OCaml语言编程基础教程

陈钢 张静 著

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

OCaml语言编程基础教程

陈钢 张静 著

人 民 邮 电 出 版 社

北 京

图书在版编目（CIP）数据

OCaml语言编程基础教程 / 陈钢，张静著. -- 北京 : 人民邮电出版社，2018.6
ISBN 978-7-115-47121-5

Ⅰ. ①O… Ⅱ. ①陈… ②张… Ⅲ. ①程序语言－程序设计－教材 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2018)第003146号

内 容 提 要

OCaml 语言是一种函数式程序设计语言。

本书重点介绍函数式编程的基础知识以及 OCaml 程序设计的技巧，同时兼顾应用软件开发的需求。全书共 8 章，前 5 章讲解 OCaml 语言的函数式控制结构、数据结构、模块化程序设计、命令式程序设计和图形程序设计；第 6 章介绍如何把 OCaml 移植到 F#，第 7 章介绍通过 C#开发的用户界面调用 OCaml 或 F#程序，第 8 章介绍面向对象程序设计。

本书适合想要学习 OCaml 程序语言或者想要学习函数式编程的读者阅读参考。

◆ 著　　陈　钢　张　静
责任编辑　陈冀康
责任印制　马振武

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
固安县铭成印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：20.5
字数：460 千字　　2018 年 6 月第 1 版
印数：1－2 000 册　　2018 年 6 月河北第 1 次印刷

定价：79.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

序

近年函数式语言和函数式编程迎来了一个新热潮，人们用函数式语言开发出越来越多的应用和系统，Scala、Lisp、Erlang、Dylan、Rust、Haskell、Scheme 等语言出现在各种语言使用排行榜中，显示出其重要性和实际价值。在这种情况下，作为计算机专业工作者，学习函数式语言和编程知识也变得越来越重要。

函数式语言与常规过程式语言有同样长的发展历史。在今天还使用较为广泛的语言中，最早的函数式语言是 John McCarthy 于 1958 年设计的 Lisp，其诞生仅比公认的第一个高级语言 Fortran 晚一点。Lisp 之后近 60 年里，人们陆续开发了许多函数式语言，其中产生了长期影响的包括 ML 和 Haskell 等。在另一相关领域，人们针对硬件设计实现的需要，开发了一批用于描述硬件的函数式语言，其中一些已经实际用于开发各种硬件。近年函数式语言的大发展，主要原因包括软件系统复杂性的快速增长，软件安全问题越来越严重，以及多核硬件和并行编程的需要。函数式语言和相关编程技术在这些方面都有本质性的优势。

函数式语言不仅本身具有实用性，还对一般程序语言和软件开发领域的发展产生了重要影响。大多数人可能不了解这方面的情况，实际上，今天常规语言和编程中的大量概念和技术出自函数式语言和相关编程实践。人们在函数式领域开发和检验了大量语言概念、实现技术和编程技术，这些工作在程序设计和软件技术的发展中起到至关重要的作用。早期的例子如动态存储分配，自动存储回收（废料收集），基于栈的语言实现技术，表和表处理，基于链接的数据结构，递归函数定义，尾递归优化，函数的函数参数（高阶函数），有关数据类型的研究和类型理论，变动性（可变对象和不可变对象），数据驱动的程序设计等。近年另一些概念和机制被频繁地融入各种新编程语言或已有语言的新版本，如 lambda 表达式、生成器（generator）、闭包、元编程、自反机制、虚拟机和字节码等。

近年人们开发并流行的许多新语言被称为多范式语言，其设计都参考了一些函数式语言的设计，并以某种方式支持函数式编程。例如，已经比较流行的 Python 和 Ruby 的设计都受到 Lisp 和 Haskell 等函数式语言重要影响；Rust 语言的设计明确说明其语言设计受到 Erlang、Haskell、OCaml、Scheme、Standard ML 等函数式语言的启发；苹果公司开发的新语言 Swift 从 Haskell 等语言中学习了许多语言设计的思想。

