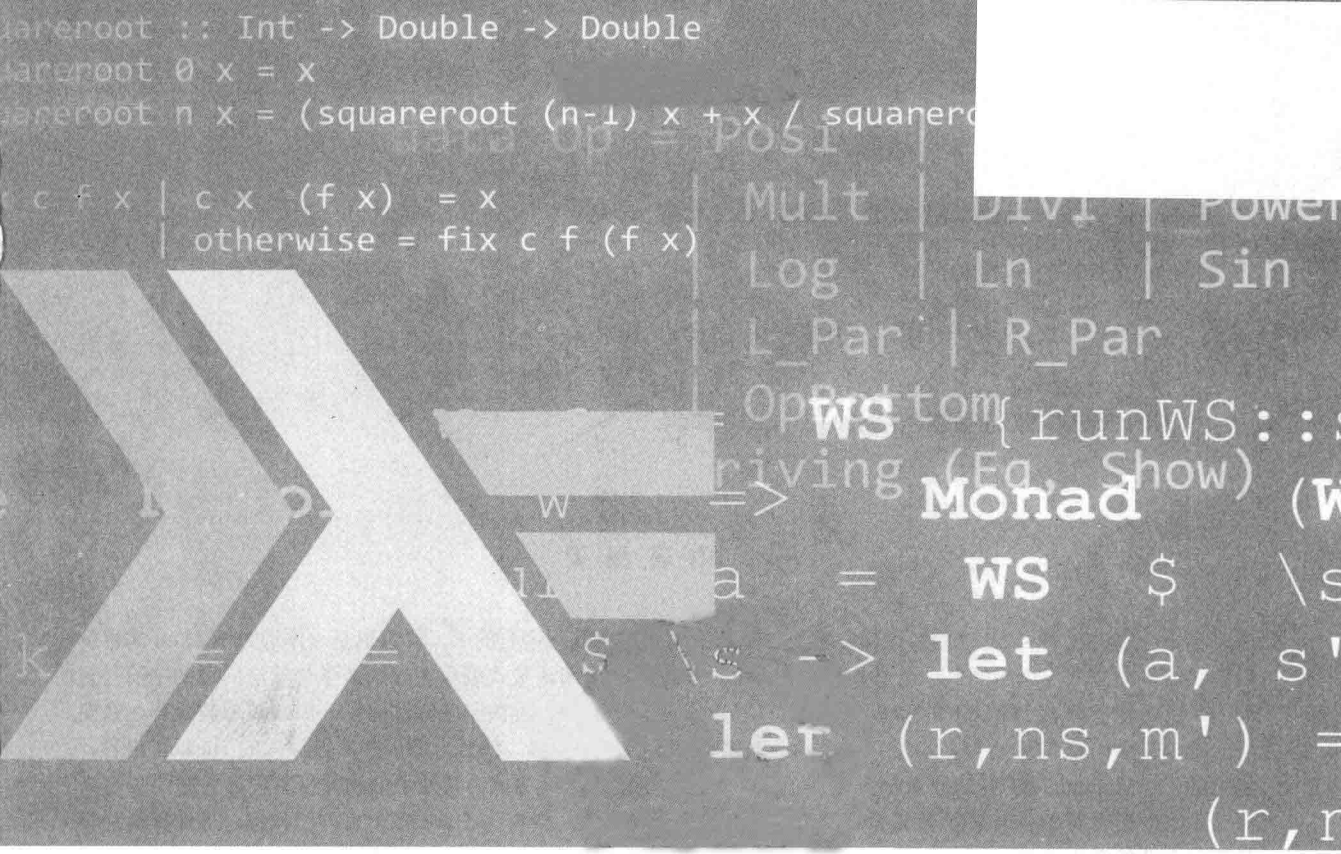
Haskell 函数式编程入门

张　淞◎编著

人民邮电出版社
POSTS & TELECOM PRESS



Haskell 函数式编程入门

张　淞◎编著

人民邮电出版社
北京

图书在版编目（CIP）数据

Haskell函数式编程入门 / 张淞编著. -- 北京 : 人民邮电出版社, 2014.3
ISBN 978-7-115-33801-3

Ⅰ. ①H… Ⅱ. ①张… Ⅲ. ①函数－程序设计 Ⅳ. ①TP311.1

中国版本图书馆CIP数据核字(2013)第277063号

内 容 提 要

这是一本讲解 Haskell 这门经过精心设计和锤炼的纯函数式编程语言的书，同时也是一本通过 Haskell 来讲解函数式编程的方法与思想的书。全书共分三个部分。第一部分介绍函数式编程在解决数学与算法问题的精简与直观的特色，让不熟悉 Haskell 的读者对其建立初步的了解，同时通过解决一些算法问题，如裴波那契数列、八皇后问题、排序问题、24 点等，引发一些对函数式编程方式的思考；第二部分介绍一些略微深入的 Haskell 内容，包括函子、Monoid、IO 与 Monad 转换器等；最后一部分则涉及快速测试、惰性求值和并行编程等主题。

本书既适合对 Hasell 和函数式编程感兴趣的程序员阅读，又适合作为 Haskell 语言入门教程，供计算机科学与数学专业的学生参考。

◆ 编　著　张　淞
责任编辑　杨海玲
责任印制　程彦红　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：23.5
字数：538 千字　2014 年 3 月第 1 版
印数：1－3 000 册　2014 年 3 月北京第 1 次印刷

定价：59.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

写给我亲爱的父母与曾经的自己。

前 言

高级计算机语言的诞生大约是在20世纪50年代。虽然那时电子计算机的发展才刚刚起步，但计算机语言的设计却逐渐分成了两个阵营。

