Scala

Sunday, 26 September 2021 12:48 pm

/opt/cloudera/parcels/CDH-6.3.2-1.cdh6.3.2.p0.1605554/lib
/home/osboxes/.sbt/boot/scala-2.12.14/org.scala-sbt/sbt/1.5.5
Scala Programming Tutorial | Learn Scala programming | Scala language

1 - Introduction to Scala

- Scala - **Sca**lable **La**nguage
- Modern multi-paradigm programming language designed to express coming programming patterns in a concise, elegant and type-safe way.
- Scala is written by Martin Odersky (from Suzerland) at EPFL
- statically typed (error at compile time), runs on JVM, full inter-op with Java (java can run in scala and scala can run in java)
- Object oriented, functional, dynamic features, blends object-oriented and functional programming in a statistically typed language.
- **Scala is practical**
  Can be used as drop-in replaced for java. Mixed scala/java projects, existing java libraries can be used, existing java tools can be used (Ant, Mavent, Junit, etc)
  Decent IDE support(NetBeans, IntelliJ, Eclipse)
- Many patterns are available and full interoperable support.

2 - Introduction to SBT (Scala Build Tool)

- SBT (Scala build tool, formally simple build tool) is an open source build tool for scala and java projects, similar to Java's Maven and Ant
- SBT is a modern build tool. Its written in Scala and provides many Scala conveniences, it's a general purpose build tool.

SBT is the de facto tool in the scala. When you install SBT, Scala gets installed automatically.

- Features of SBT
  Has native for compiling Scala code and uses apache Ivy for dependency management (maven format). Only-update-on-request model. Full Scala language for creating tasks and support for mixed java/scala projects. Launch REPL (Read-Eval-Print Loop) in project context. REPL is a

simple interactive computer programming env, it takes simple input and return the result.

3. Installing SBT on Linux

(yum didn't help to install .rpm file so I have used SDK approach)

(i) Install SDKMAN : https://sdkman.io/install

(ii) install SBT : https://www.scala-sbt.org/1.x/docs/Installing-sbt-on-Linux.html

Install IntelliJ : https://www.jetbrains.com/help/idea/installation-guide.html
Scala plugin installation and first project : https://docs.scala-lang.org/getting-started/intellij-track/getting-started-with-scala-in-intellij.html

Jetbrains Install Plugins
https://www.jetbrains.com/help/idea/managing-plugins.html

Running first spark application
https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/spark_first.html

Limit log output in scala :
https://stackoverflow.com/questions/49515280/limit-output-with-scala-on-intellij

log only errors
edit spark config
path : /etc/spark/conf
gedit log4j.properties
root.logger=ERROR,console

org/apache/spark/log4j-defaults.properties

Create an sbt project
create a folder "sbt_project"
just type "sbt" and enter. This will create SBT project with default files.

- sbt

4. Data Types and Variables

```
Boolean   true or false
Byte    8 bit signed value
Short   16 bit signed value
Char    16 bit unsigned Unicode character
Int     32 bit signed value
Long    64 bit signed value
Float   32 bit IEEE 754 single-precision float
Double  64 bit IEEE 754 double-precision float
String  A sequence of characters


Unit    Corresponds to no value
Null    null or empty reference
Nothing subtype of every other type; includes no
Any The supertype of any type; any object is of
AnyRef  The supertype of any reference type
```

New folder : variables

- Sbt console

Clear console : **Ctrl + L**

Format code : Control + Shift + F

	Mutable variable	scala> var a: Int = 12 a: Int = 12 scala> a + 30 res0: Int = 42 scala> a+40 res1: Int = 52 scala> res0 res2: Int = 42
	Immutable variable	scala> val b : Int = 50 b: Int = 50

		scala> b = 20 <console>:12: error: reassignment to val b = 20
	Variable must be initialized	var c:Int; <console>:11: error: only traits and abstract classes can have declared but undefined members (Note that variables need to be initialized to be defined) var c:Int;
	Declaration without datatype.	Boolean = true scala> var intDT = 12 intDT: Int = 12 scala> var flotDT = 12.3f flotDT: Float = 12.3 scala> var doubleDT = 123 doubleDT: Int = 123
	Define multiple variable and return the result using curly braces.	scala> val x = {val a: Int= 200; val b:Int =300; a+b} x: Int = 500 --- scala> val x = { | val a : Int = 500 | val b : Int = 600 | a+b | } x: Int = 1100
	Lazy loading or on demand loading. (lazy initializing -> value of the variable will be used/assigned only if the variable is used)	scala> lazy val x =500 x: Int = <lazy> scala> x * 2 res6: Int = 1000

