世界著名计算机教材精选

程序设计语言概念

John C. Mitchell 著

冯建华 王 益
廖雨果 李国良 译

CONCEPTS IN PROGRAMMING LANGUAGES

清华大学出版社

世界著名计算机教材精选

Concepts in Programming Languages

程序设计语言概念

John C. Mitchell 著

Stanford University

冯建华 王益 廖雨果 李国良 译

清华大学出版社

北京

Originally published by Cambridge University Press in 2000.
This reprint edition is published with the permission of the Syndicate of the Press of the University of Cambridge, Cambridge, England.

图书在版编目(CIP)数据

程序设计语言概念 /（美）米切尔（Mitchell, J. C.）著；冯建华等译. —北京：清华大学出版社，2005. 10
（世界著名计算机教材精选）
ISBN 7-302-11107-3

I. 程…　II. ①米…　②冯…　III. 程序语言-高等学校-教材　IV. TP312

中国版本图书馆 CIP 数据核字（2005）第 054271 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969
组稿编辑：龙啟铭
文稿编辑：王敏稚
印 刷 者：清华大学印刷厂
装 订 者：三河市金元装订厂
发 行 者：新华书店总店北京发行所
开　　本：185 × 260　**印张**：28.25　**字数**：704 千字
版　　次：2005 年 10 月第 1 版　2005 年 10 月第1 次印刷
书　　号：ISBN 7-302-11107-3/TP · 7346
印　　数：1 ~ 3000
定　　价：56.00 元

前　言

最好的程序设计语言就是编程思考中的概念上的世界。
——Alan Perlis，NATO 软件工程技术会议，罗马，1969

程序设计语言为程序员写出一个好的程序提供了所需的抽象机制、组织原则以及控制结构。本书所介绍的是在程序设计语言中出现的概念，即在程序设计语言的实现过程中产生的问题，以及语言的设计方式对程序开发产生的影响。

本书分为 4 个部分：

- 第 1 部分：函数与基本原理
- 第 2 部分：过程、类型、内存管理与控制
- 第 3 部分：模块、抽象与面向对象程序设计
- 第 4 部分：并发性与逻辑编程

第 1 部分将 Lisp 作为分析程序设计语言的示例，对其进行了简单介绍，内容包括编译器结构、解析、朗母达演算以及指称语义。可计算性一章还涉及了编译时程序分析和优化的限制。

第 2 部分通过过程化的 Algol 系列语言和 ML，介绍了类型、内存管理和控制结构。

第 3 部分介绍使用抽象数据类型、模块和对象来组织程序。由于目前面向对象编程广受推崇，于是我们对几种面向对象语言进行了对比。有专门的章节对 Simula、Smalltalk、C++和 Java 进行研究和比较。

第 4 部分介绍了支持并发性的语言机制和逻辑编程。

本书面向的读者是有一定编程基础的大学本科高年级学生和研究生新生。他们理解 C 或其他过程化语言，熟悉 C++或者其他面向对象的程序设计语言。如果读者具备一些 Lisp、Scheme 或者 ML 的经验将会对第 1 部分和第 2 部分的理解有所帮助，但不具备这些背景知识也同样能学好这门课程。对算法和数据结构进行简单分析的经验也对理解本书有所帮助。例如，在比较某种构造的实现方式的时候，如果能够区分常数时间复杂性、多项式时间复杂性和指数时间复杂性将有助于理解。

在学习了本书之后，读者将会对过去 40 年中所使用过的各种程序设计语言有更好的理解，对程序设计语言的设计过程中出现的问题和折衷有更深的认识，也会对所使用的程序设计语言的利弊有更透彻的了解。由于不同的语言体现了不同的编程概念，把其他语言中的思想引入到自己所编写的程序中将会提高读者的编程能力。

致谢

这本书的手稿源于我从 1993 年开始开设的一门程序设计语言课程（Standford CS 242）的笔记。每年都有精力充沛的助教帮助我调试课程的示例程序，设计课程作业和准备解决方案模型。该课程的组织和内容都受益于他们的建议。特别感谢 Kathleen Fisher，他在 1993

年和 1994 年担任助教，并于 1995 年我不在校的时候教授课程。Kanthleen 早些年帮我组织材料，并在 1995 年将我的手稿转录成在线文档。感谢 Amit Patel 主动组织布置作业和解决方案，感谢 Vitaly Shmatikov 对程序设计语言术语表做出的不懈努力。Anne Bracy、Dan Bentley 和 Stephen Freund 仔细地校对了许多章节。

