开放式协作

——开源软件的生产与维护

娜迪亚·埃格巴尔 著
X-lab开放实验室 译
开源社 校

华东师范大学出版社
·上海·

前锦排版 一校样(Z2 华师大 223407) 开本净尺寸：186mm×260mm 版心：34 字×29 行(双色) 出版科：陈 萍 2022－06－14

图书在版编目(CIP)数据

开放式协作：开源软件的生产与维护/娜迪亚·埃格巴尔著；王伟，夏小雅译. —上海：华东师范大学出版社，2022. □

ISBN 978-7-5760-□□□□-□

Ⅰ. ①开… Ⅱ. ①娜…②王…③夏… Ⅲ. ①□□□-□□□-□□□ Ⅳ. ①□□□

中国版本图书馆 CIP 数据核字(2022)第□□□□号

开放式协作：开源软件的生产与维护

著　　者　娜迪亚·埃格巴尔
译　　者　王　伟　夏小雅
责任编辑　李　琴
特约审读　□□□
装帧设计　□□□

出版发行　华东师范大学出版社
社　　址　上海市中山北路 3663 号　邮编 200062
网　　址　www.ecnupress.com.cn
电　　话　021-60821666　行政传真 021-62572105
客服电话　021-62865537　门市(邮购)电话 021-62869887
地　　址　上海市中山北路 3663 号华东师范大学校内先锋路□
网　　店　http://hdsdcbs.tmall.com

印 刷 者　□□□□□□□□□
开　　本　787×1092　16 开
印　　张　□□
字　　数　□□千字
版　　次　2022 年 3 月第 1 版
印　　次　2022 年 3 月第 1 次
书　　号　ISBN 978-7-5760-□□□-□
定　　价　□□□元

出 版 人　王　焰

前锦排版　一校样(Z2 华师大 223407)　开本净尺寸：186mm×260mm　版心：34 字×29 行(双色)　出版科：陈　萍　2022-06-14

Author recommendation Preface
作者推荐序

当我在 GitHub 工作时，还记得那是 2018 年，我们正在为即将发布的 Octoverse 报告（GitHub 对其平台上新兴趋势的年度报告）整理文案。我和数据团队的人坐在会议室里，看着同事在白板上画图。那是一张粗略的草图，显示了当年亚洲与其他地区的用户增长情况，各种颜色的线条代表世界各地的非亚洲地区，然后我的同事为亚洲画了一条线，那条标记线贯穿了整个白板。

那一年，亚洲创建的开源项目比包括美国在内的世界其他任何地方都多。中国成为全球第二大 GitHub 最受开源贡献者欢迎的国家。[①] 自 2014 年以来，GitHub 的亚洲开源贡献者数量创历史新高，2017—2018 年贡献者增长超过所有其他地区。截至 2019 年，中国占 GitHub 上亚洲开发者的近三分之一。[②]

众所周知，开源文化以西方为中心，但亚洲的区域增长表明开源的中心正在迅速转移。在《开放式协作的艺术》的开篇中，我有提醒到，我的观察主要针对居住在美国、欧洲和澳大利亚的开发者。在写作和研究的过程中，我意识到我们现在对于“开源”的认知将在未来几年发生变化，因此以一种不会过时的形式来写这本书。

正如开源项目不再像 2000 年代初那样，它们也不一定像我们在西方国家看到的那样。当我在 2018 年研究其中一些趋势时发现，很明显中国的开源开发者的行为与西方国家有所不同。中国的开发者在 GitHub 上显著地*创建*和*使用*开

① The State of Octoverse 2018. [EB/OL]. [2022 - 04 - 13]. https://octoverse.github.com/2018/people#location

② The State of Octoverse 2019. [EB/OL]. [2022 - 04 - 13]. https://octoverse.github.com/2019/#regions

前锦排版　一校样(Z2 华师大 223407)　开本净尺寸：186mm×260mm　版心：34 字×29 行(双色)　出版科：陈　萍　2022 - 06 - 14

源项目，但他们似乎更踟蹰于为他人的代码仓库做贡献。

开源文化历来强调*活跃贡献者*或深度参与开源项目的开发者的作用。但正如第 3 章所述，开源项目中有多种角色——包括维护者、活跃贡献者和用户。随着越来越多的开发者在亚洲发起、发展和维护开源项目，其中一些项目可能活跃贡献者会很少，而用户更多——正如我在第 2 章中描述的“体育场”模型——这些差异不仅受到技术决策和领导风格的影响，还会受到地域的影响。

软件发展的速度很快。这种感觉很奇妙，即使我刚刚才写完《开放式协作的艺术》，但开源历史的全新篇章才刚刚展开。我十分荣幸能够与世界上发展最快地区之一的开发者分享我所学到的知识。随着本书的读者扩展到全球，我希望本书中的思想和词汇能帮助我们描述开源新时代的篇章。

娜迪亚·埃格巴尔(Nadia Eghbal)

2021.10.10

前锦排版 一校样(Z2 华师大 223407) 开本净尺寸：186mm×260mm 版心：34 字×29 行(双色) 出版科：陈 萍 2022－06－14

Contents
目 录

前锦排版 一校样(Z2 华师大 223407) 开本净尺寸：186mm×260mm 版心：34 字×29 行(双色) 出版科：陈 萍 2022－06－14

前锦排版　一校样(Z2 华师大 223407)　开本净尺寸：186mm×260mm　版心：34 字×29 行(双色)　出版科：陈　萍　2022－06－14

Foreword
引 言

到目前为止，信息是一件很有用的事情，并且信息越多越好。如果思想的自由交流构成了繁荣社会的基础，那么我们在道德上有义务使更多的人彼此联系。

开放的精神持续了超过 200 年。我们倡导文化和教育的价值。我们修建了道路、桥梁和高速公路，将从前各自独立的社区汇聚在一起，我们对新的千禧年充满了向往。

因此，在 20 世纪末，当互联网开始从萌芽转变为肆意生长，他们和永无止尽的知识一起传播，携带了所有和平以及永不言弃的特质。万维网的发明者蒂姆·伯纳斯-李（Tim Berners-Lee）在他给欧洲核子研究中心（CERN）最初的提议中，设想了一个“信息池……可以繁衍和生长”[1]。他写道：“为了使这个成为可能，存储的方式一定不能对信息加以限制。”将他的提议作为蓝图，技术人员建立了一个规模超过了我们在物理领域中可以想象的亚历山大图书馆，将世界各地的人们及其思想紧密地联系在一起。

然后我们遇到了障碍。突然之间充斥了大量的信息。太多的通知使我们希望减少对它们的查看；太多的社交互动使我们希望减少在线发布的频率；太多的电子邮件使我们不想回复。实际上，我们之间正在互相 DDoS：这是一个全称为分布式拒绝服务攻击的术语，其中恶意行为者通过巨大的流量来淹没被攻击目标，从而使受害者丧失服务能力。我们的在线公共生活变得难以应付，导致我们许多人缩回到了私人领域。

在开源软件的世界中，也发生着类似的故事。“开源”作为一个几乎与“公共协作”同义的术语，其开发者（编写和发布任何人都可以使用的代码的人）却经常

对入站请求数量感到不堪重负。

在采访了数百名开源开发者并研究了他们的项目之后，我在福特基金会2016年的一份报告中总结了这个问题，报告题为“路与桥：数字基础设施背后的隐形工人”[2]。然后我开始尝试着手解决这个问题。

我的大部分压力测试都发生在2016年至2018年，当时我正在努力改善GitHub的开源开发者体验。GitHub是一家为开源项目提供托管服务的公司。作为一个大多数开源软件在其之上构建的平台，GitHub是理想的学习场所。这期间我遇到了更多开发者，也参与了更多项目，让我不得不切实考虑解决方案。在这过程中，我常常遇到一些言辞与现实相悖的时刻。

