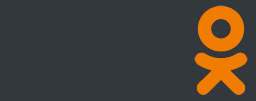
Harry Potter and Why Functional Programming Matters

2016

Развлекательная социальная сеть Одноклассники

Виталий Худобахшов

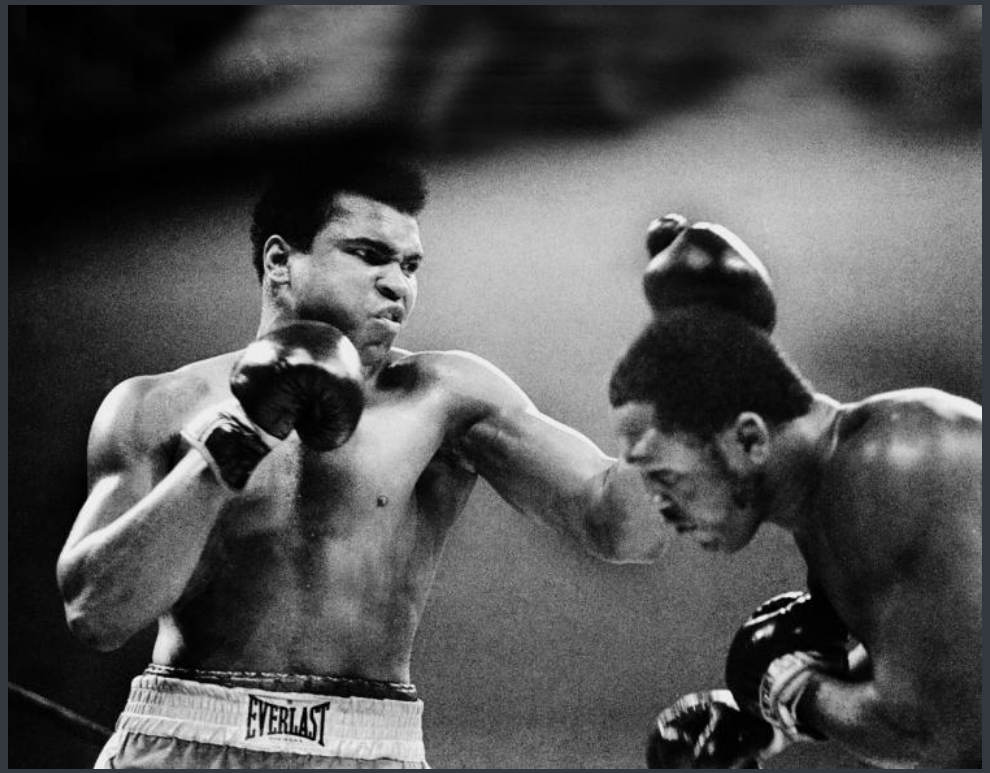
Содержание



- Не все то золото, что функционально
- Больше свободы = больше ответственности
- Анализ данных и MapReduce как функциональное программирование на каждый день
- Стримоз, ФП головного мозга и прочие болезни
- Кейсы

Функциональное программирование (Haskell)

ООП (Java)



Смешанная парадигма (Scala)

Все что вы хотели знать о ФП, но боялись спросить

Основные принципы

ООП

- Инкапсуляция
- Полиморфизм
- Наследование

Данные: название источника данных, 2016

Основные принципы

ООП

- ~~Инкапсуляция~~ (есть в любых языках кроме ассемблера)
- ~~Полиморфизм~~ (зависит от системы типов)
- Наследование

Данные: название источника данных, 2016

Основные принципы

Функциональное программирование

- Функции как объекты 1-ого класса
- Рекурсия
- Сопоставление с образцом
- ~~Алгебраические типы~~
- Каррирование
- Неизменяемость

Данные: название источника данных, 2016

Функции как объекты 1-ого класса

Что это значит

- Функции можно передавать в качестве параметра
- Возвращать в качестве значения

Где это реализовано

- Lisp (1958), ML (1973), Erlang (1987), Haskell (1990), Python (1991), R (1993), Scala (2003), Clojure (2007), C++ (2011), Kotlin (2011), Java (2014)

