SCAlable LAnguage Scala

阴　钰　yxonic@gmail.com

Keywords

Keywords

- JVM

Keywords

- JVM
- Multi-paradigm

Keywords

- JVM
- Multi-paradigm
- Advanced

Keywords

- JVM
- Multi-paradigm
- Advanced
- Convenient

Keywords

- JVM
- Multi-paradigm
- Advanced
- Convenient
- Scalable

Keywords

- JVM
- Multi-paradigm
- Advanced
- Convenient
- Scalable
- Concurrent

Keywords

- JVM
- Multi-paradigm
- Advanced
- Convenient
- Scalable
- Concurrent
- BIG

Syntax Quick Guide

Identifiers and Literals

Constants

```
val msg = "Hello, world!"
lazy val words = scala.io.Source.
  fromFile("/usr/share/dict/words").mkString
```

Variables

```
var x = 5
var y: Double = 5
```

Identifiers and Literals

XML node

```
val b = <book>
  <title>The Scala Language Specification</title>
  <version>{scalaBook.version}</version>
  <authors>{scalaBook.authors.mkList("""", ", ", 
   """")}</authors>
  </book>
```

Functions

Definitions

```
def f(x: Int) = { 3*x }
type R = Double
def f(x: R) = 3
def f(x: => R) = 3
def sum(xs: Int*): Int =
if (xs.length == 0) 0 else
  xs.head + sum(xs.tail : _*)
```

Functions

Anonymous Functions

```
val f = (x: Int) => 3*x
(1 to 5).map(f)
(1 to 5).map( 3*_ )
(1 to 5).map( x => x*x )
(1 to 5).map{ x => val y = 3*x; println(y); y }
(1 to 5).reduceLeft( _+_ )
```

For and yield

For loops

```
for (i <- 1 to 5; j <- 1 to 5 if i != j)
  yield { i*10 + j }
```

Just sugar for `foreach`, `map`, `flatMap` (or known as `>>=`. Let's put a monad here!), `filter` or `withFilter`.

Pattern Matching

```
val x = r match {
  case '0' => ... // Match value
  case ch if someProperty(ch) => ... // Guard
  case e: Employee => ... // Match runtime type
  case (x, y) => ... // Extractors
  case Some(v) => ... // Case classes
  case 0 :: tail => ... // Extractors again
  case _ => ... // Default case
}
```

Classes, Objects and Traits

Classes

```scala
class Point(val x: Double, val y: Double) {
  // x, y are now public members
  def this() { this(0, 0) }
  def distance(other: Point) = {
    val dx = x - other.x; val dy = y - other.y
    Point.length(dx, dy)
  }
}
```

Classes, Objects and Traits

Companion Objects

```
object Point {
  def length(a: Double, b: Double) = math.sqrt(a * a
   + b * b)
  val origin = new Point(0, 0)
}
```

Classes, Objects and Traits

Inheritance

```
class Employee(name: String) extends Person(name) {
  var salary = 0.0
  override def toString = super.toString + "[salary="
    + salary + "]"
}
```

Classes, Objects and Traits

Traits

```
trait Logger { def log(msg: String) }
  trait ConsoleLogger extends Logger {
  override def log(msg: String) { println(msg) }
}
trait TimestampLogger extends Logger {
  override def log(msg: String) {
    super.log(new java.util.Date() + " " + msg)
  }
}
```

Classes, Objects and Traits

Traits

```
trait ShortLogger extends Logger {
  val maxLength = 15
  override def log(msg: String) {
    super.log(if (msg.length <= maxLength) msg else
    msg.substring(0, maxLength - 3) + "...")
  }
}
val acct = new SavingsAccount with ConsoleLogger with
  TimestampLogger with ShortLogger
```

Classes, Objects and Traits

Case Classes

```scala
abstract class Amount
case class Dollar(value: Double) extends Amount
case class Currency(value: Double, unit: String)
    extends Amount
case object Nothing extends Amount
sealed abstract class Option[+A] // want a monad?
case class Some[+A](v: A) extends Option[A]
case object None extends Option[Nothing]
```

Packages and Imports

Packages

```scala
package org.yxonic.utils

private[yxonic] trait Logger { ... }
```

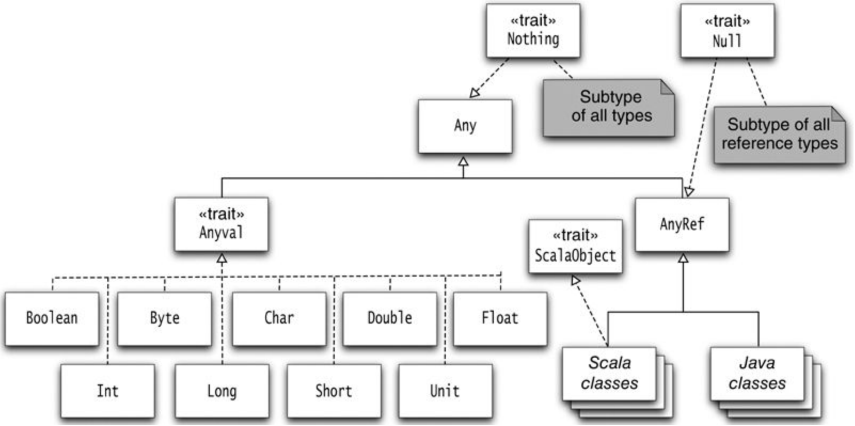
Imports

```scala
import scala.collection._

import scala.collection.{Vector => Vec, Sequence}
```

