Linguagens de Programação

Scala

Gabriel Favalessa
Gustavo Andrade

Sumário

- Histórico
- Introdução/Conceitos Básicos
- Aspectos Teóricos
- Avaliação da Linguagem
- Referências

Histórico

Scala foi desenvolvida em 2001 por **Martin Odersky** e pelo grupo dele na École Polytechnique Fédérale de Lausanne (EPFL), Lausana na Suíça.

Em 1995 ele se juntou com Philip Wadler para escrever uma linguagem de programação funcional que compila Java bytecode.

Em 1999, depois de se unir a EPFL, este trabalho mudou um pouco. A meta ainda era combinar programação orientada a objetos e programação funcional, mas sem as restrições impostas pela linguagem Java.

Martin Odersky

O primeiro passo foi o Funnel, uma linguagem minimalista baseada em redes funcionais. No entanto, foi descoberto que a linguagem não era agradável para uso na prática. Minimalismo era ótimo para desenvolvedores, mas não para usuários.

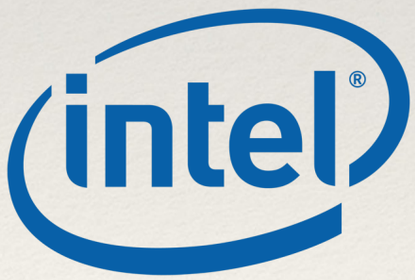
Assim surgiu Scala, que trouxe algumas das ideias do Funnel e colocou dentro de uma linguagem mais pragmática com foco especial no funcionamento com plataformas padrões. Scala não é uma extensão de Java, mas é completamente interoperável com ele.

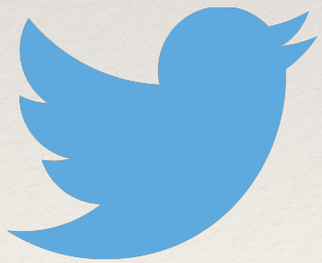
O design do Scala começou em 2001. Um primeiro lançamento ao público foi em 2003. Em 2006, uma segunda, versão remodelada foi lançada como Scala v 2.0. Desde então a linguagem tem ganhado popularidade. Atualmente a linguagem está na versão 2.11.

Introdução

Scala (Scalable language) é uma linguagem de programação de propósito geral. É uma linguagem multiparadigma, integrando recursos de linguagens **orientadas a objeto** e **funcional**.

Scala é uma linguagem de programação relativamente nova, mas mesmo assim, nos últimos anos, conquistou empresas gigantes como:





Conceitos Básicos

- **Scala é multiplataforma**: Roda na **JVM** (Java Virtual Machine) e **CLR** (plataforma .NET);
- **Puramente Orientada a Objeto**: Todo valor é um objeto;
- **Linguagem Funcional**;
- **É uma linguagem compilada**;
- **Amarração estática**;
- **Códigos reduzidos**;

Conceitos Básicos

- **Compatível com Java**: Executa código em Java puro. Imports/ Bibliotecas;
- **Possui frameworks para web**;
- **Possui Traits**;
- **Suporte a concorrência**;
- **Suporte a várias IDE's**: Eclipse, Netbeans, IntelliJ;

Ranking

Segundo o índice TIOBE:

Position	Programming Language	Ratings
21	OpenEdge ABL	0.796%
22	SAS	0.787%
23	Assembly language	0.754%
24	Dart	0.671%
25	Scala	0.628%
26	D	0.610%
27	Fortran	0.582%
28	Lua	0.546%
29	Logo	0.536%
30	Scala	0.531%
31	Prolog	0.518%
32	Lisp	0.506%
33	Transact-SQL	0.486%
34	Ada	0.433%

Como Rodar

Para compilar:

```
$ scalac OlaMundo.scala
```

Para selecionar o diretório do arquivo .class:

```
$ scalac -d classes OlaMundo.scala
```

Executando:

```
$ scala OlaMundo
```

Executando a partir de um diretório:

```
$ scala -cp classes OlaMundo
```

Como Rodar

Também podemos fazer o código direto no terminal:

```
MacBook-Pro-de-Gabriel:~ GabrielFavalessa$ scala
Welcome to Scala version 2.11.6 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_51).
Type in expressions to have them evaluated.
Type :help for more information.

scala> object HelloWorld {
     | def main(args: Array[String]){
     | println("Ola Mundo!")
     | }
     | }
defined object HelloWorld

scala> HelloWorld.main(null)
Ola Mundo!
```

Sintaxe

- **Case-sensitive:**

  exemplo ≠ Exemplo

- **Por convenção utiliza-se os nomes das classes com letras maiúsculas:**

  class ExemploClasse {…}

  Assim como também é aconselhável utilizar o nome do arquivo igual ao nome da classe.

