

TURING

图灵原创

魔力Haskell

韩冬◎著

Magic Haskell

前滴滴出行首席架构师、现美洽网总裁兼CTO李令辉作序推荐

- 贴近Haskell前沿，兼顾理论和实践的最佳参考书
- 揭秘单子变换、模板编程和泛型编程等特性
- 解答异常处理、网络编程、数据库操作等方面的常见问题



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS

TURING

图灵原创



魔力Haskell

韩冬◎著



人民邮电出版社
北京

Magic Haskell

图书在版编目 (CIP) 数据

魔力Haskell / 韩冬著. — 北京 : 人民邮电出版社,
2016.9
(图灵原创)
ISBN 978-7-115-43283-4

I. ①魔… II. ①韩… III. ①函数—程序设计 IV.
①TP311.1

中国版本图书馆CIP数据核字(2016)第193195号

内 容 提 要

本书是一本由浅入深的 Haskell 教程。书中首先介绍 Haskell 的基础语法和函数式编程的基本概念,以及 GHC、GHCi、cabal 等工具的用法;接着按照函子→应用函子→单子的顺序介绍 Haskell 中核心的三大类型类,并以列表单子、Reader 单子和 State 单子为例详细分析单子类型类的来龙去脉;最后介绍 Foldable 和 Traversable 类型类、单子变换、GHC 的语言扩展和程序标注,以及网络编程、数据库、并发和并行、序列化/反序列化与泛型编程、异常处理等内容。

不管你是刚刚开始学习编程的计算机爱好者,或是有一定编程经验的从业人员,还是对函数式编程已经有一些了解但希望进一步提高的进阶读者,在本书中都能找到你想要的内容。

-
- ◆ 著 韩 冬
责任编辑 王军花
责任印制 彭志环
 - ◆ 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>
北京天宇星印刷厂印刷
 - ◆ 开本: 800×1000 1/16
印张: 22.75
字数: 538千字
印数: 1—3 000册
- 2016年9月第1版
2016年9月北京第1次印刷
-

定价: 79.00元

读者服务热线: (010)51095186转600 印装质量热线: (010)81055316

反盗版热线: (010)81055315

广告经营许可证: 京东工商广字第 8052 号

序

虽然平时主要的工作都是用命令式语言完成的,但这并没有影响我成为一名Haskell的忠实粉丝,或者说函数式编程的粉丝。函数式编程作为另一类重要的编程模型,无论是在解决问题的大方向上,还是针对具体问题的具体思路上,对程序员都非常有帮助。即使在使用命令式编程语言的过程中,这些帮助也很有意义。如果你不会函数式编程,你可能终究无法成为一个更好的程序员(这和你是否需要用函数式编程工作没有关系),比如map-reduce框架的灵感就来自函数式编程语言,Erlang的分布式编程模型也利用了很多诸如不可变数据、高阶函数等函数式编程的特性。

在函数式编程中,我最喜欢的语言就是Haskell。Haskell从语言设计到对实际编程问题的建模,都带有那种让人心旷神怡的美。Haskell出身于学术界,包含了很多计算机语言方面前沿的、实验性的想法,是各种语言特性的试验田,学习Haskell是对程序员的内涵和品位的一个很好的提升。

但是学习Haskell其实很不易,常常导致对Haskell感兴趣的人无从下手,我个人也读过很多图书和教程,但是没有哪本是上手门槛特别低的。对于程序员来说,能对照着理论快速实践的图书比较容易学习,韩冬同学的《魔力Haskell》就是这样一本读起来轻松愉快、很有亲和力的图书,书中提供了大量实践来配合理论,学习起来没有太大压力。不像其他Haskell图书,这里不会用高不可攀的名字吓坏你,循序渐进,不知不觉的你就成了Haskeller。希望作为读者的各位也可以在学习编程知识的过程中,体会Haskell的美。

另外,出于种种原因,你可能之前学过Haskell但是未必能直接应用到工作里,这本书给出了作者本人的大量编程实践,希望它能起到抛砖引玉的作用,让你在工作中充分享受函数式编程的乐趣!

