Гуляев Г.М.
Современные технологии программирования

Лекция 3. Массивы, карты, кортежи

Центр компетенции СПО Алтайского края

Отложенные (ленивые) вычисления

- Если переменная декларируется как **lazy**, то ее вычисление откладывается до того момента, когда она будет использована в первый раз:

```
import scala.io.Source
lazy val users = Source.fromFile("/srv/список.txt").mkString
for(u <- users.split('\n')) println(u)
```

- Три варианта задания вычислений:

```
val users = Source.fromFile("/srv/список.txt").mkString
// Вычисляется в момент инициализации users
lazy val users = Source.fromFile("/srv/список.txt").mkString
// Вычисляется в момент первого использования users
def users = Source.fromFile("/srv/список.txt").mkString
// Вычисляется всякий раз когда используется users
```

Исключения в Scala

- Исключения в Scala действуют аналогично Java или C++. Пример вызова исключения:

```
if (x >= 0) math.sqrt(x)
else  throw new IllegalArgumentException("x не может быть меньше нуля")
```

- Как и в Java класс исключения должен быть унаследован от java.lang.Throwable
- Однако в Scala нет проверяемых (checked) во время компиляции исключений. Обрабатывать или нет исключение решает сам программист.
- Еще одно отличие в измененном синтаксисе блока catch:

```
try{
  …
} catch {
  case e: IOException => println("Ошибка ввода вывода "+e)
  case _: Throwable => println("Неизвестная ошибка")
} finally {...}
```

Массивы в Scala

- **Массивы фиксированной длины:**

```
val nums = new Array[Int](10) // 10 целых, инициализация: 0
val a = new Array[String](10) // 10 строк, инициализация: null
val s = Array("Привет","мир") // Array[String] длиной 2
s(0) = "Пока" // изменение элемента с индексом 0
```

- **Массивы Scala перед компиляцией для jvm приводятся к массивам Java, например Array(2,3,5,7,11) в jvm будет выглядеть как int[]**
- **Массивы переменной длины. В Java для переменных массивов мог быть использован ArrayList. В Scala же для этого используется класс ArrayBuffer.**

```
import scala.collection.mutable.ArrayBuffer
val b = ArrayBuffer[Int]() // или new ArrayBuffer[Int] - пустой массив
b += 1 // ArrayBuffer(1)
b += (1,2,3,5) // ArrayBuffer(1,1,2,3,5)
b ++= Array(8, 13, 21) // ArrayBuffer(1, 1, 2, 3, 5, 8, 13, 21)
```

Массивы в Scala

- **Массивы переменной длины (продолжение):**

```
b.trimEnd(5) // ArrayBuffer(1, 1, 2) - удалены 5 последних элементов
b.insert(2, 6) // ArrayBuffer(1, 1, 6, 2) - вставка b(2)
b.insert(2, 7, 8, 9) // ArrayBuffer(1, 1, 7, 8, 9, 6, 2) - вставка b(2),b(3),b(4)
b.remove(2) // ArrayBuffer(1, 1, 8, 9, 6, 2) - удален b(2)
b.remove(2,3) // ArrayBuffer(1, 1, 2) - удалены 3 эл-та b(2),b(3),b(4)
val a = b.toArray // Array(1, 1, 2) - приведение к Array
a.toBuffer // ArrayBuffer(1, 1, 2) - обратное преобразование
```

- **Обход массивов. В Java для разных списков разные способы обхода. В Scala все унифицировано.**

```
for (i <- 0 until a.length) println(i + ": " + a(i)) // вывод элементов
0 until 10 // Range(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
0 until (10, 2) // Range(0, 2, 4, 6, 8) - делящиеся на 2
(0 until 10).reverse // Range(9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
for (e <- a) println(e) // вывод элементов (другой способ записи)
```

Преобразование массивов

- Преобразование массива при помощи for:

```
val a = Array(2, 3, 5, 7, 11)
val res = for (e <- a) yield 2 * e  // res = Array(4, 6, 10, 14, 22)
val res = for (e <- a if e%2==0) yield 2 * e  // res = Array(4)
```