Данные: название источника данных, 2016

Рекурсия

Что это значит (в теории)

- Циклов нет (хвостовая рекурсия)

Что это значит (на практике)

- Частичный отказ от циклов в пользу функций `map`, `fold` и `filter` позволяет лучше структурировать код

Данные: название источника данных, 2016

Сопоставление с образцом

Теория

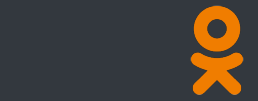
- Код красивый и понятный
- Умный компилятор точно скажет, где и что ты забыл
- Особенно хорошо, если ты пишешь компилятор/интерпретатор

Практика

- Кое где работает медленно
- Плохо сочетается с циклами
- Используется только для деконструкции

Данные: название источника данных, 2016

Левая свертка (в теории)

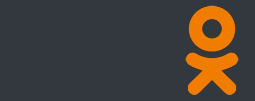


Как должно бы быть

```
@tailrec
def foldLeft[A, B](f: (B, A) => B)(z: B)
    (list: List[A]): B = list match {
     case Nil => z
     case x :: xs => foldLeft(f)(f(z, x))(xs)
    }
```

Данные: название источника данных, 2016

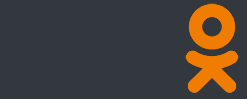
Левая свертка (на практике)



List.foldLeft в Scala

```
def foldLeft[B](z: B)(f: (B, A) => B): B = {
  var acc = z
  var these = this
  while (!these.isEmpty) {
    acc = f(acc, these.head)
    these = these.tail
  }
  acc
}
```

Левая свертка (альтернативный вариант)



Что на счет этого варианта?

```
@tailrec
def foldLeft2[A, B](f: (B, A) => B)(z: B)
                   (list: List[A]): B =
if (list.isEmpty) z else foldLeft2(f)(f(z, list.head))
                   (list.tail)
```

Каррирование

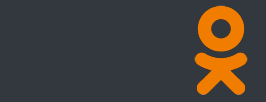
Теория

```
def add(x: Int, y: Int) = x + y
def add(x: Int)(y: Int) = x + y
def add(x: Int) = (y: Int) => x + y

def sum = foldLeft((x: Int, y: Int) => x + y)(0) _
```

Практика

1. `foldLeft(f)(z)`
2. `foldLeft(z)(f)`

Каррирование



Теория

```
def add(x: Int, y: Int) = x + y
def add(x: Int)(y: Int) = x + y
def add(x: Int) = (y: Int) => x + y

def sum = foldLeft((x: Int, y: Int) => x + y)(0) _
```

Практика

1. `foldLeft(f)(z)  => sumWithAcc = foldLeft(+) _`
2. `foldLeft(z)(f)  => ?????`

Каррирование

Вывод №1

- Если каррирование не является «синтаксически прозрачным», это сводит на нет все его плюсы

Вывод №2

- 7 раз отмерь – 1 раз отрежь

Неизменяемость

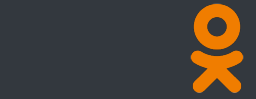
Теория

- Отсутствие присваивания
- Любая обработка структуры данных приводит к появлению новой структуры
- Высокий уровень параллелизма

Практика

- Дозированное использование состояния
- Основной паттерн обработки (больших) данных

Очень тяжелый случай



Почему это плохо?

```
val m: Map[String, Int] = ...
m.map(x => (x._1, x._2 * x._2))
```

Очень тяжелый случай

Почему это плохо?

```
val m: Map[String, Int] = ...
m.map(x => (x._1, x._2 * x._2))
```

А если так?

