Linguagem de Programação: Scala

Fabrícia de Jesus Santos[1], **Valéria dos Santos Barbosa**[2]

[1]Departamento de Sistemas de Informação
Universidade Federal de Sergipe (UFS) – Itabaiana, SE – Brasil

fabriciacooper@gmail.com, valeria_sb@outlook.com.br

1. Introdução

O Scala é uma linguagem de programação semelhante a Java que unifica a programação funcional e orientada a objetos. É uma linguagem puramente orientada a objetos, no sentido de que todo valor é um objeto. Tipos e comportamento de objetos são descritos por classes.

Scala é uma linguagem funcional no sentido de que toda função é um valor. O aninhamento de definições de função e funções de ordem superior é naturalmente suportado. O Scala também suporta uma noção geral de correspondência de padrões que pode modelar os tipos algébricos usados em muitas linguagens funcionais.

O Scala foi projetado para interoperar perfeitamente com o Java. As classes do Scala podem chamar métodos Java, criar objetos Java, herdar de classes Java e implementar interfaces Java. Nada disso requer definições de interface ou código de cola.

O Scala foi desenvolvido a partir de 2001 no laboratório de métodos de programação da EPFL. A versão 1.0 foi lançada em novembro de 2003. . Ela atua como uma referência para a definição da linguagem e alguns módulos da biblioteca principal.

2. Lexemas de Scal

No Scala, como em Java, C e em muitas outras linguagens, os identificadores podem conter uma mistura de caracteres maiúsculos e minúsculos. Esses identificadores são tratados de maneira sensível a maiúsculas e minúsculas. Por exemplo, ”index”, ”Index”e ”INDEX”seriam tratados como três identificadores ~~separados~~. Você pode definir todos os três no mesmo escopo. Isso vale para Scala, Java e a maioria, senão todos, os descendentes da linguagem C.

Vale ressaltar que, programas Scala são escritos usando o conjunto de caracteres Unicode Basic Multilingual Plane (BMP); Caracteres suplementares Unicode não são atualmente suportados.

2.1. Comentários

Um comentário de linha única é uma sequência de caracteres que começa // e se estende até o final da linha.

Um comentário de várias linhas é uma sequência de caracteres entre /* e */. Os comentários de várias linhas podem ser aninhados, mas precisam ser aninhados corretamente. Portanto, um comentário como /* /* */será rejeitado como tendo um comentário não terminado.

2.2. Palavras Reservadas

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

2.3. Operadores e Delimitadores

2.3.1. Aritméticos

Operador	Operação
+	Soma
-	Subtração
	Multiplicação
/	Divisão

2.3.2. Relacionais

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

2.3.3. Lógicos

Operador	Operação
&&	and
||	or
!	not

2.3.4. Bit a Bit

Operador	Operação
&	and
|	or
^	xor

2.4. Literais String

Uma string literal é uma sequência de caracteres entre aspas duplas. Os caracteres podem ser qualquer caractere Unicode, exceto o delimitador de aspas duplas ou ŭ000A(LF) ou ŭ000D(CR); ou qualquer caractere Unicode representado por um escape Unicode ou por uma sequência de escape .

O valor de um literal de string é uma instância de classe String. Exemplo:

```
"Hello, world!\n"
"\"Hello,\" replied the world."
```

Um literal de caractere é um único caractere entre aspas. O caractere pode ser qualquer caractere Unicode, exceto o delimitador de aspas simples ou ŭ000A(LF) ou ŭ000D(CR); ou qualquer caractere Unicode representado por um escape Unicode ou por uma sequência de escape.

```
'a'    '\u0041'    '\n'    '\t'
```

2.5. Literais Numérica

```
integerLiteral  ::=  (decimalNumeral | hexNumeral)
['L' | 'l']
decimalNumeral  ::=  '0' | nonZeroDigit {digit}
hexNumeral      ::=  '0' ('x' | 'X') hexDigit {hexDigit}
digit           ::=  '0' | nonZeroDigit
nonZeroDigit    ::=  '1' |...| '9'
```

Os literais inteiros são geralmente do tipo Int ou do tipo Long quando seguidos por um L ou l sufixo. Os valores do tipo Int são todos números inteiros entre -2^{31} e $2^{31} - 1$, inclusive.

Exemplo:

```
0.0        1e30f      3.14159f      1.0e-100      .1
```

Literais de ponto flutuante são do tipo Float quando seguidos por um sufixo de tipo de ponto flutuante F ou f, e são do tipo de Double outra forma. O tipo Float consiste em todos os valores de ponto flutuante binário de precisão única de 32 bits IEEE 754, enquanto o tipo Double consiste em todos os valores de ponto flutuante binário de precisão dupla de 64 bits de 64 bits do IEEE 754.

Exemplo:

```
0.0        1e30f      3.14159f      1.0e-100      .1
```

2.6. Literal Booleano

Os literais booleanos true e false são membros do tipo Boolean. Exemplo:

```
booleanLiteral  ::=  'true' | 'false'
```

2.7. Identificador

```
op         ::=  opchar {opchar}
varid      ::=  lower idrest
boundvarid ::=  varid
| ' `' varid ' `'
plainid    ::=  upper idrest
|   varid
|   op
id         ::=  plainid
|   ' `' { charNoBackQuoteOrNewline | UnicodeEscape | charEscapeSeq } ' `'
idrest     ::=  {letter | digit} ['_' op]
```

Existem três maneiras de formar um identificador. Primeiro, um identificador pode começar com uma letra que pode ser seguida por uma sequência arbitrária de letras e dígitos. Isso pode ser seguido por _ caracteres de sublinhado e outra sequência composta de letras e dígitos ou de caracteres do operador. Segundo, um identificador pode começar com um caractere de operador seguido por uma sequência arbitrária de caracteres do operador. Os dois formulários anteriores são chamados de identificadores simples. Finalmente, um identificador também pode ser formado por uma cadeia arbitrária entre aspas (sistemas host podem impor algumas restrições sobre quais strings são legais para identificadores). O identificador é composto de todos os caracteres, excluindo os backquotes.

Como de costume, uma regra de correspondência mais longa se aplica. Por exemplo, a string **big_bob++= 'def'**, decompõe-se em três identificadores big_bob, ++= e def.

As regras para correspondência de padrões distinguem ainda mais entre identificadores de variáveis, que começam com uma letra minúscula, e identificadores constantes. Para esse propósito, o sublinhado _ é considerado em letras minúsculas e o '$ é considerado em letra minúscula, e o caractere ' $' é considerado em letra maiúscula.

2.8. Variáveis

[escopo] [tipo da acesso] [nome da variável]: [tipo da variável] = [Valor padrão]

Exemplo:

```
private val name: String = "Diego"
```

Essa é a forma completa de declaração, porém não precisamos declarar sempre a forma completa, podemos omitir alguns parâmetros.

- Escopo: acesso da sua variável, podendo ser Public, Private, Protected;
- Tipo de Acesso: val ou var;
- Tipo da Variável: String, Int, Double ou Boolean;