剑桥大学出版社的 Lauren Cowles、Alan Harvey 和 David Tranah 给予我支持和帮助。我要特别感谢 Lauren 对草稿的所有 12 章都仔细阅读并详细做注。同时也要感谢他们邀请的校阅者，他们对本书的早期版本提出了很多宝贵的建议。Zena Ariola 从本书的初稿开始就连续几年在俄勒冈州大学教授此书，并提出了很多很好的建议；还有很多其他讲师也提供了很多建议。

最后，特别感谢 Krzystof Apt 对“逻辑编程”一章做出的贡献。

John C. Mitchell

目　录

第 1 部分　函数与基本原理

第 2 部分 过程、类型、内存管理与控制

第 4 部分 并发性与逻辑编程

第 1 部分

函数与基本原理

第1章 导 言

媒介就是消息。

——Marshall McLuhan

1.1 程序设计语言

在计算机编程中，程序设计语言是作为表达的媒介。一个理想的程序设计语言能让程序员轻松地编写出简洁、清晰的程序。由于程序在其生存期中必须易于理解、修改和维护，一个好的程序设计语言还要让其他人易于阅读程序并理解它们是怎样工作的。软件的设计和构造是一项复杂的工作。许多软件系统都由相互作用的多个部分组成。这些部分，或者说是软件的构件，它们之间的交互可能非常复杂。为了控制复杂性，必须仔细地定义这些构件之间的接口和消息传送。一个适于大规模编程的、好的程序设计语言将会帮助程序员高效地管理软件组件之间的交互。当我们评价程序设计语言的时候，必须考虑到设计、实现、测试和维护软件的工作，看看每种语言对软件生命周期中的各个部分是如何提供支持的。

在程序设计语言的设计过程中有很多时候需要折衷利弊，这是项困难的工作。有些语言的特色使得我们能够很快地编写出程序，但是在设计测试工具和方法的时候却要困难得多。而有些语言的构造方式使得编译器能够容易地对程序进行优化，但又会使得编程过程烦琐。由于不同的计算环境和应用需要不同的程序特性，于是不同的程序语言设计者选择了不同的折衷办法。实际上所有成功的程序语言最初都是为某一特殊用途而设计的。这并不是说每种语言都只适于一种目的。然而，把注意力集中在某一应用上能够帮助语言设计者作出一致的、有目的的决定。单一应用还有利于解决语言设计的最难对付的问题：遗漏好的想法。

即使你不使用本书中的许多程序设计语言，你仍能运用这些语言所体现的概念框架。在20世纪70年代中期，当我还是学生的时候，所有“严谨”的程序员（至少在我所在的大学）都使用Fortran语言。Fortran语言不允许递归，而大家普遍认为递归的效率太低，对“实际的编程”不实用。然而，我选修的一门课程的讲师认为递归是一个非常重要的思想，并向我们解释在Fortran语言中可以怎样利用递归技术管理数组数据。我很高兴我选修了那门课程，从而认识到递归并不是一个不实际的思想。在20世纪80年代，许多人认为面向对象编程效率不高、太笨拙，不适合实际的编程。但是，在20世纪90年代，学生会为自己在80年代学习了面向对象编程这一“未来派”语言而感到高兴，因为面向对象编程已经被广泛接受和使用。

尽管这不是一本讲述程序设计语言历史的书，但整本书都对历史有一定关注。讨论历史上的各种语言的目的之一是，通过事实来理解程序设计语言中的各种折衷方案选择。例

如，当机器速度很慢并且内存资源很宝贵时的程序跟现在的程序是不同的。所以 20 世纪 60 年代的程序语言设计者所关心的问题与现在的程序语言设计者所关心的问题不同。通过想像过去年代中的实际情况，我们就能更深地理解为什么语言设计者要做出那些决定。这种思考语言和计算的方法可能会在将来对我们有所帮助。例如，最近掌上型计算设备和嵌入式处理器的兴起使得人们对有限内存和有限计算能力的设备编程又重新恢复了兴趣。

在本书中当我们讨论某种特定语言的时候，我们一般都指的是这一语言的原始形式，或者是其在历史上很重要的形式。比如，“Fortran”指的是 20 世纪 60 年代和 70 年代早期的 Fortran。这些早期语言被称为 Fortran Ⅰ、Fortran Ⅱ、Fortran Ⅲ，等等。在最近几年，Fortran 逐步演化，包括了更多现代特征，Fortran 与其他语言的区别在某种程度上已经不太明显。同样，Lisp 一般都是指 60 年代的 Lisp，Smalltalk 一般指 70 年代后期和 80 年代的 Smalltalk，等等。

1.2 目标

在这本书中，我们所关心的是现代程序设计语言中出现的基本概念，它们之间的相互关系，以及程序设计语言与程序开发方式的关系。一个重复出现的话题是语言表现性与实现的简单性之间的折衷。对于我们考虑的每一个程序设计语言特征，都将研究在编程中能够使用它的各种方式，以及可以用来高效编译和执行它的各种实现技术。

