Programación funcional con Scala

Fabricio Sebastian La Torre Luque

3 de diciembre de 2021

Resumen

Scala es un lenguaje JVM que incorpora paradigmas de OOP y Functional Programming en un lenguaje conciso, de alto nivel y expresivo.

1. Introducción

2. ¿Qué es la programación funcional?

La programación funcional es un paradigma de programación que usa funciones como el bloque de construcción central del programa
En la programación funcional, el objetivo es usar valores inmutables y funciones puras.

2.1. Inmutabilidad

La inmutabilidad significa programar usando constantes, queriendo decir que el valor o el estado de las variables no puede ser cambiado. Lo mismo pasa con los objetos, podemos crear un nuevo objeto, pero no podemos cambiar el estado de el objeto existente.

2.2. Funciones Puras

Para recordar:
Una función pura, siempre retorna el mismo valor para las mismas entradas, es decir que si la entrada siempre es la misma, entonces la salida también siempre tendrá que ser la misma. Además, una función pura no tiene efectos secundarios, no hace más que retornar un resultado, significando que esta función no interactúa con el estado del programa.

3. ¿Cómo es Scala funcional?

Scala es fundamentalmente funcional en el sentido que todas las funciones son un valor, dentro del lenguaje, las funciones son tratadas como VIP.
Hay bastantes construcciones funcionales que podemos crear en un ámbito de OOP, como funciones que toman retornan otras funciones y métodos que están declarados dentro de otro método.
En OOP estas construcciones son raras, sin embargo en FP, estas ocurren naturalmente y son de alguna manera necesarias Veremos a continuación diversos ejemplos de cómo es la programación en Scala

3.1. Funciones como Ciudadanos de primera Clase

Cuando tratamos a una función como un valor, lo consideramos como una función de primera clase.
En general, una función de primera clase puede ser:
-Asignada a una variable
-Pasada como un argumento a otras funciones
-Retornada como un valor

3.2. Funciones de orden mas alto

El concepto de una Higher Order Function (HOF) esta asociada con el de una función de primera clase
Una HOF tiene al menos una de estas siguientes propiedades:
-Toma una o mas funciones como parámetro
-Retorna una función como resultado

Basicamente, con una HOF podemos trabajar con las funciones como trabajaríamos con otros tipos de valores.

```
def performAddition(x: Int, y: Int): Int = x + y

def performSubtraction(x: Int, y: Int): Int = x - y

def performMultiplication(x: Int, y: Int): Int = x * y

def performArithmeticOperation(num1: Int, num2: Int, operation: String): Int = {
  operation match {
    case "addition" => performAddition(num1, num2)
    case "subtraction" => performSubtraction(num1, num2)
    case "multiplication" => performMultiplication(num1, num2)
    case _ => -1
  }
}

val additionResult = performArithmeticOperation(2, 4, "addition")
assert(additionResult == 6)

val subtractionResult = performArithmeticOperation(10, 6, "subtraction")
assert(subtractionResult == 4)

val multiplicationResult = performArithmeticOperation(8, 5, "multiplication")
assert(multiplicationResult == 40)
```

En este programa podemos ver como funcionan las HOF y las funciones en general en Scala, usando funciones dentro de otras y llamando a cada una para que desarrolle su tarea determinada
También existen las llamadas Funciones Anónimas o Expresiones Lambda que no tiene nombre pero tiene cuerpo, parámetros, y un tipo de retorno opcional.
Map, Filter y Fold son funciones que toman como parámetro a una expresión lambda

3.3. Funciones parcialmente aplicadas

En la programación funciona, una llamada a una función que tiene parámetros también puede ser llamado .ªplicar la función.ª los parámetros, cuando una función es llamada con todos los parámetros requeridos. Sin embargo, cuando un subconjunto de todos los parámetros esta completado es una función parcialmente aplicada.
Una vez que los parámetros iniciales requeridos son llenados, una nueva función es retornada con el resto de argumentos que tiene que pasar.
Tomemos un ejemplo:

```
def calculateSellingPrice(discount: Double, productPrice: Double): Double = {
  (1 - discount/100) * productPrice
}

val discountApplied = calculateSellingPrice(25, _)
val sellingPrice = discountApplied(1000)
assert(sellingPrice == 750)
```