第一个阵营的设计直接基于计算机硬件的基本结构。这些语言的理念主要是让它们对计算机的内存、磁盘以及其他硬件的存储状态进行直接的操作，如 Pascal、C 语言等。它们常常被称为命令式（imperative）、顺序式（sequential）或者过程式（procedural）编程语言。在语句执行过程中，各个语句的先后顺序十分重要。

而另外一个阵营是希望计算机语言的设计直接从数学函数的角度出发，将计算过程抽象成函数运算，这种编程方式是对程序的一种更高层次的抽象。虽然起初在运行效率上并不是很高，但表达起来十分简洁，并且设计者也在努力，逐步使这类程序编译后更适应计算机的底层硬件以获得更高的运行效率，这一类语言就是函数式编程语言。

在顺序式计算机语言的发展进程中，它们的流行程度大致可以分成两种极端情况：一部分是像 C、C++、Java、C#这样的语言，一举成名，成为现在计算机编程语言的主流；另外是一些“冷门”的语言，可能是一些失败的实验探索性语言，也可能是一些未被很好地商业化的语言。但是，函数式编程语言在顺序式编程语言取得发展并广泛应用或者消失的几十年当中，被应用的程度却一直处于不温不火的状态，虽然没有取得广泛的商业化应用，但是有关的研究和发展从未间断。而软件行业发展到今天，由于顺序式编程语言在解决一些特定问题时会引起程序复杂度倍增和可靠程度降低等问题，人们的视角渐渐投向了函数式编程语言，这使得函数式编程语言在逐步兴起。

函数式编程语言的诞生可以追溯到20世纪50年代的 Lisp。那时的计算机处理器不是很强大，函数式编程语言程序的效率要差一些，所以在当时一直没有变成主流。而如今的处理器速度的提升以及编译器的优化，使得函数式编程语言运行的效率已经不再是一个困扰性的问题。可是，顺序式语言也在发展，并且还加入了很多新的吸引人的特性，在软件工程的发展中占据了绝对的优势。即便这样，函数式编程仍然有着其他语言不可能有的优势，所以它的热度在一点一点地上升，比如微软公司就增强了自家.NET Framework 的函数式编程语言 F#的功能，并且也能在除 Windows 之外的其他操作系统上使用。除此之外，Java、C++、C#在新的版本中也引入了 λ 表达式，这个想法其实就是源于函数式编程的。其他的语言（如 Python、Ruby、Scala 等）中同样可以看到函数式编程思想的体现。

