技术写作手册

动机、构思、工具、发布、推广、长期主义

/thoughtworks

目录

序言

我越来越觉得写作是一件蛮难的事情。简单来说，在如今的日常下，想要达到持续的写作状态，有两件本质性的事情，不得不做。

一是克制。好的写作者，必然需要源源不断的输入。但现在触手可及的屏幕，乱象丛生的讯息，正在不断将低质的词汇、句式甚至观点推到写作者面前。拒绝它们，就是在拒绝人性本能和诱惑。屏幕背后伺机而动的算法，以及线索呈发散状的微博和朋友圈，攫取的恐怕不只是时间，更是在摧毁岌岌可危的关注力。克制需要体现在，我们要有选择地信赖良好的信息源，关闭那些会侵蚀自己的东西。关注时间和心绪。

二是沉浸。写作需要某种心流的存在，而进入这样的心流需要作为过渡性的时间。很奇怪这样的说法如今很少见到了。当更多在强调高效和完成的时候，我们也在自

我物化成没有感受的机器。

我们需要为这样的过渡、沉浸直至心流，刻意腾出时间，即便当下看来是多么奢侈的事情。但如今更常见到的，人在每天被一件接着一件接着又一件的事情追赶，头绪像乱作一团的耳机线，思想只好委屈变为无力的荒漠。

得益于 Thoughtworks 的分享文化，我还是能看见这些年来不断冒出的优秀写作者，并有幸把他们关于写作的思考汇集成册，做成这样一本小书。关乎写作的很多问题，似乎都得到了解答：为什么需要写作，灵感从哪里来，用什么工具，以及发往何处。但我也相信，只有长期有写作习惯的人，才会对这些问题的背后，心有戚戚。

说是手册，但写作这件事，在多大的程度上是可以通过技术和翻查去达成的呢？虽然这本书的目录和文章结构，在观感上也许能给予读者一点理性的参照，但写作更需要的，是一个人长情的投入。

张凯峰
Thoughtworks 洞见主编

写作为什么这么难

众所周知，写作有很多的好处：可以记录和巩固学到的知识、扩大自己的影响力、提升和推销自己等等，甚至很多人通过写作出书，成名，改变了人生的轨迹，这种例子很多很多。至少我每次听到这类的故事都会陷入深深的自省，痛恨自己没有早一天开始行动，然后痛定思痛，立即新建一个博客或是把之前落满灰尘的博客打扫打扫，重新发一篇《新的开始》，然后……就没有然后了。周而复始，也慢慢失去了对自己的信心。

所以问题的关键不在于我们不知道写作的好处，而在于我们做不到，就像大多数的鸡汤文一样，要知道“总有一天”是不靠谱的。对于放弃我们总能找到很多理由：没有时间，总有更重要的事情；反正知识已经掌握了；不知道写什么；觉得自己想写的东西不值一提，怕被人笑话等等。后来我想清楚了，归根到底就是四个字：

觉得不值，假如有人给我 500 万让我写篇博客，我想上边那些理由就都不再是任何问题了。

那么什么样的事情是我们心甘情愿想去做的呢？为什么我们喜欢玩游戏而不喜欢写作业？那些我们根本不需要“坚持”就能一直做下来的事情都有什么样的特点呢？想来想去，我认为一般要符合两个条件：**快速的正向反馈、相对容易。**

我就曾经疑惑过学习与玩的区别到底在哪？玩游戏可以让我们相对容易能短时间内满足生理或是心理上的需求。但是学习首先不容易，有好处但反馈周期都相对较长，我们很难获得比较快的正向反馈。可是有些好学生就非常喜欢学习不是么？因为他们可以从每次考试，每次回答问题，从老师家长赞赏的同学羡慕的眼光中得到很多正向反馈，而更重要的是这种反馈足够快，所带来的收益已经抵消甚至大过了在学习上投入的成本，这就是为什么好学生会越来越喜欢学习的原因。

我们常说要培养孩子的兴趣，这样他就能在某一方面作出成就。我认为这个逻辑反了，应该是先促使或是找到

或是暗示孩子在某方面的成就，让他觉得自己在这方面比别的小朋友强，他就能在这件事情上持续获得更多的正向反馈（例如夸奖，奖励，其他小朋友的羡慕），就会越来越喜欢做这件事情，兴趣自然而然的就形成了。

所以说学和玩本质上没有什么大的区别，主要是付出成本和短期收益的差别，甚至通过调整这两个维度，学与玩是可以相互转换的。例如每个电子竞技选手每天都要在一个游戏（竞技项目）上玩（训练）超过 10 个小时，对每一个操作细节都要反复练习，而每提高一点点都非常的困难，每天要被教练训，在比赛中还很难获胜，需要承受着家庭和社会的压力，对于他们来讲，这已经不是在“玩游戏”了，虽然外人看起来没有什么差别。

很多牛人可以通过心智自我管理，推迟幸福感，用长远利益来驱动限制自己的行为。不过那是需要天赋和训练的，对于我等凡人，如果一定要坚持做一件我们认为需要坚持才能做到的事情，我认为可行的办法就是：**能否找到一种适合自己的鼓励方式，能让我们自己在做这件事情的时候，能够得到持续快速的正反馈，并尽量把这件事情变得容易。**

回到写作这件事情，抛开那些“总有一天”才能实现的好处外，眼前的好处无外乎就是帮助我们记录理解消化沉淀学到的知识了。不过我们的内心里总有一个声音反复出现：反正书看了，session 听了，感觉知识已经学会了，那还值得花时间写么？我用这个时间多学点东西不更好？

我的前同事李光磊曾在他的文章《知行合一》中将知识划分为信息，知识和智慧。这解开了我很多对于学习的疑惑：为什么我看了很多书还是不会用某项技术？为什么我们听了很多 Session，但是感觉却没什么用？为什么同一本书不同人看或是同一个人不同时期看会有不同的收获？原来我们之前所认为的学习只是获取信息而已，如果不思考，不行动，这些信息将一文不值，随时间消失，而浪费掉的却是宝贵的时间。而同样的信息，不同的人或是人的不同时期通过思考转化为的知识也是天壤之别的，所以说书只是一面镜子，我们读的看的其实是我们自己而已。

所以为了真正的学到知识，而不单单的是收集信息。我们需要用行动来将获取的信息加工吸收变为自己的智慧

的一部分，而写作算是比较简单廉价的方式了，所谓写作驱动学习，正是如此。就像测试驱动开发中的测试一样，写作一旦完成，理论上就可以丢掉了，发不发布、有没有人看、写得好不好，都已经不是那么重要，它已经体现了它的价值。相信只要能体会到这种好处，不断地用写作来驱动学习，注重积累和推广，那些长远的好处也会自然而然的在某一天突然到来。

最后，不论你觉得上面这段文字有没有道理，它对你来讲只是一段信息，如果没有思考，没有行动，那就只是在浪费你的时间而已了。

作者：**王健**

Thoughtworks 企业架构咨询总监、首席咨询师

如何看待写作这件事

写作是创造性活动

在现代化的社会环境中，分工越来越明确，越来越多人只工作在自己手头的事情上，甚至为生产更多所谓价值而终日劳作，难以窥探全貌或反思自省。这无异于资本对人的异化，进而产生“工具人”的自嘲。

而创造性活动，如绘画、谱曲、手作、烹饪、演讲……当然也包括写作，是能抵抗这种异化的有效方式。创造性活动，不只是被动地接受信息输入，还需要主动地收集信息，对其进行加工与整合，继而以多样的形式进行输出和呈现。

举个例子：买了一本书放到书架上，我不会认为这本书是我的，尽管从所有权上它属于我，但我并不真正知道它；当我花点儿时间读完，它也不是我的，我只是把书

中内容在脑子里过了一遍，需要时，我还是不能熟练地运用其中的知识和技能；再进一步，当我能清晰简练地把书中内容讲给他人，可以说这本书至少有部分是我的；而当我写完了一系列拆书稿，想到它时历历在目，用到它时融会贯通，它就真正变成了我的书，甚至不需要作为物理实体存在，它已经融入我的知识系统，随取随用。

感受一下其中的区别，就在于我对书中内容的加工程度不同。越深度加工和生产内容，越能获得真知灼见。这让我想起小时被要求写读后感，实在很有道理，可惜年幼时并不能领会这一点，白白浪费了大好的读书时光。

写一段文字最好的时间是现在，好在还算不晚。

可能需要一点儿心理建设

动笔前，可能需要一点儿心理建设。写作路上的阻碍不少，不同的人有着不同的担忧。首先一个问题是没得可写。

没得可写又分为两种情况：第一种是真的没素材，虽然这种情况凤毛麟角，但也不失为一种可能性。我的建议就是不要写了，好好享受大好生活，没有素材的人生多

么幸福。这可不是我说的，这是作家说的。想来也是这样，不平则鸣嘛，所谓的创造冲动，一定都是遇到了什么戳心窝的事儿，或胸含大量槽点不吐不快。冲突、悲剧、求而不得，这些才好写也好看，王子和公主最后过上了幸福生活，就不要强说愁了。

第二种情况是，素材谁都有，只是缺少发现。就像我们在培训现场，客户会问：“有没有真实项目案例可以分享经验的？” 一手素材效果最好，只有身临其境才能感受到现实世界的细枝末节。专注地倾听，专注地活在当下而非臆想的世界中，都有助于培养善于发现的敏锐度。有“开始观察”这一意识和动作，就不失为一个自我培养的契机。

解决完“没的写”这一问题，又开始担心写了没人看。综合一些官方统计和我的观察，超过 80% 的受众是会直接忽视内容的，假设推送 100 个人，理想情况也只有 20 人会点开，其中 10 人大致翻看就关闭，5 人会仔细阅读一下，1-2 人会有针对内容的反馈。所以，没人看很正常，那些针对内容产生反馈的受众，不管是支持还是误解，真爱粉无疑了，值得感激。

在反馈的内容里，或许存在一些负面的声音。有可能是误解，毕竟“我想的 ≠ 我写的 ≠ 你看的 ≠ 你理解的”，信息层层传递必然会失真。从这点上看，被误解确实是表达者的宿命。我们可以视影响程度和范围决定是否响应，必要的情况下予以回应和澄清，毕竟勘误也是表达者的职责。

也有可能负面反馈来自于恶意曲解或碰瓷，这种情况就视心情决定要不要陪他玩儿一把了。按个人经验来讲，不必过于担忧，群众的眼睛都是雪亮的，他们会根据自己的情况选择愿意信任的内容。换句话讲，真就因此粉

转路人转黑的，可能也并非你的受众。

有时候，不知如何把握内容的深浅也会造成一定的心理压力，担心大家都懂班门弄斧，或者过于晦涩缺乏理解。其实只要意识到，不管你写得多深或多浅，都有比你强和弱的人同时存在着，每个人都在不同层面不同程度地碾压别人，或被别人碾压。闻道有先后，术业有专攻，想开就好。

不需要准备更多，就可以动笔

有了一些基本的心理建设，并不需要准备更多，就可以开始动笔了。一般情况下，熟练的写作者会形成自己固有的工作流，因人而异。刚开始写的同学，就可以完整体会这个从无到有、积少成多的过程。

写作的过程可分为脑力劳动和体力劳动。脑力劳动是指素材加工、构思、立意、大纲的过程，而体力劳动就是码字、排版、修改、发布的过程。动脑的事情其实不局限于时间和地点，可以在碎片时间随时做（注意安全），比如扫除、洗澡、通勤路上、做饭时，思想不受限，任你信马由缰；而动手的事情宜集中时间统一处理，找到

自己趁手的工具也很重要。

用不了几篇的迭代，就可以形成自己独有的高效写作流程了。

关于构思与下笔，有些人习惯瀑布式的先构思好再下笔，这样可以体会一气呵成的快感。也有些人习惯初步立意后边写边构思，慢慢体会一篇文字跃然纸上水到渠成的获得感。无所谓好坏，找到适合自己的节奏就好。

在内容泛滥的今日，优质内容依然是稀缺资源。每次输出都是机会，都需要提供尽可能优质的内容。我对优质内容的理解，与洞见有较大的重合：

1. 明察洞悉，清楚地看到，透彻的了解，能引起强烈共鸣，但不带节奏
2. 能透视不易察晓的事物，看透本质，高明的见解
3. 人无我有，人有我优，人优我精，人精我绝