毫无疑问，许多开源开发者都缺乏支持。我的收件箱中充斥着渴望分享他们的故事的、来自开源领域的人们的电子邮件。但是困难的是如何确定他们的真实需求。

最初，我的探索重点是资金的缺乏。尽管产生了数万亿美元的经济价值，但许多开源开发者并不直接从开源工作中获得报酬。在缺乏额外的声誉或财务利益的情况下，维护供公众使用的代码很快就变成了他们无法推辞的无薪工作。

但随着这些年来对开发者故事的追踪，我注意到金钱只是问题的一部分。一些开发者在没有金钱补偿的情况下巧妙地处理了用户的需求，而一些有偿开源开发者也在经历着同样奇怪的行为过程。而这种行为过程在为雇主编写“专有”或私有代码的人员中似乎不那么普遍。

这个过程看起来是这样的：开源开发者公开编写和发布他们的代码。这些代码在聚光灯下度过了几个月甚至几年的时光。但是，最终人气收益递减。如果维护代码的价值未能超过回报，那么这些开发者中的许多人就会悄悄地退回到阴影中。

受雇于私人公司的开发人员主要与同事合作。在公开平台上编写代码的开发人员必须切切实实与成千上万的陌生人一起工作，因为任何可以接入互联网并关心这个项目的人都可以对其代码发表评论。经济报酬的缺失是一个明显的症状，可能是源于激励措施的错位。但是上述不可避免的开发者流失的过程似乎暗示着更深层次的事情。

当今默认的假设是，面对不断增长的需求，一个软件项目的开源“维护者”

(用于指代软件项目的主要开发者)需要找到更多的贡献者。人们通常认为开源软件是由社区构建的,这意味着任何人都可以参与其中,从而分散工作负担。从表面上看,这似乎是正确的解决方案,尤其是因为它似乎可以实现。如果一个独立的维护者对他的工作量感到精疲力尽,那么他们应该让更多的开发者参与进来。

如今,有无数旨在帮助更多开发者为开源项目做出贡献的计划。这些努力被广泛拥护为"对开源有益",并且他们通常是通过利用公众的善意来实现的。

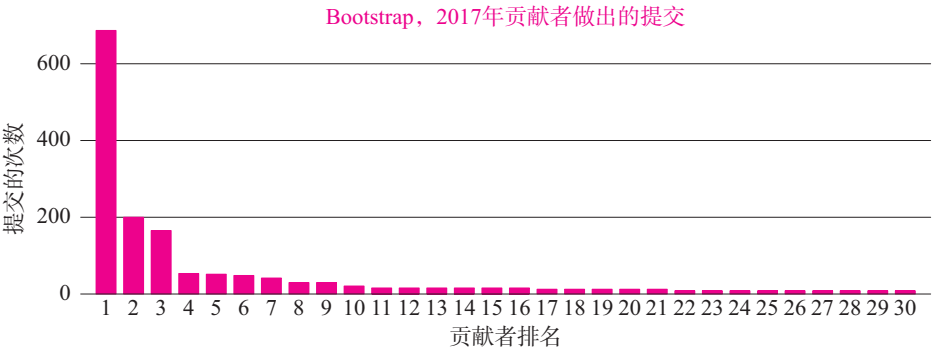
但是,在与维护者私下交谈时,我发现这些举措经常使他们焦虑不安,因为此类举措通常会吸引低质量的贡献。而这给维护者造成了更多的工作。毕竟,所有贡献都必须经过审核后才能被接受。维护人员经常缺乏将这些贡献者带入"贡献者社区"的基础设施;在许多情况下,项目背后根本没有社区,只有个人的努力。

在与维护者的对话中,我听到他们表达了一种真正意义上的矛盾:他们既希望鼓励新人参与开源工作,又感到无法亲自承担这项工作。维护者根本就没有精力去使每一个表现出兴趣的人参与其中。许多人告诉我,他们常常为那些对开源项目贡献摇摆不定而最终没有留下来的人而感到沮丧。维护者们细数了那些表达了兴趣的人,其中很多甚至在提交第一次贡献之前就消失了。

我开始发现问题不是没有人愿意为开源项目做贡献,而是有太多的贡献者,或者说,他们是错误的那一类贡献者。开源是公开的,但它不一定需要参与:维护者可能在过度需求下屈服。

一项研究发现,在使用各种编程语言的 275 个流行 GitHub 项目中,有近一半的贡献者仅贡献了一次。这些贡献者占据了总提交或总贡献的不到 2%。[3] 使"贡献者社区"这一概念受到质疑的,并不是只有一小部分开发者对项目做出了有意义的贡献,而是通常情况下成百上千的贡献者只能对项目做出微小的实质性贡献。

我意识到,我们所认为的开源工作方式与开源工作的实际开展之间存在巨大的脱节。开源项目中存在一个极长且还在不断增长的长尾,这些项目并不符合典型的协作模型。这样的项目包括 Bootstrap,一种流行的设计框架,大约有 20%的网站基于它而做,[4] 此项目中的三位开发人员承担了 73%的代码提



Bootstrap 在 2017 年的贡献者，根据提交次数进行绘制

交。*[5]另一个例子是 Godot，一个用于制作游戏的软件框架，此项目中的两名开发人员每周对项目中出现的 120 多个问题做出回应。†[6]

这种一个或几个开发者完成大部分工作，其后是大量的临时贡献者以及更多的被动用户的社区分布在开源中并不是例外，现在已经成为一种常态。从维护者的角度来看，在未解决的杂乱无章的问题中，他们悄悄地守护自己的代码，这个世界看起来不像是早期互联网先驱者拥护的陌生人之间大规模合作的乌托邦理想，甚至与这些早期倡导者的预测正好相反。一项研究发现，在他们所调查的 GitHub 上的所有开源项目中，有超过 85%的项目是由不到 5%的开发人员负责超过 95%的代码和社区互动的。[7]在他们的报告中，研究者们注意到了一种令人困惑的“已建立的理论与广泛观察到的经验效应之间的不匹配”。

这些数字看起来似乎敲响了警铃，但这仅仅是因为它们不符合大众的期望。我们假设开源项目需要不断壮大的贡献者社区才能生存。这里有一个术语叫“巴士系数”，指项目的健康程度可以通过度量“有多少个开发者被公交车撞了会使得项目陷入困境”来表示。

但这种表达不再适用于描述有多少个开源项目的场景（译者注：有多少个开源组件停滞会导致一个项目陷入困境）。鉴于如今软件开发者通常会依靠数百个开源项目来编写代码，因此不可避免地，他们只能被动依赖于这些项目。

对开源的期望本来应该是一种协作努力，因此最终孤军奋战的维护者们感到不知所措，甚至担心没有人露面是否是因为他们做错了什么。但是，如果我们从“一切本应如此”的前提开始呢？我决定重新审视诊断的方式，把这些症状作

为一个起点，而不是使用我们拥有的唯一处方——更多的参与。

就像当今其他类型的在线内容一样，代码也趋向于模块化：它是由一个个小库组成的千层蛋糕，而不是一个笨拙的大型果冻模具。如今，开发者可以像其他创作者一样轻松地在线发布一些代码以供公众使用。但是，就像推文很容易阅读和转发，但不需要标明作者信息一样，代码也很容易被复制粘贴，同时无需知道它们的来源。

npm 生态系统（据其母公司估计，构成了当今现代 Web 应用程序中 97%的代码）为未来提供了一些线索。[8] npm 全称为“Node 包管理器”（Node Package Manager），它是 JavaScript 开发者通常用于安装和管理软件包或库的平台（库是其他开发者可以使用的预先编写的代码包，而不必从头开始编写相同的代码。合法使用其他人的代码的能力使现代软件开发者可以更快、更高效地工作）。

与传统的大型整体软件项目和围绕它们所兴起的繁荣社区相反，npm 软件包的设计是小型且模块化的，这导致每个项目只需要更少的维护者，并与他们编写的代码形成暂时的关系。从这个角度来看，当今贡献者的缺乏反映的是对不断变化的环境状况的适应，其中维护者、贡献者和用户之间的关系更轻松，更易处理。