函数式编程基于表达式和无副作用的函数，不使用赋值等改变状态的操作命令。实际上，

在各种常规语言中也可以做函数式编程，一些新语言的社团中也在提倡（在某些场景中）使用函数式编程。这就造成一个情况：我们会看到越来越多的实际程序中包含采用函数式程序思想的片段。函数式编程的知识，可能帮助人更好地理解今天和未来的程序。

这些情况说明，对专业计算机工作者而言，有关函数式语言和函数式编程的知识，将越来越多地从学习中的可选项变成必修项。即使不实际地使用函数式语言编程，对这方面相关问题有些了解，也有利于我们开拓视野，理解新语言和语言中的新特征，了解更多的处理问题和解决问题方法，有利于自己的专业发展。

近几年，国内出版了一些关于函数式程序设计的书，但内容上有些偏颇，对重要语言的覆盖也不够平衡。另外，国人撰写的书籍比较少而译作较多。本书在这两方面都对目前情况有所补偿。作为本书主题的 OCaml 是 ML 语言的重要发展分支，是采用静态类型系统的函数式语言的重要代表。该语言不但包含函数式特征，包含命令式特征，还特别支持面向对象编程，支持多种编程范式。人们已经用 OCaml 开发了一些非常重要的系统。OCaml 语言一直处在开发和改进中，官网提供了支持各方面开发的大量程序包，有很强大的用户社团。

本书第一作者陈钢博士多年使用 OCaml 开发各种软件，有丰富的使用经验，对 OCaml 的各方面情况和技术有深入的理解。本书全面介绍了 OCaml 语言和相关的程序开发技术，从最基本的函数式计算结构和数据结构开始，直到各种高级特征的使用，如多语言编程和面向对象的程序开发。本书深入浅出，循序渐进，非常适合初学者从零起步阅读和学习。另一方面，书中不仅讨论了大量语言特征和编程技术问题，也介绍了一些背景和相关理论问题，以帮助读者更清晰地理解函数式编程的思想、技术和方法。本书的出版将大大改善国内计算机工作者学习 OCaml 语言及其编程技术的基础条件。

OCaml 语言是一种非常适合于开发探索性软件系统的语言，其可用程序包中包含了大量在常见语言的库里很少见到的探索性内容。了解相关情况，能拓展我国计算机工作者的眼界，对国内计算机科学技术的发展起到有益的推动作用。

裘宗燕

2018 年 1 月 15 日

前言

2016 年 6 月 14 日，华为宣布在法国设立数学研发中心。华为战略与市场总裁徐文伟在揭幕仪式上指出，法国“菲尔茨奖得主多达 12 位，仅次于美国。数学的研究正在为 ICT 产业带来全新的突破。”实际情况正是如此。法国在计算机科学方面的一项在国际上有重要影响的成果是 OCaml 语言，这一语言的诞生与发展得益于法国雄厚的数学基础。

法国菲尔兹奖获得者大都来自巴黎高等师范学院，或与其有关。这个学校占地面积不如中国一所小学，但它却是法国数理化等基础科学的研究中心。20 世纪八九十年代，巴黎高等师范学院秉承其数学传统，在范畴论的基础上研发了 λ 演算的一个语义模型，称为“范畴抽象机”（Categorical Abstract Machine），此后又在这一模型的基础上研发了函数式语言 ML 的一个新的变种，称为 Caml。之后，法国人又在 Caml 语言的基础上增添了面向对象的机制，形成了 OCaml 语言。后来，OCaml 的研发中心转移到 INRIA（法国国家信息科学研究中心）。

OCaml 语言的诞生和发展都同数学基础密切相关。OCaml 语言出现之后，所进行的一项最著名的工作，就是开发 Coq 定理证明器，它是一个基于高阶带类型 λ 演算的交互式定理证明工具。Coq 的主要用途在于两方面，一方面是用于形式化数学（Formalized Mathematics）的研究工作，就是把数学知识用形式化方式表示，并检查数学证明本身的可靠性；另一方面是进行程序的形式化验证。

