Функциональное программирование

Для разработки программ на языке Scala и выполнения лабораторных работ потребуются следующие программные средства:

1. Установленная виртуальная машина java (JRE или JDK от Oracle):

$ java -version

2. Установленная Scala:

$ scala -version

Установка scala происходит также как и java: скачивается архив с последней стабильной версией с сайта http://www.scala-lang.org/downloads, распаковывается в любую директорию и в переменную окружения PATH добавляется путь на подкаталог bin этой директории. В Ubuntu для этого редактируется файл /etc/enviroment.

Для информирования различных других программ о местонахождении java и scala также бывает полезно задать переменные окружения JAVA_HOME, JDK_HOME и SCALA_HOME.

После выполнения этих шагов Вам будет уже доступен интерактивный интерпретатор Scala (REPL), с помощью которого, используя любой текстовый редактор с подсветкой синтаксиса языка Scala (geany, gedit, ...), можно разрабатывать небольшие программы на этом языке.

Система сборки sbt.

По некоторым причинам для Scala полноценное использование IDE (NetBeans, Intellij IDEA, Eclipse) так, как это было для Java, - невозможно.

Для всех этих сред существуют плагины для поддержки подсветки синтаксиса языка и автодополнения, однако компиляция и сборка проходят в них крайне медленно, да и полноценная отладка кода также невозможна. Поэтому лучшим решением является использование для компиляции и сборки проекта системы sbt — simple build tool, основанной на репозиториях maven.

Установка: скачиваем архив с ресурса: http://www.scala-sbt.org/release/docs/GettingStarted/Setup.html, распаковываем в любой каталог (фактически из архива нужны только два файла: sbt-launch.jar и запускающий скрипт sbt) и добавляем путь на этот каталог в переменную окружения PATH. После установки команда sbt должна вызываться из любого каталога файловой системы.

Среда разработки Intellij IDEA.

Для более сложных проектов, содержащих множество каталогов и файлов, а также для фреймворка play 2, будем использовать IDE Intellij IDEA community edition (свободное программное обеспечение). В основном, для ввода текстов программ и быстрой навигации по файлам проекта.

Установка: скачиваем с ресурса http://www.jetbrains.com/idea/download/index.html архив с последней версией Intellij IDEA community edition, распаковываем в некоторый каталог и размещаем на рабочем столе запускающий скрипт, типа:

```
#!/bin/sh
cd /home/george/Soft/idea-12/bin
sh idea.sh
```

Для поддержки Scala и sbt нужно установить необходимые плагины. Делается это в самой программе Intellij IDEA: Главное меню → File → Settings → Plugins → Browse repositories → ввод в строке поиска scala → выбор Scala и SBT. После выбора произойдет скачивание выбранных плагинов и их установка.

Язык Scala

Scala - статически типизированный объектно-ориентированный язык программирования, на котором можно создавать приложения для jvm (виртуальной java-машины). Поддерживает как императивный так и функциональный стили программирования. Компилятор (scalac) для jvm написан на java Мартином Одерски (Швейцария, Лозанна). Существует также компилятор для платформы .NET.

На Scala одну и ту же программу можно написать совершенно по-разному.

Рассмотрим, как пример, следующую задачу:

f(n) = сумме всех натуральных чисел меньших n, делящихся на 3 или на 5.

Написать программу для вычисления n → f(n)

1. Решение на java:

```
int f (int n) {
        int s = 0;
        for (int i=1; i<n; i++) {
         if ((i%3==0)||(i%5==0)) s+=i;
         }
    return s;
}
```

2. Решение на Scala (императивный стиль — как в java):

```
def f(n: Int) = {
    var s = 0
    for (i <- 1 to n-1) {
    if ((i%3==0)||(i%5==0)) s+=i
     }
     s
    }
```

3. Решение на Scala (функциональный стиль):

```
def f(n: Int) = (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
```

4. Решение на Scala (задание функции как объекта):

```
val f = (n: Int) => (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
```

Для того чтобы протестировать пример на java придется дописать еще метод main:

```
public class Test {
```

```
 static int f (int n) {
 int s = 0;
 for (int i=1; i<n; i++) {
 if ((i%3==0)||(i%5==0)) s+=i;
 }
 return s;
 }
 public static void main(String[] args) {
 if (args.length>0) System.out.println(f(Integer.parseInt(args[0])));
 else System.out.println("Не задан аргумент");
 }
}
```

Примеры же на Scala можно легко протестировать в интерактивном интерпретаторе Scala (REPL), просто скопировав и вставив в него исходный код функции.

```
$ scala
Type in expressions to have them evaluated.
Type :help for more information.
scala> def f(n: Int) = (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
f: (n: Int)Int
scala> f(999)
res0: Int = 232169
scala>
```

Интерактивный интерпретатор обладает возможностью автодополнения кода для используемых классов Scala и Java и помогает быстро проверить как работает тот или иной метод, без необходимости чтения документации.