当与新手和临时贡献者讨论他们的经历时，我发现他们的故事与维护者的一样具有启发性。临时贡献者经常意识到他们对项目幕后发生的事情了解甚少。但更重要的是，他们也不想花时间熟悉项目的目标、路线图和贡献过程。这些开发者主要将自己视为项目的用户。他们不认为自己是“贡献者社区”的一部分。

维护者的角色正在演变。维护者的角色不再仅是对一组开发者进行协调，而是承担一种协调管理工作，需要从那些争夺其注意力的频繁交互——如用户问题、错误报告和功能请求——中筛选出噪音。

在有限的时间和精力下，单个维护者需要在被动任务（社区交互）与主动任务（编写代码）之间取得平衡。维护者还依赖于平台（即 GitHub）和工具（例如可以帮助管理问题、通知和代码质量的 bot）来跟上他们的工作。

如今，维护者面临的问题不是如何吸引更多的贡献者，而是如何管理大量的频繁、低接触的交互。这些开发者并不是在构建社区，他们是在指挥空中交通。

开源参与度的大幅提升并不局限于对代码的访问。使用开源代码的开发者数量呈指数增长。2001 年，SourceForge 是占主导地位的代码托管平台，那时候只有 200,000 个用户。[9] 而现在，作为继任者的 GitHub 拥有超过 4 千万注册用户，而在过去两年中，这个数字几乎翻了一番。[10]

信息量大，并且唾手可得，这是数字时代最重要的胜利之一。更重要的是：不是因为过多的代码消费，而是因为用户的过多参与在抢占维护者注意力，才使得当今的维护工作难以进行。

虽然这个开发者孤军奋战来为用户做事情的世界似乎与开源的故事有明显的偏差，但它跟更广泛的在线世界并没有什么不同。越来越多的在线世界是由个人建立的，而不仅仅是社区。‡。就像一位 Reddit 用户观察到的：

> 如果你在亚马逊上阅读评论，则主要在阅读由格雷迪·哈普（Grady Harp）等人撰写的评论。如果你阅读维基百科，则主要是阅读贾斯汀·纳普（Justin Knapp）等人撰写的文章……还有，如果你阅读 *YouTube* 评论，则主要是阅读贾斯汀·杨（Justin Young）之类的人发表的评论。你在互联网上消费任何内容，其实主要是在消费由于某些原因而花费大量时间和精力在互联网上的人创建的内容。这些人显然在某些（更重要的）方面和一般人群有所不同。[11]

作为开发者也是同理，如果你使用命令行工具 cURL，则用的是丹尼尔·斯坦伯格（Daniel Stenberg）编写的代码。如果你使用命令行界面 bash，则用的是切特·拉米（Chet Ramey）维护的代码。如果你使用 npm，则使用的是辛德勒·索尔许斯（Sindre Sorhus）和 Substack 编写的软件包。如果你使用 Python 的包管理工具，则使用的是唐纳德·斯塔夫特（Donald Stufft）维护的代码。

与任何其他创作者一样，这些开发者创作的作品与用户交织在一起，同时受到用户的影响。但他们的协作方式和我们通常认为的在线社区的协作方式有所不同。

相比于论坛用户或 Facebook 群组，GitHub 的开源开发者更类似于

Twitter、Instagram、YouTube 或 Twitch 上的个体创作者。这些创作者必须找到某种方法来管理广泛且快速增长的受众群体，同时管理与他们的互动交流。正如流行文化评论家马克·费雪(Mark Fisher)所说的那样，取代了传统的面对面交流，创作者的受众如今面对的是一个舞台。

和其他创作者一样，开源开发者也在创作可以给公众消费的制品。他们也需要处理拥挤的收件箱，管理他们有限的注意力并依赖于热情的支持者。与其他创作者一样，开源开发者也非常依赖平台来分发其作品。作为封闭的经济体，这些平台还肩负着帮助创作者提高声誉，并且捕捉他们创作内容价值的责任。

早期互联网活动的特点是大规模、分散的在线社区：邮件列表，在线论坛，会员群组。这些社区是一个个村庄集群，每个村庄都有自己的文化、历史和规范。

社交平台将所有这些社区聚集到了一起，就像橡皮泥一样将它们揉在一起。在这个过程中，我们制作和消费内容的方式被潜移默化地改变了。创作者如今可以接触到更多的潜在观众，但是这些关系是短暂的、片面的，而且常常是压倒性的。

克里斯汀·鲁佩尼安(Kristen Roupenian)在《纽约客》(*New Yorker*)的短篇小说《猫人》(*Cat Person*)广为传播之后，回顾了自己的经历，并以如下方式描述了这一点：

> 我渐渐接近一种状态，描述这个状态的词语可能听起来很戏剧性，它叫做“摧毁”(annihilating)。我和所有思考和谈论我的人面对面，这种感觉就像独自站在体育场中心，而成千上万的人正在对着我尖叫。不是为了我，而是瞄准了我。有些人可能会觉得这样令人振奋。但我不会。[13]

在创作者的世界中，开源开发者是一个非常有趣的子集。从经济角度看，代码类似于其他形式的内容。就像书籍或视频一样，代码只是一堆信息，打包起来可以分发。当然它的角色更接近于实用主义。

尽管社交媒体、新闻和娱乐都在我们的生活中扮演着至关重要的公共角色，但我们直接依靠开源代码来保持电话、笔记本电脑、汽车、银行和医院的平稳运

行。如果 YouTube 视频出现故障，我们可能仅仅为失去的有价值的信息而叹息，但是如果一个开源项目出现故障，它甚至会破坏整个互联网（我们将在第 4 章中看到）。

正因为此，与其他类型的创作者相比，我们审视开源开发者的行为会牵引出一系列基本的经济问题。更方便的是，开源开发者在完全公开的视野下工作，从而使他们的故事更易于研究。

开源一直是其他在线行为的先锋。在 20 世纪 90 年代后期，开源种下了大规模协作的希望火种，被称作为“同行生产”（peer production）。开源软件实际上也开始超过了公司销售的软件，因此经济学家认为这些开发者已经实现了不可想象的目标。随着互联网在开源的萌芽状态继续平稳推动，世界似乎确实有可能最终被自组织社区而推动。

但是在过去的 20 年中，开源莫名其妙地从协作转变为个人努力。尽管使用开源代码的人比以往任何时候都多，但其中的开发者却无法捕捉到他们创造的经济价值：重新审视这个悖论同样是件十分有趣的事情。

通过研究开源从“小型互联网”到“大型互联网”的过渡，我们可以更好地、更广泛地了解在线创作者的情况。我们仍在试图将个人创作者的崛起与报纸、书籍出版商和人才中介的衰落联系在一起：公司不再是变革的主要推动主体。作为一个案例研究，开源可以帮助我们理解为什么我们的网络世界没有像早期学者所预测的那样发展，以及我们的经济如何围绕个人创作者及其建立的平台进行自我调整。

如果创作者（而不是社区）准备成为我们在线社交系统的中心，那么我们需要对他们的工作方式有更好的了解。在当今有 45 亿人在线的世界中，一个人的作用是什么？这些创作者如何塑造我们的品味，以及当热度逐渐消减后，我们如何保护、鼓励和奖励这种工作？平台又是如何在帮助或阻碍这种工作的？

* 对于 2017 年的贡献来说确实如此。

† 该数据收集于 2017 年 8 月 29 日至 2018 年 8 月 29 日之间。

‡ 甚至 Wikipedia，世界上最大的在线百科全书，也是大规模合作的典范例子，史蒂文·普鲁特（Steven Pruitt）还是一名志愿者，他编辑了网站上所有英语文章的三分之一。[12]

