hyperink

Google程序员Steve Yegge的呐喊和吐槽
深度探讨编程语言选择、代码中的哲学
和Google工作文化

程序员的呐喊

A PROGRAMMER'S RANTINGS

[美] Steve Yegge 著　　徐旭铭 译

人民邮电出版社
POSTS & TELECOM PRESS

程序员的呐喊

A PROGRAMMER'S RANTINGS

[美] Steve Yegge 著　　徐旭铭 译

人民邮电出版社
北京

图书在版编目（CIP）数据

程序员的呐喊 / （美）雅吉（Yegge, S.）著 ；徐旭铭译. -- 北京 ：人民邮电出版社，2014.5
ISBN 978-7-115-34909-5

Ⅰ. ①程… Ⅱ. ①雅… ②徐… Ⅲ. ①程序设计
Ⅳ. ①TP311.1

中国版本图书馆CIP数据核字(2014)第056155号

版权声明

◆ 著　[美] Steve Yegge
译　徐旭铭
责任编辑　陈冀康
责任印制　彭志环　焦志炜
◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
大厂聚鑫印刷有限责任公司印刷
◆ 开本：700×1000　1/16
印张：12.75
字数：191 千字　2014 年 5 月第 1 版
印数：1 - 3 500 册　2014 年 5 月河北第 1 次印刷
著作权合同登记号　图字：01-2013-3137 号

定价：45.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

内容提要

本书的作者是业界知名的程序员——来自 Google 的 Steve Yegge，他写过很多颇富争议的文章，其中有不少就收录在本书中。本书是他的精彩文章的合集。

本书涉及编程语言文化、代码方法学、Google 公司文化等热点话题。对 IT 界的各种现象、技术、趋势等，作者都在本书中表达了自己独特犀利的观点。比如 Java 真的是一门优秀的面向对象语言吗？重构真的那么美好吗？强弱类型语言到底哪个更好？敏捷真的靠谱吗？程序员要不要懂数学等。另外，他还谈到了很多大公司的理念，比如亚马逊做平台为什么那么成功等。最后，本书还收录了他写的 Google 面试攻略，这篇文章可以说为无数应试者点亮了明灯。

本书讨论的都是程序员非常关注的热点话题，内容广泛，观点独到，非常适合广大程序员阅读参考。

作者简介

Steve Yegge 是一名程序员，也是博主，写了很多关于编程语言、生产力和软件文化的文章。他拥有华盛顿大学计算机科学本科学位，20 年的业界经验，开发领域涉及嵌入式操作系统、可扩展的电子商务系统、移动设备应用、提升软件生产力的工具等。他曾就职于亚马逊和 Google 等公司。

译者简介

徐旭铭，从事编程十几年，翻译过几本书，现在在亚马逊当码农。工作和兴趣都是写代码，喜欢看上去很麻烦的问题。他住在西雅图，闲暇时喜欢看美剧。

前言

痛苦是本书的灵感源泉。唔，还有酒精。而当痛苦累积到一定程度的时候，我就会忍不住开始抱怨。再加上酒精作祟，什么刻薄（甚至滑稽）的话我都说得出来。现在我还会时不时地回头翻看这些东西，每次都忍俊不禁。原来我很毒舌嘛。

和绝大多数程序员不同，痛苦在我身上留下了独特的印记，迫使我的世界观发生了极大的变化。我现在能一眼看穿很多老鸟都看不到的东西，因为，老实说，他们都太墨守成规了。

过去 25 年里，经历了很多形态迥异的编程类型，写了太多原本不用写的代码。弯路走多了，反倒让我培养出一种第六感——我能看见死人。这种感觉其实很不好。如果你也这么倒霉，得到了这种第六感，那可以选择的路其实只有两条：要么心怀不满，沮丧不已；要么微笑面对。

于是我试着用乐观的态度去面对它。一开始会很难，不过现在也越来越熟练了。这当中少不了酒精的作用。当然，熟能生巧也是一方面啦。你得习惯自嘲，还要学着嘲笑别人，以及我们所生活的这个疯狂世界。更重要的是要把握好分寸，适可而止。

那以开车来举例好了。每次看着那些显然没有从车祸中吸取教训的人们，我都会笑得很紧张，眼角不由自主地抽搐起来。我总是能注意到那些跟车跟得太紧，结果把前保险杠撞烂的家伙；还有突然急刹，结果把后保险杠撞得稀巴烂的车；车头太出路口，结果被撞得一塌糊涂的车；以及那些冒冒失失偏离自己的路线，在隔离带上擦出巨大刮痕的车。