1. Scala IDE (http://scala-ide.org/docs/videos.html)
   - Ensure Java is installed
   - extracted the tar and created a .desktop file to launch and copied the .desktop file to "~/.local/share/applications/"

Sample project:
- launch eclipse -> new project -> Scala -> just give the project name and Finish

Object : we can't call new keywork on object as its already initiated. Its like a singleton type. And like a class but with single instance.
Scala class will have one main method.

Run the application : right click on the file -> run as -> scala application

Scala Interpreter : Right click on the project > Scala > Create Scala Interpreter	scala> object HelloWorld {
	def main (args: Array[String]){
	println("Hello, World!")
	}
	} | | | |
	defined object HelloWorld
	
	scala> HelloWorld.main(Array())
	Hello, World!

6 - Scala String Interpolation

Replace or define variable with given string.

```
//6. string interpolation example
  val name = "mark"
  val age = 18
  //approach 1
  println(name + " is"  + age + " year old")
  //approach 2 - using s before the string
```

```
   println(s"$name is $age years old")

   //approach 3 - using f before the string (type safe manner)
   //it will throw error if there is a type mismatch
   println(f"$name%s is $age%d years old")

   //print in raw form
   println(raw"Hello \n World")
   print(s"Hello\nWorld")
```

7 - Scala - IF ELSE Statements

```
object IfElseExample {
 def main(args: Array[String]) {
   println("6. If Else Example")
   val x = 20;
   val y = 30
   var result = ""
   if (x == 20 && y == 30) {
     result = "x == 20 && y == 30"
   } else {
     result = "x != 20 OR y!= 30"
   }

   println(result)

   val result2 = if (x == 20) "x == 20" else "x != 20";
   println(result2)
   println(if (x == 20) "x == 20" else "x != 20")

 }
```

8 - Scala while Loop and do-while Loop

```
object WhileLoopAndDoWhile {
 def main(args:Array[String]){
   var x =0;
   while (x < 10) {
     println("x = "+x)
     x+= 1;  //incremental operator is not allowed (i.e ++ or --)
   }

   var y=0;
   do {
     println(s"y == $y")
      y+= 1;
   }while(y < 10);
 }
}
```

9 - Scala For Loop

```
object ForLoops {
 def main(args:Array[String]){
   for (i <-  1 to 5){
     println(s"i using to  $i")
   }
   for (i <-  1.to(5)){
     println(s"i using to  $i")
   }

   for (i <-  1 until 5){
     println(s"i using until  $i")
   }
   for (i <-  1.until(5)){
     println(s"i using until  $i")
   }
   //iterate multiple values (nesting approach : first i to all j, second i to all j, ...)
    for (i <-  1.to(5); j <- 1.to(3) ){
     println(s"i using to  $i : j using to $j")
   }
   //iterating list
    val lst = List(1,2,3,4,5,6,7,8,9,10)
    for (i <- lst) {
      println(s"i using lst $i")
    }
```

```
    //with filter
    for (i <- lst; if i< 6) {
      println(s"i using lst $i")
    }
    //for loop as expression
    val result = for {i <- lst; if i <6} yield {
      i * i
    }
    println(s"result = $result")
  }
}
```

10 - Match expressions

Its same as swithc statement. Match expression is used to select item from the list of multiple if conditions.

```
object MatchExpressions {
  def main(args:Array[String]){
    val age = 20;

    age match{
      case 20 => println(age);
      case 18 => println(age);
      case _=> println("default")
    }

    val name ="hencil"

    val result = name match{
      case "hencil" => name;
      case _ => "default"
    }
    println(name)
    println(result)

    val i = 6;
    i match{
      case 1|3|5|7|9 => println("odd number")
      case 2|4|6|8|10 => println("even number")
    }

  }
}
```

