Scala

Michael Spagon and Eric Addison
UT Austin - Parallel Algorithms - Summer 2017

Overview

- What is Scala?
- History of Scala
- Big Companies Using Scala
- Example Code Snippets
- Language Features
- Parallelism in Scala
- Spark

What is Scala?

Scala stands for “scalable language”

- Runs on JVM
- Interoperates with Java libraries
- **Functional** and Object Oriented features in one place
- Killer application - Parallel Programming!

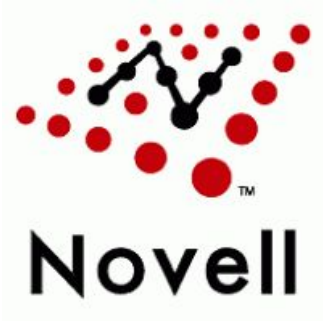
History of Scala

- **Martin Odersky**, a professor at Ecole Polytechnique Fédérale de Lausanne (EPFL) of Lausanne, Switzerland.
- Public release in 2004 on the Java Platform.
- A second version followed March 2006.
- Scala has extensive support for functional programming since it's inception.
- Java introduced lambda expressions in Java 8 (2014).
- 2011 received large research grant (€2.3 million)

Worldwide, Jun 2017 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	22.7 %	-1.2 %
2		Python	16.1 %	+3.8 %
3		PHP	9.3 %	-0.9 %
4		C#	8.2 %	-0.6 %
5		Javascript	7.9 %	+0.5 %
6		C++	6.8 %	-0.3 %
16		Scala	1.2 %	+0.2 %
17	↓↓	Perl	0.8 %	-0.3 %

Who’s Using Scala



And many more...

In Java

```
public class Car {
    final private int year;
    private int miles;

    public int getYear()
    {
        return year;
    }

    public int getMiles()
    {
        return miles;
    }

    set miles?
    Constructor?
}
```

In Scala

```
class Car(val year:Int, var miles: Int)
```

Hello, World!

Compiled Version

```
1  object HelloWorld extends App {
2    println("Hello, World!")
3  }
```

Interpreted Version

```
1  println("Hello, World!")
```

Simple Programming Features

```
// basic assignment
var x = 10  // a variable
val y = 20  // a value

x = 20      // ok
y = 100     // ERROR! immutable!
```

```
// function definition
def square(x:Double): Double {
  x*x
}
square(2)   // = 4
```

Simple Programming Features

```
// no switch! use match
i = 3
i match {
  case 1  => println("Red")
  case 2  => println("Green")
  case 3  => println("Blue")
  case default  => println("error: " + default.toString)
} // prints "Blue"
```

```
// control flow
for(a <- 1 to 10)
  println(a)

for(a <- (1 to 6).map(a=>math.Pi/a))
  println("%1.3f".format(math.cos(a)))

for(a <- 1 to 3; b <- 3 to 6)
  yield(a,b)

if (1<2) {
  println("yup")
}

var j = 0
while(j<10){
  println(j)
  j = j+1
}
```

Language Features

- Hybrid OO / Functional programming
- Static typing
- Compiled *and* interpreted!
- REPL
- Standard Library
  - Data structures (sequential and parallel)
  - Concurrency
  - Math
  - IO
  - System
  - Regex
- Some cool features not found in Java
  - Operator overloading
  - Named parameters
  - Default values
  - Raw Strings

Simple OO in Scala

```scala
// this is a class
// the class parameters (x,y) have default arguments
class Point(
    val x: Double = 0.0, val y: Double = 0.0
) {
  import Point._

  def this() = this(0.0, 0.0)   // defining a no-arg ctor

  def distanceToPoint(other: Point) =
    distanceBetweenPoints(x, y, other.x, other.y)
}

// this is a companion object to the Point class
// it is a singleton object
object Point {
  def distanceBetweenPoints(x1: Double, y1: Double,
      x2: Double, y2: Double) = {
    math.hypot(x1 - x2, y1 - y2)
  }
}
```

```scala
object PointDemo extends App {
  val point1 = new Point(1,2)
  val point2 = new Point(x=3) // named param
  println("point1 = (" + point1.x + ", " + p
  println("point2 = (" + point2.x + ", " + point2.y + ")")
  println("Distance between points: " + point1.distanceToPoint(point2))
}
```

Functional Programming Elements

Lambda Functions

```
// a lambda function
val func = (x:Int) => x+x
func(10)
```

Map / Reduce

```
// map reduce
val l = List(1,2,3,4,5)
l.map( x => x*2 )
l.reduce( (a,b) => a+b )
```

First Class Functions

```
1  def modFunc(x:Int ) = {
2    x%2==0
3  }
4
5  val func = modFunc(_)
6  val list = List range(1, 10)
7  val filteredList = list.filter(func)
```

Concurrency in Scala

- Scala has access to all the same thread APIs as Java
  - Runnable, Callable, Executors, Futures, etc.

- Scala has *parallel* collections!
  - Provides easy to use parallel operations
  - Similar operations to sequential collections
  - Very easy to create parallel structures
  - Careful! Need to avoid side-effecting and non-associative operations!