只要你注意到这些东西，它们可以说随处可见。很多司机似乎都不知道吸取教训。他们总是觉得自己“知道”怎么开车，之前的那次车祸根本不是自己的错！

你周围有多少成年人真的花时间去提高自己的驾驶水平？你认识的人里，是不是有些人的驾驶习惯让你难以忍受，想要帮他们指出来？只不过你知道他们肯定不爱听。

而事实上，每个人都觉得自己会开车。好像开车这件事和用微波炉加热玉米煎饼一样简单似的。任何指责他们驾驶技术不娴熟的人都是不知所谓。

其实这中间是有微妙区别的。大多数司机的问题在于他们觉得“会开车”和“好司机”是一回事。只要考到了驾照，再开上几年车，自己就算是经验丰富的司机了！该知道的自己都知道了。除了一些罕见的情况，比如雪地、沙漠、湿地、烟雾、极冷极热、强风、陡峭山地、人群，等等。那时只要开得慢一点儿不就万事大吉了嘛。而且大多时候是可以避免开车的。如果真的有必要，到时候再专门去学一下就好了。你一定是这么想的吧？

好吧……只不过书到用时方恨少，到了真的需要这些驾驶技巧的时候，通常连佛脚也没处抱了。比如车子正在打滑冲向电线杆的时候。运气好的话或许还能躲过一劫。只不过事故过后，你会想到要去接受培训，提高驾驶水平吗？当然不会啦，除非法官逼你去。毕竟你是“会开车”的人。

如果罗列一下所有的特种驾驶类型——专业赛车手、追捕车辆的警察、在森林公园防火道上的巡逻员、巨型车司机，还有很多闻所未闻的驾驶种类——只要稍微动点儿脑筋就能明白，其实你的驾驶技术根本不算什么，充其量也就是过得去而已。既然有一半的司机低于平均线，那么很可能你也是其中一员，大家都心知肚明，只有你自己还感觉良好。

这似乎是让所谓的僵尸启示录显得合理的原因之一。因为我们都知道，没人对此有所准备。大多数司机都只是僵尸的猎物罢了，因为他们都觉得自己“会开车”，别人也不会点穿他们。

事实证明，编程和开车非常像。只要写几年代码，行了！自己就算是“会写程

序”了，好像用微波炉加热玉米煎饼一样。大多数程序员这时都会陷入舒适区，再也走不出来，就像司机会尽量避开自己不熟悉的情况一样。要是有人给他们提点建议和意见，那这个人肯定是有毛病，要么就是搞不清楚状况。

但这种狗屁不通的观点和很多父母对待游泳课的态度不是一回事吗？无数的父母不让自己的小孩学游泳，因为他们有可能会溺水？

程序员和司机一样，总是自我安慰说等到需要的时候再去学新技能也来得及。但是在内心深处他们都明白，其实当需求出现的时候就已经太晚了。因此现实情况是这样的，旱鸭子会和水保持距离，司机会绕开泥泞的路段，而程序员会躲在舒适区里，搭建围栏把自己保护起来，然后祈祷世界和平。

可能你觉得这没什么大不了的——只要每个岗位上都有专门的人才不就万事大吉了嘛。但只要稍微深入思考一下编程和开车的比喻，就会发现事情并没有那么简单。开车很复杂，也是一件很自我的事情。我们都不愿承认这一点，因此产生了很多现实生活中的问题。

比如每个地区的驾驶风格就各有不同。尽管西雅图和洛杉矶的交规基本一样，但是两地的司机完全不一样。有一次，有个土木工程师告诉我说，一个地区的车速和咄咄逼人的程度，和这个地区糟糕的交通状况历史成正比。洛杉矶的交通几十年来一直都很差，所以每个人都知道，不凶狠一点是开不了车的。而西雅图的交通变糟只是这十几年来的事情，而且还没糟到洛杉矶的那种程度，所以西雅图的司机开车都像老奶奶。

不过西雅图的司机很不擅长在雨里开车，这一点相当搞笑。就算把车速降到接近零，每年还是会发生无数次车祸。

为什么会这样呢？这是因为西雅图的驾驶风格正处在转变之中，从休闲变得咄咄逼人。其中一部分原因是因为很多加利福尼亚居民沿着海岸线北上的同时，把原来的驾驶习惯也一起带了过来。另一部分原因是当地人口的快速增长，使得西雅图公路上的汽车数量年年翻新。所以现在西雅图的路上，既有彬彬有礼的司机，也有凶狠粗鲁的司机。

