2 Brad Fitzpatrick

丁雪丰 译

Brad Fitzpatrick是所有受访者中最年轻的一位，也是其中唯一一位从未在没有因特网或个人电脑的世界里生活过的。他出生于1980年，很早就开始了自己的程序员生涯，5岁时就在一台自制的Apple II克隆机上学习编程。在十几岁时，正好赶上因特网革命的大潮，他一头扎入其中，在高中时就建立了自己的第一个商业网站，在进入大学前的那个夏天创立了著名社区LiveJournal。

LiveJournal的日渐流行迫使Fitzpatrick走上了学习构建可伸缩网站的艰

难之旅，期间他和他创办的Danga交互技术公司里的程序员们开发了几个开源软件，其中包括memcached、Perlbal和MogileFS，现在被用于很多世界上最繁忙的网站的服务器上。

Fitzpatrick是个典型的极有才华的世纪之交的Web程序员，他的主要编程语言是Perl和C，需要时也会用Java、C++、Python、JavaScript和C#。他做的所有编程工作基本都与网络相关，比如为网站构建更好的后端基础设施，设计协议和软件来让博客阅读软件获知博客更新，甚至为他的手机编写代码以便在摩托车上就能自动打开车库门。

我们将谈到他在读著名儿童系列丛书*Clifford the Big Red Dog*的年龄就开始学习编程，为什么能够很高兴地一边念大学，一边运行LiveJournal，以及他是如何学会不惧怕去阅读他人的代码的。

Seibel：你是怎么成为一个程序员的？

Fitzpatrick：我父亲曾在Mostek[①]工作。这个公司是制造内存的，他对电脑很感兴趣。他做了台Apple II电脑，材料几乎都是多余的废弃部件。他和我

① 集成电路制造商，成立于1969年，在其巅峰时期曾占据全球85%的DRAM内存芯片市场份额。

母亲坐在电视机旁把部件焊起来，这个工作花了他们好几个月，只是把它们焊起来而已。然后我父亲从公司里拿了些不能卖的ROM，这些ROM有一位或几位不能用，有的在高位，有的在低位。后来不知怎么着他们弄到了Apple II ROM，接着就不停地将ROM烧到无法工作的芯片上，直到找到一块能用的为止，损坏的那位正巧是好的。最终，他和他的一帮同事终于做成了自制Apple II。我差不多从两岁起就在上面玩或者看他编程。

Seibel：他是个程序员还是个硬件工程师？

Fitzpatrick：他是个电气工程师，偶尔也写写程序。我五岁时他就教我编程，搞笑的是我六七岁时就超过他了。我母亲说我是一边读*Clifford the Big Red Dog*，一边读从图书馆借来的Apple II程序员手册。我会把“变量”念成“贝量”。我早期的一些记忆就是和父亲一起编程。比如他把我拖进厨房，在纸上写下一段程序，问我：“你觉得这段程序是什么意思？”我记得那程序好像是“10 PRINT HELLO, 20 GOTO 10”。

Seibel：那么说你是从BASIC开始的？

Fitzpatrick：是的，就是BASIC。我当时还不能使用鼠标、高级图形模式和彩色，直到我们家的一个朋友向我介绍了C并给了我Turbo C。那年我大概八

岁或者十岁。我父亲在1984年去了Intel，我们就搬去了波特兰。他帮助设计了386和486，现在仍在Intel。我们总是能有新的有趣的电脑。

Seibel：那你有没有试过汇编语言呢？

Fitzpatrick：我在计算器上做过些汇编，比如TI计算器上的Z80，但仅此而已。

Seibel：你还记得是什么吸引你开始编程的吗？

Fitzpatrick：我不记得了，只是好玩吧。我母亲不得不限制我使用电脑，好让我出去和朋友们一起玩。我的朋友们会跑过来说："Brad又在玩电脑。他太无聊了。"我母亲则会对我说："到外面去玩吧。"

Seibel：你还记得写的第一个比较有意思的程序吗？

Fitzpatrick：我们以前有台Epson打印机，它配有几本又大又厚的手册，手册最后是程序员指南。我就在Apple上写了点东西，我可以在高级图形模式下画些东西，当程序完成绘制（线段、图案或别的什么）之后，按下Control C，在后台一个不会显示的帧缓冲区里键入一段内容，加载另一个程序，它会读取屏幕并打印出来。

在那之前，我记得还写过一个程序，每当我敲击一个键，它就移动稿面，我按退格稿面会向回退，这样打字时就感觉像是在用打字机一样了。

这是我的第一个程序，好比方K是抓取的下一个字符，如果K等于a，打印a；如果K等于b，打印b。我几乎处理了每个字母、数字和一些标点。后来一个念头一闪而过：“等等，我可以说‘打印一个变量！”然后用1行代码替换掉了40行。“天啊，这太棒了！”对一个六岁的孩子来说，这已经是抽象能力的极限了。

那些都是比较有意思的早期作品。到了中学，我开始开发游戏，为朋友们制作图形编辑器和关卡编辑器，他们会把图形做进关卡中，随后我们再把它卖给其他同学。我记得我不得不检测EGA和VGA。如果VGA跑不了，就退回到EGA模式，使用另一组适合当前屏幕的贴图，为此我们必须为所有的东西做两组图形。学校的人会出大概5美元买它，然后安装，接着发现它不能用，他们的家长就会给我爸妈打电话大喊：“你儿子拿那没用的东西从我孩子那里骗了5美元。”我母亲就开车带我过去，坐在死胡同里等我进同学家调试并修复我的程序。

Seibel：那段日子里你有没有上过关于编程的课？

Fitzpatrick：没有。就是从图书馆里借了一两本书，然后随便玩玩。当时没有真正的论坛或因特网。后来我连上了一个BBS，但上面并没什么内容，它

没有联上因特网，只是一群人在玩棋类游戏而已。

Seibel：你的学校有AP[①] C.S.（计算机科学）或类似的课程么？

Fitzpatrick：呃，我们没有AP C.S.课程，但我们有计算机程序设计课。有一个老师在上这门课，不过我都可以在教室后面讲高级课程了。他们还在用我写的图形编辑器和图形库，他们要做的项目是开发一个游戏。我偶尔还会碰到那个计算机老师，他是我家的一个朋友，我会在我哥哥的足球比赛上看到他，他会说："是啊，我们还在用你的库呢。"

我的确参加了AP C.S.考试，那是在考试从Pascal换成C语言前的最后一年，一年后又从C换到了Java之类的语言。我不懂Pascal，所以去附近有AP C.S.课程的高中上了一些夜间班，大概三四次吧。后来我找了本书来学Pascal，我把大多数时间花在用Pascal画星形线上，因为那时我刚学三角学。我会说："哇哦，正弦和余弦太有趣了。我又可以一显身手了。"

Seibel：那你考得怎么样？

Fitzpatrick：我得了5分[②]。考试内容是写一个大整数类。现在这是我招聘人

① Advanced Placement项目，在美国和加拿大的高中提供大学级别的课程。

② AP考试最高5分。

员的一道面试题：“写一个能够进行任意大整数乘法及除法的类。”既然我能在高中的一次AP考试中做出这道题，那么他们在这儿应该也能做得出来。

Seibel：你大学第一年的夏天在Intel工作，那在高中时期有没有做过程序员的工作呢？

Fitzpatrick：是的，我在Tektronix[①]工作过一段时间。在正式工作之前，我有几个主机账号。我写了些机器人程序，往聊天室里灌水，把AOL惹翻了，这么做确实让人讨厌。我在另一个Windows程序中编写AOL客户端脚本，还写机器人程序猛提交AOL的线上表单，获赠CD。因为不想让他们发现这是重复表单而只寄一张CD，我用了自己名字的各种变体。这些账户有100个免费小时，或者是5000个免费小时。在这几千次表单提交后，整整一个星期，邮递员会天天带着很多CD过来。

我妈妈说：“见鬼，Brad，你会有麻烦的。”我回答：“呃，这是他们的错，对吗？”后来有一天，有一个找我的电话，我接了（通常我不去接电话），电话那头是一个AOL的人，他对我大叫：“别再向我们提交表单了！”我平时的反应并不敏捷，但这次我立刻回敬了他“：为什么你们会寄给我这些废物？

① 测试、测量与监测领域的知名公司。

每天都有邮递员过来，扔下这些CD！”他说：“对不起，先生。不会再发生这样的事情了。”然后我把这些CD都用来布置大学宿舍了，它们现在还放在我家车库的一个盒子里呢。我记得它们曾是很好的装饰品，所以没有扔掉。

恶搞完AOL后，我得到了一个当地ISP的shell账号。大概上我就是在那时学的Unix。虽然不能运行CGI脚本，但我可以用FTP上传东西，所以我就在家里的台式机上运行Perl程序来生成整个Web站点，接着再上传到服务器上。后来我在Tektronix得到了一份工作，类似暑期实习。当时我已经很懂Perl和Web了，但还没有做过动态Web。在1994年、1995年时，Web还是个很新的东西。

我去Tektronix上班的第一天，他们给我介绍我的办公用品：“这是你的电脑。”那是一台大的SPARC工作站，也可能是别的什么运行了X和Motif的机器。“这是你的浏览器。”可能是Netscape 2，我记不清了。“如果你有CGI的东西，放到这个目录里。”我记得那天晚上我写了个最基本的Hello World的CGI程序，大约就三行吧，我感叹道：“天哪，这太有意思了。”第二天早上六点我就来工作了，继续疯狂地写CGI。