- Такие способы не нужно использовать, поскольку у списков есть много встроенных методов (map, filter, ...).

```
val res = a.map(x => x * 2)  // res = Array(4, 6, 10, 14, 22)
val res = a.filter(x => x%2==0).map(x => x * 2)  // res = Array(4)
val res = a.map(_ * 2)  // res = Array(4, 6, 10, 14, 22)
val res = a.filter(_ % 2 ==0).map(_ * 2)  // res = Array(4)
```

- Использование шаблонов (символ _) также желательно избегать, предпочитая анонимные функции.
- Агрегатные функции:

```
a.sum  // Int = 28
a.filter(x => x%2==1).sum  // Int = 26
a.max  // Int = 11
```

Преобразование массивов

- **Агрегатные функции со строковым массивом:**

```
val a = Array("один", "два", "три", "четыре", "пять")
a.max  // String = четыре
a.min  // String = два
scala.util.Sorting.quickSort(a) // Array(два, один, пять, три, четыре)
```

- **Преобразование к строке:**

```
a.mkString(" и ")  // String = два и один и пять и три и четыре
a.mkString("<", ",", ">")  // String = <два,один,пять,три,четыре>
a.toString  // String = [Ljava.lang.String;@7a5b2b1a
a.toBuffer.toString  // ArrayBuffer(два, один, пять, три, четыре)
```

- **Массивы в Scala являются частным случаем списков (Seq) и все методы для списков работают и для массивов.**
- **Методов достаточно много, рекомендуется при поиске новой возможности обращаться к документации.**

Многомерные массивы

- Как и в Java многомерный массив определяется как массив массивов.
- Например, двумерный массив из Double это Array[Array[Double]]
- Для построения такого массива используется метод ofDim объекта Array

```
val matrix = Array.ofDim[Double](3, 4)  // 3 строки и 4 столбца
```

- Доступ к элементам при помощи задания двух индексов строки и столбца (row,column): matrix(row)(column) = 42

```
matrix(2)(3)  // Double = 0.0
matrix(2)(3) = 42
matrix(2)(3)  // Double = 42.0
```

- Так как внутренние массивы могут быть разной длины, то можно строить любые непрямоугольные матрицы.

Карты(Maps)

- Карта в Scala - это обычная хэш-таблица (ключ -> значение):
  val scores = Map("Иванов" -> 75, "Петров" -> 60, "Сидоров" -> 26)
- Здесь **scores** - неизменяемая (immutable) **Map[String,Int]**. Если нужна изменяемая (mutable), то импортируем **scala.collection.mutable.Map**
- Пары **"Петров" -> 60** можно задавать так: **("Петров", 60)**, поэтому можно было написать и без стрелок:
  val scores = Map(("Иванов", 75), ("Петров", 60), ("Сидоров",26))
- Получение данных:

```
scores("Петров")  // Int = 60
if (scores.contains("Петров")) scores("Петров") else 0  // Int = 60
scores.getOrElse("Петров",0)  // Int = 60 - краткая запись предыдущего
scores.get("Петров")  // Option[Int] = Some(60)
scores.get("Егоров")  // Option[Int] = None
scores.get("Петров").getOrElse(0)  // Int = 60
scores.get("Егоров").getOrElse(0)  // Int = 0
```

Карты(Maps)

- **Изменение данных (требуется mutable Map):**

```
import scala.collection.mutable.Map
val scores = Map("Иванов" -> 75, "Петров" -> 60, "Сидоров" -> 26)
scores("Петров") = 55  // изменение значения
scores("Егоров") = 78  // добавление пары
scores += ("Петров" -> 55, "Егоров" -> 78)  // другой способ
scores -= "Егоров"  // удаление
val scores1 = scores + ("Петров" -> 55, "Егоров" -> 78) // новая карта
val scores2 = scores - "Егоров" // новая карта
```

- **Обход элементов:**

```
for((k,v) <- scores) println((k,v))  // вывод пар: (ключ,значение)
for((k,v) <- scores) println(k+" -> "+v)  // вывод пар: ключ -> значение
scores.keySet  // приведение ключей к Set
for (v <- scores.values) println(v)  // вывод только значений
for((k,v) <- scores) yield (v,k)  // меняем местами ключ и значение
```

Кортежи(Tuples)

