Scala Clinic

Scala Standard Libary

Jim Powers

Patch.com

15 February 2012

Intro to the Scala Library

Intro to the Scala Library

What's covered by the Scala Library?

Intro to the Scala Library

What's covered by the Scala Library?

- Collections

Intro to the Scala Library

What's covered by the Scala Library?

- Collections
  - Immutable

Intro to the Scala Library

What's covered by the Scala Library?

- Collections
  - Immutable
  - Mutable

Intro to the Scala Library

What's covered by the Scala Library?

- Collections
  - Immutable
  - Mutable
  - Parallel

Intro to the Scala Library

What's covered by the Scala Library?

- Collections
  - Immutable
  - Mutable
  - Parallel
- XML

Intro to the Scala Library

What's covered by the Scala Library?

- Collections
  - Immutable
  - Mutable
  - Parallel
- XML
- Parsing

Intro to the Scala Library

What's covered by the Scala Library?

- Collections
  - Immutable
  - Mutable
  - Parallel
- XML
- Parsing
- Actors

Intro to the Scala Library

What's covered by the Scala Library?

- Collections
  - Immutable
  - Mutable
  - Parallel
- XML
- Parsing
- Actors
- Concurrency

Intro to the Scala Library

What's covered by the Scala Library?

- Collections
  - Immutable
  - Mutable
  - Parallel
- XML
- Parsing
- Actors
- Concurrency
- Process management

Intro to the Scala Library

What's covered by the Scala Library?

- Collections
  - Immutable
  - Mutable
  - Parallel
- XML
- Parsing
- Actors
- Concurrency
- Process management
- Swing

Intro to the Scala Library

What's covered by the Scala Library?

- Collections
  - Immutable
  - Mutable
  - Parallel
- XML
- Parsing
- Actors
- Concurrency
- Process management
- Swing
- Testing

Collections

- Mutable

Collections

- Mutable
- Immutable

Collections

- Mutable
- Immutable
- Parallel

Collections

- Arrays

Collections

- Arrays
  - Plain old Java Arrays

Collections

- Arrays
  - Plain old Java Arrays
  - "Pimped" with *implicit conversions*

Collections

- Arrays
  - Plain old Java Arrays
  - "Pimped" with *implicit conversions*
- Buffers

Collections

- Arrays
  - Plain old Java Arrays
  - "Pimped" with *implicit conversions*
- Buffers
- Lists

Collections

- Arrays
  - Plain old Java Arrays
  - "Pimped" with *implicit conversions*
- Buffers
- Lists
- Stacks

Collections

- Arrays
  - Plain old Java Arrays
  - "Pimped" with *implicit conversions*
- Buffers
- Lists
- Stacks
- Vectors

Collections

- Arrays
  - Plain old Java Arrays
  - "Pimped" with *implicit conversions*
- Buffers
- Lists
- Stacks
- Vectors
- Sets (and BitSets)

Collections

- Arrays
  - Plain old Java Arrays
  - "Pimped" with *implicit conversions*
- Buffers
- Lists
- Stacks
- Vectors
- Sets (and BitSets)
- Maps

Collections

- Arrays
  - Plain old Java Arrays
  - "Pimped" with *implicit conversions*
- Buffers
- Lists
- Stacks
- Vectors
- Sets (and BitSets)
- Maps
- Queues

Collections

- Arrays
  - Plain old Java Arrays
  - "Pimped" with *implicit conversions*
- Buffers
- Lists
- Stacks
- Vectors
- Sets (and BitSets)
- Maps
- Queues
- Ranges

Collections

- Arrays
  - Plain old Java Arrays
  - "Pimped" with *implicit conversions*
- Buffers
- Lists
- Stacks
- Vectors
- Sets (and BitSets)
- Maps
- Queues
- Ranges
- Streams

Collections

Type	Immutable	Mutable	Parallel
Arrays		✓	✓
Buffers		✓	✓
Lists	✓	✓	✓
Stacks	✓	✓	✓
Vectors	✓	✓	✓
Sets	✓	✓	✓
BitSets	✓	✓	✓
Maps	✓	✓	✓
Queues	✓	✓	✓
Ranges	✓		✓
Streams	✓		✓