后来我开始自己做动态Web编程。当时我找了台支持CGI的Windows

Web服务器。我最终说服了我的ISP（也许是和他们交情不错，或者之前给了他们不少帮助让他们信任我）：“OK，我们会运行你的CGI，但开始前会对它们做审核。”他们仔细查看了代码，然后把那些代码扔到他们的目录里。那是一个投票站的脚本，你可以用它来创建投票程序，例如：“你最喜欢的电影是哪部？”添加完选项后就能开始投票了。后来的几年里它变得越来越流行了。

Seibel：是FreeVote吗？

Fitzpatrick：是的，在弄爆我的主机后它变成了FreeVote。那时Banner广告真的很流行，也许就是那段时间开始流行起来的，我通过它赚到了越来越多的钱，得到越来越好的合同，每次点击的收益也越来越高。最高时每点击一次广告我就能赚到27美分，就算是按今天的标准，我觉得也是相当夸张的。有了它，我每个月在Banner广告点击方面能有2.5万～2.7万美元的收入。

这些都是在高中时做的，我整个高中时期都在私下做这事。我还在Intel做了两个夏天，随后在进入大学前的最后一个夏天，我办起了LiveJournal。为此，在进入大学的第一年，我卖了FreeVote，基本上算是送给一个朋友的，大概只要了1.1万美元，因为我想摆脱它，还有相应的法律责任。

Seibel：你有了ISP并开始使用Unix，这有没有对你编程带来什么变化？

Fitzpatrick：Unix并没让我抓狂。我没办法理解Windows上都发生了什么，也许你看过Windows API——每个函数都有差不多20个参数，它们都是标志位，而且一半都赋了0值。完全不知道发生了什么。当有东西无法正常工作时，你根本没办法知道究竟是怎么回事。

Seibel：你早期的编程方法或编程风格和你现在所想的有什么明显的不同之处吗？

Fitzpatrick：我用过很多种编程风格，面向对象的、函数式的，现在用的这种有点怪异，混合了面向对象和函数式编程。这是我热爱Perl的原因，虽然语法很丑陋，有很多历史包袱和瑕疵，但它从不限制我写代码的风格。用你喜欢的风格去写就是了。你能让代码优雅一致，却没有和特定语言相关的风格。直到我进入Google，我写的Perl代码才慢慢少了下来。

运营LiveJournal后，我也做了很多测试工作。尤其是当我开始和他人共事时，我意识到永远也甩不掉自己写的代码，要一辈子维护它们时，我开始写测试了。在十年前的博客文章上，我收到了这样的评论：“嘿，我看到这段代码，发现了一个问题。”然后我立刻着手维护代码。

我现在维护着很多代码，还有其他很多人在和它们打交道，如果有什么地方不清楚，我会假设有人理解不了我写的某些不变式。因此，当我要搞些小聪明时，通常我都会保证在代码出现问题时有测试及时跳出来。我也强迫其他人写测试，他们基本上都为我工作。我会为自己的代码写测试，当别人写代码时我会告诉他们："你确定这段代码能工作吗？写个测试证明给我看。"有时，他们会意识到"天哪，这么做太有用了"，尤其是在后期维护的时候。

Seibel：你是从什么时候开始和他人一同共事的？

Fitzpatrick：差不多是在大学结束的时候，我开始雇用其他人，尤其是毕业搬回波特兰后。

早期的雇员是客户支持，所以他们不用写任何代码。慢慢地，我开始雇用程序员。我雇佣的第一个人是我的一个网友，他的名字是Brad Whitaker，我们都有名为BradleyLand或BradleyWorld的网站，因此我们找到了对方的网站。我比他早几年开始Web编程，也可能是早一年，他问我："嘿，你的网站是怎么做的？"就是说它到底是HTML、Frame、CGI还是Perl的。后来我开始接很多合同项目，我就把一些我不做的项目给他。有一次我们有个大项

目，谁都没有办法独自完成，我找到他说：“这个项目需要两个人来做。”他让我飞去宾夕法尼亚，也许是匹兹堡？我对东海岸完全没概念，我是个生活在西海岸的人。也许是费城？有牛肉奶酪三明治[①]的地方。

Seibel：是费城。

Fitzpatrick：是的，我们第一次见面是在一家便宜的旅馆里，我感觉好像早就认识他一样。他和我打招呼：“嘿，最近怎么样？”他走了进来，在我旅馆的卫生间里上了个厕所，当时我就站在那儿，可他连门都不关。我回答道：“还不错。你还真惬意。”虽然我们从未谋面，可就像认识了四五年一样。然后，我们就开始干活了。

他住到我空余的一间卧室里，我们差不多把厨房给搬空了，架起几张桌子，放上电脑就开工了。我们通常10点或11点起床，干到中午，看会儿电视（穿着短裤坐着看电视），然后不间断地工作到早上三四点钟。后来，我的另一个朋友从华盛顿大学过来度暑假。他是我大学一年级后认识的，我们三个在一起忙碌着。这个朋友住在市区，早上坐轻轨过来，然后滑滑板到我家。他就坐在外面用Wi-Fi上网、写程序，直到我们醒过来去给他开门，让他进来。

一起有了三个人，房子就有点挤了，我就说：“哦，好吧，我们搞间办公室吧。”于是我们弄了间办公室；“我们既然有了这些空间，不如再雇点人吧！”在随后的两年里，我们慢慢扩大到了12个人，而LiveJournal也逐渐流行了起来，当然压力也更大了，因为我还得处理人事问题。

后来，我妈妈来处理人事，随后我们的关系就有点紧张了，因为她为我工作。我给她定了几条规矩：“如果你给我打电话，要分清楚是私事还是公事。要么只谈私事，要么只谈公事。你不能在工作与私人问题间换来换去的。”如果她转变话题，我就会挂电话。然后她再打回来，我会说“你搞混了”。

① cheesesteak，费城牛肉奶酪三明治，这是费城及其附近地区的食品。用一种细长的意大利面包，里面有切得很薄的肋眼牛肉片加炒过的洋葱及蘑菇，然后淋上融化的奶酪酱，也可加上红椒或者是很辣的小尖椒。（摘自维基百科。）

这真的搞得很紧张，当我把公司卖掉时她真的很高兴，她不用再为我工作了，我们也不用争吵了。

Seibel：那时你的公司还接合同工项目吗，还是说这就是整个LiveJournal了？

Fitzpatrick：差不多这就是整个LiveJournal了。我们还打算开个照片托管服务，这方面Flickr做的比我们好，我们的那个有些过度设计了：漂亮的抽象，能结合到任何东西中。为LiveJournal做的每个新的基础设施，我们都会问自己："它怎么和FotoBilder结合到一起？"为此，我们把每样东西都抽象出来。Memcached是抽象的，因为没有必要把它和LiveJournal绑在一起。随后我们还做了个类似GFS的文件系统，还有一个任务队列。为了提高可扩展性，我们不停地开发基础设施组件，它们能被应用于我们的各项产品之中，由于没有复杂的依赖关系，它们的维护也更方便了。虽然可能会增加一些工作量，但如果能减少依赖，那还是很值得的，所以我们开发了所有这些通用基础设施。

Seibel：我对于你扩展LiveJournal的过程有些好奇，你是从什么地方开始的，一路上又是怎么学到你需要的东西的？

Fitzpatrick：我们和其他客户共享一个Unix主机，差点把它搞挂了。

Seibel：以CGI的方式运行的吗？

Fitzpatrick：是的。我想从严格意义上来说那应该是个CGI，派生出所有的东西然后终止掉。ISP分配给我一个家伙。我的服务器当时总是死机，我对他说："我每月为这台服务器付10美元，它为什么不能工作？"他告诉我："哦，你该这么做。"很快我就学会了Unix，知道正在发生的事情。

我从CGI转到FastCGI，调整了Apache，关闭反向DNS查询功能。经过了这些步骤，性能有了好转。最终，我遇到了IO和CPU的瓶颈，我买了自己专用的服务器，但那也只是一台机器，它经常死机，而且我的硬盘快没有空间了。起先这台服务器只对我的朋友开放，我没有取消注册页面。他们邀请了他们的朋友，而这些朋友又邀请了其他朋友，但我并没想过把这个站点变成公开网站。它只是正好有个开放的注册页面。于是，我在LiveJournal的新闻页面里写了点东西："我们需要大家的帮助，我们需要购买服务器。"

我记得那次大概募集了六七千美金，我买了两台大的Dell服务器，托管在西雅图市区的Speakeasy[①]里。有人推荐了一些Dell服务器，这些6U的大家伙差不多有90磅[②]一台。数据库服务器和Web服务器从逻辑上是独立的。因为我运行了一个MySQL进程和一个Apache进程，所以我只知道这样去分离。

就这样过了一阵子。Web服务器直接面向大众，服务器上有两块网卡，通过交叉线缆（crossover cable）连接数据库服务器。后来Web服务器过载了，但那还比较好对付，那时我有了几台1U的服务器。之后我们有了3台Web服务器和1台数据库服务器。这时我先后用了三四个HTTP负载均衡器——

① 一家提供宽带接入、服务器托管等服务的公司。

② 1磅约等于0.454千克。

mod_bachhand、mod_proxy和Squid。这些我一个都不喜欢，我就是从这时开始讨厌HTTP负载均衡器的。

接下来过载的就是数据库，那时我抱怨道：“哦，见鬼。”Web服务器可以很好地扩展，它们都是无状态的，只要投入更多的机器，分散负载就可以了。那段时间很难熬，“我可以优化查询来应付一下，”但那只能坚持一周，一周后它又会过载。那时我就开始思考一个独立的请求到底需要什么。

