Scala Tutorial

Kuat Yessenov

September 21, 2010

Learning Scala

Resources on the web:

1. Download latest version (2.8.0): http://www.scala-lang.org/
2. API: http://www.scala-lang.org/api
3. Q&A: http://stackoverflow.com/

IDEs and tools:

1. IntelliJ IDEA 9 with Scala plugin: http://www.jetbrains.com/idea/
2. Emacs/Vim modes and syntax files
3. Simple Build Tool: http://code.google.com/p/simple-build-tool/
4. ScalaCheck: http://code.google.com/p/scalacheck/

Scala for Java/C# Refugees

Similarities

1. object-oriented
2. support Java style
3. compiles to Java byte code and interoperates with existing Java code

Scala for Java/C# Refugees

Differences

1. == always corresponds to `equals`
2. use **`var`** to declare mutable variables
3. use **`val`** to declare immutable variables
4. use **`def`** to declare functions
5. use **`class`** to declare a class
6. use **`object`** to decalre a companion object
7. use **`package`** to declare a module
8. default access is `public`
9. parametric types `List [T]`
10. two collection libraries: mutable and immutable; immutable by default
11. . . .

Types

1. Numeric types: Int, Long, ...
2. Other basic types: Boolean, String
3. Symbols: 'ident instances of scala.Symbol
4. Topmost types: Any is top, AnyVal, AnyRef (=Object), Unit
5. Bottom types: Null is subtype of all reference classes, Nothing is bottom
6. Traits: like interfaces but permit method bodies

Scala has a sophisticated type inference.

Functions are first-class

```
args.foreach{arg => println(arg)}
args.foreach(println _)
```
Scala function closures capture the variables themselves, not the values.

Case Classes and Pattern Matching

Sample case class

```
sealed abstract class Option[A]
final case class Some[+A] (x: A) extends Option[A]
case object None extends Option[Nothing]
```

Case Classes and Pattern Matching

Sample case class

```
sealed abstract class Option[A]
final case class Some[+A] (x: A) extends Option[A]
case object None extends Option[Nothing]
```

Scala compiler treats case classes specially

1. adds a factory method with the name of the class (Some(x))
2. arguments in the parameters list are *vals*
3. auto-generates toString , hashCode, and structural equality
4. auto-generates copy using named/default parameters

Case Classes and Pattern Matching

Sample case class

```
sealed abstract class Option[A]
final case class Some[+A] (x: A) extends Option[A]
case object None extends Option[Nothing]
```

Scala compiler treats case classes specially

1. adds a factory method with the name of the class (Some(x))
2. arguments in the parameters list are *vals*
3. auto-generates toString , hashCode, and structural equality
4. auto-generates copy using named/default parameters

Beware!

Make abstract class *sealed* and never inherit case classes from case

Implicit Definitions

Implicit definitions are methods that the compiler is allowed to insert into a program in order to fix any of its type errors:

```
def convert(a: T): U = ...
```

Implicit Definitions

Implicit definitions are methods that the compiler is allowed to insert into a program in order to fix any of its type errors:

```
def convert(a: T): U = ...
```

Rules

1. *Marking*: Only definitions marked *implicit* are available.
2. *Scope*: Implicit conversion must be in scope as a single identifier or be associated with the source or target type of the conversion.
3. *Non-ambiguity*: A conversion is inserted only if there is no other possible conversion to insert.
4. *One-at-a-time*: Only one implicit conversion is tried.
5. *Explicits-First*: Whenever code type checks as it is, no conversions are attempted.
6. *Naming*: Implicit conversion methods can have arbitrary names.

Implicit Parameters

```
def maxList[T](elements: List[T])
      (implicit orderer: T => Ordered[T]): T =
  elements match {
    case List() =>
      throw new
        IllegalArgumentException("empty list!")
    case List(x) => x
    case x :: rest =>
      val maxRest = maxList(rest)
        // (orderer) is implicit
      if (x > maxRest) x
        // orderer(x) is implicit
      else maxRest
  }
```

Internal DSL Design

Syntactic sugar

`0 to 2`	`0.to(2)`
`Console println 10`	`Console.println(10)`
`o m (p,q)`	`o.m(p,q)`
`a(i)`	`a.apply(i)`
`a(i) = j`	`a.update(i,j)`

Internal DSL Design

Syntactic sugar

`0 to 2`	`0.to(2)`
`Console println 10`	`Console.println(10)`
`o m (p,q)`	`o.m(p,q)`
`a(i)`	`a.apply(i)`
`a(i) = j`	`a.update(i,j)`

All operators are resolved to method calls

Scala decides *precedence* based on the first character (unless it ends with = and not a comparison operator.) Consistent with precedence rules for arithmetic operators.
Scala decides *associativity* based on the last character of an operator. Any method that ends in a `:` is invoked on its right operand, passing in the left operand.
Unary operators correspond to methods prefixed with `unary_`.