Aquí, discountApplied es una función que necesita un parámetro mas para funcionar, asi que ahora es una función que solo toma un parámetro que es el segundo (ProductPrice)

3.4. Conclusión de funcionalidad

Scala nos provee con suficiente OOP como para sentir que estamos en terreno familiar, sin embargo es una forma excelente de conocer la programación funcional a un paso continuo .

4. Ejemplo

En este programa el objetivo es encontrar la suma de los cuadrados de los números impares pasados en una lista , este seria un ejemplo en JAVA:

```
  import java.util.*;

public class SumOfSquaresOfOdd
{
    public static void main(String... args)
    {
        List<Integer> intList = new ArrayList<Integer>();

        for(int i = 1; i <= 5; i++) {
                intList.add(i);
        }

        System.out.println("intList - Initial : " + intList);

        List<Integer> oddNoList = new ArrayList<Integer>();

        for(int i = 0; i < intList.size(); i++) {
                int intlistI = intList.get(i);
                if(intlistI % 2==1) {
                        oddNoList.add(intList.get(i));
                }
        }

        System.out.println("oddNoList : " + oddNoList);

        List<Integer> squareList  = new  ArrayList<Integer>();

        for(int i=0; i < oddNoList.size(); i++) {
                squareList.add(oddNoList.get(i) * oddNoList.get(i));
        }

        System.out.println("squareList : " + squareList);

        int sum = 0;
        for(Integer i : squareList) {
                sum += i;
        }

        System.out.println("Sum : " + sum);
    }
}
```

el output de este programa es:

```
intList - Initial : [1, 2, 3, 4, 5]
oddNoList : [1, 3, 5]
squareList : [1, 9, 25]
Sum : 35
```

Sin embargo, si es que nosotros escribimos el mismo programa en Scala:

```
import scala.collection.immutable.List

object SumOfSquaresOfOdd
{
        def main(args:Array[String]):Unit =
        {
                var intList = List(1,2,3,4,5)
                def sum = intList.filter(x => x % 2 ==1).map(x => x * x).reduce((x,y) => x+y)
                println(sum)
        }
}
```

El programa se ha hecho mucho mas conciso en Scala ya que tenemos métodos como Filter y Map. Utilizamos Filter para recoger los números impares, Map para iterar entre todos los impares, y reduce para sumarlos todos. El output de este programa es solo:

```
35
```

5. Ventajas de programación funcional

La programación funcional tiene código limpio y conciso, fuerza a que los problemas grandes sean divididos en instancias mas pequeñas, esto hace que el código se vuelve mas modular
Reduce la dependencia entre objetos, ya que las funciones puras son independientes, evita que las variables globales tengan efectos secundarios.
El testing se hace mucho mas fácil ya que se puede probar una funcion a la vez y por separado, debido a que son funciones puras.

6. Conclusiones

Scala, y la programación funcional en general, tiene muchos beeficios y en una opinion personal una mejor implementación y paradigma que la OOP, Scala es una herramienta excelente para poder adentrarse al mundo de la programación funcional y con una sintaxis limpia y poder, es un lenguaje de programación con mucho potencial, ademas de ser práctico logra resolver problemas de maneras creativas, siendo un muy buen martillo en el cinturón de un constructor.

Referencias

2021. Functional Programming. [online] Available at: ¡https://www.baeldung.com/scala/functional-programming¿

Examples Java Code Geeks. 2021. Functional Programming in Scala. [online] Available at: ¡https://examples.javacodegeeks.com/jvm-languages/scala/functional-programming-scala/¿

Chiusano, P. and Bjarnason, R., 2015. Functional programming in scala. Shelter Island, NY: Manning.