函数式编程语言有哪些呢？函数式编程语言其实也是一个很大的家族，它是一类语言，就

像命令式编程语言并不单单指 C 语言或 Java 一样，一些常见的函数式编程语言有：Lisp[①]、Scheme、Ocaml、ML(Meta-Language)、Coq[②]、Erlang[③]、Haskell、Agda[④]、F#、Clojure[⑤]等。函数式语言表达程序十分精炼，但是这些并不说明像 Java 这样面向对象的顺序式编程语言冗长，各种语言在计算机发展进程中百花齐放，各有各的长处与用途。面向对象语言有很大的优势，各种设计模式在商业开发的路上也发展得非常成熟，而函数式编程的优势在于程序的严谨与可靠性，程序正确性的证明与测试时的简易性，另外，还有开发周期相对短，编写并发程序十分简洁且运行稳定。函数式编程的应用虽然在很长时间内都处于不温不火的状态，但它们的用途却非常广泛，常见的领域有人工智能、定理证明、无线通信、金融数据分析系统等。

另外值得一提的是，在 Matlab 和 Wolfram Mathematica 这类理工科、工程学软件中，也常常能见到函数式编程的身影与思想的体现，很多基本的函数应用其实都是基于函数式编程的思想。例如，在《Mathematica Cookbook》一书中，第 2 章讲的便是 Functional Programming，即函数式编程，并且书中说学习函数式编程是掌握 Mathematica 最重要的部分。所以，学习函数式编程不仅仅是学习一种新的编程语言，而是在更好地体会另外一种编程思想，这种编程思想在日后的学习与工作中可能会尤为重要。

本书通过 Haskell 这样一门语法经过精心设计和锤炼的纯函数式编程语言来讲解函数式编程的方法与思想。虽然很多例子都是解决数学与算法问题的，但也有一些例子是对 Haskell 在实际当中应用的展示，旨在说明函数式编程语言不单单是另外一种编程方式或者只是在教学与研究中使用的语言，同时也是一门可以解决现实编程问题的、非常务实的编程语言。Haskell 实现了很多软件中的精品，如窗口管理器 XMonad、Perl 6 的 Haskell 实现 Pugs 以及高性能的网页框架 Yesod、Snap 等。

书中的例子都尽量保证以短小为主，以方便读者根据需要做跳跃、选择性的阅读。前半部分将简单介绍函数式编程在解决数学与算法问题时精简与直观的特色，使一些不熟悉 Haskell 的读者对其建立初步的了解，通过解决一些数学算法问题，比如斐波那契数列、八皇后问题、排序问题、24 点等，引发一些对这种新的编程方式的思考。

中间部分则是一些关于 Haskell 略为深入的介绍，包括函子、monoid、IO 与 monad 转换器等。其中，monad 的概念在 Haskell 中十分重要，所以建议读者可以花相对比较长的时间来实践、理解并做一些其他相关的阅读。这里值得说明的是，由于 Haskell 语言的特殊性，笔者将处理输入与输出的内容安排在了中间这一部分，也就是说，在第 11 章以前，读者将不会见到可在操作系统上直接运行的程序。

这本书最后几章的主题则是关于快速测试、惰性求值还有并行与并发编程的。这几章的内

① 被认为是首个函数式编程语言，其他函数式编程语言的鼻祖，1958 由斯坦福大学教授 John McCarthy 开发。
② 基于 Ocaml，主要用于辅助定理证明，是一个证明辅助工具（proof assistant）。
③ Erlang 是瑞典爱立信计算机实验室为开发实时、高并发、分布式系统还有提高软件安全与稳定性而研发的语言。
④ Agda 是基于 Haskell 的一门依赖类型的函数式编程语言，与 Coq 相似，也是主要用于辅助定义证明的工具。
⑤ Clojure 为 Lisp 的变种，由 Rich Hickey 开发，可运行于 Java 虚拟机之上，具有良好的可移植性。

容相对零散，但是都讨论的是关于 Haskell 的非常重要的内容。其中，快速测试一章将讨论如何使用 QuickTest 库来测试函数的正确性。测试这一课题在软件开发中非常重要而且是比较困难的，而我们会看到 Haskell 却能以一种较好的方式处理。惰性求值是 Haskell 最重要的特性之一，也就是说，在函数估值计算时，仅仅计算需要计算的部分。这在某些情况下是个好主意，但是在另外一些情形下可能会严重降低程序的效率，所以了解它并且正确地用好它很重要。最后一章将讨论如何在 Haskell 中写可以并发运行的函数。由于 Haskell 的纯函数特性，它可以随意地并发多个线程来计算纯函数，因为纯函数计算不会互相干扰。当需要共享变量时，可以使用另外一种非常好的处理并发的机制就是软件事务内存（STM）。经过多年的研究，STM 已成为了一种使用简单、运行高效、易于扩展的处理并发计算的方式。

