第一章

基本表达式

带括号的前缀形式

谓词——可被判断为真或假的表达式

命名与环境——最基本的抽象机制

通过构造和命名，将值或过程存入变量，这种对“名字- 对象”的存储称为“环境”

区别于C语言，Scheme的变量无类型，不用做静态类型检查，也就带来了灵活性

预定义基本过程／操作和特殊形式是构造程序的基本构件

过程定义是分解和控制程序复杂性的最重要技术之一

- 过程应用的代换模型只是为了帮助直观理解过程应用的行为
- 本课程要研究解释器工作工程的一组模型

表达式的求值过程：应用序和正则序求值

先求值参数后应用运算符 vs 完全展开之后归约

Scheme采用应用序求值，即会对每个变量求值，不管后面会不会用到

表达式的求值过程<-表达式语义的实现

C语言通过运算符的优先级、结合性、括号等确定运算顺序

对于Scheme和C，没规定运算对象的求值顺序，所以不要写只有按特定求值顺序才能得到所需结果的表达式。

Scheme的基本类型和表达式

1. 数值类型：任意大的整数、实数——加减乘除、比较判断、常用函数、随机数生成

2. 布尔类型 #t #f

3. 字符类型 #a #B #{ #space——字符比较、其他操作、数值字符转换

4. 字符串类型——比较、模式匹配

5. 组合数据类型

过程 = 黑箱抽象

过程抽象的本质是功能分解

定义过程时，关注计算的过程式描述，使用时只关心说明式描述

过程抽象是控制和分解程序复杂性的一种重要手段，也是记录和重用已有开发成果的单位

思考：习题1.6 P16

```
(define (sqrt-iter guess x)
 (new-if (good-enough? guess x)
 guess
 (sqrt-iter (improve guess x)
 x)))
```

会产生无限展开，不会返回。

因为new-if是一个过程，对参数的求值顺序是应用序，在这里可以明显看得出来：

https://gist.github.com/jolisper/3688628

但是if就是采用正则序的求值顺序：计算第一个谓词，算完之后才发现是false的话，接下去算第二个谓词。

序

计算机的程序设计：

- 不关心程序具体的用途
- 关心性能
- 关心能不能和其他程序平滑衔接成更大的程序

程序员应该追求好的算法和惯用法。即使某些程序难以精确描述，程序员也要估计性能并且设法优化。

前言

本书的目标：我们要有好的程序设计风格要素和审美观：

- 程序必须写得好读
- 最基本的材料是用于控制大型软件系统的复杂性的
    - 包括：抽象、混合与匹配

本书的思想方法：过程性认识论

即如何从命令式的观点去研究知识的结构

即弄清楚要计算什么，怎么将问题分解为可控的部分，如何对可控的部分展开工作

- 所以Lisp很适合用来教学

从更多角度观察和理解程序和程序设计中的问题

- 函数式程序设计
- 各种程序组织方式，分解和控制程序复杂性的技术
- 丰富的编程模式
- 对一些基础问题的理解
- 上述各方面与常规（命令式，eg.C/C++等）程序设计的关系和启示等

说明式知识vs过程性知识

是什么vs怎么做

软件开发的工作就在说明式雨过程式知识的结合点上

命令式计算和函数式计算

1. 程序设计的主流：命令式程序设计。基于一组基本操作，在一个环境里运行，操作效果是改变环境的状态，体现在所创建和修改的状态中。
2. 函数式程序设计：计算过程被看成是数据的交换，程序的行为就是数据的一系列变换。

常规程序以命令方式描述计算，接近实际硬件，可能高效，但编程时需要关注很多细节

函数式程序设计层次较高，更抽象，程序可能更清晰，但需要复杂的运行时支持，可能效率较低

Lisp是函数式语言之祖

Scheme不是纯函数式语言，为了效率和对计算的控制，提供了命令式特征（状态操作）