第一部分

人们怎样生产

第一部分 人们怎样生产

Part I

01
GitHub 开源平台

本书通过讲述开源的故事，试图厘清并扩展“在线”在当今的真实内涵。通过开源，个体开发者编写的代码可以被数百万人使用。这些个体开发者们的工作并不是要最大程度地参与，与之相反，他们的工作可以被定义为：需要进行过滤和策划大量的互动。我试图解释为什么会出现这种情况，以及平台是如何将焦点从在线社区转移到独立创造者身上的。

当我探讨这个主题时，我将主要关注使用 GitHub 的开发者。* 这并不是在贬低 GitHub 之外平台的开源价值，也不是在暗示开源只能在单一平台上开发，或者不应该存在迁移到其他平台的机会。我主要写的是个人经历，而不是机构或公司参与者在开源中的角色——这是一个不同的、复杂的主题，值得单独写一本书。

开源在保持独立于任何专有软件、工具或平台方面有着悠久而丰富的历史。开源的故事比 GitHub 早了 20 多年。在很大程度上，开源缺乏标准工具的原因是在于设计考虑：开发者喜欢选择，他们希望可以选择最自己喜欢的工具来工作。

GitHub 对开源产生了巨大的影响。它冲破了自由和开源软件所在建筑的屋顶，落在建筑内的长凳上，压碎了下面的一切。

虽然没有要求开发者必须使用 GitHub 来编写开源软件，但 GitHub 是目前占主导地位的代码托管平台。事实上，大多数开源开发者现在使用 GitHub 不仅仅代表了个人品味的转变；这也意味着开发者文化的转变。

平台和它们的创造者之间的关系，对于讨论我们的网络世界如何改变是至关重要的。而且，因为 GitHub 是开源领域的主流平台，如果不讨论平台上所提供的服务，以及在平台上如何使用这些服务，受益于它的开发者，就无法理解平台与创造者之间的关系。

代码自由

在实际托管代码的平台出现之前，大多数开放源代码都是以“tarball”（以 .tar 文件命名，它将文件捆绑在一起）的形式发布在一些自托管的独立网站上。开发者使用邮件列表进行沟通和协作。每个项目对其贡献的管理都略有不同。

如果开发者想要做出贡献，他们就像访问另一个国家一样，必须首先了解每个平台的风俗习惯。

开发者使用版本控制来跟踪和同步他们的更改，当有多个人（其中许多人是陌生人）在世界各地不同的时区编写相同的代码时，这一点就变得尤其重要。因此，如果小明和小红对同一行代码进行了更改，版本控制将帮助开发者协调这些差异，并在不破坏任何代码的情况下保持井井有条。

但是现在最流行的版本控制系统 Git 直到 2005 年才发布。在此之前，开发者主要使用集中式版本控制系统，如 Subversion 或 CVS（如果他们使用版本控制的话）。这些系统不是为去中心化的大规模协作而设计的。

在集中式版本控制下，开发者必须将代码提交回相同的服务器，但在分布式版本控制下，每个人都可以分别处理自己的代码副本，然后再将更改的内容与其他人同步。Git（以及几乎同时发布的竞争性系统 Mercurial）是第一个成为主流的主要分布式版本控制系统，这使得开发者可以在技术上独立地工作。

然而，即使在 Git 发布之后，仍然没有一个标准化的开发工作流程。在某种程度上，早期的自由和开源的开发者喜欢这种工具、习惯和过程之间的不和谐，因为这意味着没有一个工具主宰这个领域，没有人能够完全抓住开发者的注意力。

理查德·斯托曼（Richard Stallman）是一位促使自由软件运动开始的 MIT 黑客。他受到鼓舞去发起 GNU 项目——一个自由软件操作系统。缘由于 1983 年，他在麻省理工学院人工智能实验室试图定制施乐打印机时，发现无法访问或修改其源代码。

斯托曼想把代码从私有使用中解放出来。术语“free”指的是能够使用代码做你想做的事情，而不是指代码是免费的。（因此，斯托曼经常重复短语“Free 是指自由，而不是指免费的啤酒”，以及偶尔使用西班牙语单词 *libre*，而不是 *gratis*，来指自由软件，以区分这两种含义）[15]

描述出代码自由对于自由软件开发者的重要性是件艰难的事情。非营利性组织软件自由保护协会（Software Freedom Conservancy，SFC）的负责人布拉德利·库恩（Bradley Kuhn）将他的生活方式比作素食主义者。就像素食者不吃肉一样，布拉德利也不使用专有软件。[16] 这意味着不使用 Twitter、Medium、

YouTube 或 GitHub 等网站。代码在此处好比是家畜，需要从人类中解放出来，即使是以牺牲个人便利为代价。

因此，编写自由软件就是要摆脱通常困扰商业软件环境的种种限制。自由软件是反主流文化，正好与那个时代蓬勃发展的黑客文化相一致。

“黑客”这个词是由作家史蒂文·利维(Steven Levy)普及的，他在《黑客：计算机革命的英雄》(*Hackers*：*Heroes of The Computer Revolution*)一书中描绘了 20 世纪 80 年代的黑客一代，令人印象深刻。在《黑客》一书中，利维介绍了当时许多著名的程序员，包括比尔·盖茨(Bill Gates)、史蒂夫·乔布斯(Steve Jobs)、史蒂夫·沃兹尼亚克(Steve Wozniak)和理查德·斯托曼。他认为黑客相信共享、开放和去中心化，他称之为“黑客伦理”。[17] 根据利维的描述，黑客关心改善世界，但不相信遵循规则就能达到目的。

黑客的特点是虚张声势、表现欲、恶作剧和对权威的极度不信任。黑客文化今天依然存在，就像垮掉的一代、嬉皮士仍然存在一样，但黑客不再像过去那样抓住软件文化的时代精神。如今，黑客一代的接班人可能是密码破译人员和涉猎信息安全的人。

尽管利维在他的书中并没有只关注自由和开源的开发者，但是 20 世纪 80 年代和 90 年代的黑客文化与早期的自由与开源软件紧密相连，正如三位领导者：理查德·斯托曼、埃里克·S. 雷蒙德(Eric S. Raymond)和莱纳斯·托瓦兹(Linus Torvalds)所证实的那样。

理查德·斯托曼(也被称为 RMS)是 20 世纪 80 年代在麻省理工学院发起自由软件运动的黑客。埃里克·S. 雷蒙德(也被称为 ESR)在 20 世纪 90 年代将自由软件重塑为“开源”，他被视为早期开源的非官方人类学家。莱纳斯·托瓦兹则是一个在 1991 年创建了 Linux，一个驱动当今许多操作系统的开源内核，以及在 2005 年创造了 Git 的程序员。前一个项目成为早期大规模合作的典范，而后一个项目无意中(也许让托瓦兹懊恼)催生了 GitHub。†

如果自由软件是一种意识形态，那么斯托曼就是它的忠实信徒、衣衫不齐的传道者。我曾经听人这样描述他：“那个穿着夏威夷长袍、拎着塑料购物袋去上课的人。”他以坚持区分“自由软件”和“开源软件”而闻名，他对植物有一种奇怪的恐惧，他还不请自来地出席演讲，就自由软件的一些小问题向不情愿回答他问

题的讲师提问。

埃里克·S. 雷蒙德是拥护“开源”这个术语的一群程序员中的一员，他们希望与自由软件的意识形态定位保持距离，并使开源这个想法看起来更有利于商业。他在 1997 年的一篇文章《大教堂和集市：一个偶然的革命者对 Linux 和开源的思考》(*The Cathedral and The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*)中对开源软件的好处给出了令人信服的解释。这篇文章认为，比起被限制在一个更小的开发团队中(就像一个“大教堂”)，当过程是高度参与的(就像一个“集市”)时，开发者会发现更多的软件 bug。雷蒙德和一群志趣相投的开发者组成了开源，并在第二年开始宣传他们的想法。‡

