Haskell函数式程序设计

Haskell简介

Haskell

- Haskell是一种纯函数式编程语言

- 1980年代以前对函数编程有很多研究, 但不同的研究者使用各自不同的语法记号, 一起交流时造成一些不便. 后来1987年的时候, 在FPCA'87会议上制定了统一的Haskell语言. Haskell吸收了各家的长处, 是一种纯粹的函数编程语言,并根据科学家Haskell B.Curry的名字命名. Haskell经过多年的发展完善, 目前使用的版本是Haskell 98.

Haskell的特点

- 1.Haskell具有引用透明性，没有副作用，适合并发编程——多线程之间不存在竞态条件，不需要加锁保护共享资源。
- 2.Haskell是惰性的——允许创建无限长度的数据结构
- 3.Haskell是静态类型的，支持类型推导

Haskell与C++、Python等区别

- 相对Haskell来说,传统的Basic,Pascal,C++,C#,Java,Python等都是命令(imperative)编程语言, 程序语句有一定的执行次序. 函数(functional)编程语言则给出执行的内容, 关注于更高层次的"做什么"而不是"怎么做", 这就是二者最明显的一个区别。

- 换言之，Haskell语言是写给人看的，而不是写给机器看的。

示例

- a = 2
- b = 4
- c = a + b
- 在函数编程中解释为: 定义常数a为2, 定义b为4, 定义c为a,b之和.
- 在命令编程中解释为: 给变量a赋值2, 给b赋值4, 求a,b之和赋给c.
- 定义和赋值的区别在于, 定义不分次序, 赋值有次序, 以上程序在Haskell中完全可以倒过来写

Haskell基本操作

Haskell工具

- 编辑器
- 文件格式要求.hs

- 编译器：Glasgow Haskell Compiler（GHC），Hugs
- http://hackage.haskell.org/platform

Downloads

There are three widely used ways to install the Haskell toolchain on supported platforms. These are:

- Minimal installers: Just GHC (the compiler), and build tools (primarily Cabal and Stack) are installed globally on your system, using your system's package manager.
- Stack: Installs the `stack` command globally: a project-centric build tool to automatically download and manage Haskell dependencies on a project-by-project basis.
- Haskell Platform: Installs GHC, Cabal, and some other tools, along with a starter set of libraries in a global location on your system.

These options make different choices as to what is installed globally on your system and what is maintained in project-specific environments. Global installations allow more sharing across users and projects, but at the cost of potential conflicts between projects. To avoid these conflicts, each option has a lightweight *sandboxing* feature that creates largely self-contained, per-project environments. With Minimal you can optionally sandbox the libraries, avoiding most conflicts. Stack sandboxes the compiler, tools and libraries, so avoids nearly all kinds of conflicts between projects. With Platform you can also optionally sandbox libraries, but not the globally installed platform libraries.