李令辉

前滴滴出行首席架构师,现美洽网总裁兼CTO

前言

从未有过一门编程语言像 Haskell 这般打动我：

Elegance is not optional.
优雅是不可或缺的。

—— Richard O’Keef

以致我打算写一本关于编程的书，计划主题涉及 Haskell、范畴论以及如何使用它们解决实际的编程问题。此外，写作本书的原因还有：

- Haskell 的进化速度太快，关于 Haskell 的资料多少都有些过时了，例如著名的 RWH、LYAH，很多东西都已经不适用；
- 国内关于 Haskell 的资料太少；
- 范畴论是程序员解决问题的有力工具，可是很多关于范畴论的文章都太过学术；
- Haskell 是一门十分有趣、有用的语言，了解的人太少实在可惜。

综上，这本书力图能够给喜欢编程的读者带去很多有趣、有用的东西，让读者在工作中享受 Haskell 和范畴论的美妙。

书名一方面来源于：

Any sufficiently advanced technology is indistinguishable from magic.
先进的科技无异于魔法。

—— Arthur C. Clarke

另一方面是因为熟悉 Haskell 的人都承认它是一门充满了魔法咒语的语言。另外，我很喜欢电影《魔力麦克》。

本书分为三部分：基础知识、重要的类型和类型类、高级类型类和项目实践，是一门由浅入深的 Haskell 教程。

第一部分主要介绍 Haskell 的基础语法和函数式编程的基本概念，以及 GHC、GHCi、cabal 等工具的用法。

第二部分按照函子→应用函子→单子的顺序介绍 Haskell 中核心的三大类型类，并以列表单子、Reader 单子和 State 单子为例详细分析单子类型类的来龙去脉。

最后一部分主要介绍最新加入 Haskell 的 Foldable 和 Traversable 类型类、单子变换、GHC

的语言扩展和程序标注，以及在网络编程、数据库、并发和并行等方面的一些实例，希望能给读者带去很多有用的参考。

因此，本书面向的读者面也较为广泛，不管你是刚刚开始学习编程的计算机爱好者，或是有一定编程经验的从业人员，还是对函数式编程已经有了一些了解但希望进一步提高的进阶读者，我相信在本书中都能找到你想要的内容。不过这里需要提醒的是，很多拥有其他语言经验的程序员在刚刚学习一门新语言时，往往喜欢从案例入手，粗略看了几眼基本语法后就直接进入自己熟悉的领域，分析案例，例如十分钟写出一个 Web Server，半小时写出一个 GUI 记事本等，然后根据之前自己的经验来消化新语言的语义和使用技巧。因为以他们以往的经验来看，不同的语言无非就是缩进和括号这类细枝末节的语法不同，或者某些特性支持与不支持的区别。但我强烈建议学习 Haskell 的时候，把之前的语言经验统统忘记，因为这是一门非常“不一样”的语言。

- Haskell 是纯函数式编程语言，所有的函数只要传递的参数相同，计算的结果也一定相同，即使是和外界产生交互的函数，例如读取用户输入 `getLine`，我们会为了和外界交互创造一整个世界。
- Haskell 没有任何的控制结构声明，例如你所熟知的 `for` 和 `while`，但却拥有许多强大的结构控制函数，可以方便地表达复杂的确定性、非确定性、同步、异步等计算过程。
- Haskell 没有可以改变的变量，却可以实现非常复杂的状态转换。把 `State` 的操作用强类型标记，可以从编译阶段杜绝大量 bug。另外，本书将使用“绑定”一词取代其他资料中的变量，以减少歧义。
- Haskell 的抽象能力非常强，所以要求你理解很多抽象的概念。但这不是一件坏事，并且当你熟练掌握它们之后，它们可以帮你节省很多无用的代码。而且相比其他语言，你可能会把大部分时间都花在细枝末节的处理上。