有些章节后会有一些开放性的思考题，但并不是很难，读者可以试着独立完成，也可以在 Haskell 的中文论坛上与其他读者讨论，得到解决问题的思路或者更好的解决方法。

书的大部分是由笔者在课余时间以及假期完成的，但其中部分章节是由来自安徽的中国科技大学不愿透露姓名的博士朋友执笔撰写的。这些部分是 9.5 节的定长列表以及 9.6 节的运行时重载，还有 11.3 节中的实现 `printf` 函数。但是笔者对其原稿的一些内容做了适当的调整与修改，以保证全书风格的一致和内容的连贯与顺畅，在此感谢他的帮助与支持。

如果读者没有学过任何编程语言，那么以 Haskell 函数式语言作为第一语言就不必对思维进行转换了，我想说你非常幸运。如果读者有一些其他函数式编程语言的基础，那么想再了解一下 Haskell 那就再好不过了。了解其他顺序式语言的读者在学习 Haskell 时需要注意，由于 Haskell 中是没有内存变量和循环的，因此不会出现变量的赋值、循环。所以，如果读者已经深谙 C 语言或者 Java 等顺序式编程之道的话，那么在开始学习函数式编程时可能会产生一些不适应，毕竟没有 `for`、`while` 循环还有内存变量的编程方式不太容易一开始就能让熟练使用 C 语言与 Java 的人所接受。但是，一旦适应了 Haskell 函数式编程的这种方式，读者可能会发现这种编程方式的效率是相当高的，也有助于看清与理解程序算法的本质。另外，值得再次指出的是，由于 Haskell 的纯函数特性，输入/输出是在 IO Monad 中处理的，所以笔者决定等读者对 Monad 有了一定的了解之后再讨论，因此，在学习 IO Monad 以前，只能在解释器（GHCi）中运行程序。有些读者可能会因此感觉单调，但是了解了 IO Monad 以后，读者会发现这种设计是很有道理的。此外，有一些理论部分可能看上去有些枯燥，读者也没有在其他编程语言中涉及过，并且乍看上去感觉并没有什么用途，但是在编程实践中却恰恰相反，所以读者一定要看懂理论部分，让理论来指导编程实践。

相信读者已经感觉到本书虽然是用中文编写，但在阅读的过程当中却少不了谈论一些英文术语。有些术语是无法生硬地一对一翻译的，所以笔者给出了自己的翻译的同时保留了其英文术语，也有的术语并未做出任何翻译，因为生硬的翻译并不会帮助读者理解，所以保留了原有的英文，以便读者参阅相关的英文资料。

本书编写的目的是希望可以让越来越多的计算机专业的同学以及其他计算机行业的人只

需要花费相对较少的时间就对函数式编程有一定的了解，同时也希望能够达到抛砖引玉的效果。因为本书的编写是从笔者自己对 Haskell 和函数式编程了解不多之时开始的，希望这能成为本书的优点，因为非常熟悉了解 Haskell 的人可能很难再以一个初学者的角度来阐述它。当然，这样的缺点则使书的内容会局限于我对 Haskell 的理解程度。如果读者对有些内容有更深入的理解或者认为书中有解释不合理或者内容性的错误，欢迎发邮件同我探讨。最后，当然无论作者与编辑多么努力，总还是会有一些错误藏匿书中，希望广大读者指出。

读者有任何问题欢迎通过 Haskell.Zhang.Song@hotmail.com 与我联系。

张淞

2013 年 10 月于诺丁汉

致　谢

在这里，我想衷心地感谢诺丁汉大学和那些曾经无私地帮助过我的人们。正因为有你们的关怀与教导才使我学到越来越多关于计算机科学的知识，让我看到了计算机科学的前景并意识到了在中国有关函数式编程中文书籍的缺乏，致使我萌生了写一本关于函数式编程的书的念头，正是由于你们不断地支持我，最后促使我完成这本书的编写。

首先感谢 Graham Hutton 教授在我学习高级函数式编程课的时候给予我的详细讲解和对我本科的 Haskell 程序毕业设计的悉心指导。

感谢 JP Moresmau 先生对 EclipseFP 的无偿开发与在 EclipseFP Github 论坛上对 EclipseFP 的使用中出现的问题给予的耐心解决。