当然，一下就洞见并不容易，可以先从持续练笔开始，不断的练习和迭代。

写完之后做什么

分享一下我的个人习惯：以发布划分，发布前看三遍，发布后寻求反馈，隔段时间回顾。

发布前看三遍，第一遍看内容结构，第二遍看遣词造句，第三遍代入读者视角。基本上三遍过后，就没啥自己能改的了，可以找信任的他人看看，补充更多的第三方视角。

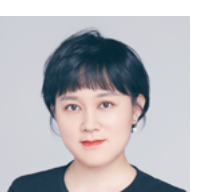
发布后，可以主动寻求一些社区反馈。如有相对稳定的发布渠道，经过一定的传播后，可分析文章相关数据，作为后续内容改进的输入。有时会发现一些有趣的关联，或者意料之外的惊喜，有机会我们再展开讨论。

然后过一段时间，不必拘泥是隔了三月五月，还是半载一年，重温一下自己的文字，回想成文时的心境，因此沟通而认识的人，便能深刻感受到自己的成长，感受到自己与这个世界密切的关联。这也是我最喜欢的阶段，不管是嘲笑之前的幼稚，还是同情当时的遭遇，时过境迁后，你都能与彼时的自己隔空相对，会心一笑。

最后，且写且随缘

大家会发现，较大的篇幅都在前面。因为不存在一个恰好准备好的时机，只要卸下心理负担，开始动笔写，坚持行动就能甩掉大部分人。持续地写作和输出，坚持一年、五年、十年，你会看到自己的进步，而那些影响力和因此带来的其他什么，都是你不断完善自我的附赠品。

最后想说：表达是每个人都有的权利，只有充分表达才有可能被看到、被理解、被需要。而表达过后，受众如何看待内容和作者本身，也是受众的权利，作者无权干涉。不论是有缘交流，还是无缘错过，都很好。所以如何看待写作这件事呢？这真是一件既有益又有趣、甚至有点小浪漫的事儿，坚持写吧，然后顺其自然，随遇而安。



作者：**于晓南**

Thoughtworks 资深咨询师、
赋能专家

这几年技术写作的感悟

从翻译开始

大概在 4 年前，我开始技术写作。当时在学习 Shiro，顺手翻译了几篇官方文档。我觉得这几篇文档写得很不错，但是英文不好的同学阅读起来会比较费劲。另外，翻译的过程也比较费时。既然我已经翻译过，为何不分享出来？于是便注册了博客，发布了三篇 Apache Shiro 的官方文档翻译。

再次登陆博客，时间已是半年之后。发现其中一篇有 7000 多的阅读量，这个阅读量还是挺出乎我的意料。不过其他两篇阅读量只有几百。为什么有一篇阅读量这么高？当时的我没想明白。很久之后才发现可能是因为这篇文章在百度检索靠前的原因。在百度中搜索“Apache Shiro 文档”，可以看到三年前的这篇翻译

排在最前面。

7000 多阅读量也给了我写作的信心，于是写博客的念头也就延续了下来。

开始原创

一年后，由于工作上需要使用 Apache Kafka，于是花了些时间学习。后来在团队内做了一次分享。在这个过程中，我阅读了很多文档和书籍，逐一消化成为自己的知识。如果把自己学习到的内容以系列文章的形式留存下来，不仅让自己能够学习的更深入、记忆的更牢固，而且也能够帮助到初学者。其实之前我没写过技术博客，更没写过教程，没有任何经验。但我对于自己把一个问题说清楚的能力比较自信，另外也相信自己能坚持下来。于是一上来我就定位写系列教程文章。

我对于文章的要求是通俗易懂。所以多采用生活中的例子以及绘制了大量的图表用于文章中。彻底理解了的内容我才会写上来，并且一定力求读者能够看明白。在之前学习过一遍的基础上，大概又花了一个月的时间完成《Apache Kafka 入门教程轻松学》系列 8 篇博客。

接下来我又完成了《ZooKeeper 入门实战教程》系列，一共 5 篇。后面写了一些 Apache pulsar 的翻译文章及其他一些零散的文章。

有何回响?

写完博客当然希望有更多的人来看。但是如果一开始就报这个目的去写，一定是急功近利。为了点击量有一万种写法。为了博客的关注量，也可以有一万种写法和做法。但这对文章质量可能毫无帮助。其实投机取巧的方法有很多，不过我不想走这种捷径。

我最初的想法很简单，踏踏实实写一些对自己和读者真正有帮助的文章。不浪费读者的时间，也不浪费自己的时间。最开始就坚定以内容质量为核心价值来写文章。

博客一开始自然没有什么点击量，不过也并不影响写博客的心情。不过偶尔有人留言问问题或者给我鼓励，我还是很欣慰。写作目的不一样，写作的心情就不一样，最终写出来的内容也不会一样。管那么多呢，先把心思花在文章质量上。

一次和同事聊天，他说那天我在百度搜索 Pulsar，第

一个结果就是你的博客。当时我很诧异，因为之前从没有把我的博客和百度联系起来。当时 Pulsar 确实属于比较新的工具，文章比较少。我也只是随便写了一篇介绍文章而已，毫无质量可言。但是这篇文章的访问量却比用心写的那些博客高的多，也不知道该高兴还是难过。不过通过这个事情，我了解到百度的力量。怪不得那么多公司不惜重金在百度上做 SEM 或者 SEO。

当时我觉得这只是因为 Pulsar 比较新，我钻了个空子被百度排在了前面。因为文章本身没什么价值。所以其实并没有给我带来什么成就感。

有段时间我换了新的工作后，工作比较忙，没时间继续写博客。后来又有段时间去别的平台写专栏。但这段时间博客的关注量、评论量、点赞量一直是上升的趋势。因为有之前 Pulsar 的事情，我想到是不是有哪篇文章在百度排到前面去了。于是去百度搜索了下，发现按照技术名称 + 教程 或者 + 标题中关键字 检索，我的一些博客排在了前面，比如按照 “apache kafka 概念”、“apache kafka 教程”、“apache kafka 入门”、“zookeeper 教程”、“zookeeper 入门” 等相似关

键词搜索，我的博客都能排在前三名。

这次和 Pulsar 文章排在前面比较起来，感觉是不一样的。因为这些文章确实是我花费心思去创作的。自己的文章能被更多人看到，能帮助到更多初学者，还是很有成就感。最重要的是坚定了继续写下去的信心和动力。

一些感想

我开始技术写作到现在已经四年多，时间不算太长，不过也有些个人有些感悟。随便说一说。

写作目的

首先想说的是写作的目的。写博客没有利，说俗点就是赚不到钱。那么写博客会出名吗？不排除有些大牛会通过博客出名。但是对于绝大多数写博客的人来说是不会出名的。读者看到你的文章能解决他的问题，并且心情好时，可能会给你点个赞。但谁也不知道你是谁，而且技术博客这么少的阅读量也不足以让你出名。但是技术博客是一个窗口，能让更多人能看到你、了解你。如果你的文章质量足够好，当然也会有更多的机会找上门，比如说写专栏、甚至出书。不过这些毕竟是少数，刚开

始技术写作的朋友还是应该脚踏实地，写好每一篇文章，并且坚持下来。

从我个人而言，我写博客出于以下几个目的：

1. 巩固学过的知识
2. 分享知识，帮助更多同行
3. 成就感

最开始我只是想通过文章巩固自己所学知识。学习是输入的过程，写文章是一个输出的过程。如果能在工作中实践当然是最好的输出，但并不是学习的内容都有实践的机会。那么写文章输出也是个不错的选择。

如果写文章只是为了巩固知识自己看，那么文章的质量就没那么重要，自己看懂就好了。当你的文章将要对外发布，需要考虑的因素就会更多。比如是不是通俗易懂，有没有模糊的知识点、有没有错误、语句是否通顺等。

如果想让更多的人看到你的文章，文章的质量就变得尤其重要。技术领域一切都以质量说话，时间沉淀下来的肯定是有质量的文章。比如我之前在百度排在前面的

Pulsar 的文章现如今已经在几页开外了。

成就感对于我来说是个副产品，是随着你的写作而来的。但是很重要，因为这是正向的反馈。有人在读，说明你写的还可以，并且确实也帮助到了一些读者。那么继续写下去才有更大价值。

在这个自媒体泛滥的时代，好多文章选择标题党或者其他走捷径的方式。但在技术领域行不通。不要一开始就冲着文章的点击量去写文章。其实你只需要搞定一件事，那就是文章的质量。其他都是随之而来的。当然好的标题也很重要，不过再好的标题也要有同等质量的文章相匹配。

长期主义和积累

写博客是个积累的过程，从你的第一篇博客开始，到 10 篇、100 篇，以至更多。这绝对是一个长期的过程。当然也有办法让你的博客一天就刷到 100 篇，比如转载。但是这样做没有价值和意义。我也不清楚我的博客在百度中排到前面用了多久，但我相信肯定不是三个月就能做到的。

文章开始没有点击量很正常，不用气馁。至少从我的经验来看，只要文章写的不错，终归会被读者认可。

还记得最初写了几篇博客后，没有人关注，阅读量也很低，难免心里会有些失落。但是随着你积累的文章越来越多，你会发现各种数据都会加速上涨。因为写文章是一次付出，持续收获。每一篇文章都是你的存款。即使有段时间你不写博客，还是会新增很多关注、点赞。当然，写作是一种习惯，需要坚持，最好不要中断。

写博客的快乐和痛苦

写博客很快乐，否则也不会写下去。学到知识的快乐、

分享的快乐、被认可的快乐。然而痛苦也会很多，要写出一篇质量比较好的博客，需要付出很多。首先自己要彻底搞懂要写的技术，然后再用自己的语言通俗易懂的写出来，还要自己绘制图表。这个过程需要付出很多时间和精力。

以我的经验，一篇 2000-3000 字的博客至少需要 4 个小时左右（学和写）。要求更高的专栏文章，起码要 6 个小时。只是写写博客还好，毕竟没有交稿的压力。如果和平台合作写专栏，面对交稿的时间点，会给你巨大的压力。我曾经写了半年专栏，业余时间除了写就是写。每晚几乎都是 1 点以后睡。周末要带小孩，于是抓各种空子写。小说、电影、电视剧基本和我告别了。

有人认可你的文章，自然也会有人认为你的文章是垃圾。不要因为个别负面声音影响自己的信心。自己的文章当然也会有错误。读者指出来，最初会觉得有点没面子。是人就会犯错嘛，既然放到互联网上，就不要怕犯错。其实读者也是在帮助你纠正错误的知识点，是个好事情。

说了这么多写作的痛苦，但写博客的好处会更多。比如

下面这些：

能力锻炼

- 写博客是吸收知识、消化知识、再次形成知识的过程。在这个过程中，你的学习能力、总结能力、表述能力都能得到提升。

巩固学习的知识

- 博客也可以认为是你的学习笔记。通过写博客可以巩固学习到的知识。在你工作中或者面试前，可以用博客来复习。

个人能力背书

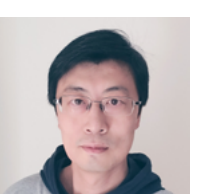
- 短短几十分钟的面试，根本无法让面试官对你有全面的认知。如果你的简历中有博客地址，面试者可以通过阅读你的博客，对你有更全面的认知。这是我在面试过程中的切身感受，很多面试官都会去看你的博客。

此外，博客也是你对外展示的窗口，更多的人可以通过博客认识你。通过写博客，我才有了和专栏平台合作的

机会。这是我最初写博客没有想到的。最初专栏平台通过博客找到我写专栏，我还有些犹豫，不知道自己能不能写出来并坚持下去。后来想给自己一个挑战，于是就把这个事情接下来了。过程是相当痛苦的，这里就不再详述，不过总归是完成了专栏。写专栏的过程对我后来工作上碰到的一些困难还是很有帮助 —— 只要有信心，朝着正确的方向坚持前进，那么一定会胜利抵达终点。

最后

这几年写技术博客的快乐肯定大于痛苦。而且之前所谓的痛苦，现在看来也不算什么。以后肯定还会坚持写下去。学习是长期的事情。学习之余，把学到的知识总结成博客，对自己有益，又能帮助到其他有需要的人，哪怕只是一小部分，那又何妨？学无止境，笔不停耕。



作者：**李一鸣**

Thoughtworks 资深软件工程师

写了十年技术博客的收获

今年刚好是我写技术博客的第十个年头。恰逢最近也有些所想、所感、所得，所以想把发自内心的对技术博客这件事唠叨几句。算是想到哪写哪，请见谅。

我的技术博客经历

我的第一篇（姑且算是技术分类，但实际是我自己的做的一个软件介绍）技术博客发布于 2010 年，在我大二的时候，那时候的技术博客还托管在博客园上；之后在 13 年到 14 年左右的时候，为了熟悉 Node.js ，用 Node.js + MangoDB 写了一个内容管理的 cms，并且部署在 heroku 上（后来因为没有续费，账号被删，源码也没有了）。最后因为维护成本过高，迁移到 github 的 jekyll 上，一直沿用至今。我博客的地址是 http://qingbob.com/， 但你可能更听说过它的另一个

