

The State of Scala 2010

Ben Hutchison

Senior Developer, REA Group

Coordinator, Melbourne Scala User Group

Scala in 1 slide...

- A modern, statically-typed programming language
 - Targets the Java Virtual Machine (JVM)
 - Deeply Java-compatible
- "*Object-Functional* "
 - attempt to unify Object-oriented & Functional programming
 - Concise (*really!*) and elegant
 - Strong type-safety
 - Quite a learning curve
- Created in 2004 by Martin Odersky at Swiss University EPFL
- Open source. Academic funding. Trying to break into commercial mainstream

Scala code sample 1: Conveniences

```
object Example1 {  
  val Vic = "Victoria"  
  val Nsw = "NSW"  
  val Act = "ACT"  
  
  def main(args: Array[String]) {  
    val states = Set(Vic, Nsw, Act)  
  
    var stateCapitals = Map(Vic -> "Melbourne", Nsw -> "Sydney")  
    stateCapitals += (Act -> "Canberra")  
  
    //output: Set(Melbourne, Sydney, Canberra)  
    println(states.map(stateCapitals))  
  }  
}
```

Scala code sample 2: DSLs in ScalaTest

```
class StackSpec extends WordSpec with ShouldMatchers {  
  "A Stack" when {  
  
    "empty" should {  
  
      val stack = new Stack[Int]  
  
      "be empty" in {  
        stack should be ('empty)  
      }  
  
      "complain when popped" in {  
        evaluating { stack.pop() } should produce [NoSuchElementException]  
      }  
    }  
  }  
}
```

Scala code sample 3: Std. Library

```
package scala.collection
import generic._
import mutable.{Builder, StringBuilder, Buffer, ArrayBuffer,
  ListBuffer}

trait TraversableLike[+A, +Repr] extends HasNewBuilder[A, Repr]
                                   with FilterMonadic[A, Repr]
                                   with TraversableOnce[A] {
//...
  def filter(p: A => Boolean): Repr = {
    val b = newBuilder
    for (x <- this)
      if (p(x)) b += x
    b.result
  }
}
```

Favorite Features: Java Interop

- Interoperability with Java is near *seamless*
 - Akin to C and C++
 - Call and be called by Java code without glue code, wrappers or recompilation
 - Inherit from Java classes
 - Identical data model
 - Reuse Java tools & skills: Junit, Eclipse, ANT

```
import org.joda.time._
def datesBetween(start: DateTime, end: DateTime): Seq[DateTime] = {
  for (i <- 0 until Days.daysBetween(start, end).getDays()) yield
    start.plusDays(i)
}
def mkTimeOfDay = new LocalTime(choose(0, 23), choose(0, 59))
```

Favorite Features: Type Inference

- Type Inference:
 - **Don't:** explicitly specify types of variables & functions
 - **Instead:** try to infer it from the context it is used in
- Enables
 - Rich semantic structure in code, plus lightweight syntax
 - Approaches the consistency of dynamically typed languages
- Scala uses *local* not *global* type inference
 - Types inferred at method-level scope, not across whole program
 - Method parameter types must still be explicitly specified
 - Global type inference for OO languages is a hard, unsolved problem
 - OO type systems are inherently “open universes”

Minimal Type Info (highlighted in red)

```
object Example1 {  
  val Vic = "Victoria"  
  val Nsw = "NSW"  
  val Act = "ACT"  
  
  def main(args: Array[String]) {  
    val states = Set(Vic, Nsw, Act)  
  
    var stateCapitals = Map(Vic -> "Melbourne", Nsw -> "Sydney")  
    stateCapitals += (Act -> "Canberra")  
  
    //output: Set(Melbourne, Sydney, Canberra)  
    println(states.map(stateCapitals))  
  }  
}
```