Collections: Immutable

- No direct modification

Collections: Immutable

- No direct modification
- Operations return new objects

Collections: Immutable

- No direct modification
- Operations return new objects
  - Structural sharing

Collections: Immutable

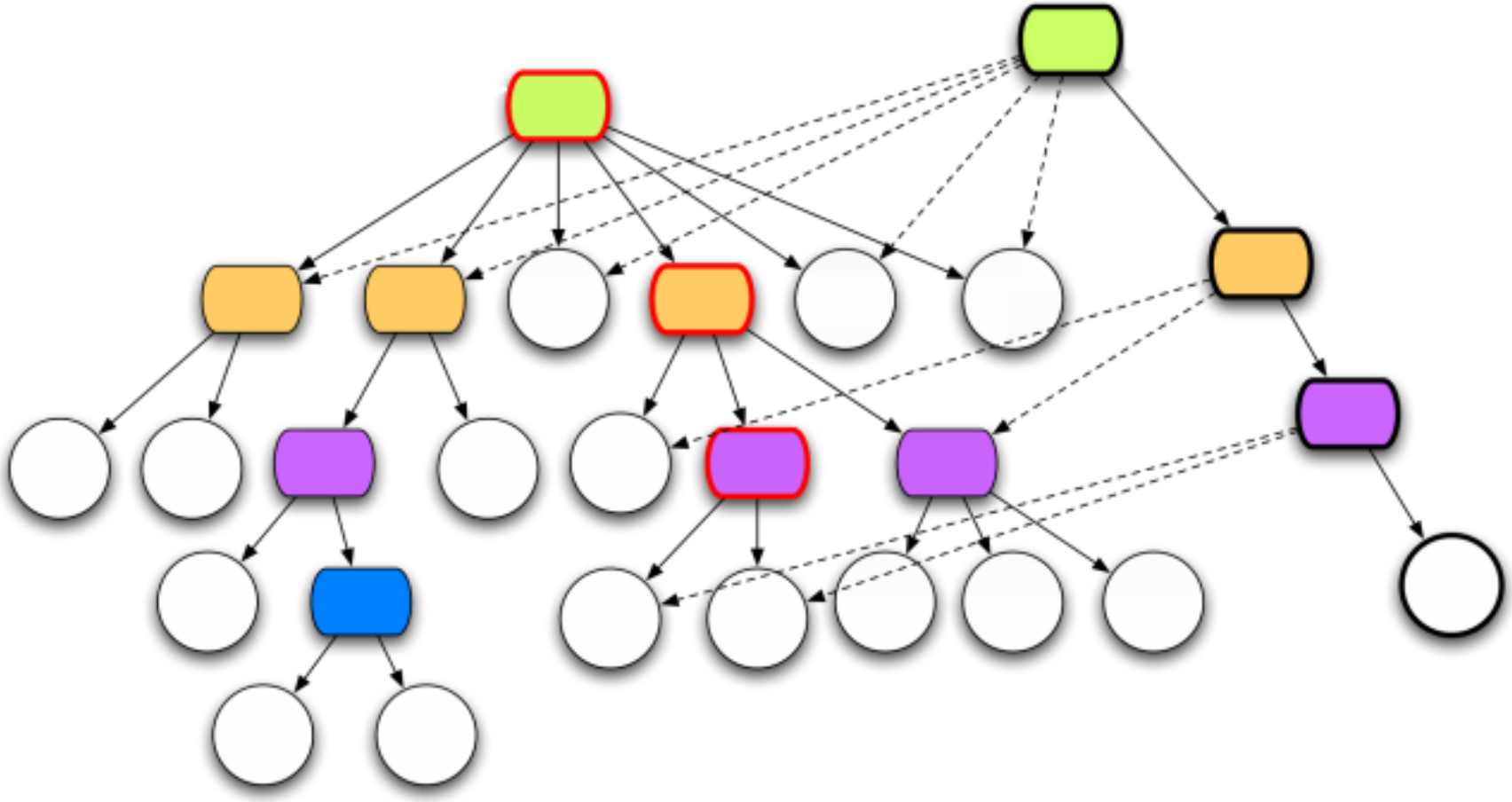
- No direct modification
- Operations return new objects
  - Structural sharing
  - Easily shared in a concurrent environment