- **E para métodos utiliza-se letras minúsculas no começo:**

  def exemploMetodo () {…}

- **Ponto e virgula (;) é opcional:**

  Pode ser usado caso queira tudo em uma única linha.

  …

  var a = 2; **println** (a)

  …

Sintaxe

- **Declaração de Classes:**

Classes em Scala são declaradas semelhante a Java

```
class PontoCartesiano (x: Float, y: Float){
        def abiscissas() = x
        def ordenadas() = y
}
```

*métodos também podem ser sem parâmetros.

Assim, para criar um novo objeto:

```
val objeto = new PontoCartesiano(5.0, 7.2)
```

Sintaxe

- **Modificadores de acesso:**

```
public class PontoCartesiano (x: Float, y: Float){
        protected def abiscissas() = x
        private def ordenadas() = y
}
```

public: visibilidade aberta.

protected: visibilidade dentro do package, subclasses.

private: visibilidade apenas dentro da própria classe.

Sintaxe

- **Funções e/ou métodos**

```
def adivinhaDia(ano:Int, mes:Int, dia:int)
{
    println(“voce nasceu na data: “ + dia + “/“ + mes + “/“ + ano)
}
```

```
public class PontoCartesiano (x: Float, y: Float){
        protected def abiscissas() = x
        private def ordenadas() = y

        def distancia (a: Float, b:Float) : Float {…}
}
```

Sintaxe

- **Palavras Reservadas:**

abstract	case	catch	class
def	do	else	extends
false	final	finally	for
forSome	if	implicit	import
lazy	match	new	null
object	override	package	private
protected	return	sealed	super
this	throw	trait	try
true	type	val	var
while	with	yield	
-	:	=	=>
<-	<:	<%	>:
#	@		

Amarrações

- Qualquer nome em Scala identifica um método, tipo, valor ou uma classe.
- Amarrações possuem diferentes tipos de precedências. Por exemplo declarações tem precedências sobre imports.
- Uma amarração em um escopo mais interno faz sombra em amarrações de precedência menor no mesmo escopo ou em escopos mais externos.
- Existe dois espaços de nomes distintos: Para **tipos** e **nomes**.

Tipos de Dados

Tipo de dado	Descrição
Byte	Valor de 8 bits com sinal. Range de -128 à 127
Short	Valor de 16 bits com sinal. Range de -32768 à 32767
Int	Valor de 32 bits com sinal. Range de -2147483648 à 2147483647
Long	Valor de 64 bits com sinal. De -9223372036854775808 à 9223372036854775807
Float	Ponto flutuante de 32 bits com precisão simples IEEE 754
Double	Ponto flutuante de 64 bits com precisão dupla IEEE 754
Char	Caracter Unicode de 16 bits sem sinal. Range de U+0000 à U+FFFF
String	Uma cadeia de Chars
Boolean	Assume os valores literais true ou false
Unit	Corresponde à sem valor
Null	Referência vazia ou nula
Nothing	O subtipo de todos os outros tipos; inclui Unit
Any	O supertipo de qualquer tipo; qualquer objeto é do tipo Any
AnyRef	O supertipo de qualquer referência

Variáveis e Constantes

Variáveis em Scala não necessitam de declaração por tipo:

- **Variáveis (var):**

  var a = 5

  var b = “String variável”

  Valores podem ser alterados posteriormente (mesmo tipo).

- **Constantes (val):**

  val a = 5

  val b = “String constante”

  Não podemos alterar os valores posteriormente.

Variáveis e Constantes

- **Alocação em pilha e/ou monte;**
- **Utiliza a estratégia utilizada por JVM para alocar e desalocar recursos (stack, heap, garbage collector);**
- **A JVM possui otimizações para decidir onde alocar cada variável;**

Operadores

- **Aritméticos**

Operador	Operação
+	Soma
-	Subtração
*	Multiplicação
/	Divisão

- **Lógicos**

Operador	Operação
&&	and
II	or
!	not

- **Atribuição**

- **Relacionais**

Operador	Operação
==	Verifica igualdade
!=	Verifica diferença
>	Verifica se é maior
<	Verifica se é menor
>=	Verifica se é maior igual
<=	Verifica se é menor igual

- **Bit a bit**

Operador	Operação
&	and
I	or
^	xor

Condicionais

- **if**, **else:** Funcionam como em Java.

```
if(x > 10){…}
else{…}
```

- **match:** Equivalente ao **switch** em Java.

```
var x = 1
x match {
    case 0 => …
    case 1 => …
    case 2 => …
  }
```

Iterações

- **while** e **do {...} while:** Funcionam exatamente como em Java.

