
# 函数式程序设计

## 尽享程序设计之奥秘：描述 what，推理 why，演算 how！

作者：胡振江

组织：北京大学

时间：2020 年 7 月 20 日

版本：0.0.1

电子邮件：huzj@pku.edu.cn

ElegantLaTeX Program

你所温柔正确的人总是难以生存，因为这世界既不温柔，也不正确。——比企谷八幡

# 目录

# 第一部分

# 函数式语言的基础

# 第 1 章 基本概念

内容提要

- 为什么要学习函数式程序设计
- 什么是函数式程序设计
- Haskell 程序设计环境

## 1.1 为什么要学习函数式程序设计

强调用数学思维能力在程序中的作用。而正是这种能力使得函数式程序设计成为有史以来最棒的程序设计方法。这其中涉及到的数学知识并不复杂，就像我们中学时学过的各种等式理论以及根据这些等式理论进行化简。这使得我们可以从一个简单，明显却不太高效的方法入手，通过应用一些熟知的恒等式进行变换，最后等到一个非常高效的解。这种程序设计方法其乐无穷。

另外，函数式程序设计也能培养和提高简单而清晰地表达计算思想和方法的能力。

函数式程序设计不是屠龙之技。在我上大学的时候，国内几乎没有人搞函数式语言，我的恩师上海交通大学的孙永强教授是为数很少的函数式程序语言研究者之一，由于但是的计算机计算能力的低下，尽管写出的程序简洁漂亮，但是运行起来很慢，给人以一种仅仅存在于某些偏门语言里的学究气的概论。然而我们观察当今的主流语言，会发现函数式程序设计近乎成为一种标配，关键的函数式程序设计的特征或深或浅地嵌入到各式语言中去。

1989 年，John Hughes 曾经写了一篇非常有影响的论文，叫 Why Funcntional Programmig Matters，阐述了函数式程序设计的特征及其重要性。函数式语言最主要的特征是用纯函数和高阶函数最简洁地描写计算，和有惰性计算方式来进行计算。这迎合了人们表达问题的能力和高效解决大规模开发问腿的需要。

学习一种全新的程序设计方式，困难不在于掌握新的语言，真正考验人的是怎么学会用另一种方式去思考。尽管如此，我们还是使用一种函数式语言，从某种意义上而言，来激励大家用函数式的思维方式。比如，用函数式设计语言，你无法用我们平时离不开的 loop，而需要用递归结构来描述重复的计算。

计算机科学的进步经常是间歇式的，好想法有时被搁置数十年后才突然间变成主流。举个例子来说，深度学习是 xxxx 提出来的，... 早年 Java 总被认为太慢，内存消耗太高，不适合于高性能的应用，如今硬件的发展使它成为利用最为广泛的一种语言。

函数式语言的发展轨迹也十分类似，诞生于学院，经过几十年的时间渐渐流行，并浸染了几乎所有现代语言。

在计算机科学短短的发展史上，有时候会从技术主流分流出一些树杈，有源于实务的，也有源于学术的。例如，在 230 世纪 90 年代个人电脑大发展的时期，第四代程序设计语言（4GL）出现了爆发式的流行，。。。这些语言的最大买点之一是比 C 等第三代语言

（3GL）有更高层次的抽象，4GL 的一行命令，3GL 需要很多行才能写出来。

Working with Lecacy Code 的作者 Michael Feathers 总结出函数式抽象与面向对象抽象的区别：面向对象的程序设计通过封装不确定因素来使代码被人理解；函数式程序设计通过尽量减少不确定因素来使代码被人理解。OOP 用封装，作用域，可见性来控制谁能感知状态，谁能控制状态，使得实现和接口分开。而函数式语言采用了另一种做法；与其建立机制来控制可变状态，不如尽可能消灭可变状态这个不确定因素。假如语言不对外暴露那么多有出错可能的特性，那么开发者就不那么容易出错。

