КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра анализа данных и технологий программирования

А.И. ЕНИКЕЕВ

ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ SCALA

Казань – 2023

УДК 025.4.03

ББК 32.971.353

Принято на заседании учебно-методической комиссии ИВМИТ Протокол № 6 от 22 февраля 2023 года

Рецензенты:

кандидат физико-математических наук, доцент кафедры анализа данных и технологий программирования КФУ **Р.А. Бурнашев.**

Еникеев А.И.

Функциональное программирование на языке SCALA /А.И.Еникеев- – Казань: Казанский федеральный университет, 2023. – 16 с.

Методическое пособие посвящено изучению функциональных методов программирования на языке SCALA. Язык SCALA является современным объектно- ориентированным языком программирования, включающим в себя в качестве подмножества функциональные средства программирования.

Настоящее учебно-методическое пособие адресовано, в первую очередь, студентам таких специальностей, как «Прикладная информатика», «Бизнес-информатика», «Прикладная математика и информатика» и т.д., а также широкому кругу читателей, интересующихся направлением в области функционального программирования.

СОДЕРЖАНИЕ

Введение

SCALA является мультипарадигмальным языком программирования, поддерживающим императивное, объектно- ориентированное и функциональное программирование и др[1].

1.Парадигмы программирования

Парадигма программи́рования — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Основными парадигмами программирования являются императивная, функциональная и логическая. Объектная ориентация ортогональна делению на парадигмы, то есть объектная ориентация может быть в любой из перечисленных парадигм. Парадигма программирования не определяется однозначно языком программирования; практически все современные языки программирования в той или иной мере допускают использование различных парадигм мультипарадигмальное программирование). Так, на языке Си, который не является объектно-ориентированным, можно работать в соответствии с принципами объектно--ориентированного программирования, хотя это и сопряжено с определёнными сложностями; функциональное программирование можно применять при работе на любом императивном языке, в котором имеются функции, и т. д.

Также существующие парадигмы зачастую пересекаются друг с другом в деталях (например, модульное и объектно-ориентированное программирование) , поэтому можно встретить ситуации, когда разные авторы употребляют названия из разных парадигм, говоря при этом, по сути, об одном и том же явлении. Императивное программирование связано с изменением значений изменяемых переменных, использованием присваиваний и управляющих структур, таких как

условное исполнение, циклы, выходы и продолжения, возвраты из процедур. Функциональное программирование (ФП) в узком смысле означает программирование без изменяемых переменных, присваиваний, циклов и других императивных управляющих структур. В широком смысле — это программирование, сосредоточенное на функциях. В частности, функции могут выступать в качестве порождаемых, используемых и сочетаемых значений. Языки Функционального Программирования (ЯФП) в широком смысле позволяют строить изящные программы, сосредоточенные на функциях. В частности, функции выступают как полноправные значения, которые могут определяться где угодно (в том числе, внутри других функций); передаваться в качестве параметров и возвращаться в качестве результатов других функций; подвергаться композиции. Примеры ЯФП: LISP, SCALA, RUBY, HASKELL, OCAML и другие. Большую популярность в последнее время получил язык SCALA.

Растущая популярность языка SCALA обусловлена привлекательной методикой задействования параллелизма при многоядерных и облачных вычислениях. В создании и разработке языка SCALA приняли участие следующие ученые [1] .

Мартин Одерски, создатель языка Scala, — профессор Федеральной политехнической школы Лозанны, Швейцария (EPFL), и основатель Lightbend, Inc. Работает над языками программирования и системами, в частности над темой совмещения объектно-ориентированного и функционального подходов. С 2001 года сосредоточен на проектировании, реализации и улучшении Scala. Ранее внес вклад в разработку Java, выступив соавтором обобщенных типов и создателем текущего эталонного компилятора javac. Мартину присвоено звание действительного члена ACM.

Лекс Спун — разработчик программного обеспечения в Semmle, Ltd. Занимался разработкой Scala на протяжении двух лет в ходе постдокторантуры в EPFL. Свою докторскую степень получил в Технологическом институте

Джорджии, где темой его исследований был статический анализ динамических языков. Помимо Scala, участвовал в разработке самых разнообразных языков, включая динамический язык Smalltalk, научный язык X10 и логический язык, лежащий в основе Semmle. В настоящее время Лекс Спун проживает в Атланте.