马甲——知乎专栏前端技术漫游指南。

说真的我很好奇，在阅读这篇文章的你们有多少是冲着标题进来的，被“收获了什么”这个带有些神秘色彩的语句诱惑的不行，迫切想知道最终的答案是什么。

我不敢说收获是零，但如果我告诉你，写技术博客的这十年，我收获的可能并不会比你更多，你会不会很失望？

请允许我解释这件事，让我从一个工作的实际例子开始。

在加入项目的一年多的时间里，我负责的其中一项工作内容，是和前端代码中用到的一个第三方类库 Handsontable 斗智斗勇。这个类库据说能够承担海量级别的数据渲染。但是在我们的项目中，在有的场景里使用它渲染上千条数据的时，就会导致浏览器呈现一种假死的状态，无法响应用户的请求。

我的工作就是需要找到它的瓶颈，修复它，还原它的能力。

在解决这个问题相当长的一段时间内，我都是抱着一种类似于刑侦探案，缉拿真凶的心态，试图找到性能问题

背后的罪魁祸首。我认定真凶只有一个，一定存在某一处并不周正的代码拖累了整个页面的性能。

但结局是……这么说吧，好比其实是我找到了十处存在性能缺陷的代码，每处会导致 500 毫秒的延迟，单拎出来任意一个缺陷都在可以接收的范围之内。但是十处合并在一起拖累的就不只是 5 秒，而是会产生一加一大于二的效果。

说到底它还是一个写代码写糙了，或者说代码逐渐腐化了的问题，没什么大惊小怪的。但我总幻想着背后有什么惊天大阴谋，解决它就能带来质的飞跃。

哪有那么多事半功倍的好东西。

我想说的是，写技术博客也是一样，别指望它在某个时刻瞬间给你带技术上的突飞猛进，让你焕然一新。它和所有的“活”一样，通过长时间的实践，能让某些技巧变得娴熟，并锻炼你的一些能力。而所谓的“意外收获”？门都没有。我见过很多很厉害的、让我钦佩的程序员，他们也并非各个都写技术博客，但我相信他们有他们的方式训练自己。

所以我其实一直觉得，写技术博客是一种训练自己的过程。和反复地写代码，在上台前反复的练习演讲是一个道理。

对于我个人而言，我发现“发表是吸收的利器”。比如，你现在可以尝试把你今天学习到的知识马上向身边的小伙伴复述一遍，你会发现过程中你会对一些概念有些模糊，或者他会向你提出一些你没法立即回答的问题。这些都是值得再回过头去学习的知识点，而技术博客就是这么一个无声的小伙伴。

写作的迷思

即使是说训练，我觉得还是有一些写作过程中的迷思和陷阱想提醒大家，纯粹是我个人的经验。

首先需要明确的是，博客究竟是写给谁看的，读者还是我自己？

除非你在刻意运营公众号，需要迎合流量，否则一定是为你自己而写。

首先写作出发点就是你自己，只有当你有写作欲望的时

候，才有可能下笔，才想把它尽善尽美的写好。

其次，也是更重要的是，**读者关注的并非是你关注的。**

分享一个很有意思的事情。有一段时间，我的技术文章会同步发布在知乎和掘金上。于是这部分文章有了三个视角: 我看待它们，知乎的读者看待它们（点赞和收藏），和掘金的读者看待它们（点赞和收藏）。我发现，一般我个人花了最多时间去整理和学习知识之后写出来的最满意的文章，或者是我认为价值最高的文章，通常在这两个平台无人问津。而“快速入门”和“全解”之类的文章收藏量和阅读量却最高。

我理解他们，但是我觉得很可惜。

某一年我拍脑袋想出来的一个算法相关的 idea，只花了三天时间就写完了，但这篇技术文章的点击率之高竟然成为了掘金的年度文章，让我哭笑不得。而且知乎和掘金的群体又不太一样，因为这篇文章在知乎上就没有任何的反响。

你永远都不知道你的读者在想什么。

第二，“写”这件事也是需要刻意提高的。

就算你工作了十年，但是你每天都在写 hello world，你的技术也不会有任何的长进。

写文章也是一样，如果你只是为了写而写，习惯了在文章中贴大段大段的代码，抱着一种“自己去理解吧”的心态，而不是尝试用最简单的文字去引导和解释，你的写作水平和表达能力永远也不会有提升。

无论是在工作中还是文章里，我都反对不分原由的把文档、代码、或者是《XX 权威指南》扔给某个人，然后让他自己去理解。这就好比你把一本汉语词典递给一个小学生让他自学语文。可以，但很难，也完全没有必要。

无论是在文字还是语言表达上，我习惯把我正在叙述的对象，想象为一名刚入门的程序员，或者只是一名有少许编程背景的 QA 同学，甚至是完全没有只是背景的 HR。**然后思考我应该如何通过最通俗易懂的表述，让他们理解我想表达的概念。**

甚至你还可以铺垫，卖关子，起承转合，控制预期等等，为你的文采添砖加瓦，但这一切都离不开刻意的练习。

这一套训练对口头表达同样有帮助。

第三，别怕不友好的评论

我不知道这是不是大家的问题，但这个是我的困恼。我的文章当然被怼过，我曾经因为被某个大 V 不友好的评论把文章的评论功能关闭了两年。

后来我对这件事的焦虑有所缓解和理解是基于两点：

我们很难通过文章评论的一来一回来表达我们想要表达的观点。别说是文章评论，在公司内部，我们在会议室里交流几个小时都不可能达成一致。在文章的背后，有我没有完全表达的背景、上下文或者大家对一件事的理解不同又没有办法统一。误解太正常了。

虽然“他”评论了，但是并不代表“他”有资格对你进行评论。也就是说，虽然我们有平等的网络地位，在现实生活中我们有平等的政治权力，但并不意味着在某些专业问题上，我们的评论有相同的分量。好比一个人不

敢评论交响乐、油画，却又为什么敢评论脱口秀和相声呢，就因为大家都能说话是吗？你可以发出“地球是平的”的声音，大家也都能听到你的声音，但可能没人会把你的评论当一件事。回到技术文章上，**至少你应该对你的技术文章是有自信的，对于质疑的声音，你要学会尝试去自行判断和理解。**

写技术博客这十年，让我感到最遗憾的事情是，有很多在我刚入行时候关注的觉得有价值的技术博客，作者都早已不再维护了，甚至域名呈现的内容都已经面目全非了。 我还是希望在下一个十年我能继续走下去，对于我个人来说，好奇心在，表达欲就在。

作者：**李光毅**

Thoughtworks 高级咨询师

技术人员如何写一本书?

我在过去的几年中，写了 4 本书。有传统意义上的两本实体书：《JavaScript 核心概念及实践》和《轻量级 Web 应用开发》，还有两本电子书《3 周 3 页面》和《函数式编程乐趣》。当然对我而言，主职工作是软件开发，写作是个副业。

在写作的过程中，有一些有趣的心得。

- 写作本身是一个很好的学习过程（至少是一个驱动你学习的动力）
- 写书非常枯燥，特别是校对的时候
- 写作不会让你变得富有，但是有时候会让你开心（不总是）

写文章 vs 写书

写博客 / 文章和写书还是有很大差别的，一个明显的差异是写文章会比较随意，而且应该尽量保持精简。一篇文章提供一些信息即可，应该尽量远离细节（如果写一篇教程，则另当别论）。而写书则应该尽可能的深入细节，尽可能可以让读者依书自修。

投入与回报

首先要明白的一点是，不要指望用写书来赚钱，至少前 4 本是这样的。粗略的算一下：我的第一本书卖了 3000 册，每卖一本我可以得到 4 元 RMB，一共就是 12,000 元 RMB。而这本书我断断续续写了三年。那是很多个周末，很多个假期，很多个夜晚的付出换来的，如果真正要计算投入产出比的话（纯粹金钱上），这显然是一个毫不合算的事情。

作为一个参考，IBM developerWorks 的投稿，千字 200 元，一般写 5,000 字以内，也就是 800 元 RMB 左右。而要写一篇自我介绍这样的文章，我只需要一天（当然需要数周 / 数月的积累）。12,000 元 RMB 需要写 15

篇文章，如果每周写一篇，不到 4 个月就可以写完，而且写文章比写书容易多了，毕竟篇幅比较短小，易于校对。而且对于大部分开发者来说，固定在一个主题上的难度要比 15 个独立的主题小的多，因为无需特别深入。

所以根据经验，要抱着公益的情怀来写书。也就是说为了让知识更好的分享，让你学习到的先进科学技术来帮助更多的开发者，提高他们的开发效率，让他们可以在周末多休息一天。而至于翻译技术书籍，那基本上就是免费的了，完全是一个公益活动（耗时数月，斟酌字句，推敲表达方式，但是价格极为低廉：千字 60 元 RMB），所以下次见了技术书籍的译者，就多少给他捐点吧，他们才是在为人民服务。

知识的诅咒

“知识的诅咒”是指人们在获得了某种知识之后，就无法想象没有这种知识的情况了。这种现象随处可见，比如一个你到了一个从未去过的陌生城市，遇到以为当地人，然后向他问路。当地人觉得已经说的很清楚了，但是你还是不知道该怎么走。另一个例子是：假设你不认

识泰文，然后你打开任何一本泰文写的小说，你只能依稀感觉到这是一种文字，除此之外你并不能从中获取任何的信息。但是当你学习了一段时间泰文之后，再来看这本小说，之前的那种感受就再也没有了。

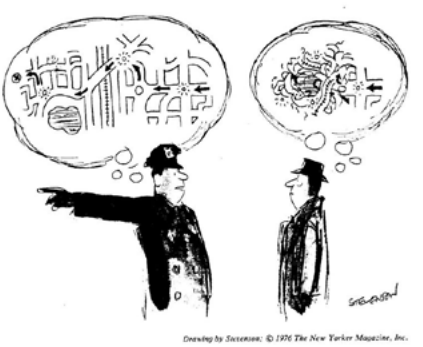


图 : New Yorker Magazine 1976

写书的时候，你首先需要具备某种知识。但是写书的目的是将这些知识传递给那些不具备此知识的人，而根据“知识的诅咒”，你又无法确知那些初学者会遇到哪些问题！解决这个问题的方法就是找初学者来试读。而且为了保险起见，还应该找尽可能多的人来试读。

写作方式

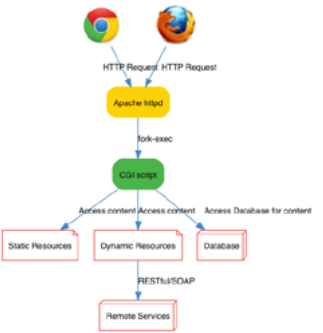
一种方式是自下而上的，写一些独立的文章，最后发现可以串起来，然后形成一本书，另一种方式是自上而下，但是又会逐步调整。根据经验，不论是写一篇简单的博客，还是写一本书，都需要按照自上而下的方式。随心所欲的写下去，基本上都收不住，而且整个文章支离破碎，貌似有很多内容，但是不成章法，读者也无法轻松的获取知识。

先列出大的章节，然后逐步细化，但是未必是按照顺序来写。先编写自己最熟悉的部分，然后逐步完善。例子的选取需要精妙而恰当，最好有图例来说明。

配图制作

一般而言，我在书中会使用两种图：流程图和一些截屏。截屏通常使用 Mac OSX 自身的功能就已经足够，而流程图我会采用一些额外的工具如：

- graphviz
- keynote/sketch



用 Graphviz 画图的好处就是可以将图像代码一样放入版本库来管理。

除此之外，我还学习了一些设计软件的基本用法，事实上只需要用一些简单的元素就可以做出非常专业的配图：

- 字形 / 字体（大小，粗细的变化）
- 颜色（基本的配色理论就可以做出很舒服的配色）
- 层次（尺寸，位置，颜色的深浅）
- 阴影

代码格式

书中实例需要很多代码来说明，如果是制作电子书的话，可以使用 Markdown 预处理器自带的功能来高亮。另外如果需要 RTF 格式，可以使用这些工具：

- highlight 工具
- intelij 中的插件 copy on steriod

一些潜在的坑

在写作的过程中，会有一些潜在的坑。这些所谓的坑是新人可能无法想到的。相对于言之无物，不知如何下笔，最痛苦的其实在于平淡。大部分时候，你可能很容易就能写出开头，但是很难坚持到最后。即使好不容易写完了第一版，后续的重读和修改，会让你苦不堪言。