人们常常把 Haskell 和另一门远古魔法 Lisp 作比较。作为一个出现较晚的函数式编程语言，Haskell 从数学界引入了大量强力的概念，这使得它异常严谨，每一个层次的抽象都建立在坚固的理论基础之上。

所以每一个在你看来很简单的概念，都会在未来更加庞大的概念中出现，千万不要因为它们看起来没用而忽略它们，这会导致你快速翻阅到后面章节时，因为错过太多简单的概念而无法理解后续出现的精髓，等到实际应用时，你会觉得 Haskell 难以使用。

本书会提供很多 Haskell 的实际应用，因为本人工作的原因，这里提供的应用实例大部分将集中在网络编程方面。也希望读者能够耐心阅读，慢慢体验 Haskell 的优雅。当你读完本书后，十分钟写出一个 Web Server 之类的事情将完全不是问题。

此外，本书还提供了源代码供读者下载，详情可参见 <http://magic-haskell.com/> 或者图灵社区本书主页 <http://www.ituring.com.cn/book/1882>。

感谢图灵公司，尤其是王军花编辑对我的大力支持，没有你们的努力，本书不可能和读者见面。也感谢在美团所有和我一起共事过的同事，我的两个前端组 Leader 潘魏增、夏娇娇，之前百度的 Leader 陆泰宁，没有你们的帮助，本书不可能成稿。特别感谢周围在我写作期间给予的关心和帮助，你是我不懈努力的动力。

目 录

第一部分 基础知识

第 1 章 基本语法和 GHCi	3
1.1 注释	3
1.2 表达式	3
1.3 声明	4
1.3.1 类型声明和绑定声明	4
1.3.2 模块声明和导入声明	5
1.4 函数	6
1.5 GHCi	8
1.6 初级函数	10
第 2 章 data 和模式匹配	13
2.1 数据声明 data	13
2.2 模式匹配	14
2.2.1 无处不在的模式匹配	15
2.2.2 @pattern	16
2.3 各式各样的数据类型	16
2.3.1 多构造函数	16
2.3.2 完备性检查	17
2.3.3 无参数构造函数	18
2.3.4 data 与类型变量	18
2.3.5 记录语法	20
2.4 排版规则	22
第 3 章 列表、递归和盒子比喻	23
3.1 列表	23
3.1.1 等差数列	24
3.1.2 匹配列表	24
3.2 递归操作	26
3.3 盒子比喻	30

第 4 章 元组、类型推断和高阶函数	34
4.1 元组	34
4.2 类型推断	35
4.3 高阶函数	37
4.3.1 拉链和 zipWith	39
4.3.2 柯里化	41
第 5 章 常用的高阶函数和函数的补充语法	43
5.1 应用函数 \$ 和 &	43
5.2 匿名函数	44
5.3 组合函数	45
5.4 函数的补充语法	46
5.4.1 where	46
5.4.2 guard	47
5.4.3 MultiWayIf	49
5.4.4 where 与 let	49
5.5 Point free	49
5.6 黑魔法词汇表	51
第 6 章 常用的列表操作：映射、过滤、 折叠和扫描	53
6.1 映射	53
6.2 过滤	54
6.3 折叠	55
6.4 扫描	57
6.5 方向是相对的	60
第 7 章 类型类	62
7.1 实例声明	62
7.2 类声明	65
7.3 类型类的实现	66