Для использования программы вне интерпретатора, необходимо, как и в java создать объект, реализующий метод main.

```
object Test {
 def f(n: Int) = (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
 def main(args: Array[String]) = {
 if (args.length>0) println(f(args(0).toInt))
 else println("Не задан аргумент")
 }
}
```

Сохраним этот текст в файл с именем script.scala. Теперь мы можем его использовать просто как скрипт, выполняя команду типа:

```
$ scala script.scala 999
```

Но можем также его скомпилировать для jvm:

```
$ scalac script.scala
```

Правда запустить программу на выполнение командой (как для java):

$ java Test 999

не удастся, поскольку библиотека Scala (scala-library.jar) не содержится в classpath,

однако, если мы добавим ее в classpath, то все запустится:

$ java -cp .:/srv/server/scala-2.10.0/lib/scala-library.jar Test 999

232169

Из этого примера уже можно заметить снятие в синтаксисе Scala многих ограничений языка Java и также некоторые существенные различия:

1. Необязательность точки с запятой в конце строк

2. Имя объекта с методом main не обязано совпадать с именем файла

3. Необязательность задания типа переменной, если он может быть вычислен компилятором из контекста

4. Необязательность директивы return для возвращаемого значения

5. Необязательность круглых скобок у метода без аргументов (sum = sum())

6. Возможность объявить не класс, а объект устраняет директивы static

7. Типы объявляются после переменной или функции через двоеточие (n: Int)

8. Элементы массива(списка) выбираются при помощи круглых, а не квадратных скобок (args(0))

9. Типизация массива(списка) задается квадратными, а не угловыми скобками (Array[String])

10.Функции определяются директивой def, изменяемые переменные var, неизменяемые val (тем самым отпадает необходимость в директиве final)

11. При задании функций между определением и реализацией используется знак = как и при задании переменных

Отметим еще, что в Scala нет примитивных типов и числа сразу являются объектами (как строки), поэтому арифметическая операция +, например, это обычный метод у объекта-числа:

```
scala> val a = 2; val b = 3
a: Int = 2
b: Int = 3
scala> a + b
res17: Int = 5
scala> a.+(b)
res18: Int = 5
scala>
```

Сравнение == объектов в Scala осуществляется по их внутреннему содержанию, а не по указателям, как в java:

```
scala> val x = "abc"; val y = "abc"
x: String = abc
y: String = abc
scala> x == y
```

```
res32: Boolean = true
scala> val l = List(1, 2, 3, 2, 1)
l: List[Int] = List(1, 2, 3, 2, 1)
scala> l == l.reverse
res33: Boolean = true
scala>
```

Для значительной части стандартных операций Java в Scala имеются их упрощенные эквиваленты, например так можно ввести строку в консоли и преобразовать ее к целому:

```
scala> val x = readLine
x: String = 12345
scala> x.toInt
res2: Int = 12345
scala>
```

При этом, если какая-то функциональность отсутствует в Scala, то Вывсегда можете использовать любые стандартные классы java, например,

```
 scala> def date = {
 | import java.util.Date
 | import java.text.SimpleDateFormat
 | (new SimpleDateFormat("dd.MM.yyyy k:m:s")).format(new Date)
 | }
date: String
scala> date
res10: String = 06.01.2013 17:42:52
scala>
```

Обратите внимание на то, что import можно делать прямо внутри функции. Кстати, вместо * для импортирования всех классов пакета используется символ _

```
import java.util._
```

На Scala легко начать писать программы в императивном стиле, если Вы знаете Java. И даже в этом случае Вы уже получите значительную свободу от ограничений языка Java. Однако главная задача - научиться использовать функциональные возможности языка Scala. Проанализируем, например, наше решение:

```
def f(n: Int) = (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
```

Лучше всего это сделать в интерактивном интерпретаторе Scala:

```
scala> val n = 10
n: Int = 10
scala> 1 to n-1
res19: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8, 9)
scala> (1 to n-1).filter(x => (x%3==0) || (x%5==0))
res20: scala.collection.immutable.IndexedSeq[Int] = Vector(3, 5, 6, 9)
```

```
scala> (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
res21: Int = 23
scala>
```

При n = 10 мы видим, что 1 to n-1 возвращает тип Range(1, 2, 3, 4, 5, 6, 7, 8, 9) – диапазон чисел от 1 до 9. У Range есть метод filter, которому в качестве аргумента передается анонимная функция x => (x%3==0) || (x%5==0) и этот метод выбирает (фильтрует) только те элементы, на которых эта функция возвращает true. В результате мы получаем Vector(3, 5, 6, 9) , у которого есть метод sum, возвращающий сумму элементов вектора.

Как научиться писать программы в функциональном стиле на Scala? Прежде всего следует изучить возможности самого языка, в котором по сравнению с Java добавлено значительное число уже готовых функций для этого (таких как filter, map, exists, count, find, …).

