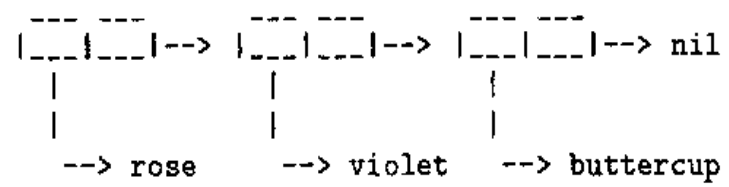
第9章 列表是如何实现的

在 Lisp 中，原子是以一种直观的方式记录在计算机中的；如果说它在实际的实现上不那么简单，那么它至少在理论上是简单的。例如，原子“rose”是由紧挨着的四个字符“r”、“o”、“s”、“e”记录下来的。在另一方面，列表是用一种不同的方式保存的。列表的保存机制同样简单，但是要理解这个思想需要花一点时间。列表是用一系列成对的指针保存的。在这个成对的指针系列中，每一对指针的第一个指针要么指向一个原子，要么指向另外一个列表；而其第二个指针要么指向下一个指针对，要么指向符号 nil，这个符号标记一个列表的结束。

指针本身相当简单，就是它指向的电子地址。因此，一个列表实际上就是被保存为一系列电子地址。

例如，列表 (rose violet buttercup) 有三个元素：“rose”、“violet”和“buttercup”。在计算机中，“rose”的电子地址存储在计算机内存片段中，其后紧跟着给出原子“violet”存放位置的地址。这个地址又与给出“buttercup”存放位置的地址连接在一起。

这听起来似乎很复杂，用一个图来说明就很简单了：

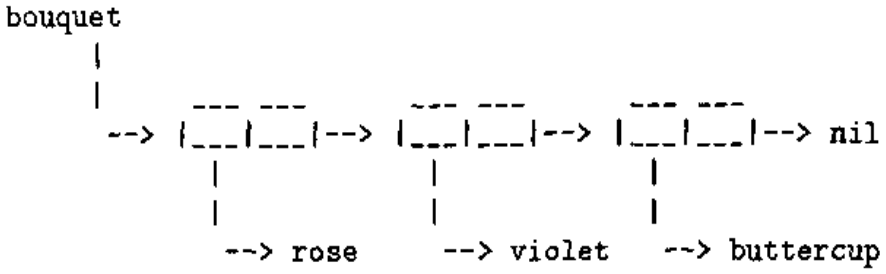


在这个图中，每一个方框代表计算机内存中的一个“字”（word），它通常是以内存地址的方式存放一个 Lisp 对象。这些方框，也就是这些地址，是成对的。图中的箭头要么是指向一个原子的地址，要么是指向另外一对地址的地址。第一个方框是“rose”的地址，其箭头指向“rose”；第二个方框是下一对方框的地址。这第二对方框中的第一个方框是“violet”的地址，而其第二个方框指向下一对方框的地址。最后一对方框的第二个方框（也就是最后一个方框）指向符号 nil。这个符号标记一个列表的结束。

当用一个函数（如 setq）将一个列表赋给一个变量时，实际上就是将列表的第一个方框的地址赋给那个变量。因此，对下面这个表达式求值

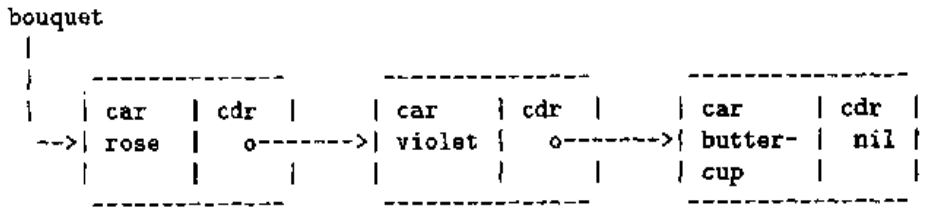
```
(setq bouquet '(rose violet buttercup))
```

产生如下图展示出来的情况：



在这个例子中，符号 bouquet 中保存第一个方框对的地址。确实，这个符号由一组地址框组成，其中第一个地址框就是“bouquet”一词的地址；如果有同名的函数定义加到这个符号上，其中第二个地址框是这个函数定义的地址；其中第三个地址框就是列表 (rose violet buttercup) 的成对地址框系列中第一对的地址；等等。

