Scala: Primeiros passos com o paradigma funcional

Diego Saraiva
Ronualdo Maciel

24 de Maio de 2011

Diego Saraiva

- Mestre em Sistemas Distribuidos pela UFRN
- Desenvolvedor C/C++
- jdiego@gmail.com

Ronualdo Maciel

- Tecnologo em Desenvolvimento de Software pelo CEFET-RN
- Desenvolvedor Java
- raxmac@gmail.com

Sumário

Sumário

Sumário

Sumário

Sumário

Sumário

Sumário

Introdução

Exigências do mercado atual

- Utilização efetiva de máquinas multi-core por programas concorrentes.
- Criação de aplicações distribuídas voltadas para a Web ou para a Internet.

Introdução

Linguagens

paradigmas-imagem.png

Introdução

Programação orientada a objetos

- Estilo imperativo
- Tenta simular o mundo real
- A computação é realizada em termos de estados e expressões que podem mudar o estado do programa.

progImp.png

Introdução

A Programação Funcional

- Estilo de programação baseado no uso de funções
- Funções são entidades de 1ª classe
- Toda a programação é baseada na avaliação de expressões para gerar valores.
- Cada valor tem um tipo associado.
- Funções podem ser nomeadas ou anônimas (lambda)

Introdução

Scala

- O nome Scala significa “linguagem escalável”
- Projetada para integrar linguagem orientada a objetos e programação funcional
- Executa na JVM

Scala

Histórico

- O design começou em 2001 na École Polytechnique Fédérale de Lausanne por Matin Odersky
- Primeiro release na plataforma Java: no fim de 2003 e início de 2004
- Versão para plataforma .NET liberada em Junho de 2004

Scala

Por quê utilizar Scala?

- Linguagem híbrida: simplicidade de funcional + poder de objetos
- Fortemente tipada
- Linguagem Concisa
- Multiplataforma: JVM e .NET

Introdução a programação funcional

Entidades de primeira-classe

- Entidades que podem ser passadas como parâmetro
- Podem ser retornadas como resultado
- Podem ser armazenadas em estruturas de dados

A Programação Funcional

Modelo computacional

- Função de x em y : Mapeamento de valores de entrada em valores de saída
- Ausência de estado e comandos (atribuição + controle)

modelo_funcional.png

Paradigma Funcional

Mudando o foco

- sem laços?
- sem efeitos colaterais?
- sem mudar o valor de variáveis?

Aplicações

- Verificação de programas: checagem de corretude
  - Princípio da invariância.
- Otimização de programas para computação paralela

Paradigma Funcional

Linguagem Funcional

- O corpo de uma função é uma expressão
- A aplicação da função a um argumento retorna um valor (expressão)
- Um programa é uma expressão

O contexto

- Mapeamento de identificadores (nomes de função) em definições de função
- Mapeamento de identificadores em valores

Sintaxe

Variáveis

```
1  val s = "Hello World"
2
3  var i = 1
4
5  private var j = 3
```

Sintaxe

Definição de métodos

```
1  def adicionar(x: Int, y: Int): Int = {
2    x + y
3  }
4
5  def adicionar(x: Int, y: Int) = x + y
6
7  def fazerAlgumaCoisa(text: String) {
8    //fazendo alguma coisa
9  }
```

Sintaxe

Chamadas a métodos

```
1  meuObjeto.meuMetodo(1)
2  meuObjeto meuMetodo(1)
3  meuObjeto meuMetodo 1
4
5  meuObjeto.outroMetodo(1, 2)
6  meuObjeto outroMetodo(1, 2)
7
8  meuObjeto.metodoSemParametros()
9  meuObjeto.metodoSemParametros
10 meuObjeto metodoSemParametros
```

Sintaxe

Classes - Scala

```scala
class Pessoa(val nome: String)
```

Classes - Java

```java
public class Pessoa{
  private final String nome;

  public Pessoa(String nome){ this.nome = nome;}

  public String getNome() {return nome;}
}
```

Sintaxe

Classes - Scala

```scala
class Pessoa(var nome: String)
```

Classes - Java

```java
public class Pessoa{
  private String nome;

  public Pessoa(String nome){this.nome = nome}

  public void getNome(){return nome;}

  public void setNome(String nome){this.nome = nome;}
}
```

Sintaxe

If

```
1 if(condicao) {
2   //primeiro caso
3 } else if(outraCondicao) {
4   //segundo caso
5 } else {
6   //entao
7 }
```

Sintaxe

For

```
1  for ( i <- 1 to 5){
2    ...
3  }
4
5  for ( i <- 1 until 5) {
6    ...
7  }
8
9  for (s <- args){
10   println(s)
11 }
```

Sintaxe

While

```
1  while(condicao){
2      ....
3  }
4
5  do {
6      ...
7  } while(condicao)
```

Sintaxe

Try

```
1 try{
2    ...
3 } catch {
4    case e: IOException
5 } finally {
6
7 }
```

Principais conceitos

Funções como entidades de primeira ordem

- As funções podem ser criadas em qualquer lugar do programa
- As funções podem armazenadas em estruturas de dados
- As funções podem ser passadas como argumento para outras funções
- As funções podem retornar outras funções como resultados

```
1 (x: Int, y: Int) => x + y
```

Principais conceitos

Funções de alta-ordem

- Recebem uma ou mais funções como entrada
- Retornam uma função como saída
  - Exemplo: map

Principais conceitos

Closures

- Um closure é uma função que referencia variáveis livres no contexto léxico.
- Função anônima definida on-the-fly dentro de uma função ou expressão
- Uma closure pode ser definida pelo compilador como resultado de outra função
- Closure é diferente de uma função anônima

```
1 val incremento = 1
2 val adicionarIncremento = (x: Int) => x + incremento
```

Principais conceitos

Currying

Currying é o processo de transformar uma função que recebe múltiplos argumentos em uma função que recebe um único argumento e retorna outra função.

```
1 def soma(x: Int)(y: Int) = x + y
2 val umMais = soma(1)(_)
3 umMais(3)
```

Principais conceitos

Pattern Matching

Pattern matching é uma forma de associar nomes a coisas e possivelmente quebrar as expressões em sub-expressões ao mesmo tempo em que associa cada sub-expressão a cosias.

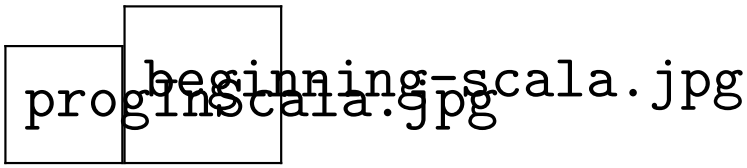
- Reconhecer valores
- Associar variáveis para reconhecer valores
- "Quebrar"valores em partes

```
1 val res = myObject match {
2   case i: Int if i == 1 => "Found an int"
3   case s: String => "Found a String"
4   case other => "Unknown " + other
5 }
```

Exemplos

Some code...

Livros



Sites

- **Site oficial -** http://www.scala-lang.org/
- **Blog Scala-BR -** http://scala-br.org/
- **Grupo Scala-BR -** http://groups.google.com/group/scala-br

Fim

Perguntas ???