第一章　Lisp 简介

1.1　符号计算

到目前为止，我们所做的只是以简单的袖珍计算器的方式操纵数字。Lisp 比计算器更有用有两个主要原因。首先，它允许我们操作数字以外的对象，其次，它允许定义在后续计算中可能有用的新对象。我们将依次研究这两个重要属性。

除了数字，Lisp 还可以表示字符（字母）、字符串和任意符号，我们可以自由地将这些符号解释为数学世界之外的事物。Lisp 还可以通过将多个对象组合成一个列表来构建非原子对象。这种能力是基本的，在语言中得到了很好的支持；事实上，Lisp 这个名字是 LISt Processing 的缩写。

下面是一个列表计算的实例：

```
> (append '(Pat Kim) '(Robin Sandy)) => (PAT KIM ROBIN
  SANDY)
```

这个表达式将两个名字的列表连接在一起。计算此表达式的规则与数值计算的规则相同：将函数（在本例中为 append）应用于参数的值。

不寻常的部分是引号（'），它用于阻止对以下表达式的求值，并按字面意思返回它。如果我们只有表达式（Pat Kim），则可以通过将 Pat 视为函数并将其应用于表达式 Kim 的值来对其进行评估。这不是我们想要的。引号指示 Lisp 将列表视为一段数据，而不是函数调用：

```
> '(Pat Kim) => (PAT KIM)
```

在其他计算机语言（和英语）中，引号通常成对出现：一个标记开始，一个标记结束。在 Lisp 中，单引号用于标记表达式的开头。由于我们总是知道一个表达式有多长——要么到原子的末尾，要么到列表的匹配括号——我们不需要明确的标点符号来告诉我们表达式在哪里结束。引号可以用于列表，如'(Pat

Kim)，也可以用于符号，如'Robin，事实上还可以用于其他任何东西。以下是一些示例：

```
> 'John => JOHN

> '(John Q Public) => (JOHN Q PUBLIC)

> '2 => 2

> 2 => 2

> '(+ 2 2) => (+ 2 2)

> (+ 2 2) 4

> John => *Error: JOHN is not a bound variable*

> (John Q Public) => *Error: JOHN is not a function*
```

请注意，'2 的计算结果为 2，因为它是一个引号表达式，而 2 的计算值为 2，是因为数字本身的计算结果。同样的结果，不同的原因。相比之下，'John 求值为 John 是因为它是一个引号表达式，但求值 John 会导致错误，因为求值一个符号意味着得到符号的值，而没有给 John 赋值。

符号计算可以嵌套，甚至可以与数值计算混合。以下表达式使用内置函数 list 以与我们之前看到的略有不同的方式构建名称列表。然后，我们将看到如何使用内置函数 length 查找列表中的元素数量：

```
> (append '(Pat Kim) (list '(John Q Public) 'Sandy))
(PAT KIM (JOHN Q PUBLIC) SANDY)

> (length (append '(Pat Kim) (list '(John Q Public) '
    Sandy)))
4
```

关于符号，有四点要强调：

- 首先，重要的是要记住，Lisp 不会给它所操纵的对象附加任何外部意义。例如，我们自然会认为（Robin Sandy）是两个名字的列表，而（John Q

Public）是一个人的名字、中间的首字母和姓氏的列表。Lisp 没有这样的先入为主的观念。对于 Lisp 来说，Robin 和 xyzzy 都是非常好的符号。

- 其次，为了进行上述计算，我们必须知道 append、length 和 + 是 Common Lisp 中定义的函数。学习一门语言涉及记忆词汇（或知道在哪里查找），以及学习形成表达式和确定其含义的基本规则。Common Lisp 提供了 700 多个内置函数。在某些时候，读者应该翻阅参考文本，看看里面有什么，但大多数重要功能都在本书的第一部分中介绍。

- 第三，请注意 Common Lisp 中的符号不区分大小写。我的意思是，输入 John, john, 和 jOhN 都引用了相同的符号，通常打印为 JOHN。

- 第四，请注意，符号中允许使用各种各样的字符：数字、字母和其他标点符号，如 “+” 或 “!” 构成符号的确切规则有点复杂，但通常的惯例是使用主要由字母组成的符号，单词之间用破折号（-）分隔，最后可能还有一个数字。一些程序员在命名变量时更为自由，并包含 “?! $/<=>” 等字符。例如，在 Lisp 中，一个将美元转换为日元的函数可能会用符号 $-to-yen 或 $->yen 命名，而在 Pascal 或 C 中，可能会使用 DollarsToYen、dollars_to_yen 或 dol2yen 这样的符号。这些命名约定有一些例外，将在出现时处理。