7.3.1 层级和约束	67	12.4 函数库	128
7.3.2 推导类型类	68	第 13 章 应用函子	129
7.3.3 Show/Read	69	13.1 函子的局限	129
第 8 章 数字相关的类型类	71	13.2 什么是函子	133
8.1 顺序类	71	13.2.1 Reader 应用函子	134
8.2 data 和类型约束	72	13.2.2 自然升格	135
8.3 枚举类	73	13.3 IO 应用函子	138
8.4 边界类	75	第 14 章 单位半群和一些有趣的应用	
8.5 数字类	76	函子	139
第 9 章 type、newtype 和惰性求值	80	14.1 单位半群	139
9.1 类型别名 type	80	14.1.1 Endo 单位半群	141
9.2 新类型声明 newtype	81	14.1.2 自由单位半群	142
9.3 惰性求值	84	14.1.3 逆	144
9.3.1 标记语义、常态和弱常态	87	14.2 当单位半群遇上应用函子	145
9.3.2 seq 和 deepseq	90	14.2.1 Const a 的应用函子实例	145
第 10 章 模块语法以及 cabal、Haddock		14.2.2 选择应用函子	146
工具	94	14.2.3 拉链应用函子	147
10.1 模块语法	94	第 15 章 解析器	149
10.2 使用 cabal	96	15.1 参数解析	149
10.2.1 使用 cabal 安装依赖	96	15.2 optparse-applicative	151
10.2.2 项目的 cabal 配置	98	15.3 选择解析	153
10.3 Haddock	101	第 16 章 单子	158
第二部分 重要的类型和类型类		16.1 应用函子的局限	158
第 11 章 函子	107	16.2 什么是单子	159
11.1 容器抽象	107	16.2.1 粘合函数	161
11.2 范畴	110	16.2.2 do 语法糖	163
11.3 Identity 和 Const	115	16.3 IO 单子	165
11.4 IO 函子	117	第 17 章 八皇后问题和列表单子	168
第 12 章 透镜组	118	17.1 列表单子与数组归纳	168
12.1 getter 和 setter	118	17.2 八皇后问题	169
12.2 透镜组	120	17.3 MonadPlus	171
12.3 view、set 和 over 函数	122	17.4 结构控制函数	173
12.3.1 over 函数	122	17.4.1 sequence/sequence_	173
12.3.2 set 函数	124	17.4.2 mapM/mapM_	174
12.3.3 view 函数	125	17.4.3 replicateM/replicateM_	175
		17.4.4 forever	176

17.4.5 filterM	176	第 24 章 单子变换	245
17.4.6 foldM/foldM_	177	24.1 Kleisli 范畴	245
第 18 章 Reader 单子	179	24.2 ReaderT	246
18.1 (->)a 的单子实例声明	179	24.3 Identity 和 IdentityT	248
18.2 模板渲染	180	24.4 StateT	250
18.3 Reader 新类型	185	24.5 RandT	253
第 19 章 State 单子	187	第 25 章 单子变换的升格操作	256
19.1 什么是 State 单子	187	25.1 MonadIO	258
19.2 随机数	192	25.2 MonadState 和 MonadReader	258
19.3 简易计算器	195	25.3 类型家族	260
第 20 章 IO 和它的伙伴们	197	25.4 Lazy StateT 和 Strict StateT	262
20.1 IO 单子的本质	197	25.5 Writer 单子	263
20.2 基本 IO 操作	199	第 26 章 高效字符串处理	266
20.3 IO 中的变量	202	26.1 bytestring 函数库	266
20.4 forkIO	203	26.1.1 Lazy ByteString	267
20.5 ST 单子	204	26.1.2 ByteString Builder	268
20.6 后门函数	206	26.2 text 和 utf8-string 函数库	272
第 21 章 语言扩展和程序标注	211	26.3 mono-traversable 函数库	274
21.1 语言扩展	211	第 27 章 网络编程	276
21.2 严格求值数据项	214	27.1 wai/warp	276
21.3 惰性模式	214	27.2 wai-extra	279
21.4 程序标注	215	27.3 HTTP 的单子抽象	280
21.5 编译选项	217	27.4 WebSocket 编程	281
21.6 运行分析	218	27.5 Socket 编程	283
第 22 章 Foldable 和 Traversable	221	第 28 章 Haskell 与数据库	286
22.1 Foldable	221	28.1 persistent	286
22.2 折叠与单位半群	224	28.2 esqueleto	292
22.3 Traversable	226	第 29 章 模板编程	296
22.4 推导规则	230	29.1 什么是模板	296
22.5 Data.Coerce	231	29.2 Q 单子	299
第 23 章 列表、数组和散列表	233	29.3 拼接	301
23.1 列表	233	29.4 reify	303
23.2 数组	236	第 30 章 并发和并行编程	306
23.3 散列表	242	30.1 运行时工作原理	306
		30.2 并行编程	308
		30.3 并发编程	310
		30.3.1 MVar	311