那时我认为自己是世界上第一个想到这个方法的人，我们可以切分数据库，将它分区存放。我画了几张图来做设计文档，简单说明我们的代码会是如何工作的。“我们的主数据库只存放全局相关的元数据，这些是低流量的。各个博客和评论相关的东西会分别存放到每个用户的数据库集群里。每个分区都有自己的用户ID。”现在看来，每个人都会这么做。但那时候在不间断服务的情况下迁移代码着实是件很费力的事。

Seibel：在你迁移时有没有暂停服务？

Fitzpatrick：没有。每个用户都一个标志位，标明他在哪个集群上。如果标志位是0，说明他在主数据库上；如果非零，则说明他已经被分区了。有个版本号是“你的账号正被锁定”。这时该账号处于锁定状态，尝试迁移数据，

如果在这时你做了什么变更则需要重试。基本上在我们完成迁移前你在主库上不能进行写操作，迁移完成后声明：“OK，现在你的账号可以使用了。”

迁移程序在后台运行了两个月。我们计算过，如果只是把数据导出来，写点程序拆分SQL文件随后重新加载，可能要花一周左右。摆在我们面前有两个选择——停机一周或者缓慢迁移两个月。我们先迁10%的用户，这时对其他用户而言，整站的可用性提高了，随后我们再慢慢提升迁移到集群的用户比例。

Seibel：这就是memcached和Perlbal的前身了吧。

Fitzpatrick：是的，那确实是Perlbal的前身。memcached是在那之后的事情了。在大学结束离开学校前，我都没想过要做memcached。站点越来越慢，一天我在浴室里洗澡的时候灵光乍现，突然意识到我们有这么多空闲的内存可以利用。那晚我写了个原型，服务端和客户端都是用Perl写的，服务端很快就崩溃了，因为对于一个Perl服务器而言CPU的使用率太高了。于是我们着手用C来重写它。

Seibel：这样就省去了购买更多数据库服务器的开销了。

Fitzpatrick：是的，数据库服务器又贵，迁移又慢。Web服务器非常便宜，

把它们加上马上就能见效。如果你买一台新的数据库服务器，差不多要花一周来进行配置和验证：测试磁盘、配置和调优。

Seibel：这么说来，你所开发的所有这些基础设施，比如memcached和Perlbal，都是为了响应LiveJournal的实际扩展需要？

Fitzpatrick：是的。我们开发这些东西都是因为LiveJournal承受不了负载，我们挑灯夜战开发新的基础设施。我们甚至还买过NetApp[①]。我们问：“这要多少钱？”他们回答：“说说你们的商业模型。”“我们有付费账户。”“你们有多少客户？怎么收费？”你就能看到他们在那里做乘法。“价格是你们在不破产的情况下的所有可支配收入。”我们心说：“去你的。”但我们确实需要它，所以还是买了一台。我们对它的I/O性能并不满意，它不仅很贵，还会形成单点故障。他们试图卖给我们一套高可用配置，我们想：“去你的！我们不会再买这种东西了。”

后来我们开始写自己的文件系统。我不确定当时GFS的论文是不是发表了，我觉得我应该从谁那里听到过。那时我的内存很分散，取个散列的键，从分区里取值。为什么不能把这招也用在文件上呢？文件是永久性的。因为

① 知名高端存储和数据管理品牌。

增加存储节点时配置会发生变化，所以应该记录下文件的实际位置。那用不了多少I/O，只需跟踪文件的位置就可以了，但如何保证高可用性呢？我们想出了一个解决方案，我提出一个计划："这是我们确定文件位置所需的全部的读和写。"我先写了主控的MySQL Schema和文件位置追踪器。随后，我发现："上帝啊！这部分用HTTP就能搞定了。它根本不难！"

我记得在想了整夜之后，我们就投入工作了。我们在公用办公楼下面有个会议室——一间昏暗的大会议室。"好了，各位，停下你们手头的事情。我们下楼去画图。"每次我有设计思路时差不多都会这么说，我们应该找块白板把它画出来。

我解释了整个设计，谁和谁交互，谁处理请求。然后大家就上楼了，我先预定了所有的硬件，因为差不多要两周左右它们才能到货。然后开始编码，我们希望能在机器到货前完成编码工作。一切总能挑出毛病，总有东西出问题，因此我们总在写新的基础设施组件。

Seibel：有没有这样的情况，某人在开始时让你坐下，告诉你"你需要知道*X*、*Y*和*Z*"，你的生活会不会更容易些？

Fitzpatrick：从一开始就把事情做好总是比迁移一个线上服务要容易很多。

这一直都是最让人头痛的。我所说的每件事，你都能在一台机器上做到。开始时就这么设计，你就不需要假定能把这两个用户数据关联起来或是别的什么了。假设你想加载这20条数据——你的实现可以把它们从同一张表里加载上来，但在更高层次的代码里只需要说一句“我想要这20个对象”，就能实现从一组机器中收集数据的工作了。如果从一开始我就这么做，可以免去不少迁移的痛苦。

Seibel：这么说，基本上你学到的就是“要为数据无法容纳进一个数据库的那天做准备”。

Fitzpatrick：我认为这已是Web社区中的常识。人们总是假定他们的站点会变得很大，有些想过头了。但在那时，常识却是，Apache和MySQL就是你需要的一切。

Seibel：看起来你写这些组件是因为你需要它们，同时你也乐在其中。

Fitzpatrick：哦，当然！我确实是在找理由去使用各种东西，去学习它们。因为如果不实际用它写点什么，不和它生活在一起，你永远学不到东西。出于兴趣去学一门语言和学会它是两回事，如果不用它写些大的、复杂的系统，那你不能算是真的学会了。

Seibel：那你觉得对你而言，你和哪些语言生活在一起呢？

Fitzpatrick：Perl、C。以前的话，还有BASIC，不过我不确定BASIC算不算。我还写过很多Logo。在小学的Logo课上，大家都在提笔、落笔，而我不在图形模式里——有些键能跳出图形模式，我在写函数。老师会走过来说："你在干什么？你做的不对，你应该在画房子。""不，我在写Logo。你看。""不，你弄错了。"在课程结束的时候，我做出了些东西，我写了个类，能以任意尺度和任意方向画出所有字母。有了它，我能在波状横幅上打印整条消息，并加入距离和填充。每个人都很吃惊："这是什么？"我也不知道这个算不算。

现在在用很多的Perl和C，在大学里还为工作和Windows程序写了很多C++。后来我的C++忘得差不多了，或者说是退化了。现在我在Google，去年我写了很多C++、Python和Java。在Java刚出来的时候我也写过不少，但后来厌倦了。现在我又开始写Java，又开始有些不爽了。

Seibel：使用何种语言很重要吗？

Fitzpatrick：目前还没找到能让我满意的语言。我还不太清楚究竟什么才能让我完全满意。我讨厌在一个项目里一直切换语言。我想要的语言要在我需

要时拥有静态类型，在编译时检查所有内容。Perl比较接近这个要求，我能用各种喜欢的方式来写代码。它在编译时没有足够的静态检查，但我可以在运行时对其进行补充。不过Perl还是不够好。

我想要可选的静态类型。在Perlbal里，除了核心（要复制字节），没理由让半数的东西都有这么高的性能。我希望在特定的部分代码和类型声明时能给运行时一些提示。但如果想使用惰性求值或模拟一些东西时，我可以采用另一种方式。

Seibel：这么说来，你主要就是希望语言能有静态类型，这样编译器能更好地进行优化？

Fitzpatrick：不是这样的。我还希望在编译时它能告诉我“你正在做傻事。”有时我并不关心编译时的情况，我希望它能在运行时有些强制措施，能做任何事。我并不想对Perl 6过于乐观，他们的确在讨论很多我希望看到的东西。但我并不认为它们最终能实现出来。

Seibel：你喜欢C++吗？

Fitzpatrick：我并不是很喜欢它。C++的语法很糟糕而且并不一致，它的错误消息也很可笑，至少GCC的错误消息是那样的，往往会因为漏了一个分号

而显示40页的错误信息。但和别的东西一样，你很快能记住各种模式，之后甚至不用看那些信息就能知道“哦，我大概忘记在头文件里关闭名字空间了。”我认为新的C++规范，尽管加入了很多复杂性，但还是添加了不少能减少打字痛苦（击键次数）的东西。例如，自动变量和for循环，它的风格变得更像Python了。还有lambda表达式，尽管用的是C++，但这足以让我误以为自己在写Python了。

Seibel：你是出于性能目的使用C++的吗？

Fitzpatrick：是的，差不多吧。我在Google主要使用C++。各种有性能要求的东西都会用C++来写。我在Google还写了不少Java。

Seibel：据我所知，Google有以C++为中心的文化，因为那是Google最早使用的语言，而且他们围绕它构建了整套软件基础设施。你不能改变历史，但在Google可能有很多C++代码对提升性能而言并不是必需的。

Fitzpatrick：原因是这样的，随着时间的推移，Java越来越快了，JVM越来越聪明了。关于Java，有件事让我很不爽，那就是每个人都很讨厌JNI。有时一个库是用C++写的，Python开发者（不管Google里的还是外面的）并不在乎，他们会说：“哦，我们会用SWIG包装它。”他们继续自己的工作，做