1.2 变量

我们已经了解了符号计算的一些基础知识。现在，我们继续讨论编程语言最重要的特征：能够根据其他对象定义新对象，并为这些对象命名以供将来使用。在这里，符号再次发挥着重要作用，它们用于命名变量。变量可以取一个值，该值可以是任何 Lisp 对象。给变量赋值的一种方法是使用 setf：

```
> (setf p '(John Q Public)) => (JOHN Q PUBLIC)
> p => (JOHN Q PUBLIC)
> (setf x 10) => 10
> (+ x x) => 20
> (+ x (length p)) => 13
```

在将值（John Q Public）赋值给名为 p 的变量后，我们可以引用名为 p 的值。同样，在为名为 x 的变量赋值后，我们也可以引用 x 和 p。

符号也用于在 Common Lisp 中命名函数。每个符号都可以用作变量或函数的名称，或两者兼而有之，尽管很少（也可能令人困惑）同时使用符号名称。例如，append 和 length 是命名函数但没有值作为变量的符号，pi 不命名函数，而是一个值为 3.1415926535897936（或其附近）的变量。

1.3 特殊形式

细心的读者会注意到 setf 违反了求值规则。我们之前说过，像 +、-和 append 这样的函数通过首先评估它们的所有参数，然后将函数应用于结果来工作。但是 setf 并不遵循这个规则，因为 setf 根本不是一个函数。相反，它是 Lisp 基本语法的一部分。除了原子和函数调用的语法，Lisp 还有少量的语法表达式。它们被称为特殊形式。它们与其他编程语言中的语句具有相同的目的，并且确实具有一些相同的语法标记，例如 if 和 loop。Lisp 的语法和其他语言有两个主要区别。首先，Lisp 的语法形式总是列表，其中第一个元素是少数特权符号之一。setf 是这些符号之一，因此 (setf x 10) 是一种特殊形式。其次，特殊形式是返回值的表达式。这与大多数语言中的语句形成鲜明对比，后者具有效果但不返回值。

在计算像 (setf x(+ 1 2)) 这样的表达式时，我们将符号 x 命名的变量设置为值 (+ 1 2)，即 3。如果 setf 是一个普通函数，我们将计算符号 x 和表达式 (+ 1 2)，并对这两个值进行处理，这根本不是我们想要的。setf 之所以被称为特殊形式，是因为它做了一些特殊的事情：如果它不存在，就不可能编写一个为变量赋值的函数。Lisp 的哲学是提供少量的特殊形式来完成原本无法完成的事情，然后期望用户将其他所有内容都作为函数编写。

特殊形式一词被混淆地用来指代像 setf 这样的符号和以它们开头的表达式，比如 (setf x 3)。在《Common LISPcraft》一书中，Wilensky 通过调用 setf 一个特殊函数并为 (setf x 3) 保留特殊形式来解决歧义。这个术语意味着 setf 只是另一个函数，但它是一个特殊的函数，因为它的第一个参数没有被求值。在 Lisp 主要是一种解释性语言的时代，这种观点是有道理的。现代观点认为，setf 不应被视为某种异常函数，而应被视为由编译器专门处理的特殊语法的标记。因此，特殊形式 (setf x (+ 2 1)) 应被视为 x = 2+1 的等价物。当存在混淆的风险时，我们将 setf 称为特殊形式运算符，将 (setf x 3) 称为特殊格式表达式。

原来引号只是另一种特殊形式的缩写。表达式'x 等价于（quote x），一个计算结果为 x 的特殊形式表达式。本章中使用的特殊形式运算符是：

- defun : define function
- defparameter : define special variable
- setf : set variable or field to new value
- let : bind local variable(s)
- case : choose one of several alternatives
- if : do one thing or another, depending on a test
- function(#') : refer to a function
- quote(') : introduce constant data

1.4 列表

目前为止我们看到了两个操作 list 的函数: append 和 length。下面是更多关于 list 的处理函数:

```
> p => (JOHN Q PUBLIC)

> (first p) JOHN

> (rest p) => (Q PUBLIC)

> (second p) => Q

> (third p) => PUBLIC

> (fourth p) => NIL

> (length p) => 3
```