Common LISP 程序设计

韩俊刚　殷　勇　　编著

西安电子科技大学出版社

序　言

长期以来，人们把 LISP 语言称为人工智能语言，似乎只有研究人工智能才会使用它。实际上，只要你想用计算机解决问题，就可以使用 LISP 语言，它同其它的程序设计语言，例如 FORTRAN、C 一样，可以应用于广泛的领域，是一种通用的程序设计语言。近年来国外一些大学甚至把 LISP 语言做为语言的入门课程，或者做为必修课程，一些非计算机专业，包括社会科学的某些专业也开设了 LISP 语言。可以说，LISP 这个具有三十多年历史的老语言再次焕发了青春。

LISP 固然是人工智能研究中使用最广泛的语言。随着人工智能技术的广泛应用和实用化，LISP 已成为越来越多的人手中的工具。LISP 语言经过多年的发展，其优点日益明显。LISP 语言的实现有了重大的改进，编程环境和用户界面不断完善。LISP 语言简单易学，趣味性强，编程效率高，调试方便。权威人士估计，它在今后几十年中仍将保持勃勃生机。

由于 LISP 语言一直没有标准化，各种实现版本都被称为“方言”。1984 年前后，一批世界闻名的程序设计语言的专家在美国主要名牌大学和关键的研究机构的支持下，设计出通用的 LISP 语言——Common LISP。近年来的发展趋势表明，Common LISP 将成为 LISP 语言的标准。

国内对 LISP 感兴趣的读者也在日益增加。IBM PC 微机上的 GCLISP、auto LISP 和 XLISP 都是 Common LISP 的子集，它们可以做为学习 LISP 的有力工具。各种小型机和工作站上的 Common LISP 已为许多人所熟知。但是至今国内还没有一本系统地介绍 Common LISP 的教材。国内有些大学把 LISP 和 Prolog 做为一门课开设。根据作者的经验，学生在短期内学习这两种风格迥然不同的语言，很难“转过弯儿来”。因此专门开设 LISP 语言课很有必要。事实上，已有半数以上的高校已经这样做了。本书的目的之一就是为大学本科和研究生提供 LISP 语言的教材。

由于 LISP 语言的语法极其简单，语义清晰易懂，因而适于做为计算机语言的入门课。本书的写法尽量通俗易懂，以便使没有计算机专业基础知识的读者也能轻松地阅读本书的基本内容。因此本书也可以作为非计算机专业的 LISP 语言教材。本书还希望成为计算机业余爱好者的朋友，作者和本书出版部门可向读者提供 XLISP 做为做为学习的工具。

为了达到上述目的，本书分为两部分。第一部分介绍 Common LISP 的基本内容，包括 LISP 的基本函数、用户自定义函数、迭代、递归，输入输出函数、程序的调试、数组、结构、Lambda 定义和宏定义等，并介绍一些简单的 LISP 程序，使读者掌握基本的 LISP 程序设计技术，为进一步学习打下基础。学习这部分内容不需要其它计算机语言的知识和人工智能的知识。本书的第二部分则涉及 LISP 语言和程序设计较为深入的内容，包括 LISP 在博弈、模式匹配等人工智能领域的 LISP 应用实例，和在系统程序设计、面向对象的程序设计

中的应用。对于计算机专业的研究生和有经验的 LISP 程序员也有参考价值。因此本书的第一部分可以做为 Common LISP 的入门教材，而全书可做为具有中等深度的 LISP 程序设计教材。第一部分可以在 20～30 学时内讲授完毕，讲授全书则需 30～40 学时。

为了教学方便，本书备有习题。对于自学的读者应完成习题后再阅读后面的内容。本书的习题答案，大部分可以用 IBM－PC 机上的 GCLISP 和 XLISP 来验证。

目　录

第一部分

基础篇

第 1 章 绪 论

本书的内容是讲述 Common LISP 程序设计。Common LISP 是各种 LISP 方言中最通用的一种，它正在成为 LISP 语言的标准。本章介绍 LISP 语言的历史、特点和应用领域。