得很开心。Python能很快从用C++写的东西中获得支持，因为Python开发者并不介意它到底是用什么语言写的。

Java的人却会说：“这必须得是纯Java的。我们不用JNI，否则若JVM崩溃了，我们根本不会知道是为什么。”这个问题最后会导致所有的东西都要写两次，一次为C++、Python和所有其他语言，另一次专为Java而写。如果他们能讨论出一个好的嵌入方案或者克服对JNI的恐惧，那我就没什么好说的了。

Seibel：那你觉得内存管理和垃圾回收相比怎么样？人们还在争论这个问题，你有没有什么强烈的倾向呢？

Fitzpatrick：没有。看见人们怀着那种强烈的倾向，尤其是通常他们背后并没有什么依据支持时，我还挺开心的。我个人并不觉得管理内存有什么讨厌的，至少在C++里用限定作用域的指针时是这样的。我能用C++连着写上几天代码，期间不用new或delete。看上去这什么都能做。

我在Google时重写了memcached，让它能同Google的基础设施协同工作，并将其加入到App Engine里。这里全部使用C++，因为我需要对内存进行独占式控制，这样可以减少碎片。我很欣慰C++能做到。

Seibel：memcached最早是用C写的。你把它用C++重写了是因为C++在Google被普遍接受，还是有什么别的优势？

Fitzpatrick：我开始时用现成的代码做迁移，但发现这样做工作量反而更大。memcached并没有太多代码，所以直接用C++重写能更快些。重写后代码差不多相当于以前的一半。

Seibel：那你认为这是因为用了C++还是你比当时更聪明了呢？

Fitzpatrick：有可能是我聪明了。记得我大约在11岁或12岁时，做了次环美旅行，我在一台TI-85计算器上写了个Mastermind游戏[①]。我在这个小屏幕上写了两百行程序来记住我在哪儿。最后我把这鬼东西删了两次，所以我一共写了三遍代码，但最后它变容易了。这是很有道理的，因为第二次做某件事时，它就变容易了。

Seibel：你的很多工作都是用Perl完成的，这是一种相对高级的语言。你认为程序员应该了解多少底层的东西？程序员还需要了解汇编以及芯片是如何工作的吗？

Fitzpatrick：我不知道。我遇见过很聪明的人，我觉得他们是好程序员，但

他们只懂Java。他们解决问题的思路被局限在他们的知识范围内。他们不会全面地思考问题。虽然你不用操作整个系统，但对其有所了解还是很有必要的。

我在做LiveJournal时，要考虑从JavaScript到如何与内核交互的所有内容。我读了Linux内核中关于epoll的代码，于是想：“假如我们持有全部连接到这台负载均衡器上的长TCP连接，JavaScript在轮询它们时又会怎么样呢？”我试着计算每个结构体要占用多少内存。这还是比较高级的，我们还会思考些这样的问题：我们的以太网卡有太多的中断，每当收到数据包时网卡都会发送一个中断，它们加到一起就达到网卡的极限了，就算是一块千兆网卡也只能达到百兆的速度，要是我们在内核里切换到NAPI技术会怎么样呢？我们还收集数据，看看什么地方做切换比较好，能释放处理器资源。

我们从这些真正低级的东西上受益颇多。最近有人对我说：“Java会考虑这些问题的，我们不需要处理它们。”我说：“不，Java不会处理这些，因为我知道你正在使用的内核版本，这个版本不支持这个特性。你的虚拟机把这些都隐藏了，它给你一个抽象，让你觉得它很有效，但实际上只有当你使

① 由以色列电讯专家Mordecai Meirowitz发明的经典益智游戏。

用特定版本的内核时它才会生效。”如果人们连整个系统的最表层都不明白，那我会感觉很失败的。

从实践角度来看，这些东西没一样是行之有效的。所有这些漂亮的抽象背后都一塌糊涂。那些看起来很美的库实现也很糟糕。所以说，如果你是那个对购买服务器的经费负责的人，或者对可用性负责的人（如果你在发生重要事件时要求随叫随到），那么去了解底层正在发生的事情，不轻信别人的库、代码和接口是非常有帮助的。

如果我是现在才开始编程的，我几乎不认为自己今天会变成一个程序员。这东西太丑陋了。这就是我对App Engine这类东西感到兴奋的原因。有人把Google App Engine说成是当代的BASIC，因为这个时代里一切都是联网的。在我写程序时，就只有一种语言，运行在我自己的机器上，部署是up回车或者RUN回车。现在的孩子不想在自己的机器上写些傻乎乎的东西，比如益智类游戏的“bounce a ball”应用。他们想要一个可以交互的Web站点。

我还是会收到一些人的邮件：“嘿，我有个主意，我想整合Wikipedia和YouTube，整合……”每个人都想做一个Web站点，他们最喜欢的4个Web站

点都不太顺眼，他们想做一个符合他们心意的。

事实上，App Engine给了你一个按钮，“把这个放到Web上”，你只需用一种语言来写程序，Python就很完美（相对比较易学，关于这点还有些争议）。这对编程而言是个很好的入门，我们有太多的分层，它摒弃了一些无用的层次。

Seibel：前面那个人告诉你“Java会替你处理它”，你感到不爽。那这不一样吗？“App Engine会替你处理它的”。

Fitzpatrick：我不清楚。也许因为这次我知道底层在发生的事情。实际上JVM也不是那么糟糕。我猜人们只是盲目相信他们的抽象，不清楚底层的东西。

Seibel：你在进入大学学习计算机科学前就有很多编程经验。你觉得大学的课程怎么样？

Fitzpatrick：我跳过了很多早期的C.S.课，因为它们真的很无聊。但我会去参加考试。后来开始上300-Level和400-Level[①]的课时就比较有意思了，可正当我有兴趣去学时，就毕业了。他们也不让我上研究生级别的课程，因为我不是研究生。

① 国外的课程在学科后跟上3位数字，100、200、300、400，数字越大表示难度越大。

我记得在讲编译器的课上，最终的项目是给我们正在使用的语言添加一组特性，还包含一个自选特性作为加分部分。我选择了实现运行时数组边界校验。教授会拿他的测试集去运行我们编译出的二进制文件，有些测试失败了。他说：“对不起，你的分数是C，因为你的单元测试失败了。”我看了下他的测试，说：“你的测试里有个off-by-one错误[①]。”于是他重新给我打分，我得了个A，但我没有得到为语言添加新特性的附加分。在学校那会儿我对此很生气。

我还记得教我们数据库的老师是一个看起来完全没有实际经验的人。那时我用过Oracle、Microsoft Server和MySQL。我会问一些我很希望得到答案的、我们网站遇到的现实问题，但他只是给我一些书本中的答案。我只能说：“不，不。那样没用。”

Seibel：你毕业于2002年。现在是否会对当时学校所教你的东西心存感激？

Fitzpatrick：半数的课我都很喜欢，无论是学习那些当时还不知道的新东西，还是学习背景知识或一些术语，都是如此。在此之前，我虽然很懂编程，但却无法用合适的词汇去描述我正在做的事情。如果我用自己的语言来进行描

① off-by-one错误，又称“差一错误”或“栏杆错误”，常见的关于边界条件的错误。详见

述，其他人会觉得我根本不懂我自己在说什么。早期的C.S.教育让我能更好地与人进行交流。

Seibel：关于在校期间经营自己的事业，你有没有感到后悔过？是不是应该从中选择一样来做？

Fitzpatrick：不，我觉得这是最好的方式。我别的朋友在大学里只是完成课程，而我对这些已经很了解了，只读书的话会很无聊。有个朋友，同样很懂这些课程内容，但他的想法是在学校就该学习，而不是为了考高分，所以他在业余时间学习阿拉伯语、汉语和日语，还有很多疯狂的编程语言。每周他都会说："我喜欢上了一门新语言，这周我只用OCaml来写东西。"他以这种方式让自己保持忙碌的状态。而我则选择了一种繁忙而又不怎么无趣的方式。

我还有些朋友在大学第一年就退学从事Web相关的工作。有对夫妻做着成人网站之类的东西，他们觉得"我们正在赚钱呢"。但他们只是在工作，在自己的地下室里忙碌着。大学校园里最棒的是认识人和参加聚会。如果我只是在做LiveJournal，我会把自己折腾死的。

---

http://en.wikipedia.org/wiki/Off-by-one_error。

Seibel：你喜欢学习计算机科学专业吗？

Fitzpatrick：不读这个专业我可能也干得挺好。我做了很多一般情况下不会去做的事，因此我想这个专业还不错。也许我还应该再学点别的，在学校多待一年，修个完全不相关的第二专业，多学点语言学。离开校园后我有些伤感，觉得自己才完成了一半的学业，因为有这么多东西都是之前就知道的。早期的C.S.课我基本都没去，等到后来课程变得有意思起来，他们却说："OK，你毕业了。"

Seibel：你有没有想过要读研？

Fitzpatrick：有，读研可能很有意思，不过我太忙了。

Seibel：你现在还会关注C.S.相关的文献吗？

Fitzpatrick：我和我的朋友们还是会互相转发好的论文。不久前，我刚读了一篇关于运行时调整布隆过滤器（Bloom Filter）大小的新技术的文章，这看起来太棒了。我会去读那些来自存储会议、业界和学院的论文，那些关于不同系统的论文。Google有很多阅读小组——系统阅读小组或者存储阅读小组。我会看Reddit上的文章、朋友转发的论文，或者博客上的链接。

