4. 定义函数

1.简介

在前面的章节中，我已经解释了：

1.如何去安装MIT-Scheme

2.Scheme解释器是如何对S-表达式求值

3.基本的表操作

在本章中，我会介绍如何自定义函数。正如Scheme是函数式编程语言，你需要通过构建小的函数来构造你的程序。因此，明白如何构造并结合这些函数对掌握Scheme是非常关键的。

因为在前端定义函数非常不方便，因此你通常需要在编译器中编辑好代码，并在解释器中加载他们。

2.如何定义简易的函数并加载它们

你可以使用define来将一个符号与一个值绑定。通过这个操作符，你可以定义比如数，字符，表等任何类型的参数与函数。

让我们使用任意一款编辑器（记事本亦可）来编辑代码片段1中展示的代码，并将它们存储为‘hello.scm’，放置在类似于‘C:\doc\scheme\’文件夹下。如果可能的话，把这些文件放在你在第一章定义的MIT-Scheme默认文件夹下。

代码片段1（hello.scm）

; Hello world as a variable

(define vhello "Hello world") ;1

; Hello world as a function

(define fhello (lambda () ;2

"Hello world"))

接下来，给Scheme解释器下面的命令：

(cd "C:\\doc\\scheme")

;Value 14: #[pathname 14 "c:\\doc\\scheme\\"]

(load "hello.scm")

;Loading "hello.scm" -- done

;Value: fhello

hello.scm就通过这些命令。如果你的当前目录被设定在了脚本所在目录，那么你就不需要再键入第一行的命令了。然后，向解释器键入下面的命令：

vhello

;Value 15: "Hello world"

fhello

;Value 16: #[compound-procedure 16 fhello]

(fhello)

;Value 17: "Hello world"

操作符define有两个参数，用于申明变量。Define操作符会使用第一个参数作为全局参数并将其与第二个参数绑定起来。因此，代码片段地1行中，我们声明了一个全局参数vhello，并将其与“Hello，World”绑定起来。

另一方面，在第二行声明了一个返回“Hello World”的过程。

特殊形式lambda用于定义过程。Lambda需要至少一个的参数，第一个参数是由定义的过程所需的参数组成的表。因为本例fhello没有参数，所以参数表是空表。

在解释器中键入vhello，解释器返回“Hello，World”。如果你在解释器中键入fhello，它也会返回向下面这样的值：

**#[compound-procedure 16 fhello]**,

这说明了Scheme解释器把过程和Conventional数据类型用同样的方式对待。正如我们在前面章节解释的那样，Scheme解释器通过内存空间中的数据地址操作所有的数据，因此，所有存在于内存空间中的对象都以同样的方式处理。

如果把fhello当过程对待，你应该用括号扩住这些符号，比如(fhello)。

然后解释器会按照第二章讲述的规则那样对它求值并返回“Hello World”。

3 定义有参数的函数

可以通过在lambda后放一个参数表来定义有参数的函数。将代码片段2保存为farg.scm并放在同hello.scm一致的目录。

[code 2] (farg.scm)

; hello with name

(define hello

(lambda (name)

(string-append "Hello " name "!")))

; sum of three numbers

(define sum3

(lambda (a b c)

(+ a b c)))

保存文件，并在解释器中载入此文件，然后调用我们定义的函数。

(load "farg.scm")

;Loading "farg.scm" -- done

;Value: sum3

(hello "Lucy")

;Value 20: "Hello Lucy!"

(sum3 10 20 30)

;Value: 60

Hello

函数hello有一个参数(name)，并会把“Hello”，name的值，和“！”连结在一起并返回。

预定义函数string-append可以处理任意多个数的参数，并返回将这些参数连结在一起后的字符串。

Sum3

此函数有三个参数并返回这三个参数的和。