程序语言与……

程序语言的设计类似于其它很多东西的设计。有些微妙的地方只有用过更好的设计的人才能明白。现在我就简要介绍一下我自己的体会。

程序语言与微波炉

有的程序语言就像左边的，现在中国市场上流行的微波炉。布满了花哨的一年都用不到一次的专用菜单，却连最基本的 0–9 数字键都没有。输入个时间都要费脑筋组合一下，按键位置不顺手，不能一次按到位，而且还不能达到需要的精度。

有的程序语言就像右边的，美国市场上常见的微波炉，几十年不变的设计。虽然按键很少，但十个数字键总是少不了，而且采用标准的“电话键盘”排列。十个数字能够组合产生出任意的时间，所以不管是在自己家里，别人家里，公司或者学校，你总是可以按照自己的经验，食物包装或者菜谱上的说明，迅速而精确的输入想要的时间。

可惜的是，在中国你已经买不到这么简单实惠的微波炉了。我们中国人学会了美国的很多糟粕，却没有把这么简单，这么好的设计思想学过去。

中国的微波炉厂商之所以放上这么多的花样，是因为商家抓住了中国人的贪便宜心理。看，一个微波炉可以煮米饭，烤肉串，还可以蒸排骨，那其他的厨具都可以不用买啦！可惜因为所以，科学道理，微波就是微波。加热牛奶剩饭之类的事它做得很好，可是要做美味佳肴它就不行了。煮米饭不如电饭煲，烤肉串不如烧烤架，蒸排骨不如蒸锅，炖东西不如砂锅…… 美国人和稍微有点经验的中国人早就知道这个道理，所以从来不期望微波炉能做超越它所擅长的事情。

虽然美国人在这些硬件上非常精明，可是在软件上还没发展到那种地步，很多时候对一些不可救药的软件技术寄予太多的希望。左边的微波炉就好像某些程序语言，本来当初设计就是给标准没那么高的人用来处理很简单的网页的。可是后来有人忽然想让它成为一个“万能语言”，用来做复杂的，对性能和可靠性都很高的服务器程序甚至机器人控制程序。然后你就发现类似微波炉的问题，因为一些不可逾越的设计差别决定了它是不可能把那些事情做好的，而且对有些应用还有严重的安全隐患。当然你可以缓慢的“改进”这语言，让它慢慢的提

高做这些事的水平。可是这种改进的终点也许只是另一种早已存在的语言。而且由于不想破坏已有的代码和特性，所以每一步的改进都异常艰难。这种方式远远不如直接针对需要选择不同的语言，或者设计新的语言来的迅速和有效。

程序语言与减肥

很多人都想减肥，就像很多人都想学会编程。姑且不说一味的减肥好不好，现在只谈一下什么是有效的减肥方法。

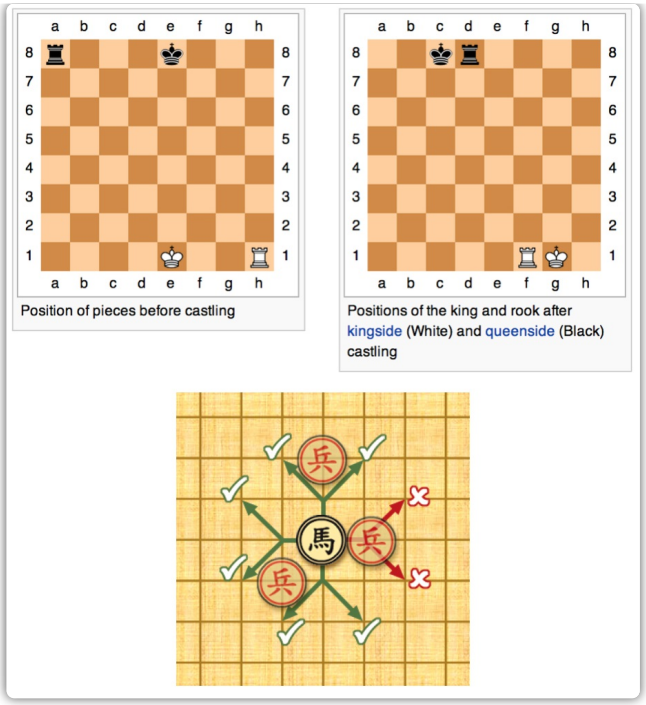
我自己也有一段时间很胖，也有减肥的经历，而且非常成功。如果有一天我不小心又变胖了，我有非常科学而可靠的办法减回去。我的方法就是一句话：让每天吃进去的热量比消耗的少一些，但是不至于难受，另外适当运动来增加热量的消耗。很显然嘛，根据热力学定律，每天消耗的能量比摄入的多，多出来的部分只能通过分解你身上的物质（脂肪）来产生。我的减肥方法就像某些程序语言教会我的编程理念，是不随潮流而改变的真理。它让我的程序不管用什么语言写都优美而精悍。

