

编写显然正确的代码

Author: 黄毅 众禄基金程序员

email: yi.codeplayer@gmail.com □

blog: <http://yi-programmer.com/> □

github: <http://github.com/yihuang> □

今天分享啥

- 码农的代码分享会
- 函数式编程风格 和 静态类型系统
- Haskell介绍

Haskell速成 - 函数定义

```
filter :: (a->Bool) -> [a] -> [a]
filter p [] = []
filter p (x:xs)
| p x       = x : filter p xs
| otherwise  = filter p xs
```

Haskell速成 - 函数定义

```
filter :: (a->Bool) -> [a] -> [a]
-- filter p [] = []
filter p (x:xs)
| p x      = x : filter p xs
| otherwise = filter p xs
```

Haskell速成 - 函数定义

```
filter :: (a->Bool) -> [a] -> [a]
-- filter p [] = []
filter p (x:xs)
| p x      = x : filter p xs
| otherwise = filter p xs
```

```
$ ghc -Wall foo.hs
Warning: Pattern match(es) are
          non-exhaustive
```

Haskell速成 - Curry化

add :: Int -> Int -> Int

(add 1) :: Int -> Int

Haskell速成 - Curry化

add :: Int -> Int -> Int

(add 1) :: Int -> Int

```
>>> let inc = add 1
>>> inc 2
3
```

Haskell速成 - 前缀/中缀表示法

```
>>> 1 + 2  
3  
>>> (+) 1 2  
3  
>>> add 1 2  
3  
>>> 1 `add` 2  
3
```

Haskell速成 - 前缀/中缀表示法

1 `add` 2 `add` 3 `add` 4

vs

add (add (add 1 2) 3) 4

Haskell速成 - lambda

\a b -> a + b

Haskell速成 - 结束

恭喜你， 你已经学会了Haskell
50% 常用语法

显然正确的代码

一、贴近自然语言描述

如何让代码更直接地表达你的想法

問題1

从列表取大于10且小于100的数

問題1

从列表取大于10且小于100的数

```
filter ((>10) `and` (<100))
```

分解：从列表取 ? 的数

分解：从列表取 ? 的数

```
\x -> filter ? x
```

分解：从列表取 ? 的数

\x -> filter ? x

不如直接点：

分解：从列表取 ? 的数

\x -> filter ? x

不如直接点：

filter ?

分解： 大于10

分解： 大于10

```
\x -> x > 10
```

分解： 大于10

```
\x -> x > 10
```

不如直接点：

分解： 大于10

\x -> x > 10

不如直接点：

(>10)

分解： 小于100

分解： 小于100

同样：

分解： 小于100

同样：

(<100)

分解：且

分解：且

&&

分解：且

&&

:: Bool -> Bool -> Bool

分解：且

&&

:: Bool -> Bool -> Bool

类型不对

分解：且

?

```
:: (a -> Bool)  
-> (a -> Bool)  
-> (a -> Bool)
```

分解：且

```
and f g = \x -> f x && g x
```

```
:: (a -> Bool)  
-> (a -> Bool)  
-> (a -> Bool)
```

分解：且

```
$ lambdabot
>>> :pl \f g x -> f x && g x
liftM2 (&&)
```

分解：且

```
and = liftM2 (&&)
```

```
:: (a -> Bool)  
-> (a -> Bool)  
-> (a -> Bool)
```

合并

```
filter (>10) `and` (<100)
```

函数管道 (.)

```
(.) :: (b -> c)
      -> (a -> b)
      -> (a -> c)
(f . g) x = f (g x)
```

```
+-----+
|       +---+       +---+   |
<<-c---c   b---b   a---a-<<-
|       +---+       +---+   |
+-----+
```

问题2

在二维数组里找长度大于5的子数组

在符合要求的子数组里找所有偶数

如果数据小于10则乘以2,大于10除以2

最后统计符合要求的数据的和

問題2

```
sum' = sum
  . map (\x -> if x<10
              then x*x
              else x `div` 2)
  . filter ((==0) . (`mod` 2))
  . concat
  . filter ((>5) . length)
```