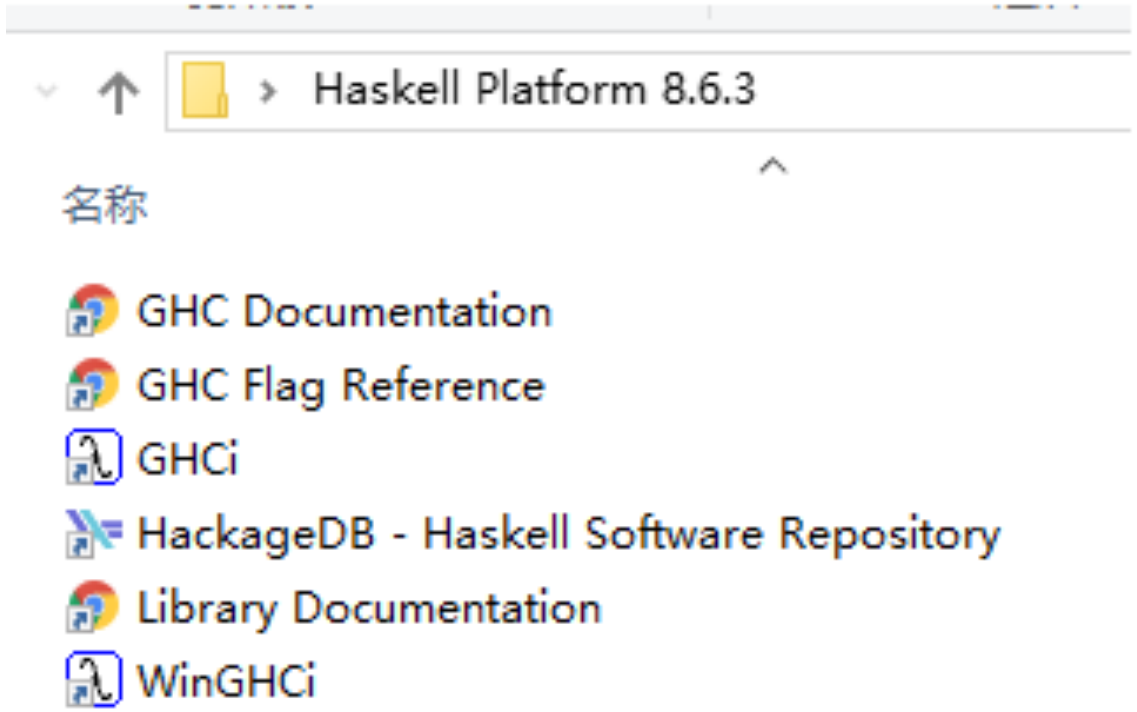
Haskell IDEs & other distributions

In addition to the generic, cross-platform Haskell toolchain described above, there are also easy-to-use, platform-specific distributions and IDEs. The Haskell Wiki contains a list of the most popular ones.

Minimal installers

Haskell基本操作

- 1. 在安装了ghci后，便可以进行Haskell的编译，点击GHCi 即可在命令行中打开ghci，也可点击WinGHCi，也可直接在命令行中ghci进入。



Haskell基本操作

- 2. :? 显示操作命令，常用操作有：
- :cd 进入指定路径
- :load 载入文件
- :quit 退出ghci

- 3.Haskell使用缩进来表示上一行的续写。
- 单行注释符号 --
- 块注释 {- -}

基本运算

- Linux或者Mac OS X系统：打开终端窗口、Windows：打开命令提示符，之后输入ghci并按回车键，进入交互模式

- Prelude> 2 + 15
- 17
- Prelude> 49 * 100
- 4900
- Prelude> 1892 - 1472
- 420
- Prelude> 5 / 2
- 2.5

执行顺序：按照运算符优先级顺序执行或者通过括号指定运算次序

表达式中存在负数常量，最好用括号括起来

函数调用

- 用编辑器创建函数
- doubleMe x = x + x
- 保存为baby.hs或任意名称，执行
- ：l baby　或 ：load baby
- 装载函数
- Prelude> doubleMe 9
- 18
- 修改.hs文档改变函数

Haskell数据类型

类型

●整数类型：

Integer是数字意义上的整数，理论上是无限大的，实际跟电脑内存有关

Int的范围为电脑存储一个字的大小，是有限的

若计算结果超出Int范围，则在表达式上用:: Integer表示为Integer类型。实际使用过程中当超出Int时，ghci会自动使用Integer类型表示，无须表明。超过Int的结果用Int类型表示则显示为0

