commonlisp

入门

万泽[1] | 编者[2]

版本: 1.0

[1]作者：湖南常德人氏
[2]编者：邮箱：a358003542@gmail.com。

前言

这本小册子主要记录了我收集的关于 commonlisp 的入门知识。

目 录

前言 **i**

目录 **ii**

1 准备工作 **1**

1.1 xverbatim 环境 1

1.2 emacs 的 commonlisp 环境配置 2

2 基础知识 **4**

准备工作

xverbatim 环境

为了显示 commonlisp 的代码，现在 xverbatim 宏包建立的 xverbatim 环境中经过一些优化之后一切照旧，可选项 129 表示显示代码并执行并显示结果。但是命令行脚本执行 commonlisp 文件还是和 repl-mode[1]有点区别的，最大的区别就是常规的如 (+ 1 2) 这样的输入没有任何输出信息的。为了克服这点，早期采取的是这样的形式。

```
1 (format t "~&~s" (* 2 8) )
```

16

这里 lisp 是标识了等下要生成 common-lisp 文件的后缀名为 lisp，然后染色方案识别为 common-lisp。

这里的~& 表示确认等下的输出为新的一行。这里的~s 表示等下输出一个内容，这个内容是后面表达式的 eval 之后的结果。

在早期说明一些基本的知识的时候我会采取这种形式，读者需

[1] read-eval-print-loop

要记住的是输出的结果就是 (* 2 8) 运行之后的结果。随著学习的深入，一切没必要显示出来的内部操作将不会采取这种形式了。

emacs 的 commonlisp 环境配置

我现在是在 ubuntu13.10 的 emacs24 之下，安装 **emacs24** 运行 sudo apt-get install emacs24 安装即可。然后需要安装 common-lisp 的编译运行环境，推荐 **sbcl** 或者 **clisp**，同样类似的用 apt-get 命令安装即可。记得以前要配置 slime 环境，需要经过一番周折。现在似乎只需要用 apt-get 命令安装 slime 即可。

测试运行情况就是进入 emacs24，按住 Alt 键和 X 字母[2]，然后输入 slime。调配窗口的命令是 Ctrl+X，然后输入数字 **1**，这将只显示一个窗口。类似有数字 **2**，生成上下两个窗口；数字 **3**，生成左右两个窗口。

知识更新换代太快了，而我又是一个喜欢简洁的人，所以我在这里忍住谈论太多。只是简单提一下，在 Ubuntu 系统的主文件夹里面有个隐藏文件——**.emacs**。这个文件里面写著 emacs 用户自己的一些 DIY。有一些配置可以在 emacs 主面板上设置，然后保存配置即可。

这里简单提一下我觉得这里有用的配置:

```
(global-linum-mode 1)
```

左侧显示行数

```
(cua-mode 1)
```

支持 Ctrl+C，Ctrl+V 复制粘贴快捷键

```
(setq show-paren-delay 0 show-paren-style 'parenthesis)
```

[2]在 emacs 中，Alt+X 将调用一些外部命令，Ctrl+X 将调用一些外部命令。

```
(show-paren-mode 1)
```

高亮显示括号

```
(setq scheme-program-name "guile")
```

这个也在这里说了，通过 apt-get 可以直接安装 guile 软件——scheme 语言的编译器。scheme 语言和 common-lisp 语言很类似，同时在说明编程语言思想上更加直白和透彻，推荐有空接触下。通过上面这一行代码，就可以在 emacs 上运行 Alt+X run-scheme 直接调用 guile 模式来运行 scheme 的 repl-mode 了。

基础知识

Common-lisp 以下全部都简称 lisp，lisp 的数据类型分为符号和数字 [1]。数字又分为整数 integer、实数 floatingpoint 和 ratio 分数三类。其中分数都会自动化为最小分母的形式。比如：

```
(format t  "~&~s" (/ 1 2))
(format t  "~&~s" (/ 2 4))
```

