

Interpreters

本节内容与 Scheme Project 密切相关

解析与 scheme-read

输入程序 `tokenize-lines()` → 令牌列表 `Buffer()` → 缓冲区 `scheme-read()` → (可计算的) 表达式
此处每调用一次 `scheme-read()` 就会从缓冲区读入 tokens 直至形成一个完整表达式。

应用与 scheme-apply

`scheme-apply` 与上一节中的 `apply` 函数相似，只是功能更为强大。

它接受 `function` 和 (数个) `arguments`，返回 `function("arguments")` 的结果。

若 `function` 是用户自定义的函数，还需新建 `frame`，传入 `arguments` 并调用 `scheme-eval` 得到结果。

变量符号与 symbols

实际上 Scheme 中亦有帧的概念，我们可用字典完成变量名与值的绑定 (bindings)

其他函数与 special forms

实现细节略，包括 `define`, `lambda`, `cond`, `begin` … 等内置函数的实现方法。

优化递归调用 Getting Iteration via Recursion to Work

Tail contexts

尾部上下文

我们把返回值的表达式称为 tail context，它们的值决定了整个表达式的值。

在下面的例子中

```
(define (f x) (g (+ x 1)))  
(define (g x) (h x))  
(define (h x) (* x 2))
```

想办法告诉 f：调用 (g x) 实际上就是调用 (h x)。这样就减少了一次递归调用。

尾调用优化

上面的这个例子就是 tail call optimization

即在尾部上下文中用一个调用代替另一个调用。