超越平凡

一家新兴企业的故事

1995 年的夏天，我和我的朋友 Robert Morris 创立了一家名为 Viaweb 的公司。我们的计划是开发一个软件，让我们的终端用户能够创建网上商店。这个软件的创新之处就在于，它自始至终都是运行在我们的服务器上的，而且拥有一个像传统网页一样的用户界面。

在当时，有很多人都有这样的想法。但据我所知，Viaweb 是第一个基于网络的（Web-based）应用程序。这对于我们来说是一个非常有创意的思路，我们公司的名字也正是来自于此——“Viaweb”，因为我们的软件是经由网络（via the web）来工作的，而不是运行在你的桌面电脑上。

这个软件的另一个与众不同之处便是，它主要是由一种名为 Lisp 的语言编写而成的。在当时，这可算是最初的几个用 Lisp 写成的大型终端用户应用之一。而时至如今，用 Lisp 写成的软件已经充斥在大学和研究室中了。相较于那些使用不那么强大的语言进行开发的竞争对手来说，Lisp 着实带给了我们极大的优势。

秘密武器

Eric Raymond 曾经写过一篇名为**怎样成为一名黑客**的文章。在那篇文章中，他向那些希望成为一名黑客的人们提出了一些学习的建议。他建议从 Python 和 Java 学起，因为这些语言学习起来比较容易。而对自己要求稍微严格一些的黑客，应当去学习 C 和 Perl。这些语言在对 Unix 系统的攻击，以及系统管理和 CGI 脚本等方面有重要的用处。而一个真正的黑客，应当考虑去学习 Lisp：

> **Lisp 值得你花时间去学习。当你最终弄明白它的时候，你会从中获得深刻的启示。这种经验会让你在今后的时间中成为一名更加优秀的程序员，即使你可能并不会真正频繁地去使用 Lisp 语言本身。**

你或许会在一些对学习拉丁语的评论中听到过类似的论调。拉丁语不会为你提供一个工作岗位，除非你是一位研究古典文学的教授。但是它确实会启发你的思想，让你在使用其他语言，如英语时，更加游刃有余。

但是，且慢。这个比喻并没有揭示出问题的全部。拉丁语不能给你带来工作机会，原因是现在已经没人说这门语言了。如果你用拉丁文写作，没有人会看得懂。但是 Lisp 不同，它是一门程序设计语言。如果你使用 Lisp，计算机会知道你在说什么。

因此，倘若确如 Eric Raymond 所言，Lisp 真的能让你成为一个更优秀的程序员的话，那为什么你不会去使用它呢？试想，如果一个画家拥有了一把神奇笔刷，这个笔刷能让他创作出更加优秀的画作，那么他必然会用这把笔刷去创作所有他的画作，不是么？我并没有拿 Eric Raymond 先生开玩笑。总的来说，他的建议是中肯的。他有关 Lisp 的评价确实代表着大家的传统看法。但是这种看法中确实地存在着一个矛盾：Lisp 会让你变得更加优秀，但是你不会选择去使用它。

这是为什么呢？毕竟程序设计语言仅仅是工具而已。倘若 Lisp 能让你写出更好的程序，你就应该使用它。如果它不能，自然不会有人去用它。

这并不只是一个理论上的问题。软件开发领域的商业竞争非常激烈而且很容易形成垄断。在其他方面都相同的条件下，一家能够在更短时间内开发出更好软件的公司会把它的竞争对手打得落花流水。当你在经营一家初创公司的时候，这种感受就更加真切而现实。初创公司经常两极分化。你要么变得富有，要么输个精光。经营一家初创公司，如果你在错误的技术上下注，你的竞争对手便会彻底地摧毁你。

我和 Robert 都非常熟悉 Lisp。我们看不到任何理由去违背我们的直觉——使用 Lisp。我们知道其他所有人都在用着 C++或是 Perl，同时我们也知道这并不意味着什么。如果你选择了那些技术手段，就意味着你的程序将在 Windows 环境下运行。当你在选择技术时，你不应考虑别人怎么做，要考虑的仅仅是哪种技术最好用。

在初创公司时这更是一条真理。对于一家大公司，你可以选择和其他大公司步调一致。但是一家初创公司绝不能随波逐流。我认为很多人都没有意识到这一点，即使他们正在一家新兴企业中就职。

平均下来，大公司每年能增长 10%。如果你正经营着一家大公司，并且你所做的和大公司的平均水平做的事情完全一样，那么你可以预期你做的将和平均水平一样好，也就是说，10%的年增长率。