OCaml 语言是一种函数式程序设计语言。函数式语言是一群追求数学美的学者所研发的语言。他们对完美性的重视甚于对实用性的重视，这并不等于说他们不看重实用性，只是他们把完美性放在优先的位置。法国是一个特别注重理想主义的国家，这种理想主义品格也深深地注入到了 OCaml 语言当中。为了达到数学上的完美性，就需要有更多的付出，同时也会带来更好的长期回报。

函数式语言的发展经历了一个曲折的过程。其间走过很多弯路，遇到很多障碍，也有很多成功的惊喜。如果你喜欢冒险和挑战，愿意接受前进路上的不确定性，对失败和挫折有承受能力，乐于克服困难并享受解决问题之后的喜悦，你应该选择函数式语言。

函数式语言的历史同计算机语言的历史一样长。最早的函数式语言是 LISP，它出现在 C 语言之前。当然，它远不如 C 语言等命令式语言那样普及。在计算机的发展历史中，人们不断地掀起对函数式语言的热情，但是，在相当长的一段时间内，能够在这个领域中坚持下来

的人为数不多，不过，这种情形近年来正在发生改变。

如果想更具体地了解函数式语言的特点，最好的方式是看一个典型的函数式语言程序，并把它同 C 语言程序做比较。要讲解一个有意义的程序例子，需要先讲解一定的基础知识，这些内容超出了前言的范围。急性子的读者可以翻阅一下 2.5.4 节中的排序函数的例子。它是一个很能说明函数式语言优缺点的案例。我们把用 OCaml 实现的排序函数同 C 语言实现的排序函数做了比较，并且归纳了函数式语言的 3 个优点：

第一，灵活性和通用性。C 排序算法仅仅针对一种特定数据类型的元素进行排序，例如，对整数数组排序。但 OCaml 写出的排序算法是通用的，不但可以对整数序列排序，而且可以对实数序列、字符串序列，以及各种结构化元素构成的序列进行排序。OCaml 能够做到这一点是因为语言中包含了多态类型和高阶函数等成分，这些概念都是 C 语言没有的。

第二，安全性。C 语言编写的排序程序中包含了大量的数组元素访问操作，这类操作很容易出现数组越界错误，从而引起程序崩溃。OCaml 的排序程序使用了表数据结构，表操作不会发生类似数组越界这样的严重错误。因此，OCaml 的排序程序代码比 C 排序代码安全。

第三，简洁性。排序的核心操作是一个 partition 函数。C 代码写的 partition 函数可读性差，而 OCaml 所写的 partition 函数简洁易懂，同算法逻辑的吻合很好，代码行数不足 C 函数的三分之一。

具有上述优点的函数式语言 20 多年前已经存在。今天的函数式语言具备更丰富的特性，例如，模块化程序设计能力、复杂的类型推导系统等。然而，函数式语言的普及和推广至今还没有完成。这是因为，影响一个语言的推广应用有很多因素，例如，是否有功能丰富的函数库，是否有足够多的成功案例，是否易学易懂，等等。

一个语言的典型应用对语言的普及有很大的影响。C 语言被成功用于开发 UNIX 操作系统，随后，围绕 UNIX 进行的系统程序开发和应用程序开发都在 C 语言之上进行。OCaml 语言的首个重要应用是 Coq 定理证明器的开发。同操作系统相比，定理证明器的用户要少得多，而且这些用户并不一定需要用 OCaml 编程。所以，OCaml 语言的推广发展远不如 C 语言那么快。在很长一段时间内，OCaml 的主要用户是一些大学和研究单位，它被用于开发新的定理证明器及其他研究性项目。

同 C 语言相比，OCaml 的发展比较缓慢，但是它一直在稳步前进，用户群也在不断扩大。在现有的函数式语言当中，OCaml 是用户最多的语言之一。在应用软件开发中，OCaml 的知名度也在不断提高。