- Кортеж - это объединение различных объектов в одну структуру, например: (1, 3.14, "Иван")

```
val t = (1, 3.14, "Иван")  // t: (Int, Double, String) = (1,3.14,Иван)
t._1  // Int = 1
t._2  // Double = 3.14
t._3  // String = Иван
val (first, second, third) = t  // first = 1 second = 3.14 third = Иван
val (first, second, _) = t  // first = 1 second = 3.14 (нужны не все)
```

- Упаковка (zipping):

```
val symbols = Array("<", "-", ">")
val counts = Array(2, 10, 2)
val pairs = symbols.zip(counts)  // Array[(String, Int)] =
                                                    Array((<,2), (-,10), (>,2))
for ((s, n) <- pairs) print(s*n)  // <<---------->>
```

- Для сведения в карту двух массивов одинаковой длины keys и values можно использовать конструкцию:

```
keys.zip(values).toMap
```

Решение задач

- **Рассмотрим следующую задачу:** задан массив целых чисел и требуется удалить из него все отрицательные числа кроме первого.
- **Обычное решение (императивный стиль):**

```
def upd(a: Array[Int]) = {
 var first = true
 val b = ArrayBuffer[Int]()
 for(n <- a) {
  if (n>=0) b += n
  else {
   if (first) {
    b += n
    first = false
   }
  }
 }
 b.toArray
}
```

- **Пришлось ввести флаг first для обнаружения первого отрицательного элемента. Функция выглядит сложно.**

Решение задач

- **Пытаемся найти другое решение:**

```
val a = Array(1,3,-5,2,-11,-8,12,-21)  // Пример массива
a.filter(x => x<0).tail  // Array(-11, -8, -21) - элементы для удаления
a diff a.filter(x => x<0).tail  // Array(1, 3, -5, 2, 12) - решение???
```

- **К сожалению найденное «решение» неверно (diff удаляет первые встреченные элементы):**

```
val b = Array(1,3,-5,2,-5,-5,12,-5)
b diff b.filter(x => x<0).tail  // Array(1, 3, 2, 12, -5)
```

- **Пробуем применить ту же идею к индексам массива:**

```
a.indices  // Range(0, 1, 2, 3, 4, 5, 6, 7)
a.indices.filter(i => a(i)<0).tail  // Vector(4, 5, 7)
val r = a.indices diff a.indices.filter(i => a(i)<0).tail  // Vector(0, 1, 2, 3, 6)
((for(i <- r) yield a(i)) или r.map(i => a(i))).toArray // Array(1, 3, -5, 2, 12)
```

- **Решение найдено:**

```
def upd(a: Array[Int]) = {
  val ind = a.indices
  (ind diff (ind.filter(i => a(i)<0).tail)).map(i => a(i)).toArray
}
```

Задания для самостоятельной работы

- Напишите функцию **f(a: Array[Int]): Array[Int]** меняющую рядом стоящие элементы массива - аргумента. Например, f(Array(1,2,3,4,5)) = Array(2,1,4,3,5).
- Дан массив **a** целых чисел. Получить массив, содержащий сначала все положительные элементы **a** в оригинальном порядке, затем все нулевые и отрицательные элементы **a** в оригинальном порядке.
- Вычислить среднее-арифметическое значение элементов массива **Array[Double]**?
- Из массива удалить все повторяющиеся элементы.
- Создайте коллекцию всех часовых поясов для Азии методом **java.util.TimeZone.getAvailableIDs**. Отбросьте префикс "Asia/" и отсортируйте результат.
- Напишите функцию **minmax(a: Array[Int])**, возвращающую пару, содержащую наибольшее и наименьшее значения.
- Напишите функцию **lteqgt(a: Array[Int], k: Int)**, возвращающую тройку, содержащую количество элементов массива больших k, равных k и меньших k.
- Напишите функцию, считающую повторения слов в тексте, возвращающую **Array(Map[String,Int])**.

Задания для самостоятельной работы

Написать функции:

1. f(n,d) - суммарное количество использований цифры d (0,1,2,3,4,5,6,7,8,9) для записи всех чисел от 1 до n включительно.

2. f(n) - наибольшее из чисел от 1 до n включительно, обладающее свойством: сумма цифр n в некоторой степени > 1 равна самому числу n. Пример: $512 = 8^3$, f(512) = 512, f(513) = 512, f(514) = 512, ...

Спасибо за внимание !

www.altailand.ru