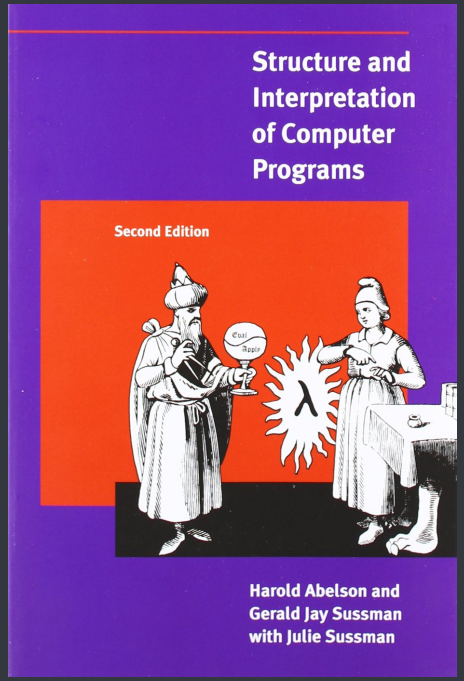
```
m.map(x => ("foo", x._2))
```

Почему это плохо на самом деле?

```
map(id) = id
map(g ∘ f) = map(g) ∘ map(f)
```

4 Вывод

Компилятор языка Scala должен останавливать каждого, кто не читал эту книгу!



Анализ данных и ФП

Кто такие датамайнеры?

Это те люди, которые используют функциональное программирование чаще других

Виталий Худобахшов, 2016

Датамайнинг и функциональное программирование

Функциональное программирование

Анализ данных

Из чего состоит анализ данных

Обработка данных

- Выборка
- Очистка
- Агрегация

MapReduce/SQL

Построение модели

- Машинное обучение
- Тестирование модели

MapReduce

Больше данных – больше ответственность

Особенности MapReduce парадигмы

- Неизменяемость
- Ассоциативность редукции
- Данные представляют собой пары (key, value)
- Эти данные являются распределенными

Что такое shuffle?

Shuffle для обычного программиста за 2 минуты

```
val rdd = sc.textFile(...)
rdd.flatMap {
      line => line.split(' ')
}.map((_, 1))
.reduceByKey((a, b) => a + b).collect()
```

- Чтобы просуммировать значение для двух одинаковых ключей, соответствующие пары *должны быть на одном вычислительном узле*

Кейсы

Кейс №1: The Good, the Bad and the Ugly

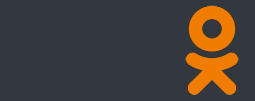
Постановка задачи

- Около 200 млн пользователей
- Не все пользователи указали свой истинный возраст
- Как оценить истинный возраст пользователей?

Исходные данные

- Социальный граф как список смежности с некоторыми размеченными связями
- Таблица известных возрастов пользователей

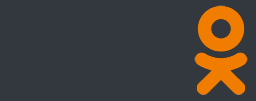
Вариант 1



JOIN, JOIN - и в продакшн

```
(userId, [friendId1, friendId2, ...]) => // flatMap

(friendId, userId) =>                    // join

(friendId, (userId, birthYear)) =>       // map

(userId, [(friendId, birthYear)]) =>     // reduceByKey

(userId, [(friendId1, birthYear), ...])
```

Вариант 2



Есть же DataFrames и SQL

```
graph.flatMap {
  r =>
    val userId = r._1
    val friends = r._2
    friends.map(x => (userId, x._1))
}.toDF("USER_ID", ”OTHER_USER_ID").
   write.parquet("users/graph_pairs")
```

Вариант 3: свободный стиль

А вдруг shuffle будет очень большим?

- 200 млн пользователей * среднее количество друзей = очень много пар
- А можно совсем без JOIN?

Если очень хочется, то можно!