这个道理在初创公司中当然也是成立的。如果你做的事情和平均水平一样，那么你的收益也就和平均水平一样。问题是业绩平平就意味着被淘汰出局。初创公司的存活率低于 15%，所以如果你正在经营一家初创公司，你最好能够做一些与众不同的事情。否则的话，你的麻烦就大了。

回到 1995 年，那时的我们认识到了一些我们竞争对手所不了解的事情。这些事情即使放在现在也并非人尽皆知：当开发的软件只运行在自己的服务器上时，你可以选择任何一种语言来开发。如果是开发桌面应用，那么你很可能会偏向和操作系统使用相同的语言。十年前，“写程序”等同于“用 C 写程序”。但对于基于网络的软件，特别是当你同时拥有语言和操作系统的源代码时，你可以使用任何一种你喜欢的语言来开发。

当然，这种新的自由度也是一把双刃剑。既然你可以使用任何一种语言，那么你就有必要仔细斟酌选择哪一个。那些对此假装视而不见的公司会面临巨大的风险，因为它们的竞争对手并非对此视而不见。

如果让你来选择，你会选择哪一种语言？我们的选择是 Lisp。一方面，我们需要快速开发的能力，这对于市场是非常重要的。我们是白手起家，从无到有，能比竞争者更快地提供更多新功能将会是一大优势。我们知道，Lisp 非常适合于软件的快速开发，而基于服务器的应用程序则能将效果进一步放大：因为你可以在软件开发完成的同时就发布它。

如果其他对手不打算用 Lisp，那是再好不过的了。这可能会给我们带来技术层面的加分，但同时我们还需要在其他各个方面寻求帮助。在创立 Viaweb 的时候，我和 Robert 两人都对经商一无所知。我们不知道有关市场、雇佣、营收和招徕顾客的各种知识。我们两人甚至都没有过一份真正的工作。我们所擅长的就是开发软件。我们寄希望于此，因此一切软件开发中的优势资源，我们都要利用起来。

所以你可以认为，我们使用 Lisp 的本身也是一种尝试。按照我们的设想，我们将能比竞争对手更快地提供功能，完成他们力所不能及的事。同时，Lisp 是一门非常高级的语言，因此我们不需要一个大的开发团队，这将大大节省我们的开支。如果情况真是这样，我们就能用更少的钱提供更好的产品，保证盈利水平，击败竞争对手，将他们依数赶出这个市场。至少，这是我们的美好愿景。

那么这场试验的结果如何呢？或许有些出乎意料，情况确实如此。我们后来的确有许多

竞争对手，大约有 20 – 30 个之多。但没有一家公司的产品能够和我们的软件相匹敌。我们做出的软件是一个所见即所得的网上商店创建器。它运行在我们的服务器上，但是用起来就像一款桌面应用一样，而我们的竞争对手使用的则是 CGI 脚本。同时，我们在发布特性方面也大幅领先于对手。绝望的他们有时会试图引进一些我们软件所没有的特性。但是由于使用了 Lisp，我们的开发周期非常紧凑，可以在一两天时间就把竞争对手刚刚发布的新特性模仿出来。这样当各路记者来采访我们的时候，我们早已有了同样的新特性了。

在竞争对手看来，我们一定是有了某种秘密武器——解密他们加密过的数据包或是其他的什么。事实上，我们的确有一件秘密武器，但是没有他们想象的那样神秘。没有人向我们泄露任何关于竞争对手产品的任何信息。我们只不过是在以任何人都无法想象的飞快速度编写软件而已。

当我九岁那年，我获得了一本 Frederick Forsyth 编写的 **THE DAY OF THE JACKAL**。书中的主角是一名被雇来刺杀法国总统的刺客。有一次，刺客需要穿越警察队伍，到一间公寓里监视总统的行踪。他假扮成一名拄着拐杖的老人，跟在警察旁边一起走，瞒天过海，逃过了警察的视线。

我们的秘密武器与此类似。我们用一个看起来十分古怪的、多用于人工智能领域的语言来开发我们的软件。这种语言中充斥着令人迷惑的圆括号。多少年来，我都反感 Lisp 代码的这种外观，但现在它却成了我们最有力的伙伴。在商业界，还有什么能够比掌握一手独门绝技更加有价值的呢。商场如战场，奇兵才能制胜。

我不得不承认，我从未在公共场合谈论过任何有关 Lisp 和我们开发工作之间的关系。我们从未对媒体说起过此事。如果你在我们的网站中搜索“Lisp”，你只会找到我简历中提及的两本书的书名。这并非巧合。一家初创公司应该对其竞争对手尽可能保密。如果他们不知道我们用于开发程序的语言，或者根本不在意的话，我就希望维持现状。