OOP 的世界提倡开发者针对具体问题建立专门的数据结构，并设计处理这个数据结构的一套方法。而函数式语言实现重用的思想完全不同。函数式语言提倡在有限的几种关键数据（list，set，tree）上运用针对这些数据结构高度优化过的操作，以此构成基本的运转机制。用户根据需求，插入自己的数据结构和高阶函数去调整机构的运转方式。函数式程序设计的结构很方便我们在比较细小的层面上重用代码。

## 1.2 什么是函数式程序设计

函数式程序设计是程序的一种构造方法。它强调函数（function）及其函数作用（fuction application）。函数式程序设计语言是通过函数来描述计算的语言，而函数式程序设计就是利用函数及其组合来进行程序设计。

函数式程序设计使用简单的数学语言，清晰地描述问题和算法。

函数式语言设计的数学基础简答，支持对程序的性质进行推理（reasoning）

那么，什么是函数呢？根据维基百科，函数是描述哦两个集合间的一种对应关系：输入值集合中的每项元素皆能对应 唯一一项输出值集合中的元素。我们通常叫这个输出为函数的结果，而这个输入为函数的参数。

### 1.2.1 函数和类型

对于函数，我们用下面的记号

```
f :: X -> Y
```

来表示 `f` 是一个函数，其参数类型为 `X` ，返回值的类型为 `Y` 。在这里，类型可以简单地理解某类数据的集合。例如：

```
sin :: Float -> Float
```

表示 `sin` 是一个从浮点数类型到浮点数类型的一个函数。

数学上，我么用 `f(x)` 表示函数 `f` 作用于其参数 `x` ，但是在 Haskell 中我们可以省略括号，而使用 `f x` 来表示这个函数作用。例如， `sin 3.14` 和 `sin(3.14)` 都可以表示将函数 `sin` 作用于 `3.14` 。这里需要注意的有两点。一是，函数作用是左结合的，因此 `log sin x` 表示的是 `log sin) x` ，而不是 `log (sin x)` 。二是，函数作用结合最紧密，因此， `sin x + 3` 表示的是 `(sin x) + 3` ，而不是 `sin (x+3)` 。

### 1.2.2 函数复合

给定两个函数 `f :: X -> Y` 和 `g :: Y -> Z`，我们可以将他们复合成一个新的函数：

```
g . f :: X -> Z
```

该函数将 `f` 作用于类型为 `X` 的参数，得到类型为 `Y` 的结果，然后再将 `g` 作用于这个结果，最后得到类型 `Z` 为的结果。

```
(g . f) x = f (g x)
```

### 1.2.3 表达式

表达式（expression）是用来表示计算的。每个表达式都有某个类型的值（value）。

## 1.3 Haskell 程序设计环境

格拉斯哥 Haskell 编译器 (GHC: Glasgow Haskell Compiler) 是标准的 Haskell 编译器。它将 Haskell 编译成本地代码，支持并行执行，并带有更好的性能分析工具和调试工具。由于这些因素，在本书中我们将采用 GHC。GHC 主要有三个部分组成。

- ghc 是生成本地原生代码的优化编译器。
- ghci 是一个交互解释器和调试器。
- runghc 是一个以脚本形式 (并不要首先编译) 运行 Haskell 代码的程序，

如果你用的是 Windows 或 Mac，强烈推荐你下载 Haskell Platform。在 Linux 上，很多发行版在官方仓库里包含了 Haskell Platform。比如在基于 Debian 的系统中，你可以通过运行 sudo apt-get install haskell-platform 来安装。如果你用的发行版没有包含 Haskell Platform，你可以按照 Haskell Platform 网页上的介绍手动安装。

alex 这个重要的包你需要手动更新。Haskell Platform 包含的 alex 是版本 2，而 Yesod 使用的 Javascript 最小化 (minifier) 工具 hjsmin，需要版本 3。一定要在 Haskell Platform 搭建好后运行 cabal install alex，否则会有关于 language-javascript 包的报错信息。

