“更糟即更好”的兴起

Richard P. Gabriel

Lucid, Inc

摘自《Lisp：好消息，坏消息，如何大获全胜》。[html]}

2.1 “越糟越好”的兴起

我，以及几乎所有 Common Lisp 和 CLOS 的设计师，都对麻省理工学院/斯坦福大学的设计风格有着深入的了解。这种风格的精髓可以用“做正确的事”来概括。对于这样的设计师来说，掌握以下所有特征至关重要：

- 简单性——设计必须简单，包括实现和接口。接口的简单性比实现的简单性更重要。
- 正确性——设计必须在所有可观察到的方面都正确。绝对不允许出现错误。
- 一致性——设计必须保持一致性。为了避免不一致，设计可以稍微简化一些，完整性也可以稍微降低一些。一致性与正确性同等重要。
- 完整性——设计必须尽可能多地涵盖实际需要的重要情况。所有合理预期的情况都必须涵盖。简单性不能过度降低完整性。

我相信大多数人都会认同这些都是很好的特性。我将这种设计理念的运用称为 MIT 方法。Common Lisp（包含 CLOS）和 Scheme 代表了 MIT 的设计和实现方法。

“更糟即是更好的”哲学只有一点不同：

- 简单性——设计必须简单，包括实现和接口。实现的简单性比接口的简单性更重要。简单性是设计中最重要的考虑因素。
- 正确性——设计必须在所有可观察到的方面都正确。简单比正确略胜一筹。
- 一致性——设计不能过于不一致。在某些情况下，为了简单性可以牺牲一致性，但最好放弃那些处理不常见情况的设计部分，而不是引入实现的复杂性或不一致性。
- 完整性——设计必须尽可能多地涵盖重要的实际情况。所有合理预期的情况都应涵盖。完整性可以为了其他任何特性而牺牲。事实上，只要实现的简单性受到威胁，就必须牺牲完整性。如果要保留简单性，为了实现完整性可以牺牲一致性；接口的一致性尤其没有价值。

早期的 Unix 和 C 就是使用这种设计流派的例子，我将这种设计策略的使用称为新泽西方法，我故意讽刺“更差即是更好的”哲学，以让你相信这显然是一种糟糕的哲学，而新泽西方法是一种糟糕的方法。

然而，我相信“更差即是更好”，即使是稻草人形式，也比“正确的事情”具有更好的生存特性，并且新泽西方法用于软件时比麻省理工学院的方法更好。

首先，让我重述一个故事，以表明麻省理工学院/新泽西大学的区别是有效的，并且每种哲学的支持者实际上都认为他们的哲学更好。

两位名人，一位来自麻省理工学院，另一位来自伯克利大学（但从事 Unix 相关工作），曾聚在一起讨论操作系统问题。麻省理工学院的那位对 ITS（麻省理工学院人工智能实验室）很了解。

他是一位资深程序员（他研究过许多操作系统），一直在阅读 Unix 源代码。他对 Unix 如何解决 PC loser-ing 问题很感兴趣。PC loser-ing 问题发生在用户程序调用系统例程执行可能具有重要状态（例如 IO 缓冲区）的冗长操作时。如果在操作期间发生中断，则必须保存用户程序的状态。由于系统例程的调用通常是一条指令，因此用户程序的 PC 无法充分捕获进程的状态。系统例程必须退出或继续执行。正确的做法是退出并将用户程序 PC 恢复到调用系统例程的指令，以便在中断后恢复用户程序时（例如）重新进入系统例程。这被称为 PC loser-ing，因为 PC 被强制进入 loser 模式，其中 loser 是麻省理工学院对用户的昵称。

麻省理工学院的那位同学没有看到任何处理这种情况的代码，于是就问了新泽西的那位同学，这个问题是怎么解决的。新泽西的那位同学说，Unix 的程序员都知道这个问题，但解决方案是让系统例程始终完成，但有时会返回一个错误代码，表示系统例程未能完成其操作。正确的用户程序必须检查错误代码，以确定是否只需再次尝试执行系统例程即可。麻省理工学院的那位同学不喜欢这个解决方案，因为它并不合适。

