Гуляев Г.М.

Современные технологии программирования (часть 2)

Лекция 1. Язык Scala

Курс лекций для студентов АлтГТУ

Языки программирования для jvm

- **Для виртуальной машины java (jvm) сегодня можно писать программы не только на языке java:**
  - **Groovy** (ООП, типизация: динамическая, возможен синтаксис java)
  - **JRuby** (ООП, типизация: динамическая, ruby для jvm)
  - **Jython** (ООП+функц, типизация: динамическая, python для jvm)
  - **Clojure** (Функц, типизация: динамическая, диалект Lisp)
  - **Scala** (ООП+функц+конкурент, тип: статическая, возможна java)
  - **Kotlin** (универс, тип: статическая, компил-ся в jvm и JavaScript)
  - **Rhino**, **Ceylon**, **Fantom** - другие языки
- **Для больших проектов: статическая типизация, ООП, функц. программирование, интеграция с java**
- **Scala - компилятор для jvm разработан в 2003 году группой под руководством Мартина Одерски**
- **Объектно-функциональная модель (scala) - это интеграция функционального и объектного подхода**

Особенности использования scala

Преимущества

1. Все преимущества java и jvm (статическая типизация, мультиплатформенность, быстродействие, библиотеки java, …)
2. Логичный синтаксис, отсутствия дублирования кода
3. Функциональное программирование (ФП) упрощает код
4. Неизменяемые объекты - более безопасный код
5. Конкурентное программирование (actors)
6. Интерактивный интерпретатор (REPL)

Недостатки

1. Много свободы (много способов решения одной и той же задачи)
2. Высокий порог вхождения (требуется понимание java и ФП)
3. Слабая поддержка IDE (медленная компиляция, нет отладки)

Решения

- Выработать свой стиль программирования
- Использовать IDE только для ввода кода
- Использовать для сборки другие средства (sbt)

Рекомендации и требования

- **Для понимания материала лекций и выполнения лабораторных рекомендуется использование ОС Linux (демонстрация примеров на лекциях в Ubuntu 12.04 LTS)**
- **В качестве среды разработки в большинстве случаев будет использоваться NetBeans и Intellij IDEA (для scala)**
- **Программирование - это, прежде всего, правильно спроектированная реализация задачи и хорошо написанный код, поэтому в работах главное идея и код (аргументы типа «она работает» - не принимаются)**
- **Оформленные в соответствии с СТП работы присылаются на aomai.ru. Документы в формате pdf, архивы в формате tar.gz (Linux). Форматы MS Office, rar и т. п. не принимаются**
- **Защита работы возможна только после получения ее электронной копии преподавателем. На защите студент должен объяснить идею решения задачи и особенности ее реализации в коде и ответить на вопросы**

Требования к оформлению кода на Scala

- Отступы для блоков кода (рекомендуется 2 пробела)
- Имена пакетов только из строчных букв
- Имена классов начинать с большой буквы (MyClass)
- Имена полей и методов с маленькой (newWord)
- В составных именах каждое новое подслово с заглавной
- Имена констант состоят из больших букв и _ (ARRAY_MIN_VALUE)
- В именах использовать правильные английские слова
- Имя должно отражать смысл действия или состояния
- Краткость и ясность кода - главное требование
- Не использовать префиксы get и set как в java
- Дополнительные рекомендации по ссылке:

  http://twitter.github.com/effectivescala/index-ru.html#Форматирование

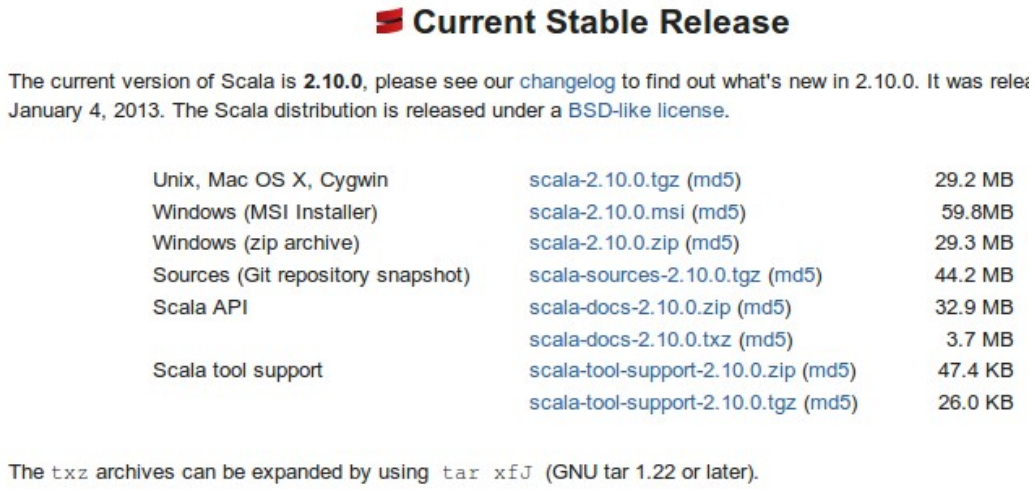
Получение и установка Scala

- С сайта

  http://www.scala-lang.org/downloads

скачиваем последнюю версию Scala

- Разворачиваем архив в любой каталог, например, scala-2.10.0



Current Stable Release

The current version of Scala is **2.10.0**, please see our changelog to find out what's new in 2.10.0. It was relea
January 4, 2013. The Scala distribution is released under a BSD-like license.

Unix, Mac OS X, Cygwin	scala-2.10.0.tgz (md5)	29.2 MB
Windows (MSI Installer)	scala-2.10.0.msi (md5)	59.8MB
Windows (zip archive)	scala-2.10.0.zip (md5)	29.3 MB
Sources (Git repository snapshot)	scala-sources-2.10.0.tgz (md5)	44.2 MB
Scala API	scala-docs-2.10.0.zip (md5)	32.9 MB
	scala-docs-2.10.0.txz (md5)	3.7 MB
Scala tool support	scala-tool-support-2.10.0.zip (md5)	47.4 KB
	scala-tool-support-2.10.0.tgz (md5)	26.0 KB

The `txz` archives can be expanded by using `tar xfJ` (GNU tar 1.22 or later).

- После установки к переменной среды PATH добавляем путь на исполняемые файлы Scala:

  для linux: :/home/user/scala-2.10.0/bin/

- В Ubuntu для этого редактируется файл /etc/enviroment:

  PATH="... :/home/user/scala-2.10.0/bin"

  JAVA_HOME="/srv/server/jdk1.7.0_09"

  JDK_HOME="/srv/server/jdk1.7.0_09"

  SCALA_HOME="/home/user/scala-2.10.0"

  CLASSPATH=".:/srv/server/jdk1.7.0_09/bin"

Интерактивный интерпретатор (REPL)

- Выполнив команду **scala** мы попадаем в интерактивный интерпретатор языка Scala, в котором можем выполнять команды и проверять работу кода:

```
scala> 8 * 5 + 2
res0: Int = 42
scala>  0.5 * res0
res1: Double = 21.0
scala> "8 * 5 + 2 = " + res0
res2: String = 8 * 5 + 2 = 42
```

- Интерпретатор Scala читает выражение, вычисляет его, печатает результат, читает следующее выражение и т. д. Поэтому его называют REPL (read-eval-print loop).
- Точнее говоря, REPL - не интерпретатор, ибо код компилируется и исполняется jvm (виртуальной машиной java)
- REPL отличный помощник во многом заменяющий IDE

Объявление переменных

- В REPL работает автодополнение кода (клавиша tab) и история команд (↑ ↓).
- Изменяемые переменные в Scala объявляются при помощи декларации var, а неизменяемые (константы) при помощи val

```
scala> val x = 8 * 5 + 2
x: Int = 42
scala>  0.5 * x
res3: Double = 21.0
scala> x = 0
<console>:8: error: reassignment to val
           x = 0
             ^
scala> var counter = 0
counter: Int = 0
scala> counter = 1
counter: Int = 1
```