我们的产品究竟如何，顾客最有发言权。他们不关心 Viaweb 到底是用什么语言写的，他们只关心它是否管用、好用。他们用 Viaweb 能在几分钟内就创建好一个漂亮的网上商店。就这样口口相传，我们的用户便越来越多。到了 1996 年年底，我们拥有 500 位用户。而当六个月后当我们被 Yahoo 收购时，这一数字便已经增长到了 1070。如今的 Viaweb 已经变身成为了 Yahoo Store 统治了整个市场。它是 Yahoo 最盈利的一部分之一。用他构建的网上商店则成了 Yahoo Shopping 的基石。我在 1999 年离开了 Yahoo 公司，不知道最后用户的确切数字。但是我最后一次听到的是 14000。

人们有时会问我，Yahoo Store 是否还在用 Lisp。我的回答是：是的。所有的 Lisp 代码都还在工作着。Yahoo 的服务器端的软件是由所有 Eric Raymond 推荐给黑客的五种语言写成的。

Blub 悖论

为什么 Lisp 这么棒？如果它真的这么厉害，那为什么不是所有人都使用它？这些问题听起来很值得思考，但他们的答案却是非常直接。Lisp 这么好用，不是因为他拥有一两个仅仅被爱好者所了解的神奇功能，而是因为它就是最强大的语言。至于说为什么并不是人人都使用 Lisp，那是因为程序设计语言并不仅仅是技术，更是一种思维方式。当然，上面的每一句话都需要细细分析。

我先给出一个令人吃惊的、与传统观念格格不入的论断：程序设计语言的能力是有强有弱的。

至少大家公认，高级编程语言比机器语言要强大得多。如今，一般情况下，大多数程序

员都不会建议你使用机器语言编程序。你应该在一个高级语言中编程，然后让编译器帮你把它翻译为机器代码。这个思想甚至已经被带入硬件领域了：自 1980 年开始，指令集就已经是为编译器而非程序员设计的了。

人人都知道，把整个程序用机器代码一句句地写下来绝对是一个错误。而一个并不那么为人所知的事情是这样一个更一般的原则：如果你有一系列语言可供选择，那么在其他情况都相等的条件下，你应当选择最强大的那一个语言来编写程序，别的选择都是错误。

这个原则有很多例外。比如说如果你正在写的程序要与另一个用某种特定语言写的程序密切交互，那么你最好也使用相同的语言进行开发。如果你只需要写一个能完成最基础工作的小程序，例如数值计算或是位操作，那么你最好选一个不那么抽象的语言，因为这样程序运行的效率可能会略高一些。如果你写的是一个简短的，用过就扔的程序，你可能会选择一个语言仅仅是因为它拥有一个能完成你的任务的库。但是一般来说，对于应用软件的开发而言，你需要使用最强大（并且相对有效率）的语言。否则的话，就和使用机器语言编程一样，是一个弥天大错。

显而易见，机器语言非常低级。但至少从传统上来说，高级语言之间是被画等号的。可实际上它们并不等价。从技术上说，“高级语言”一词并没有把事情定义清楚。并没有一条清晰的分割线把机器语言分在一边，而把全部高级语言分到另一边。语言们是根据它们的抽象层次一个接着一个地排列在一起的。从最强大的一直到机器语言，彼此之间的差距很大。

以 Cobol 为例。Cobol 是一个高级语言，这仅仅是因为它需要被编译为机器代码。但是试想，会有人严肃地声称说，Cobol 的能力和，例如 Python，是相同的么？Cobol 或许更像机器语言而不是 Python。

或者来看看 Perl 4？Perl 4 发展到 Perl 5 的时候引入了语法闭包（lexical closures）这一概念。大多数的 Perl 黑客们都认为 Perl 5 要比 Perl 4 的功能更加强大。一旦你承认了这一点，你也就承认了一种高级语言的确可以比另一种高级语言来的强大。无情的推论就是，除了在极个别的特殊情形下，你都应该从所有能够使用的语言中选择功能最强大的那一个来工作。

然而这种思想很少能得到贯彻落实。过了一定的年纪之后，程序员们就几乎不再主动地更换自己使用的语言了。不管用的是哪一种语言，他们都认为它已经足够好了。

程序员对自己钟爱的语言总是显得难舍难分。我不希望伤害任何一个人的感情，所以我会用一个假想中的语言 Blub 来说明这一点。Blub 恰好坐落在语言抽象层次的中间部分。它并非最强大的语言，但是确实比 Cobol 和机器语言要强大一些。