30.3.2	STM	314	32.1.2	使用泛型提供的 ToJSON/FromJSON	335
30.3.3	async	316	32.2	泛型	336
第 31 章	高级类型编程	317	第 33 章	Haskell 中的异常处理	342
31.1	Typeable 和 Dynamic	317	33.1	使用 Either/Maybe 表示异常	342
31.2	存在类型	320	33.2	运行时异常	343
31.3	类型家族、数据家族和 GADT	321	33.2.1	异步异常	346
31.3.1	类型家族	321	33.2.2	资源的清理和释放	348
31.3.2	数据家族	324	33.3	纯函数中的异常处理	349
31.3.3	GADT	325	33.4	异常和单子变换	350
31.4	数据类别 DataKinds	327	33.4.1	exceptions	350
第 32 章	序列化/反序列化与泛型编程	331	33.4.2	monad-control	351
32.1	aeson 函数库	331	33.5	常见的异常处理问题	353
32.1.1	使用模板编程自动生成 ToJSON/FromJSON 实例	334			

Part 1

第一部分

基础知识

第 1 章到第 10 章是本书的第一部分，内容包括 Haskell 的基本语法、函数式编程的基本概念以及常见的数据类型和类型类，主要面向之前从未接触过 Haskell 或者想要全面了解 Haskell 语言的读者。在继续阅读本部分内容之前，建议读者先配置好计算机上的 Haskell 环境。书中的示例都是以 Haskell 的官方实现 GHC 为基础编写的，下面简要介绍一下 GHC 的安装步骤。

从 Haskell 的官方网站 (<https://www.haskell.org/downloads>) 获得 Haskell 语言最主要的实现 GHC (The Glasgow Haskell Compiler)。

如果没有其他原因，推荐下载包含 GHC 和 cabal 的 Minimal installers。安装完成之后，命令行里会出现以下几个命令。

- ❑ **GHC**。这是 GHC 的可执行文件，你可以直接调用它来编译 Haskell 源代码文件 (*.hs)。
- ❑ **GHCi** (GHC's interactive environment)。类似 python 和 irb，这是 GHC 的交互命令模式，方便在开发过程中快速查看、求值以及试验等，用过 Lisp 实现 REPL 的同学会发现很熟悉。在后面的章节中，GHCi 将会是非常有用的试验工具。
- ❑ **runHaskell**。快速执行 Haskell 文件。假设你的 runHaskell 路径是 /usr/local/bin/runHaskell，则可以在 Haskell 源代码文件顶部加上 shebang 符号（即 #!，这是 Unix 系统中帮助操作系统找到执行文件的注释）：

```
#!/usr/local/bin/runHaskell
```

把 Haskell 当作脚本使用。

- ❑ **cabal**。这是 Haskell 社区使用最为广泛的包管理工具，同时身兼自动化编译工具的功能，所以 Haskell 项目不需要使用传统的工具 make。

Haskell 社区最大的开源仓库非 Hackage 莫属：

<http://hackage.haskell.org>

你可以使用上面说到的 cabal 下载并安装 Hackage 上的代码库。在 Linux 下，很多包管理工具会提供 Haskell 的库，不要轻易使用它们。因为它们会自动安装到全局空间，带来很多不必要的麻烦。如果书中需要用到额外的库，除非是很常用的库，其他情况下默认安装到项目空间，这在之后的章节中会说明。

