

Scala

函数式编程

monad

Haskell

副作用

✎ 修改

如何理解extensible effect? ✎ 修改

cs.indiana.edu/~sabry/p... okmij.org/ftp/Haskell/e.....显示全部 ▾

关注问题

✎ 写回答

+ 邀请回答

9 条评论

🔗 分享

★ 邀请回答

🚩 举报

...

关注者

91

被浏览

4,357

他们也关注了该问题



查看全部 4 个回答



Felis sapiens 🌟
函数式编程、编程语言、编程 话题的优秀回答者

刘雨培、酿酿酿酿酿泉、Belleve、彭飞、祖与占等 33 人赞同了该回答

目的：我们希望在函数式语言中处理一些副作用，比如非确定性、状态读写、异常处理等等。理想的处理副作用的框架应该满足这些特性：

- 0. 带类型：使用了哪些副作用可以在类型中显式指定，而非在一个大杂烩式的IO类型中
- 1. 可组合：涉及不同副作用的代码能够组合使用，而且组合之后的行为是可预测的，不会因为混入其他副作用产生奇怪的行为
- 2. 可扩展：两个维度上的可扩展：可以定义新的副作用；可以对已有的副作用定义新的解释器
- 3. 高效率：用到了哪些副作用，就只付出与之对应的开销，且开销越小越好
- 4. 支持类型推导：对使用副作用的代码并不强制要求类型签名

Haskell中传统的处理方式是monad transformer和monad transformer class。一个monad transformer能够在不破坏monad laws的前提下为一个base monad增添新的操作的支持，同时能将base monad给lift到新的monad之中来，这样我们可以从一个trivial的base monad出发（纯的代码用Identity，非纯的代码用IO），通过apply一系列transformer，获得对一系列副作用处理方式的支持。


monad transformer的问题很明显，需要手动lift，而且经常不止一层lift，如果已有的代码套一层新transformer，那么现有代码全部作废，需要多加一层lift。另外，transformer的定义和解释是合一的，比如StateT的定义是newtype StateT s m a = State { runStateT :: s -> m (a, s) }，然后我们有get、put、modify等操作，但是这些操作都是仅针对这一个StateT实现的，我希望达到的效果是，自己的代码里用到这些操作，然后只要修改一下入口处一点点代码，就可以切换不同 的StateT实现，用纯函数模拟/用IORef读写/访问数据库等等。

解决方案是mtl为代表的monad transformer class。现在我们的代码类型签名不再是ReaderT r (StateT s (WriterT w ...)) a了，而是(MonadReader r m, MonadState s m, MonadWriter w m, ..) => m a。一个class可以用不同的transformer来实现，与此同时mtl中针对几个常用transformer有一些自动的lifting操作，比如MonadReader r m => MonadReader r (StateT s m)，这样一来很大程度上消除了手写lift的必要性。然后从mtl出发更进一步地，可以做indexed monad transformer class，比如ether库，通过类型层面的标签，可以支持多个不同reader/state等的并存。

至于extensible effects，经常与free/freer monad一同提起。这里略过free，直接先讲freer了。freer monad就是operational monad，具体而言，假设我们用一个数据类型表示“状态读写”这个副作用：

```
data State s a where
  Get :: State s s
  Put :: s -> State s ()
```

广告 X



腾讯云

校园专享服务器套餐10元/月

腾讯云服务器高性能计算能力,更有学生优惠套餐云服务器/域名/存储等服务.

➔

关于作者



Felis sapiens

🌟 函数式编程、编程语言、编程 话题的优秀回答者

👤 电影旅行敲代码、Antokha Yuuki、暮无井见铃也关注了她

回答	文章	关注者
181	40	14,872

已关注

💬 发私信

被收藏 15 次

编程语言与编译原理

酿酿酿酿酿泉 创建

5,338 人关注

Programming Languages

彭飞 创建

3,642 人关注

收藏夹

圆角骑士魔理沙 创建

273 人关注

Haskell

parker liu 创建

10 人关注

Haskell

5 人关注



现在，我们希望写一段用到State的程序，并且用monad来表达顺序执行的逻辑。现在问题来了，我们需要支持lift和bind两个操作，而以上State定义是完全没法定义lift和bind的。不过我们可以作弊！

```
data State s a where
  Get :: State s s
  Put :: s -> State s ()
  Lift :: a -> State s a
  Bind :: State s a -> (a -> State s b) -> State s b
```