实际上，我们假想出来的 Blub 程序员将不会使用 Cobol 或是机器语言中的任何一个来编程。他们当然不会用机器语言编程，那是编译器要做的事情。至于说 Cobol，他们无法理解怎么能够用那个语言来干事情，它连甲（Blub 中的某个特性）都没有。

当我们的 Blub 程序员在抽象层级上往下看时，他知道他在往下看。没有 Blub 语言强大的语言显然没有 Blub 强大，因为那些语言里没有 Blub 拥有的一些特性。这些特性正是 Blub 程序员所依赖的。但是当我们的 Blub 程序员向着另一个方向看去的时候，他并没有意识到他在往上看。他所看到的仅仅是一些古怪的语言。他或许会认为这些语言的能力和 Blub 是一样的，只不过在此之上又塞进了一些乱七八糟的东西。Blub 对他来说已经够好了，因为他一直在用 Blub 的方式思考问题。

让我们转换视角，作为一个位于更高的抽象层级语言的程序员往下看。这时，我们会发现这个程序员知道自己正在朝下看，朝下看 Blub。“你怎么能够用 Blub 做事情呢？它根本就没有乙。”

由此可见，能够认清每一种语言的能力之间差异的程序员一定是那些了解了最强大的程序语言的程序员。（这恐怕也就是 Eric Raymond 说 Lisp 能让你成为一个更加优秀的程序员的原因。）你不能够盲目听信别人的意见，因为这里有 Blub 悖论：人们总是对当下正在使用

的语言感到非常满意，因为它决定了程序员们思考问题的方式，哪怕这个语言只是一开始是他们碰巧用到的。

作为一个从高中开始就接触 Basic 程序的人来说，我对此感触颇深。Basic 语言甚至不支持递归。现在来看，很难想象不使用递归来书写程序。但是那时的我，几乎一刻也忘不掉我的 Basic 程序。我用 Basic 的方式思考问题，干练老道，掌握了我所能看得到的一切。

Eric Raymond 给黑客们推荐的五种语言都分别坐落在抽象层级的不同位置上。它们之间的相互位置关系是一个非常敏感的话题。而我要说的是，我认为 Lisp 位于这个等级链的顶端。我会告诉你一个 Lisp 拥有而其他几种语言都没有的特性来支持我的论断。在我看来，你怎么能够在那些语言里做事情，它们都没有丙？而最最重要的一个丙就是，Lisp 中的宏。

许多语言都有宏这个概念，但是 Lisp 的宏与众不同。无论你相信与否，这些宏所做的事情都和括号有关。Lisp 的设计者在语言中引入那么多的括号并不仅仅是为了让它看上去非常独特。对于 Blub 程序员来说，Lisp 代码看起来非常诡异。但是那些圆括号出现在那是有原因的。他们诡异的外观正是 Lisp 语言和其他语言之间本质不同的一种外在的表现形式。

Lisp 代码是由 Lisp 数据结构构造出来的。这并不是简单地说，源文件包含一些字符，而字符串是语言支持的一种数据结构。Lisp 的代码在被语法分析器处理之后，就成为了你可以直接操作和处理的数据结构了。

如果你了解编译器的工作原理，那么你会知道这相当于说 Lisp 的语法非常奇特，就好像它根本没有语法一样。你可以直接在语法分析树中编写程序，由编译器来生成程序，而其他的语言则是被语法分析器分析。这些语法分析树是能够被你的程序所完全操控的。你可以写一个程序来操纵它们，这就是 Lisp 中的宏。宏是写程序的程序。

写程序的程序？什么时候竟会用到这种东西？如果你用 Cobol 思考，那确实不是很经常。但是如果你用 Lisp 思考的话，则时时刻刻都需要。如果我能在这里给一个宏的具体例子，讨论会方便许多。但是如果我这么做了的话，势必会让那些不了解 Lisp 的人眼花缭乱、云里雾里。这里没有足够的空间来解释所有那些了解宏所需的知识。在 **ANSI COMMON LISP** 一书中，我已经尽可能快的推进叙述的进度。但即使是那样，讲到宏也是 160 页以后的事了。

但我想，我可以给出一些能够让人信服的佐证。Viaweb 源代码中有大约 20% – 25%是宏。宏比一般的 Lisp 函数要难写，而且滥用宏是一种不好的风格。因此，代码中的每一个宏都是经过精心设计和考量的。这意味着在我们的代码中，有 20% – 25%的部分在做着那些你无法在其他语言中轻易完成的工作。这下不管他们有多怀疑我的论断，至少那些 Blub 程序员对 Lisp 的神奇能力产生了好奇。我们并不是出于好玩才去写这些代码的。我们正在经营一家小小的初创公司，努力编程以在我们和竞争对手之间设置技术壁垒。