除了写关于开源软件，雷蒙德还写一些个人文章，并形成了一个极具辨识度的 ESR 品牌。他因出版《极客性爱小贴士》(*Sex Tips for Geeks*)而臭名昭著，这是一本关于泡妞和床上功夫的文章集。[19] 自称“枪支狂人”和自由主义者，雷蒙德还写了一些关于枪支所有权的文章，这些文章被收集在“埃里克的枪支狂人页面”上。[20]

这三人中的最后一个是莱纳斯·托瓦兹，他在很多方面都很出名，但最臭名昭著的可能是他鲁莽的交流风格，以及他对开源项目的铁腕控制。托瓦兹的邮件列表中充满了咒骂，有时全用大写。2012 年，他在阿尔托大学做了一次演讲。当被问及英伟达缺乏对 Linux 的支持时(英伟达是图形处理单元(GPUs)的制造商)，他转向摄像头，竖起中指，咆哮道：“英伟达，去你的！”[21]

不仅仅是沟通技巧，托瓦兹的管理风格也使他声名狼藉。Raymond 在他其中一篇文章中称这种风格为“仁慈的独裁者”[23]，后来被 Python 编程语言的作者吉多·范罗苏姆(Guido van Rossum)改编为更著名的短语“一生仁慈的独裁者”，用来描述即使在项目发展过程中仍然保持控制权的开源项目作者。尽管 Linux 基金会报告说，自 2005 年以来，[24] Linux 内核的贡献者已经超过了 14 000 人，但托瓦兹仍然是唯一一个被允许将这些贡献合并到主项目中的人。[25]

尽管不乏令人难忘的黑客人物，但自由和早期的开源精神也被一些不受关注的人们所定义。在我最早与自由软件运动的一位杰出成员(我不愿透露其姓名)的一次谈话中，他对开源与开发它的开发者有任何关系的观点都会愤怒地吐

槽。他告诉我，代码是“无政府主义的”和“不可触及的”，它必须能够在个人欲望或需要支付租金以外独立生存。他说：“它需要成为一种任何人都无法夺走的东西。”“系统才是重要的。如果一个开发者离开了，另一个开发者就会介入并维护它。”代码的自由也延伸到编写代码的人的自由。

卡尔·福格尔(Karl Fogel)是流行版本控制系统 Subversion(Git 的集中式前身)的合著者，[§] 他对我的观点表示赞同：

> 当我们说：“那是我的自行车”或者“那是我的鞋子”时，我们的意思是我们拥有它们。我们对我们的自行车有最终决定权。但是当我们说“那是我的父亲”或“我的妹妹”时，我们的意思是我们和他们有联系。很明显我们并没有拥有他们。在开源中，你只能在关联意义上说“我的”。在开源中没有所有格的“我的”。[26]

因此，可以理解为什么许多开发者对将代码绑定到特定平台的想法感到愤怒。鉴于开源的意识形态根源在于将代码从私有控制中“解放”出来，许多开发者关心的是保持在互联网上任何地方发布和协作代码的自由。转移到其他地方的能力在开源设计中无处不在，无论是 *forking*——制作代码副本，还是能够 *clone*——将一个项目下载到本地计算机上。

令这些开发者苦恼的是，GitHub 本身并不是开源的，这意味着，就像斯托曼和他的施乐打印机一样，用户修改平台以满足自己需求的能力是有限的。Git 是用来管理代码的版本控制系统，独立于 GitHub 或任何其他托管平台，因此任何人都可以在其他地方复制和托管项目代码。但是 GitHub 的社区功能——所有的对话都是在平台上进行的——很难导出。(GitHub 紧随其后的竞争对手 GitLab 于 2011 年推出，它大力宣传自己的平台是开源的。GitHub 目前仍是市场的主导者：虽然很难找到 GitLab 采用的公开数字，但它的网站声称有超过 10 万个组织使用它的产品，[27] 而 GitHub 声称有超过 290 万个组织[28])

许多自由软件开发者拒绝使用 GitHub。埃里克·黄(Eric Wong)是开源网络服务器 Unicorn 的维护者，他在回应一位用户“请移到 Github 上”的请求时

写道：

> 不。绝对不会。GitHub 是一个专有的通信工具，它要求用户登陆和接受服务条款。这给了一个单一实体（以及一个盈利组织）权力和影响力。我向自由软件贡献的原因是我反对任何形式的被绑定或专有特性。当我遇到没有专用通信工具就不能使用 Git 的用户时，我感到非常难受。[29]

不管是出于意识形态还是商业原因，自由和早期的开源倡导者都专注于传播开源的理念。但是今天的开发者几乎不再注意到“开源”这个概念。他们只是想要编写并发布自己的代码，而 GitHub 正是让这一切变得容易的平台。

便利的胜利

GitHub 由四名 Ruby 开发者于 2008 年创立，他们将其宣传为一个“社交编程”（social coding）的平台，当时 Web 2.0——Facebook、Twitter、YouTube 和 Tumblr 刚刚起步。通过将所有开源开发者带到一个平台上，GitHub 帮助规范了他们的工作方式。

GitHub 并不是第一个代码托管平台。它的前身是成立于 1999 年的 SourceForge。如果说 GitHub 就像 Facebook，那么 SourceForge 就是代码托管平台中的 MySpace：它是同类产品中的第一个重要产品，尽管至今仍然存在，但最终都只是作为一幅蓝图被记住。和 GitHub 一样，SourceForge 给开发者提供了一个托管和下载代码的地方，但它更像是一个文件共享网站，而不是一个协作的地方。SourceForge 专注于代码分发；GitHub 专注于改善开发者的工作流程。对于 SourceForge 失败的原因，每个人都有自己的看法。许多人将其归咎于对广告的依赖（这给用户体验带来了负面影响）以及总体上较差的产品开发。其他人则指出 SourceForge 一直不愿意支持 Git 作为版本控制系统。

顾名思义，GitHub 的创始人把它的声誉押在了只有几年历史的 Git 上，作为软件协作的未来。在相对较新的技术上加倍投入，帮助 GitHub 吸引了一批

刚刚意识到 Git 好处的开发者。(现有的代码托管平台 SourceForge 当时并不支持 Git)GitHub 友好的用户体验帮助提高了人们对 Git 的兴趣,而 Git 的学习曲线是出了名的陡峭。很难说是 Git 还是 GitHub 促成了对方的成功,但它们结合在一起成为了大规模分布式软件协作的主导工具集。¶[30]

GitHub 将开发者工作流程的每个部分都整合到它的产品中,包括问题跟踪器、开发者个人资料,以及它所谓的"PR":一种提交、评审和合并贡献的方式。与在繁杂的互联网上搜索压缩文件(tarballs)和邮件列表相比,GitHub 的用户界面简单直观。你可以注册,选择一个用户名,并使用搜索栏来查找项目。打开一个 issue 或 PR 就像点击一个大的绿色按钮一样简单。

到 2017 年,GitHub 托管了 2 500 万个公共项目,而且该服务仍在增长。GitHub 称,其 2018 年的新用户数量超过了前六年的总和。(译者按:据维基百科数据显示,截至 2020 年 9 月份,GitHub 宣布已有超过 5 600 万的注册用户)[31]

通过吸引所有人到 GitHub 的平台,GitHub 让人们更容易发现新项目,以及每个开发者的历史和声誉。网站创建者还增加了"star"项目的能力,这样就可以知道哪些项目更受欢迎或不受欢迎。不管这个指标有多大的缺陷,现在 star 已经成为开发者们成功的标志,让他们可以将自己的项目与他人的项目进行排名。

GitHub 还帮助普及了宽松许可证的广泛使用,这极大地改善了开放源代码的覆盖范围和分发方式。自由软件开发者经常拥护 copyleft 授权,比如 GNU 通用公共许可证(GPL)。Copyleft 许可会像病毒一样附着在任何使用它们的代码上。也就是说任何包含 GPL 许可代码的代码也必须在 GPL 下进行授权。所以,如果像 Facebook 这样的公司在其移动应用中使用 GPL 授权的代码,该应用也必须按照 GPL 授权其他人使用。可以想象,copyleft 许可在商业上并不友好,因为公司必须以相同的条款许可他们自己的软件。所以早期的开源倡导者开始强调宽松的许可证,比如 BSD 许可证(the Berkeley Software Distribution)和 MIT 许可证(the Massachusetts Institute of Technology),这些许可证允许开发者在不改变他们自己项目条款的情况下,对代码做任何他们想做的事情。