11. Functions

If no return, last statement will be returned. For some datatype, return type can be ignored.
There are 4 ways of writing functions in scala.

```
object Functions {
  object Math {

    def add(x: Int, y: Int): Int = {
      return x + y
    }

    def subtract(x: Int, y: Int): Int = {
      x - y
    }

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

    def divide(x: Int, y: Int) = x / y

    def square(x:Int) = x * x
  }

  def main(args: Array[String]) {
    println( Math.add(10,20))
    println(Math.subtract(10, 23))
    println(Math.multiply(10, 23))
    println(Math.divide(10, 23))
    println(Math square 5)  // if single argument, no need to use ()
  }
}
```

12 - Anonymous Functions + Default Values Function + more

```
object FunctDefaultValue {
  object Math{
```

```
  def add(x:Int, y:Int) : Int ={
   return x+y
  }
  def +(x:Int, y:Int) : Int ={ //not operator overloading.
   return x+y              //but function
  }
 }
 def print(x:Int, y:Int): Unit = { //Unit == not return value.
    println(x+y);
  }

 def defaltAdd(x: Int, y: Int=20) : Int ={
  return x + y
 }
 def main(args:Array[String]){
  print(10,20) //Unit - no return value
  println(Math.+(10, 25));//operator as function
  println(defaltAdd(3)) //default argument
  var add = (x: Int, y: Int) => x+y //anonymous function
  println(add(10,23))

 }
}
```

13 - Scala - Higher Order Functions

- functions that can take function as argument and return function as result.
  - **right arrow => which separates the function's argument list from its body.**

_ refers wildcard. _+_ i.e add something with something.

```
object HigherOrderFunction {
 def math(x: Double, y: Double, fun: (Double, Double)=> Double): Double = fun(x,y);

 def mathNestedFunction(x:Double, y:Double, z:Double, fun: (Double, Double)=> Double): Double=
fun(fun(x,y),z);


 def main(args:Array[String]){
  //first approach
  val result1 = math(50,20, (x,y)=> x+y)
  print(result1)
  val result2 = math(50,20, (x,y)=> x*y)
  print(result2)
  val result3 = math(50, 20, (x,y) => x max y);
  println(result3)

  //second approach
  val result4 = mathNestedFunction(10, 20, 30, (x,y) => x *y)
  println(result4)

  // using "_". its whildcard.
  val result5 = mathNestedFunction(10, 20, 30,  _*_)
  println(result4)

 }
}
```

14 - Scala - Partially Applied Functions

Some parameters are prefixed and rest will be passed later stage when required.

```
import java.util.Date

object PartiallyAppliedFunctions {

 def log(date: Date, message: String) = {
  println(date + "  " + message)
 }

 def main(args: Array[String]) {
  val sum = (a: Int, b: Int, c: Int) => a + b + c
  //partially applied function
  val f = sum(10, 20, _: Int) //_ is wildcard. it will be fed later.
  println(f(100))

  //real world example.
  val date = new Date;
  val newLog = log(date, _: String) //partially applied function
  newLog("The messae 1")
  newLog("The messae 2")
 }
}
```

15 - How to use closures in Scala

Closure is a function which use one or more variables which are declared outside the function.

//impure closure - if the variable value changed inside the function.

//pure closure - variable is declared as literal (val). so it can't be changed.

```
object Closure {
  var number = 10
  val add = (x: Int) => x + number
  def main(args: Array[String]) {
    number = 100
    //add takes the last value of variable number (declared outside)
    println(add(20))
     }
}
```

16. Function Currying in Scala

- Currying is the technique of transforming a function that takes multiple arguments into a function that takes single argument.

```
object Currying {
  def add(x:Int, y: Int) = x+ y
  def add2(x:Int) = (y:Int) => x+ y;
  def add3(x:Int)(y:Int) = x+ y;
  def main(args:Array[String]){
     println(add(5,10))
     println(add2(5)(10))

     //second approach of calling
     val sum40 = add2(40)
     println(sum40(20))
     //third approach - new signature
     println(add3(100)(200));
     //fourth appraoch
     //val Sum50 = add3(50) // now error. so use partial argument
     val sum50 = add3(50)_;
     println(sum50(400))
  }
}
```