内容写好之后，样式是下一个重要的问题，好的内容需要有与之匹配的排版。在中国，作者不但要负责内容，还要负责一些排版的事情。这一点非常奇葩，但又是实情。这也是我更推荐电子版的原因（排版更加美观，选择更加多样，而且一旦有问题可以更容易的修改）。

另外一个问题是错别字检查！检查错别字对于作者来说，是一件非常困难的实情。而对于读者来说，则是一件很容易的事情。这跟知识的诅咒的道理一样。

图片来源：网络

发布方式

实体书

传统的出版方式有一些明显的问题，这些问题已经和现代的知识传递方式产生了冲突：

1. 时滞性（新技术的更新速度远远超过审批，印刷等流程的时间）
2. 排版（如何低成本做到语法高亮，或者彩图）
3. 更新频率（当技术更新之后，如何更新，是传统纸质书无法解决的问题）

传统的出版方式有点像传统的软件开发，一本书从开始写作到最终出版，要经过很多环节。忽略掉写作过程，从交稿到出版会经历很多次审核和校对，可能会历时4-8个月，着这个过程中，很多东西都可能发生了变化，一个典型的例子是《用AngularJS开发下一代Web应用》，原版为英文版，翻译成中文版再到出版之后，书中的很大一部分内容已经过时。读者拿到书之后，会发现书中的内容已经和当前的版本 / 文档不匹配了。这种现状随着技术的更新速度和频率还会再加剧。

第二点是排版。我听说国内有些出版社已经开始接受 Markdown 作为稿件的格式，但是大部分还采用 Word 或者 WPS 等格式，这样排版就变成了一个大问题。以我自己为例，我的原稿用 Markdown 写，但是写了几章之后不得不切换到 Microsoft word 上，而我自己的 Mac OSX 下的排版到编辑的 Windows 下就会变样，而且还会涉及字符集，字体，Word 版本等等问题的影响，最后导致印刷出来和原始稿件出入很大。

最后一点是更新频率，如果发现了错别字或者错误的地方（即使之前检查过多次，仍然会有漏网之鱼），由于实体书的特殊性，一般需要等到再次印刷才能解决。这意味着先购买的读者会承担一些风险，更新后的版本又如何让读者看到呢？总不能又买一本吧。

但是这些问题都可以通过电子书来很好的解决。首先，电子书可以随时更新，最低限度的降低时滞性。排版上来说，作者可以使用 Markdown 来编写，而展现则可以应用一些预定义的模板来完成。最后，更新频率完全可以控制，对读者来说风险更低，因为电子版书籍的可以很容易的追踪交易记录，从而得到免费的更新过的版本。

电子书

目前已经有很多的渠道可以发布电子书，比如gitbook，知笔墨。这些应用的出现，大大降低了发布书籍的成本，我的《函数式编程乐趣》，用了 3 天就完成了草稿，而发布只需要数秒。

另外一个问题是书籍的价格和作者的收入。一本书定价 50 元 RMB，出版社给作者的版税是 8%，也就是说，每卖出一本，作者可以得到 4 元，如果你的书非常畅销，这还是一个不错的价格。但是可能 90% 的书籍都不会是畅销书（就好比每个班级都会有优等生，但是他们仅占全班人数的 10% 一样）。这对作者是一种浪费：你需要耗时数月甚至数年来写一本书，然后市场的反馈又非常慢（毕竟你无法出版一本未完成的书）。

我在 selfstore.io 上有两本电子书:《3 周 3 页面》和《函数式编程乐趣》，《3 周 3 页面》定价为 16 元，每卖出一本，扣除掉交易费之后，我可以得到 14.7 元。

对我来说，这样可以得到更多的回报，对于读者则可以更加快速的得到更新，而且由于有预览版和一系列的其

他信息，还可以在很大程度上降低读者的风险（更不用说快递费，等待时间等问题）。我在 gitbook 上的统计显示，《3 周 3 页面》已经被累计下载了 28,861 次，实际的读者也将近 5,000。而且没有任何的审核流程，也没有排版的时间浪费，我只需要关注内容即可。

作者：**邱俊涛**

Thoughtworks 软件工程师、
技术作家

我的第一本英文技术书

缘起

实话实说，写这本书纯属意外之举，最开始的时候对于内容和形式是完全没有计划的，更不用说用另外一门语言作为载体了。应该是在 2017 年年末的时候，我在客户的无线部门做一个网管产品的 Web 化，人员能力提升显然是 Web 化的核心了——毕竟功能需要开发人员一行行去实现。当时该部门中的大部分开发同事的 Web 技能都比较薄弱，一些有经验的同事则每日被业务需求缠身无法顾及能力构建，因此引入外部的培训来填补这个空缺。

这个看起来是 Web 开发 101 的培训还有一些其他需求，比如用户故事拆分、自动化测试和 TDD 等等敏捷开发的基本内容。我为团队设计了一些循序渐进的例子作为

培训的素材，并在后来的几轮迭代中进行了内容的删减和补充。培训的结果还是不错，有一半以上的同事对 Web 开发产生了兴趣，有人则开始意识到自动化测试（以及 TDD）可以减少回归测试的工作量。

正如你所料，这个培训的一个副产品就是这本书的原型了。在培训结束之后我正好有了几周不在项目的时间，我用这些时间将内容变成一个教程：从一个简单的例子，逐步完善成一个前端应用。其中涉及了诸如 Tasking，ATDD，用户故事等。

写作过程

初稿

原型是很容易的，素材是现成的，只需要将其以符合逻辑的方式连接起来即可。比如我想要描述通过 TDD 的方式开发一个应用的全过程，那么首先我需要讲清楚我们要做一个什么应用，此外我需要说清楚如何用 TDD 来完成这个应用。根据经验，人们喜欢循序渐进的方式来阅读，先做一个简单需求，然后逐步增强，并在过程中将学到的内容应用，然后推广到后续需求的开发中，并通过实现更加复杂需求习得我想要传递的知识/技巧。

- TDD 的 101（通过例子来说明 Tasking 和测试驱动的过程）
- 介绍 demo 应用，搭建应用所需的环境
- 设计并实现一个简单需求
- 实现一个更复杂的需求
- 循序渐进……

编辑初稿的过程中，正好有个编辑联系我有没有意向写个小册子。虽然后来由于篇幅和主题的选择问题没有继续合作，但是和编辑沟通的过程帮助我很好的梳理了草稿的结构。

英文版

在草稿写完之后，我找到几个同事做了快速的评审并做了一些修改。到了 2018 年 3 月，我开始准备到澳洲工作，既然英语会是我接下来几年的主要语言，那么提升英文能力就成了优先级很高的事情。而学习语言的窍门就是:**没有什么比实际使用一门语言更能提升语言能力的了**。

于是我开始尝试把草稿翻译成英文版。开始的时候我还会保持双语版本的同步，到后来带宽不足的时候就只能把精力放到英文版上了。第一个完整的英文版发布于 2018 年 5 月，后面虽然陆续有些小的修改，主体部分没有太大的变化。

到了澳洲之后，读了很多英文原版的书籍 / 文章之后，发现了书中很多的语言错误，又在一个假期里集中修改过一轮，可读性有了一些提高。

重构

到了 2019 年的 9 月，经过了近 1 年 2 个纯 React 的项目后，我对与 React 及其生态的理解，以及在应用 TDD（特别是和传统上认为的 TDD 很难在复杂 / 时间不允许的项目上实施）的认识上都有了新的发展。于是又将其中很多章节重写了，比如丢弃了 pupeeteer 改成 cypress，采用 react-testing-library 而不是 enzyme 等等。另外，结构上也做了重新整理。

由于这些颠覆性的修改，我将其重新命名为 Mastering Test Driven Development with React，并声明其为前一个版本的第二版。

到了 2020 年 4 月，由于 COVID-19 在澳洲的肆虐，我们开始全面 WFH。一方面突然有了很多时间，另一方面我意识到线上沟通时英文在有些时候还是会变成障碍，于是我又花费了一些时间来提升英文。于是又开始了一轮的 re-wording。

在你今天阅读这本书的时候，你大约会惊叹于英文表达之地道，语法之准确以及用词之精准。这些都和我的英

文水平没有太大关系。在 5 月的一天，一位澳洲同事写邮件给我，表示她通读了这本书，非常喜欢其中的讲述方式和实例以及节奏，她表示愿意帮我近一步润色。她通过 PR 的方式重新整理了本书的语言（目前已经完成了 70%），由于她本身就是前端 Dev，所以很多表达方式也被重写为更容易被读者理解的方式。

内容调整

一些收获

关于英文写作

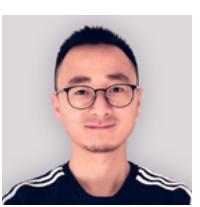
语言的学习是个漫长而痛苦的过程，同样的内容，要从一门语言中的表述翻译成另一门语言需要的更多的是重写而不是 literal translation。从到澳洲工作开始我就

开始刻意的用英文写作，还尝试把一些文章翻译并投稿给 Thoughtworks 官网的英文版洞见。

通过这些练习，我觉得我至少不再惧怕这件事情本身。英文表达当然有不地道的地方，但是读者也不是语言学家，大多数时候他们都可以准确无误 get 到你的意思。即使有些复杂概念无法一次理解，通过评注或者提问等等，总是会搞清楚的。

关于自信构建

在这本书的写作过程中，我个人最大的收获应该是：当你制定了一个目标，不论这个目标开始开起来有多么的不切实际，一旦你开始细化这个目标并逐步实施，你就已经离这个目标不远了。当然，和每个任务一样，事情走到最后可能会和最开始的目标并不完全契合，但这大约是我们无法掌控的那部分了，就随他去吧。



作者：**邱俊涛**

Thoughtworks 软件工程师、
技术作家

用什么工具写书

用好工具

开始正式写作，要选择好一个好用的工具，常见的工具有：

- Word：相对比较容易掌握，所见即所得，但较难进行版本管理。
- Markdown：程序员用的较多，纯文本，有很多可视化工具，方便进行版本管理。
- Latex：科研论文用的较多，纯文本，能够做出很专业的文档，但学习成本、配置成本较高。

为了减少工具对写作的负担（包括录入、调整格式等干扰），平衡了利弊后选择以 Markdown 作为初稿的编辑工具。当初稿写完需要进行排版的时候，再提交给编

辑一个 Word 文章进行后续的编辑。

一本书大概有 30 万字左右，会分为多个章节。大多数人都会拆成多个章节来处理，每个章节一个文件。但我尝试了一下发现，单一 Markdown 文件就非常利于管理：

- 方便预览整书情况
- 方便章节之间相互引用
- 方便导出

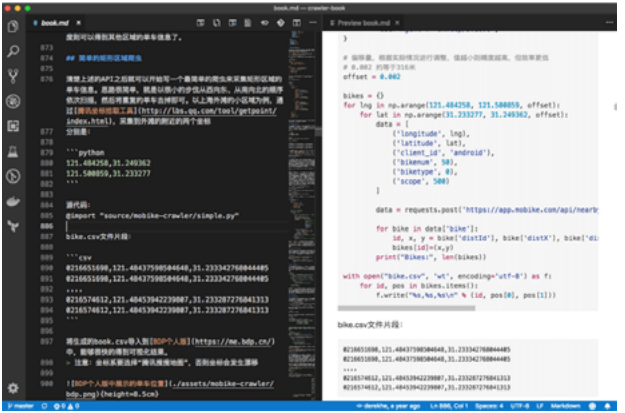
在工具的选择上，我在 Visual Studio 中安装了 Markdown Preview Enhanced 插件，这个工具很好：

- 自动预览
- 自动生成目录
- 自动内嵌图片、内嵌代码
- 能够导出成多种格式（Word 格式、PDF 格式等）

对于一本需要引入大量代码的文章来讲，代码的正确性

是很重要的。在这个插件中，增强的 Markdown 语法中中可以直接引用代码。这样我更改了代码以后，书中的代码也更新了，避免了不同步的问题。

整本书从头到尾都是用 Git 管理的，这也是程序员标准的工具，就不用多说了。



图：使用 Markdown 编辑书籍

迭代交付

迭代交付在敏捷项目的成功中扮演者很重要的角色。写一本书也可以使用迭代交付：每一个章节作为一个迭代，

每个迭代大概一个月左右。从开始写作开始，一鼓作气，坚持每天花时间写，避免拖拉。每个迭代完成后，将写好的文稿第一时间给编辑进行预览并提出修改建议。

在迭代的交付过程中，针对本书的性质，还是做了一些相应的修改以符合法律法规的要求。比如原书中有一些地图相关的图片，在 2018 年国家新的地图管理中，地图是必须要相关机构审核才能在书中出现的。为了避免审核的时间消耗，我对有相关图片的内容进行了一定的删减。得益于迭代交付，在书稿早期，就对内容进行可刻意的控制，避免了大规模返工的情况。