我不是自私的人，我希望大家都健康一点，养眼一点。我已经轻易地告诉了你减肥的终极真理，一分钱都不收，可是你不相信我。你觉得肯定没那么简单，或者你觉得那样太辛苦，自己不可能照办。这就像很多人对编程的希望：要是我不学编程也能编程该多好啊！

很多程序语言就是针对这群人而产生的，它们大部分的工作花在了研究人的心理和做广告上面。它们就像电视广告里铺天盖地的减肥药：不需运动，不用节食，一个星期瘦 20 斤！它们提出各种新的术语，什么减肥茶，片，胶囊，螺旋，燃脂，纤维，宫廷，祖传，秘方，各种生化术语…… 再加上一些 PS 出来的前后效果对比图，你痛快地花不菲的价钱买了这药，然后每天好几次的像做化学实验一样精确的按时按量服用。这时候任何人跟你说这药不灵的话你都不会相信，你觉得这些人都是想跟你争夺异性的目光故意想让你继续胖下去而其实她（他）们自己背地里也吃这药，所以你对此减肥药必胜的信心有增无减。

当然你不会成功。在持续服用好多个月，甚至好几年之后，你按照广告里说的“无效退款”条例要求退款。可是减肥药公司说，是你自己没有按说明书服用，或者你吃药之前肯定比现在还胖很多。你拿不出证据，后悔当初没到公证处开你当时体重的证明。可是你仍然相信，世界上一定会有真正有效的减肥药。你觉得国内的公司喜欢骗人，所以你到了美国，寻找传说中那世界一流的减肥药……

程序语言与棋

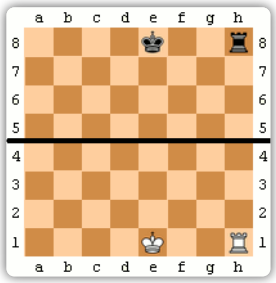


有人说好的程序语言就像国际象棋（chess），在了解简单的规则之后，你就可以用它们组合出变幻无穷的棋局。而我认为，好的程序语言应该像国际象棋去掉像“王车易位”（castling）一类复杂古怪的规则。实际上，好的程序语言会更加近似于中国象棋，而不是国际象棋。中国象棋只有一条规则比较特殊—“蹩脚马”，可是它其实很直观，容易理解。其它的规则，比如兵卒过河才能横行，几乎都画在棋盘上了。

可不要小看国际象棋里这少数几个特殊规则，它们需要在好几个非常特殊的条件满足之后才会生效，而且路线诡异。比如，王车易位必须满足：

1. 王和跟他换位的车都没有移动过
2. 王和车之间没有其它棋子
3. 王不能处于被“将军”的状态而且王在换位之后不能处于被攻击的位置但是车可以在换位后处于被攻击位置
4. 王和车处于同一条水平线上

另外换位的时候王和车不是直接互换位置那么简单，而是这样的路线：



一条这样的特殊规则就够伤脑筋了，据我所知国际象棋还有至少其它两条类似的规则。它们跟其他的规则组合在一起的时候就产生了组合爆炸效应，你发现每走一步，甚至貌似无关的动作都得检查它们是否会出现。你不得不随时把这么复杂的规则放在脑子里。没事找事也不要找这么麻烦的事啊。

这些规则就像是要你记住 C 语言里的 ++i+i++ 或者 if (x = “foo”) {…} 是什么意思。经过多年的痛苦经历之后，你多希望不再需要理解这样的代码。可是一旦这样的规则被加到语言里面，总会有人为了显示自己的水平和记忆力去用它们。不得已，你只好陪他们玩。

如果你觉得多了这些无厘头的规则会让国际象棋比中国象棋难度大或者更加有

趣，那你就低估了中国象棋了。中国象棋的“游戏树复杂度”其实比国际象棋还要高，高达 10^{150}，而国际象棋只有 10^{123}。这跟中国象棋的棋盘要稍微大点有关系，但是总比记忆那些麻烦的规则好多了。所以相对来说中国象棋既简单又耐玩。