Scala Type System

Class Hierachy



Arrays and Functions

Array

```scala
val a = new Array[Int](10)
println(a(0)) // a.apply(0)
a(1) = 5 // a.update(1, 5)
```

Function

```scala
def f(x: Double) = x*x // turn into FunctionN trait
```

Type Inference

So called "Colored Local Type Inference". [1]

[1] See *Martin Odersky, Christoph Zenger, and Matthias Zenger.* Colored Local Type Inference.

Type Inference

So called "Colored Local Type Inference". [1]

Why this matters?

[1] See *Martin Odersky, Christoph Zenger, and Matthias Zenger.* Colored Local Type Inference.

Generic Classes (Type Constructors)

Generic Classes

```scala
class Pair[T, S](val first: T, val second: S)
```

Generic Functions

```scala
def getMiddle[T](a: Array[T]) = a(a.length / 2)
```

Generic Classes (Type Constructors)

Bounds （表情包 1）

```scala
class Pair[T <: Comparable[T]](val first: T, var
    second: T) {
  def smaller = if (first.compareTo(second) < 0)
    first else second
  def replaceFirst[R >: T](newFirst: R) = new
    Pair(newFirst, second)
}
```

Generic Classes (Type Constructors)

View Bounds （表情包 2）

```
class Pair[T <% Comparable[T]](val first: T,
                                var second: T) { ... }
```

Context Bounds

```
class Pair[T : Ordering](val first: T,
                          val second: T) {
  def smaller(implicit ord: Ordering[T]) { ... }
}
```

Generic Classes (Type Constructors)

Type Constraints （表情包 3）

```
class Pair[T](val first: T, val second: T) {
  def smaller(implicit ev: T <:< Ordered[T]) =
    if (first < second) first else second
}
```

and also `=:=`, `<%<`

Generic Classes (Type Constructors)

Variance

```
class A[T]  { ... }
class A[+T] { ... }
class A[-T] { ... }
```

Implicits

Implicit Conversions —

Implicits

Implicit Conversions —

- What and when?

Implicits

Implicit Conversions —

- What and when?
- Decorator pattern

Implicits

Implicit Conversions —

- What and when?
- Decorator pattern

Implicit Values

Implicits

Implicit Parameters

```
def smaller[T](a: T, b: T)(implicit ord: T =>
    Ordered[T]) =
  if (a < b) a else b

smaller(1, 2)
```

Implicits

More on Type Constraints

```scala
abstract class <:<[-From, +To] extends
   Function1[From, To]
object <:< {
  implicit def conform[A] = new (A <:< A) {
    def apply(x: A) = x
  }
}
```

Implicits

More on Type Constraints (cont.)

```
def firstLast[A, C](it: C)(implicit ev: C <:<
    Iterable[A]) =
  (it.head, it.last)
```

Advanced Types

Advanced Types

Well, I'm not talking about this today...

Above the Type System

	C++	SML	OCaml	Haskell	Java	C#	Cecil	C++0X	G	JavaGI	Scala
Multi-type concepts	-	●	○	●	●[2]	●[2]	◐	●	●	●	●[2]
Multiple constraints	-	◐	◐	●	●	●	●	●	●	●	●
Associated type access	●	●	◐	●	◐	◐	◐	●	●	◐	●[1]
Constraints on assoc. types	-	●	●	●	◐	◐	●	●	●	◐	●[1]
Retroactive modeling	-	●	●	●	◐[2]	◐[2]	●	●	●	●	●[23]
Type aliases	●	●	●	●	○	○	○	●	●	○	●
Separate compilation	○	●	●	●	●	●	◐	○	●	●	●
Implicit arg. deduction	●	○	●	●	◐[5]	◐[5]	◐	●	●	●	●[3]
Modular type checking	○	●	◐	●	●	●	◐	◐	●	◐	●
Lexically scoped models	○	●	○	○	○	○	○	○	●	○	●
Concept-based overloading	●	○	○	○	○	○	●	●	◐	○	◐[4]
Equality constraints	-	●	○	●	○	○	○	●	●	○	●
First-class functions	○	●	●	●	○	◐	●	●	◐	○	●

Figure 12. Level of support for generic programming in several languages. Key: ●='good', ◐='sufficient', ○='poor' support. The rating "-" in the C++ column indicates that while C++ does not explicitly support the feature, one can still program as if the feature were supported. Notes: 1) supported via type members and dependent method types 2) supported via the CONCEPT pattern 3) supported via implicits 4) partially supported by prioritized overlapping implicits 5) decreased score due to the use of the CONCEPT pattern

From `http://ropas.snu.ac.kr/ bruno/papers/TypeClasses.pdf`

Above the Type System

Let's get rid of sugar. The key (bitter) concepts are:

Above the Type System

Let's get rid of sugar. The key (bitter) concepts are:

- Classes and traits with inheritance and mixin

Above the Type System

Let's get rid of sugar. The key (bitter) concepts are:

- Classes and traits with inheritance and mixin
- Type constructors with bounds and variance

Above the Type System

Let's get rid of sugar. The key (bitter) concepts are:

- Classes and traits with inheritance and mixin
- Type constructors with bounds and variance
- Implicits

Discuss

Discuss

- Why this complexity?