Seibel：你刚提到来自学院和业界的论文。你觉得如今这两者是否有交叉的地方？

Fitzpatrick：它们对我来说都差不多。很多时候读业界的论文会更有意思些，因为你知道他们用它来解决问题，他们的解决方案绝对不是“ 如果……，我们认为会很酷”。学院派里也会出些疯狂的东西，根本不能用，只是些疯狂的想法。也许今后能把它变成商业化的东西。

Seibel：你是如何设计软件的？

Fitzpatrick：我会从接口开始，比如常用方法、常用RPC或者常用查询。以存储为例，我会试着去思考，常用的查询是什么？我们需要什么索引？数据是如何存储在磁盘上的？然后给不同的部分写点试验模型，再慢慢充实它们。

Seibel：你在测试中是凭直觉写吗？这样可以随心所欲地进行测试。

Fitzpatrick：越来越常这么做了，我总是这样设计软件的，甚至是在测试之前。开始时，我只设计接口和存储，随后再开始真正实现。

Seibel：你的设计是什么形式的？伪代码？实际代码？还是白板草图？

Fitzpatrick：通常我会打开编辑器，写下关于Schema的伪代码。等它差不多了，就写真实的Schema，再复制粘贴一下，看看“ create table”命令是不是好用。等一切都准备就绪，就开始动工实现它，我总是从spec.txt开始。

Seibel：在写了一大堆代码之后，有没有碰到过要重新考虑原始计划的情况？

Fitzpatrick：有时会这样。但我总是从最难或者最不确定的部分开始实现。我不会把任何难的或者让人吃惊的东西留到最后，我喜欢开始就做最难的部分。至于那些我没完成的项目，我的朋友会取笑我，不过我已经把难的部分做完了，我已经学到了想学的东西，那些无聊的工作就让它去吧。

Seibel：对于那些自学的程序员，你有什么建议吗？

Fitzpatrick：要试着做点更难的东西，超出能力范围的东西。要多读代码，我以前常听人这么说，不过那时一直没有静下心来去做。那么多年里我写了很多代码，可从来没读过别人的。后来我上了因特网，有那么多我可以贡献力量的开源代码，但我却被吓倒了，如果不是我自己的代码，脑子里没有对全局的认识，我根本无法深入理解它。

后来我开始给Gaim发补丁，这是GTK下的即时聊天工具，我通过阅读代码来了解整个设计。看了一部分之后，我明白了。我意识到看完别人的代

码后，并不需要记住它们，我开始去了解模式。看着他们的代码，我会说：“哦，OK。我明白代码的结构。”

慢慢地，我变得很喜欢读代码，因为当我遇到不理解的模式时，我会问：“等一下，他们为什么要这么做？”随后再多看看就明白过来了：“哇，这真是个好办法，很有效。”我应该早些开始，但我害怕，因为我觉得如果不是自己的代码，我就没办法理解它。

Seibel：你是如何阅读他人的代码的？比如先说说，阅读代码是为了全面了解它是如何工作的，还是自己想要对代码做些修改？

Fitzpatrick：通常我是想要修改一些东西。如果你崇拜某个程序员，也可以去读读他的代码。也许你会意识到他们也是凡人，不该成为你崇拜的对象。你也可能会从他们的代码中学到一些东西。

Seibel：你刚说到你想做些修改，一般你是怎么做的？

Fitzpatrick：第一步，找个原始的tar文件或者从SVN检出代码，试着把那该死的东西构建起来。你一定要跨过那道坎，那对多数人来说是最大的障碍——构建系统时的依赖或者开发者假定你已经安装了这个库。我希望这些大项目都能自带一个虚拟机，也就是它的构建环境。

Seibel：你是说像VMware那样的虚拟机？

Fitzpatrick：是的，如果你想立刻动手修改，这里有全部的依赖。人们的连接速度够快了，这是完全可以实现的。

不管怎么样，一旦你有了一个干净的、可工作的构建版本，干掉它，做一个修改。把标题栏改成“Brad says, ‘Hello world.’”，改变一些东西，即使一切都很丑，只要动手开始去改变就好。

然后，把你的补丁发出去。我发现这是最好的开始对话的方法。如果你在邮件列表里说“嘿，我想添加特性*X*”，维护者可能会回答：“哦，见鬼，我很忙，一边呆着去，我讨厌特性*X*。”但如果你找到他们后这样说：“我想添加特性*X*。我正在考虑与附件里的补丁类似的东西。”——当然那是错的，你说：“但我认为这完全是错的，我想正确的方式也许是做*X*。”然后给出个更复杂的方法，通常他们会回答：“上帝啊，他们试过了，看看，完全做错了。”

这可能会让维护者很苦恼，他们会说：“哦，我不敢相信他们花了这么多精力来做这个。如果方法正确，做这个很简单。”或者是：“哦，哇，他们完全做错方向了。我希望他们不要再继续了。”随后他们就会回复了。

这是开始对话的最佳途径。甚至在Google，这也是我与很多不认识的团队开始对话的方法。当我修复了他们产品的一个bug后，第一件事就是用邮件寄给他们一个补丁，告诉他们：“你们怎么看这个？”或者在内部代码审查工具里写道：“这是审查结果。你怎么认为？”他们可能会说：“见鬼，这段修正的代码完全搞错了。”

Seibel：你现在还会出于兴趣去读代码，而不是因为要用它才去读它？

Fitzpatrick：有时会。我会毫无目的地签出Android和Chrome的源码。当它开源后，我会镜像一个它们的代码库，然后随便看看。我对Firefox和Open Office也做过同样的事。一些你一直在用的程序，突然有一天你能访问到它的代码了，你也会去看一看的。

Seibel：那种程序的代码库非常大，当你出于兴趣去看东西的时候，会有多深入呢？

Fitzpatrick：一般来说，我只是大致地看一下，试着去了解目录结构。然后如果有东西吸引我的注意，或者我对什么东西不太理解，我就随便选个文件，感受一下。随后漫无目的地看看周围的文件，直到我厌倦了，再挑个新文件去看。

很多时候，我会边构建边读代码，因为通常这是可以并行的任务，尤其是当构建很困难时。完成构建后，如果我愿意，就可以开始调整它了。

Seibel：你读过好代码，它可能符合你已知的模式，或者你会发现新模式。但并非所有代码都那么好，坏代码最初的征兆是什么？

Fitzpatrick：嗯，我现在特别傲慢，在Google工作，对所有语言都有非常严格的风格指导方针。针对使用最多的六七种语言，方针里写道："我们是这样安排代码布局的。要这样来命名变量。要这样来使用空格和缩进，这是你要使用的模式和惯例，你要这样声明静态域。"

我们也会把这些指导方针放到网上，以供我们项目的外部贡献者作参考。我们想要有一份成文的指南，这样就不用再说"我们不喜欢你的风格"了。

现在，当我用C语言做项目时，第一件事就是添加一份风格指南。当项目成熟后，有很多人会来修改它，他们会有一份风格指南。也不是总能有这么一份东西，但程序员应该尊重已有的代码风格。也许他们不喜欢大括号风格，但要知道，在一个文件、一个项目中保持一致的风格要比按自己喜欢的方式来做更重要。

Seibel：你有没有做过结对编程？

Fitzpatrick：我觉得那很有趣。结对编程对很多事都很有益处。有时你只是需要思考，想要一个人独处。我并不总是采用结对编程，但它的确很有趣。

我发起过很多项目。如果不完成它们会让我有负罪感，但我确实切换得过于频繁，精力过于分散了。这是我需要结对编程的原因——它强迫我坐定三小时，甚至是两小时或一个小时和别人一起做一件事，他们也会让我不那么无聊。如果我碰巧遇到一个无聊的补丁，他们会说“来吧，我们要搞定它”，然后我们就做成了。

我喜欢一个人工作，但在工作时会到处跑。在飞机上我总带块备用笔记本电池，笔记本上有整个开发环境和本地Web服务器，我能在浏览器里进行测试。但我还是会打开一个新标签页，键入reddit或lwn——我常上的网站。自动补全后按下回车，然后出现错误信息。一分钟里能重复好几次。见鬼！我在工作时就这么做吗？我甚至都没意识到自己经常看网页？这太吓人了。我有个朋友，他有些iptable的规则，当在一天里的特定几个小时去访问特定IP时，会重定向到一个页面，上面写着“你应该在工作”。我没有这个东西，但我想也许需要做一个类似的玩意儿。

Seibel：你是怎么看待代码所有权的？是独立拥有代码重要，还是团队分享

代码更好呢？

Fitzpatrick：我不认为代码应该被谁拥有。我不认为谁会真的有那种想法。Google里的方式是这样的，有一棵巨大的代码树，一个根目录，一个适用于所有代码的统一构建系统。每个人都能动手修改任意的代码。但我们有代码审查，每个目录都有拥有者，考虑到可能有人会退出或休假，至少会有两个拥有者。

要签入你的代码需要符合三个条件：有人给你做代码审查，并予以肯定。在语言方面得到认可——起码你得证明你知道这门语言的风格，这叫“可读性”。此外，你还需要得到那个目录的拥有者的批准。如果你已经是拥有者了，又能保证语言的可读性，那你只需要找个人说“耶，它看上去很棒。”这是个很好的体制，因为这里最少有两个，最多有二十到三十个拥有者。只要你在一个代码库上做了一段时间，就会有人把你加为拥有者。我认为这是个很好的体制。