1.2.1 总体目标

本书有以下总体目标。

- 理解程序设计语言的设计空间。这包括过去的程序设计语言的概念和构造，以及将来可能会被广泛使用的概念和构造。我们还会试图理解语言特征之间的一些主要的冲突和折中，包括实现代价。
- 通过各种程序设计语言之间的比较，更好地理解我们目前所使用的程序设计语言。
- 理解与各种语言特征所关联的编程技术。学习程序设计语言的一个方面，就是要学习解决问题、软件构造和开发的概念框架。本书中的很多思想都是职业程序员的共识。本书所介绍的内容和思考方式将会对你以后的编程有所帮助，如果你在一个软件公司工作或者获得了一个工作面试机会，这也会对你有所帮助。当这门课程结束的时候，你将能够评价一个语言的各种特征、它们的代价以及它们是如何结合在一起的。

1.2.2 特殊主题

下面是本书中将反复强调的内容。

- **可计算性** 有些问题是计算机无法解决的。停机问题的不可判定性意味着程序设计语言的编译器和解释器并不能完成我们希望它们能够完成的所有工作。

- **静态分析** 编译时和运行时是有区别的。在编译时，程序是确定的但是输入是未知的。在运行时，程序和输入对于运行时系统来说都是确定的。尽管程序设计者或实现者希望能在编译时找出错误，但很多错误都是在运行时才能被发现的。在编译时检测程序错误的方法通常都是保守的，即当它们说某个程序不含有某种类型的错误，这句话肯定是正确的。然而，编译时错误检测方法会说程序含有某些错误，而实际上程序在运行的时候可能并不会出现错误。
- **可表达性与效率** 在很多情况下如果程序设计语言的实现工具能够自动地做一些工作则会带来很大的便利。其中一个示例就是在第 3 章中讨论的内存管理：Lisp 运行时系统使用垃圾收集器来检测程序中无用的存储单元。如果系统能自动完成一些工作，则必然要付出代价。尽管一个自动化的方法能够让程序员不再考虑某些事情，但是语言的执行速度将会慢很多。在某些情况下，自动化方法会使得编写程序变得更容易并且编程不容易出错。在另一些情况下，程序执行速度的减慢是无法容忍的，这时自动化方法就不可行了。

1.3 程序设计语言的历史

在过去的 50 年中，人们已经设计并实现了上百种程序设计语言。这些程序设计语言中的 50 种都包含了新的概念、有用的改进，或者值得一提的创新。这些程序设计语言之多，在这里我们无法一一研究，但是我们将集中于 6 种程序设计语言：Lisp、ML、C、C++、Smalltalk 和 Java。这 6 种语言涵盖了自高级程序语言在 1960 年前后从最初的汇编语言中脱颖而出以来所发明的大多数的重要语言特征。

现代程序设计语言的历史始于大约 1958—1960 年 Algol、Cobol、Fortran 和 Lisp 的发展。本书的大部分都讲的是 Lisp，对于 Algol 以及随后出现的相关语言只有简短的讨论。鉴于有些读者对程序设计语言的史前史感到好奇，在这里我们简单的介绍一些早期的语言。

20 世纪 50 年代，人们开发了一些语言来简化书写一系列计算机命令的工作。用现代的标准来看，那十年中的计算机是相当原始的。大多数的程序是用底层硬件的本地机器语言书写的。由于当时的程序很短小并且效率非常重要，人们还能接受这样的语言。在 50 年代开发的最重要的两种语言是 Fortran 和 Cobol。

Fortran 是在大约 1954—1956 年由 IBM 的 John Backus 所率领的小组开发出来的。Fortran（formular translation 的缩写）主要的创新在于可以在表达式中使用普通的数学符号。例如，将 i 的值加上两倍 j 的值，Fortran 表达式是 i+2*j。在 Fortran 开发出来以前，则需要把 i 的值存放在一个寄存器里面，把 j 的值存放在另一个寄存器里面，把 j 的值乘以 2 之后再与 i 相加。在 Fortran 中，通过对变量使用符号名，程序员能更自然地思考数字计算而把计算顺序等问题留给编译器处理。Fortran 也有子程序（一种程序或函数的形式）、数组、格式化输入输出，以及声明。声明让程序员能够显式地控制变量和数组在内存中的存放位置。然而，这就是它的全部。关于 Fortran 的局限性，这里仅举几例。很多早期的 Fortran 编译器把数字 1，2，3……存放在内存中，这样的话程序员就有可能由于疏忽大意而改变了这些数字的值。另外，Fortran 子程序不能调用其本身，因为支持它的内存管理技术在当