Haskell 狂信徒的 Scala 入坑笔记（1）

Felis sap...
函数式编程、编程语言、编程 话题的优秀回答者

First things first..

- 这不是一篇 self-contained 的 Scala 教程，也不是一篇教人如何开始学习 Scala 或者函数式编程的资源贴。这只是一个 SAN 值归零的 Haskell 信徒的一些月夜呓语。包含不系统、不客观、不友善的功能列举和点评。长期更新。欢迎评论各种吐槽。
- 作者背景：Scala 入坑一月有余。之前则专注于 Haskell 和静态类型函数式编程，有时候写点 Python 或者 JavaScript。对 JVM 系语言零基础。对面向对象编程。。算了这种 ill-defined 的概念咱们先不管。先交代到这儿。
- 开始学习 Scala 之前，我把那个 Martin Odersky 的 “Haskellator” 的梗当真了（Scala 是 Haskell 的 gateway drug）！！我以为学会 Haskell 以后 Scala 什么的就可以横扫啊，分分钟就会啊！！被彻底教做人了（不过更有可能是我已经学傻了）。另外谁说 Scala 是函数式语言我跟谁急。

配置开发环境

- 首先，可以选择的 Scala 编译器版本很多。在 JVM 上跑的有 Lightbend Scala / Typelevel Scala / Dotty，另外还有 scalajs 和 Scala Native。不同实现的对比网上很多，这里就不赘述了，上手的话选择官方实现 Lightbend Scala 即可。
- 两个 Scala 相关的在线服务推荐一下：在线运行一段 Scala 代码，可以用 ScalaFiddle；浏览 GitHub 上的 Scala 代码，可以装一下 Insight.io 的 Chrome 插件，这个平台上对很多 Scala 的 repo 进行过索引，浏览代码时可以点击跳转。（对 sbt 构建时在 src_managed 里生成的 Scala 代码也做了索引，不过点击看不到，有点小遗憾）
- Scala 可以用 sbt 或者 maven 管理依赖，两个工具我都不熟，不过 sbt 看上去可定制性要好得

以用 scalafmt 格式化 Scala 代码，scalafmt 像 clang-format 一样可以用配置文件配置 code style。

- Scala 官方编译器带有一个 REPL，另外一些 Scala 项目（比如 Spark）也提供了自定义的 REPL，但是最好用的 REPL 还是 jupyter-scala。这是一个 Jupyter Notebook 的 Scala 内核，底层基于 Ammonite 定制，所以支持 Ammonite 的一些神奇的特性，比如 magic import 语法，可以在一个 notebook 里面动态装新的依赖，不需要写 sbt 配置之类的，使用体验极佳。现在我的 workflow 基本就是开一个 notebook 把雏形做得差不多以后再把代码片段迁移到 IntelliJ IDEA 里面做后续的开发。

学习资源（我看过的）

- Scala for the Impatient。比较精炼的 Scala 教材，适合刚入坑时快速浏览一遍。国内有翻译版。
- Tour of Scala。Scala 官网的短教程系列，一篇一个特性。
- Scala’s Types of Types。各种 Scala 类型特性的简介。
- Scalable Component Abstractions。一篇 Martin Odersky 的早年论文，可以一窥 Scala 为什么要长成这个样子。

关于副作用

普通语言里有 block statement，而 Scala 则有 block expression，一个 code block 本身即是一个合法的表达式，可以嵌套在其他表达式中，而 block expression 最后一个表达式的类型则为整个表达式的类型。

block expression 的根源：Scala 是一门 non-pure，默认 strict 求值的语言，与 Haskell 针锋相对。一个 expression 被求值可能触发任意副作用。在 Haskell 中，副作用通常使用某个 kind 为 (* -> *) 的 DSL 来表示（比如最简单的 IO），这个 DSL 可以通过 monad class 的 return 和 bind 的操作进行组合。然而 Scala 里只需要在 block expression 中顺序写下带副作用的表达式，block expression 被求值时，这些表达式的副作用即会被依次触发，底层并没有用到 monad bind，类型系统也并不参与副作用的管理。