这里，我想特别感谢 Nilsson Henrik 教授在百忙之中抽出时间对本书撰写的关注、支持，以及对我在编写本书中遇到的疑问的解答。是您的编译原理和编程基础两门课让我对 Haskell 与函数式编程有了更为深入的了解和认识。同时，我也非常荣幸能经常与您在诺丁汉大学 Jubilee 校园里的 Amenities Cafe 一起讨论函数式编程和 Haskell。

感谢我的私人导师 Roland Backhouse 教授在算法课上对于一些算法构造的细致讲解，以及对于我提出的有关算法构造与复杂度问题的耐心解答，也谢谢您在生活上给予我的帮助。此外，也感谢您为我研究生申请提供的推荐信，成为您的学生我感到无比的荣幸。

感谢 Thomas Anberree 讲师的函数式编程课，正是这门课使我了解了一种全新的编程方式。此外，我很感激您对学习函数式编程有困难的学生进行了课外辅导并且给出了更多的练习题，还有您在教师公寓楼下单独为我讲解复合函数的类型。

此外，我十分感谢 Roland Backhouse 教授的学生陈炜博士，感谢您在大学课后组织的数学和算法问题的讨论课，虽然范围很宽松，涉及排列组合、命题逻辑证明、机器学习和 RSA 加密等，但是让我学到了课堂以外的更多内容。同时，我也十分高兴在平时常常能有机会和您一起运动和讨论问题。

感谢 Haskell 中文论坛 http://www.haskellcn.org 的创始人吴海生为喜欢 Haskell 与函数式编程的朋友提供了一个非常好的平台。

感谢来自中国科技大学的朋友在博士在读期间帮忙编写函数响应式编程等章节，虽然函数响应式编程一章由于结构与内容的问题最终未出现在本书中，但还是特别感谢。

最后，我还想感谢曾经教过我的老师们，他们是费继娟老师、李丹老师、李加福老师和于华民老师。谢谢您们给我的关心与帮助，伴我一点一滴地从一个孩子成长起来。谢谢您们带我走入这个充满挑战的学科，让我踏上了一条充满奇幻色彩的路。

目 录

第 1 章 Haskell 简介 ····· 1

1.1 Haskell 的由来 ····· 1
1.2 Haskell 编译器的安装以及 编写环境 ····· 3
1.3 GHCi 的使用 ····· 4
1.3.1 GHCi 中的命令 ····· 5
1.3.2 在 GHCi 中调用函数 ····· 5
1.4 .hs 和.lhs 文件、注释与库函数 ····· 7
1.5 第一个 Haskell 程序 HelloWorld! ····· 7
本章小结 ····· 8

第 2 章 类型系统和函数 ····· 9

2.1 Haskell 的类型与数据 ····· 9
2.1.1 Haskell 常用数据类型 ····· 9
2.1.2 函数类型 ····· 14
2.1.3 类型的别名 ····· 17
2.1.4 类型的重要性 ····· 18
2.2 Haskell 中的类型类 ····· 19
2.2.1 相等类型类：`Eq` ····· 20
2.2.2 有序类型类：`Ord` ····· 20
2.2.3 枚举类型类：`Emum` ····· 21
2.2.4 有界类型类：`Bounded` ····· 21
2.2.5 数字类型类：`Num` ····· 22
2.2.6 可显示类型类：`Show` ····· 25
2.2.7 小结 ····· 25
2.3 Haskell 中的函数 ····· 26
2.3.1 Haskell 中的值 ····· 26
2.3.2 函数思想入门 ····· 27
2.3.3 函数的基本定义格式 ····· 28
2.3.4 λ 表达式 ····· 30
2.3.5 参数的绑定 ····· 34
2.4 Haskell 中的表达式 ····· 35
2.4.1 条件表达式 ····· 35
2.4.2 情况分析表达式 ····· 36
2.4.3 守卫表达式 ····· 37