Discuss

- Why this complexity?
  - Same concepts but more restrictions than C++

Discuss

- Why this complexity?
  - Same concepts but more restrictions than C++
  - Less mathematical than Haskell

Discuss

- Why this complexity?
  - Same concepts but more restrictions than C++
  - Less mathematical than Haskell
  - JVM limitation

Discuss

- Why this complexity?
  - Same concepts but more restrictions than C++
  - Less mathematical than Haskell
  - JVM limitation
  - Practical (?)

Discuss

- Why this complexity?
  - Same concepts but more restrictions than C++
  - Less mathematical than Haskell
  - JVM limitation
  - Practical (?)
- How is this implemented?

Thinking in Scala

Thinking in Scala

From the view of C++ and C++11:

Thinking in Scala

From the view of C++ and C++11:

- Similar but deeper thoughts on programming paradigms.

Thinking in Scala

From the view of C++ and C++11:

- Similar but deeper thoughts on programming paradigms.
- Similar but better type system.

Thinking in Scala

From the view of C++ and C++11:

- Similar but deeper thoughts on programming paradigms.
- Similar but better type system.
- Fine-grained construction and access control.

Thinking in Scala

From the view of C++ and C++11:

- Similar but deeper thoughts on programming paradigms.
- Similar but better type system.
- Fine-grained construction and access control.
- Give restrictions and clear definitions on those ambiguous parts in C++.

Thinking in Scala

From the view of Java:

Thinking in Scala

From the view of Java:

- Avoid peculiarities.

Thinking in Scala

From the view of Java:

- Avoid peculiarities.
- Brand new patterns.

Thinking in Scala

From the view of Java:

- Avoid peculiarities.
- Brand new patterns.
- Higher level abstraction.

Thinking in Scala

From the view of Java:

- Avoid peculiarities.
- Brand new patterns.
- Higher level abstraction.
- To be more expressive (maybe too deliberately).

Thinking in Scala

From the view of Haskell:

Thinking in Scala

From the view of Haskell:

- Borrow good things.

Thinking in Scala

From the view of Haskell:

- Borrow good things.
- More intuitive, less mathematical.

Thinking in Scala

From the view of Go:

[2] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

[3] http://www.scala-lang.org/docu/files/ScalaReference.pdf

[4] http://golang.org/ref/spec

Thinking in Scala

From the view of Go:

- Java 8 language spec is a 780 page PDF (ouch). [2]

[2] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

[3] http://www.scala-lang.org/docu/files/ScalaReference.pdf

[4] http://golang.org/ref/spec

Thinking in Scala

From the view of Go:

- Java 8 language spec is a 780 page PDF (ouch). [2]
- Scala language spec is a 191 page PDF. [3]

[2] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

[3] http://www.scala-lang.org/docu/files/ScalaReference.pdf

[4] http://golang.org/ref/spec

Thinking in Scala

From the view of Go:

- Java 8 language spec is a 780 page PDF (ouch). [2]
- Scala language spec is a 191 page PDF. [3]
- C# Language Specification (.docx): 511 pages.

[2] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

[3] http://www.scala-lang.org/docu/files/ScalaReference.pdf

[4] http://golang.org/ref/spec

Thinking in Scala

From the view of Go:

- Java 8 language spec is a 780 page PDF (ouch). [2]
- Scala language spec is a 191 page PDF. [3]
- C# Language Specification (.docx): 511 pages.
- Go: a webpage that prints as a 51 page PDF. [4]

[2] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

[3] http://www.scala-lang.org/docu/files/ScalaReference.pdf

[4] http://golang.org/ref/spec

Thinking in Scala

From the view of Go:

- Java 8 language spec is a 780 page PDF (ouch). [2]
- Scala language spec is a 191 page PDF. [3]
- C# Language Specification (.docx): 511 pages.
- Go: a webpage that prints as a 51 page PDF. [4]

So what are you talking about?

---

[2] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

[3] http://www.scala-lang.org/docu/files/ScalaReference.pdf

[4] http://golang.org/ref/spec

Real World Scala

Collections

- Seq, Set and Map
- Operators and methods
- Mutable and immutable collections
- Lazy views

Parallelism and Concurrency

Parallel Collections

```scala
val list = (1 to 10000).toList
list.par.map(_ + 2)
```

Parallelism and Concurrency

Example of Futures

```
val x = future { someExpensiveComputation() }
val y = future { someOtherExpensiveComputation() }
val z = for (a <- x; b <- y) yield a*b
for (c <- z) println("Result: " + c)
println("Meanwhile, the main thread goes on!")
```

On Multi-Paradigm Programming

Q&A