OCaml 是否容易学习？这个问题因人而异。网上有一位清华学生说，他（她）跟法国 OCaml 专家学了一段时间，发现学 OCaml 比零基础学 C 语言难多了。但是也有同学说，他们一周时间就学会了 OCaml。一般而言，一些数学基础好的人比较容易适应 OCaml。而对 C 语言经验丰富的程序员已经养成了一种编程习惯，他们可能反而觉得 OCaml 难学。因此，学好 OCaml 要注重培养新的编程习惯，这也是本书写作的重点。

OCaml 语言是 λ 演算理论（包括无类型 λ 演算和带类型 λ 演算）的一次出色的应用，同时也是学习λ演算以及基于 λ 演算的众多理论和技术的入门课程。建议有条件的读者把 OCaml 语言同 λ 演算两门课程结合在一起学习。本书也讲解了一些 λ 演算的基本概念。

事实上，在法国，OCaml 语言是学习一系列后续课程的基础课程。这些后续课程包括 λ 演算、类型理论、重写系统、形式语义、程序语言实现方法、高阶定理证明器、进程演算、同步语言、程序分析等。

在互联网时代，能否防范黑客攻击成了软件质量的一个重要指标。在这方面，OCaml 语言的优势进一步显现出来。黑客攻击的主要方法是利用软件本身的缺陷入侵到软件内部，而 OCaml 语言编写的程序可以避免很多 C 程序中的缺陷，例如缓存溢出和存储泄漏，因此也能够更好地抵抗黑客的攻击。

近 20 年来各种新型语言。例如 Javascript、Go、Scala、Groovy、Elixir、Clojure、Ruby、F# 等层出不穷。几乎所有的新语言都深受函数式语言的影响，一些新语言本身就是函数式语言。这些新语言在一定程度上是面向应用的函数式语言，它们在函数式程序设计概念的基础上添加了针对各种专门应用的语言成分。OCaml 语言出现在这些新语言之前，它的主要历史功绩是推动了各种函数式程序概念和技术的发展，尤其是语言中的类型系统的发展。它是函数式语言领域做出重要创新的先驱之一，很多新语言中的函数式概念都来自 OCaml 以及其他一些专业性的函数式语言。因此，对于系统性地学习函数式编程技术，认真学习 OCaml 语言是很有帮助的。

F#是微软公司开发的 OCaml 语言变体。它在 OCaml 基础上添加了大量的具有实用价值的功能，使用户能够利用 Windows 平台中丰富的库函数，为普通用户提供了很多方便。但是，从语言特性角度看，F#同 OCaml 依然有差距，例如在模块和函子方面，F#的能力远不如 OCaml 强大。因此，如果想在函数式程序设计方面得到充分的训练，F#不能替代 OCaml。尽管如此，考虑到 F#对应用开发的重要性，本书中拿出了一定的篇幅讲 OCaml 程序怎样移植到 F#。

在动手写这本书的时候，国内 OCaml 教材只有一本《Real World OCaml》【4】的译著，该书对有经验的 OCaml 程序员很有帮助，但不太适合初学者，在基础训练方面做得不够。在英文教材中，可以考虑的教材有【1，2，3，4，5】。【1，2】是经典的 OCaml 著作，有权威性，但写作时间比较早，有些程序在最新的 OCaml 解释器上不能运行，OCaml 的一些新发展也未能包含进去；【3】主要适合有经验的 OCaml 程序员；【5】是专门给初学者写的书，没有包括诸如函子（functor）和面向对象等重要内容。其他一些教材主要讲 OCaml 语言在某个专业领域的应用，因此并不很适合教学。因此，我们想写一本能够包括 OCaml 的主要特征，同时能够提供函数式编程基本训练的教材。