到目前为止,MIT 许可证是 GitHub 项目中最广泛使用的许可证。一篇 2015 年的公司博客文章称,45%的开源项目都在使用它。虽然 MIT 许可证使

开源代码的发布变得顺利，但它“设置后忘记”的风格也被批评为在开源代码和它的意识形态起源之间制造了裂痕。奇怪的是，从长远来看，GPL 可能对开源开发者更友好，因为它让开发者对其他人如何使用他们的代码有更多的控制。但是很难想象 GPL 如何成为 GitHub 这样的平台上的主导许可证，因为它与平台的一个最大的优势不兼容：自由、不受限制的发布。[32]

GitHub 在现代开发者中的流行性是便利战胜了个人价值的经典技术故事。对早期的自由和开源开发者来说，由 GitHub（2018 年被微软收购）这样一家公司引领的向标准工具和工作流的转变，代表着他们自 20 世纪 80 年代以来一直为之奋斗的一切的倒退。代码协作不应该属于任何人，尤其不应该属于一家价值数十亿美元、制作专有软件的公司。

但是 GitHub 一代的开源开发者并不这么认为，因为他们优先考虑便利性，不像自由软件倡导者优先考虑自由，或早期的开源倡导者优先考虑开放。这一代的人不知道，也不真正关心自由和开源软件之间的区别，他们也不热衷于宣传开源理念本身。他们只是在 GitHub 上发布他们的代码，因为就像现在任何其他形式的在线内容一样，共享是默认的。

开发者布雷特 · 坎农（Brett Cannon）写了关于将 Python 项目转移到 GitHub 的决定，他解释说，他和他的合作者选择 GitHub 而不是 GitLab，部分原因是前者是开发者现在最满意的平台，另一部分原因是它有更好的自动化开发工具。但他引用的第三个原因最能说明早期和现代开源的区别：

> GitLab 没有杀手级的功能。现在有些人会说，GitLab 是开源的，这是它的杀手特性。但对我来说，开发过程比担心基于云计算的服务是否发布源代码更重要。[33]

GitHub 为开发者提供了完成工作所需的工具。与 20 世纪 80 年代到 21 世纪初的早期工作流相比，GitHub 易于使用、可靠、并且能够处理大规模交互。如果有人知道如何使用 GitHub，他们就很容易进入一个不熟悉的项目并做出贡献。GitHub 用户的个人资料中有照片、简介、和他们过去活动的公共链接，所有

这些都是自动生成的，这让他们的声誉对于一个项目来说是可见的。

GNU，一个自由软件的旗帜项目，并不需要在 GitHub 上获得成功。鉴于它是基于“代码解放”的思想而建立的，我甚至可以说 GNU 只有在无视 GitHub 作为一个平台的情况下才能真正流行起来。GNU 之所以是 GNU，是因为它并不托管在 GitHub 上。

相比之下，今天的这一代开源开发者需要 GitHub 来做他们最好的工作。就如同有 Instagram 影响力者和 Twitch 主播一样，这里也有 GitHub 的开发者。这些活动也可以在平台之外进行——你仍然可以把你在夏威夷度假的照片或视频上传到一个自托管的网站上——但你为什么要这样做呢？对于那些希望获得受众的人来说，平台和创造者已经变得密不可分。

平台通常会被描述为与创造者对立。《应用：人类故事》（*App*：*The Human Story*）是一部关于苹果应用商店开发者们如何与平台局限性作斗争的纪录片。[34] 与此同时，Facebook 经常被指责“用一种算法取代了大量出版商的受众”。[35] 但是，除了平台可能造成的所有问题，它们也带来了不可估量的价值。今天的开源开发者似乎真心喜欢 GitHub，认为它是一个编写、分享和发现代码的地方。

本·汤普森（Ben Thompson）在他的博客 Stratechery 上写了一些关于商业和技术的文章，他甚至认为传达这种价值本身就是平台的定义，而不仅仅是作为一个聚合器。平台将价值传递给基于平台建立的第三方，而聚合只是纯粹的中介作用。汤普森引用了比尔·盖茨的一句话，他对平台的定义是“当每个使用它的人的经济价值超过创造它的公司的价值时”。[36] 基于这个定义，汤普森认为 Facebook 实际上不是一个平台，而是一个聚合器，因为“除非是出于好心，Facebook 没有理由为（媒体）出版商做任何事情”。

就像人才经纪公司一样，平台首先会改善创作者的分销渠道，让他们接触潜在的数百万人，从而为他们增加价值。这种发现主要有利于那些仍在建立受众的创作者。这种反馈是正向循环的，更多创造者会被鼓励加入。只要越来越多的人继续使用这个平台，就不存在说某一个创造者会吸走这个房间里所有的氧气。

不同于人才经纪公司，平台没有合同来阻止创作者把观众带到别的地方去

(有些创作者有足够的影响力对平台造成影响,如果这么做的话:顶级主播 Ninja 离开 Twitch 去了微软的独家竞争平台,Mixer。碧昂斯(Beyoncé)在她的专辑 Lemonade 发布后的头三年,都只在 Tidal,一个新兴的订阅音乐服务上独家放送**)。*所以平台必须让自身变的不可或缺,来最大限度地鼓励创作者留下来。这个措施通常与*降低创作者的成本划上等号。正如布雷特·坎农在他关于 Python 迁移的文章中解释的那样,GitHub 帮助他的团队"尽可能地自动化开发过程,同时降低 Python 开发者必须维护的基础设施成本"。[37] Python 不再需要 GitHub 才能够被发现,但一个成熟的编程语言项目尤其受益于 GitHub 所提供的能够降低成本的基础设施。

因此,平台与创造者的关系必然类似于一种共生的、"俄罗斯套娃"式的混合生产模式,在这种模式中,创作性人才广泛分布,但却被嵌在一个集中平台的茧中。(任何人都可以成为创作者……在 Instagram,或 Spotify,或 Medium 上)尽管创作者来自世界各地,但他们需要这些平台来成长和生存。

事实上,今天的开发者不仅仅是使用,而是积极地偏爱 GitHub,这表明了一套与前几代不同的价值观。使用 GitHub 的简单行为已经将这些开发者与那些拒绝使用任何平台的教条主义的自由软件倡导者区分开来。而现代的开发者不加思考就给他们的项目贴上 MIT 许可证的事实——如果他们愿意为他们的项目申请许可证的话——将他们与早期的开源倡导者区分开来,后者是在"开放"的教条上大肆宣扬与传播的,这种教条通过许可来体现。

GitHub 一代的开源开发者对这些问题并没有特别强烈的感觉。他们只是想创造东西,而分享是对他们努力成果的自然衍生。流行前端框架 Bootstrap 的联合创始人雅克布·桑顿(Jacob Thornton)曾在一次会议上承认,尽管他从事高可见性的开源项目多年,但他"真的不知道开源是什么"。[39]

史蒂夫·克拉布尼克(Steve Klabnik),一个因他在两个流行的编程语言社区 Rust 和 Ruby 中的工作而出名的开发者,指出过时的词汇限制了我们谈论开源是如何产生的:

> 为什么自由软件和开源的概念本质上与许可证捆绑在一起是一个问题?因为这两个运动的目的和目标都是关于分配和消费,但今天人们最关

> 心的是软件的生产。软件许可证协议规定分发，但不能规范生产……这也是开源之后的主要挑战。[40]

通过关注开发者体验，GitHub 让开源更多地关注人而不是项目，这就是开发者迈克尔·罗杰斯（Mikeal Rogers）所说的“开源的业余化”，“push 代码几乎变成了和发 Twitter 一样常规”：