Basic Threading

```scala
// simpleThread.scala
// a VERY simple example of threading in scala

println("Creating a thread...")

val thread = new Thread{
  override def run{
    println("Thread starting!")
    Thread.sleep(2000)
    println("Thread done!")
  }
}

thread.start()
println("Thread created and started")
```

Parallel Collections

Scala.collection.parallel.mutable

Scala.collection.parallel.immutable

Two ways to create them

1.

```
import scala.collection.parallel.immutable.ParVector
val pv = new ParVector[Int]
```

2.

```
val pv = Vector(1,2,3,4,5,6,7,8,9).par
```

Change back to sequential by calling .seq

Sequential Fibonacci

```
object ParallelCollect extends App{
  def fib(n:Int):Int = if (n<2) 1 else fib(n-1)+fib(n-2)


  for(i <- (40 to 15 by -1)) {
    println(fib(i))
  }
}
```

```
165580141
102334155
63245986
39088169
24157817
14930352
9227465
5702887
3524578
2178309
1346269
832040
514229
317811
196418
121393
```

Parallel Fibonacci

```scala
object ParallelCollect extends App{
  def fib(n:Int):Int = if (n<2) 1 else fib(n-1)+fib(n-2)


  for(i <- (40 to 15 by -1).par) {
    println(fib(i))
  }
}
```

```
17711
10946
6765
4181
2584
1597
987
196418
121393
75025
46368
28657
2178309
1346269
832040
317811
```

Using Parallel Collections: Example 1

```
def time[R](block: => R): R = {
    val t0 = System.nanoTime()
    val result = block    // call-by-name
    val t1 = System.nanoTime()
    println("Elapsed time: " + (t1 - t0)/1000000.0 + "ms\n\n")
    result
}

val n = 1000000

// a sequential array
val array = (1 to n).toArray

// a parallel array
val parArray = (1 to n).toArray.par

println("\n\nSequential reduce")
println("------------------")
time[Int](array.reduce( (a,b) => a+b))

println("Parallel reduce")
println("------------------")
time[Int](parArray.reduce( (a,b) => a+b))
```

Using Parallel Collections: Example 1

```
eric@eric-Lenovo-U410 ~/UT/Parallel/scala $ scala src/parStructures.scala


Sequential reduce
------------------
Elapsed time: 298.713211ms


Parallel reduce
------------------
Elapsed time: 45.450122ms
```

Using Parallel Collections: Example 2

```
import util.Random.nextInt
println()

val n = 10
val x = 5
val y = 10

val A = Seq.fill(n)(nextInt%10+10).par
println("A = " + A)

val p = (a:Int) => {if ((a>=x)&&(a<=y)) 1 else 0}
val B = A.map(p)
println("B = " + B)

val C = B.scanLeft(0)(_+_)
println("C = " + C)

val size = C(n)
val E = Array.fill[Int](size)(0).par

B.zipWithIndex.foreach(s=> if(s._1==1) E(C(s._2)) = A(s._2))
println("E = " + E)

// or just use filter :)
val p2 = (a:Int) => {(a>=x)&&(a<=y)}
val F = A.filter(p2)
println("F = " + F)

println()
```

Using Parallel Collections: Example 2

```
A = ParVector(3, 5, 4, 7, 6, 16, 9, 14, 17, 5)
B = ParVector(0, 1, 0, 1, 1, 0, 1, 0, 0, 1)
C = ParVector(0, 0, 1, 1, 2, 3, 3, 4, 4, 4, 5)
E = ParArray(5, 7, 6, 9, 5)
F = ParVector(5, 7, 6, 9, 5)
```

Spark

- A large-scale data processing engine
- An open source cluster-computing framework
- Faster mapReduce than Hadoop
- Bindings for *Scala,* Python, Java, R
- Operates with popular big data technologies (HDFS, YARN, Cassandra, etc)
- While Spark claims to be a general purpose cluster-computing framework:

By far the biggest adoption of Spark (and Scala) has been in the ***Big Data*** *and* ***Data Science*** *communities*

Spark

- Includes APIs for distributed:
  - Data processing
  - SQL queries
  - Streaming analytics
  - Machine learning
  - Graph processing
- Spark is built around the Resilient Distributed Dataset (RDD)
  - Distributed among nodes in a cluster
  - Fault tolerant
  - Immutable
  - Lazy-evaluation

Spark

- How good is Spark?

… Spark sorted 100TB of data using 206 EC2 i2.8xlarge machines in 23 minutes. The previous world record was 72 minutes, set by a Hadoop MapReduce cluster of 2100 nodes. This means that Spark sorted the same data 3X faster using 10X fewer machines. All the sorting took place on disk (HDFS), without using Spark's in-memory cache.

-https://spark.apache.org/news/spark-wins-daytona-gray-sort-100tb-benchmark.html

Spark Example: Word Count

```
val textFile = sc.textFile("warandpeace.txt")
val splitFile = textFile.flatMap(line => line.split(" "))
val counts = splitFile.map(word=>(word,1)).reduceByKey(_ + _)
counts.saveAsTextFile("counts.txt")

println(counts.counts)
counts.sortBy(_._2, ascending=false).take(10).foreach(s=>println(s))
```

Spark Example: Word Count

```
scala> counts.sortBy(_._2, ascending=false).take(10).foreach(s=>println(s))
(the,31796)
(and,20731)
(,19664)
(to,16446)
(of,14893)
(a,10113)
(in,8288)
(he,7701)
(his,7698)
(that,7327)
```

Resources

- https://en.wikipedia.org/wiki/Scala_(programming_language)
- https://twitter.github.io/scala_school/concurrency.html
- http://docs.scala-lang.org/overviews/parallel-collections/overview.html
- https://www.scala-lang.org/api/2.12.2/
- https://spark.apache.org/examples.html

- https://www.toptal.com/spark/introduction-to-apache-spark