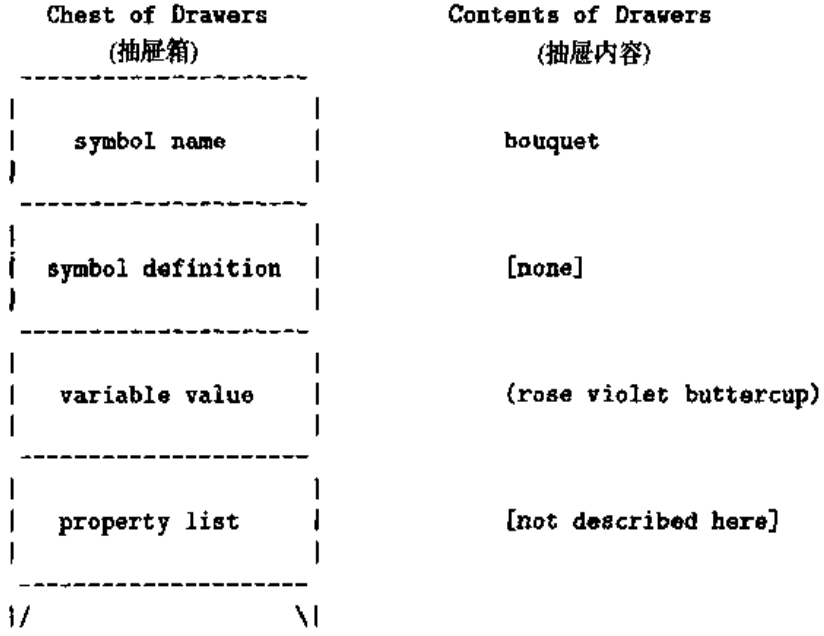
这个列表同样可以用下面这种不同的方式来表示：



在前面的一节中，我曾经建议将一个符号想象成为一个抽屉箱。函数定义被放在其中一个抽屉中，符号的值放在另外一个抽屉中，等等。保存符号值的抽屉中的内容的改变，不影响保存函数定义的抽屉中的内容。反之亦然。实际上，放在每一个抽屉中的都是符号值的地址或者函数定义的地址。这就像你在阁楼中找到一个旧抽屉箱，在其中一个抽屉中发现了一张地图，这张地图告诉你财宝存放的位置。

（除了符号名、符号定义和变量的值之外，符号还有一个“抽屉”保存其属性列表。这个属性列表能用于记录其他信息。属性列表不在这里介绍，有关它的内容可以参见《*GNU Emacs Lisp* 技术手册》的“属性列表”一节。）

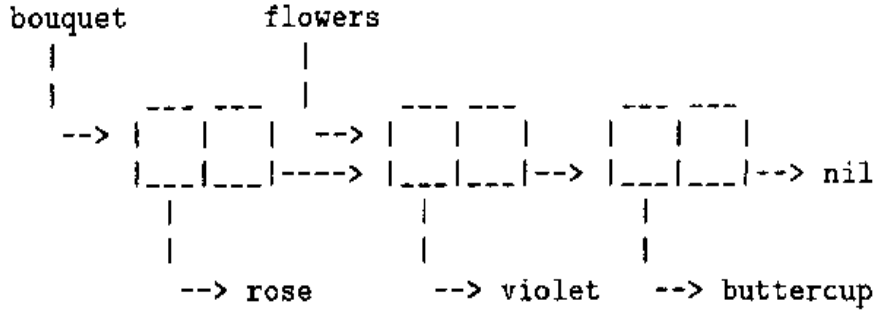
下面是一个充满想象力的表示：



如果一个符号被设置为一个列表的 cdr，这个列表本身没有改变；符号仅仅只有列表的地址。（用术语来说，car和 cdr 是非破坏性的。）因此，对下面的表达式求值

```
(setq flowers (cdr bouquet))
```

将产生这样的结果：



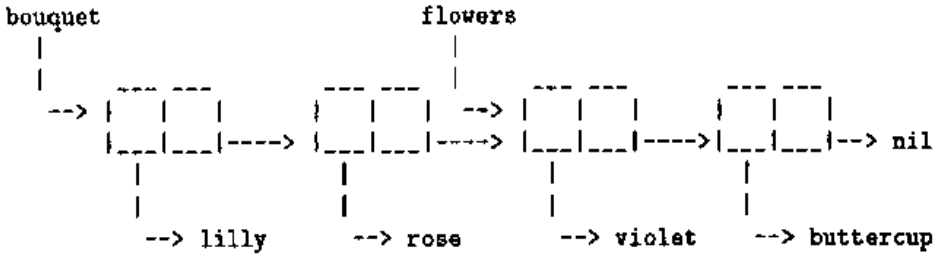
flowers 的值是（violet buttercup），这就是说，符号 flowers 存放成对的地址框的地址，成对的地址框的第一个地址对中存放violet 的地址，第二个地址对中存放 buttercup 的地址。

一个成对的地址框被称为一个“cons 原胞”（*cons cell*）或者一个“带点偶对”（*dotted pair*）。具体内容参见《*GNU Emacs Lisp*技术手册》中的“Lisp 类型”一节以及“带点偶对注解”一节。

就像前面说的那样，cons函数在一系列地址对前面插入一个新的地址对。例如，对下面的表达式求值

```
(setq bouquet (cons 'lilly bouquet))
```

会得到：



然而，这并不改变符号 flowers 的值，不信可以通过对下面的表达式求值来看一看：

```
(eq (cdr (cdr bouquet)) flowers)
```

这个表达式的返回值为“真”（t）。

除非被重新赋值，否则符号 flowers 的值仍旧是 (violet buttercup)。这就是说，它拥有首地址为 violet 的地址的cons 原胞的地址。同样，这并不改变任何已经存在的 cons 原胞，它们还是原封不动地存在那里。

因面，在 Lisp 中，要得到一个列表的cdr，只要得到地址系列中下一个cons原胞的地址即可；要得到一个列表的 car，就是得到这个列表的第一个元素的地址；要用 cons 函数在一个列表中插入一个新元素，其作用就是往列表中插入了一个新的 cons 原胞。这就是列表实现的方式。Lisp 底层的结构就是这样不可思议的简单。

cons 原胞系列中最后一个地址指向什么呢？就是指向空列表(即符号 nil)的地址。

总之，当为一个 Lisp 变量赋值时，它提供的是列表的地址，变量就指向这个列表的地址。

练习

将符号 flowers 设置为 violet 和 buttercup 两个元素组成的列表。往这个列表中增加两种新的花名，并将这个新的列表赋值给 more-flowers 变量。将 flowers 的car设置为一种鱼的名字。看一看 more-flowers 列表现在的内容是什么？