17. Strings

```
object Strings {
  val str1: String = "Hello World" // same as java and it uses java lib.
  val str2:String = "max"
  val num1 = 75
  val num2 = 100.25
  def main(args:Array[String]){
    println(str1.length())
    println(str1.concat(" Max"))
    println(str1 + str2)
    val result = printf("(%d----%f --%s)", num1, num2, str1)
    print(result)
    println("(%d------%f-----%s)".format(num1, num2, str1))
    printf("(%d------%f-----%s)", num1, num2, str1)
  }
}
```

18. Arrays

Arrays can store same type of consecutive values.

```
import Array._

object Arrays {

  //default values for array
  //string -> null
  //Boolean -> false
  //Int = 0;
  //Double = 0.0
  //approach 1
  val myArray1: Array[Int] = new Array[Int](4)

  //approach 2
  val myArray2 =  new Array[Int](5)

  //approach 3
  val myArray3 = Array(1,2,3,4,5,6,7);
  def main(args:Array[String]){
    myArray1(0) = 20;
    myArray1(1) = 30;
    myArray1(2) = 40;
    myArray1(3) = 50;
    println(myArray1)

    for(x <- myArray1){
      println(x)
```

	}
	
	for(i <- 0 to (myArray1.length -1)) {
	println(myArray1(i))
	}
	// concatination
	println("Concatination Result")
	val result=concat(myArray1, myArray2)
	for (x <- result){
	println(x)
	}
	
	}
	}

19. Lists
//similar to array but two difference.
1. arrays are mutable but list are immutable.
2. arrays are flat but list represent linked list.
	object ListExample {
	val myList: List[Int] = List(1,2,3,4,5,6);
	val names: List[String] = List("Tom", "Max");
	def main(args:Array[String]) {
	println(0::myList) // prepend values
	println(myList)
	println(names)
	println(1:: 5 :: 9 :: Nil) // :: --cons
	print(names.tail)
	println(names.isEmpty)
	//uniform list
	println(List.fill(5) (2)) // fill 5 elements of 2
	myList.foreach(println) //loop
	var sum : Int = 0;
	myList.foreach(sum+=_)
	println(f"sum $sum")
	for (name <- names) {
	println(name)
	}
	}
	}

20. Scala Sets
Collection of different elements of same datatype. Can't have duplicate values. Sets in Scala are not ordered.
Immutable sets : Object itself can't changed inside the set.
	2
	
	object ScalaSets {
	//by default all sets are immutable
	val mySet : Set[Int] = Set(1,1,1,12,3,4,5,6);
	val mySet2 : Set[Int] = Set(12,13,14,15,16,17,18,19);
	val nameSet : Set[String] = Set("Tom", "Ben", "Adam");
	
	//mutable Sets
	//var myMutableSet: scala.collection.immutable.
	
	def main (args:Array[String]){
	println(mySet)
	println(mySet(12)) //return boolean. true if it exist.
	println(nameSet("not exist"))
	println(nameSet.head);
	println(nameSet.tail);//except head, all will be return.
	println(nameSet.isEmpty)
	
	println(mySet ++ mySet2) // ++ operator
	println(mySet.++(mySet2)) //another approach.
	//intersection
	println(mySet.&(mySet2))
	println(mySet.intersect(mySet2))
	println(mySet.min)
	//for loop
	for(x <- mySet){
	println(x)
	}
	}
	}

21 - Scala Maps
Collection of key value pairs and keys are unique in the map. If the key does not exist, it
	object ScalaMaps {
	//default Maps are mutable
	val myMap: Map[Int, String] = Map(1-> "Pete",

will throw exception. If the keys are duplicated, it takes the last value.	2-> "Tom", 3-> "Ben", 4-> "George")
	val myMap2: Map[Int, String] = Map(5-> "Joe")
	def main(args: Array[String]){
	println(myMap)
	for(x <- myMap){
	println(x._1)
	println(x._2)
	}
	println(myMap.keys)
	println(myMap.values)
	println(myMap.isEmpty)
	println(myMap(4))
	//println(myMap(9)) //exception as 9 key does not exist
	//iterate the map
	myMap.keys.foreach{
	key=>
	println("key "+ key)
	println("value "+ myMap(key))
	}
	//key present in the map
	println(myMap.contains(5))
	println(myMap ++ myMap2)
	
