Гуляев Г.М.

Современные технологии программирования (часть 2)

Лекция 2. Язык Scala

Курс лекций для студентов АлтГТУ

Функции и процедуры

- Функции декларируются при помощи def:

```
def abs(x: Double) = if (x >= 0) x else -x
```

- Типы параметров обязательно должны быть заданы. Тип возвращаемого значения задавать обязательно только если функция рекурсивная:

```
def fact(n: Int): Int = if (n <= 0) 1 else n * fact(n - 1)
```

- Если в теле функции более одного выражения, то используются фигурные скобки:

```
def fact(n : Int) = {
  var r = 1
  for (i <- 1 to n) r = r * i
  r
}
```

Функции и процедуры

- **Определение параметров по умолчанию (left и right):**

```
def decorate(str: String, left: String = "[", right: String = "]") = left + str + right
```

- **Примеры:**

```
decorate("Hello") // [Hello]
decorate("Hello", "<<", ">>") // <<Hello>>
decorate("Hello", "<<") // <<Hello]
decorate(left = "<<<", str = "Hello", right = ">>>") // <<<Hello>>>
```

- **Неопределенное число параметров:**

```
def sum(args: Int*) = {
  var res = 0
  for (arg <- args) res += arg
  res
}
```

- **Пример:**

```
val s = sum(1, 4, 9, 16, 25) // Int = 55
```

Функции и процедуры

- Int* преобразуется к списку Seq[Int] и в цикле for (arg <- args) перебираются все элементы списка.
- Попытка выполнить функцию sum, передавая ей аргументом 1 to 5 вызывает ошибку: sum(1 to 5) // Ошибка
- Однако, если преобразовать тип аргумента так: (1 to 5): _* , то все работает: sum((1 to 5): _*) // 15
- Смысл символов шаблона в Scala: _ - любой элемент, * - повторение любое число раз (включая 0).
- Пример суммы с рекурсией:

```
def sum(args: Int*): Int = {
  if (args.length == 0) 0
  else args.head + sum(args.tail: _*)
}
```

Функции и процедуры

- Процедуры Scala - это функции, которые возвращают тип Unit.

```
def box(s : String): Unit = {
  ...
}
```

- Для процедур допускается запись без знака = :

```
def box(s : String) {
  ...
}
```

- Типичная ошибка начинающего программиста на Scala - это пропуск знака = в определении функции. Поэтому, возможно и не стоит использовать упрощенную форму записи для процедур.
- То есть можно считать (и это правда), что процедур в Scala нет, а есть только функции.

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)
```

- **Такой способ не нужно использовать, поскольку у списков есть много встроенных методов (map, filter, ...).**

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

- **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.

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)).toArray  // Array(1, 3, -5, 2, 12)
```

- **Решение найдено:**

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

Карты(Maps)

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

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

Карты(Maps)

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

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

Спасибо за внимание !

www.altailand.ru