教材【1】为函数式语言基本技能训练提供了很好的材料。征得原书作者 Guy Cousineau 和 Micheal Mauny 的同意，本书第 1 章和第 2 章部分内容翻译自教材【1】。在此对原书的作者表示深切的谢意。同时，我们也增补了很多近年来的新内容。针对国内初学者的特点，本书对函数式编程各方面的概念做了进一步的解释，并增加了 OCaml 语言同命令式语言的对比。

本书的重点在于讲解函数式编程的基础知识以及 OCaml 程序设计的技巧，同时兼顾应用软件开发的需求。在写作过程中，很多地方我们把 OCaml 编程方式同其他语言的编程方式进行比较，便于熟悉其他语言的程序员理解 OCaml 的特点。我们将看到，同 C 语言相比，OCaml 程序更加精巧，便于进行程序分析；同无类型的函数式语言 LISP 相比，OCaml 增加了类型推导机制，提高了程序的安全性。

本书第 1～5 章及第 8 章全部是关于 OCaml 本身的内容。由于 OCaml 在用户界面开发等方面不够理想，影响了 OCaml 的普及。因此第 6 章和第 7 章讲了如何通过引入 F#和 C#来补充 OCaml 在这方面的不足。F#是微软在 OCaml 语言基础上开发的一种语言，它同 OCaml 部分兼容，同时又能利用微软的大量库函数。第 6 章讲了如何把 OCaml 移植到 F#。第 7 章讲了怎样通过 C#开发的用户界面调用 OCaml 或 F#程序，这个内容也是多语言联合软件开发的一个案例。现代实用软件的开发往往需要结合多种语言的优势。第 8 章讲了面向对象程序设计。

本书一方面介绍了 OCaml 程序设计的基本概念以及函数式编程的基本方法，同时尽作者所能介绍了 OCaml 语言的最新发展，此外还提供了一组规模适度的软件开发案例。3.7 节提供了一个质数生成模块案例；4.10 节包含了一个四方向链表案例；第 5 章结合基于模块的开发方法介绍了一个电机作图的案例；第 6 章结合对 F#的介绍讲解了基于 F#的电机作图的案例；第 7 章结合多语言混合设计继续用电机作图的图形化用户界面的设计；8.15 节用面向对象方式实现电机作图。

由于作者水平有限，书中错误在所难免，欢迎读者及时指正。

陈钢

航天科工集团

北京京航计算通讯研究所

2018 年 1 月

致谢

感谢我的博士导师 Guissepe Longo 和 Guissepe Castagna。他们的帮助和指导使我不但获得了在法国学习的宝贵机会，而且能够进入程序语言设计这一激动人心的领域。他们在学业上给了我很大的帮助，尤其是在范畴理论和面向对象的程序语言的类型理论方面，使我学到很多重要的理论，他们的帮助使我能够顺利完成博士学位。感谢给我讲授 OCaml 语言的老师：Guy Cousineau 和 Micheal Mauny。本书前两章部分内容翻译自他们所写的 Caml 语言教材。他们是 Caml 语言的主要发明人，Caml 语言是 OCaml 的前身。感谢给我们讲授λ演算的老师 Thérèse Hardin，λ演算是函数式语言的基础；感谢 OCaml 语言类型理论的老师 Xavier Leroy 和 Didier Rémy，他们也是 OCaml 语言的主要开发人员；感谢讲授形式化语义的 Roberto Di Cosmo 老师；感谢讲述构造演算和高阶类型理论的老师 Gilles Dowek，构造演算是 COQ 定理证明器的基础理论，COQ 也是用 OCaml 开发的一个重要的软件。