Seibel：让我们再往回倒一点，你是怎么开始LiveJournal的？

Fitzpatrick：其实就是和朋友们闹着玩，我所要的和我们所想的都很有趣。LiveJournal上的评论就源于一个真实的玩笑。我在上课前登录了一下

LiveJournal，当时我们刚加入好友页面，看到我朋友写的东西，真的写得很蠢，我想取笑一下他。“噢，但我不能回复。”然后我去上课了，整堂课都在想：“我该怎么添加回复的系统呢？”我考虑了现有的Schema以及如何呈现的问题。课间有两小时的休息时间，我用那段时间添加了评论系统，回复了些自作聪明又有点挖苦他的话，然后去上其他课了。第二节课回来后，他对我说：“我们现在居然可以发表评论了？”

LiveJournal上的一切都很搞笑。整个安全体系，例如好友帖和私人帖，都是因为一个朋友写了他去出席一个派对，第二天酒醒了发现自己掉在一条沟里。他父母读到了，说：“什么？你居然喝酒？”他对我说：“Brad，我们需要找个办法把这些内容锁起来。”我告诉他：“我这就去做！”我们已经有好友了，所以只要让一些帖子只能被好友看到，别和父母成为好友就行了。

Seibel：LiveJournal的早期，你的生活看起来就是无休止的熬夜，睡得很晚，长时间工作。编程真的必须这么干吗？

Fitzpatrick：这对我来说是压力最小的时候。白天总有各种事情会找上门来，比如要吃饭，要上课，或者要接电话，总会被打断。我没办法放松下来。如果在会议前工作两小时，那这两小时比起没那会议或者一早开会要低产得

多。知道没事会来烦我，我会放松得多。

我会觉得晚上是我的时间，我正在偷时间，因为其他人都睡觉去了。没有噪音，没有干扰，我能做各种事情。我有时也会熬到很晚，忙着不同的事情，通常周末会这样。只是那样做会让我花上几天时间来调整睡眠。这么做主要是因为我还在大学里，有些项目要做，又要做LiveJournal。唯一可以用的时间就只有晚上了，而且服务器的维护也只能放在晚上。到了夏天，还有什么原因不这么做呢？早上不用起很早，不用去上课或做别的事情，所以晚上可以做得晚一些。

Seibel：那工作的长度和强度如何？你肯定一周工作80小时、100小时甚至120小时。这是必须的吗？在什么环境下这才是必不可少的，在什么时候这只是我们体现的一种男子气概？

Fitzpatrick：在我看来，我不确定这是必需的还是男子气概。我就是觉得很有趣，这是我想要做的。有时会遇到麻烦，但就算没问题，我也会这么做，因为我正在做那些我希望实现的新特性。

Seibel：你有没有遇到过不得不估算事情要花多少时间的情况呢？

Fitzpatrick：刚迁移到Six Apart的时候遇到过一次。那是三年半以前，我想

那是我第一次有这样的经历。我们已经开始做迁移了，一位客户问：“你们能移动这个数据吗？”这么做要为代码添加支持，做测试，还要发布。这让我很惶恐。也许现在我还会为此类问题感到惶恐，因为我总是忘记某些因素，比如工作时常被打断，还有我从来没摆脱要维护一堆项目的局面。

我想现在我越做越好了，也幸好他们不再经常提这种要求了。现在当我真的遇到某件事有最后期限时，我会说“耶！有个最后期限！”我变得很兴奋，肾上腺素激增，开始干活，把那件事做完。在Google没有什么真的最后期限，我们总是说：“你觉得这个东西什么时候可以发布？它看上去怎么样？”很少有真的最后期限。大多数时候，我们都认为它最好在某个时候发布，然后大家都很努力。但如果真的没有完成，你只是会让那些希望看到它准时上线的人感到失望，仅此而已。我所做的多数事情都很顺利，我总是说“船到桥头自然直”。

Seibel：你以前为LiveJournal招聘程序员时，你还要管理他们吗？

Fitzpatrick：我更喜欢假设他们都不需要管理，能像我一样自我驱动。HR的经验中有这么一条——一些人只会做让他们做的事，没有那种追求极致的热情。他们会说“做好了，接下来干什么？”他们甚至不告诉你，自己就上

网玩去了。我有一些痛苦的经历，但一两年后，我发现原来人与人是不同的。

还有些人是纯粹主义者，他们总是抽象、抽象、再抽象，动作很慢，而且十分崇尚这种风格。他们会说“程序设计是门艺术”。而我则回答：“你的代码跑不起来，效率太低，而且它和那些与之交互的代码格格不入。”

Seibel：你找到让这种人充分发挥的方法了吗？

Fitzpatrick：有一个人，我试过各种方法。我想他大概比我大10岁。我不知道究竟大几岁，因为我从没问过，我害怕问那些法定的雇佣问题。但我有种感觉，他不想为年轻人工作，我当时22岁，他始终没想通。这是唯一一个我让他走人的人。

其他人，我最终都找到了激励他们的方法。有个家伙很擅长做原型。他写了sysadmin的Perl脚本。他把所有的东西整到一起，写shell脚本，虽然他的Perl和C写得真的很烂，不过他能让东西跑起来。随后我们惊奇万分：“天哪，这些你都研究过了啊，是你让它们互相协作的？”

我们当时正在搭建LiveJournal上的语音系统，有了它你就能录音并发送给LiveJournal，其中涉及了很多要迁移的部分。我觉得那就和地狱一样，而他却很喜欢。他找出了所有的部分，让它们都运行了起来。然后我们重写了

这些部分，我发现这就是和他工作的方式。他找出所有的接口，而我们修正它们。一旦我找准了他的定位，我们就能和睦相处了。

Seibel：你以前为自己的公司招聘，现在我想你也参与Google的招聘工作，你怎么识别好的程序员呢？

Fitzpatrick：我总是找这种类型的人，他们会做很多别人没要求他做的事，不仅是学校的项目或者前雇主要求他做的。他们对某些事情充满激情，有额外的项目。我问清楚他们如何维护它，对它有多上心，还是就草草调整随后放弃它们。

Seibel：你有什么特别喜欢问的面试问题吗？

Fitzpatrick：有一个问题我问了很多次，因为那是我在AP程序设计考试上做过的，题目是给定两个任意长度的十进制数字符串，将它们相乘。这个问题有很多种不同的答案。如果应聘者数学很好（不像我），他们能找出很多种聪明高效的方法。最坏的情况，搞一个类，不停地做加法。

我一开始就告诉他们："不用紧张，你不用给出很高效的解答，能做出来就可以了。"有些人会感到紧张，不知道从何下手。那可是个不好的信号。

最坏的情况下，实现一个小学里才做的算法。

我真的在小学里写过程序做长除法和乘法，还展示整个结果，包括所有的步骤，要删掉的地方。以后我们再拿到这种题目时，例如一页10道题之类的，我就会把它输进电脑，再把结果草草地写下来。化学里找电子轨道时我也做过同样的事。我发现通过写程序去作弊也能得到学习，因为你需要深入学习之后才能写出那个程序。

Seibel：你认为这对别人有帮助吗？除了教孩子长除法，我们是不是应该再教他们如何编程，告诉他们“OK，现在你们的任务是写个乘法实现长除法过程”？在他们动手写程序前，他们会明白除法是怎么回事，还是说这种方法只对那些天生对此有偏好的才管用？

Fitzpatrick：这对我很管用。很多时候，别人教你东西时，你会说：“嗯，当然，我明白了。”你其实是在欺骗自己，可是一旦真的要动手去做，就需要了解所有的东西，这迫使你去学习。但我不清楚这是否对所有人都有用。

Seibel：Google和微软都以面试出难题而闻名。

Fitzpatrick：我觉得这些都是被禁止的，或者不被推荐的。可能有些人还是在用，但总的来说，我们不鼓励这样做。

Seibel：他们在面试你时问了什么？

Fitzpatrick：有一个问题是假设你有一组电脑，连在一台交换机上，开启整个机架，请给出一个算法，让机架上的每台机器都能知道其他机器的情况，是开着的还是关着的。基本上就是个存在性问题，条件就这些了。他们简单描述了一下Ethernet：你可以给所有人发广播，或者发给特定的MAC地址。我只是遍历了各种不同策略，降低带宽，减少发现某台机器宕机的延时。这道题目比较有趣。

Seibel：你遇到的最大的bug是什么？

Fitzpatrick：我尽力不去回想它们。我讨厌那些离自己的假设差距很大的东西。记得有一天（显然这不是最糟糕的情况），我花了90分钟调试一个问题，因为我把输出写到了一个文件，然后从一个同名文件中去读，但路径里却少了一项。我不停地返回这个巨大的MapReduce，检查输出，把它放进GDB，做单步调试。“到底怎么回事？什么都没变！”最后我看了下路径，天啊！我不知道为什么我在这个问题上花了90分钟，我当时一定是鬼迷心窍了，就是没想到回过头来检查一下命令行是不是正确。

还有很多类似的情况。我们总会在代码里用些Perl的语法糖，例如没有

文法作用域的$_。但如果你在排序中用$_，你就会把别人的东西搞得乱七八糟。我们永远都摆脱不了这个bug，总是有东西弄错。最后我们把它解决了，我审查了所有的代码，定了条新策略：永远都不许这么干。