Collections: Immutable

- No direct modification
- Operations return new objects
  - Structural sharing
  - Easily shared in a concurrent environment
    - *No locks!*

Collections: Immutable

Structural Sharing



Collections: Parallel

- `.par` $\rightarrow$ returns a parallel collection

Collections: Parallel

- `.par` $\rightarrow$ returns a parallel collection
  - $O(1)$ conversion time!

Collections: Parallel

- `.par` $\rightarrow$ returns a parallel collection
  - $O(1)$ conversion time!
- A number of operations can be done concurrently

Collections: Parallel

- `.par` $\rightarrow$ returns a parallel collection
  - $O(1)$ conversion time!
- A number of operations can be done concurrently
- Uses Fork/Join

Collections: Parallel

- `.par` $\rightarrow$ returns a parallel collection
  - $O(1)$ conversion time!
- A number of operations can be done concurrently
- Uses Fork/Join
- Example parallel functions

Collections: Parallel

- `.par` $\rightarrow$ returns a parallel collection
  - $O(1)$ conversion time!
- A number of operations can be done concurrently
- Uses Fork/Join
- Example parallel functions
  - `map`

Collections: Parallel

- `.par` $\rightarrow$ returns a parallel collection
  - $O(1)$ conversion time!
- A number of operations can be done concurrently
- Uses Fork/Join
- Example parallel functions
  - `map`
  - `flatMap`

Collections: Parallel

- `.par` $\rightarrow$ returns a parallel collection
  - $O(1)$ conversion time!
- A number of operations can be done concurrently
- Uses Fork/Join
- Example parallel functions
  - `map`
  - `flatMap`
  - `foreach`

Collections: Parallel

- `.par` $\rightarrow$ returns a parallel collection
  - $O(1)$ conversion time!
- A number of operations can be done concurrently
- Uses Fork/Join
- Example parallel functions
  - `map`
  - `flatMap`
  - `foreach`
  - `filter`

Collections: Parallel

- `.par` $\rightarrow$ returns a parallel collection
  - $O(1)$ conversion time!
- A number of operations can be done concurrently
- Uses Fork/Join
- Example parallel functions
  - `map`
  - `flatMap`
  - `foreach`
  - `filter`
  - `reduce`

Collections: Demo

Demo Collections

XML

XML Literals!

Example: *XML in your code*

```
val foo = <div>
            <span>This is some XML</span>
          </div>
// foo:scala.xml.Elem
```

XML

XML Literals!

Example: *XML producing functions*

```
def foo(x:String) = <div>
                      <span>{x}</span>
                    </div>
// foo:String => scala.xml.Elem
```

XML

XML Literals!

Example: *XML element splicing*

```
import scala.xml.Elem
def foo(x:Elem) = <div>
                    <span>{x}</span>
                  </div>
// foo:scala.xml.Elem => scala.xml.Elem
```

XML

XML Literals!

Example: *Generalized XML splicing*

```
import scala.xml.NodeSeq
def foo(x:NodeSeq) = <div>
                       <span>{x}</span>
                     </div>
// foo:scala.xml.NodeSeq => scala.xml.Elem
```

XML

XML Literals!

Example: *Pattern Matching*

```
import scala.xml.Elem

def foo(x:Elem):String = (x match {
    case <div>{y}</div> => y
    case <span>{y}</span> => y
    case <script>{y}</script> => y
    case _ => "Unhandled case"
  }).toString
```

XML

- XML support generally quite good, but has a fair number of warts

XML

- XML support generally quite good, but has a fair number of warts
- EPFL/TypeSafe actively looking to overhaul XML support in the language and standard libraries at some point.