目前，主流的编辑器对于 Haskell 提供的支持都很好。

- ❑ Emacs 的用户推荐安装 Haskell-mode 插件。
- ❑ Vim 用户推荐安装的插件有 syntastic、vim-hdevtools 或者 ghcmmod-vim、hlint 等。除了安装插件，vim-hdevtools 或者 ghcmmod-vim 还需要安装对应的程序并将其放置在 path 下。当然，对于初学者来说，这些都不是必需的，Vim 对 Haskell 原生的支持就很好了。
- ❑ Sublime 用户可以安装强大的 SublimeHaskell 插件。当然，如果需要使用这个插件的额外功能，也需要安装 ghc-mod 等程序。

遵循所有编程语言的惯例，先新建一个 Main.hs 文件，其内容如下：

```
main :: IO ()
main = print "hello world"
```

然后使用 runHaskell 执行它：

```
runHaskell Main.hs
```

此时你的屏幕上显示出 hello world 了吗？如果没有，请检查下你的电脑是不是坏掉了。

最后要提的是，有一本最好的 Haskell 参考书，你一定要打印出来，遇到不清楚的地方务必前去查阅，它就是“the Haskell Report”！最新的一版是 Haskell 2010。

- ❑ 在线阅读：<https://www.haskell.org/onlinereport/haskell2010/>。
- ❑ 下载打印：<https://www.haskell.org/definition/haskell2010.pdf>。

记得一定要打印，和本书一起放在你的枕边，因为这是 Haskell 最官方、最全面的参考书！

第 1 章

基本语法和GHCi



Haskell的语法以简洁、优雅著称，但是和Lisp的极简语法不同，它最强调的是可读性，其次才是语法结构的简洁性。很多语法的设计都以牺牲一定的简洁性来提高可读性，所以阅读Haskell代码往往给人一种很干净的感觉，本章呈现给读者的，就是Haskell的基本语法结构。

GHCi是GHC的交互模式，在其中输入一些代码，可以即时获得运行结果。此外，GHCi还提供了一些特殊的命令来帮助我们分析正在调试的代码。本章主要介绍一些基本的使用方法。

1.1 注释

我们先从和其他语言相同的地方说起。首先，好的程序员一定要会写注释，Haskell中的注释长这个样子：

```
-- 这是一行注释
{-
    这是一段注释
-}
```

和大多数编程语言一样，Haskell程序包含表达式（expression）和声明（declaration）两种类型。

1.2 表达式

表达式就是有值（value）的式子，例如：

```
1
True
x + 4
3 :: Float
sort [3,8,1,4]
case Foo of True -> 1
           False -> 2
```

下面简述上述代码的作用。

□ `3 :: Float`表示的是一个浮点数3，`::`在这里说明了3的类型是Float。

□ `sort`是排序函数，`sort [3,8,1,4]`是一次函数调用，该表达式的值是排序后的列表。

这里需要说明的是，和大部分语言不同，Haskell中的函数调用不需要加括号，多个参数之间也不用加逗号，这个以后还会解释。

- `case ... of ...` 是一个完整的表达式。在上面的例子中，最后一个表达式的值在 `Foo` 是 `True` 的时候等于1，否则等于2。

值得注意的是，在其他语言中作为控制结构的声明，例如 `if...then...else...` 和 `case...of...`，在Haskell中大部分都是表达式。

有时候，一个表达式由好几个部分组成，我们可以使用 `let...in...` 来把复杂的表达式拆分成若干部分：

```
let x = 3 * a
    y = 4 * a
in sqrt (x ^ 2 + y ^ 2)
```

整个表达式的值等于把 `x`、`y` 带入 `sqrt (x ^ 2 + y ^ 2)` 的值。此外，`let...in...` 还有一个作用，那就是把复杂表达式中需要重复计算的部分提取出来。假设刚刚的表达式中还出现了若干个 `3 * a`，我们只需要计算一次 `x = 3 * a` 即可。

1.3 声明

仅仅有表达式还不足以表达程序的结构，一个完整的Haskell程序是由若干声明组成的。Haskell的声明和其他语言不太一样，其中有很多种类型的声明，这里先讲几个基本类型的声明。