2.4.4 模式匹配 …… 37
2.4.5 运算符与函数 …… 38
2.4.6 运算符与自定义运算符 …… 38
本章小结 …… 41

第 3 章 基于布尔值的函数 …… 42

3.1 关键字 module 与 import 简介 …… 42
3.2 简易布尔值的函数 …… 43
3.3 与非门和或非门 …… 46
本章小结 …… 47

第 4 章 库函数及其应用 …… 48

4.1 预加载库函数 …… 48
4.1.1 常用函数 …… 48
4.1.2 基于列表的函数 …… 50
4.1.3 定义历法公式 …… 57
4.1.4 字符串处理的函数 …… 58
4.2 字符与位函数库简介 …… 60
4.2.1 Data.Char …… 60
4.2.2 Data.Bits …… 60
本章小结 …… 61

第 5 章 递归函数 …… 62

5.1 递归函数的概念 …… 62
5.2 简单递归函数 …… 64
5.3 扩展递归与尾递归 …… 66
5.4 互调递归 …… 68
5.5 麦卡锡的 91 函数 …… 69
5.6 斐波那契数列 …… 69
5.7 十进制数字转成罗马数字 …… 73
5.8 二分法查找 …… 74
5.9 汉诺塔 …… 75
5.10 排序算法 …… 78
5.10.1 插入排序 …… 78
5.10.2 冒泡排序 …… 81
5.10.3 选择排序 …… 83
5.10.4 快速排序 …… 84
5.10.5 归并排序 …… 86
小结 …… 91
5.11 递归基本条件与程序终止 …… 91
5.12 递归与不动点 …… 92
5.13 无基本条件递归和惰性求值 …… 94

本章小结 96

第 6 章 列表内包 97

6.1 列表生成器 97
6.2 素数相关趣题 99
6.3 凯撒加密 101
6.3.1 加密 102
6.3.2 解密 102
6.4 排列与组合问题 104
6.4.1 排列问题 104
6.4.2 错位排列问题 105
6.4.3 组合问题 106
6.5 八皇后问题 107
6.6 计算矩阵乘法 111
6.7 最短路径算法与矩阵乘法 112
本章小结 116

第 7 章 高阶函数与复合函数 117

7.1 简单高阶函数 117
7.2 折叠函数 foldr 与 foldl 119
7.3 mapAccumL 与 mapAccumR 函数 125
7.4 复合函数 126
本章小结 128

第 8 章 定义数据类型 129

8.1 数据类型的定义 129
8.1.1 枚举类型 129
8.1.2 构造类型 132
8.1.3 参数化类型 134
8.1.4 递归类型 138
8.1.5 杂合定义类型 140
8.2 类型的同构 142
8.3 使用 newtype 定义类型 146
8.4 数学归纳法的有效性 148
8.5 树 150
8.6 卡特兰数问题 151
8.7 霍夫曼编码 152
8.8 解 24 点 154
8.9 zipper 157
8.10 一般化的代数数据类型 159
8.11 类型的 kind 162
8.11.1 类型的 kind 162

8.11.2 空类型的声明 …… 164
本章小结 …… 165

第 9 章 定义类型类 …… 166

9.1 定义类型类 …… 166
9.2 Haskell 中常见类型类 …… 169
9.2.1 常用类型类 …… 169
9.2.2 Functor …… 171
9.2.3 Applicative …… 173
9.2.4 Alternative …… 177
9.2.5 简易字符识别器 …… 179
9.2.6 Read 类型类 …… 182
9.2.7 单位半群（Monoid） …… 182
9.2.8 Foldable 与 Monoid 类型类 …… 184
9.2.9 小结 …… 186
9.3 类型类中的类型依赖 …… 187
9.4 类型类中的关联类型 …… 192
9.5 定长列表 …… 193
9.6 运行时重载 …… 197
9.7 Existential 类型 …… 198
本章小结 …… 199

第 10 章 Monad 初步 …… 201

10.1 Monad 简介 …… 201
10.2 从 Identity Monad 开始 …… 204
10.3 Maybe Monad …… 206
10.4 Monad 定律 …… 209
10.5 列表 Monad …… 210
10.6 Monad 相关运算符 …… 210
10.7 MonadPlus …… 211
10.8 Functor、Applicative 与 Monad 的关系 …… 213
本章小结 …… 215

第 11 章 系统编程及输入/输出 …… 216

11.1 不纯函数与副作用 …… 216
11.2 IO Monad …… 218
11.3 输入/输出处理 …… 222
11.3.1 Control.Monad 中的函数 …… 222
11.3.2 系统环境变量与命令行参数 …… 224
11.3.3 数据的读写 …… 225
11.3.4 格式化输出 printf 函数 …… 228
11.3.5 printf 函数的简易实现 …… 229