XML

- XML support generally quite good, but has a fair number of warts
- EPFL/TypeSafe actively looking to overhaul XML support in the language and standard libraries at some point.
  - Perhaps with the forthcoming quasi-quoting and macro systems.

XML

- XML support generally quite good, but has a fair number of warts
- EPFL/TypeSafe actively looking to overhaul XML support in the language and standard libraries at some point.
  - Perhaps with the forthcoming quasi-quoting and macro systems.
- Of course there's other library stuff like functions for manipulating/querying XML documents.

XML

- XML support generally quite good, but has a fair number of warts
- EPFL/TypeSafe actively looking to overhaul XML support in the language and standard libraries at some point.
  - Perhaps with the forthcoming quasi-quoting and macro systems.
- Of course there's other library stuff like functions for manipulating/querying XML documents.
- Of course you can use any other Java or Scala library to working with XML (Anti-XML is *very* good http://anti-xml.org/), but they will not *natively* work with the Scala XML types.

Parsing

- Scala library provides a rich *parser combinator library*

Parsing

- Scala library provides a rich *parser combinator library*
- Can define parsers directly in Scala without using external tools like *Antlr* http://www.antlr.org/

Parsing

Parsing Combinators

Example: *Simple parser*

```
import scala.util.parsing.combinator.syntactical._
import scala.util.parsing.combinator.{PackratParsers,RegexParsers}

class ExpressionParser extends RegexParsers with PackratParsers {
  val ws = "\\s*".r ^^ {_ => ()}
  val number = ws ~> "-?\\d+".r ^^ {x => x}
  val termOp = ws ~> "\\+|-".r ^^ {
    case "+" => "ADD"
    case "-" => "SUBTRACT"
  }
  val prodOp = ws ~> "\\*|/".r ^^ {
    case "*" => "MULT"
    case "/" => "DIVIDE"
  }
  def exp:Parser[String] = number ~ prodOp ~ exp <~ ws ^^ {case l ~ op
~ r => "%s(%s,%s)".format(op,l,r)} | number
  def term:Parser[String] = exp ~ termOp ~ term <~ ws ^^ {case l ~ op ~
r => "%s(%s,%s)".format(op,l,r)} | exp
  def test(s:String) {
    println(parseAll(term,s))
  }
}
```

Parsing: Demo

Demo Parsing Combinators

Actors

- Actors are a model concurrency based on sending messages between logical “processes”.

Actors

- Actors are a model concurrency based on sending messages between logical “processes”.
- Actors are not synonymous with *threads*. Because of this Actors are lighter-weight than threads and are often managed with a variety of execution strategies.

Actors

- Actors are a model concurrency based on sending messages between logical “processes”.
- Actors are not synonymous with *threads*. Because of this Actors are lighter-weight than threads and are often managed with a variety of execution strategies.
- Scala comes with a respectable Actor library

Actors

- Actors are a model concurrency based on sending messages between logical “processes”.
- Actors are not synonymous with *threads*. Because of this Actors are lighter-weight than threads and are often managed with a variety of execution strategies.
- Scala comes with a respectable Actor library
  - Does support local and remote actors

Actors

- Actors are a model concurrency based on sending messages between logical “processes”.
- Actors are not synonymous with *threads*. Because of this Actors are lighter-weight than threads and are often managed with a variety of execution strategies.
- Scala comes with a respectable Actor library
  - Does support local and remote actors
  - Akka provides a much more robust actor implementation.

Actors

- Actors are a model concurrency based on sending messages between logical "processes".
- Actors are not synonymous with *threads*. Because of this Actors are lighter-weight than threads and are often managed with a variety of execution strategies.
- Scala comes with a respectable Actor library
  - Does support local and remote actors
  - Akka provides a much more robust actor implementation.
  - Scalaz also has a nice (local only) actor implementation that is also type-safe.