1.3.1 类型声明和绑定声明

Haskell是一门强类型语言，所有的表达式都有确定的类型，你可以使用 `::` 手动添加类型标注：

```
addOne :: Int -> Int
addOne x = x + 1

welcomeMsg :: String
welcomeMsg = "hello world"
```

上面的例子中，`welcomeMsg :: String` 读作“`welcomeMsg`的类型是`String`”。`=`则是把右侧的表达式绑定（binding）到左侧的名称上，在代码其余的地方，每当你需要使用“hello world”的时候，使用 `welcomeMsg` 即可。

和其他语言中声明/初始化变量不同，在Haskell中一旦一个名称被绑定给了表达式，这个名称包含的表达式就永远不会再变了。Haskell中不存在变量，只存在绑定，任何时刻一个名称对应的表达式都是唯一确定的。

如果`=`左侧的名称有多个，例如 `addOne x = x + 1`，那么这是一个函数绑定，第一个名称 `addOne` 是函数，后面的 `x` 是参数的模式（pattern）。关于函数绑定，后面还会详细介绍，现在把函数绑定左侧的部分，理解为函数加参数列表即可，`addOne x = x + 1` 基本上和下面的C语言代码等价：

```
int addOne(int x){
```

```
    return(x+1);
}
```

在代码其余的地方，如果需要计算 $3 + 1$ ，只需调用`addOne 3`即可。

其实在上面说到的`let ... in ...`语法中，`let`和`in`之间的部分也都是绑定，这些绑定只在`in`后面的表达式里可见，在整个`let`之外不可见：

```
x = 1

let x = 3 * a
    y = 4 * a
in ... -- 此处 x = 3 * a

y = x + 1
-- 此处 y = 1 + 1 == 2
-- 此处 x 的绑定仍然是 x = 1
```

代码中可以看见`x`的区域叫作`x`的作用域（scope）。在Haskell中，一个绑定的作用域可以根据代码的词法（lexical）结构（例如`let ... in ...`）分析出来，所以说Haskell的绑定遵循词法作用域（lexical scope）：

```
let x = 1
in let y = x * 2 -- 这里可以看见x
    in x + y -- 这里既可以看见x，也可以看见y
```

词法作用域其实就是指在嵌套的语法结构中，被嵌套的代码段可以看到外层代码段的绑定。

1.3.2 模块声明和导入声明

下面的代码定义了一个模块`Main`。在该模块内部，我们导入了模块`Data.List`，然后定义了`main`函数，它的类型是`IO ()`（关于这个类型，还需要好几个章节才能理解，先不管它）。`main`函数使用`Data.List`中的函数`nub`去除列表中的重复元素，并把它打印出来。和C语言一样，`main`函数也是整个程序的入口：

```
module Main where

import Data.List

main :: IO ()
main = print (nub [1,2,3,2,3])
```

在把模块A导入到自己的模块之后，模块A中所有的函数在我们的模块中都是可见的。如上面说到的，Haskell中所有的名称都是唯一的，所以在上面的例子中，你无法再定义一个`nub`函数。关于模块和导入，还有很多高级语法来解决命名冲突问题，将在后面介绍。

现在你可能会奇怪`print`函数是在哪里定义的？事实上，为了方便书写，Haskell默认会在所有模块中导入一个叫作`Prelude`的模块，这个模块包含许多常用函数。Haskell中许多重要的模块（包括`Prelude`）一起构成了base库，这套函数库会随着Haskell编译器一起发行。只要有可以工作的GHC，就可以直接导入base中的所有模块，它的文档放置在Hackage上：<http://hackage.haskell>。