驾驶风格发生碰撞的时候，车辆自然也会撞到一起。等到风格统一了，也就是同一个地区的人开车方式“都一样”的时候，车祸数量自然也会减少了——哪怕在外人看来，这些司机都像疯子一样。

编程和开车其实一样错综复杂。编程的世界里也有司机、技工、汽车生产商、交通工程师、地区性的交通法规、交通执法、不同地区的驾驶风格，当然还有大规模的汽车追尾事故等。编程也有自己的亚文化，若硬要把它们放在一起，肯定会出现不协调。

这就是我要写这本书的原因，也是怎么写这本书的指引。我选择了一条不寻常的小路，几十年来不断探索大多数程序员从未接触过的领域，了解了人们在不同的风格和文化里是如何开发软件的。我领导过一帮拥有令人难忘的怪癖的专家，用过十几种语言，活跃于各个社区，见证了巨型项目的崛起、辉煌、衰败和消亡。虽然我不敢自称编程界的“百晓生”，但也是见过大世面的。

现在的狭隘可以说随处可见。就好像不同地区的司机，程序员也会在工业界和学术界里组成自己的小圈子，形成自己的术语、惯例、禁忌、学问等文化产物。他们能创造出自己的知识领地，就觉得自己天下第一、独一无二。

不管发生何种重大事故或交通问题，人们都会把责任怪在不合格的司机和程序员头上。虽然有时候的确是他们不好，但很多时候，冲突都是由文化差异导致的理念不同所产生的。

可惜，大多数自认为“会写程序”的程序员都会狭隘地给异己者贴上“错”的标签。这是人的本性，是我们最容易犯的错误之一，我自己也不曾幸免。

刚刚开始发牢骚的时候，我还是个丑陋的美国佬，在帖子里上蹿下跳，除了大喊“你们这帮家伙到底在搞什么”什么都不会。不过在接下来的十年里，我觉得我渐渐变成了一个业余的软件人类学家。现在我非常赞同文化相对论，尽量不去对那些和我意见不同的人下结论。

当然我不会因此就放弃开他们玩笑的机会，同样我也不介意别人来拿我寻开心。我最终希望能说服那些还在摇摆的程序员认同我对编程的看法，因为编程和开

车一样，只有在大家步调一致的时候才是最好的。所以我会继续宣扬自己的观点，也就是所谓的“软件自由主义”，它是完全合理的，甚至可能是很多软件开发都应该采用的方式。

妄图让所有人都更自由肯定是行不通的，这一点毋庸置疑。即便如此，我还是希望能帮助来自不同软件文化的人们更好地理解对方。

我会继续摇旗呐喊，因为这似乎是可以让大众听到我声音的唯一的方法。现在还有人会跟我说我的博客写得太啰嗦了，他们觉得我的观点完全可以在 100 个字里表达清楚。我发现这些评论主要是来自我的反对者，其实他们真正想要的是在反驳我的时候可以少写点字。不过一些赞同我的人也在抱怨，觉得我的博客太长，没办法抓住他们的注意力。在这里我要说，没抓住重点的人是他们，我的博客根本不长，没有足够的“分量”是无法直指人心的。经过这么多年的试错，我发现最容易抓住听众的办法还是讲故事。而不深入其中，享受过程，是讲不好故事的。

这就是本书的大致内容了。它是由一系列故事组成的。它们形式各异，可能是文章、论文、指南、抱怨，也可能是小说。但不论文体是什么，每篇都和你分享了一个故事。就算你不会同意我所有的观点，至少我希望你能喜欢我的故事，若还能让你觉得豁然开朗的话，就再幸运不过了。

Hyperink 的编辑们挑选了要收录的帖子，而且大部分章节段落的组织都是他们负责的。虽然我自己也做了一点修改，但是你现在所看到的这本书基本上是根据他们的想法组织起来的。他们干得非常漂亮。

愿你能和我一样享受这段旅程。

Steve Yegge

2012 年 8 月

目　录

第 1 章
Chapter 1

编程语言里的宗教

作者手记：巴别塔

这篇是本书最老的文章，写于 2004 年 9 月，当时我已经在亚马逊干了差不多 6 年了。当时亚马逊正饱受其庞大代码库的困扰，我曾经一度认为它的代码库规模失控是因为语言问题，后来才意识到企业文化是主因。