Actors

Ping-Pong

Example: *Messages*

```
case object Ping
case object Pong
case object Stop
```

Actors

Ping-Pong

Example: *Ponger*

```
class Ponger(count:Int,pinger:Actor) extends Actor {
  def act() {
    var c = count
    var continue = true
    while (continue) {
      receive {
        case Ping =>
          println("PING: %d".format(c))
          sleep(1000)
          c -= 1
          if (c > 0) pinger ! Pong
          else {
            pinger ! Stop
            continue = false
          }

      }
    }
  }
}
```

Actors

Ping-Pong

Example: *Pinger*

```
class Pinger extends Actor {
  def act() {
    var continue = true
    while (continue) {
      receive {
        case Pong =>
          println("PONG")
          sleep(1000)
          sender ! Ping
        case Stop =>
          println("Stopping")
          continue = false
      }
    }
  }
}
```

Actors

Ping-Pong

Example: *Runner*

```
object Runner {
  def run {
    val pinger:Actor = (new Pinger()).start()
    val ponger:Actor = (new Ponger(10,pinger)).start()
    ponger ! Ping
  }
}
```

Actors: Demo

Demo Actors

Concurrency

- There are a lot of nifty low-level concurrency primitives in the standard Scala library

Concurrency

- There are a lot of nifty low-level concurrency primitives in the standard Scala library
  - Fair to say that many of them will be replaced by higher-performing versions coming from Akka

Concurrency

- There are a lot of nifty low-level concurrency primitives in the standard Scala library
  - Fair to say that many of them will be replaced by higher-performing versions coming from Akka
- The most ubiquitous concurrency mechanism in Scala is the *Future*

Concurrency: Futures

- A *Future* represents an asynchronous computation that will eventually yield a value

Concurrency: Futures

- A *Future* represents an asynchronous computation that will eventually yield a value
- *Futures* can have a variety of execution strategies

Futures

Delayed Execution

Example: *Bad way*

```
import scala.actors.Futures._
import java.lang.Thread._

object TehFutureWoooo {
  def testBad {
    val x = future {
      sleep(1000)
      5
    }
    println("Before")
    println(x())
    println("After")
  }
}
```

Futures

Monadic composition

Example: *Good way*

```
import scala.Responder
import scala.actors.{Future,Futures}, Futures._
import java.lang.Thread._

object TehFutureWoooo {
  def testGood:Responder[Int] = {
    val x = future {
      sleep(1000)
      5
    }
    println("Before")
    for (i <- x) yield {
      println("Inside: %d".format(i))
      i + 1
    }
  }
  def runner =
    for (i <- testGood) {
      println("After: %d".format(i))
    }

}
```

Futures: Demo

Demo Futures

Process Management

- Stole all the goodies from SBT

Process Management: Demo

Demo Process Management

Process Management

Shell interaction

Example:

```
import scala.sys.process.{Process,ProcessIO}
object Lister {
  val ls = Process("ls")
  val pio = new ProcessIO(_ => (),
                          stdout => scala.io.Source.fromInputStream(stdout)
                               .getLines.foreach(println),
                             _ => ())
  def runner {
    ls.run(pio)
  }
}
```

Swing

- Swing is a desktop UI framework from Java

Swing

- Swing is a desktop UI framework from Java
- Provides nice wrappers for this library

Testing

- Bare minimum testing system (much more to come)

Testing

- Bare minimum testing system (much more to come)
- Bare minimum benchmarking tool

Testing

- Bare minimum testing system (much more to come)
- Bare minimum benchmarking tool
- Several very robust Scala-oriented testing frameworks

Testing

- Bare minimum testing system (much more to come)
- Bare minimum benchmarking tool
- Several very robust Scala-oriented testing frameworks
- Can also use Java testing frameworks

Testing

Simple test

Example:

```
import scala.testing.Show

class Foo {
  def foo(i:Int):Int = i+1
}

object TestRunner extends Show {
  val f = new Foo
  def testFoo(i:Int):Int = f.foo(i)
  def tester {
    println(test('testFoo,5))
  }
}
```

Testing: Demo

Demo Testing