org/package/base。

现在看不懂Hackage的文档没关系，大概感受一下就好，之后你会慢慢发现Hackage上的文档是你最好的帮手。现在需要牢记的几件事情如下。

- Haskell中没有变量和赋值，只有绑定，绑定不能改变。
- Haskell中没有条件、循环、分支等控制声明，条件和分支在Haskell中是表达式的一部分。

1.4 函数

在Haskell中，函数是最重要、最基本的元素，所以我们有一些特别的函数语法来提高代码的可读性，简化书写。

- 函数分为普通函数和中缀函数两类。普通函数的调用方法就是函数跟上参数，中缀函数则和算术中的加减乘除差不多，先写第一个参数，再写函数，最后跟上第二个参数：

```
print "hello world"
-- 普通函数调用，print是打印函数，"hello world"是参数
```

```
2 + 3
-- 中缀函数调用，+是加法函数
(+) 1.5 2.5
-- 等价于 1.5 + 2.5
-- 中缀函数调用的另一种方法
```

```
Foo == False
-- 函数==判断是否相等
```

```
Bar /= True
-- 函数/=判断是否不相等
```

```
[1] ++ [] ++ [2,3,4]
-- 函数++用于连接两个列表
```

```
elem 3 [1,2,3]
-- elem x xs判断列表xs中是否出现x
3 `elem` [1,2,3]
-- 普通函数调用的另一种方法
```

上面最后一个例子是Haskell中的一个特殊语法。当需要把一个普通函数当作中缀函数使用时，只需在函数两侧加上'即可。而(+) 1.5 2.5则是把中缀函数+当作普通函数使用的方法，即在中缀函数两侧加上括号。

- 普通函数调用优先级最高，高于任何中缀函数，所以你可以这样写：

```
zs :: [Int]
zs = sort xs ++ sort ys
-- == (sort xs) ++ (sort ys)
```

- 中缀函数的优先级从0到9。和结合性一起定义，使用infix（不结合）、infixl（左结合）、infixr（右结合）说明：

```

infixl 6 +
-- + 的优先级是6, 左结合
infixl 7 *
-- * 的优先级是7, 左结合
infixr 8 ^
-- ^ 的优先级是8, 右结合
infix 4 `elem`
-- `elem`的优先级是4, 不能结合

```

```

2 + 3 * 4
-- == 2 + (3 * 4)
-- 因为*的优先级高于+

```

```

2 ^ 3 ^ 4
-- == 2 ^ (3 ^ 4)
-- /= (2 ^ 3) ^ 4
-- 因为^是右结合的

```

```

x `elem` xs `elem` ys
-- 报错, 没有结合性的中缀函数不能这么连着写, 必须加括号

```

如果一个中缀函数（包括使用`包围的函数）没提供优先级和结合性的说明，那么默认为 `infixl 9`，即左结合，最高中缀优先级。

- -既可以在中缀函数中表示相减，也可以在普通函数中表示求相反数，而两者的优先级都是6，于是：

```

3 * -2
-- 报错
3 * (-2)
-- OK
-- == 3 * negate 2
-- == - 6

```

- `::`类型说明符的优先级最低，所以`::`说明的是前面整个表达式的类型（有几个特例后面会提到）：

```

test 2 + 3 :: Double
-- == (test 2 + 3) :: Double
-- /= test 2 + (3 :: Double)

```

此外，初学者需要注意Haskell中的一些命名规则，否则程序无法通过编译。

- Haskell的绑定名称统一使用驼式命名法，虽然也允许出现`_`，甚至允许`'`，但一般都有特殊含义，例如`x'`一般代表稍加修改之后的`x`，和数学推导过程中使用的规则很像。
- Haskell的中缀函数允许使用的字符有`!`、`#`、`$`、`%`、`&`、`*`、`+`、`.`、`/`、`<`、`=`、`>`、`?`、`@`、`\`、`^`、`|`、`-`、`~`、`'`和`:`，但不能以`:`开头（后面会说到）。只要不要和一些内置的语法冲突，例如`->`和`=`，其他都可以定义，详细规则请参见Haskell 2010 Report。

中缀函数是Haskell中很有特点的一个设计，很多时候可以极大地改善代码的书写，但是切忌乱用。因为它们不像普通函数，可以从名字推测出很多信息，所以大部分时候，尽量使用有意义的名称来绑定你的函数。