Объявление переменных

- Как увидим далее, в Scala в большинстве своем используются неизменяемые переменные, поэтому декларация var будет использоваться довольно редко.
- Тип переменной выводится компилятором из контекста, однако, при желании, его можно объявить:

```
val greeting: String = null
val greeting: Any = "Привет"
```

- В Scala тип переменной или функции всегда пишется после имени через двоеточие. В большинстве случаев это удобнее.
- В Scala точки с запятой нужны только, если в одной строке находятся несколько определений:

```
val a = 1; var b = 3; val c = 2.4
```

- Присвоение одного значения нескольким переменным:

```
val xmax, ymax = 100
var greeting, message: String = null
```

Типы в Scala

- Scala имеет семь числовых типов: **Byte**, **Char**, **Short**, **Int**, **Long**, **Float**, и **Double** и логический **Boolean**.
- В отличие от Java все эти типы - классы:

```
1.toString // String = 1
1.to(10)   // Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

- В Scala нет примитивных типов и оболочечных классов. Компилятор сам производит приведение оболочечных классов к примитивным и наоборот.
- Класс String в Scala имеет в основе java.lang.String. Но имеет дополнительно еще много функций:

```
"Привет".intersect("мир") // String = ри
```

В этом примере java.lang.String неявно преобразуется к StringOps, у которого уже есть метод intersect

- Аналогично для Int, Double, Char, ... имеются классы RichInt, RichDouble, RichChar и так далее. В примере **1.to(10)** метод **to** имеется именно у RichInt

Типы в Scala

- Наконец, есть классы **BigInt** и **BigDecimal** для вычислений с любым количеством цифр.
- Они опираются на java.math.BigInteger и java.math.BigDecimal классов, но являются более удобным.
- Для преобразования одного типа к другому имеются соответствующие методы:

```
99.44.toInt // Int = 99
99.toChar // Char = 'c'
"99.44".toDouble // Double = 99.44
```

- Также как и в java метод toString позволяет приводить объекты к строке:

```
99.44.toString // String = 99.44
```

Арифметические операторы

- Операторы + - * / % работают обычным образом. Как обычно работают и битовые операторы & | ^ >> <<
- Но есть важное отличие: операторы являются методами. Например, a + b это просто синоним a.+(b)
- В Scala в названии метода можно использовать любые символы, например у класса BigInt есть метод /%, который возвращает пару (частное, остаток):

```
BigInt(155) /% 10 // (15,5)
```

- Вообще, если мы определили a.method(b), то это равносильно записи a method b. Так вместо 1.to(10) можно писать 1 to 10
- В отличие от Java или C++ в Scala нет операторов ++ или –. Вместо них следует использовать +=1 и -=1.

```
x+=1 // увеличение x на 1
```

- Для BigInt и BigDecimal в отличие от java можно использовать обычные операторы + - * / %.

Вызов функций

- Кроме методов в Scala есть функции.

```
import math._ // импорт всех функций пакета math
sqrt(2) // Double = 1.4142135623730951
pow(2,4) // Double = 16.0
min(3,Pi) // Double = 3.0
```

- В Scala нет статических переменных и методов класса, но вместе с классом может быть объявлен объект с именем класса, у которого могут быть методы:

```
BigInt.probablePrime(100, scala.util.Random) // простое число 100 бит
```

- Это похоже на вызов статического метода у класса BigInt, хотя на самом деле здесь BigInt - объект.
- scala.util.Random - это тоже объект, который не нужно создавать при помощи new
- Методы без аргументов, которые не меняют объект могут записываться без круглых скобок:

```
"аабббввв".distinct // String = абв
```

Метод apply

- В Scala повсюду используется синтаксис, который выглядит как вызов функции. Например, если s - строка, то получить ее i-й символ можно:

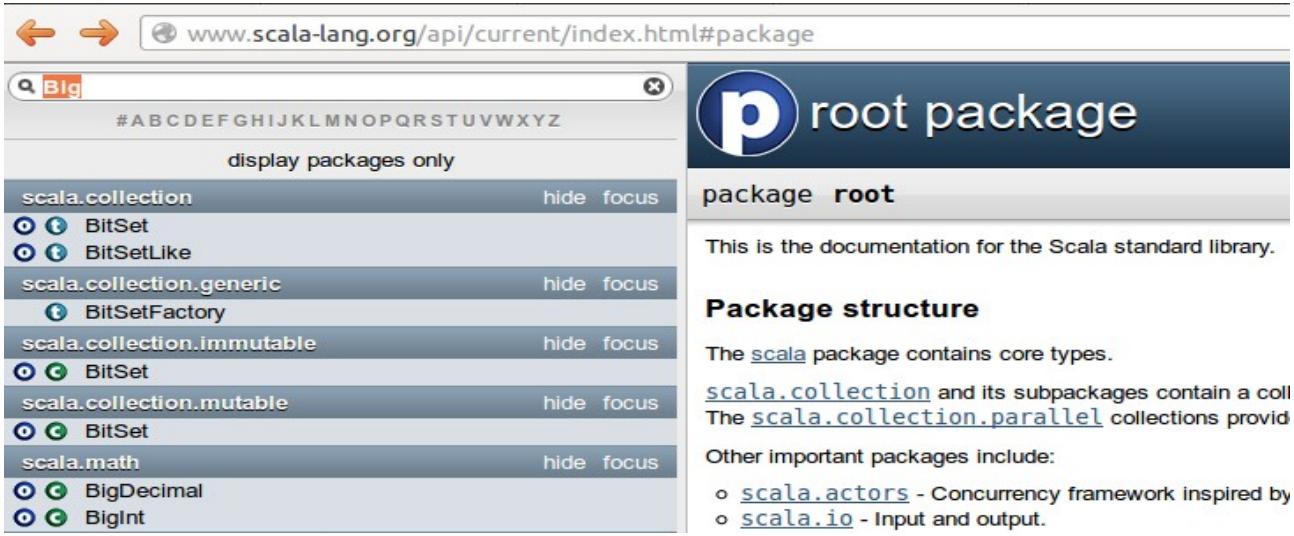
```
s[i] // в C++
s.charAt(i) // в Java
s(i) // в Scala
"Привет"(3) // Char = в
```

- Можно считать это перегрузкой оператора (), которая реализуется вызовом метода apply у StringOps. То есть `"Привет"(3)` эквивалентно `"Привет".apply(3)`
- Точно также, например, `BigInt("1234567890")` вызывает `BigInt.apply("1234567890")`.
- Благодаря такому сокращению можно обходиться без использования слова new:

```
BigInt("1234567890") * BigInt("112358111321")
Array(1, 4, 9, 16) // Array[Int] = Array(1, 4, 9, 16)
```

Scaladoc

- В Scala подобно java существует генератор документации, который называется Scaladoc (см. документацию к scala на сайте: www.scala-lang.org/api).
- Документацию к текущей версии можно скачать со страницы http://www.scala-lang.org/downloads#api установить и использовать локально.
- Работа со Scaladoc более удобна, чем с Javadoc. Например вывод при поиске класса сортируется по пакетам



Scaladoc (рекомендации)

- Для того чтобы очистить фильтр нажмите на x (крестик).
- Ищите **RichInt**, **RichDouble**, и так далее если хотите узнать про числовые типы.
- Ищите **StringOps**, если хотите узнать про тип String.
- Математические функции находятся в пакете scala.math
- Методы отмеченные как **implicit** задают автоматическое преобразование, по мере необходимости. Например, объект **BigInt** умеет преобразовать **int** и **long** к **BigInt** когда это потребуется
- Методы могут иметь функции в качестве параметров. Например, метод **count** у **StringOps** :

```
def count(p: (Char) => Boolean) : Int
// число символов для которых функция p вернет true
```

- Легко запутаться в большом числе методов. Такова Scala, здесь для каждого возможного случая существует свой метод. Научитесь находить их и использовать.

Условные выражения

- В Scala синтаксис конструкции **if … else** такой же как в C++ или Java. За исключением того, что она всегда возвращает значение:

```
val s = if (x > 0) 1 else -1 // присваиваем в s 1 если x>0 иначе -1
x > 0 ? 1 : -1 // Этой конструкции из Java или C++ в Scala нет
```

- В Scala, каждое выражение имеет тип. Например, выражение **if (x > 0) 1 else -1** имеет тип **Int**
- У смешанных выражений типа **if (x>0) "плюс" else -1** возвращаемое значение имеет общий тип **Any**
- Если одна часть опущена **if (x>0) 1**, то это эквивалентно **if (x>0) 1 else ()** и возвращаемое значение имеет тип **AnyVal**
- Пустое выражение **()** эквивалентно классу **Unit**, который заменяет декларацию **void** для методов ничего не возвращающих:

```
def main(args: Array[String]): Unit
```

Возвращаемое значение

- Возвращаемое значение типа Unit означает "нет значения". void - нет значения, Unit - возвращаем "нет значения".
- В Scala, блок кода {...} также возвращает значение. Это значение есть результат вычисления последнего выражения в блоке:

```
val distance = { val dx = x - x0; val dy = y - y0; sqrt(dx * dx + dy * dy) }
```

- Блок, который завершается оператором присваивания возвращает Unit:

```
{ r = r * n; n -= 1 }
```

- Есть ли ошибка следующем ниже в коде?

```
var x,y = 0
x=y=1
```

Стандартный ввод и вывод

- Для вывода на stdout используются функции print и println:

```
print("Ответ: ")
println(42)
println("Ответ: " + 42)
```

- Можно также использовать любимую функцию программистов на C:

```
printf("Привет, %s! Кажется Вам %d лет.\n", "Иван", 42)
```

- Для чтения из stdin существует много функций: readLine, readInt, readDouble, readByte, readShort, readLong, readFloat, readBoolean, readChar
- Пример:

```
val name = readLine("Ваше имя: ")
print("Ваш возраст: ")
val age = readInt
printf("Привет, %s! Через год Вам будет %d.\n", name, age + 1)
```

Циклы

- В Scala тот же самый while и do ... while как в Java или C++. Например,

```
while (n > 0) {
  r = r * n
  n -= 1
}
```

- Базовый цикл for в Scala имеет вид for (i <- expr). Переменная i принимает все значения выражения expr:

```
for (i <- 1 to n) r = r * i // i от 1 до n
for (i <- 1 until n) r = r * i // i от 1 до n-1
var sum = 0
for (ch <- "Hello") sum += ch // 500 - сумма ASCII кодов
```

- Цикл for в Scala перебирает список, поэтому тип переменной цикла определяется списком.
- При функциональном стиле программирования циклы for и while будут использоваться достаточно редко.

Расширенный цикл for

- В цикле допускается множество конструкций i <- expr разделяемых точкой с запятой:

```
for (i <- 1 to 3; j <- 1 to 3) print((10 * i + j) + " ") // 11 12 13 21 22 23 31 32 33
```

- В каждой конструкции можно выставлять условие:

```
for (i <- 1 to 3; j <- 1 to 3 if (i != j)) print((10 * i + j) + " ") // 12 13 21 23 31 32
```

- Допускается любое число определений внутри for:

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

- Когда тело цикла начинается с декларации yield тогда цикл генерирует коллекцию значений:

```
for (i <- 1 to 10) yield i % 3 // Vector(1, 2, 0, 1, 2, 0, 1, 2, 0, 1)
```

- Этот тип цикла называется "включением".

```
for (c <- "Hello"; i <- 0 to 1) yield (c + i).toChar // String = HleflmlmoP
for (i <- 0 to 1; c <- "Hello") yield (c + i).toChar
// Vector('H', 'e', 'l', 'l', 'o', 'l', 'f', 'm', 'm', 'p')
```

Спасибо за внимание !

www.altailand.ru