1. Разделить информацию о возрасте на части
2. Для всех полученных частей
   - Сделать так, чтобы эта часть была на всех узлах
   - Для каждого пользователя в графе проставить нужную информацию о возрасте его друзей
   - Для тех друзей, которые не попали в эту часть, оставить значение по умолчанию
3. Рассчитать возраст исходя из возраста друзей

Шаг 1 - Split

```
val customers = sqlc.sql("SELECT USER_ID, BIRTH_YEAR
FROM customers")

customers.map(r => (r.getLong(0), r.getInt(1)))
    .randomSplit(splitWeights, 3)
    .zipWithIndex
    .foreach {
     x =>
       x._1.saveAsObjectFile(s"split-${x._2}")
    }
```

Шаг 2 - Broadcast

```
(0 until splitFactor).foreach { s =>
   val splitGraph = sc.objectFile(s"users-graph-$s")
   val split = sc.objectFile(s"split-$s")
   split.collect()
        .foreach(x => hashMap.put(x._1, x._2))
   val bc = sc.broadcast(hashMap)
   val next = s + 1
   ...
```

Шаг 2 – вместо JOIN

```
...
splitGraph.map { x =>
   val hm = bc.value
   val friendsWithBD = x._2.map { u =>
      if (u._3 == 0) {
         val v = hm.get(u._1) // 0 если нет
         (u._1, u._2, v)      // (Long, Long, Int)
      } else u
   }
   (x._1, friendsWithBD)
}.saveAsObjectFile(s"users-graph-$next")
...
```

Шаг 3 – все просто

Расчет вариантов возраста

- Можно взять одноклассников и посмотреть их возраст
- То же самое с сокурсниками
- Можно найти самую большую группу пользователей среди друзей с одинаковым возрастом
- Выбрать наиболее подходящий вариант

Who is who?

Результаты для разных вариантов

• Вариант 1 (ФП)	15 минут	The Good
• Вариант 2 (ФП + SQL)	15 минут	The Ugly
• Вариант 3 (Mix)	~5 часов	The Bad

Кейс №2: Как сделать рекомендации

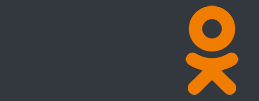
Постановка задачи

- 200 млн пользователей
- 10 млн групп
- Как рекомендовать группу конкретному пользователю

Исходные данные

- Набор пар (userID, groupID)

Как сделать рекомендации



```
(userId, groupId) =>                                  // map
(userId, [groupId]) =>                                // reduceByKey

(userId, [group_1, group_2, group_3 …]) =>  // flatMap

((group_1, group_2), 1)
((group_1, group_3), 1)
((group_2, group_3), 1)
... =>                                                // reduceByKey

((gid1, gid2), #number_of_shared_users)
```

Немного императивного кода

Функция genPairs()

```
def genPairs(seq: Seq[Long]): Seq[((Long, Long), Int)] = {
  val buf = new ArrayBuffer[((Long, Long), Int)](seq.size * seq.size >> 1)
  var i = 0
  var j = 0
  while (i < seq.size) {
    while (j < seq.size) {
      val x = seq(i)
      val y = seq(j)
      if (x > y) buf.append(((x, y), 1))
      j += 1
    }
    i += 1
  }
  buf
}
```

Как сделать рекомендации

Реализация

```
pairs.map{
    case (userId, groupId) => (userId, Seq(groupId))
}.reduceByKey((a, b) => a ++ b)
 .flatMap(m => genPairs(m._2))
 .reduceByKey((a, b) => a + b)
 .filter(x => x._2 > minimumSharedUsers)
 .map {
    case ((groupId1, groupId2), sharedUsers) =>
        ...
 }
```

Выводы

- Нужно перестать бояться и начать использовать Spark
- Оптимизируйте правильно
- Держитесь в рамках выбранной парадигмы, если это не вредит
- Используйте сильные стороны разных парадигм

Кое что о названии и что почитать

Why Functional Programming Matters by John Hughes

- Why Functional Programming Matters by John Hughes from “Research Topics in Functional Programming” ed. D. Turner, Addison-Wesley, 1990, pp 17–42.

Structure and Interpretation of Computer Programs

- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-001-structure-and-interpretation-of-computer-programs-spring-2005


одноклассники