图：删减地图底图

总结

经历过原创书籍的写作，我觉得写一本技术书看似很难，但如果在平时做好了素材的积累，相对还是比较容易的。在写书的过程中，选择称心如意的工具以及一鼓作气的作风，会加速写书的进程。迭代交付的思想会拉近与编辑的距离，给到最及时的反馈，避免最后时刻无法交付的尴尬。

作者：**贺思聪**

西门子软件架构师

写作的排版、发布和推广

通过知识输出的方式去学习是更高级的终身学习方式。我们可以通过写博客、写 Tweet、社区回答问题及录制视频等诸多方式去输出知识。

写作的意义

- 写作是深度终身学习：当你阅读时，你是被动接受作者的观点，你的大脑并未全面调动起来去分析问题。而一篇文章，需要你耗费很多时间查阅资料，甚至数年的经验总结。当文章发表后，可以与读者互动而相互学习，这种学习是一种深度学习；

- 写作是高质量的社交：什么叫无效社交？在群里灌水很难让别人信任你。而一篇好的文章会让读者与作者建立更深度的连接。作者可以通过文章去影响别人，如果这种影响是积极的，那这篇文章就很有

价值与影响力；

- 写作能创造新的机会：在我写作的一些文章里，有一篇文章让我找到一个新领域的工作机会，有一篇文章让更有经验的人与我合作，有一些文章让别人信任我并给我推荐新的合作机会，有一些文章让我认识更多全国各地的朋友，这些都是无法用流量（阅读量）价值来衡量；

如何写作

写作最难的是开始动笔写，当你开始去写并且能坚持，这已经算解决了最主要的问题。如果你还犹豫是否要开始写，可以考虑下我上面介绍的写作的意义。

我一般会在日常阅读或思考时收集一些我可写或感兴趣要学习的主题，先把一些关键点记录下来，然后在有时间时选一篇开始动笔写。一般一篇文章有时候会在一小时写完，有时候会耗费几天甚至一周的时间去写。

在写的时候，我先会搜索下这个题材有没有人写（基于 SEO 的方式去写作），如果有的话大体有哪些热门的文章，先把这些文章简单浏览一遍。之后耗费半小时去罗

列提纲，如果这个题材刚好在我的笔记有记录片段信息的话，我会把这些信息也给复制进来。然后开始思考是否需要绘制一些配图，如果需要绘制的话，先开始绘制这些配图，具体的绘图工具可以参考我的网站文章 我的绘图工具箱 。

之后开始进入写作流程，一般开始写整个人是懵逼的，很难进入状态。这时我会戴着耳机不断重复某首很熟悉的旋律歌曲，这种无限循环的歌曲可以让我大脑很快进入一种写作的心流状态。

在写作正文的过程中，基于 SEO 的写作模式，我会考虑是否可以与我之前写的文章建立一些链接关系。譬如这篇文章里有一些我之前写过的文章，这样可以把这些分散的文章串联起来，也会让读者对该主题了解的更全面一些 (迫使他在你的网站上停留更长时间) 。

写作排版

排版规范可参考 中文文案排版指北 。

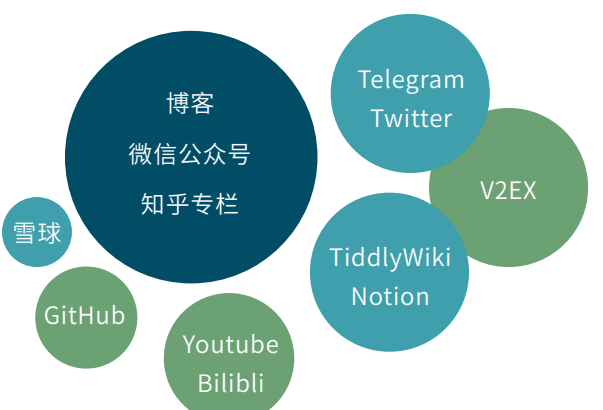
也可以使用 赫蹏 这个库自动对内容进行优化排版。

如果使用 VSCode 编辑 Markdown 的话还可以安装 Pangu-Markdown-VSCode 插件，在写完文章后一键自动排出上述规范要求的效果。

文章配图可以使用 TinyPNG 压缩后上传，这样不仅可以使网页加载速度变得更快，还可以降低你 CDN 和用户的流量费用。

如果你对中文排版设计感兴趣，还可以看 W3C 的中文排版需求。

写作渠道



我在 零成本搭建现代博客指南 这个系列里写了几篇关于如何搭建个人博客的系列文章，感兴趣的可以看看。

博客 / 微信公众号 / 知乎专栏

对于长文，我的发布平台是本博客、微信公众号及知乎专栏。

一般正文写完后，在 Unsplash 上找个封面配图，之后 一键推送 到 GitHub，自动触发发布流程，文章就上线了。

知乎专栏在博客文章上线后，直接复制粘贴到知乎专栏编辑器就可以了。而微信的排版是用 Markdown 原文粘贴到 微信公众号在线排版工具 ，然后复制到微信公众号编辑器，所有配图还得单独上传上去。

Twitter/Telegram

Twitter 里我一般会发一些简单的短文。也可以选择在长文里复制一些比较精彩的单独发布到 Twitter。

Telegram 里我创建了三个不同主题的 Channel：一个

关于 自我成长 ，一个是关于 被动收入 ，一个是关于 流量研究 。

Telegram 里的玩法比较多，这三个主题的 Channel 里的消息通过机器人转发又会自动汇合至 学习 & 技术 & 投资 这个电报群里。在这三个 Channel 里发的消息也会自动通过 Twitter 发布出去。考虑到 SEO 及某些神秘的网络原因导致无法访问的需要，我又创建了一个 电报频道静态化工具 ，这个工具可以自动将三个 Channel 里的消息自动发布到我的博客这个 思考 页面里。

Youtube/Bilibili

视频的制作成本很高，在制作了几次质量一般的视频后，我暂时放弃了视频类的发布。目前还是写文字为主，未来等找到低成本制作视频的方式后，我可能会继续尝试以视频的方式输出知识。

写作推广

文章写完并发布后，最头疼的事情在于如何推广：**推广就是让别人知道你写了篇牛逼但还没有人看的文章。**我一般选择去微信群、朋友圈、V2EX、Twitter 及

Telegram 里写一段简介（一般是网页描述标签的内容），然后附带文章链接发布。之后看别人是否吐槽还是点赞，然后与他们（可能愉快）的交流下文章心得。当然如果没人响应你也无所谓，不要气馁，继续坚持写下去，没人看不代表你的文章质量不好，很可能是取了一个令人无感的标题或者选题过于缥缈导致曲高和寡。

在推广时，如果有好的配图记得附上，比如在 Twitter 上你可以把文章里精彩的配图附上，好的配图比标题更吸引人。

作者：**马大伟**

Thoughtworks 高级咨询师

翻译这件小事

在前辈们的牵线搭桥之下，我非常机缘巧合地参与了两本书的翻译，虽然加起来 10 多万字，远远未到“足以谈翻译这件事”的地步，但还是希望在本文中从经验的角度分享出一些真实简单的感受，给想要入坑的伙伴们一些参考。

翻译的目的和价值是什么？

答案肯定因人而异。就我本人来说，一开始是误打误撞纯粹图个名，后来发现翻译几乎就是比精读更精读的阅读一本书，你需要了解作者提到的各种术语，知道举证案例背后的事实，有的时候为了一句话，查资料越跑越远，像是发现新大陆。

在这个过程中，且不论最终翻译的质量如何，一定会对原作的每一句话形成深刻的理解，因为你需要不断去思

考作者为什么这么说，才能保证译文的准确性。至于花上这么长的时间精读一本书是否值当，那就取决于这本书本身了，所以这里首先建议有尝试意向的人，一定要选择自己感兴趣的领域内容，这对于克服懒惰这个天敌意义重大。

“有了 Google Translate，还需要人工翻译么？”在机器学习技术红起来之后，很多人对翻译这件事情的价值理解就打了个更大的折扣，如果从前人们还相信书籍的翻译出版有“信达雅”的标准，现在大多数人估计认为翻译无非就是把 Google Translate 的译文随便调整一下，能通畅地表达就行了。其实没错，还真就是这样的。

从某种意义上来说，Google 翻译确实大大拉低了翻译的门槛，这倒不是因为翻译的要求降低了，而是因为机器翻译大大提升了翻译的效率。翻译一直以来都是一门对技术敏感度非常高的学科，计算机辅助翻译（Computer Aided Translate，简称 CAT）的技术发展已经超过三十年，翻译实践很早就从纯人工行为转向了人工和信息技术相结合的策略。

随着机器学习技术的兴起和语料库的不断扩充与完善，机器翻译已经成为译者倚赖的工具，这没有什么不好意思承认的，发明和改善工具，原本就是为了提高生产效率。当然，这里不打算讨论机器翻译是否会完全替代人工翻译这个话题，写这篇博客的目的还是为了说回翻译这件事情本身。

翻译是什么？

通常我们所讨论的翻译，就是将某一语言的言语产物转换到另一语言当中。更广义来说，它还可以包括语言和非语言符号之间的转换，比如转换为手语，以及一种语言中不同变体之间的切换，例如从文言文到白话文的翻译。但是究其根本，翻译的核心终归是在译文中完整准确地表达原文的意思。如果将人工翻译的过程拆解开，那就是阅读原文、理解原文、输出译文。

理解原文

关于翻译的理论研究成果非常丰富，关于翻译实践的技巧也很多，但是这些无一都是聚焦在“输出”这个环节上的。**而实际上，对原文的理解才是翻译的源头。这一**

点有多重要以及有多难，从事文学翻译的译者最有发言权。

我们通常接触的原著作一般是非文学领域的，例如技术书籍或是商业分析等等。这类比较偏标准化文本的著作，出于其本身的写作目的，通常都是为了说清楚某个问题、介绍某种方法，或是表达某个观点，大体都是表意清晰客观，语言规范且较少存在歧义的。在上下文和逻辑的帮衬下，要准确理解原文并不会太难。

如果碰上句式结构复杂，拿不准断句和原意的时候，Google Translate 就可以派上用场了。以英译中来看，谷歌翻译的译文虽然无法直接拿来用，但是不得不承认，机器学习输出的译文对原文的理解准确度是非常值得肯定的。并且，谷歌翻译对专业术语或是一些惯用表达的译语，很大程度上可以帮助提升效率。

也千万切记，Google Translate 的释义并不是完全 make sense 的，所以一定要不断联系上下文来看整体表达是否切意，对不通之处要寻求各种帮助来弄懂。如果译者自己都没有弄明白作者在说什么，只是语焉不详

地照搬原表达，不知何以然，那么读者的混乱程度只会更高。

输出译文

准确理解原文之后，接下来就是译语的转化了。如果说上一个阶段要求的是对原语言的理解力，那么这一阶段需要的就是目标语言的写作能力了。写作能力越高，对目标语言的掌控力越高，在理解准确的前提下，当然也就可以更为精准地还原原文的意思。是的，道理大家都懂，可是写作能力这件事儿不是看几本书就能立刻得到提升的，这是一个需要不断练习和打磨的技能。大家可以参加一些写作工坊，尝试“21 天持续写作”等方式，来持续提升自己的写作能力。

直译还是意译？

进行译文创作（如果翻译这件事情真的存在“创”作的话），永恒的问题就来了，到底是选择意译还是直译？

一般来说，比较遵照原文语言结构的译法就是直译，脱离原文语言结构的束缚，只翻译意思的译法就是意译。逐字翻译、直译、意译和解释翻译之间并没有非常清楚

的界限，但是四者的顺序反映出了一个自由度提升的过程。直译好还是意译好，其实并没有一个高下之分。不同的人就有不同的翻译风格，直译能保留更多原作的味道和比喻，但是经常被人吐槽“翻译腔”，其实就是表达方式过于西化；而意译就像滤镜，对于目标语言的融合度更高，更符合读者的阅读习惯，但是也会丢失更多原作的风格，读者离原作也就更远一些。

直译和意译的选择并非绝对，对于非文学翻译来说，尤金奈达的“功能对等”的理论非常适合。我们的目的是将原作的知识和信息准确、清晰地传达给目标语言的读者（以读者为中心），首先是重现意义，然后是风格。那么我们能得到这样的优先级排序：准确、易读 / 易懂、尽可能还原作者的风格（例如幽默）。如果直译的结构十分不符合中文的表达习惯，那么我们应该在保证意思准确的前提下，选择意译。

