Дата последней сборки документа:
13 августа 2020 г. в 00:43

Заметки. Практика использования и наиболее полезные конструкции языка Scala

Подвойский А.О.

Здесь приводятся заметки по некоторым вопросам, касающимся машинного обучения, анализа данных, программирования на языках Scala и прочим сопряженным вопросам так или иначе, затрагивающим работу с данными.

Содержание

1 Вызов функций и методов **1**

2 Метод apply **1**

3 Управляющие структуры и функции **2**

3.1 Условные выражения 2

4 Ввод и вывод **3**

5 Циклы **4**

5.1 Расширенные циклы for и for-генераторы 5

Список литературы **6**

1. Вызов функций и методов

Математические функции определены в пакете scala.math. Их можно импортировать инструкцией

```
import scala.math._ // импорт всех элементов пакета
```

Здесь символ «_» – «групповой» символ, аналог «*» в Python: from math import *.

Замечание

При использовании пакета, имя которого начинается с префикса scala., этот префикс можно опустить. Например, инструкция import math._ эквивалентна инструкции import scala.math._, а вызов math.sqrt(2) эквивалентен вызову scala.math.sqrt(2)

---

2. Метод apply

В языке принято использовать синтаксис, напоминающий вызовы функций. Например, если s – это строка, тогда выражение s(i) вернет i-ый символ строки

```
"Fortran"(4)  // вернет ‘r‘ как 4-ый символ строки
// тоже самое с использованием метода ‘apply‘ "Fortran".apply(4)
```

Функции часто передаются методам в очень компактной форме записи. Например, чтобы вернуть количество символов верхнего регистра в строке можно воспользоваться конструкцией

```
// --- Scala ---
val s: String = "PythonTheBestLanguage"
s.count(_.isUpper)  // 4
```

На Python эту задачу можно решить так

```
# --- Python ---
# с помощью генератных выражений и генераторов списков
In[]: %timeit -n10 len(list(char for char in s if char.isupper()))
Out[]: 7.62 milis +/- 383 ns per loop (mean +/- std. dev. of 7 runs, 10 loops each)
In []: %timeit -n10 len([char for char in s if char.isupper()])
Out[]: 5.04 milis +/- 406 ns per loop (mean +/- std. dev. of 7 runs, 10 loops each)
# с помощью теоретико-множественных операций
In[]: from string import ascii_uppercase
In []: %timeit -n10 len(set(s).intersection(set(ascii_uppercase)))
Out[]: 7.5 milis +/- 812 ns per loop (mean +/- std. dev. of 7 runs, 10 loops each)
```

3. Управляющие структуры и функции

В Java или C++ мы различаем *выражения*, такие как `3 + 4`, и *инструкции*, например `if`. Выражение имеет значение; инструкция выполняет действие. В Scala практически все конструкции имеют значения, то есть являются *выражениями*. Это позволяет писать более короткие и более удобочитаемые программы.

3.1. Условные выражения

В Scala `if/else` возвращает значение, а именно значение выражения, следующего за `if` или `else`. Например,

```
val x: Int = 10
val n = if (x > 0) 1 else -1  // тернарное выражение
```

В Python это выглядело бы так

```
x = 10
n = 1 if (x > 0) else -1
```

В Scala *каждое выражение имеет тип*. Например, выражение `if (x > 0) 1 else -1` имеет тип `Int`, потому что обе ветви имеют тип `Int`. Типом выражения, способного возвращать значения разных типов, такого как `if (x > 0) "positive"else -1`, является супертип для обеих ветвей. В данном примере одна ветвь имеет тип `java.lang.String`, а другая – тип `Int`. Их общий супертип называется `AnyVal`.