§1.1 LISP 语言的历史

在各种高级程序设计语言中，FORTRAN 和 LISP 是历史最长的两种语言。LISP 语言的名字来源 LISt Processing，因为它最初被看作是表处理语言。50 年代末，美国麻省理工学院（MIT）的 John McCarthy 和他的学生首先设计并实现了第一个 LISP 系统——LISP1.5。这在程序设计语言发展史上是一个重要的里程碑，因为它标志着与传统的程序设计语言很不相同的函数型程序设计语言的出现。早期的 LISP 语言被认为是有趣而不实际的语言，效率很差。第一个高性能的 LISP 实现是 PDP10 上的 Mac LISP，同时出现的有 Inter LISP，它们成为 70 年代两个最重要的 LISP“方言”。为了为 LISP 语言提供专用的硬件支持，MIT 和 Xerox 公司的 PARC 研究中心研制了 LISP 机，它的以栈为基础的系统结构及指令系统大大提高了 LISP 语言的效率，70 年代末和 80 年代初，LISP 机成为商品化的专门运行 LISP 语言的计算机。它所提供的功能很强的程序开发环境至今仍是最好的程序开发环境。70 年代中期出现的 LISP“方言”还有 SCHEME，它具有简洁、优美的特色，至今仍然影响很广泛。

为了解决 LISP 程序在不同处理器上的可移植问题，Utah 大学研制了 Portable Standard LISP（PSL），但它与 Mac LISP 和 Inter LISP 都不兼容。随着 VAX 系列机和 Unix 操作系统的广泛使用，加州大学 Berkeley 分校的 Franz LISP 成为一个重要的 LISP“方言”。

80 年代初，LISP 的应用迅速发展。大约有十几种 LISP“方言”。每种方言对 LISP 语言的解释与其实现有关，而做为 LISP 语言还没有统一的定义。因此 LISP 标准化问题十分迫切。美国一些名牌大学和工业界的重要单位的专家经过两年的共同努力，在 1984 年推出了 Common LISP。G. L. Steele 所著《COMMON LISP：the Language》一书的问世，说明 LISP 已经有了语言的定义。至此，各种 Common LISP 的实现就有了统一的依据。Common LISP 在大学和工业界均得到广泛的支持，因为它包括了各种重要的 LISP 方言的优点，具有丰富的功能和数据类型。近几年国外的教科书和软件公司都纷纷转向 Common LISP，大部分人对“Common LISP 将成为 LISP 的标准”表示赞同。

Common LISP 已在许多小型机和工作站上实现。在 IBM PC 机上也有其子集 GCLISP。auto CAD 中的 auto LISP 也是 Common LISP 的子集。在 286 以上的微机上的 Common LISP 全集也已经实现。90 年代的 LISP 语言将会在通用计算机上提高效率，达到 LISP 机器上的已有的水平。本书讲述 Common LISP 的基本内容，尽量使用在 GCLISP 中已实现的函数，因而书中的实例可以在各种 Common LISP 包括 GCLISP 系统上运行。因为 Common LISP 包括了许多重要的 LISP 方言的功能，所以，本书可用来在 Mac LISP 系统上学习 LISP 语言。XLISP 是 PC 机上的 Common LISP 的子集，并扩展了面向对象的成分，读者也可用 XLISP 做为学习的工具。它是一个公共软件，我们可以免费提供。