Билл Веннерс занимает должность президента Artima, Inc. Занимается публикацией материалов на сайте Artima (www.artima.com), предоставляет консультации,учебные материалы, книги и инструменты, относящиеся к Scala. Автор книги *Inside the Java Virtual Machine*, в которой программисты могут познакомиться с архитектурой и внутренним устройством платформы Java. Его популярные статьи в журнале JavaWorld посвящены внутреннему строению Java, объектно-ориентированному проектированию и Jini. Билл занимает позицию представителя сообщества в консультативном совете Scala Center и является ведущим разработчиком и проектировщиком фреймворка тестирования ScalaTest и библиотеки Scalactic, предназначенной для функционального и объектно-ориентированного программирования.

2.Элементы функционального программирования на языке SCALA

Для программирования на на Scala разработан ряд облачных компиляторов, среди которых рекомендуется использовать «compile scala online – Rextester»
В этом компиляторе программирование осуществляется посредством так называемого трейта scala.App. Трейты в Scala являются фундаментальными повторно используемыми блоками кода. В трейте инкапсулируются определения тех методов и полей, которые затем могут повторно использоваться путем их примешивания в классы. В отличие от наследования классов, в котором каждый класс должен быть наследником только одного суперкласса, в класс может примешиваться любое количество трейтов.

2.1.Примеры кодов

1) object Rextester extends App {

println("Hello, World!")

Результат: вывод текста "Hello, World!".

2)object Rextester extends App {

def add(x:Int, y:Int) = x + y;

print("sum of x + y = " + add(25,10)); }

Результат: sum of x + y = 35

3) object Rextester extends App{

def cube(x:Int)={x*x*x};

print("cube of x = " + cube(2)); }

Результат: *cube of x = 8*

2.2.Условные выражения

Условные выражения используются для выбора между двумя альтернативами. Например:

4) def abs(x: Int) = if (x >= 0) x else –x

println((abs(-5))) }

Результат: 5

«x >= 0 » здесь выступает предикатом, предикат имеет тип Boolean . Скобки вокруг предиката в скале опускать нельзя.

В логическом выражении могут использоваться:

— константы true и false;

— логические операции отрицание (!b),

конъюнкция (b1 && b2), дизъюнкция (b1 || b2) (где b,

b1, b2 — логические выражения);

— операции сравнения (e1 <= e2), (e1 >= e2), (e1 < e2), (e1 > e2), (e1 == e2), (e1 != e2) (где e1, e2 — выражения численного типа или другие сравнимые).

5)object Rextester extends App {

def max(x: Int, y: Int): Int = {

if (x > y) x else y}

println((max(5,7))) }

Результат: 7

6) object Rextester extends App { def fact(n: Int): Int =

{if (n==0) 1 else n*fact(n-1)};

println(fact(5))}

Результат:*120*

7) object Rextester extends App {

def nod(m:Int,n:Int): Int =

{if (m % n==0) n else nod(n,m % n)};

println(nod(8,12)) }

Результат: 4

2.3.Работа с переменными

В Scala имеется две разновидности переменных:

val-переменные и var-переменные.

После инициализации val-переменная уже никогда не

может быть присвоена повторно. В отличие от нее

var-переменной может быть присвоено значение

повторно. На следующих ниже примерах иллюстрируются val-переменные

и var-переменные.

1) object Rextester extends App {

```
    val msg = "Hello, world!";
    println(msg) }
```
2)
```
object Rextester extends App {
    var msg:java.lang.String="";
    msg = "Hello";
    msg =msg+" world"
    println(msg) }
```
4)
```
object Rextester extends App {
    var i = 0
    while (i < 10) {
    print(i)
    i=i+1
    }     }
```

2.4. Передача функций в качестве параметров

В ЯФП функции, как и любые другие значения, они могут передаваться в качестве параметров другим функциям и возвращаться ими в качестве результата. Это обеспечивает гибкость при составлении программ. Функции, принимающие в качестве параметра или возвращающие в качестве значения другие функции, называются функциями высокого порядка.

Рассмотрим ряд примеров:

1) Суммирование всех целых чисел от a по b.

```
object Rextester extends App {
  def sumInts(a: Int, b: Int): Int =
if (a > b) 0 else a + sumInts(a + 1, b);
print(sumInts(5,7))}
```

2) Суммирование кубов всех целых от a по b

```
object Rextester extends App {
def cube(x: Int): Int = x * x * x
def sumCubes(a: Int, b: Int): Int =
if (a > b) 0 else cube(a) + sumCubes(a + 1, b)
print(sumCubes(1,3)) }
```

3) Суммирование факториалов всех целых от a по b

```
object Rextester extends App {
def fact(n: Int): Int =
{if (n==0) 1 else n*fact(n-1)};
def sumфакты(a: Int, b: Int): Int =
{if (a > b) 0 else fact(a)+sumфакты(a + 1, b)}
print(sumфакты(1,3)) }
```

Можно ли обнаружить общую закономерность?

```
def sum(f: Int , Int, a: Int, b: Int): Int = if (a > b) 0  else f(a) + sum(f, a + 1, b)
def sumInts(a: Int, b: Int) = sum(id, a, b)
def sumCubes(a: Int, b: Int) = sum(cube, a, b)
def sumфакты(a: Int, b: Int) = sum(fact, a, b)
def id(x: Int): Int = x
def cube(x: Int): Int = x * x * x
def fact(x: Int): Int = if (x == 0) 1 else fact(x – 1)
```

Тип A ⇒ B — это тип функции, принимающей аргумент типа A и возвращающей значение типа B.