- **for:** Existem 4 tipos principais (utilizando ranges, filtros, coleções e yield).
  - Utilizando ranges:

```
for(i <- 0 to 100){…}                          [0..100]
for(i <- 0 until 100){…}                       [0..99]
for(i <- Range(0, 100)){…}                     [0..99]
for(i <- 0 to 100; j <- 0 to 100){…}           [0..100]
```

Iterações

- Utilizando filtros:

```
for(i <- 0 to 10 if (i != 3); if(i != 6)){
    println (i)
}
```

```
0 1 2 4 5 7 8 9 10
```

- Utilizando coleções:

```
var lista = List(0,1,2,3,4,5)
for(i <- lista){
    println (i)
}
```

```
0 1 2 3 4 5
```

- Utilizando yield:

```
var valor = for(i <- 0 to 5 if(i < 3)) yield i
println (valor)
```

```
Vector(0, 1, 2)
```

Modularização

- Modularização por pacotes e parâmetros
  - Pacotes aninhados: é possível definir pacotes dentro de um pacote.

```
package padaria {
 package oak {
    class Clientes{…}
    class Vendas{…}
}
  class Produtos{…}
}
...
import padaria._
import padaria.oak.{Clientes, Vendas}
```

Modularização

Valores default:

```
class Hash_tag[K,V](capacidadeInicial : Int = 16, limite : Float = 0.75) {...}

val hash1 = new Hash_tag[String,Int]
val hash2 = new Hash_tag[String,Int](20)
val hash3 = new Hash_tag[String,Int](20,0.8)
val hash4 = new Hash_tag[String,Int](limite = 0.8)
val hash5 = new Hash_tag[String,Int](limite = 0.8, capacidadeInicial = 20)

def cartesiano (x : Float = 1.0, y : Float = 1.0)
{
  x + y
}
```

Polimorfismo

- **Coerção**

Scala suporta polimorfismo de coerção.

Tipos menores são convertidos para tipos maiores:

Int -> **Float**

Feito implicitamente.

Não é feito o contrário. Ou seja, operações de estreitamento **não** são realizadas implicitamente.

Float -> **Int** (Não é valido)

Polimorfismo

- **Sobrecarga**

Scala possui sobrecarga de métodos e operadores, sendo possível usar o mesmo nome em diferentes amarrações.

Em herança, declaramos a sobrecarga de métodos com a palavra reservada **override**.

Permite a definição de novos operadores.

Assim como a **precedência**, a **associatividade** dos operadores podem ser alteradas, por exemplo, da esquerda para direita ou da direita para esquerda.

Polimorfismo

- **Sobrecarga**

  Exemplo:

```
class deAula
{
  def chamada(qtd:Int)
  {...}

  def chamada()
  {...}
}
```

Polimorfismo

- **Inclusão**

Sistema de herança com **extends**.

A herança pode ser simples ou múltipla. Na herança múltipla utilizamos a palavra reservada **with**.

Colisão de nomes pode ser tratada com as palavras chaves **abstract override**, assim a classes extensoras são forçadas a implementarem seus próprios métodos.

Polimorfismo

- **Inclusão**

  Exemplo:

```
public class Pneu {...}

public class Motor {...}

public class Freios {...}

public class Carro extends Pneu with Motor with Freios{...}
```

Polimorfismo

- **Paramétrico**

```
var vetor : Array[String] = Array();

var mapa : Map[Int, String] = Map();
```

Polimorfismo paramétrico com o uso de classes genéricas.

Verificação de Tipos

- Scala é fortemente tipada.
- Possui verificação de tipos estática e dinâmica.
- Maior parte das verificações é em tempo de compilação.
- Para poder dar suporte à Orientação a Objeto faz algumas verificações em tempo de execução.
- O compilador é capaz de realizar inferência de tipos para variáveis e funções. Já para os parâmetros é necessário declarar os tipos explicitamente.