§ 1.2 LISP 语言的特点

LISP 语言历经 30 多年历史而不衰落，这是因为它有其独特的优点。

LISP 最突出的特点是它的函数性。最初的 LISP 语言是一种纯函数型语言。J. McCarthy 发明 LISP 的出发点是当时已有二三十年历史的递归函数论。函数型语言的特点是用函数定义和函数调用构成程序。程序员用函数定义和函数调用组成的表达式描述求解问题的算法，而表达式的值就是问题的解。传统的程序设计语言如 FORTAN、PASCAL 和 C 等都属于强制型或命令式（Imperative）语言。用这些语言编出的程序表达了按一定顺序执行的一系列命令，执行结果就是问题的解。用这些语言编程时，程序员要规定求解的顺序，即描述控制流。LISP 语言的编程则不考虑程序执行顺序的细节，因而提高了编程效率并减少了出错的可能性。LISP 是更加面向用户的语言，它把函数之间的调用关系、程序执行的细节交给 LISP 系统，减轻了用户的负担。如果把传统的语言叫做冯·诺依曼式的语言（因为它们以冯·诺依曼式的计算机系统结构为出发点），那么 LISP 语言则是第一个非冯·诺依曼式的语言。随着计算机硬件性能的提高和计算机系统结构的发展，函数型语言有着越来越广阔的前途。但是在当前的计算机系统结构仍未摆脱冯·诺依曼式的结构的情况下，LISP 的效率确实要低一些。为了适应当前的硬件发展水平和使用传统的程序语言的程序员的习惯，LISP 语言后来加入了许多非函数型的语言成分，例如 Prog、go 等函数，因而现在的 Common LISP 语言已不是纯函数型语言，它既包括了函数语言的成分，又具有传统式语言的功能。本书将鼓励读者充分利用函数式程序设计特点，因为这是 LISP 语言的现代编程方式。

LISP 语言的第二个特点是它的递归性。定义一个递归函数就是在函数的定义中调用函数本身。递归函数论证明了所有的可计算函数都可以用递归函数定义，因而这是一种通用的重复计算的方法。例如阶乘函数可定义为

$$n! = \begin{cases} 1 & \text{当 } n = 0 \text{ 时} \\ n * (n-1)! & \text{当 } n > 0 \text{ 时} \end{cases}$$

其中 n! 用 (n−1)! 来定义，因而这是一个递归定义。LISP 的主要数据结构——表，也是用递归方法定义的。递归的方法具有简明、优美的特点。本书鼓励读者充分利用递归程序设计方法。

LISP 的另一特点是数据与程序的一致性。LISP 的一段程序可以做为另一段程序的数据进行处理，而程序的执行结果也可以是一段 LISP 程序。这一特点使 LISP 可以定义宏函数，从而提高了程序的效率和可读性，并使 LISP 可以对 LISP 程序进行编译和解释。这也使 LISP

可以用来方便地构成用户自己领域的专用语言。

LISP 还有自动进行存贮分配的特点。数、函数、表等都能按程序要求自动生成，对不再需要的数据，LISP 自动释放其占用的存贮区，通过无用单元搜集程序自动把它们变成可再利用的存贮区。这比 C 语言等是优越的，程序员完全不必考虑存贮分配问题。

LISP 经过多年的发展，形成了自己成熟的程序设计环境。在这个环境下，程序员可以很方便地编辑、修改、调试自己的程序。当前，LISP 机上的无与伦比的程序设计环境在 90 年代将也会在通用机器上出现。

另外，LISP 的语法极其简单，不需要变量和类型说明，可进行动态类型检查。所有上述特点都将促使越来越多的程序员使用 LISP 语言。

由于早期 LISP 实现的一些缺点，对 LISP 语言至今还存在一些误解。其中有一种认为 LISP 只能处理表、整数和符号。事实上，Common LISP 提供了广泛的数据类型，包括浮点数、复数、数组、结构、序列等，用户还可容易地用 LISP 产生新的数据类型，使之与原有的数据类型和运算有机地结合在一起。

还有一种误解认为 LISP 不能做数值运算。其实，好的 LISP 实现系统早已克服了这方面的问题，有些 LISP 系统的数值运算的能力不亚于 FORTRAN 或 C。LISP 的函数处理能力强，构成新的程序很方便，这就使之很善长于建造复杂的数值算法库。

就执行速度而言，LISP 比其它某些语言如 C 要慢。LISP 的动态存贮分配和动态的类型检查使得 LISP 很难在通用的计算机上高速运行。由于早期的 LISP 系统很不讲究实现的效率，使人们夸大了 LISP 的这一缺点。随着高质量的 LISP 系统的出现和硬件水平的急剧提高，LISP 在这方面的弱点正在克服，也可以说在很大程度上已经克服。