Seibel：你们的调试工具是什么？调试器？打印语句？还是别的东西？

Fitzpatrick：如果环境允许，我会选打印语句；要是所处的环境有好的调试器，那就用调试器。Google把GDB维护得很好，当需要时，它真的是无可替代。我尽量不去用它，虽然我对GDB不是很熟，但通常只要看看，我就能找出问题所在。如果一定要深入其中，我有自己的办法。我喜欢strace，没它我都不知道该怎么活。当我不知道某个程序在做什么，或者我自己的程序在做什么，就用strace来跑一下，看看究竟发生了什么。要是只能选一个工具，我想就是strace了。还有Valgrind、Callgrind等工具都很好。

好多次有奇怪的问题发生时，我会说："OK，那个功能太大了，我们把它拆开，独立进行单元测试，看看我的假设哪里错了，而不是没头地打印。"

然后，在重构的过程中，我会更多地去考虑代码的细节，慢慢地就清楚了。这时，我可以回到那个庞大的、丑陋的函数，修复它，但我其实已经重构到一半了，我也可以为下一个维护者继续进行简化。

Seibel：你在代码里是怎么用不变式的？有些人把它扔在单独的断言中，有些人每步都会放上不变式，以便能证明程序中的形式化属性，在这两种极端情况之间还有很多做法。

Fitzpatrick：我尽量不走形式化路线。我的基本规则就是如果它来自终端用户，那就不算运行时崩溃。但如果是来自我的代码，那我就要尽力让它早崩溃，尽可能早地失败。

我尽量以前置条件的方式来思考问题，在构造器和函数的开始时进行检查。如果可能的话，调试检查，以此保证编译通过。有很多思维流派，也没人教过我怎么做才算合适。有些语言将所有这些内容都作为语言的形式化部分，基本上我用过的所有语言都是如此，选哪种就看你自己的了。

Seibel：你曾经写到过优化是编程中你比较喜欢的一件事，现在还是这样吗？

Fitzpatrick：因为优化不是必需的，所以它才有趣。如果你正在做优化，那么没什么比让东西跑起来更重要的了。你要么为了省钱而优化，要么就是因为它像Perl Golf比赛[①]——我能让它精简多少，或者快多少。我们会找出LiveJournal的热点，发起竞赛。“ 这里有些代码。这是性能基准。让它更快

① 一个Perl的比赛，给定一个问题，寻找解决该问题的最短代码，参见http://perlgolf.source-

一些。”我给出负载均衡器的解析头。我们都埋头写疯狂的正则表达式，它们没有回溯，尽力用最有效的捕获组去捕获内容。大家都在互相竞争，让速度变得更快。第二天，一个哥们儿跑了过来，把所有的东西用C++的XS模块写了一遍，他说：“我赢了。”

Seibel：现在看那样做的负面影响是什么？

Fitzpatrick：程序员的时间更值钱，不该浪费在这里，究竟什么情况下这是对的呢？机器少的时候这是对的。一旦你有了很多机器，比起这个程序要部署的机器数量，程序员的时间一下子就不那么值钱了。因此，用C写代码，剖析出问题所在，修正编译器，再付钱找人用GCC让它编译得更快些。

Seibel：就算是Google也是使用C++而非汇编，是不是可以理解为有些地方并不值得去挖掘性能的极致。还是说好的C++编译器生成的代码比那些异想天开的稀有汇编程序员生成的要好？

Fitzpatrick：我们仍旧有些用汇编写的东西，但是非常非常地少。我们会对很多东西进行剖析，仔细确定是否需要将它从Perl改写到C，然后再由C改写为汇编。就算全是x86体系，还是存在很多x86的变体。你确定自己真的愿意

forge.net/。

为每种x86的变体写一段汇编吗？这个用SSE 2，那个用SSE 3.1。还是让编译器来处理这些吧。

Seibel：你还是个孩子的时候就从编程手册上学习编程。有没有什么书是你强烈推荐给新程序员的，或者是你觉得每个人都应该读一下的？

Fitzpatrick：我还在写Perl的时候——甚至是对那些很了解Perl的人——我都会推荐MJD（Mark Jason Dominus）的《高阶Perl》（*Higher-Order Perl*）。这本书真的很有意思，它先从简单的东西入手，你会觉得“嗯，我知道什么是闭包”，然后再慢慢被带入云里雾里。看完这本书后，你就会有极深的印象。尽管我知道所有这些内容，读这本书还是彻底改变了我的想法。我把它推荐给了很多朋友，都让他们对Perl有了新的认识。总的来说，每本书都会给人带来不同的想法。我想这是我能想到的最近的一本书了。

Seibel：我看你在那里放了本《计算机程序设计艺术》，它看上去还没翻旧，你看了多少了？

Fitzpatrick：哦，我拿到那本书还不到五年，也许正好五年。我大致翻了下，有兴趣了就读一读。看这本书之前，我已经从C.S.课中学到了书中的很多内容了。因此，也许这本书放在以前会更有价值，但我在联上因特网前并不知

道这本书。

Seibel：你觉得程序员需要了解多少数学知识？读Knuth的书并要真正理解它，需要很好的数学功底，但作为程序员真的需要这些吗？

Fitzpatrick：你并不需要这么多数学知识。对大多数程序员而言，在日常工作中统计知识更重要一些。如果你是做图形相关的工作，数学会更重要些，但对于从事Java企业级应用或Web开发的人来说就不是了。逻辑和统计知识会更有帮助。

Seibel：你显然还很享受编程带来的乐趣。但我读了你在大学时LiveJournal上写的一些内容，看起来你承受了不少的压力而且讨厌电脑。

Fitzpatrick：哦，我其实一直讨厌电脑。在这么长的一段时间里，我并不认为我们真的取得了多少进步。电脑看上去比以前更慢、更容易崩溃而且问题更多。不过我是个乐天派，我还是相信它们会变得更好。看起来我十年前使用电脑的经历比现在要好得多，我十年前的电脑要更快些，工作得也更好。硬件变快的同时，软件却变慢了，而且漏洞百出。

Seibel：为什么你会这么想？

Fitzpatrick：我不知道。是门槛变低了？还是电脑太快了，让你不用再讲效率了？亦或是你无须知道自己在做什么了？我不清楚。也许是上述多种情况，也可能是存在太多的抽象让你不知道下面究竟发生了什么，因为电脑实在是太快了，它掩盖了人的愚蠢。

Seibel：按照现有电脑的速度，也许有些东西还未达到它应有的速度。但十年前，作为用户，人们根本没办法做到今天能用Google做到的事。

Fitzpatrick：是的。所以有人会写高效的代码并加以利用。我不玩游戏，但偶尔会看人玩，我会说："天哪，这怎么可能？"它征服了我。显然，有人做得很好。

我猜我是对我台式机的状态失望透了。在后端有那么多有趣的东西，但当我在用我的电脑时却对它越来越失望。我的Mac不应该总出现那个海滨球[①]。

Seibel：你有没有兴趣写更好的桌面软件？

Fitzpatrick：问题是没人会去用桌面软件。你想有人用你写的东西，就该做成Web应用。如果有一天我把笔记本电脑弄丢了，有人可能会问："噢，上

① Mac OS等待时的鼠标图案。

帝啊，你有没有掉什么资料？”其实那上面根本没有我的文件，它只是个因特网终端，一个加密磁盘，我并不担心我的密码、Cookie或类似的东西。我不认为人们会去下载程序。

Seibel：你的动力更多是源于拥有用户，还是出于编程的乐趣？

Fitzpatrick：确实有些东西是为我写的，而且只为我一个人，我是唯一的用户，如果我有补丁或其他事要做时，对它的关注就会少些。但很多时候我愿意和别人合作。有用户是获得贡献者的关键之一。有更多的用户就能找到更多的问题和更多的用例。和他人合作起来也更有意思，特别是在做开源的东西时。

每当看到有人写信说“嘿，我们正在什么什么上用你的软件”时感觉都会很棒，这太酷了。当我看到使用memcached或负载均衡器或别的我写的东西的网站数量那么多时，我会说：“啊，那太酷了。”还有那些成人站点也告诉我他们正在使用我的文件系统。好吧，这也说明了一些东西，我在帮助建设成人站点，呵呵。在Craigslist上，每个请求通过的Web服务器可以算是memcached的一个前端。这太酷了！

Seibel：你认为程序员过分热衷于新鲜事物吗？新的语言，新的工具，任何

新的东西？

Fitzpatrick：可能是这样吧。我不知道要是没什么可期待的是不是件让人失望的事，比如说我想要门新语言，让它实现我们要的所有功能。用户也这么想，他们总想要更高的版本，就算它可能会更糟。

我不知道从统计学角度来看程序员是否通常都有别于常人。新的东西必须比旧的好。尽管人们是这么希望的，但理想和现实往往有些距离。

我记得以前和我的牙医聊过一阵子，她总是不断地说着五年来牙科方面的进步，她对那些进步感到很激动。

Seibel：想要成为一个当代程序员，就要找到正确的东西——你要用的东西，并理解透彻。这方面你是怎么做的？

Fitzpatrick：CPAN[①]上有各种东西，那里光ID3解析器就有14个，选一个就好了。

Seibel：从某方面来说，这也是当代程序员所面临的一个问题——有14个可供选择，你该选哪个呢？

Fitzpatrick：用Google搜索，看看哪个排名高，哪个是人们更偏爱的。再去

认识些人。我深深扎入开源社区，一切都是从参加所有这些会议开始的，因为这样我能认识不少人，知道谁值得尊敬，谁比较酷。

然后再去看他们的代码。我记得一个家伙，他棒极了，既有趣，又友好，还很体贴，真的很关心自己的代码。当有人抱怨他的代码时，他会很激动。我会选这个软件，因为如果我发现任何问题，我知道他会积极地修正它们。那些性情乖戾的家伙则相反，他们也许能写出很好的代码，但脾气很坏，当你遇到问题或错误时不太好交流。因此要选一个你信任或敬佩的维护者。