> 我已经为开源项目贡献了 10 多年，但现在不同的是，我不是这些项目的“成员”——我只是一个“用户”，贡献一点点就是作为一个用户的一部分。[41]

编写代码就像以多种方式发推文一样。包管理器的出现为开发者提供了一种简单的方法来安装和管理与其所选择的编程语言相关联的软件库。包管理器最初出现在 20 世纪 90 年代，以支持 Linux 生态系统，但最终它们成为大多数编程语言的标准工具。例如，编程语言 Ruby 有一个名为 RubyGems 的包管理器，而 Python 有 PyPI。

包管理器使得为软件开发创建、发布和共享可重用组件变得更加容易，这些组件通过一个注册中心进行管理：你可以想象为一个公共图书馆，其中包含数千本书，任何人都可以使用他们的借书证访问这些书。这些管理器极大地加快了开发者从命令行完成的工作。

通过一个安装命令，开发者现在可以导入数百个包（由其他开发者编写的代码块），并在自己的代码中使用它们。结果就是，构成“开源项目”的概念也变得更小了，就像从博客文章到推文的转变一样。

开发者拉斯·考克斯（Russ Cox）在描述这种转变时解释道：

> 在依赖管理器出现之前，发布一个只有 8 行的代码库是不可想象的：开销太大而收益太少。但是 npm（JavaScript 的包管理器）已经把开销几乎降到了零，其结果是几乎微不足道的功能都可以打包和重用。[42]

因此，这些新出现的行为成为 JavaScript 的标志性特征之一就不足为奇了。JavaScript 是在 GitHub 上发展起来的语言生态系统。

从黑客到哈士奇

JavaScript 诞生于 1995 年，但最近的技术发展赋予了它新的当代意义。它最初是由布伦丹·艾希（Brendan Eich）为 Netscape Navigator 编写的。因为 JavaScript 是唯一支持在现代浏览器中内置引擎的语言，所以从那时起它就一直与开发者息息相关。

虽然 JavaScript 已经出现了 20 多年，但在它的大部分历史中，主要被用作网页的脚本语言。它与标记语言 HTML 和样式表语言 CSS 一起构成了前端开发的神圣三合一。HTML 为网页提供了脚手架，CSS 样式化了这些元素，JavaScript 使它们成为动态的。如果网页是一座建筑，HTML 构成了骨架，CSS 构成了油漆和干墙，JavaScript 构成了电力和管道。

JavaScript 很容易编写和修改，因为开发者可以直接从浏览器中完成。每个主流浏览器都有一个“查看源代码”选项，这使得检查构成任何网页的 HTML、CSS 和 JavaScript 很容易。对于许多软件开发者来说，JavaScript 为他们提供了一个可访问的编程第一个切入口。

在 2000 年代中期，开发者开始寻找创造性的方法在他们的应用程序后端（对用户不可见的底层逻辑）使用 JavaScript。2009 年，瑞恩·达尔（Ryan Dahl）发布了 Node.js，这是一个在客户端和服务器端都可以轻松运行 JavaScript 的平台。现在，开发者不必为应用程序的后端和前端各学习一种语言，而只需要学习 JavaScript 一种语言，然后在任何地方都能使用它。

现在的 GitHub 项目是用各种各样的编程语言开发的，包括 Java、Ruby 和 PHP，但 JavaScript 比其他所有语言都更占优势。在 GitHub 上，JavaScript 的受欢迎程度是排名第二的 Python 的两倍多。[43] 根据 Stack Overflow 网站的说法，它已经迅速成长为开发者中最常用的编程语言。[44]

JavaScript 的普遍吸引力使它具有极高的可触达性并且十分强大。从文化的角度来看，它让前端和后端开发者成为了奇怪的伙伴。当代 JavaScript 是在

后 Web 2.0 时代发展起来的。它既友好又精致，但也带有政治色彩。它吸引了那些喜欢解释事物的开发者，他们用表情符号和色彩鲜艳的标识来解释事物。

JavaScript 开发者欣然接受“开源的业余化”，尽管他们的前辈偶尔会对他们翻白眼。npm，JavaScript 的包管理器，通过让同时发布和依赖多个包变得更容易，从而鼓励模块化设计。而且由于 JavaScript 必须保持跨浏览器的兼容性，它的官方核心库比大多数其他语言都要小，开发者不得不定制化编写自己的包来填补空白。因此，每个项目都趋向于更小，更易于使用，就像组装在一起的乐高积木，而不是用石头雕刻的城堡。

自由软件黑客将自己定义为独立于平台的人，但如果没有 GitHub 这样的平台提供的优势，JavaScript 可能不会像现在这样流行，更不用说 Twitter 和 Slack 这样的现代通信渠道了。与那些强调代码自由高于一切的人不同，JavaScript 社区散发出相反的吸引力：当代码很小时，脱颖而出的是那些人。

JavaScript 最知名的开发者以他们的演讲、他们录制的视频、他们写的推文和博客帖子而闻名。他们拥有大量的追随者，以一种 PHP 开发者做不到的方式吸引着热切的观众。例如，肯特·C. 多兹（Kent C. Dodds）花了大量时间制作关于 React 的内容，以至于他辞去了 PayPal 的 JavaScript 工程师工作，成为了一名全职教育家。[45]

JavaScript 开发者不再总是因为特定的项目而出名，就像斯托曼因为 GNU 而出名，托瓦兹因为 Git 和 Linux 而出名一样。与其只和一个或几个项目相关联，杰出的 JavaScript 开发者通常创建数百个小型但广泛使用的项目。人们知道他们是谁，而不是他们参与了什么项目。

安东尼·凯宾斯基（Antoni Kepinski）是一名十几岁的开发者，他开发了一个开源的披萨跟踪应用程序。当一位采访者问他是如何进入编程行业的时候，他回答说：

> 事实上，我开始用 C# 编程，但这不是我真正喜欢的东西，我偶然发现了一个 GitHub 用户辛德勒·索尔许斯的个人资料。当我第一次看到 JavaScript 代码时，我就知道这是我将来想要学习的东西。事情就是这样开始的。[46]

在随后的采访中，他开玩笑说，他使用了索尔许斯的一个项目，而不是另一个 JavaScript 开发者费罗斯·阿布哈迪耶(Feross Aboukhadijeh)的项目：

> 凯宾斯基：对不起，我没有使用你的 linter(一种捕获代码中印刷错误的工具)。
>
> 阿布哈迪耶：是的，我注意到了。你用的是辛德勒·索尔许斯的，因为我觉得你更喜欢他。这很好。(笑声)
>
> 凯宾斯基：不，不……
>
> 罗杰斯：就是这样。你跟随你最喜欢的开发者，“我想要他们的风格指南”。就这么简单。

对个性的崇拜在 JavaScript 中盛行，即使最著名的开发者不愿承认这一点，也许是因为这种态度与公开宣称的开源“社区构建”的理想相冲突。与早期著名的黑客托瓦兹、雷蒙德和斯托曼相比，今天的许多 JavaScript 开发者都非常谦逊。

举几个例子：流行前端框架 React 的维护者丹·阿布拉莫夫(Dan Abramov)写了一篇博文，列出了所有“人们常常错误地认为我知道”的编程主题。[47] Babel 是一个帮助 JavaScript 在不同版本间进行转换的库，它的维护者亨利·朱(Henry Zhu)说，开源帮助他“在我的耐心和谦逊中接受测试，并学会更好地专注和沟通”。[48] 另一个流行的前端框架 Vue 的维护者莎拉·德拉斯纳(Sarah Drasner)共同编写了一个“开源礼仪”，在其中，他们强调贡献者应该“礼貌、尊重和善良”。[49] 辛德勒·索尔许斯曾经在 Patreon 网页(一款付费创作社区)上为他的创作募捐，主页，显示他抱着三只快乐的哈士奇；[50] 费罗斯·阿布哈迪耶的 Patreon 页面显示了他和一只柔软毛茸茸的小猫依偎在一起的画面。[51] 这些开发者中的每一个都拥有大量的用户，这些用户都追随他们个人；他们拥有成千上万开发者的关注，他们选择利用这种影响力来展示善意和尊重。