一些实用的小 Tips

这里有些简单的技巧，可以为我们在处理特定结构的表达时提供参考：

1. 不要受困于词性

对词性的敏感（名词、动词、形容词等）在翻译过程中很容易束缚住表达。因为词性的概念建立在语言的表层结构上，而英汉两种语言的表层结构恰好差别很大。执着于词性的对应会严重影响意义的表达，例如：

- It took a long Presidential drive to get them talk again. （在总统不顾路途遥远，驱车前往调停后，双方才恢复了对话。）

这句话的上下文是叙利亚外长和以色列总理访美期间谈判陷入僵局，美国总统前往调停，译语里改变了 long, Presidential, drive, talk 等词的词性。为了确保可读性，一定要跳出词性在英语学习过程中建立起的牢固概念。特别是使用 Google Translate 辅助时，机器翻译大体还是遵从了词性的对应，译者必须有意识地进行调整。

2. 拆解句子结构，分合移位

切分、合并、移位是翻译中很常用的方法。目的是将原语言更自然地用目标语言表达出来，可以对原文进行切分、合并或是调整词组 / 从句的顺序。

- My father was not wrong in judging me too young to manage business of importance. （我父亲认定我太年轻，办不了大事。他算是没说错。）

原本很简单的一句话，译成中文后成了两句，但是表达的层次更清晰也更自然。再比如 Thank you for your advice and counsel 译为“谢谢你的忠告”，因为两个词实际是一个意思，重复只是一种语言手段，翻译可以考虑合并。

如果想要表达更通俗易懂，可以尽量使用短句。短句是否一定就是好，这点尚存争议，也取决于译者风格。句子越短，意味着对原文的拆解越彻底，译者发挥的自由度越高，也更有利于读者的理解与阅读，比如本段的表达。但与此同时，这种译法表达错误的几率也偏高，且更容易损失原文的味道。

3. 被动语态的处理

被动语态是英语的惯用表达，例如 I was told that…中文里虽也有被动结构，比如上面这句话就可以翻作“我曾被告知……”，但是实际应用场景中，除非特意强调，例如“他被捕了”，通常并不会颠倒主谓宾。我们需要意识到，大多数情况下，被动结构本身并没有什么意义。特别是在技术类书籍中，因为没有特定的主语，所以被动结构成了常规表达方式，翻译时什么时候应该转为主动，什么时候应该沿用被动取决于译者当下的判断，并没有直接的判断标准。可以注意以下两点：

- 被动转主动时，汉语中可以不需要主语；
- 即便使用被动语态，被字有时也是可以省略的，例

如“广告随处可见”；就算无法省略，汉语中还有其他助词可以表示被动，例如：受、遭、让、叫，等等，尽量避免“被”字连篇。

4. 增词不增意，删词不减意

增词通常出于两种目的: 一是用更多的词解释清楚原意，例如前面提到过的 long Presidential drive；二是增加上下文之间的衔接词，比如“而且”、“所以”、“于是”等等，使表意更通畅。

减词的目的是改善汉语行文，例如 He shakes his head 就没有必要翻译为“他摇着他的头”，“他的”两个字就可以去掉。英文中因为严格的语法结构，其实很多代词、连词和介词都不需要逐词翻译出来，减掉不影响表达的词是保证汉语简洁的常用手段。

所以不论增词还是减词，目的都是使表达更符合汉语的表达习惯，而不是对原文的意思进行增减，在保证含义准确的前提下，译者自己可以把控这个度。

5. 校对

翻译完成后一定要自己校对一遍！这点很重要！很多在

翻译时觉得挺正常的表达，在最后校对时会觉得完全不通。校对一般脱离原稿，这时的目的除了修改错别字以外，最重要的就是调整表达。把自己当作一个陌生的读者，看译文是否达意。

写在最后

总结来说，对于偏标准化的专业类书籍，翻译没有所谓的门槛，是一种人人都可以尝试的学习方式。最大的障碍只有懒惰和不求甚解（其实是一回事）。马上又要到大量书籍等待认领的时候了，分享一些个人建议：

1. 正如前面提到过的，选择的原著最好是自己感兴趣的领域；
2. 翻译的效率高低与专注度正相关，因此翻译的时间尽量不要打得太碎；
3. 针对专业书籍，最好创建并维护一个术语表，保证译语统一，特别是多人合作的情况下；
4. 如果原作的格式较多，推荐使用 markdown 编辑器，Ulysses 或者 MacDown 都不错；
5. 魔鬼都在细节里，翻译过程中这点尤甚。格式、注释、

术语等等细节问题尽量不要积压到最后一起处理，每一次的任务结束都尽量保证完整性和准确性，因为最后校对阶段极有可能不会再逐句对照原文检查。

最后，引用张玳老师在引荐《精益扩张》这本书的翻译工作时说过的话：“翻译是比较累的活，经济收入也不成正比，但是能带来名誉，锻炼语言，也可以磨练心志。”希望所有正在这条路上受苦受累的人，最终都能有所收获并乐在其中。

作者：**王静源**
Thoughtworks 首席咨询师

《风格的要素》中的编程格言

写在前面：

之所以翻译这篇文章，因为它与我之前关于如何 / 为什么要写出更易懂的代码的思考有很多关联，但更进一步的是，这篇文章直接指出了写作与编程在原则上的相似性，本质上他们都是一种面向人类的表达。

其次，本文的英文原文博客虽然已经关闭了，但在英文语境下的人们依然对这篇“著名”文章议论纷纷（Reddit, Medium, Ycombinator），一部分原因是本文中提到的 The Elements of Style / 《风格的要素》这本书在英语世界里举足轻重，加之绝大多数编程语言都是以英语为基础，人们对于这种表达的相似性理解起来会更加自然；相对在中文语境下，鲜有人从英文写作这样的角度出发解释这种清晰表达是一种基础能力的重要性。

最后，《风格的要素》也是非常值得一读的书，对如何写出更好的文章有所帮助。

The Programming Aphorisms of Strunk & White

《风格的要素》中的编程格言

It's been the bane of English students and the bible of creative writing teachers since the second World War. It's older than you are, meaner than you are, and presumably, built from sterner stuff than you are. This book has survived air raids and artillery shellings. Its simple lessons of thrift and economy have denuded the world of more bad writing, over the last 70 years, than the shutdown of Google Reader and Geocities combined. That is why The Elements of Style, by William Strunk and E.B. White, is a classic, and that's why you'll find a copy of this book (no thicker than the width of your finger) ensconced firmly in the "software" section of our bookshelf.

自二战以来，它就是英文学生的祸根和创造性写作教师的圣经。它比你更老，比你更刻薄，而且很可能比你更坚韧。这本书从空袭和炮火连天中生存下来。在这七十多年中，它关于精简节约的训导清理了这世界上太多的烂文章，甚至比关闭 Google Reader 和 Geocities 加起来还要多。这就是为什么这本由 William Strunk 和 E.B. White 所著的《风格的要素》如此经典，甚至你会发现，这就是为什么这本比你手指还要细的书的一份副本竟被我们摆放在书架关于“软件”的区域。

Of course, “Strunk and White”, as the book is usually called, has nothing to do with software (it was originally written in 1920) and everything to do with writing: grammar, composition, and style for users of the English language. But in its 100 pages this book has more to say about the craft of software than many books you’ll find in the “Software” section of your local bookstore.

当然，这本经常被称为 “Strunk and White” 的书，确实跟软件没什么关系（它最早写于 1920 年），它所

有内容都是关于写作的：语法、行文、和英文使用者的风格等。但在这 100 页中，这本书其实讲了很多关于软件技艺的道理，比你从你那的书店里“软件”区域中找到的许多书还要多。

2.12. Choose a suitable design and hold to it.

A basic structural design underlies every kind of ~~writing~~ programming. ~~Writers~~ Programmers will in part follow this design, in part deviate from it, according to their skills, their needs, and the unexpected events that accompany the act of composition. ~~Writing~~ Programming, to be effective, must follow closely the thoughts of the ~~writer~~ programmer, but not necessarily in the order in which those thoughts occur. This calls for a scheme of procedure....in most cases, planning must be a deliberate prelude to writing programming. The first principle of ~~composition~~ software development, therefore, is to foresee or determine the shape of what is to come and

pursue that shape.

2.12 选一个适当的设计并保持下去

每一种 写作 程序都有一个基本的结构设计。 作家们 程序员们一方面力图遵循这种结构设计，一方面又会偏离它，这完全取决于他们的技巧、需要以及创作过程中出现的各种意外事件。写作 程序要写得好，必须紧紧地顺着 作者 程序员的思路，但并不一定完全按照这些思路出现的先后顺序。这就需要一个规划……在大多数情况下，创作 编程之前必须要有一个认真考虑过的计划。因此， 写作 软件开发的第一原则就是要预先考虑或确定将要写成的模样，并尽力为之。

A casual search-and-replace for “writing” with “programming” consistently yields paragraphs that might have been plagiarized, whole cloth, from Code Complete. Paragraphs like this one, which cautions the would-be writer programmer to design appropriately for the scale and complexity of the thing being built.

只消把“writing“全都替换为”programming“，这些段落就变成好像是从《代码大全》中抄袭改编的一样。像这样的段落，它提醒那些想成为作家的程序员适当地设计正在构建的东西的规模和复杂性。

5.3. Work from a suitable design.

Before beginning to ~~compose~~ develop something, gauge the nature and extent of the enterprise and work from a suitable design. Design informs even the simplest structure, whether of brick and steel or of prose. You raise a pup tent from one sort of vision, a cathedral from another. This does not mean that you must sit with a blueprint always in front of you, merely that you had best anticipate what you are getting into. To compose a laundry list, you can work directly from the pile of soiled garments, ticking them off one by one. But to write a biography, you will need at least a rough scheme; you cannot plunge in blindly and start ticking off after the fact about your subject,

less you miss the forest for the trees and there be no end to your labors.

5.3. 从一个适当的设计着手

在开始 创作 开发某些东西之前，先衡量整件事的性质和范围并从一个适当的设计着手。设计是赋予哪怕最简单的结构以意义的，不论砖或钢或散文。你可以由某种想象搭建起小帐篷，也可以由另一种愿景建造一个大教堂。但这并不意味着你必须时刻坐在图纸前，它只是意味着你有了一个对于你着手要做的事情的最好预测。要撰写一个洗衣清单，你直接从一堆脏衣服开始，一件一件地勾掉即可。但要撰写一篇人物传记，你不能盲目莽撞地一开始也把要写的人物事迹一件一件地勾掉；你至少得有一个粗略的计划，以免见树不见林，最终劳而无功。

Or this, about the dangers of pet languages, non-standard libraries, young people, and drums. Chase it with a shot of Sailing to Byzantium for good measure.

还有这个，关于不成熟的语言、非标准库、年轻人和鼓噪声的危险。追求这些东西配点烈酒效果更佳。[译者注: Sailing to Byzantium 可能指一种爱尔兰威士忌，作者这里的意思是不要被这些东西冲昏了头脑]

5.21. Prefer the standard to the offbeat

~~Young writers~~ Inexperienced programmers will be drawn at every turn toward eccentricities in language. They will hear the beat of new ~~vocabularies~~ abstractions, the exciting rhythms of special segments of their ~~society~~ industry, each speaking a language of its own. All of us come under the spell of these unsettling drums; the problem for beginners is to listen to them, learn the words, feel the vibrations, and not be carried away.

5.21. 宁可标准不要另类

~~年轻的作~~家们 没经验的程序员们每一次都会被语言中的怪癖所吸引。他们听说了新奇的 ~~词汇~~ 抽象，像鼓点

一样，~~社会~~ 行业中特定领域令人亢奋的节奏，每个人各说各话。我们所有人都忍受这些令人不安的鼓声的咒语。对于初学者来说，问题是要倾听、吸收那些词语，感受其脉搏，不要被带跑。

Sometimes the software interpretation rings even truer than the original meaning of the phrase:

有时候这种软件解释甚至比文本的原意更真实：

2.17. Omit needless ~~words~~ code

Vigorous ~~writing~~ programming is concise.
A ~~paragraph~~ function should contain no unnecessary ~~sentences~~ statements, a ~~sentence~~ statement no unnecessary ~~words~~ expressions, for the same reason that a drawing should have no unnecessary lines and a machine no unnecessary parts. This requires not that the ~~writer~~ programmer make all ~~paragraphs~~ functions short, or avoid all detail and treat subjects only in outline, but that every ~~word~~ line of code tell.