В конечном итоге:

```
object Rextester extends App{
def sum( f:Int =>Int, a: Int, b: Int):Int={if (a>b) 0
else f(a) + sum(f, a + 1, b)}
```

```
def id(x: Int): Int = x
def cube(x: Int): Int = x * x * x
def fact(n: Int): Int =
{if (n==0) 1 else n*fact(n-1)};
def sumInts(a: Int, b: Int) = sum(id, a, b)
def sumCubes(a: Int, b: Int) = sum(cube, a, b)
def sumфакты(a: Int, b: Int) = sum(fact, a, b)
print(sumInts(5,8)) }
```

2.5. Массивы, примеры программ

```
object Rextester extends App {
var fiveToOne=new Array[Int](5)
fiveToOne=Array(1,2,3,4,5)
    for (i <- 0 to 4)
      print(fiveToOne(i)) }
object Rextester extends App {
   val greetStrings = new Array[String](3)
    greetStrings(0) = "Hello"
    greetStrings(1) = ", "
greetStrings(2) = "world!"
for (i <- 0 to 2)
 print(greetStrings(i)) }
```

2.6. Списки. Основные операции над списками

Все действия со списками можно свести к следующим основным операциям:

L.head возвращает первый элемент списка L (аналог операции (*CAR L*) в

языке LISP [2]);

L.tail возвращает список, состоящий из всех элементов, за исключением первого (аналог операции (*CDR L*) в языке LISP) ;

L.isEmpty возвращает true, если список пуст и - false в противном случае (аналог операции NULL L) в языке LISP) .

s::t -аналог операции (cons s t) в языке LISP)

s:::t – аналог операции (append s t) в языке LISP)

Эти операции определены как методы класса List.

Методы head и tail определены только для непустых списков.

Будучи примененными к пустому списку, они генерируют ошибку.

2.7.Списки. Примеры программ

1) Нахождение максимального элемента в числовом списке:

```
 object Rextester extends App {
var v:Int=0
def max2(x: Int, y: Int): Int =
   {if (x > y) x else y};
def max(L:List[Int]):Int = {
var s:List[Int]=Nil;
s=L.tail;
if (s==Nil) (L.head) else max2(L.head, max(L.tail))};
v=max(List(1,2,3,45,22));
println(v)  }
```

2) Суммирование элементов в числовом списке.

```
 object Rextester extends App {
var v:Int=0
def sum(L:List[Int]):Int = {
if (L.tail==Nil) (L.head) else L.head+sum(L.tail)};
v=sum(List(1,2,3));
println(v) }
```

3) Слияние списков

```
object Rextester extends App {
val oneTwo = List(1, 2)
val threeFour = List(3, 4)
val oneTwoThreeFour = oneTwo ::: threeFour
println(oneTwo + " и " + threeFour + " не изменились.")
println("Следовательно, " + oneTwoThreeFour + " является новым списком.")
}
```

3.Заключение

SCALA считается языком программирования общего назначения, и в теории подходит для решения любого рода задач - от создания десктопных программ до написания серверных приложений. При этом чаще всего язык применяется для разработки в веб-среде, особенно когда дело касается объемных, нагруженных проектов вроде социальных сетей, многостраничных информационных порталов и т.п. SCALA во многом представляет из себя улучшенную версию Java с расширенными возможностями для программирования в функциональном стиле. Чаще всего SCALA применяется для работы с веб-проектами и для создания кроссплатформенных приложений. Выбрав SCALA в качестве основного языка разработки, можно сократить время на реализацию программного продукта и расширить возможности его применения.

ЛИТЕРАТУРА

1. **Одерски М., Спун Л., Веннерс Б.,**

О-41 Scala. Профессиональное программирование. 4-е изд. — СПб.: Питер, 2021. —720 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1827-4

2. Горлянский С. П. Функциональное программирование. Основы языка Лисп: реализация алгоритмов и решение задач. — Казань: Бук, 2023. — 3052 с. —ISBN 978-5-907665-30-9.

Учебное издание

Еникеев Арслан Ильясович

ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ SCALA

Учебное пособие