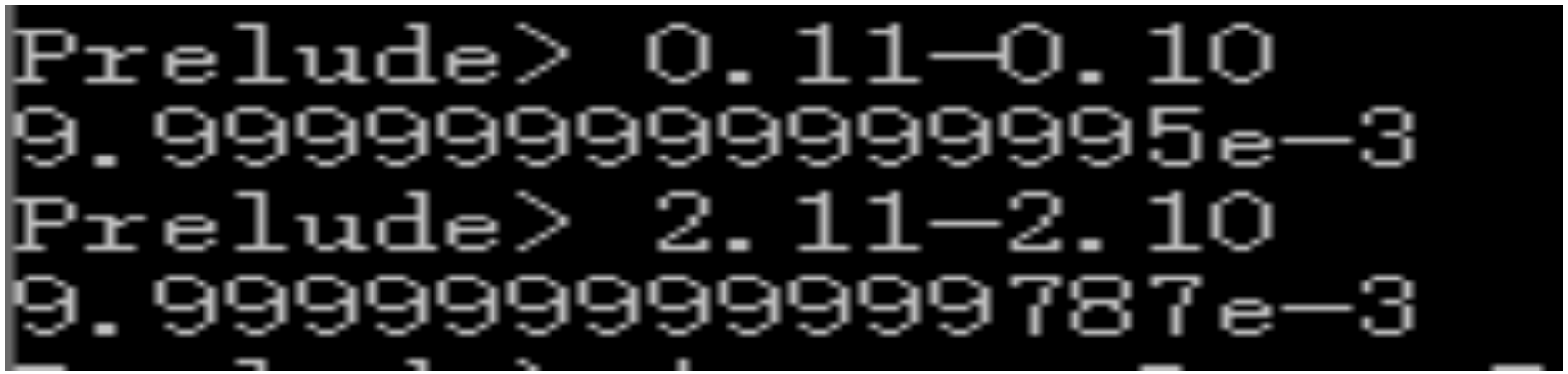
类型

- 浮点数类型：

  单精度浮点数（single-precision floating point numbers）

  双精度浮点数（double-precision floating point numbers）

浮点数是非精确表示的。比如0.11-0.10和2.11-2.10的结果应该是一样的，但是在Haskell中则是不一样的（也与电脑有关）.所以在比较两个浮点数时，应该比较其差值的绝对值是否小于一定范围



类型

- 有理数类型：Ratio Integer

  可以精确表示有理数

  有理数均可表示为分数的形式，Haskell中用（分子%分母）的形式表示有理数。使用前需要import Data.Ratio

```
Prelude> import Data.Ratio
Prelude Data.Ratio> 2%3
2 % 3
Prelude Data.Ratio> 1/6::Ratio Integer
1 % 6
Prelude Data.Ratio> 2^10
1024
```

类型

- Bool类型：布尔值——True和False

  逻辑运算：&&与；||或；not非

- Char类型：字符类型——用单引号将字符包起
- String类型：字符串（0个或多个字符的组合）——用双引号包起

- ‘a'：字符a
- “a”：包含了字符a的字符串

Hello World

- Prelude> “hello” ++ “world”
- “hello world”

- Prelude> ['h' , 'e' , 'l' , 'l' , 'o'] ++ ['w' , 'o' , 'r' , 'l' , 'd']
- “hello world”

- Prelude> putStrLn"Hello World!"
- Hello World!

Haskell基本数据结构

基本数据结构

●元组（tuple）

元组是将一系列数据值（可以是0个，但不可以是1个！）以逗号分隔放在括号里，如(2, "dog")是1个2个元素的元组，它的类型是(Int, String)。一个元组有n个元素，则成为n-tuple，2-tuples常被叫做pairs，有0-tuple，写作()，用来作为假值(dummy value)，但没有1-tuple。

基本数据结构

- 列表（list）

List是函数式语言中最常用的数据结构，相当于C语言中的数组，以中括号 [ ] 表示，以逗号 , 分隔，存储一系列相同类型的数据(也可以是list of tuples)，如[1, 2, 3]是a list of integers。List可以存储任意数量的元素（可以是0个即[ ]），但元素必须是同一类型的，如：