	}
	}

22 - Scala Tuples
class that contain different kind of element (different data type i.e. heterogeneous datatype). They are immutable and values can't be changed. Fixed length.
	object ScalaTuples {
	val myTuple = (1,2, "hello", true)
	//if we use new, we can use no.of elements after Tuple.
	//max - 1 to 22 elements.
	val myTuple2 = new Tuple3(1,2, "Hai")
	def main(args:Array[String]) {
	println(myTuple)
	//upon declaring tuple, variables created with _ prefix and index.
	println(myTuple._1)
	println(myTuple._2)
	//iteration
	myTuple.productIterator.foreach {
	x => println(x)
	}
	
	//tuple can be created using below approach as well
	//only two elements will be in a single tuple
	println(1 -> "Tom" -> true)
	
	val myTuple3 = new Tuple3(1,"Hai", (2,3));
	println(myTuple3._3._2)
	}
	}

23 - Scala Options Type
Is a container that can give two values. Ie. Instance of some of instance of None.
	object ScalaOptions {
	val lst = List(1,2,3)
	val map = Map(1-> "Tom", 2-> "Ben", 3-> "Jeff")
	def main(args: Array[String]) {
	println(lst.find(_ > 7))
	println(lst.find(_ > 1)) //if exist first instance
	//extract the value
	println(lst.find(_ > 1).get)
	println(lst.find(_ > 7).getOrElse("No name found"))
	//Option declaration
	val opt : Option[Int] =None;
	println(opt.isEmpty)
	val opt2 : Option[Int] =Some(5);
	println(opt2.isEmpty)
	
	}
	}

24 - map, flatMap, flatten and filter (Higher-order Methods)
 	
// map - iterate over collection (array, list, etc)
// and apply a function on each element
	//scala - map and filter
	
	object ScalaMapFlatMapFilter {
	
	val lst = List(1,2,3,5,7,10,13);

//flatten - combine the list of list or flattern the contents of the list
 	
//FlatMap - map the collection and flatten it
	val myMap = Map(1-> "Tom", 2-> "Ben", 3-> "Jeff");
	
	def main(args: Array[String]){
	// map - iterate over collection (array, list, etc)
	// and apply a function on each element
	println(lst.map(x => x % 2))
	println(lst.map(x => "Hi" *x))
	println(myMap.map(x => "Hi" + x))
	println("hello".map(x=> x.toUpper))
	//flatten - combine the list of list or flattern the contents of the list
	//below list of list
	println(List(List(1,2,3), List(4,5,6)).flatten)
	//FlatMap - map the collection and flatten it
	println(lst.flatMap(x => List(x, x+1)))
	//filter -
	println(lst.filter(x => x%2 ==0))
	println(lst.filter(x => x%2 !=0))
	}
	}

25 - Reduce, fold or scan
//reduceLeft - takes associate binary operator function as parameter and will use it to collapse the elements.
//i.e. apply the operator on first two operand, result with next operand and finally returns the result.
 	
//fold - similar as reduce but we can pass the initial argument in fold functions.
 	
//scan - same as fold. it takes the starting value but scan will give the map of intermediate result.
	//Scala - Reduce, fold or scan (left/right)
	//reduceLeft, reduceRight, foldLeft
	object ReduceFoldScan {
	val lst = List(1, 2, 3, 5, 7, 10, 13);
	val lst2 = List("A", "B", "C");
	def main(args: Array[String]) {
	//reduceLeft - takes associate binary operator function as parameter and will use it to collapse the elements.
	//i.e. apply the operator on first two operand, result with next operand and finally returns the result.
	println(lst.reduceLeft(_ + _))
	println(lst2.reduceLeft(_ + _))
	println(lst.reduceLeft((x, y) => { println(x + " , " + y); x + y }))
	println(lst.reduceLeft(_ - _))
	println(lst.reduceRight(_ - _))
	println(lst.reduceRight((x, y) => { println(x + " , " + y); x - y; }));
	