每个 JavaScript 开发者都是圣人吗？当然不是。和其他开源社区一样，


图 1 莱纳斯·托瓦兹和辛德勒·索尔许斯
(经阿尔托大学和辛德勒·索尔许斯许可使用的图像)

JavaScript 在 Twitter 和 GitHub 上也经历了激烈的争论和个人攻击。但是值得注意的是,那些不参与这些事情的杰出开发者,似乎都在回避自己的名人身份。

2019 年,JavaScript 社区的两个子集 Vue 和 React 之间爆发了戏剧性事件。丹·阿布拉莫夫暂时关闭了自己的 Twitter 账户,后来解释说:“这个周末我变得越来越焦虑。我对那个州的社区毫无用处。我现在感觉好多了,我是来听的。”[52]

开源文化的重心已经从推崇强权专制转向寻求深思熟虑和管理。就连莱纳斯·托瓦兹也在 2018 年发布了一份道歉声明,他离开 Linux 项目一个月,是为了反思多年来“我们社区的人们面对我这一辈子不理解情感的经历”,并称之为“照镜子”的时刻。[53] ESR 被禁止进入 OSI 的邮件列表,因为他的言辞好斗。[54] RMS 在麻省理工学院计算机科学和人工智能实验室(MIT's Computer Science and Artificial Intelligence Laboratory, CSAIL)的邮件列表上发表了有争议的言论后,辞去了他在麻省理工学院和自由软件基金会的职务。[55] 在这个问题上,双方的许多人都注意到斯托曼的评论——他在评论中用他剖析“自由软件”这个词的同样迂烂的方式剖析了“强奸”的定义——从他过去的行为来看并没有什么异常,但这是个在变化的时代。

这种善良的倾向也会导致平台的分销能力产生紧张。由于它的流行，JavaScript 吸引了没有经验的开发者，他们第一次学习如何编写代码，也不熟悉如何与开放源码项目交互。因此，在 GitHub 上 JavaScript 维护者和用户之间的交互可能很难管理。因为 GitHub 很容易使用，所以打开一个问题或提出问题的障碍很低，用户向维护者寻求一切帮助：如何解决合并冲突，如何编写测试。这些技能并不是特定于任何一个开源项目；它们都是关于“我如何开源”的问题。

负责维护数百个 JavaScript 项目的辛德勒·索尔许斯曾在 Twitter 上写道：“在 GitHub 上审核了 1 万多个 PR 之后……大约 80%的贡献者不知道如何解决合并冲突。”[56] 几年后，他又进行了更多的观察：

- 几乎没人能写出一个好的 PR 的标题
- 有一半的人不知道'Fixes＃112'的语法
- 约 30%的人在提交 PR 之前不会本地跑测试
- 约 40%的人不会包含文档/测试[57]

> 我还注意到，在过去几年里，整体 PR 质量大幅下降。我猜这是 GitHub 越来越受欢迎的结果。[58]

这不是新开发者的错。他们不知道开源是如何工作的，他们被告知提出问题是在做正确的事情。“但不要让这成为你贡献的阻碍，”索尔许斯自己补充说，[59]“你要记住，我的时间是有限的，如果我不得不在你的 PR 中来回奔波，你可能会发现自己会再看一遍，这就占用了你从其他 PR 中抽出的时间。”[60]

开源维护者实际上已经成为了学习如何贡献的开发者的老师。在过去，当新开发者试图加入项目社区时，这么做是很有意义的。时至今日，仍向陌生人不断重温这些基本知识只会令人疲惫不堪：被纸划伤一千次也会让人丧命。开发者诺兰·劳森(Nolan Lawson)将自己的经历描述为“一种反常效应，即你越成功，你就越会受到 GitHub 通知的‘惩罚’”。[61]

GitHub 的一代

开源是建立在开发者不应受制于特定平台的理念之上的。与其他类型的开

发者相比，开源开发者似乎更应该不受平台效应的影响。毕竟，在 GitHub 之前有整整两代自由和开源的开发者，他们使用的工具在没有 GitHub 的情况下也能正常工作，而且有技术娴熟的用户，他们对不方便但符合价值观的解决方案有很高的容忍能力。

然而，GitHub 继续占据主导地位。就连 1999 年成立的保护伞型组织 Apache 软件基金会（Apache Software Foundation）也在 2019 年宣布，它将把基于 Git 的项目迁移到 GitHub。[62] 人们一度认为，Apache 软件基金会不愿采用现代开源工具。[63]

人们可以哀叹真正的“开放”软件开发的死亡，就像一些反对者仍然在做的那样，并批评 GitHub“破坏”了开源的发展轨迹。但 GitHub 克服了最初的宗教狂热，成为了大多数开发者的首选平台，这表明平台为开发者所做的事情可能比我们意识到的要多。我们不应该将平台视为对手，而应该将平台视为创造者的强大盟友，并尝试着去理解它们之间所形成的奇怪的共生关系。这种紧密的联系不可避免地会滋生强烈的情感和冲突，但也许这是它们变得不可或缺的标志。

GitHub 的优点——易于使用、易于共享和发现他人的代码——也是当今开源领域面临挑战的根源。对于容易吸引新手开发者的 JavaScript 中，这些挑战显得尤为突出。

考虑一下这样一个事实：JavaScript 项目是特意设计成小型和模块化的，而不是大型的社区项目，有长期成员，可以接纳新人。然后再考虑一下 JavaScript 的领导者们不愿表现得粗鲁或不受欢迎，他们对斯托曼-托瓦兹-雷蒙德的散伙黑客中那些挥舞着鸟、拿着枪的幽灵们置之不理。

总的来说，这是一场由用户驱动的完美风暴——大多数是出于好意，其他不是那么多——淹没了单一维护者的渠道，使用了一个旨在鼓励这种行为的平台，并受到社会规范的保护，明确地阻止了对入站请求的拒绝。

就像现在的其他社交平台一样，GitHub 专为大规模分解的用例而设计。每个平台都必须找出如何适应一套尚未被很好地理解的新兴社会行为。

* 我的范围一般也仅限于美国、欧洲和澳大利亚。开源开发正在这些地区之外发展，特别是在中国和印度，但由于他们的行为在某些方面不同，我将我的重点限制在我可以更自信地与之交谈的地理区域。[14]

†在过去，托瓦兹拒绝接受从 GitHub 上拉到 Linux 内核的请求，称它“很适合托管，但是 PR 和在线提交编辑，都是纯粹的垃圾”。[18]

‡尽管“自由”和“开源”软件的定义，甚至根据斯托曼自己的说法，本质上是相同的，但这两个概念在文化上是不同的。有些人，特别是自由软件开发者，不喜欢混合使用这些术语。自由软件的提倡者因意识形态而团结在一起，这是一场“自由和正义的运动”。他们认为代码应该从私有控制中解放出来。另一方面，早期的开源倡导者专注于实用的目标，比如标准许可，使代码更容易被任何人自由发布和使用，包括商业实体。[22]

§福格尔是一个有思想的，在我看来被低估了的早期开源文化的声音。他写了迄今为止唯一一本关于开源产品的著名著作：《生产开源软件：如何运行一个成功的自由软件项目》（*Producing Open Source Software*：*How to Run a Successful Free Software Project*），该书于 2005 年首次出版。你可以在 https://producingoss.com/上找到它的全部内容。

¶根据 Stack Overflow 2018 年的开发者调查，87%的受访者使用 Git。

**来自碧昂斯，她在歌曲《Nice》中说道：“耐心等待我的死亡/因为我的成功无法被量化/如果我对流媒体流量有多在乎/就会把《Lemonade》放到 Spotify 上。”[38]