Verificação de Tipos

```
object exemplo {

  def main(args: Array[String])
  {
    println(retorno(1))
    println(retorno(15))
  }

  def retorno (i : Int) =
    if(i < 10) "menor que dez"

    else 20
}
```

```
menor que dez
20
```

Parâmetros - varargs

- Suporte para parâmetros com tamanho variável.
- A declaração deve estar na última posição do método ou função.
- Utilizamos o operador *.

```
object ExemploVar
{
  def main(args: Array[String])
  {
    imprime(10, "um", "dois", "tres", "etc")
  }

  def imprime(x: Int, strings: String*)
  {
    println(x)
    strings.map(println)
  }
}
```

Exceções

- Muito semelhante a Java. Com a criação de blocos **try** {…} **catch** {…}.

```
try {

  throwsException();

} catch {

  case e: IllegalArgumentException => println("illegal arg. exception");
  case e: IllegalStateException    => println("illegal state exception");
  case e: IOException              => println("IO exception");

} finally {

  println("this code is always executed");

}
```

Utiliza **pattern matching** no **catch** para saber qual exceção foi capturada.

Concorrência

- Scala suporta os mesmos métodos de concorrência utilizados por Java.
- Possui um pacote de concorrência nativo, o **scala.concurrent**.
- Implementa alguns métodos que Haskell utiliza.
- A resposta da Scala para facilitar a programação concorrente são os **Actors** (ou atores), que são análogos aos **Threads** do Java, mas muitos recursos para facilitar é uma grande diferença, **Actors** em Scala possuem um protocolo definido para se comunicarem, eles enviam mensagens uns para os outros em vez de acessar as mesmas variáveis que outras **Threads**.

Avaliação da Linguagem

Critério	C	Java	Scala
Aplicabilidade	Sim	Parcial	**Parcial**
Confiabilidade	Não	Sim	**Sim**
Aprendizado	Não	Não	**Parcial**
Eficiência	Sim	Parcial	**Parcial**
Portabilidade	Não	Sim	**Sim**
Método de projeto	Estruturado	OO	**OO** e **Funcional**
Evolutibilidade	Não	Sim	**Sim**
Reusabilidade	Sim	Sim	**Sim**
Integração	Sim	Parcial	**Sim**
Custo	Depende da aplicação	Depende da ferramenta	**Depende**

Avaliação da Linguagem

Critério	C	Java	Scala
Aplicabilidade	Sim	Parcial	**Parcial**

Por ser uma linguagem de uso geral, podemos resolver vários problemas com Scala.

Avaliação da Linguagem

Critério	C	Java	Scala
Confiabilidade	Não	Sim	**Sim**

Uma linguagem bastante confiável com tratamento de exceções muito semelhante a Java, além de ser fortemente tipada.

Avaliação da Linguagem

Critério	C	Java	Scala
Aprendizado	Não	**Parcial**	**Parcial**

Por ser uma linguagem que opta por maior redigibilidade, torna-se um pouco confusa a primeira vista. (Aqueles que já conhecem Java não terão grandes problemas)

Avaliação da Linguagem

Critério	C	Java	Scala
Eficiência	Sim	Parcial	**Parcial**

Em comparação à Java apresenta os mesmos benefícios.

Avaliação da Linguagem

Critério	C	Java	Scala
Portabilidade	Não	Sim	**Sim**

Uma linguagem bastante portável, além de rodar na JVM também roda na plataforma .NET (CLR).

Avaliação da Linguagem

Critério	C	Java	Scala
Método de projeto	Estruturado	OO	**OO** e **Funcional**

Avaliação da Linguagem

Critério	C	Java	Scala
Evolutibilidade	Não	Sim	**Sim**

Como o próprio nome ja diz, uma linguagem escalável.

Avaliação da Linguagem

Critério	C	Java	Scala
Reusabilidade	Sim	Sim	**Sim**

Scala da suporte a classes e possui mecanismos de pacotes. O polimorfismo universal também auxilia na reusabilidade do código a medida que oferece mecanismo de herança e tipos genéricos.

Avaliação da Linguagem

Critério	C	Java	Scala
Integração	Sim	Parcial	**Parcial**

Completamente interoperável com Java.

Avaliação da Linguagem

Critério	C	Java	Scala
Custo	Depende da aplicação	Depende da ferramenta	**Depende**

Referências

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

2. https://wiki.scala-lang.org/pages/viewpage.action?pageId=294934

3. http://ccsl.ime.usp.br/pt-br/uma-introducao-a-linguagem-scala

4. http://www.scala-lang.org/api/2.10.1/index.html#package