抽象与性能不是死敌

GHC 编译器优化

- 内联（跨模块）
- 等价代码转换

查看中间代码

```
ghc -O  
-ddump-simpl  
foo.hs
```

GHC编译器中间代码是Haskell的子集

查看中间代码

查看中间代码

```
(==0) . ( `mod` 2)
```

查看中间代码

```
(==0) . ( `mod` 2)
```

优化后：

```
\x -> case modInt#(x, 2) of
    0 -> True;
    _ -> False;
```

查看中间代码

查看中间代码

```
map (*2)
. filter ((==1) . (`mod` 2))
```

查看中间代码

```
map (*2)
. filter ((==1) . (`mod` 2))
```

```
go xs = case xs of
  []    -> []
  x:xs ->
    case modInt#(x, 2) of
      1 -> (x*2) : go xs
      _ -> go xs
```

問題3

取http get参数"name"，前面加上"hello"返回回去。

```
webapp :: Application
webapp req = do
    let name = lookup "name"
        (queryString req)
    response ("hello "++name)
```

問題3

取http get参数"name"，前面加上"hello"返回回去。

```
webapp :: Application
webapp req = do
    let name = lookup "name"
        (queryString req)
    response ("hello "++name)
```

但是，如果用户没有传参数的话。。。

显然正确的代码

二、要能主动暴露自然语言不严谨之处

显然正确的代码

二、要能主动暴露自然语言不严谨之处

解决方案：精确的类型

问题3 - 继续

lookup :: k -> Map k v
-> ?

lookup 应该返回什么类型？

问题3 - 继续

v ?

问题3 - 继续

v ?

lookup :: k -> Map k v
-> v

问题3 - 继续

v ?
lookup :: k -> Map k v
-> v

process :: v -> something

问题3 - 继续

v ?
lookup :: k -> Map k v
-> v

process :: v -> something

```
>>> process (lookup k empty)
```

问题3 - 继续

v ?
lookup :: k -> Map k v
-> v

process :: v -> something

>>> process (lookup k empty)

crash

问题3 - 继续

答案: Maybe v

```
lookup :: k -> Map k v  
        -> Maybe v
```

问题3 - 继续

答案: Maybe v

lookup :: k -> Map k v
-> Maybe v

```
process (lookup k empty)
```

问题3 - 继续

答案: Maybe v

```
lookup :: k -> Map k v  
        -> Maybe v
```

```
process (lookup k empty)
```

type error

Maybe - 显式表达异常情况

```
data Maybe a = Just a  
              | Nothing
```

类型系统的终极目标

- 排除所有错误的程序
- 允许所有正确的程序
- 一言以蔽之：精确！

Haskell类型系统特点

- 自动类型推导（不挡路）
 - 表达能力强
 - 类型变量和类型构造器（泛型）
 - typeclass 对类型的约束
 - 恕不一一列举
- [http://www.haskell.org/ghc/docs/7.4.1/html/users_guide/type-extensions.html □]

Haskell类型系统作用

隔离纯函数式代码和命令式代码

upper :: String -> String

bomb :: String -> IO String

Haskell类型系统作用

精确的文档

readChan :: Chan a -> IO a

这个函数会阻塞吗？

Haskell类型系统作用

精确的文档

`readChan :: Chan a ->
IO (Maybe a)`

这个呢？

Haskell类型系统作用

Hoogλe 根据类型搜实现的搜索引擎

Hoogλe

(a -> Bool) -> [a] -> ([a], [a])

Packages

base

[break](#) :: (a -> Bool) -> [a] -> ([a], [a])
base Prelude, base Data.List
⊕ break, applied to a predicate p and a list xs, returns a tuple where first element is the longest prefix of xs for which p holds and the rest is the suffix.

[span](#) :: (a -> Bool) -> [a] -> ([a], [a])
base Prelude, base Data.List
⊕ span, applied to a predicate p and a list xs, returns a tuple where first element is the longest prefix of xs for which p holds and the rest is the suffix.

[partition](#) :: (a -> Bool) -> [a] -> ([a], [a])
base Data.List
⊕ The partition function takes a predicate a list and returns the pair of lists

回顾

显然正确的代码

- 贴近自然语言描述
- 精确的类型

Haskell is lazy

```
>>> let l = [1..]  
>>> take 5 l  
[1,2,3,4,5]
```

只在需要的时候进行计算

Haskell is lazy

在一个400米的环形跑道上

A以每秒一米的速度开跑

B以每秒两米的速度开跑

问他们何时相遇？

Haskell is lazy

```
iterate :: (a -> a) -> a -> [a]
iterate f a = [a, f a, f f a, ...]

>>> take 5 (iterate (+1) 0)
[0,1,2,3,4]
```

Haskell is lazy

```
a = iterate ((`mod` 400) . (+1)) 0
-- [0, 1, 2, 3, 4...]
```

```
b = iterate ((`mod` 400) . (+2)) 0
-- [0, 2, 4, 8, 10...]
```

```
findIndex (uncurry (==)) (tail (zip a b))
-- Just 399
```

Monad !