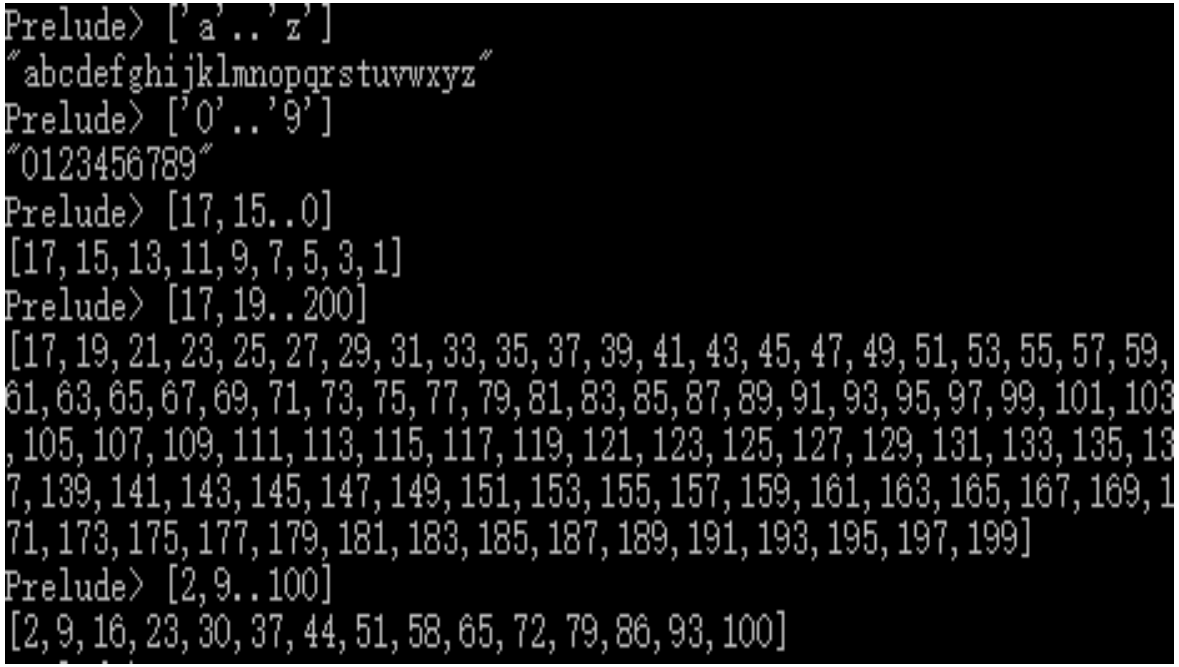
[13, 9, -2, 100] :: [Int]

["cat", "dog"] :: [String]

[[1, 2], [3, 7, 1], [ ], [900]] :: [[Int]]

基本数据结构

字符串String实际上就是字符Char的list，"abc" 与['a', 'b', 'c']是相同的。表示一系列数字或字符时，可以只写开头1个或2个和最后1个，中间用 .. 代替，如果只写开头1个，默认增量为1，写2个则以差值为增量。



基本数据结构

- ：用于将新元素添加到list的首部

- 所有的list都是由[]和：组成的

- 基础的列表解析是由一个表达式和一个生成器构成的，定义形式基于数学集合的定义形式 $\{x^2 | x \in S\}$

$$\{expression | generator\}$$

```
Prelude> 1:[2,3]
[1,2,3]
Prelude> (1:(2:(3:[])))
[1,2,3]
Prelude> 1:2:3:[]
[1,2,3]
```

```
Prelude> [y`mod`3|y<-[1..6]]
[1,2,0,1,2,0]
Prelude> [10*x|x<-[1,2,3]]
[10,20,30]
Prelude> [x^2|x<-[1..5]]
[1,4,9,16,25]
Prelude> [y`mod`3|y<-[1..6]]
[1,2,0,1,2,0]
```

基本数据结构

- 如果使用字符，需要先import Data.Char
- 如果是元组，取值也应该注意变化
- 可以有多个生成器，操作类似多重循环
- 生成器还可以添加限制条件

```
Prelude Data.Char> [toLower c|c<-"Too Marry CAPITALs"]
"too marry capitals"
Prelude Data.Char> [a*b|(a,b)<-[(1,2),(10,20),(6,6)]]
[2,200,36]
Prelude Data.Char> [(x,y)|x<-[1,2,3],y<-['a','b']]
[(1,'a'),(1,'b'),(2,'a'),(2,'b'),(3,'a'),(3,'b')]
Prelude Data.Char> [x|x<-[0..100],x `mod` 2 == 0 && x `mod` 7 == 0]
[0,14,28,42,56,70,84,98]
Prelude Data.Char>
```

Haskell操作符

基本操作符

- 二元操作符其实也是函数，但放在参数前面要加（）

```
Prelude Data.Char> (+)1 2
3
Prelude Data.Char> (&&) True False
False
Prelude Data.Char> (/)1 2
0.5
Prelude Data.Char> (**)2 0.5
1.4142135623730951
Prelude Data.Char>
```