2.17. 去除不必要的 ~~词汇~~ 代码

生动有力的 ~~写作~~ 程序需要简洁明了。一幅画上不应该有多余的线条，一台机器不应该有多余的零件。同样的道理，一个 ~~段落~~ 函数不应含有不必要的 ~~句子~~ 语句，一个 ~~句子~~ 语句不应含有不必要的 ~~单词~~ 表达式。这并不是要求 ~~写作者~~ 程序员要把所有的 ~~段落~~ 函数写得很短，或者避免所有的细节只把主题当作主线；而是说，每 ~~个字~~ 行代码都表达了本意。

2.19. ~~Write~~ Express coordinate ideas in similar form

This principle, that of parallel construction, requires that expressions similar in content and function be outwardly similar. The likeness of form enables the reader to recognize more readily the likeness of content and function.

2.19. 用相似的形式 ~~写出~~ 表达对等的意思

平行结构这个原则要求内容及功能上相似的表述要表现地相似。这种形式上的相似性使读者能够更容易识别出功能和内容本身的相似性。

Sometimes the attempt to replace writing with programming yields new insights.

有时候把写作替换成编程还会产生新的见解。

5.9. Do not affect a breezy manner

The volume of ~~writing~~ source code is enormous, these days, and much of it has a sort of windiness about it, almost as though the ~~author~~ programmer were in a state of euphoria. “Spontaneous me,” sang Whitman, and, in his innocence, let loose the hordes of uninspired scribblers script kiddies who would one day confuse spontaneity with genius.

5.9. 不要肆意妄为

如今的 ~~文章~~ 源代码篇幅都很长，而且其中不少是夸夸其谈的，仿佛 ~~作者~~ 编程的人得了欣快症似的。Whitman 赞颂“不由自主的自我”，却无意地怂恿了一批 平凡拙劣的作家 毫无创意的脚本小子，总会把自然的冲动当成天赋的才华。

What does it mean to have a “breezy” style as a programmer? A plausible definition might be coding things in the order they occur to you, or to the degree they interest you. It’s a great recipe for spaghetti or, for the true code gourmand, mud. But without being “breezy” it’s also important to:

一个程序员“肆意妄为”是什么意思？一个可能的定义是，想到什么就写什么，或者取决于对它们感兴趣的程度。这其实就是产生意大利面条式代码的原因 [译者注：意大利面条指一种很难维护的代码风格]，而对于真正的代码美食家来说，那就是一滩浆糊 [译者注：很容易让人联想到大泥球]。同时，尽管不能肆意妄为，仍需注意的是：

5.2. ~~Write~~ Program in a way that comes naturally

~~Write~~ Program in a way that comes easily and naturally to you, using ~~words and phrases~~ APIs and statements that come readily to hand. But

do not assume that because you have acted naturally your product is without flaw.

5.2. 文章 程序要写得自然

文章 程序要写得自然流畅，要使用顺手易用的 ~~单词和句子~~ APIs 和语句。但不要以为只要顺畅自然就可以写出毫无瑕疵的作品。

The act of writing code is a simultaneous conversation with two different audiences: the compiler/interpreter, and the other programmers on your team, including the future version of yourself. Good conversationalists—good coders—know how to speak to both audiences naturally and without affect. They know how to achieve clarity...

编写代码这件事其实是在同时与两个截然不同的听众对话：一个是编译器 / 解释器，另一个是你团队中其他的程序员，包括未来的你自己。会聊天的人——优秀的编码者——懂得如何同时与这两个听众顺畅自然地对话而

不被影响。他们懂得如何做到清楚明了…

5.16. Be clear.

Clarity is not the prize in ~~writing~~ programming, nor is it always the principle mark of good style. There are occasions when obscurity serves a literary programmer yearning, if not a purpose, and there are ~~writers~~ programmers whose mien is more overcast than clear. But since ~~writing~~ programming is communication, clarity can only be a virtue.

5.16. 要写得清楚明了

清楚明了并不是 ~~写作~~ 编程的目标，也不总是良好风格的原则。有时候晦涩朦胧可以满足程序员自己的某种情结，即使那并不是最终目的，并且有的 ~~作家~~ 程序员的风格更是风云变幻而非清晰明朗。不过既然 ~~写作~~ 编程是一种交流，那么清楚明了便是美德。

…without being verbose:

……不要太啰嗦：

5.11. Do not explain too much.

It is seldom advisable to tell all. Be sparing, for instance, in the use of ~~adverbs~~ code comments… Let the conversation code itself disclose the ~~speaker's manner or condition~~ coder's intention.

5.11. 不要解释过多

把话说尽是不可取的。比如尽量少用 ~~副词~~ 代码注释……让 对话 代码本身来显现 ~~说话者的态度和状态~~ 编码者的意图。

When we put on our "Bad Programmer" hats, on the other hand, we may not even be aware that the second audience, the human audience, exists. If we are aware it exists, we consider it secondary. In this state of temporary insanity, we might forget that, while part of our job is to talk to the compiler, the interpreter, and the toolchain, the bigger part of our job is to talk,

through code, to our team members, maintainers, and stakeholders. **Humans are the ultimate and primary audience for code.** In that light, Strunk & White takes on new meaning and new relevance not just for coders, but for builders and creators of any stripe.

另一方面，当我们戴上“坏程序员”帽子时，我们甚至可能都不会意识到第二听众，即人类听众的存在。即使意识到了，我们也只是认为它是次要的。在这种短暂的失智中，我们也许忘了：尽管我们一部分工作是在与编译器、解释器和工具链交谈，但更多部分的是，通过代码，与我们团队成员、维护者和利益相关者们交谈。**人类才是代码的终极听众。**从这个角度来看，《风格的要素》不仅对写代码的人而且对各行各业的建造者和创造者都具有新的意义和联系。

翻译参考：

- 斯特伦克，怀特，& 陈一鸣 . (1992). 英文写作指南 .

上海 : 上海译文出版社 .

- The Programming Aphorisms of Strunk & White | Coding the Wheel. (2015, February 23).

扩展阅读 :

- Coding: It’ s Just Writing. (n.d.). Retrieved July 1, 2019.

作者：**董强**
Thoughtworks 首席咨询师

技术写作的困境

在进行技术写作这么多年之后，有时候不禁会想，有什么颇具价值的经验是能够分享出来让大多数人受用的？当然这些经验可能很多，那么再进一步说，如果只允许我选取其中几则的话，我的回答会是什么？

但是这些内容可能对大部分没有创作经历的人来说意义不大，因为这些答案尝试解决的痛点只能让少部分有创作阅历的人才能感同身受。我们不如尝试回答一些更现实和更迫切的问题，最好对于哪怕是饶有兴趣的人也有所帮助。这些问题在过去一段时间内在和其他作者沟通的过程中频繁被提及：

- 如何保证文章效果？
- 如何找到内容受众？

- 如何把握作品长度?
- 如何才能得到更多的曝光?

数字枷锁让创作过程步履维艰，我们在忠于自我表达和躲避数字的嘲讽间惴惴不安的徘徊。

二

不仅文字创作者会产生此类疑惑，相信对于例如播客或者视频等其他内容形式的创作者而言它们也同样成立。仔细端详就会发现，所有这些问题并非源自内容实质本身，我们焦虑的对象其实是媒介二次赋予的传播属性。内容异化的原因便是在此，数据原本在播种内容之后成长于顺其自然，但当我们无意识的踏入数字陷阱之后，煞费苦心的换取点赞成了创作的第一要务。

纯粹的数值评判有失偏颇，因为互联网反映的仅仅是大众偏好而已。吊诡的是这个体系本无权力准确代表任何一个人，但在体系里每个人似乎都在被代表。如果你把这种数值评判亦或是算法推荐定义为高效的话，这种高效恰恰是建立在懒惰之上的，代价即个体的真实需求一

再被忽略，立体的人被碾压为平面数据最后汇总为标签。互联网的偏见早在波兹曼的《技术垄断》一书中就被预言：每一种工具里都嵌入了意识形态偏向，它用一种方式而不是另一种方式给一种事物赋予更高价值。

点赞大行其道的另一个原因是无论对于平台还是创作者而言，注意力等同于潜在收益。于是在利益相关者的合谋下，努力讨好大多数内容消费者成了不折不扣的现状，越来越多的机构号在替代个人意见粉墨登场。如果说他人意见天生具有疏离感且允许带有攻击性，相反机构号则是无条件主动迎合于你，他们从标题到行文都试图在挑拨你的正反馈神经。结果是偏见被进一步被放大，仿佛我所见皆为我所共鸣。信息茧房带有极强的迷惑性，我们期待着发现新鲜事物，期待和有趣的人连接在一起，但正如韩炳哲在《同质化的恐怖》中所断言的：数字化的全联网和全交际并未使人们更容易遇见他者。相反，它恰恰更便于人们从陌生者和他者身边经过，无视他们的存在……同质化的恐怖席卷当今社会各个生活领域。人们踏遍千山，却未总结任何经验。人们纵览万物，却未形成任何洞见。

独立创作者竭尽所能去赢下每一局数字游戏非明智之举。一方面在争夺用户注意力的游戏赛道上，你越是把专业领域的深度展现的淋漓精致，越是容易将读者拒之门外。零思考，少付出，强共鸣，一针见效才是这类产品最理想的形态。当用户沉溺在与倒影的嬉戏中不可自拔，当有价值的内容在节节败退，在这场游戏里谁是才是最后的赢家是个耐人寻味的问题。消费者当然有娱乐的权力，但选择的权力同样不可被剥夺。

另一方面你很难也无需凭一己之力迎合每一类细分受众。虽然互联网式民主使得所有作品都有机会被一视同

仁的观赏，但观赏的门槛并未消失，约翰伯格在《观看之道》开篇便指出 " 我们见到的与我们知道的，二者的关系从未被澄清…… 我们观看事物的方式，受知识与信仰的影响 "——简而言之，看见不等价于理解，任何内容都会天然缺失一部分读者的支持，甚至是招来反对的声音，这是在预期之中的。如果所有观点都要被拉到力求每个人都能读懂的最低的共同标准上，那这对创作者而言也过于不公了。我并不是在批评大众的无知，而是在提醒各位无论内容的载体或者媒介如何变化，知识壁垒从来没有消除，内容和受众间的双向筛选天然具有正当性。

逃离数字游戏不会让你不堪，优先拥抱内容至少是一种自洽的创作方式，你的专业性不仅能够为你创造一条不同于博眼球的新赛道，还能为你的内容构筑一条甚至算法也无法逾越的护城河，而你收获的关注者也会更加垂直忠诚度更高。

此类创作也非易事。创作的门槛很低且是一个内耗的过程。无论你现在的人生处于什么样的阶段，你的所学所感总是能让你产出几段洞见性质的文字或者几集高质量

的播客内容，然而在这几千字或者几十分钟之后呢？如何保证长达几个月甚至几年有节奏的产出？过了蜜月期才见真章。

三

此时一个更尖锐问题顺理成章的出现了：如果我能用流量文章赚的盆满钵满，写高质量文章对我来说是否还有意义呢？这个问题的反面问题是，如果我大费周章写出一篇专业性极强的文章但是却收获不到多数人的肯定，高质量文章的意义又在哪里？

叫好与叫座虽然不是对立面，但想在同一个作品中达到双重效果很难。以电影为例，叫座用票房就可以衡量，叫好由专业人士或者学院派奖项予以评定，它们属于不同的评价体系。如果把所有作品最终都要以数值进行量化，那便是落入了功利主义的圈套中：我们能否将内容蕴含的所有价值都转换为统一的货币，以及在转换的过程中确保不会任何的折损？

如果你热衷于讨大众欢心，那我百分百支持你去做自己热爱的事情，商业和艺术无孰高孰低；同样的，如果你

的付出暂时得不到相应的收获，不妨考虑调整一下评价坐标系以及多一些耐心。

至于我的个人意见，我想引用导演姜文的一段话来回答你：

姜文在一次谈话节目中也被主持人问及了相似的问题，主持人抛出了两个观点，一是电影只不过是用来满足人的低俗需求，二是电影中的考究对观众来说真的有所谓吗。

姜文的回答是：“俗人是要往高处走的……给人点好东西吃，他会记着你原来对人的尊重来着……我是一个非常正能量，一个乐观的人，我觉得他今天不懂，他有一天会懂的。等他懂的那一天回来想，老姜真对得起我……给人点好东西吃，他会记得你原来对人的尊重来着。您天天想着接地气，掏人兜里的钱，人家也进步着，回头一看，孙子，你当时给我拍什么烂电影看”