Favorite Features: Implicit Conversions

- Implicit conversions are Scala's way to *retrofit behaviour onto someone else's code* or API
 - Java *has no good way to do this*
 - More controlled, fewer side-effects than Ruby-style *open classes*
 - More powerful than C#'s *extension methods*
 - Ubiquitous in Scala code
- *Defining* an implicit conversion tells the compiler:
 - “Here's how convert objects of type A into type B”, where typically
 - Type “A” = Someone else's class you wish to extend
 - Type “B” = Your extensions to type A
- *Importing* an implicit conversion into a scope
 - Enables “on demand” conversions by compiler when needed

Using Implicit Conversions

```
//a conversion from String to RichString is imported from PreDef by
//default
object ConversionExample {

  def main(args: Array[String]) {
    //output: MyClass
    println("MyClass.scala".stripSuffix(".scala"))

    //what the implicit conversion is doing under the covers
    println(new RichString("MyClass.scala").stripSuffix(".scala"))
  }
}
```

Boxing across the Object/Primitive divide

- Java (the JVM) has two memory representations for data
 - *Objects*: Variable size record stored individually on the heap
 - *Primitives* (aka Value Types, Structs): Fixed size records embedded within another object, array or stack frame
- We often want to write generic code that works over either primitive or object types
 - Eg A HashMap that can store Int, Double, Char or Object types
 - This requires *Boxing*, an inefficient & slow process whereby primitive data is copied into & out of object wrappers
 - Because Scala encourages abstraction and generic code, boxing has been a major performance challenge for Scala to date
- *Specialization* is a new Scala 2.8 feature to address this problem

Specialization: Making Generic Code Efficient

- `@specialized`: An annotation to ask the Scala compiler to transparently generate and utilize multiple versions of a method or class
 - One default generic version
 - Versions specific to a particular primitive data type, eg `Int`
- Specialization is probably the first time Scala code *runs faster than equivalent Java code*

```
trait Function1[@specialized(scala.Int, scala.Long, scala.Float,
  scala.Double) -T1, @specialized(scala.Unit, scala.Boolean,
  scala.Int, scala.Float, scala.Long, scala.Double) +R] extends
  AnyRef { self =>
  def apply(v1:T1): R
  ...
}
```

Where is Scala in 2010?

- In transition...
- From:
 - “academically interesting” research project
 - Popular only among niche of programming elite
 - Mainly research & hobby usage
- Aspires to:
 - Be a useful, productive language for pragmatic/commercial usage
 - Appeal to mainstream developers

Where is Scala in 2010?

Commercial uptake & backing is beginning

- Twitter
- LinkedIn
- Siemens
- Grid Gain
- Sony Imageworks
- Électricité de France Trading
- Novell “Pulse” collaboration app
- The Guardian newspaper's “Open Platform” content API
- Scala Job market
 - perhaps 100 globally, AFAIK nothing in Melbourne

Meet the Team



- Scala dev is dependent upon a small team of people
 - No large corporate (Oracle/MS/IBM/Apple/Sun etc) sponsors Scala
- Academic contributors
 - Martin Odersky [lead]
 - Adriaan Moors [type system & compiler]
 - Iulian Dragos, Lucas Rytz, Hubert Plociniczak, Aleksandar Prokopec, Gilles Dubochet, Philipp Haller, Stéphane Micheloud, Tiark Rompf, Ingo Maier, Antonio Cunei
- Community/Industry contributors
 - Paul Phillips, retired world poker champion [50% of all commits]
 - Mark Harrah, Jason Zaugg, Johannes Rudolf, Ismael Juma, Nathan Bronson, Seth Tissue, Ilya Sergey
- IDEs

Scala Solutions: Commercial Support

- Founded by Odersky
- On sabbatical from EPFL
- Consulting
- Training
- Support
- Scala version migration



The screenshot shows a Mozilla Firefox browser window displaying the Scala Solutions website. The address bar shows the URL <http://www.scalasolutions.com/>. The website features a navigation menu with links for Home, The Company, Products, Training, Consulting, Partners, Community, What is Scala?, and Jobs. The main content area is titled "Community" and includes a sub-header "More than 100,000 Scala Users" and a list of associated entities: "Twitter, LinkedIn, Sony, The Guardian, Xerox...". Below this, there is a section titled "We have founded Scala Solutions to provide the tools and services needed by Scala developers to create mission-critical applications that are reliable, quick to develop and easy to maintain." followed by a paragraph about Scala Solutions' commercial services. A small portrait of a man is visible next to this text. The right sidebar contains sections for "Training in Lausanne", "Training Other Locations", "Products", and "Consulting", each with a list of items and descriptions.