§1.3 LISP 语言的应用

自从 LISP 诞生以来，它最重要的应用是在人工智能领域。研究人工智能的人们用计算机表达和验证他们的想法，LISP 语言是他们不可缺少的工具。故此，人们称 LISP 为人工智能语言。但是现在这种提法已经不恰当了，因为 LISP 已经发展成通用的程序设计语言，大大超出了人工智能的范围。我们只能说人工智能的主要语言是 LISP，而不能把 LISP 限制成人工智能语言。事实上，目前 LISP 语言应用最多的领域是人工智能和计算机辅助设计。在系统程序设计、程序验证、文字处理、计算机科学教育等方面都有广泛的应用。

程序设计语言是用来解决问题的工具。对于要解决的问题，用程序设计语言表达清楚，并指导计算机去解决。传统的程序设计语言要求程序员把问题和解决问题的繁琐的细节毫无遗漏地描述出来，然后才能写出可工作的程序。这对以后程序的改进非常麻烦。LISP 则要灵活得多。例如变量的类型可以不必说明、函数的自变量个数或叫参数也可以不限定。LISP 开发系统允许我们随时地程序进行修改、测试和纠错，从而边试验边设计，很容易构成复杂的系统。因此 LISP 最适合于建立复杂系统的快速原型，验证人们的思想和解决问题的方法，探索解决难题的途径。因而 LISP 大多用于非常有吸引力的研究领域。下面简述一些应用。

·计算机视觉 用计算机模仿人的视觉是非常困难的，其中涉及很大的计算量。现在这个领域已有了显著的进展，许多工作都是借助于 LISP 完成的。计算机视觉的研究需要很

多数值计算，因而这也成为 LISP 发展自己的数值计算能力的动力。

·自然语言理解 用英语或其它语言直接与计算机打交道当然是不容易办到的。但在某些限制的领域内，已经建立了一些实际的系统。LISP 灵活的数据结构和可扩展的开发环境使它成为这个领域内流行的语言。

·专家系统 最早的一个 LISP 程序就是用来解决微积分问题，其能力达到大学一、二年级学生的水平。后来的专家系统用于疾病诊断、测井数据解释、电子线路分析等，这些大都是用 LISP 实现的。

·计算机辅助设计 许多机械和电子产品辅助设计系统是用 LISP 实现的。还有不少系统具有与 LISP 语言的接口，允许用户使用 LISP 来处理系统中的对象。例如 auto CAD 系统可以用它自己的 Auto LISP 来给用户提供编程工具。现在已有许多模拟程序是用 LISP 实现的。

·系统程序设计 专用的 LISP 机用硬件实现 LISP 的基本操作，而它所有的软件都是用 LISP 写的，包括设备驱动程序和窗口系统、操作系统、编译程序和各种实用程序。其它语言例如 Ada 的编译程序也有用 LISP 实现的。用 LISP 编写系统程序可以大大节省程序开发费用。

·计算机科学教育 LISP 已经成为流行的计算机程序设计语言的入门教学语言。由于它的语法简单、交互性好，有利于对程序设计概念的理解。它的处理函数的能力对引入程序设计思想和培养良好的程序设计风格都十分方便，这是别的语言所无法代替的。

另外，在符号代数、定理证明、程序验证、机器人规划、文字处理等许多领域都广泛地使用着 LISP 语言。专家们预计，在今后几十年内，LISP 语言不仅不会衰落，还会更加完善和成熟。它将进一步提高在通用计算机上的效率；加强与 C 语言联编的能力；基于 LISP 的面向对象的程序设计系统也将不断完善。在对复杂系统进行快速原型开发方面，LISP 的优势也必将得到进一步的发挥。

第 2 章　LISP 的基本函数

本章介绍 LISP 语言中最常用的基本函数，包括数值计算和表处理函数。同时介绍 LISP 的解释程序，使读者尽快学会与解释程序对话，从而理解 LISP 的语法和符号表达式。通过本章的学习，读者可以利用 LISP 的基本函数进行各种数值运算和简单的符号处理。