11.4 星际译王词典……233
11.4.1 二分法查找……234
11.4.2 散列表的使用……237
11.5 简易异常处理……239
11.6 Haskell 中的时间……244
本章小结……245

第 12 章 记录器 Monad、读取器 Monad、状态 Monad……246

12.1 记录器 Monad……246
12.1.1 `MonadWriter`……248
12.1.2 记录归并排序过程……249
12.2 读取器 Monad……250
12.2.1 `MonadReader`……251
12.2.2 变量环境的引用……252
12.3 状态 Monad……253
12.3.1 状态 Monad 标签器……254
12.3.2 用状态 Monad 实现栈结构……255
12.3.3 状态 Monad、`FunApp` 单位半群和读取器 Monad 的关系……257
12.3.4 `MonadState`……258
12.3.5 基于栈的计算器……258
12.4 随机数的生成……270
本章小结……271

第 13 章 Monad 转换器……273

13.1 从 `IdentityT` Monad 转换器开始……273
13.2 Monad 转换器组合与复合 Monad 的区别……276
13.3 Monad 转换器的组合顺序……278
13.4 `lift` 与 `liftIO`……281
13.5 简易 Monad 编译器……282
13.6 语法分析器 Monad 组合子……286
13.6.1 简易语法分析器的实现……287
13.6.2 Parsec 库简介……291
13.6.3 上下文无关文法……296
13.6.4 基于语法分析器的计算器……300
本章小结……304

第 14 章 QuickCheck 简介……305

14.1 测试函数属性……305
14.2 测试数据生成器……308
本章小结……310

第 15 章 惰性求值简介……311

15.1 λ 演算简介……311

15.2 ⊥Bottom ····· 313
15.3 表达式形态和 thunk ····· 314
15.3.1 WHNF、HNF 与 NF ····· 314
15.3.2 thunk 与严格求值 ····· 315
15.4 求值策略 ····· 319
15.4.1 引值调用 ····· 319
15.4.2 按名调用 ····· 320
15.4.3 常序求值 ····· 320
15.5 惰性求值 ····· 321
15.6 严格模式匹配与惰性模式匹配 ····· 322

第 16 章 并行与并发编程 ····· 324

16.1 确定性的并行计算 ····· 325
16.2 轻量级线程 ····· 333
16.2.1 调度的不确定性 ····· 333
16.2.2 基本线程通信 ····· 334
16.2.3 信道 ····· 337
16.2.4 简易聊天服务器 ····· 337
16.3 软件事务内存 ····· 341
16.3.1 软件事务内存简介 ····· 341
16.3.2 软件事务内存的使用 ····· 343
16.3.3 哲学家就餐问题 ····· 347
16.3.4 圣诞老人问题 ····· 350
16.4 异步并发库简介 ····· 355
本章小结 ····· 357

参考文献 ····· 358

后记 ····· 359

第 1 章

Haskell 简介

本章首先介绍 Haskell 相关的历史，从 Lisp 诞生到多种函数式编程语言百花齐放，再到 Haskell 诞生的过程和现在发展的概况；接下来讲解安装 Haskell 编译器和编写 Haskell 程序所需要的软件，以及调试与测试函数的工具 GHCi；然后介绍 Haskell 中定义的两种源代码文件（一个是以 lhs 为扩展名，另外一个是以 hs 为扩展名）有着怎样的不同；最后编写一个 Helloworld 程序当做学习 Haskell 旅程的第一步。

1.1 Haskell 的由来

要讲述 Haskell 的由来还要从函数式编程的诞生说起。函数式编程有着非常悠久的历史，比 C 语言还要久远。20 世纪 30 年代，美国数学家 Alonzo Church 引入了 λ 演算（Lambda Calculus），这是一个通过使用符号表达变量的绑定和替换来定义函数计算的系统，它是函数式编程语言的重要基石。也就是说，早在电子计算机还没有诞生的 20 世纪 30 年代，函数式编程语言就已经在孕育之中了。

