什么是“脚本语言”

很多人都会用一些“脚本语言”（scripting language），却很少有人真正的知道到底什么是脚本语言。很多人用 shell 写一些“脚本”来完成日常的任务，用 Perl 或者 sed 来处理一些文本文件，很多公司用“脚本”来跑它们的“build”（叫做 build script）。那么，到底什么是“脚本语言”与“非脚本语言”的区别呢？

其实“脚本语言”与“非脚本语言”并没有语义上，或者执行方式上的区别。它们的区别只在于它们设计的初衷：脚本语言的设计，往往是作为一种临时的“补丁”。它的设计者并没有考虑把它作为一种“通用程序语言”，没有考虑用它构建大型的软件。这些设计者往往没有经过系统的训练，有些甚至连最基本的程序语言概念都没搞清楚。相反，“非脚本”的通用程序语言，往往由经过严格训练的专家甚至一个小组的专家设计，它们从一开头就考虑到了“通用性”，以及在大型工程中的可靠性和可扩展性。

首先我们来看看“脚本”这个概念是如何产生的。使用 Unix 系统的人都会敲入一些命令，而命令貌似都是“一次性”或者“可抛弃”的。然而不久，人们就发现这些命令其实并不是那么的“一次性”，自己其实一直在重复的敲入类似的命令，所以有人就发明了“脚本”这东西。它的设计初衷是“批量式”的执行命令，你在一个文件里把命令都写进去，然后执行这个文件。可是不久人们就发现，这些命令行其实可以用更加聪明的方法构造，比如定义一些变量，或者根据系统类型的不同执行不同的命令。于是，人们为这脚本语言加入了变量，条件语句，数组，等等构造。“脚本语言”就这样产生了。

然而人们却没有发现，其实他们根本就不需要脚本语言。因为脚本语言里面的这些结构，在任何一种“严肃”的程序语言（比如 Java，Scheme）里面，早就已经存在了，而且设计得更加完善。所以脚本语言往往是在重新发明轮子，甚至连轮子都设计不好。早期脚本语言的“优势”，也许只在于它不需要事先“编译”，它“调用程序”的时候，貌似可以少打几个字。脚本语言对于 C 这样的语言，也许有一定的价值。然而，如果跟 Scheme 或者 Java 这样的语言来比，这个优势就非常不明显了。比如，你完全可以想一个自动的办法，写了 Java 代码之后，先调用 Java 编译器，然后调用 JVM，最后删掉 class 文件。或者你可以选择一种有解释执行方式的“严肃语言”，比如 Scheme。

很多人把 Scheme 误称为“脚本语言”，就是因为它像脚本语言一样可以解释执行，然而 Scheme 其实是比 C 和 Java 还要“严肃”的语言。Scheme 从一开头就被设计为一种“通用程序语言”，而不是用来进行某种单一简单的任务。Scheme 的设计者比Java 的设计者造诣更加深厚，所以他们对 Java 的一些设计错误看得非常清楚。像 Chez Scheme 这样的编译器，其实早就可以把 Scheme 编译成高效的机器代码。实际上，很多 Scheme 解释器也会进行一定程度的“编译”，有些编译为字节码，有些编译为机器代码，然后再执行。所以在这种情况下，通常人们所谓的“编译性语言”与“解释性语言”，几乎没有本质上的区别，因为你看到的“解释器”，不过是自动的先编译再执行。

跟 Java 或者 Scheme 这样的语言截然不同，“脚本语言”往往意味着异常拙劣的设计，它的设计初衷往往是目光短浅的。这些语言里面充满了历史遗留下来的各种临时的 hack，几乎没有“原则”可言。Unix 的 shell（比如 bash，csh，……），一般都是这样的语言。Java 的设计也有很多问题，但也跟“脚本语言”有天壤之别。然而，在当今现实的工程项目中，脚本语言却占据了它们不该占有的地位。例如很多公司使用 shell 脚本来处理整个软件的“build”过程或者测试过程，其实是相当错误的决定。因为一旦这种 shell 脚本日益扩展，就变得非常难以控制。经常出现一些莫名其妙的问题，却很难找到问题的所在。Linux 使用 shell 脚本来管理很多启动项目，系统配置等等，其实也是一个历史遗留错误。所以，不要因为看到 Linux 用那么多 shell 脚本就认为 shell 语言是什么好东西。

如果你在 shell 脚本里使用通常的程序设计技巧，比如函数等，那么写几百行的脚本还不至于到达不可收拾的地步。可是我发现，很多人头脑里清晰的程序设计原则，一遇到“写脚本”这样的任务就完全崩溃了似的，他们仿佛认为写脚本就是应该“松散”一些。很多平时写非常聪明的程序的人，到了需要处理“系统管理”任务的时候，就开始写一些 shell 脚本，或者 Perl 脚本。他们写这些脚本的时候，往往完全的忘记了程序设计的基本原则，例如“模块化”，“抽象”等等。他们大量的使用“环境变量”一类的东西来传递信息，他们忘记了使用函数，他们到处打一些临时性的补丁，只求当时不出问题就好。到后来，他们开始耗费大量的时间来处理脚本带来的麻烦，却始终没有发现问题的罪魁祸首，其实是他们错误的认为自己需要“脚本语言”，然后认为写脚本的时候就是应该随便一点。

所以我认为脚本语言是一个祸害，它几乎永远是错误的决定。我们应该尽一切可能避免使用脚本语言。在没有办法的情况下（比如老板要求），也应该在脚本里面尽可能的使用通常的程序设计原则。