§ 2.1　LISP 的解释程序与算术运算

学习 LISP 语言同学习其它语言一样，不应只在其语法条文上下功夫，而是应该大胆地接触有趣的程序，通过大量的练习增加对语言的感性知识。我们建议读者尽快与 LISP 的核心——解释程序打交道。如果你的计算机中已安装了 Common LISP 或其它 LISP 系统，在操作系统提示符下打入 lisp，（在 PC 机上安装了 GCLISP 或 XLISP 时打入 GCLISP 或 XLISP），这时屏幕上出现一些信息显示 LISP 的版权、版本信息等，然后出现 LISP 的提示符。我们以后假定提示符为＞　（GCLISP 的提示符为 * ）。提示符的出现表明 LISP 的解释程序已装入内存，并准备好接受用户的命令，这时我们说 LISP 处于顶层(Top - level)。如果在顶层打入(exit)就退出 LISP 回到操作系统。下面是与 LISP 解释程序的一段对话：

```
> (+ 25 2.86)
27.86
>
```

我们在 LISP 的提示符后面打入命令(+ 25 2.86)，它是一个 LISP 表达式。打入这个表达式之后，有的系统需要一个回车，有的则不需要。LISP 解释程序马上对这个表达式进行计算，用 LISP 的惯用说法是对表达式求值(evaluate)，求值的结果立刻显示出来，即上面的求和结果为 27.86。然后又重新出现 LISP 的提示符，等待用户的其它命令。这个过程叫读入－求值－打印循环(read—eval—print loop)。下面是另外两个循环过程：

```
> (* 12 8 2)
192
> (- 15 3)
12
>
```

读者可能已经发现 LISP 表达式的语法。(* 12 8 2)是一个合法的 LISP 符号表达式(symbol-expression)。它在括号内的第一个元素 * 表示乘法运算符，其余元素为运算数。这种前缀表示看起来不自然，实际上好处很多。比如 * 12 8 2 比中缀表示 12 * 8 * 2 就省了一个 * 号。现在看看 LISP 解释程序如何处理符号表达式。当它读入(* 12 8 2)后，由最前面的左括号知道这是一个表(任何由括号括起来的式子都叫表，表中的元素用空格分开)。由表的第一个元素 * 知道要进行乘法运算，* 后面的元素是乘数，或者说是要计算乘法函数的值，* 是函数名，12、8、2 是函数的三个实参(自变量的值)。注意，LISP 解释程序在计算乘法函数值之前，先对其各个实参求值，即对 12、8、2 三个数分别求值，LISP 规定对数求值的结果还是数本身，然后再把这三个数乘起来得到结果 192。一般情况下，LISP 解释程序对输入的表做如下的工作：

1. 把表的第一个元素(符号)做为函数名，到内部符号表中查找这个函数是否已定义过。如果没有找到，则显示错误信息，告诉用户这个函数没有定义。

2. 若找到函数的定义，则检查定义中的变元个数是否与表后面的实参个数相符。不相符时，给出错误信息。否则对函数的各实参自左向右分别求值。

3. 把上面求值的结果做为函数自变量的值，进行函数值的计算，并回送计算结果。

为什么在计算函数值之前要对各个实参进行求值呢？这是因为这些实参本身又可能是另外的符号表达式或函数调用。例如计算(*(* 2 6) 8 2)时，外面的乘法函数的第一个实参是(* 2 6)，因而计算外面的乘法之前必须先计算出(* 2 6)的值。所以对实参求值是为了进行函数的复合求值。对最里层的数的求值本身没有什么意义，而是为了照顾一般情况而这样做的。我们可以验证 LISP 对数求值的结果仍然是数本身。例如：

```
> 12
12
> 8
8
```

总之，LISP 解释程序对用户打入的任何东西都进行求值，并回送结果。

在(*(* 2 6)8 2)中的 8 和 2 是外层乘法函数的实参，而(* 2 6)的计算结果 12 也是一个实参，但我们也把(* 2 6)叫做外层乘法函数的一个实参。只要不引起混乱，我们不在名词术语上下功夫。

前面的+，-，* 都是 Common LISP 的算术运算函数。常用的算术运算还有 1+和 1-，分别是增 1 和减 1 函数。注意，1+的中间不要用空格分开，它是一个函数名。另外，除法符号为/。可以在下面的对话中理解这些函数：

```
> (1+5)
6
> (1-7)
6
> (/  10  2  2)
5/2
> (/  10.0  2  2)
```

```
2.5
>
```