1958 年，斯坦福大学的教授 John McCarthy 受卡内基梅隆大学开发的一个名为 IPL（Information Processing Language）语言的影响，开发了一个名为 Lisp 的函数式编程语言。虽然 IPL 并不是严格意义上的函数式编程语言，但是它已经有了一些基本思想，如列表内包、高阶函数等，本书会在后面的章节中依次做介绍。这些特性深深地影响了 Lisp 的设计。在 Lisp 诞生之后，越来越多的人开始加入到函数式编程的阵营中来。同时，Lisp 的诞生也影响了很多编程语言的设计，时至今日，仍然有相当一部分人在使用 Lisp。

在 20 世纪 60 年代，牛津大学的 Peter Landin 和 Christopher Strachey 明确了 λ 演算对函数式编程具有极高的重要性，并于 1966 年开发出了一个名为 ISWIM（If you See What I Mean）的函数式编程语言，这是一个基于 λ 演算的纯函数式编程语言，一定程度上奠定了函数式编程语言设计的基础。

函数式编程发展到 20 世纪 70 年代早期的时候，爱丁堡大学的 Rod Burstall 和 John Darlington

使用函数式编程语言通过模式匹配来进行程序变换，并且明确地引入了列表内包的语法来更好地生成列表。几乎在同一时期，David Tuner 教授在英国圣安德鲁斯大学，又开发了一个名为SASL（St Andrews Static Language）的纯函数式编程语言，一定基础上奠定了圣安德鲁斯大学在函数式编程领域研究的地位。

1975年，麻省理工学院的Gerry Sussman与Guy Steele开发了Scheme，这是一个更接近λ演算的语言，广泛应用于实践编程与教学中。如今，Scheme仍然被很多高校作为首要的课程来讲解。

不久，美国计算机科学家John Backus致力于函数式编程的研究，开发了一种名为FP的函数式编程语言。他引入了BNF符号系统以及对语言编译器系统的开发做出了突出的贡献，于1977年获得了图灵奖——他著名的图灵奖报告《编程是否能从冯诺依曼的体系风格中解放出来》（Can Programming be Liberated from the von Neumann Style?）[①]中提到的“解放”的方式，实际上指的就是函数式编程。

也几乎是在同一时期，剑桥大学的Robin Milner开发了ML（Meta-Language），并发展出了多态类型系统——HM-System（Hindley-Milner type system），其中包括类型推断以及类型安全与异常处理机制。这个语言中的多态类型系统对函数式编程的发展意义重大，值得一说的是，Haskell使用的正是这种类型系统。

Scheme与ML都具有一定的顺序式语言的特性，但是这些语言改善了函数式编程的风格并且明确地引入了高阶函数的概念。20世纪70年代末与80年代初，惰性求值这一概念被重新被发展。SASL在1976年加入了惰性求值的功能，也出现了Lazy ML，即ML的惰性版本。在程序的运行过程中，惰性求值可使计算机仅仅计算需要的数据。

在近几十年的时间里，各个大学的学者们通过对函数式编程的研究，涌现出了很多基于函数式编程的理论。当然，与此同时还出现了很多不同的函数式编程语言。1987年，在美国俄勒冈州举行的函数式编程与计算机结构的会议（Functional Programming and Computer Architecture Conference，FPCA）上，与会者们就当时函数式编程语言种类过多、语法相似且大多数效率不高的现状进行了讨论。他们认为，这样下去的结果将会是越来越多的人研究和使用函数式编程语言，但是人们所使用的语言却得不到很好的统一。这种情形不利于函数式编程的研究、应用与发展。于是会议决定设计一个开源、免费、语法简单、运行稳定、效率高，可适用于函数式编程教学、研究，并且可编写应用软件的纯函数式编程语言来缓解函数式编程语言过多的混乱的局面，它的名字就是Haskell。它的命名源于美国数学家Haskell Brooks Curry，为纪念他在λ演算与组合逻辑（combinatory logic）方面作出的突出贡献。Haskell Curry使得函数式编程语言的设计在理论上有了非常坚实的基础。由此可见，Haskell是函数式编程发展了近60年的结晶，汇集了其他函数式编程语言的精华于一身，经过了20余年的发展，如今还在不断地壮大。

Haskell仅仅意为一门计算机编程语言，语言标准的1.0版本于1990年发布，语法的主要设计者

① 链接 http://www.thocp.net/biographies/papers/backus_turingaward_lecture.pdf。