本书是我在北京京航计算通讯研究所工作期间完成的。感谢集团高红卫等领导对我的工作的鼓励和支持。感谢老所长李艳志和所领导王俊、于林宇、于会、刘军、张津荣、郑德利对我的工作的支持和帮助；感谢舒毅、张静、占银玉，他们在担任我的助手期间做了很多重要的工作。张静在实习期间开始翻译 Guy Cousineau 和 Micheal Mauny 的 OCaml 教材，并且和我一起开始着手这本书的撰写，实习结束后继续花费大量的时间进行本书的撰写和修改工作，非常认真仔细，并且从读者角度提出许多有益的看法，在排版和美化方面做了主要的工作。感谢所内各部门领导李昆、朱琳、王颖、刘伟、王家安、韩旭东、李娜、宋文、魏伟波、魏鑫在我的工作和生活上多方面的帮助；感谢所内同事孟伟、吕宗辉、郑金燕、于润泽、张志刚、王栋、杨楠、张国宇、张明敏、李卓、李丽华、刁立峰、彭鸣、姚可成、高飞、赵静、王佳佳、黄云、宋悦、刘玉峰、李思等在日常工作中的协助。感谢李芳、焦留、杨杰、房静在后勤方面的支持。

感谢裘宗燕教授为本书撰写序言并且提出宝贵意见。感谢孙家广、宋晓宇、蒋颖、顾明、王戟、杨志斌教授，在同他们的合作中，我进一步了解了国内对 OCaml 语言的需求。

感谢我的妻子胡萍，长期以来她一如既往地支持我的研究工作，在生活和精神上都给了我很大的支持。

陈钢

航天科工集团

北京京航计算通讯研究所

2018 年 1 月 16 日

目录

第1章

函数式控制结构

OCaml 支持函数式、命令式以及面向对象程序设计。但是它的主要特点是支持函数式程序设计。

在函数式程序设计提出了“纯函数”的概念指的是只做输入输出变换，没有副作用的函数。为了说明这一概念，我们先来看两个 C 函数的例子，第一个：

```
int f (int a) { return (a+1); }
```

这个函数的输入值是一个整数 *a*，输出值是 *a*+1。这个函数实现了输入 *a* 到输出 *a*+1 的变换，除此之外没有其他功能。所以，这个函数是“函数式程序设计”意义上的函数。再看第二个例子：

```
int g (int a) { b = a; return (a+1); }
```

这个函数同样把输入的整数 *a* 加一之后输出。但是，它还对一个全局变量 *b* 进行了赋值，这个操作就称为函数的副作用。因此，*g* 就不是纯函数式程序设计意义下的函数。

函数式程序设计风格的一个重要优点是提高了程序的可靠性和可理解性。它的一个缺点是有时会降低程序的执行效率。

函数式语言的基本控制结构主要有函数定义和函数调用。

核心 OCaml 程序主要由 let 定义（definition）和表达式（expression）组成。和 C 语言相比，OCaml 的核心部分严格地说没有“语句”概念。OCaml 表达式既起到 C 语言中表达式的作用，又起到语句的作用。OCaml 中的定义类似于 C 语言中变量声明和函数定义的结合体，它本质上是把一个名字和一个表达式相关联。

表达式的特点就是能够通过计算得到一个值（value）。值是无法继续计算的表达式，函数也是一种值。值都有类型，因此表达式都有类型。OCaml 的 let 定义把变量声明、变量初始化和函数定义等概念整合在一起。本章主要讲 OCaml 中的表达式以及基本的 let 定义。表达式和 let 定义是 OCaml 解释器能够执行的基本单元。

1.1 OCaml 解释器

C 语言是基于编译器的语言，而 OCaml 是基于解释器的语言。对于一个基于编译器的语言，必须写出一个完整的程序才能编译执行。而对于一个基于解释器的语言，程序可以划分为一组可独立执行的代码，解释器可对这组代码顺序执行。基本的 OCaml 程序由一组定义和一组表达式构成，这些定义和表达式可以在 OCaml 解释器中按顺序单独执行。OCaml 程序也可以编译执行，实用的 OCaml 应用开发都需要编译。在学习 OCaml 的时候，我们首先从解释执行开始。程序的解释执行使得语言的学习更为方便。在 OCaml 解释器中执行 OCaml 定义和表达式的过程称为交互式会话（interactive session）。