	//fold - similar as reduce but we can pass the initial argument in fold functions.
	println(lst.foldLeft(10)(_ + _)) //initial value 10 is added.
	println(lst2.foldRight("End")(_ + _))
	//scan - same as fold. it takes the starting value but scan will give the map of intermediate result.
	println(lst.scanLeft(10)(_ + _))
	println(lst2.scanRight("End")(_ + _))
	
	}
	
	}

26 - Scala Classes
classes - blueprint for creating object.
Normal object act as singleton class and can't create object.
	//data type
	//construct must have var/val infront of member variables.
	//var getter setter
	//val getter ----
	//default -- --
	class User (var name:String, private var age: Int){
	def printName{
	println(name)
	}
	}
	
	object ScalaClass {
	
	def main(args : Array[String]) {
	//var user = new User;
	var user = new User("Tom", 25);
	println(user.name)
	user.name = "Ben"
	println(user.name)
	//if we use val, we can't overwrite the value.
	
	user.printName
	}

	}

27 - Auxiliary constructors
alternative constructor for a class.
	/.....Getter? Setter?
	//----------
	//var yes yes
	//val yes no
	//default no no
	//primary construtor - must have different constrctor than auxiliary constructors.
	//auxiliary constructor - will call the previously defined constructor with required parameter.
	class User1(private var name: String, val age: Int) {
	def this(){
	this("Tom", 32)
	}
	
	def this(name : String){
	this(name, 32);
	}
	def printName{
	println("Name " + name)
	}
	}
	
	object ClassAuxiliaryConstructor {
	
	def main(args: Array[String]) {
	var user1 = new User1("Tom", 23);
	var user2 = new User1();
	var user3 = new User1("Tom");
	println(user3.printName)
	}
	
	}

28 - How To Extend Class - Class Inheritance
Classes in scala can be extended, creating new classes which retain characteristics of the base class. This process known as inheritance. Involves a superclass and a subclass.
	package Inheritance
	
	class Polygon {
	def area: Double = 0.0;
	}
	
	object Polygon{
	def main(args: Array[String]){
	var poly = new Polygon;
	var rect = new Rectangle(10,22);
	var triangle = new Triangle(10,22);
	printArea(poly)
	printArea(rect)
	printArea(triangle)
	}
	
	def printArea(p: Polygon) {
	println(p.area)
	}
	}
	
	class Rectangle(var width : Double, var height:Double) extends Polygon{
	override def area: Double = width * height;
	}

29 - Scala Abstract Class
1. restrict the instantiation of the super class.
2. Important method must be implemented in child classes.
Abstract class can't be instantiated directly but can be instantiated via child class. Abstract class may or may not contain abstract method. If the class contain at least one abstract method, it can't be instantiated directly. It provides common interface which allows the subclass to be interchanged with all other subclasses.
	package AbstractClass
	
	class Triangle(var width: Double, var height:Double) extends Polygon{
	override def area: Double = width * height /2
	}
	
	package Inheritance
	
	abstract class Polygon {
	def area: Double;
	}

	object Polygon{
	  def main(args: Array[String]){
	  var rect = new Rectangle(10,22);
	  var triangle = new Triangle(10,22);
	  printArea(rect)
	  printArea(triangle)
	  }
	
	  def printArea(p: Polygon) {
	  println(p.area)
	  }
	}

30 - Scala Lazy Evaluation
lazy evaluation - every expression evaluation waits for its first use. Scala support strict as well as Lazy evaluation.
	Select project -> right click -> scala -> scala interpreter
	scala> val e = 9
	e: Int = 9
	
	scala> lazy val l=9
	l: Int = <lazy>
	
	scala> l
	res0: Int = 9
	
	class strict{
	 val e = {
	  println("strict")
	 9
	 }
	}
	
	class LazyEval{
	 lazy val l= {
	  println("lazy")
	 9
	 }
	}
	
	object LazyEvaluationDemo {
	 def main(args: Array[String]) {
	  //1st case - lazy is not evaluated as its not used.
	  val x = new strict
	  val y = new LazyEval
	  println("----------")
	  //2nd case
	  println(x.e)
	  println(y.l)
	 }
	}

31 - Scala Trait - Scala does not allow multiple inheritance. - interface - define set of methods. Traits in scala are partially implemented interface.	