Может получиться так, что инструкция `if` не будет иметь значения. Например, в случае, когда `if (x > 0) 1`, а `x` отрицательный. Однако в Scala каждое выражение предполагает наличие какого-либо значения. Эта проблема была решена введением класса `Unit`, единственное значение которого записывается как `()`[1]. Инструкция `if` без ветви `else` эквивалентна инструкции

```
if (x > 0) 1 else ()
```

[1]Эту комбинацию можно воспринимать как пустое значение и считать тип `Unit` аналогом типа `void` в Java или C++. Но, строго говоря, `void` означает отсутствие значения, тогда как `Unit` имеет единственное значение, означающее «нет значения»

Многострочное выражение в интерактивной оболочке можно заключить в фигурные скобки

```
{
  if (x > 0) 1
  else
    if (x == 0) 0
    else -1
}
```

Если потребуется перенести длинную строку на другую строку, первая строка должна оканчиваться символом, который не может интерпретироваться как конец инструкции. Для этого подойдет любой оператор

```
s = s0 + (v - v0)*t + // оператор + сообщает парсеру, что это не конец
0.5*(a - a0)*t*t
```

На практике длинные строки можно обрамлять фигурными скобками

```
if (n > 0) {  // открывающая скобка в конце строки явно свидетельствует,
              // что инструкция будет продолжена на следующих строках
  r = r*n
  n -= 1
}
```

В языке Scala блок `{}` содержит последовательность *выражений* и сам считается выражением, результатом которого является результат последнего выражения.

Это может пригодиться для инициализации значений `val`, когда требуется выполнить более одного действия. Например

```
import scala.math
val x: Int = 10
val x0: Int = 1
val y: Int = 24
val y0:Int = 50
val distance = { val dx = x - x0; val dy = y - y0; scala.math.sqrt(dx*dx + dy*dy) }
// вернет 27.51363298439521
```

Поскольку инструкции присвоения возвращают значение `Unit`, их нельзя объединять в цепочки

```
// -- Scala
x = y = 1 // Неправильно!
```

В Python можно

```
# здесь просто x и y ссылаются на один и тот же объект целочисленного типа данных со значением 1
x = y = 1
```

4. Ввод и вывод

Чтобы вывести значение, используйте функцию `print` или `println`. Последняя добавляет символ перевода строки в конце. Имеется также функция `printf`, принимающая строку описания формата в стиле языка C

```
printf("CAE-package %s has %d cores", "Nastran", 32)
```

Прочитать строку, введенную в консоли с клавиатуры, можно с помощью функции `readLine`. Чтобы прочитать число, логическое или символьное значение, используйте `readInt`, `readDouble`, `readByte`, `readShort`, `readLong`, `readFloat`, `readBoolean` или `readChar`.

Метод `readLine`, в отличие от других, принимает строку приглашения к вводу

```
import scala.io.StdIn.readLine
import scala.io.StdIn.readInt
val name = readLine("Your name: ") // ввод имени
print("Your age: ")
val age = readInt()  // ввод возраста
printf("Hello, %s! Next year, you will be %d.", name, age+1)
// Hello, Leor! Next year, you will be 33.
```

5. Циклы

В Scala отсутствует прямой аналог цикла `for (инициализация;проверка;обновление)`. Если такой цикл потребуется, у вас есть два варианта на выбор – использовать цикл `while` или инструкцию `for`, как показано ниже

```
val n:Int = 10 // константа
var r: Int = 1 // переменная со значением по умолчанию
for (i <- 1 to n)
    r = r*i
printf("Result: %d", r) // Result: 3628800
```

В Python эта задача решалась бы так

```
n = 10
r = 1
for i in range(1,n+1):
    r *= i
print(f'Result: {r}')  # Result: 3628800
```

Вызов `1 to n` вернет объект `Range`, представляющий числа в диапазоне от 1 до `n` (включительно).

Конструкция

```
for (i <- expr)
```

обеспечивает последовательное присваивание переменной `i` всех значений выражения `expr` справа от `<-`. Порядок присвоения зависит от типа выражения. Для коллекций, таких как `Range`, присвоит переменной `i` каждое значение по очереди.