OCaml 是一个免费的软件，可以从 ocaml.org 上下载。在安装了 OCaml 软件的 Linux 或 Cygwin 中，可以通过 OCaml 命令启动 OCaml 解释器。在 Windows 平台上，过去有一个 OCamlWin 窗口程序，可以在窗口界面中运行 OCaml 解释器。近几年 OCaml 的发布方式经常有变化，带窗口界面的 OCaml 软件不太容易找到。OCaml 官方下载页面上现在可以看到三种 Window 平台下的 OCaml 安装方案，推荐使用最后一个，即下载 OCaml64.exe 程序，该程序执行之后会创建一个目录，缺省情况下的目录名是 C:\OCaml64，其中包含一个 cygwin 工作环境。通过在桌面上新创建的 OCaml64 图标可以启动这个工作环境，它是一个类似 linux 终端的命令行操作界面。在这个界面中可以执行 linux 命令，并且可以执行一个 opam 程序，它是一个用于安装 OCaml 及相关软件的专用工具。执行 opam init 将对这个工具做初始化，同时下载并安装最新的 OCaml 软件。安装完毕之后，可以键入 ocaml 命令启动 OCaml 解释系统，如图 1-1 所示：

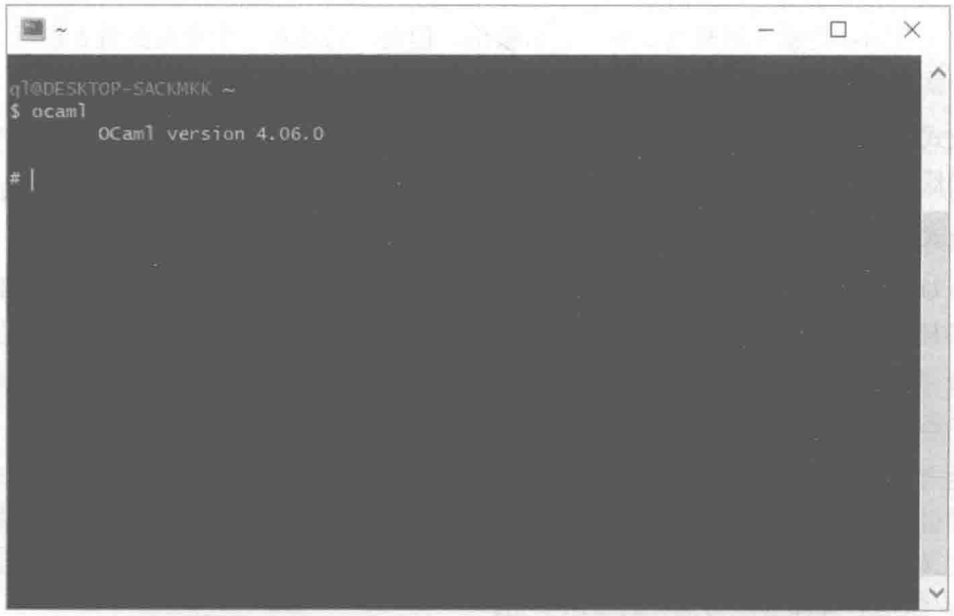

图 1-1

解释执行 OCaml 比较好的方式是在 XEmacs 中安装 Tuareg 模式，这样不仅可以得到一个支持 OCaml 编辑的 XEmacs 模式，而且可以在编辑器中直接运行 OCaml 解释器。如果没有在本地机器中安装 OCaml 软件，也可以在网上直接使用在线的 OCaml 解释器。网站 http://try.ocamlpro.com/上不仅提供了可执行 OCaml 的在线解释器 Try OCaml，而且还提供了一些教学材料。

进入 OCaml 解释器之后就可以进行交互式会话。解释器会显示一个提示符“#”，等待用户输入。此时用户可以输入一个 OCaml 表达式，用“;;”结尾，按<Enter>键输入到系统中。如果表达式语法正确，解释器会做出一个回应（response），它包括表达式的计算结果，以及对表达式的类型分析结果，前者称为表达式的“值”（value），后者称为表达式的类型（type）。下面是交互式会话的一个例子：

```
# "Hello World!" ;;
- : string = "Hello World!"
```