С другой стороны этого может оказаться недостаточно, нужно научиться мыслить по-другому и использовать абстракции. Нужно привыкнуть к тому, что функции — это обычные объекты типа (x: Int, y: Int) => x + y и мы можем писать функции, которые возвращают объекты-функции.

Как правило, если нам нужно написать метод в функциональном стиле и в языке нет готовых функций для этого, то следует искать возможность использования рекурсии. Рекомендуется внимательно изучить примеры главы «Функции первого класса» из книги Мартина Одерски «Scala в примерах»: http://ru.wikibooks.org/wiki/Scala_%D0%B2_%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D1%80%D0%B0%D1%85

Задание на лабораторную работу

1. Переменные res – это значения val или настоящие переменные var?
2. "crazy" * 3 в REPL
3. Что означает выражение 10 max 2? В каком классе определен метод max?
4. Используя число типа BigInt, вычислите 2^1024
5. Что нужно импортировать, чтобы найти случайное простое число вызовом метода probablePrime(100, Random) без использования каких-либо префиксов перед именами probablePrime и Random?
6. Один из способов создать файл или каталог со случайным именемсостоит в том, чтобы сгенерировать случайное число типа BigInt и преобразовать его в систему счисления по основанию 36, в результате получится строка, такая как "qsnvbevtomcj38o06kul". Отыщите в Scaladoc методы, которые можно было бы использовать для этого.
7. Как получить первый символ строки в языке Scala? А последний символ?

8. Что делают строковые функции take, drop, takeRight и dropRight? Какие преимущества и недостатки они имеют в сравнении с substring?

9. Сигнум числа равен 1, если число положительное. -1 – если отрицательное, и 0 – если равно нулю. Напишите функцию, вычисляющую это значение.

10. Какое значение возвращает блок {}? Каков его тип?

11. Напишите на языке Scala цикл, эквивалентный циклу на языке Java

for (int i=10; i>=0; i--) System.out.println(i);

12. Напишите процедуру countdown (n: Int), которая выводит числа от n до 0

13. Напишите цикл for для вычисления кодовых пунктов Юникода всех букв в строке. Например, произведение символов в строке «Hello» равно 9415087488L.

14. Решите предыдущее упражнение без применения цикла. Напишите функцию product(s: String), вычисляющую произведение, как описано в предыдущих упражнениях.

16. Сделайте функцию из предыдущего упражнения рекурсивной.

17. Напишите функцию, вычисляющую xn, где n – целое число.

Используйте следующее рекурсивное определение:

• xn = y2, если n – четное и положительное число, где y=xn/2

• xn = x*xn-1, если n – нечетное и положительное число.

• x0=1.

• xn=1/x-n, если n – отрицательное число.

Не используйте инструкцию return.

18. f(m,n) - сумма всех натуральных чисел от m до n включительно, в десятичной записи которых нет одинаковых цифр.

19. Список содержит целые числа, а также другие списки, такие же как и первоначальный. Получить список, содержащий только целые числа из всех вложенных списков. Пример:

f(List(List(1, 1), 2, List(3, List(5, 8)))) = List(1, 1, 2, 3, 5, 8)

20. f(n) - сумма цифр наибольшего простого делителя натурального числа n.

21. Список содержит элементы одного, но любого типа. Получить список, содержащий каждый имеющийся элемент старого списка k раз подряд. Число k задается при выполнении программы.

22. f(n) - сумма цифр наибольшего простого делителя натурального числа n.

23. Список содержит элементы одного, но любого типа. Получить список, содержащий каждый имеющийся элемент старого списка k раз подряд. Число k задается при выполнении программы.

24. f(m,n) - наименьшее общее кратное натуральных чисел m и n.

25. Список содержит элементы одного, но любого типа. Получить список, из элементов исходного, удаляя каждый k-й элемент. Число k задается при выполнении программы.

26. f(n,k) - число размещений из n по k. Факториал не использовать.

27. Список содержит элементы одного, но любого типа. Получить новый список, перемещая циклически каждый элемент на k позиций влево (при перемещении на одну позицию первый элемент становится последним, второй первым и так далее). Число k задается при выполнении программы. Если k отрицательное, то перемещение происходит вправо.

28. f(n) - наибольшее совершенное число не превосходящее n. Совершенным называется натуральное число n равное сумме своих делителей, меньших n, например 6 = 1 + 2 + 3 (f(6) = 6, f(7) = 6, ...).

29. Список содержит элементы одного, но любого типа. Получить два списка из элементов исходного, выбирая в первый элементы с четными индексами, а во второй с нечетными.

30. 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 = 83

31. Список в качестве элементов содержит кортежи типа: (n, s), где n — целые числа, а s — строки. Получить два списка из элементов исходного, выбирая в первый числа, а во второй строки из кортежей.