如果国际象棋还凑合算是简单的话，大部分的程序语言就像是魔鬼棋，飞行棋，或者三国杀。它们几乎完全由类似的特殊规则构成。哇，那么多的人物，道具和特殊技，好玩！可是会玩象棋或者国际象棋的人都会觉得它们无聊透顶。

那么是不是规则越简单越少的棋越好呢？围棋就比中国象棋还简单，那么围棋是不是更好玩呢？我觉得不是的。围棋对我来说太慢，太单调，棋盘太大，耗时太多，而且胜负居然不能一眼就看出来，要数好一会儿！这哪里是在玩，纯粹就是在做组合优化题嘛。我觉得这种任务适合交给电脑去做。所以其实简单也有一个界限，超过了这个界限对于人就没有很大区别了，反而会开始感觉缺少一些东西。

我觉得中国象棋和围棋一样简单，它的规则虽然比围棋多，但是仍然处于人脑容易记忆的范围，而且每条规则都很直接了当，没有很隐晦的条件。中国象棋的长距离武器（车和炮）让它比围棋多了很多乐趣，而对于象，马和王的走法的限制，让它比起国际象棋多了几分安心和舒适。国际象棋的后，两个车，两个相的攻击距离和范围太大，让人觉得眼睛很辛苦，因为每一个位置都可能被从太多个方向远距离攻击。而那个王，由于可以到处乱跑，以至于你感觉不是在抓一个住在戒备森严的城堡里的人，而是一只在野外乱跑的老鼠。

什么游戏会让人觉得有趣，真是一个值得研究的问题。我觉得象棋和我以前推荐过的一个游戏 Braid 里面含有同一种吸引人的设计：屈指可数但又有足够变化的简单规则，组合起来制造出许许多多的变化。这种特征其实也是鉴别一个优秀的程序语言的标准。

程序语言与音乐

<img src=”http://www.yinwang.org/images/goldberg.png” width=200>

程序语言就像音乐。当听过很好的音乐之后，你会自然而然的厌倦以前曾经喜欢过，为之疯狂过的那些，觉得它们很无趣，甚至很惊讶自己以前怎么会喜欢它们。当有人问你为什么不喜欢他们推荐给你的音乐，你却说不出来。你只是自然而然觉得太单调，不入耳，不对劲，甚至扰乱你美好的心情。你的判断完全是依靠声波对鼓膜的震动而引起的脑电波的起伏，而不带有任何的成见。完全根据这音乐自己，而不需要知道它的作者是谁。就像玩过像《Braid》之类的游戏之后，你再也不想玩像《生化危机》那种搞不清楚到底是自己在玩游戏还是游戏在玩自己的。你的脑子里有一种对“趣味”的新定义，但是你却说不出来它到底是怎么回事。

程序语言是同样的感觉，这是一个“流行语言”招摇过市的年代。每当有人问我喜欢什么程序语言我都不好跟他说，因为一旦说出来就有显摆之嫌，而其实真正显摆的是其他人。很多人期望你的回答是他所膜拜的那个最近很热门的语言，你一旦告诉他你喜欢的语言就会被冷嘲热讽，因为你的语言不热门。他们会说你是“学院派”，而他们是“工程派”，而其实这只是给垃圾的存在找借口。他们利用你害怕自己被认为是附庸风雅或者居高临下的心理来变相地压制你，让你不敢直率的袒露自己的兴趣。你不敢显示对有些东西的不屑，而他们却可

以任意的显示对真正优秀的技术的不屑。你觉得应该手下留情一些，谦虚一些，结果最后一些垃圾一样的语言就骑到你头上来，让你不得不用它们。

用过很好的语言，然后自己设计过程序语言之后，我再也不对很多新的语言，或者有些人很崇拜的古老的语言感兴趣了。我完全是凭自己的感觉来判断，一些所谓的“新特性”其实是老酒换新瓶，或者是勾兑的假酒。程序语言本来就只有那么点东西，为什么有人仍然像对那些扮相的流行歌手一样热衷和疯狂。

我知道这些话说了也白说，因为他们没有用过我用过的语言，他们只看到名字却感觉不到本质，他们靠别人的评价来判断，而不是靠自己的心。所以像音乐一样，只有等有一天他们忽然觉悟，就像很多年前的我一样。

程序语言与武器

<img src="http://www.yinwang.org/images/AK-47.png" width=400>

前段时间 AK-47 的设计者 Kalashnikov 去世的时候，我从一篇文章了解到他设计 AK-47 的故事，发现 AK 跟我喜欢的程序语言设计有异曲同工之妙。