两门语言对于赋值的处理也不一样。Haskell 中没有赋值，只有一次性的绑定，所谓的赋值操作只是针对某个 ref 类型的 monadic 值而已；OCaml 之类的 ML 系语言还算矜持，有 ref/array 类型，有针对这些类型的非纯的 read/write 操作；Scala 就完全放开了，声明一个绑定时可以用 val/var 加以区分，使用 var 则直接支持赋值（val 声明的绑定不支持赋值，但是仍然可以用对象本身的带副作用的方法修改对象状态）。

Scala 虽然默认 strict 求值，但是支持局部 lazy 求值。一个绑定被声明为 lazy val 时不会被立即计算，而只会在第一次被用到时计算一次并缓存（call by need）。被声明为 def 时，按照零参函数处理，每次被用到时都会被求值（call by name）。函数的参数列表里，如果是 f(x: A)，代表

不过，使用一个 DSL 管理副作用的做法有时还是会用到。比如需要异步编程时，可以使用 scala 标准库提供的 Future 或者 scalaz 提供的 Task，它们也都支持使用 flatMap 等组合子显式地进行组合。这时可以使用 ThoughtWorksInc/each 提供的语法糖，用写普通带副作用表达式的写法来写 monadic 的表达式。

个人观点，懒惰不懒惰事小，纯洁不纯洁事大。默认求值策略是 strict 或者 non-strict，都可以很方便地添加一些语言构造在局部转换求值策略，但是如果允许任意位置插入副作用，那么再想提倡使用类型系统管理副作用的风格就很难办了，这个设定不仅会阻止很多编译优化，也容易让不称职的程序员写出依赖某些迷之全局状态的糟糕代码。类型是人类的好朋友，类型不能阻止程序员作死，但可以让作死的代码写起来更费劲、看起来更刺眼。

关于函数

Scala 里的函数类型很丰（hun）富（luan），原生支持多参、带名字参数和变长参数。参数也可以带上 lazy 的声明（见上一节）或者 implicit 的声明。如果怀念 Curry-style 的话。。一个 Scala 函数签名是可以带上多个参数列表的，所以写成 Curry-style 也没毛病。

个人还是更加钟爱 Haskell 里只提供单参函数的做法。需要多参可以传 tuple 啊，需要带名字参数可以传 record 啊，需要变长参数可以传 List 或者 HList 啊。。而且 Haskell 也无需针对零参函数有特殊的处理（因为默认 pure 所以很容易做到）。类型和语法更简洁了，但是并没有失去什么力量。

Scala 的 method call 语法有个语法糖：a.f(b) 可以写成 a f b 的写法，这样一来可以有限度地定制自己的 binary operator，比如 a -> b，只需要实现 a.-> 的 method 即可。operator 定制上 Haskell 灵活很多，支持自定义 operator 的优先级和结合性。不过跟走火入魔的 Agda 相比也还算矜持的。

Scala 中的 function call 语法，只要实现 apply method 即可使用，类似 C++ 中函数对象实现 operator ()。实际上 Scala 中，你看见 A => B 类型，以为是一个原生的函数类型，实际上只是个 Function1[A, B] 这个 trait 的实例而已，所谓的 lambda 表达式也只不过是“免得手动创建 FunctionN 的 trait 实例并提供 apply method 实现”的一个语法糖而已。可以，这很面向对象.jpg

最后一提：Scala 函数的类型推导就是个战斗力 0.5 的渣渣，谁用谁知道。

关于 newtype 和 Value Type

Haskell 里面有 type alias 和 newtype 可以对一个已有类型进行标记，其中前者不会创造一个新类型，后者则会（newtype 的 Constructor 应用过的值不能作为原类型值使用）。newtype 功能的美妙之处在于：

- newtype 严格保证 runtime representation 与原类型一致，这样一来可以进行 zero-cost type-safe 的 coerce，并且这种 coerce 不仅针对 newtype 和 原类型，也针对任何复杂类型中 newtype 和原类型的相互替换，比如 (forall r . (Stream NInt -> IO r) -> IO r) 和 (forall r . (Stream Int -> IO r) -> IO r) 之类的东西可以直接相互 coerce，假设 newtype NInt = Int