基本操作符

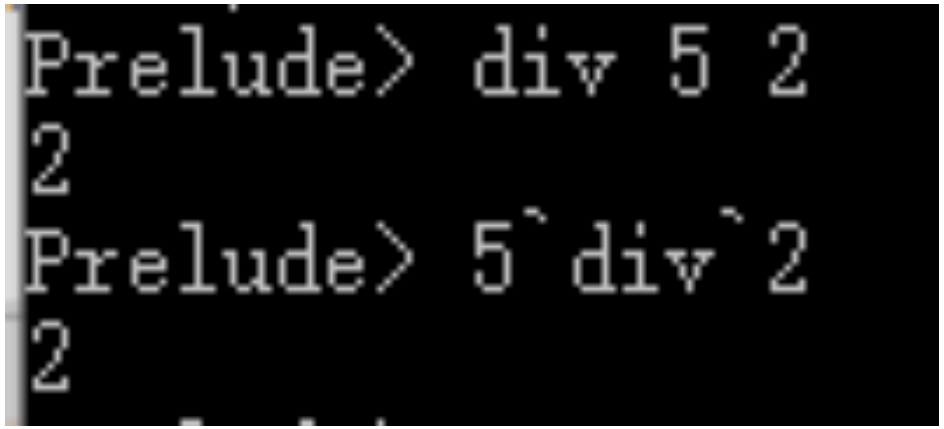
- :: operator　指明该表达式的类型，如 2::Int says 2 has type Int
- + 加 addition
- - 减 subtraction
- * 乘 multiplication
- / 除 division 如5/2 => 2.5
- ^ 幂函数 exponentiation
- ** 浮点数幂函数，如 2**0.5=>1.41421
- ++ 字符串操作符 用于连接连个字符串，如 "abc" ++ "def" => "abcdef"
- == 布尔类型操作符 等于
- /= 布尔类型操作符 不等于
- < 布尔类型操作符 小于
- <= 布尔类型操作符 小于等于
- > 布尔类型操作符 大于
- >= 布尔类型操作符 大于等于
- && 布尔类型操作符 与
- || 布尔类型操作符 或
- not 布尔类型操作符 非

Haskell函数设计

常用函数

- 函数名+空格+表达式即可，不用括号，当然在嵌套时要括号。

- 二元函数表示时，若放在元素的前面则直接打出，如div 5 2，若放在中间（二元函数）则要用反引号（back-quote，和~一个键，数字键1的左边）表示，如 5 `div` 2。



常用函数

- div 整除
- mod 求余
- max 求两者中的较大值 如max 3 8 => 8
- min 求两者中的较小值 如min 3 8 => 3

- toUpper 返回该字母的大写形式，需要import Data.Char
- toLower 返回该字母的小写形式，需要import Data.Char
- length 返回该字符串的长度

- fst 返回二元元组的第一个元素，如fst (1, 'a') =>1
- snd 返回二元数组的第二个元素，如snd (1, 'a')=>'a'

函数脚本

- 将函数定义在Haskell脚本文件(.hs)中，载入(在ghci中 :load)后，即可使用。

- 函数的声明：

$$function\_name :: \arg Type_1 -> \arg Type_2 -> ... -> \arg Type_n -> resultType$$

- 函数体：

$$function\_name \arg_1 \arg_2 ... \arg_n = \exp ression$$

应对多值

- 在定义函数时，为应对不同情况，可以有多个函数体（模式匹配）如：

```
is_three :: Int->Bool
is_three 3 = False
is_three x = True
f :: Integer -> String
f 1 = "one"
f 2 = "two"
f 3 = "three"
```

另一种应对多种值的方法，即使用bool表达式判断是否符合条件，符合则执行该行

```
fact :: Integer -> Integer
fact n
    | n < 0   = 0
    | n == 0 = 1
    | otherwise = n * fact (n - 1)
```

Haskell课堂实践练习

实践练习

1. Write a function that takes a character and returns True if the character is 'a' and False otherwise.
2. Write a function that takes a string and returns True if the string is “hello” and False otherwise. This can be done by specifying each element of the string in the list pattern.(e.g. 'h':'i':[])
3. Write a function that takes a string and removes a leading space if it exists.