什么是Monad

Monad !

IO Monad - 命令式编程风格

```
do input <- getLine  
    forM_ [1..3] $ \i ->  
        printf "echo%d:%s" i input
```

Monad !

IO Monad - 命令式编程风格

```
do input <- getLine  
    forM_ [1..3] $ \i ->  
        printf "echo%d:%s" i input
```

```
> haskell  
echo1:haskell  
echo2:haskell  
echo3:haskell
```

Monad !

Resource Monad

在 IO 的基础上提供资源管理的能力。

```
do f <- openFile "data"  
    register (closeFile f)  
    process f  
    ...
```

Monad !

Resource Monad - 也可以主动取消。

```
do f <- io $ openFile "data"  
key <- register (closeFile f)
```

-- processing...

```
io $ closeFile f  
release key
```

-- processing...

Monad是对语句的抽象

```
do a <- ma  
  b <- mb  
  return c
```

Monad是对语句的抽象

```
do a <- ma  
    b <- mb  
    return c
```

```
ma >>= (\a ->  
            mb >>= (\b -> c))
```

Monad是对语句的抽象

```
class Monad m where  
    return :: a -> m a  
    (=>) :: m a -> (a -> m b)
```

Monad是对语句的抽象

```
class Monad m where
    return :: a -> m a
    (=>) :: m a -> (a -> m b)
```

```
instance Monad Foo where
    return a = ...
    ma >= f = ...
```

Monad不一定是IO

List Monad

```
do a <- [1..10]
   b <- [1..10]
   guard (a+b>10)
   return (a, b)
```

Monad不一定是IO

List Monad

```
do a <- [1..10]
   b <- [1..10]
   guard (a+b>10)
   return (a, b)
```

```
[(1,10),(2,9),(2,10),(3,8)...]
```

Monad不一定是IO

List Monad

```
instance Monad [] where
    return a = [a]
    xs >>= f = concatMap f xs
```

Q & A

Thanks

Can Programming Be Liberated from the von Neumann Style?

by John Backus 1978

冯诺依曼模型的问题

冯诺依曼模型的问题

依赖执行顺序的复杂的状态机模型

冯诺依曼模型的问题

依赖执行顺序的复杂的状态机模型

- 不容易理解
- 不容易组合

The rise of Haskell

- **September 1987.** Initial meeting at FPCA.
- **1 April 1990.** Version 1.0 Report was published.
- **May 1996.** Version 1.3 Report with Monadic I/O.
- **February 1999** Haskell 98 Report was published.
- **July 2010** Haskell 2010 Report was

GHC - 工业级Haskell实现

- 支持Haskell 2010以及大量扩展功能
- 强大的优化能力，能够跨模块优化
- 能生成高效的代码，并发程序尤其表现突出
[<http://shootout.alioth.debian.org/> □]
- 完美的并发和并行实现，包括M-N微线程和STM实现
- 跨平台支持 (Windows, Linux, Mac, 有非官方的iOS的支持)
- Profiling支持，包括time/allocation以及多种heap profiling。

其他实现

- UHC 有字节码解释器和Javascript后端。
- 其他

[<http://www.haskell.org/haskellwiki/Implementations>]

[]

ArchSummit

中国·深圳 2012.08

INTERNATIONAL ARCHITECT SUMMIT

全 球 架 构 师 峰 会

详情请访问: architectsummit.com

•3天 •6场主题演讲

•3场圆桌论坛 •9场专题会议

•国内外30余家IT、互联网公司的50多位来自一线的讲师齐聚一堂

主办方: InfoQ

战略合作伙伴: Tencent 腾讯

特别支持:



<http://architectsummit.com>



杭州站 · 2012年10月25日~27日
www.qconhangzhou.com (6月启动)

QCon北京站官方网站和资料下载
www.qconbeijing.com