一个monad instance实现立即呼之欲出了。现在，我们可以写一些诸如

```
runStatePure :: State s a -> s -> (a,s)
runStateFileIO :: Serialize s => State s a -> Handle -> IO a
```

等，用不同的方式去解释State这个API的解释器。这就是operational monad的基本思想：用一个kind为* -> *的数据类型来表达一个DSL支持哪些操作，然后直接用作弊的方法获得monad instance实现，最后可以写不同的解释器去解释这个DSL的程序。

最后终于可以谈到extensible effect了。首先，extensible effect中所用的freer作弊方式特殊一点，不是将Lift和Bind塞进DSL的数据类型，而是

```
data Freer f a where
  Pure :: a -> Freer f a
  Impure :: f a -> (a -> Freer f b) -> Freer f b
```

，对于任何kind为* -> *的f，都可以定义Freer f的monad instance。（实际实现中，Impure部分中a -> Freer f b这个continuation不会简单地用一个函数表示，而是用一个type-aligned sequence）

然后是effect的组合——单个的effect可以表达为* -> *的数据类型，多个effect可以用类型列表[* -> *]来表示，最后用type class来自动化inject（单个effect注入一段程序）和project（抽取单个effect的代码）操作。具体操作方式参考大名鼎鼎的《Data types à la carte》，或者用Data.Typeable做运行时类型标签来解决。

发布于 2016-06-05

赞同 33

1 条评论

分享

收藏

感谢

...

收起

更多回答



Belleve

编程 话题的优秀回答者

19 人赞同了该回答

我有两个操作

```
data Interaction : Type -> Type where
  Say : String -> Interaction ()
  Ask : Interaction String
```

以及解释它们的方法

```
interface (Monad m) => Handler (g : Type -> Type) (m : Type -> Type) where
  handle : (action : g a) -> (continuation : a -> m b) -> m b

Handler Interaction IO where
```

相关问题

如何理解 Edward Kmett 所写的 machines 这个库？ 2 个回答

如何理解 Objective-C 中的 strong 和 weak？ 9 个回答

如何理解c++中的引用折叠？ 6 个回答

如何理解 dependent type？ 4 个回答

如何理解虚拟DOM？ 17 个回答

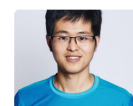
相关推荐



森懂物理学：理解世界的极简指南

共 31 节课

试听



如何矫正跳外翻？

康复师 于老师

★★★★★ 387 人参与



如何高效阅读

(美) 彼得·孔普 (Peter ...)

54 人读过

阅读



刘看山 · 知乎指南 · 知乎协议 · 隐私政策

应用 · 工作 · 申请开通知乎机构号

侵权举报 · 网上有害信息举报专区

违法和不良信息举报: 010-82716601

儿童色情信息举报专区

电信与服务业务经营许可证

网络文化经营许可证

联系我们 © 2018 知乎





```
handle (Say s) k = putStrLn s >>= k
handle (Ask) k = getLine >>= k
```

[展开阅读全文](#)

赞同 19 添加评论 分享 收藏 感谢 ...



夏梓耀

硬核码农

8 人赞同了该回答

谢邀 & 抱歉，来晚了，之前了解extensible effect，都是靠脑补，觉得还是花点时间看看源码后再来回答比较好(我也不知道该怎么理解，所以只能看代码了)。
parper (okmij.org/ftp/Haskell/e...) 一开始讲到Effect的概念，若想更好的理解还是看看effect handlers (math.andrej.com/wp-cont...) 比较好。
相对于monads with a hole (这个比喻真好)， extensible effect将effects视为是一种 ‘interaction and intro-duced side-effect-request handlers’，其effects requests可以写成特殊的数据类型，如：

```
data It i a = Pure a
           | Get (i -> It i a)
```

这种由可能的操作组成的代数模型（我不规范的称为AST啦），在Haskell里叫 ‘operational’（我不是很清楚）
它可以得出monad instance，而其AST的interpreter就是requests handlers，比如把上面的It i a

[展开阅读全文](#)

赞同 8 3 条评论 分享 收藏 感谢 ...

查看全部 4 个回答