这一段文字不需加以评论，从原始字里行间里传达出的态度就是最好的诠释。

我依然支持创作者尽可能多争取多的关注者，毕竟让有价值的内容传播的更远只会是双赢的结果。但要功利，同时也要超越功利，这是理想状态而非常态。对于精力有限的创作者而言，决定优先投入资源的渠道是个必须要做出的抉择。无论你此后的运营技巧多么娴熟高超，这里都会是你的起点。

本文开头的问题最适合用存在主义哲学来回答：存在先于本质，即我的本性要通过我的选择来塑造，我总是先我自己一步，边前行，边构筑自己——你的文章风格或者你的受众并非是你能够提前设定的，而是需要由你创造的作品去创造。

作者：**李光毅**

Thoughtworks 高级咨询师

如何评价一篇文章

怎样算是一篇好的洞见？

运营了将近七年的洞见，也看了七年同事们的文章，我们积攒了一些内容方面的经验，但也意识到，去评价一篇文章是很主观的事情。内容因为写作者的主观和场景的客观存在不同，而没有定形，尝试用统一的“标准”来衡量文章的好与坏，自然是一件徒劳的事情。

但仍然要去决定文章能否发表在 Thoughtworks 洞见，以及给予写作者在改进方向上的反馈，我想这里会有一点指导性的框架，在这个框架内的空间，每个写作者可以自由地腾挪。那这一篇东西，就是会讲这个框架性的东西。

简单的说，这个对于洞见的指导性框架，包括以下几个部分：

1. 写作文章的动机及思考
2. 洞见文章要解答的元问题
3. 优秀洞见的加分项
4. 文章的表面功夫
5. 文章写好还有最后一公里

下面一个个来谈。

1. 写作文章的动机和思考

即便不是写作洞见，就是写日记，写随意的文章，也是有一定的动机存在。这个动机可能是抒发情绪，可能是记录，但一定要是能说服自己的内容，自己愿意写下来的东西。所以如果你讲的故事，你写的东西，连自己都说服不了，你会愿意写出来吗？

在开始谈好的洞见文章具象表现之前，它们之所以好的本质，一定是自己愿意写下来的真实经历和情绪，然后才会有吸引观者的可能。然后再来一些不同的东西，自己独特的想法，加入你的思考，甚至一点点调性，都会让你的内容处于一个不错的起点。

1.1 写作洞见前的思考

每一次，有人跟我说，我想写点什么，或者说我写点什么凯峰你帮我看看。如有必要，我都会问 TA 几个问题：

1. 我想传达什么讯息？
2. 我的目标受众是谁？
3. 读完以后，希望受众做点什么？

大家可以在写之前，以及写之后，也自言自语问自己这几个问题。

即使是只为自己写，不发表给别人看，在文章中能清楚回答这三个问题的，也肯定好过目标受众乱，讯息不明确，却被发表出去的内容。

而传达明确的信息，对于希望自己写作变得更好，能发表洞见，是尤为重要的。

不少优秀记者（比如柴静）在自己的文章或者电视里说，记者写的，不要强加自己的判断和观点，而是需要把事实陈述清楚，让观者自己来判断，而不是代替他们去思考。这点我很赞同。

但对于自己想发表文章，不管在洞见，或者在技术媒体，则是需要陈述你的观点的，尤其在媒体有自己的趣味和倾向的时候，这点我们接下来详述。

但更重要的，在如今信息如此过剩的时代，如何快速让读者 get 到你的点，你要传达的讯息，是需要考虑的事情。我不能说这是对受众的迎合和妥协，但至少是对他们的尊重。

2. 洞见文章要解答的元问题

所以写作前有动机的选择，也有需要在写作洞见前有需要思考的三个小问题，即使这样，也无法解决写作洞见的“标准”这样的问题。但虽然说写作本没有标准，当把文章放到一个具体的时间和空间里来考量的时候，就会有这种类“标准”存在的余地。

那这个具体的时间和空间，就是指的是当下，在现在这个时期，Thoughtworks 目前的业务形态和组织发展状态下，对于影响力，或者对于具体的内容的诉求，从而有所明确的指向。

这也意味着，我们在写作洞见，发表洞见，只不过是把

你要创作的内容，放到了一个更加特殊的场景和上下文里。这样就会有相对更特别的环境，更特别范围的受众，即使进去之后，还会有进一步的划分。

在这之前，所有我们之前讲的关于内容的一切都还成立，无论是写作的动机，还是提笔前需要思考的三个小问题。（传达的讯息，面向的受众，他们要采取的行动）

我在几年的运营洞见中总结，发现无论不同文章的作者是谁，主题是什么，不管是技术面向，还是文化面向，乃至职业类和招聘类内容，都在尝试回答这三个元问题:

1. 我们是一家怎样的公司?
2. 我们是做什么的?
3. 我们需要怎样的人?

这三个元问题，其实是在分别映射 Thoughtworks 的具体状态：

1. 我们这个公司，追求技术卓越，追求社会公正，专注于软件交付。
2. 我们有不同的业务方向，涉足多个不同的行业领域，

为客户交付最终价值。

3. 我们需要跟我们一样志向的人加入进来。

可以看见，洞见不会仅仅是纯技术内容，而是它的内容会去尝试回答以上的三个元问题中的一个或两个。

举个例子，对于我们需要跟我们一样志向的人加入我们，那就会需要同事们的招聘故事，应聘故事，工作体验之类的文章，来侧面体现我们需要怎样的人。但肯定不会是非常直白的说：我们需要多少人，以及什么样的人。

3. 优秀洞见的加分项

在文章已经尝试去回答洞见的三个元问题之后，我们还发现一些优秀的洞见普遍特质，为这些文章添了不少光彩，虽然不是必需选项，但的确容易从数量较多的文章中脱颖而出，赢得读者更多的关注。

下面是对这些加分项的描述，很难用很精确的语言去描述他们，但希望能提供参考的角度。

3.1 典型的特质

争议性话题。博取眼球不是重点，重点在于引起的讨论和更多观点的输出，抛砖引玉。

总结性经验。带有“N 个 XXX”字样的标题总是会吸引人关注并收藏。

洞见性观点。独特的视角，执拗的语调，让人感同身受的情绪，让我们的内容区别于其他那些平淡无奇，味同嚼蜡的文章。

故事性描述。引人入胜的故事，胜过波澜不惊的叙述，总是更能在现在注意力缺失的阅读中，吸引读者更多。

举例：

1. 《企业实施 DevOps 的七大挑战》
2. 《DevOps 发展的 9 个趋势》
3. 《RESTful 架构风格下的 4 大常见安全问题》
4. 《敏捷实践 Showcase 的七宗罪》

3.2 代入感

如果你写作的目的是招聘，销售，或者营销，要把焦点

放在读者身上，而不是 Thoughtworks 身上。要移情，要挖掘他们的痛点和欲望。

用他们的语言，写出那些对他们来说很重要的东西，让他们知道我们都是志趣相投的一群人。

而不仅仅是谈到我们是多么好，多么聪明，如何不同，多么的颠覆性，我们多慈善。要做出来，显示出来，而不是告诉。

同时请注意人称的使用：

- 第一人称会让作者成为参与者，用“我”“我们”“我们的”。在文章中可以陈述一个观点。
- 第二人称更像是跟读者在对话，可以用在鼓励读者采取行动，订阅，点击之类。
- 第三人称既不代表说话的这个人，也不代表对话的那个人，更适合用在客户故事或者新闻稿。

举例：

- 《学会拒绝》

3.3 使命感

文章可以选择用合适的调调，所谓调性或者尿性，在丰富的信息背后，是热情饱满的表达，或者发人深省的思考，甚至是尖锐的抨击，来体现 Thoughtworks 的这种文化和价值观上的使命感。

比如于晓强的这篇《软件无用论》：

> 回顾我自己这些年的技术生涯，所做过的大大小小的项目，几乎都有着各自的悲惨，无论是成功的、失败的，都谈不上令人满意。所做过的技术决策，随着技术的革新，无一例外都是错误，若是以结果来衡量软件开发本身，一定是“人终有一死”。
>
> 然而我的感觉却没那么糟，因为我更相信自己能够在新的环境下，更好的认清楚自己要解决问题，更好的搞清楚约束是什么，更好的保持开放心态、同时谨慎前行。
>
> 除此之外，我却找不到任何一个与技术相关的词汇来描述自己的收获。

这样的文字，比起伟光正的不容置疑，更有鲜活的特征，值得信赖。

3.4 一目了然的结构

除非是散文，不然明确的结构会更容易让读者快速了解文章的整体和观点。

开宗明义，使用总－分－总的结构，清晰的小标题，每个部分之间逻辑过渡流畅，结尾有力，这都是写作的基本功，却也是最容易被忽略的部分。

举例：

- 《前端框架这么多，该何去何从？》

这篇文章就是设立了总－分－总的结构：

1. 选择很多，该怎么选？
2. 选择需要考虑的维度
3. 总结优劣势，具体问题具体分析

在已经获得足够的读者信任之前，长长的套路或者故弄玄虚，都是耍小聪明的表现，长久看是要伤害好不容易建立起来的信赖。

4. 文章的表面功夫

在解决了内容的主题和定位之后，文章的表面功夫是增强阅读体验和粘性不可缺少的部分，尤其考虑到目前更流行的移动阅读，在编辑帮助排版和美化并发表之前，写作者仍然要处理好第一遍文章的表面功夫。

4.1 排版

1. 尽可能使用短句，使用 3-5 行以内的小段落。
2. 对文字按需适当加粗，并适时引用外部文字。
3. 正确使用标点符号，多尝试使用（无）编号列表。
4. 中英文混居，英文单词前后留半角空格。
5. 一篇文章中出现的颜色、字体、字号等最好不要超过 3 种。经常看到一些文章用 N 种字体和颜色来标示重点内容，效果适得其反。不仅没起到强调的作用，反而给人一种很混乱的感觉。
6. 段首缩进在互联网时代已不是必需，留段间距的方式来保持段落之间的层次感，保证良好的阅读体验。
7. 排版体现呼吸感，特别要提出的是考虑针对移动阅读的排版。

4.2 图片

- 在不影响表达效果的前提下，建议少用图片。现在读者大多用移动设备浏览文章，非重要图片的过度使用会导致文章浏览量下降。
- 和谐、完美的版式效果依赖于合适的图文比例。尽量避免大篇幅独句成段、图片高频率出现等问题。以一个读者的视角来把握图文比例。
- 需要看清楚其上文字的图片一定要清晰，否则建议用文字描述代替图片展示。
- 图片上加文字时，尽量从图片中吸取一个恰当的颜色来设置文字颜色。

举例：

- 《一名 Thoughtworks 咨询师的心路历程》

5. 文章写好还有最后一公里

好了，到这里为止，一篇图文并茂言之有物的洞见文章也准备好了，是发表完就万事大吉了吗？

这部分内容严格意义上其实超出了这篇文章主题的范围，但传播的过程以及效果却会影响表面上看已经完成的文章。简单的说，我们需要为洞见文章的传播做充分的准备，并不断地改进它。

5.1 脱敏

在发表和传播之前，很重要的一件事情是需要对文章内容做全面的脱敏或者洗白。需要脱敏的信息包括但不限于：

1. 涉及客户和项目的名称
2. 涉及公司机密的信息
3. 涉及同事的信息
4. 涉及所有用文字和图片的版权信息

5.2 传播

如果想让更多的人看到自己的文章，获取到更多的反馈，就需要在尽可能多的合适的渠道上做广泛的传播。要熟悉不同的社交媒体，要了解不同渠道的品味定位和受众

群，如果有可能，要跟不同的媒体合作，去投稿，去申请，之类。

考虑到传播的渠道需求不同，可能要准备：

1. 所属文章的一个清晰可读，便于 SEO 的 URL 链接。
2. 可以作为引用文字，方便传播文章的一段话（不超过 70 字）。
3. 可以作用引用图片，方便传播文章的一幅图。

我把这叫做内容的最后一公里，做好了，才能最大得发挥一篇投入了很多精力时间的好文章的效用。

5.3 交互和改进

在社交媒体上，可以与读者做适度的交互，尊重读者的评论，并根据积极的反馈，持续地修改自己的文章内容，以获得进一步的反馈。

总结

讲了这么多，我们可以看到，并没有一个绝对意义上的标准，可以证实一篇文章是好的洞见文章，或者不是。

更重要的是，从里到外，从写作到传播，都是在不断思考和改进的过程，这才有了成为好的洞见文章的可能。

作者：**张凯峰**

Thoughtworks
洞见主编、首席咨询师

编 辑：张凯峰
设 计：陈思容
宣 发：马文朝 何颖

Thoughtworks 洞见