Scala: A solid history

- Precursors
 - Java, ML, Haskell, Funnel
- 2001
 - Design begun on Scala
- 2003
 - 1.0.0-b2
- 2004
 - 1.1.0 → 1.3.0.9
- 2005
 - 1.3.0.9 → 1.4.0.3
- 2006
 - 1.4.0.4 → 2.3.1
- 2007
 - 2.3.2 → 2.6.1
- 2008
 - 2.7.0 → 2.7.2
- 2009
 - 2.7.3 → 2.7.7
- **2010**
 - **2.8.0, 2.8.1**

Scala Releases in 2010

- Scala 2.8.0 [August]
 - Very protracted, difficult release
 - Ran late, ended up taking 18 months to go final
 - Major new features and changes. Martin Odersky commented “in retrospect, this should have been Scala 3.0”
 - Collections API refactored
 - Named and Default parameters
 - Performance optimizations for primitive data types
 - Design of Arrays finally “fixed”
 - Manifests (“un erased generic types”)
- Scala 2.8.1 [November]
 - Stabilization: heaps (~100) tickets closed and bugs fixed
 - No new features

Simple Build Tool (SBT)

- One of the most popular open source tools yet written in Scala (other than Scala itself)
 - *"a simple build tool for Scala projects that aims to do the basics well"*
 - Created by Mark Harrah
 - Becoming the defacto standard for Scala (like Rake, Ant)
- Understands standard Maven layout, `src/main/scala`, `src/test/scala` etc
 - Compile and run tests with no configuration
- Scala-based config to declare Maven-style dependencies

```
import sbt._  
  
class Configuration(info: ProjectInfo) extends DefaultProject(info) {  
  val releases = "ScalaTools Releases" at "http://scala-tools.org/repo-releases/"  
  val scalatest = "org.scalatest" % "scalatest" % "1.2" % "test" withSources()  
  val junit = "junit" % "junit" % "4.4" % "test" withSources()  
}
```

ScalaDays 2010



- First official Scala conference held at EPFL, Lausanne, Switzerland over 2 days
 - 32 speakers including an Australian (Tony Sloane, Macquarie Uni)
 - Announced: Stanford's Pervasive Parallelism Lab embracing Scala, entering collaboration with EPFL
 - Videos of all sessions freely available
- Preceded in 2009 by Scala Liftoff
 - “Unconference” held in San Francisco

Criticisms of Scala

- Scala has its fans but also its critics.
- I hear two recurring themes
 - “Scala is too complicated. Its learning curve is too steep”
 - *Eg Hairy compiler errors “contravariant type T occurs in covariant position in type (implicit ev: <: <[D,math.dimension.D2Plus])T of method y”*
 - “Scala is too academic and theoretical. It includes lots of research features that are of little use or value in practice”
 - Existential Types, Higher-Kinded Types, Type Members
- Yes, but ...
- Scala is advanced technology incorporating a lot of ideas new/unfamiliar to mainstream programming
- Even excellent new ideas take time to accept & appreciated

Acceptance of Mathematical Vectors

- Vectors and vector maths are so central to modern maths
 - Eg $\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta$
- ..we might forget that they were viewed with distrust and skepticism less than a century ago
- Eminent physicist & mathematician A.E.Milne wrote *Vectorial Mechanics* in 1948. From the preface:

"Professor Sydney Chapman, my former teacher...first expounded to me the view that vectors were not merely a pretty toy, suitable for elegant proofs of general theorems, but were a powerful weapon of workday mathematical investigation..

I did not at first believe him; I had been brought up in the idea that ... vectors were like a pocket-rule, that needs to be unfolded before it can be applied an used...."

Thank You