Перед именем переменной в цикле `for` не требуется указывать `val` или `var`. Тип переменной соотвествует типу элементов коллекции. Область видимости переменной цикла ограничивается телом цикла.

Для обхода элементов строки или массива зачастую нужно определить диапазон от 0 до $n-1$. В этом случае используйте метод `until` вместо `to`. Он возвращает диапазон, не включающий верхнюю границу. Например

```
val s: String = "Hello"
var sum: Int = 0
for (i <- 0 until s.length)
  sum += s(i)
```

В действительности в данном примере нет необходимости использовать индексы. Цикл можно выполнить непосредственно по символам

```
var sum: Int = 0
for (ch <- "Hello")
  sum += ch
```

Как и в Python. В Python тоже можно перебирать элементы последовательности прямо в цикле без индексов. В Scala циклы используются те так часто, как в других языках. Значения в последовательностях зачастую можно обрабатывать, применяя функцию сразу ко всем элементам, для чего достаточно произвести единственный вызов метода.

В Scala нет инструкций `break` или `continue` для преждевременного завершения цикла. Но как же быть, если это потребуется? Есть несколько вариантов:

1. Использовать логическую переменную управления циклом,
2. Используйте вложенные функции – при необходимости можно выполнить инструкцию `return` в середине функции,
3. Используйте метод `break` объекта `Breaks`

```
// здесь передача управления за пределы цикла выполняется путем возбуждения и перехвата и
   сключения, поэтому избегайте пользоваться этим механизмом, когда скорость выполнения кр
   итична
import scala.util.control.Breaks._
breakable {
  for (...) {
    if (...) break; // выход из прерываемого блока
      ...
  }
}
```

5.1. Расширенные циклы for и for-генераторы

В предыдущем разделе была представлена базовая форма цикла `for`. Однако эта конструкция намного богаче, чем в Java или C++. В заголовке цикла `for` допускается указывать несколько *генераторов* в форме *переменная* <- *выражение*, разделяя их «;»

```
// --- Scala
// несколько генераторов
for (i <- 1 to 3; j <- 1 to 5)
  print(f"${10*i + j} ")
```

```
# --- Python
# вложенные циклы
for i in range(1, 3+1):
    for j in range(1, 5+1):
        print(f'{10*i+j} ')
```

Что касается форматирования, то можно использовать конструкции с f-строками. Пример

```
import scala.io.StdIn.readLine
import scala.io.StdIn.readInt

val package_name = readLine("Enter package's name: ")
print("Enter number of cores: ")
val n_cores = readInt()
print(f"CAE-package: ${package_name}, number of cores: ${n_cores}") // f-строка
```

Управлять форматом вывода чисел можно так

```
val n: Double = 0.345345345345
print(f"This is a number: ${n}%.3f") // This is a number: 0,345
```

Каждый генератор может иметь *ограничитель* – логическое условие с предшествующим ему ключевым словом `if`

```
// --- Scala
for (i <- 1 to 3; j <- 1 to 3 if i != j)
  print(f"${10*i + j} ")
```

```
# --- Python
for i in range(1,3+1):
    for j in range(1,3+1):
        if i != j:
            print(f'{10*i + j} ', end=' ')
```

Допускается любое количество *определений*, вводящих переменные для использования внутри цикла

```
// --- Scala
for (i <- 1 to 3; from = 4 - i; j <- from to 3)
  print(f"${10*i + j} ")
  // 13 22 23 31 32 33
```

На Python эта задача могла бы быть решена так

```
for i in range(1,3+1):
    frm = 4 - i
  for j in range(frm,3+1):
      print(f'{10*i + j} ')
# 13 22 23 31 32 33
```

Список литературы

1. *Хостаманн К.* Scala для нетерпеливых. – М.: ДМК Пресс, 2013. – 408 с.