首当其冲的是，亚马逊的主流语言里有两门非常啰嗦的语言 C++和 Java，外加一门精练的语言 Perl。但是 Perl 正受到排挤，渐渐退出主流。我觉得这是因为 Perl 程序员能用更少的人力完成和 Java/C++程序员同样的工作量，所以要是比人多的话，他们注定是赢不了的。根据我们的估算，亚马逊的代码量比它的功能数量膨胀得更快。

第二个因素是，亚马逊的很多技术问题完全可以用自定义领域语言（DSL）的方式来解决。比如大规模的查询、分布式计算、产品配置等，他们写了太多不必要的代码了。我后来跳到 Google，发现他们为这些完全一样的问题专门编写了强大的自定义 DSL。这证实了我心中的疑虑，亚马逊的工程师在这些问题上和无头苍蝇没什么两样。我敢说这句话误伤的概率极低。

最后一点就是，和绝大多数公司一样，亚马逊非常抗拒用新语言来解决问题。他们会避免使用表达能力更强的通用语言，比如 Ruby 或 Erlang。他们也几乎从来不会想到自己去写 DSL。

结果就是，我知道他们的问题在哪里，也知道怎么解决这些问题，但是我的主管和经理们完全不买我的账。除了少数例外，大多数人都抱着非礼勿视，非礼勿听的态度：任何问题都可以用C++来解决。要是你喜欢Java的话，也同样照此办理。其他的办法都不在考虑之列。他们连听一听的意愿都没有。

终于有一天我忍无可忍，决定彻底发泄一下对这些亚马逊同事的不满。

这大概是我第一篇认认真真写的博客。在这篇名为“牢骚”的文章里我承认自己有点失控，花了整整一半的篇幅来发牢骚。当时肯定是没人读的啦。但是这些年来，很多人都跑来跟我说，正是因为读了这篇文章，他们才决定花毕生精力去掌握Emacs和Lisp——其中不乏高一新生！

巴别塔

我在这篇文章里会大致谈一谈对各种语言的看法——本来是想给这个月的亚马逊开发者期刊投稿的，后来却发现改来改去都难以示人。

一方面是因为，我时不时地就用上一些粗鲁的字眼，说出一些得罪人的话来，实在是不适合在亚马逊的官方刊物上发表。所以我还是把它贴在博客上，反正也没人会看。除了你。没错，就是你。你好！

另一个原因是，我还没写完呢，通篇都是东一榔头西一棒子的片段，完全没有经过润色打磨。这也是把它放在博客里的又一个理由，不用去考虑漂亮和完整性的问题，可以想到哪儿就写到哪儿。爱看不看。

这场旋风之旅涵盖了C、C++、Lisp、Java、Perl（这些是亚马逊在用的语言）、Ruby（因为我就是喜欢这门语言），还有Python（提到它是因为……唔，我还是先卖个关子吧）。

C

C是必修课。为什么？因为就一切实际用途来说，这个世界上你遇到的每一台

电脑都是冯·诺伊曼结构的，而 C 以精悍的语法展现了冯·诺伊曼机的能力。

今时今日，冯·诺依曼体系结构就是计算机体系结构的标准：一个 CPU、RAM、一个磁盘、一套总线。多 CPU 其实并不影响其本质。冯诺伊曼机是一个在 20 世纪 50 年代就实现的图灵机（这是一个进行计算的抽象模型，非常有名），它很实用，性价比也很高。

其他类型的机器也是存在的。比如 Lisp 机，它实现了 20 世纪 50 年代的 Lisp。Lisp 是一种基于 lambda 演算的编程语言。而 lambda 演算则是另一种进行计算的模型。与图灵机不同的是，人也可以理解和编写 lambda 演算。不过这两种模型是等价的。它们都精确地描述了计算机的能力。

除了在跳蚤市场，Lisp 机并不常见。相比之下，冯·诺伊曼机的接受程度要高得多。此外还有诸如神经网络、细胞自动机等各种类型的计算机，只不过它们都谈不上流行，至少目前还没有。

所以你是躲不开 C 的。

还有一个原因就是，Unix 是用 C 写的。不仅如此，包括 Windows 等在内的几乎所有的操作系统都是用 C 写成的，因为它们全部属于冯·诺伊曼机操作系统。你觉得自己还有其他选择吗？至少在操作系统领域里，任何与 C 迥异的语言都发挥不出硬件的实际能力——至少这句话放在近一百年里都是对的，这些系统都诞生于这段时期内。

Lisp 是另一门必修课。倒不是说真正干活的时候要用到它，只不过要是了解一点的话，在遇到很多 GNU 的应用程序时会觉得很趁手。特别是应该学一下 Scheme，它是一种短小纯正的 Lisp 方言。对应的 GNU 版本则名叫 Guile。