Scala 里有一个类似的优化，叫 Value Type。将单个类型的值实现一个 wrapper class 时，指定让 wrapper class extends AnyVal 即可，编译器会尝试去掉额外的对象分配的开销。但是优化不总能成功，而且也没有 zero-cost coerce 和获取原类型的任意 type class instance 这两大好处。

关于 case class 与模式匹配

Haskell 和 ML 系语言的共同点是大量使用 ADT（Algebraic Data Types）及针对 ADT 的模式匹配。 ADT 的一大特点是 sealed，添加或者删除其中的 variant 会使得所有对该 ADT 进行匹配的函数需要重新编译，而 sealed 使得针对模式匹配的 exhaustiveness check 容易实现。

Scala 里的 case class 支持模式匹配。如果每一个 case class 所 extends 的那个 base class/trait 带有 sealed 的声明，那就可以模拟普通的 sealed ADT。如果不带 sealed 声明，那么实际上可以当作一个类型不安全的 expression problem 解决方案：当某一个 datatype 的新的 variant 在另一个 package 中被声明时，原来的那些对该 datatype 进行模式匹配的函数仍然可以通过编译，并且可以给这些函数传入新的 variant，这个新的 variant 会在运行时报告匹配失败。。。

OCaml 的 polymorphic variant 是同时确保了可扩展性和类型安全的解决方案，一个函数如果对 polymorphic variant 进行了类型匹配，那么函数的类型签名可以告诉你它支持哪些 variant，传入不支持的 variant 会造成类型错误。在 Haskell 里可以用类似实现 HList 的手段实现 HSum，模拟 polymorphic variant 的效果。不清楚 Scala 里面有没有人这么做过，原理上讲是办得到的。

另外，既然是 case class，而且仍然是用 class 语法声明，所以可以在 base class/trait 里面声明这个 ADT 支持的方法，然后用普通 method call 的语法调用。

What comes next

今天先写最简单的几个点吧。之后有空再慢慢更新深一点的内容，比如 HKT/RankN 函数、implicit 和 type class 相关、variance、path-dependent type、反射、scalaz 和 shapeless 相关，等等。晚安。

编辑于 2017-08-10

Haskell Scala 函数式编程

文章被以下专栏收录

推荐阅读

Notes on Haskell Debugging

elis... 发表于不动点高校...

幻想中的Haskell - Compiling Combinator

圆角骑士魔... 发表于雾雨魔法店

Haskell开发环境配置

寸光寸阴 发表于Haske...

学 H

几年
滴le
开 G
FTP
Fold
然了

Cosr

18 条评论

切换为时间排序

写下你的评论...

dram 1 年前

我怎么觉得 f(x: => Int) 是 call by name 每次试用求值（

1

以上为精选评论

Cykalert 1 年前

1

dram 1 年前

Scaskellator!

2

Cykalert 回复 dram 1 年前

是call by name来着，spec写了

赞 查看对话

Felis sapiens (作者) 回复 dram 1 年前

嗯，updated。。多谢提醒

赞 查看对话

SeanDragon 1 年前

mark

赞

Yutong Zhang 1 年前

parametricity……（算了……

赞

祖与占 1 年前

司机到底要开多少坑 (

赞

Geek.cs 1 年前

HSum 在著名的库 shapeless 里有，叫 Coproduct

赞

北南 1 年前

非常赞，从纯fp过来的人是不一样！

赞

jilen 1 年前

司机，那个并行编程，改成异步更准确

赞赞赞，haskell大佬视角的scala评价超级有价值(☆_☆)

赞

Felis sapiens (作者) 回复 jilen 1 年前

嗯，多谢提醒

赞 查看对话

Felis sapiens (作者) 回复 Yutong Zhang 1 年前

之后在反射的部分会讲的

1 查看对话

余维浩 4 个月前

Local type inference 其实也够了 (虽然不如 Hindley–Milner type inference 强大)

赞

baozii 2 个月前

看得过瘾！第二篇呢?

赞

andwxh 2 个月前

诶？！已经一年过去了吗（暗示（逃

赞

Felis sapiens (作者) 回复 andwxh 2 个月前

我早就脱离苦海了（逃