fpzen Documentation

Выпуск latest

июл. 23, 2018

Содержание:

Глава 1

Вступление

1.1 Предисловие

Помнится, в возрасте 14и лет я купил свою первую книгу по программированию, которой оказалась монография Бьерна Страуструпа — «Язык программирования C++». Во вступлении я обратил внимание на следующее высказывание: написать книгу — это колоссальный труд. Я сразу же ей поверил, но всё таки долгое время не понимал, что именно входит в состав сложности написания книги. Т.к. это была моя первая книга по программированию, я тогда ешё не подозревал, что есть и плохие книги. Прочитав множество книг по разным темам в области разработки ПО, я точно могу сказать, что хороших книг очень мало. К этому моменту у меня накопилось большое количество учебного материала по языку Scala и по парадигме функционального программирования (ФП), которое я хотел бы изложить в виде текста. Не успев даже начать писать это вступление, я вспомнил слова Бьерна. Писать книгу — сложно. Писать хорошую книгу — в разы сложнее. У меня есть большой опыт проведения очных уроков для своей команды, но это совершенная другая среда передачи информации. Вы можете махать руками, делать оговорки, отвечать на вопросы, ругаться матом, тыкать пальцем на экране, забирать свои слова обратно, смешить людей, видеть их реакцию, соответственно подстраиваться. При написании книги вы всего этого лишены. Перед вами черный экран с белыми буквами. Автору даёт сил лишь мысль, что в голове читателя что-то щёлкнет, произойдёт тот самый момент откровения, когда ранее разнозненные участки сознания наконец-то соединяются в единую картину.

1.2 Цели книги

Я ставлю перед собой три цели: рассказать читателю о языке Scala и ввести его в мир функциональной парадигмы в контексте этого языка. Сначала я думал разделить книгу на две части, где сначала рассказываю просто про Scala, а потом отдельно про функциональную парадигму. Но понял, что функциональная парадигма сильно проникает во все элементы языка, и что разделять их будет крайне неудачной идеей. А идею изучать Scala саму по себе как просто ещё один язык без применения функциональной парадигмы — считаю бесполезной тратой времени. Если бы существовал какой-то численный способ измерить навык программиста, то он точно бы не состоял из количества языков, известных программисту. Одним из главных показателей было бы разнообразие парадигм, подходов,

концепций, техник, которые освоил инженер. Третья цель — написать хорошую и интересную книгу. Тяжело определить, что делает хорошую книгу хорошей. Будем надеяться, что у меня получится. TODO что именно хотел

1.3 Стиль и формальность определений

Я стараюсь придерживаться крайне нестрого стиля изложения. Я не претендую на точность математических определений. Считаю, что уровня определений в дальнейшем материале крайне достаточно для понимания функциональной парадигмы. TODO

1.4 Как писалась эта книга

Я решил писать эту книгу параллельно с преподаванием этого материалы для команды разработчиков, в которой я являюсь техническим руководителем. TODO: (типа опробовано на людях, feedback все дела) Даже если из этого не получится хорошей книги, то как минимум выйдет хорошая отправная точка нового члена команды на пути изучения Scala и ФП. Многие из членов команды помогали в редактировании и рецензировании материала.

1.5 Функциональное программирование

Книга предназначена для инженеров с достаточным опытом программирования в популярной императивной парадигме. В противоложность императивному, функциональное программирование (ФП) — это программирование с помощью чистых функций. Чистой является та функция, которая детерминирована и не обладает наблюдаемыми побочными эффектами. Функция является детерминированной, если для одного и того же набора чистых входных значений она возвращает одинаковый результат. Дать определение наблюдаемым побочным эффектам сложнее. Самое ёмкое определение, которое я могу предложить на данном этапе, на самом деле, не сильно проясняет нам картину. Но всё же, я называю наблюдаемым побочным эффектом в контексте вызова функции любую мутацию состояния, ссылка на которое существует за пределами этой функции или от состояния которого зависит работа самой программы. Типичными примерами побочных эффектов являются: - модификация переменных (edit-in-place), - внешний ввод-вывод (I/O), - выбрасывание исключения.