麻省理工和伯克利的新生都会学一到两个学期的 Scheme，但是没人知道为什么要学这样一门怪异的语言。老实说，它其实并不适合作为入门语言，可能将它作为第二门语言也不是好选择。虽然是一定要学的语言，但是等一等也无妨。

Lisp 不好学，它的门槛很高。用 C 的思维来学习编写 Lisp 程序是不够的，而

且也没什么意义。C 和 Lisp 是两个极端，它们都擅长对方不在行的事情。

假如说 C 最擅长的是映射计算机是如何工作的话，那么 Lisp 最擅长的就是映射计算是如何进行的。你真的不需要太深入 Lisp，只要掌握 Scheme 就足够了，它最简单、简洁。其他 Lisp 都已经变成了像 C++和 Java 那样复杂的编程环境，附带了一大堆库和工具，那些都是你不需要了解的东西。你要掌握的是用 Scheme 来写程序。如果你能做完《The Little Schemer》和《The Seasoned Schemer》后面的全部习题，那水平在我看来就已经绰绰有余了。

在选择日常的工作语言时，标准应该是它提供的库、文档、工具支持、操作系统集成、各种资源，以及很多其他和计算机怎么工作没什么关系，但是和人怎么工作大有关系的东西。

今时今日，C 的应用依然十分广泛，它是必须掌握的语言！

C++

C++非常冷漠，可以说是地球上最糟糕的语言。

它连自己是谁也不知道，完全缺乏自省的能力。好吧，C 也做不到，但是 C 不是“面向对象”语言，而让程序了解自身对面向对象来说是非常重要的。对象也是参与者。所以作为一门面向对象语言，一定要有运行时反射和获取类型的能力。而 C++不具备这种能力，好吧，应该说不是真的具备这种能力，至少你不会想要去用它。

对于 C 来说，你可以很方便地写一个 C 编译器，然后在 C 之上构建一些工具，让它具备自省的能力。可是 C++基本上是无法解析的，因此，假如你想要写个很聪明的工具，告诉你虚拟函数的签名是什么，或是帮你重构代码的话，就只能去用别人写好的工具集，因为自己解析实在太麻烦了。可是目前所有解析 C++的工具集都不好用。（作者注：现在 clang 还算不错，不过距离我写这篇东西已经 8 年多了。抗战都打完了啊！）

C++非常愚钝，用愚钝的语言是写不出聪明的系统来的。语言能塑造世界观。愚蠢的语言只能创造愚蠢的世界。

一切计算都是以抽象为基础的。高阶必须构建在低阶之上。直接在分子的层面上搭建一个城市是不现实的。采用过于低阶的抽象纯粹是自找麻烦。

麻烦来了。

C 能负担的最大项目应该是操作系统，老实说，操作系统并不是什么庞然大物。它们之所以看起来很恐怖，都是因为系统之上的应用程序，其实内核本身是很小的。

而 C++的最大负荷……也是操作系统。好吧，或许还能再大一点。就算 3 倍大好了，10 倍也行。操作系统内核最多有多少行代码？100 万行差不多了吧。那么 C++可以驾驭的系统规模大概在 1000 万行代码，超过这个数字后，系统就会开始失控，就好像《异形奇花》里的那棵植物一样。“我要吃东西……”

这还是在假设你能成功编译的情况下。

我们拥有 5000 万行 C++代码。不对，这个数字现在肯定更大了。我都不知道有多少了。5000 万是 9 个月之前的数字，它还在以每个季度 800 万的速度增长，而且增长的速度也在加快。天哪……

这里任何事情都进展缓慢。曾经有个亚马逊工程师把我们的代码库比作“像山一样高的排泄物，规模超过你见过的任何山脉。每当你要修复什么东西的时候，非得爬到最中间的地方才行”。

这是 4 年前的事情了，伙计们。现在他已经跳槽到更好的地方了。他是非常出色的程序员，你说可惜不可惜。

这都是 C++的错。别跟我吵。就是它的错。我们用的是全世界最愚蠢的语言。你不觉得这简直蠢到家了吗？

尽管如此，漂亮的 C++代码还是可以写出来的，这样的代码绝大部分都是 C，外加一些 C++特性，而且用得优雅，有节制。可惜这几乎是痴人说梦。C++是个巨大无比的坑，越是了解它，优越感就越强，最后一定会忍不住要用上各种特性。可是要用得好真的非常难，这门语言实在是太糟糕了。不管你有多牛，最后肯定会弄得一团糟。