> AK 简单得就像一把锤子。它身上没有太空时代的材料。大多数汽车修理店都有可以制造出 AK 的工具。

这篇文章首先提到，AK 的高可靠性最主要来自于它的简单，而其实简单也是程序语言最重要的东西。程序员需要解决的问题一般都挺复杂，如果他们的工具再被设计得复杂，那么他们大量的脑力就被浪费在解决这语言的问题，而不是真正需要解决的问题了。

> Kalashnikov 开始的时候把任何有可能出问题的设计都排除在外了。

与简单的设计背道而驰，现在很多程序语言为了赶潮流或者吸引眼球，喜欢标新立异，喜欢加入很多“特性”，可是这些特性很有可能不但不解决问题，而且会制造问题。绝大部分程序员都不理解这个道理，所以有些人听说我在设计自己的语言就问我：“它有什么新特性吗？”我没法回答他们，因为我的设计几乎没有新的特性。我现在所做的一切思考和试验都是在去掉不必要的麻烦。一个语言缺少一些好的特性，以后还可以加进去，可是它如果多了一些问题特性，那一旦有人开始用就没法去掉了。

> AK 上面没有袖珍和娇气的部件。这样你就不用费事在草丛里，泥地上或者溪流里找它们了。

士兵是人，会摔跤犯错误，程序员也是人，所以程序员的武器应该像士兵的武器一样，方便他们找到问题。可是很多程序语言让程序员犯错误之后花很多时间和精力才能找到错误的所在，浪费大量本来可以用来解决问题的时间。我的前同事 TJ 说他刚进入博士学习的时候花了好几个月，就为了找到 C 代码里面一个指针计算错误，导致内存结构破坏和莫名其妙的错误结果，而出现指针计算错误的位置跟错误结果出现的位置毫无关系。我也遇到过类似的问题。C 语言的指针不就像是某些武器上面的袖珍部件吗？一不小心掉在地上就找不到了。

> AK 只有一个复杂一点的部件，那就是它的弹夹。弹夹的设计很大程度上影响到枪的整体性能，所以 Kalashnikov 在上面花了很多设计时间。

<img src="http://www.yinwang.org/images/ak-magazine.png" width=300>

这个工程经验其实对于程序语言的设计者有启发意义，因为弹夹与枪主体的接口，和程序语言的函数接口很类似。Tony Hoare 在他的《给程序语言设计的建议》中也提到，函数的调用必须简单而且快速，因为调用的开销会累积起来形成很大的性能问题。可惜的是很多语言没有注意到这个问题，函数调用时总是有一堆的动态检查和重载要做，很大程度的影响了它们的性能。

> AK 的美，在于它身上没有部件具有不必要的精确性。

这对于程序语言或者编程来说也是有启发意义的。有些人为了所谓的程序“正确性”，损害了它的简单性。他们的代码异常复杂，而且喜欢写很多测试，让自己感觉对程序的“质量”有个底。然而这其实是自欺欺人。这些测试不但不能保证程序的正确，它们阻碍了程序员对程序进行彻底性的修改，防止了他们看到更加简单，甚至一眼就知道是正确的解决方案。

程序语言的设计也是。有些语言（特别是所谓 dependent type 的语言）想达到程序的完全正确，加入了很多很多的限制条件，要求程序员写很多的辅助声明甚至机器证明。结果很简单一个问题都需要很长的代码才能写出来，这些辅助的逻辑代码严重的影响了程序的阅读和转换。而且由于数理逻辑本身的局限性，它们经常迫使程序员的思路绕弯子。其实起到了相反的结果，让他们看不到更简单的方法。

> Kalashnikov 不是天才，他不是为了发明而发明，他解决不了问题的时候就高兴地从别人那里学过来。

这是非常值得我们程序语言设计者学习的。很多程序语言专家都有盲目排斥“对手”的心理，“自己人”的东西就不假思索的表扬，对手的东西就一味的批评。最后的结果是没有把敌人的好东西学过来，让自己人吃亏。在操作系统和数据库等领域也有类似的思维方式，这是非常有害的。

> 直到被更好的东西取代，AK 会继续和我们在一起。什么才是“更好”，这是由历史和民族来定义的，而不是枪支设计专家。

在计算机的世界里也是一样，程序语言，操作系统，数据库…… 它们的好坏不应该是由它们的设计者决定的，而是看它们是否经得起时间的考验。很多几十年前以为是好的设计，到现在已经很明显的显示出了它们的缺点。这就是为什么我喜欢批评一些语言，操作系统和数据库的设计，因为我看到了它们在历史的长河中已经快要到达终点。自欺欺人的掩盖这些缺陷只会让我们输掉战争。