Glasgow Haskell Compiler (GHC) 是一个学术 GHCi 是一个广泛使用的用于开发和运行 Haskell 程序的环境。

在 Haskell 语言的众多实现中，有两个被广泛应用，Hugs 和 GHC。其中 Hugs 是一个解释器，主要用于教学。而 GHC(Glasgow Haskell Compiler) 更加注重实践，

在本书中，我们假定你在使用最新版 6.8.2 版本的 GHC，这个版本是 2007 年发布的。大多数例子不要额外的修改也能在老的版本上运行。然而，我们建议使用最新版本。如果你是 Windows 或者 Mac OS X 操作系统，你可以使用预编译的安装包快速上手。你可以从 GHC 下载页面找到合适的二进制包或者安装包。

对于大多数的 Linux 版本，BSD 提供版和其他 Unix 系列，你可以找到自定义的 GHC 二进制包。由于这些包要基于特性的环境编译，所以安装和使用显得更加容易。你可以在 GHC 的二进制发布包页面找到相关下载。

我们在 [附录 A] 中提供了更多详细的信息介绍如何在各个流行平台上安装 GHC。

初识解释器 ghci ghci 程序是 GHC 的交互式解释器。它可以让用户输入 Haskell 表达式并对其求值，浏览模块以及调试代码。如果你熟悉 Python 或是 Ruby，那么 ghci 一定程度上和 python，irb 很像，这两者分别是 Python 和 Ruby 的交互式解释器。

The ghci command has a narrow focus We typically can not copy some code out of a haskell source file and paste it into ghci. This does not have a significant effect on debugging pieces of code, but it can initially be surprising if you are used to , say, the interactive Python interpreter. 在类 Unix 系统中，我们在 shell 视窗下运行 ghci。而在 Windows 系统下，你可以通过开始菜单找到它。比如，如果你在 Windows XP 下安装了 GHC，你应该从”所有程序”，然后”GHC”下找到 ghci。(参考附录 A 章节 Windows 里的截图。)

当我们运行 ghci 时，它会首先显示一个初始 banner，然后就显示提示符 Prelude>。下载例子展示的是 Linux 环境下的 6.8.3 版本。

```
$ ghci
GHCi, version 6.8.3: http://www.haskell.org/ghc/  :? for help
Loading package base ... linking ... done.
Prelude>
```

提示符 Prelude 标识一个很常用的库 Prelude 已经被加载并可以使用。同样的，当加载了其他模块或是源文件时，它们也会在出现在提示符的位子。

# 第 2 章　基本数据类型

**内容提要**

- ❑ 布尔类型
- ❑ 整数类型
- ❑ 浮点数类型
- ❑ 字符类型
- ❑ 字符串类型

这一章主要介绍 Haskell 的主要基本类型和相关的基本函数，并说明函数的定义方法。类型可以理解为一个集合。

## 2.1 布尔类型：Bool

布尔类型包含两个布尔值：`True` 和 `False` 。名字中的大写很重要。作用于布尔值得操作符类似于 C 语言的情况：() 表示“逻辑与”，(||) 表示“逻辑或”。

# 第 3 章　递归数据类型

# 第 4 章　规约模型及程序效率

# 第 5 章　抽象数据类型

# 第 6 章　副作用的抽象

# 第 7 章 应用 1: 并行计算

# 第 8 章  应用 2: 数据分析

# 第二部分

# 函数式程序的演算基础

# 第 9 章　基本概念

# 第 10 章　序列类型上的程序演算理论

# 第 11 章　递归函数的结构化

# 第 12 章 演算规则及其开发

# 第 13 章 应用 1: 最优化问题

# 第 14 章  应用 2: 并行自动化

# 第三部分

# 函数式算法的设计

# 第 15 章　基本概念

# 第 16 章 最小自由数问题

# 第 17 章　数独游戏

# 第四部分

# 多范式程序语言中的函数式思维

# 第 18 章 简介

# 第 19 章 OCaml

# 第 20 章 Scala

# 第 21 章 Javascript

# 第 22 章 Python