后来我好不容易说服了某位书商，帮我订了一本。

读者还建立了专门的组织和网站来讨论和交流书中的思想。要想了解温伯格的其他作品，请访问其个人网站：www.geraldmweinberg.com。

这些年来，我常常收到此类信件，并乐此不疲：

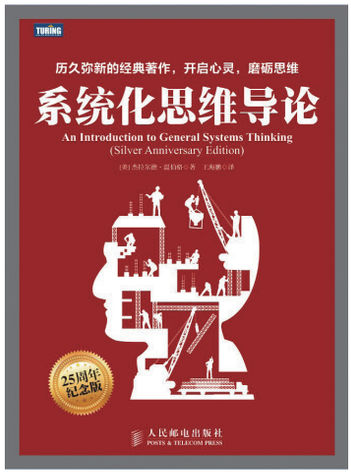
- 来自地球的另一端（如中国南部）；
- 来自我从未想过会影响的专业人士（如兽医行业）；
- 说这“是我读过的对我影响最大的书之一”。

但是，我对这本书近些年的遭遇感觉有点疲倦。显然，在前一家出版商的计划中，没准备出版那些25年一直不过时、一直有需求的书。结果，一系列生活成本的自动增长将这本书的价格推到了不合理的高度，印刷量完全不能保证供应，即使已重印了20多次。二手书溢价卖出，我的小库存也萎缩了，所以我决定拿回这本书的版权，将它转给更懂这本书的出版商Dorset House Publishing。结果就有了这个版本的面市。

开始编写本书时，我已经写了6本关于思维的书，但都是基于计算机编程思维的视角来写的。我写了很长的时间，然后意识到计算机语言的变化比人的变化要快得多，所以决定将编程语言的生意让给别人，专注于更一般的思维法则。结果，我先编写了《程序开发心理学》（*The Psychology of Computer Programming*），然后才是这本书。现在，经过了20多年，这两本书还在静静地发挥着它们的作用。也是我的作用。

我想没有多少人在25年后会重读他们自己的作品，但既然我重读了两次，就反思了一下多年后自己有哪些改变。

- 那时我肯定比较年轻，至少现在看来是如此。那时，我觉得自己相当成熟能干。我怀疑今天的自己是否会有这样的勇气，去写这样雄心勃勃的书。
- 现在我知道得更多，这源于更多的经历，但我最大的兴趣仍未改变。我仍然完全痴迷于人的思维，以及它多彩的可能性。
- 我没有改变自己的信念，即大多数人如果学习一些思维法则，他们的思维能力就会比现在强很多。
- 我的写作风格改变了，我发现以前的某些用词有点离奇。例如，自从出版了这些书



《系统化思维导论》初版于1975年面世，此后四分之一个世纪始终畅销不衰。21世纪初，银年纪念版出版，再次掀起阅读风潮。这是一本全面介绍一般系统思维的权威指南，旨在帮助人们掌握科学的思维法则，揭开科学与技术的神秘面纱。书中通过基本的代数原理，使用大量图表、符号，乃至方程来展示探索项目、产品、组织机构等各类系统的方式方法。另外，作者还通过有启发性的举例说明、大量的章后练习，以及附加的数学符号练习，强化读者对问题、系统和解决方案的思考能力。

后，在一些读者反馈的督促下，我在写作时有意识地去除了一些带有性别歧视的语言。我很高兴自己这样做了。当看到有些作者说无性别歧视的语言非常“拗口”时，我想这更深刻地揭示了他们自己的想法，而他们本来是不想暴露这一点的。

- 最近写作时，我更多地使用“我”，而不是“我们”或“它”。不论好坏，这些毕竟都是我的想法，而且我写的是思维和思考者。所以，如果这些非直接的形式隐藏了思想背后的思考者，就对读者造成了伤害，毕竟他们对思维的主体感兴趣。我希望现在的读者能原谅我年轻时的荒唐，并能因此获得更多的练习机会，看到每个思维过程“幕后的人”。
- 经过大量有意识的学习，我确实感觉到，现在的我更了解个人在思维方式上的差别。我借鉴了诸如我的导师维琴尼亚·萨提亚（Virginia Satir）和阿纳托尔·拉波波特（Anatol Rapoport）的模型，还有迈尔斯？布里格斯性格类型指标（MBTI）和神经语言学（NLP）模型。这些模型就像一般系统这块蛋糕上的美味糖霜。
- 经过这些年的咨询工作，我现在更清楚如何将这些一般法则应用于具体情况。在关于软件管理、系统分析、问题定义、人际交往系统、咨询和系统设计的著作中，我已经尝试着记录这些知识。

我期待看到这些书还能再畅销20年！ ■

图灵社区 出品

出版人：武卫东
编辑：李盼
顾问：杨帆
设计：大胖

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: ebook@turingbook.com