Seibel：有没有什么讨巧的方法能快速找到满足你需要的东西呢？

Fitzpatrick：刚开始时，我不会直接把它放进代码里，我会先写个测试程序试用几个我知道会用到的函数，确认它们能用。或者为那个库写一个单元测试，用将来要用的数据测一下。很多库甚至都没有自己的测试。即使有测试，也许你在读过文档后也无法确信它真的做了它承诺的事，或者是文档对它的行为描述得不够清楚。所以我亲自为那些我关心的东西写测试。反正都要通过写些什么来了解库的用法，因此我的第一个Hello World程序也可能是一个单元测试。

① Comprehensive Perl Archive Network，http://www.cpan.org/。

Seibel：谈谈你正在使用的工具吧。你还是Emacs的用户，对吗？

Fitzpatrick：我依然是Emacs的用户。我希望自己能把Emacs用得更好些。我知道所有的按键，但很少做自定义。我会拿别人的自定义配置。当发现自己对什么东西不满时，我会说“ 我要写些Elisp来做键绑定”。但过一会儿就把这事给忘了。

Steve Yegge正在做的项目基本能用JavaScript来代替Elisp。我会等他的，这样就不用再学一门语言了。我就用JavaScript来做这事。我觉得JavaScript这门语言挺好的 ,有问题的是浏览器。在Google我用JavaScript写了不少东西，然后嵌入Java和C++中。我觉得JavaScript是一门很好的嵌入语言。

Seibel：还有什么工具是你平时一直在用却十分痛恨的？除了你的台式机。

Fitzpatrick：是的，我讨厌整个台式机。我的台式机上有很多东西，那些浏览器总是会挂住、崩溃掉，还会占用大量内存。整个操作系统也会挂住。我的同事看到我用Emacs时，就试着说服我，Eclipse和IntelliJ能自动替我完成这些事。我每隔6个月会选择其中一个试用一下，Eclipse或IntelliJ。那该死的东西就停在那里，消耗内存，也许在我敲击键盘的时候会崩溃，不让我打字。来吧，突出显示后台语法，或在另一个线程中进行编译。为什么一定要

打断我来做这些呢？OK，我过6个月再来试试吧。我很高兴不用被迫使用这些工具。我真该把Emacs再用用好。

我的学习曲线是这样的，学东西很快，直到我学好它而且相当高产为止。随后我会维持在一个80%～90%的稳定水平，这时我效率很高，不用去查阅资料，我很开心。在那之后情况还会慢慢变得更好。只有到我觉得过度安稳了之后，我才会想："我要去仔细看看这门语言的文档（man页面），了解所有细枝末节的内容。"

Seibel：这么做明智吗？有太多东西要学，你可以花一辈子学习怎么使用编辑器，那你又能写多少软件呢？

Fitzpatrick：是的，但我发现——至少对编辑器而言——花掉的时间总是会有回报的。无论我学什么，总会有回报的，也许是一两周内就会有。当我在bin目录里写些很傻的shell脚本、Perl脚本，或者别的东西做自动化，它一定会有回报的。

Seibel：那么说来你从没陷入无止境的工具开发中吗？

Fitzpatrick：没有。我总是出于某种目的去开发工具的。我确实认识一些人，他们一直在开发自己的工具却从没拿出一个成品。我会朝着这个方向再更进

一步，而且很安全。

Seibel：你觉得对程序员来说什么才是最重要的技能？

Fitzpatrick：像科学家那样思考，一次改变一样东西。有耐心，试着去了解问题的本质。尤其是在调试或者设计那些不太正常的东西时更应该这样。我看到过年轻程序员在那里抱怨“ 哦，见鬼，这个东西运行不了”，然后就把它彻底重写了。其实应该停下看看究竟发生了什么。要学会增量地开发，这样每一步你都能进行验证。

Seibel：作为一个程序员，你会不会特意做点什么来提升自己的技能呢？

Fitzpatrick：有时我会不走寻常路，虽然知道会花费更多的时间，但还是用一门我不太愿意用的语言来写东西，因为我知道最后这对我是有好处的。比如我刚到Google时，要写什么东西，我都会用Perl。然后我觉得“ 啊，不对，我应该用Python来写的”。现在我会用Python来写很多东西，这不再是我的弱项了，我甚至很少需要去查资料。Perlbal本来是用C#写的，我就是想学一学C#。

Seibel：除了编程本身，还有什么技能是那些准程序员应该学习的？

Fitzpatrick：沟通技巧，不过我不确定那是个能练习提高的东西。多和人在邮件列表里交流。书面沟通风格的养成需要很长时间。但这是一辈子的事，不是吗？有一项关于高中毕业生中成功人士的研究，是聪明孩子好，还是会交际的孩子好呢？结果证明，那些会交际的孩子一生都能赚钱，而不是那些成绩好的。我觉得这个结果很有意思。

Seibel：看起来现在的情况和以前有些不一样了。以前的程序员更像是躲在办公室里的侏儒，而现在到处都是邮件列表，大家都在谈论协作。

Fitzpatrick：在我工作的地方，无论是开源社区还是公司里，大家都相互依赖。我们的动机是 因为我知道过两周你需要这段代码或者我需要你的代码，所以我动手来写这个代码。”人与人就是这么相处的。

Seibel：人们总说最好的程序员和最差的程序员在生产力方面有着天壤之别。根据你的经验来看是这样的吗？

Fitzpatrick：是的，基本在所有的领域都是如此，这取决于你有多少经验。据我所知，通常的情况并不是两个人花同样的时间就能有同样的编程产出，时间差可能会有十倍之多。如果你不能时刻保持良好的状态，那你就可能感到疲倦，然后出局。

我觉得有些人只是在工作，而不是享受编程带来的乐趣，这没什么问题。但把他们和那些核心程序员做比较就不行了。当一个人花的时间多十倍，不停地考虑怎么写好这个程序，另一个人只是为了工作而工作，那两者生产力上的差距又岂止十倍呢？

Seibel：你前面提到用科学家的方法来进行调试。你觉得自己是一个科学家、工程师、艺术家还是工匠呢？

Fitzpatrick：科学家或者工程师吧，也许我更像个工程师，科学家排第二，但你必须要懂得科学的方法，一次改变一样东西，如何诊断问题。工程师是指设计方面的。我也有些朋友自称为艺术家或工匠的，但我从来没这么想过。

Seibel：另一方面，软件领域里也有很多工程方面的问题。有这么一个笑话：如果人们用造软件的方法来盖摩天大楼，那第一只啄木鸟就能毁掉文明世界。你认为软件生产是一门非常成熟的工程学科吗？

Fitzpatrick：不，我觉得没到那个地步。你不需要执照就能写代码。虽然我不想要那么多规矩，但如果能保证那些留下XSS隐患的PHP程序员不是写空中交通管制系统的人就太好了。我很希望对这些人能有个官方的界定。

我有个朋友是建筑工程师，他一直去学校充电，考各种工程学相关的认

证。你站在桥上，想起造桥的人花了毕生的精力学这东西，参加了无数考试，一直在学习，这种感觉很好。

Seibel：那你能给程序员什么测试，让你相信他们能写出正常工作的软件呢？

Fitzpatrick：我不知道。这有点儿吓人。

Seibel：就算没有执照，你觉得程序员对社会有什么道德上的责任吗？程序员是一个职业，是职业就有自己的职业操守。

Fitzpatrick：你不能谋杀他人，比如你写的飞行控制软件出了问题，但那只是极少数的情况。我一般会要求大家写的信用卡表单能一致，比如能让我在其中插入空格或连字符，电脑很合适处理这种情况。又比如不要告诉我如何格式化我的数字。但这里没有什么道德标准，有的只是愚蠢的行为。

Seibel：你今年28岁了。有没有担心过编程是年轻人的游戏，当自己老了就会略逊一筹？

Fitzpatrick：没有。最坏的情况就是我停下来，自己找点乐子。我不觉得自己现在是在和谁竞争，我也不太关心别人是不是比我更好，因为我觉得已经有无数人比我好了。我发现我们总是处于中间位置，而我也很乐意保持在这

个位置上。

Seibel：这么说来你是因为觉得有趣才编程的，即使你不再工作了？

Fitzpatrick：是的。我还是会继续做些小玩意儿。我的手机上有个无聊的棋类游戏，我对它有点厌烦了。我不太会正儿八经地去干什么，所以就写了个求解程序。其中试用了一些动态编程，处理了不同的棋盘大小，还有很多随机棋盘，针对不同的棋盘大小还给出了解决棋局所需步数的直方图。我把它发给了作者，因为游戏里的标准估算真的很糟糕。基本上你想在游戏中玩下去，必须要超过平均水准。邮件列表里的每个人都发现游戏玩到后面变得容易了，不知道这个标准他是怎么弄出来的。于是我把每种棋盘大小的直方图发给了他，我想他在新版本里会有所调整吧。这是件很有意思的事情，我想我可以退休后整天做这些事。

（编辑：谢灵芝）