1/2

1/2

同时我们看到 1 除以 2 并没有化成 0.5 之类的形式，这是 lisp 和其他程序语言很大的一个区别。

Lisp 中的 T 和 nil 表示逻辑的真值和假值，然后空表 () 也表示 nil，而其他所有东西在逻辑上都被视作真值。

一些逻辑上的判断函数，numberp 判断是否是数字，symbolp 判断是否是符号。Equal 判断是否两个值相等。zerop 判断是否是 0，oddp 判断是否是奇数，evenp 判断是否是偶数。还有小于号也是一个判断函数 <，还有大于号 >。not 在逻辑上反值。

举些例子：

```
(format t "~&~s"  (numberp 2))
(format t "~&~s" (symbolp 'dog2-dfdf))
(format t "~&~s"  (equal 'dog 'dog))
(format t "~&~s"   (< -4 -3))
(format t "~&~s" (oddp 3))
(format t "~&~s" (not '(a d c)))
```

T

T

T

T

T

NIL

通过上面的例子我们注意到 lisp 语言的符号几乎是没有什么限制的，除了数字之外就是符号。然后数字前面可以加引用号’，也可以不加。不过符号就一定要加，否则会出错。还有就是即使列表这样的东西在 lisp 中通过 not 操作也会返回 nil 值。

lisp 的一些基本内置函数：`+  -  *  /  abs  sqrt` 前面的加减乘除的输入参数可以是任意的输入参数。具体就是第一个数连续加（减或者乘除）上后面的数。不过加减乘只有一个参数的时候结果是什么都好想想，

在电脑中 cpu 的工作可以理解成为硬件化的某些基本函数，比如加减之类的。而其他我们接触到的所有函数都只是输入指针，输出指针。而这些指针的具体内存地址并不永久存在，只是暂时存在

在某个寄存器或者堆栈上（因为具体的内存地址也是需要处理之后生成的一个结果）。也就是所有复杂的函数都是架构在最基本的几个硬件函数至上的，所不同的是这些硬件函数会具体根据不同的地址输入不同的待处理信息或者在不同的地址上输出处理后的信息。

lisp，list processor，中文名字就是链表处理语言。显然，list 就是里面最基本的数据结构，而其他所有的数据结构都是基于 list 架构出来的。一个链表的结构非常简单，就是一个 cons cell 结构，这个结构由 car 指针地址和 cdr 指针地址两个指针地址组成。

现在演示下 (a (b c) d) 的结构

现在我们把这个列表简单地命名为 list：

```
1 (setq list '(a (b c) d))
2 (format t "~&~s"  list )
3 (car list)
```

(A (B C) D)

在这里发生的什么呢？car 函数是如此的基本，在这里 list 传递给别人的也不是别的，就是它指向那个 (a (b c) d) 的地址。然后 car 函数根据这个地址顺藤摸瓜找到了第一个地址，然后根据这个地址找到那段内存的数据，然后返回回来。所以原子（atom）类型的符号是实实在在的一个存储在内存中的信息，这段信息需要用基本函数（primitive function）quote，也就是前面加个’来调用出来。而像上面 list 通过 setq 定义的符号就成为了一个变量，这个变量和 list 不同只是一个指针地址的结构，声明他就自动调用它所映射的在内存中的信息。

那么 cdr 函数的工作也就简单了，根据 list 所提供的地址，顺藤摸瓜找到第一个链表的 cdr 地址，然后创造一个内存，这个内存中的地址信息指向那个 cdr 地址。然后将这个值作为返回结果。记住所有的括号在内存中都是不存在，而 cdr 返回后面部分内容只是基于链表的向后而不能向前查询的内在机制决定的。

关于 car，cdr 的诸多衍生如什么 caddr 之类的就不多说了，记住从右向左看就是的了。

前面说了再复杂的 list 结构都是由最基本的 cons cell 组成的。所谓 cons 结构，这个基元细胞就是 car 地址加上 cdr 地址这样的组合形式。那么内置基本函数 cons 的功能就是接受两个地址，第一个地址存入 car 地址，第二个地址存入 cdr 地址。这样形成一个 cons cell 的基元结构。而内置基本函数 list 就是迭代调用 cons 函数的结果。

基本流程控制

图片

参 考 文 献

[1] David S Touretzky. COMMON LISP: A Gentle Introduction to Symbolic Computation. Courier Dover Publications, 2013.