Мне кажется заблуждением противопоставлять объектно-ориентированное программирование (ООП) функциональному. Если мы посмотрим на основные принципы ООП: - абстрагирование, - инкапсуляция, - наследование, - полиморфизм,

то не увидим здесь ни одного, слова противорещачего идеи чистых функций. Все эти принципы также будут действовать в ФП, причём я бы даже сказал, в усиленной форме. (ТАК МОЖНО СКАЗАТЬ?) TODO: concrete examples

Нельзя сказать, что програмированние на каком-то определённом языке автоматически означает использование конкретной парадигмы. Можно лишь говорить о популярности соответствуюших парадигм в языках и степени поддержки и выразительности этих парадигм в заданных языках. Так вот, в современных популярных промышленных языках, к которым я отношу C, C++, Java, C#, ECMAScript, PHP, Python, Ruby, Pascal (и подобные), наиболее распостранена смесь объектно ориентированной и императивной парадигмы. Хотя в то же время, они предоставляют средства для функционой парадигмы. Другое дело, что насколько это выразительно и целесообразно. К современным промышленным языкам, в которых наиболее распостранена функциональная парадигма я отношу Haskell, Closure, OCaml, Erlang, TypeScript (или PureScript) и, конечно же, Scala.

1.6 Классификация языка Scala

Часто языки программирования классифицируют разделяя их по свойствам динамичности/статичноси, строгости и явности. Что точно можно сказать — Scala язык со статической типизацией. Это означает, что конечные типы переменных и функций устанавливаются на этапе компиляции. С остальными свойствами не всё так однозначно. Строгая (иногда называют сильной) типизация выделяется тем, что язык не позволяет смешивать в выражениях различные типы и не выполняет автоматические неявные преобразования, например нельзя вычесть из строки множество. Языки с нестрогой (слабой) типизацией выполняют множество неявных преобразований автоматически, даже если может произойти потеря точности или преобразование неоднозначно. Ну что можно сказать про язык в котором специально введено понятие неявных преобразований (implicit conversions)? Только то, что это язык со строгой (сильной) типизацией, одновременно позволяющий разработчику явно определять собственные неявные преобразования, которые могут быть использованы. Простите за каламбур. Далее, явно-типизированные языки отличаются тем, что тип новых переменных, функций и их аргументов нужно задавать явно. Соответственно языки с неявной типизацией перекладывают эту задачу на компилятор или интерпретатор. Как говорят, золотая всегда середина; так вот Scala именно там. Это язык со смешенной явной и неявной типизацией в том смысле, что от разработчика требуется только явное задание типов аргументов в сигнатурах функций, а остальные типы компилятор в большинстве случаев может вывести сам, и вам их указывать нет необходимости. Однако, такая возможность остаётся для более явной нотации, либо для подсказок или корректировок компилятора. TODO: Concrete examples?

Глава 2

Level 0

2.1 Holy fuckeroni

```
val x:Int = 2
```

back to page again

2.1.1 Секция (уровень 3)

Субсекция (уровень 4)

so what are we gonna write? in russian though?

пиать на русском?

даже не знаю что и писать теорию в общем

или что ещё

Day1 scala worksheey

Список 1: day1.scala

```
type Row = Map[String,String]
type Table = Seq[Row]

trait Source {
  def getRows: Table
}

class InlineSource(val table:Table) extends Source {
  def getRows = table
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
val source = new InlineSource(Seq(
  Map("A" -> "ORD-17813",   "B" -> "250.00", "C" ->  "740.00"),
  Map("A" -> "ORD-6724215", "B" -> "150.00", "C" -> "2500.00"),
  Map("A" -> "ORD-21235",   "B" -> "300.00", "C" -> "5700.00")
))


def processRow(row: Map[String,String], repo:OrderRepository):Unit = {
  // parsing phase
  val orderId  = parseCellAsString(row, "A")
  val payment  = parseCellAsMoney(row, "B")
  val delivery = parseCellAsMoney(row, "C")

  // lookup phase
  val orderRef = repo.findOrder(orderId)

  // IO phase
  repo.writeTx(orderRef, PaymentFromRecipient, payment)
  repo.writeTx(orderRef, DeliveryCost, delivery)
}

def parseCellAsString(row:Row, col:String):String = row(col)
def parseCellAsDouble(row:Row, col:String):Double = parseCellAsString(row, col).toDouble
def parseCellAsMoney(row:Row,  col:String):Money = Money(parseCellAsDouble(row, col))

class OrderRepository {
  def findOrder(id:String):OrderRef = OrderRef((Math.random() * 100000).toInt)
  def writeTx(order:OrderRef, txType: TransactionType, amount: Money): Unit = {
    println(s"Writing transaction of type $txType for order $order with amount = $amount")
  }
}

case class OrderRef(id:Long)

sealed trait TransactionType
case object  DeliveryCost         extends TransactionType
case object  PaymentFromRecipient extends TransactionType
case object  CashService          extends TransactionType

case class Money(amount: BigDecimal)


// program itself
val repo = new OrderRepository
source.getRows.foreach(processRow(_, repo)) // no result, we have no way to explore it
```

Ok?

Глава 3

Day 2

Глава 4

Day 3

Глава 5

Index

- genindex