很多语言的教学都从一个 Hello World 程序开始，这样的程序代表了一个语言中有代表性的最简单的程序。对于 OCaml 语言，最简单的程序就是"Hello World!"字符串。

在这个例子中，系统的回应分为两部分，第一部分是“- : string”，它表示用户输入表达式的类型是字符串。第二部分是“= "Hello World!"”，它表示输入表达式的计算结果是等号右边的"Hello World!"。

在这个简单的例子中，表达式的值（表达式的计算结果）就是表达式本身，即一个字符串。值得注意的是，系统自动分析出了表达式的类型“string”。

OCaml 的类型分析被称为“类型推导”（type inference）或“类型合成”（type synthesis）。它是指通过对表达式的分析自动推导出表达式的类型。这一分析工作是在计算表达式之前做的，因此 OCaml 的类型系统是静态类型（statically typing）系统。OCaml 是强类型（strongly typing）语言，它对代码进行严格的类型检查，保证了程序中不会出现类型错误。在其他语言中，需要对变量和函数进行显式的类型说明，而在 OCaml 语言中，变量和函数的定义可以不包含类型说明，系统自动推导出它们的类型。类型推导是 OCaml 语言的重要特色，对此本书会进行重点讲解。

1.2 表达式和 let 定义

OCaml 语言入门很容易，可以把它当作计算器，进行算术表达式的计算。

```
# 2 + 3 * 5 ;;
- : int = 17
```

在这个回应中，符号“-”表示这个表达式是由用户输入的，“int”是表达式的类型，它说明这是一个具有整型值的表达式，“17”是表达式的值。

如果想把表达式的计算结果保存起来，并在后续计算中使用，可以通过 let 结构把表达式的计算结果保存在一个变量中，例如：

```
# let a = 123 * 456 ;;
val a : int = 56088
```

这个 let 结构也称为 let 定义，或简称定义。它的语法格式是：

```
let  <变量>  =  <表达式>
```

其中，变量是一个由小写字母或下划线开头的，由大小写字母、数字和下划线构成的标识符。

对于 let 定义，系统的回应以“val”开始，表示标识符 *a* 是一个变量，用来保存一个值。上面的回应说明，*a* 的类型是 int，即整数类型，*a* 的值是 56088。

在这里，我们再一次看到了 OCaml 语言的类型推导能力。在 let 声明中，并没有说明 *a* 的类型，只说明了 *a* 将保存表达式 123 * 456 的计算结果。系统对表达式进行类型分析，推导出这个表达式的类型是整型，从而确定 *a* 的类型为整型。

在建立 *a* 的定义之后，后续的表达式和定义就可以引用这个变量。例如：

```
# a + 4 ;;
- a : int = 56092

# let b = a + 5 ;;
val b : int = 56089
```

在计算一个表达式的时候，表达式内的变量必须在之前已经定义过，不然会产生变量无定义的错误。例如：

```
# x + 4 ;;
Characters 0-1:
  x + 4 ;;
  ^
Error: Unbound value x
```

let 定义起到了其他语言中变量声明的作用，只是它不需要进行变量的类型声明，但是必须给变量一个“初始值”。不允许只声明变量而不给“初始值”。这样做的一个好处就是避免了忘记给变量赋初值的问题。

但是，这里的变量定义和命令式语言中的变量声明是不同的。例如，C 语言中，在一个函数体内变量不能重复声明。下面的程序会导致编译错误：

```
void main () {
  int i = 1;
  int i = 2;
}
```

但是在 OCaml 语言中，一个变量可以合法地重复定义：

```
# let i = 1 ;;
val i : int = 1
# let i = 2 ;;
val i : int = 2
```

在这里，let 定义看上去和赋值语句相似，但实际上它和赋值过程不一样。在使用赋值语句