一些怀疑心强的人可能会开始质疑这些事情之间的关联性。我们代码中有一大部分做着其他语言做起来非常困难的事情。而结果就是我们的软件能够做的事情比我们竞争对手的软件更多。或许这两者之间是有一些联系的。我鼓励你顺着这条线索一直追寻下去，真相往往不是那么轻松地就浮现在你的眼前，而是需要你勇敢地去探索。

初创公司的合气道

我并没有意图说劝说任何人，尤其是 25 岁以上的人学习并转用 Lisp。这篇文章的目的不是去改变任何一个人心意，而是给那些了解 Lisp 的强大之处、有兴趣使用 Lisp，但是却担心它的小众化的人一剂强心剂。从竞争的角度来说，Lisp 的小众是一大优势。Lisp 的威力会因为你的竞争对手没有使用它而倍增。

倘若你正考虑在一家初创公司中使用 Lisp，那么它的不为大众所知并不是你需要担心的

事情。正相反，你应该对此感到庆幸。程序设计语言的天性是让每一个使用它的人都感到满意。计算机硬件的更新换代要比编程实践传统习惯的更新换代快得多。人们一般总是落后那些先驱者 10 至 20 年。早在二十世纪六十年代初期，MIT 的一些人就已经在使用高级语言编写程序了。而许多公司直到八十世纪仍然在使用机器语言编写代码。我敢打赌，当时一定有一大批固执于使用机器语言的程序员被转而使用精简指令集的处理器抛弃，丢掉了他们的饭碗。

一般来说，技术的更新换代是飞快的。但是程序设计语言则不，它们不仅仅是一种技术，更是程序员思考问题的方式方法。程序设计语言中，一半是技术，一半是信仰。正因为如此，那些中级语言，即那些中级程序员们使用的语言，进化得和冰山一样缓慢。Lisp 在大约 1960 年时引入的废料回收技术，今天已然受到广泛赞誉。运行时类型（Runtime typing）也是如此，正在为更多人所了解。但是 Lisp 在七十年代初期引入的语法闭包概念，今天也仅仅是刚刚萌芽。而六十年代中期的宏，在现在的其他语言中，仍然是毫无进展。

显而易见，这些中间语言有着巨大的惯性，我不期望你能够抗衡这种巨大的力量。恰恰相反，我建议你应该把这一点转化为竞争对手的劣势，就像在合气道中一样。

如果你为一家大公司工作，这可能会困难重重。你固执的上司可能刚刚读了一篇报道，里面说其他的某个编程语言正在准备要统治世界。在 20 年前，Ada 说不定就是那个语言。这时的你就不得不费尽口舌来说服上司让你用 Lisp 来写东西。但是如果你像我们当年一样在一家初创公司中就职，那里没有固执的上司，你就可以把 Blub 悖论转化为你最有利的帮手。你可以用那些固执于中级语言的对手们根本不会考虑使用的强大技术。

如果你正好就在一家初创公司中就职，有一个好办法可以让你评估你的竞争对手的危险程度：看他们的招聘广告。尽管他们的网站上可能全都是些股票走势图或者其他一些无聊透顶的东西，但是招聘广告会告诉你他们真正需要的是什么。没人会在招聘广告里撒慌，不然他们就招不到正确的人了。

我在 Viaweb 工作的几年中，阅读了大量的招聘信息和职位介绍。大约每个月，都会有一个新的竞争对手出现。而我要做的第一件事情不是去看他是否有一个在线试用版本可供使用，而是去读他的职位招聘信息。经过几年的磨砺，我能够从中看出哪些公司是有威胁的，哪些则不是。招聘广告中的 IT 风味越浓厚，它就越不值得你担心。最安全是那些希望应聘者有在甲骨文公司工作经历的公司，你完全不必担心它们会对你造成威胁。如果是招聘 C++ 或是 Java 开发人员，你也不必惊慌。倘若他们想找 Perl 或者是 Python 程序员，那么你就要当心了，这说明那些公司至少在技术方面，是由一些真正黑客级的人物来管理的。如果我看到一家公司在寻找 Lisp 程序员的话，那我就真的要大惊失色了。

我以前在写关于 Lisp 的书的时候，常常希望每一个人都能理解这门语言。但是当我开始经营 Viaweb 时，我发现我的想法改变了：我希望每一个人都能理解这门语言，除了我的竞争对手以外。

译者信息：
Xiaohong Chen
2010 Undergraduate
School of Mathematical Sciences
Peking University
Cell: (+86) 15201479129