新泽西人说 Unix 的解决方案是正确的，因为 Unix 的设计理念是简洁，而正确的东西太复杂了。此外，程序员可以轻松地插入额外的测试和循环。麻省理工学院的人指出，实现很简单，但功能接口却很复杂。新泽西人说，Unix 选择了正确的权衡——即实现的简洁性比接口的简洁性更重要。

麻省理工学院的人随后嘟囔道，有时候需要坚强的人才能做出嫩鸡，但新泽西人并不理解（我也不确定自己是否理解）。

现在我想论证的是，越糟糕越好。C 语言是专为编写 Unix 系统而设计的编程语言，它采用了新泽西方法。因此，C 语言很容易编写出像样的编译器，而且它要求程序员编写易于编译器解释的文本。有些人称 C 语言为“花哨的汇编语言”。早期的 Unix 和 C 语言编译器结构简单，易于移植，运行时所需的机器资源很少，并且能够满足操作系统和编程语言所需功能的 50% 到 80%。

现有的计算机中，有一半的性能比中位数更差（更小或更慢）。Unix 和 C 语言在这些机器上运行良好。“差即是好”的理念意味着实现的简单性具有最高优先级，这意味着 Unix 和 C 语言很容易移植到这些机器上。因此，人们可以预期，如果 Unix 和 C 语言对 50% 功能的支持令人满意，它们就会开始无处不在。事实也的确如此，不是吗？

Unix 和 C 是终极计算机病毒。

“差即是好”哲学的另一个好处是，程序员习惯于牺牲一些安全性、便利性和麻烦，以获得良好的性能和适度的资源利用率。使用新泽西方法编写的程序在小型和大型机器上都能很好地运行，而且由于代码是在病毒之上编写的，因此具有可移植性。

重要的是要记住，最初的病毒必须基本良好。如果这样，只要病毒易于移植，就能确保其传播。病毒一旦传播开来，就会有改进的压力，可能需要将其功能提升近 90%，但用户已经习惯于接受比正确版本更差的版本。因此，首先，“差即是好”的软件会获得认可，其次会让用户降低期望值，第三会得到改进，使其接近正确版本。具体来说，尽管 1987 年的 Lisp 编译器与 C 编译器差不多好，但想要改进 C 编译器的编译器专家比想要改进 Lisp 编译器的编译器专家多得多。

好消息是，1995 年我们将拥有一个优秀的操作系统和编程语言；坏消息是它们将是 Unix 和 C++。

“差即是好”还有最后一个好处。由于新泽西语言和系统本身功能不足以构建复杂的单体软件，大型系统必须设计为可复用组件。因此，集成的传统应运而生。

正确的事情该如何积累？有两种基本情景：大型复杂系统情景和钻石般宝石情景。

大型复杂系统场景如下：

首先，需要设计出正确的东西。然后，需要设计它的实现。最后，它被实现了。正因为它是正确的东西，它几乎实现了 100%的预期功能，而且实现的简单性从未成为关注点，所以它需要很长时间才能实现。它庞大而复杂。需要复杂的工具才能正确使用。最后 20%的实现耗费了 80%的精力，因此，正确的东西需要很长时间才能问世，而且只有在最复杂的硬件上才能令人满意地运行。

The diamond-like jewel scenario goes like this:

正确的东西需要花很长时间去设计，但它在设计过程中的每个环节都非常小巧。实现它并让它快速运行要么是不可能的，要么超出了大多数实现者的能力。

The two scenarios correspond to Common Lisp and Scheme.

第一个场景也是经典人工智能软件的场景。

正确的东西通常是一个单体软件，但这仅仅是因为正确的东西通常都是单体设计的。也就是说，这种特性纯属偶然。

从中我们可以汲取的教训是，一开始就追求正确的东西往往是不可取的。最好先做好一半，让它像病毒一样传播开来。一旦人们迷上了它，就花时间把它改进到 90%的正确程度。

一个错误的教训是，仅仅从字面上理解这个寓言，就断定 C 语言是开发人工智能软件的正确工具。50% 的解决方案必须基本正确，但在这种情况下，它并非如此。

但最终结论是，Lisp 社区需要认真反思其在 Lisp 设计上的立场。我稍后会详细阐述这一点。