除号/后面第一个数是被除数，其后可给多个除数进行连除。

Common LISP 还提供超越函数，包括指数、对数和三角函数等。

```
> (expt 2)
7.389056
> (expt 2 3)
8
>
```

其中(expt 2)表示求 e 的平方，它总是回送浮点数，而(expt 2 3)则表示求 2 的 3 次方。其它函数的用法可参见附录。

最后我们给出计算二次方程根的例子。设一元二次方程 $ax^2 + bx + c = 0$ 的两个根为 x_1 和 x_2，并设 a = 6, b = 7, c = 1。首先，把三个系数做为 a、b、c 的值告诉 LISP：

```
> (setq  a  6    b  7    c  1)
/
>
```

然后，用 LISP 表示求根公式：

$$\frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

并把两个根的值保存在 x_1 和 x_2 中：

```
> (setq x1 (/ (+ (-b)
                 (sqrt (- (* b b) (* 4.0 a c))))
                 (* 2 a)
           x2 (/ (- (-b)
                    (sqrt (- (* b b) (* 4.0 a c))))
                 (* 2 a)))
-1.0
> x1
-0.16666667
> x2
-1.0
>
```

§2.2 符号表达式

从上节所讲的内容看，LISP 解释器很像一个计算器。如何保存计算结果并用来继续计算呢？这就要介绍符号的概念。在其它的程序设计语言中都有变量，可把计算结果赋给变量。LISP 的符号起类似的作用，任何以字母开头的不含特殊字符的字符串都可做为符号。例如 foo，long－symbol－name 都是合法的符号。符号名中可以含有连字符把不同的词连接

起来。如果我们要把一个值保存起来，可以用函数 setq 把它赋给一个符号，

```
〉(setq x 5)
5
〉
```

这里的 setq 是 LISP 的一个特殊函数。Common LISP 把特殊函数叫特殊式(special form)。特殊函数不遵守上一节讲的一般求值规则，在函数求值之前不一定对每个实参求值。setq 对它的第一个实参不求值，只对第二个实参求值，并把求值结果赋给第一个实参，并作为函数值回送。上例中第二行的 5 就是函数 setq 回送的值。现在符号 x 的值是 5，

```
〉x
5
〉(+ x 10)
15
〉
```

LISP 对打入的 x 求值，结果为 5。我们再把 x 保存的值加上 10，结果为 15。

如果对一个未曾赋值的符号求值，就会发生错误，

```
〉y
ERROR:
  ATTEMPT TO TAKE THE VALUE OF AN
  UNSIGHNED VARIABLE
……
-〉
```

在 GCLISP 中的情况是，

```
* y
  ERROR: Y HAS NO GLOBAL VALUE
……
1〉
```

可见，发生错误之后，LISP 不再给出顶层的提示符，这表示它进入了查错状态。这时可以利用查错程序的命令找出错误的原因。如果我们已知错误所在，而想回到 LISP 的顶层，可打入(:A)，在 GCLISP 中打入<Ctrl－G>，即同时按下 Ctrl 和 G 键。

至今，我们接触到的数和符号，它们统称为原子，因为它们是 LISP 中不能再分割的对象。数或符号用括号括起来就构成表。表中的元素用空格分开。没有元素的表叫空表。空表可用 () 表示，也可用 NIL 表示。空表也是原子，这是 LISP 的一个不太符合逻辑的规定。表和原子构成了 LISP 的主要对象。表是递归定义的，即表中的元素可以是另外的表。例如，

```
((BLUE SKY) (GREEN GRASS) (BROWN EARTH))
(FOO 973 (TWO WHITE DUCK))
(SETQ Y (* 2 3))
```

表中的元素个数叫表的长度。上面第一个表的长度是3，它由长度为 2 的三个子表组成，上面第三个表是个函数调用，它也是个表。

原子和表统称为 LISP 的符号表达式。因而符号表达式构成了 LISP 语言的最重要的数

据结构。图 2.1表示出 LISP 基本的数据类型及其关系。

除了上述的数据类型之外，Common LISP 还有串(string)、向量、序列、数组、结构、流等数据类型，这将在以后介绍。这里只说明串。串是用双引号引起来的字符串，例如，“abc”，“this - is - a - long - string”，“how are you”，串中可包含空格。Common LISP 中的文件名用串表示。如果你把自己编的 LISP 程序用编辑程序写在一个文件中，文件名假定是 my.lisp，那么可以在 LISP 解释程序的提示符下打入

```
〉(load “my.lisp”)
```

这里 load 是装入文件的函数。执行 load 的效果就如同把文件中的内容在 LISP 解释程序下打入一样。

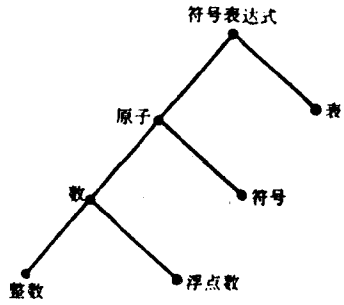

图 2.1　LISP 的符号表达式

LISP 对串求值结果仍为串本身，这与处理数是一样的。

§ 2.3　求值与赋值函数

我们已经看到，LISP 解释程序总是对打入的符号表达式进行求值。但有时我们不想这样做，这就需要本节介绍的禁止求值的函数。有时还要在我们的控制下对一个表达式求值，LISP 也提供了这种函数。

首先我们引入函数 quote。它的作用是阻止对其实参求值而直接回送。例如，

```
〉(quote (a b c))
(A B C)
〉
```

如果直接打入(a b c)，LISP 会认为 a 是函数名，b 和 c 是 a 的实参，因为 a 不是已定义的函数，所以 LISP 会发出错误信息。在(a b c)前加 quote 后，LISP 直接回送(A B C)。注意，Common LISP 对读入的字母不分大小写，回送的值显示出的字母一律用大写。这只是一个规定。我们可以让 LISP 打印出小写字母，这在讲到输入输出函数时再做介绍。

quote 函数看起来没有什么意思，但它很常用。为此，LISP 专门给它一个缩写，即用单引号“’”或“ ‘”来代替 quote：

```
〉'(a b c)
(A B C)
〉(setq x '(a b c))
(A B C)
〉
```

这里我们把表(a b c)做为值赋给符号 x。LISP 的符号的值不限于是数，可以是任何表达式。我们还可以把符号 y 做为值赋给 x，

```
〉(setq x 'y)
```

```
Y
〉x
Y
〉
```

其中我们在 y 前面加上 quote，因为 y 还没有值，所以我们禁止 LISP 对 y 求值，而只是把符号 y 做为 x 的值。因为 setq 本身已经规定对其第一个实参不求值，所以 x 前面不必加 quote。实际上 setq 就是 setquote 的简写。

与 setq 类似的函数是 set。它是个普通函数。因而 LISP 对它的所有的实参都求值，然后把第二个实参求值结果赋给第一个实参求值的结果，

```
〉x
Y
〉(set  x,   z)
Z
〉y
Z
〉x
Y
〉
```

由于前面已把 x 赋值为 y，set 对 x 求值结果为 y，对第二个实参求值结果为 z，然后把 z 赋给 y。所以 set 函数执行之后，y 的值是 z，而 x 的值仍为 y。一般情况下，我们不做这样的间接赋值，因此 set 很少用。setq 和 set 的变元个数不限于两个，但应是偶数个。可以同时对多个变量赋值。

```
〉(setq a 1 b 2 c 3)
3
〉a
1
〉b
2
〉c
3
〉
```

这里把 a、b、c 分别赋值为1、2、3，但 setq 只回送最后一对赋值的结果。

另一个重要的赋值函数是 let。它分为两部分，第一部分对变量赋值，第二部分是若干表达式，第一部分各变量的赋值只在第二部分的表达式中有效。例如，

```
〉(let ((x 1) (y 2) (z 3))
    (+ x y z))
6
〉
```

在 let 后面的第一个表中，每个子表规定了一个赋值。上例中有三个子表，分别把 x、y、