Introduction in Scala

Alexander Podkhalyuzin

September 6, 2012

About course

Why Scala?

- Actively developing modern language
- Language target is JVM
- Statically typed
- Supports functional style

Scala history

- Initial design started at 2001
- First compiler version in 2003
- A lot of researches was done during compiler development
- From around 2008 Scala started to come to industrial programming
- About one major release per year makes Scala more and more suitable for industrial programming
- From 2011 started active development for .Net Scala compiler

Scala future

- Scala development is much faster than Java
- Scala less verbose than Java
- So Scala is good competitor on JVM platform as new Java language
- A lot of work was done recently to make Scala less complex than many programmers think about it

Simple Scala

Hello, World!

As usual we should start from the ”Hello, World!” example:

```
object HelloWorld {
  def main(args: Array[String]) {
    println("Hello, world!")
  }
}
```

- Semicolon inference
- Singleton objects

”Complex” example

Next example describes much more about Scala:

```
object Primes extends App {
  def isPrime(n: Int) = (2 until n) forall (n % _ != 0)
  for (i <- 1 to 100 if isPrime(i)) println(i)
}
```

- Infix notation
- Higher order functions
- Placeholder closure syntax

Setting up environment

Compiler

- Download and setup JDK [1] (version 1.6 is recommended for now)
- Download and setup Scala compiler [2] (version 2.10 is actual for this course)
- Invoke Scala compiler using scalac command

Scala runtime and Scala interpreter

To run program use command scala with main class name as parameter.
To invoke Scala interpreter use command scala without additional parameters. It's very handy way to try Scala language features.

Scala editors

- Scala plugin for IntelliJ IDEA [3]
- Scala IDE (Eclipse plugin)
- Scala plugin for NetBeans
- ENSIME (for Emacs)
- Any text editor with Scala syntax highlighter (like Notepad++)

First steps in Scala

Values and variables

Syntax is following:

```
val x = 1
var y: String = "expression"
```

- Define mutablity (val is preferable)
- Define name
- Optionally define type
- Right side is an expression

Function definition

It’s very similar:

```
def inc(n: Int): Int = n + 1
def printInc(n: Int) {
  print(inc(n))
}
```

- Use ’def’ to define function
- Define function name
- Define parameter list
- Optionally define return type

If statement

If syntax is the following:

```
if (condition) expression1 [else expression2]
```

Difference to other languages that in Scala every expression has type. Without else branch it's Unit type, otherwise it's lub of two expression types.
So it's like ternary operator in Java or C++.

While loop

While and Do syntax is the following:

```
while (condition) expression
do expression while (condition)
```

Type of these expressions is always Unit.

For statement

For statement is complex syntactic sugar. Let's take a short look on it:

```
val array = Array(1, 2, 3)
array.foreach((x: Int) => println(x + 1))
array.foreach(x => println(x + 1))
array.foreach(println(_ + 1))
for (x <- array) println(x + 1)
```

All of this is the same. Type of for statement is the same as type of all foreach calls, i.e. Unit.

Arrays


```
Array.empty[Int]
Array.ofDim[Int](2)
new Array[Int](2)
Array(1, 2, 3)
val a = Array.fill(1, 2)(0)

a.apply(0).apply(1)
a(0)(1)
a.apply(0).update(1, 2)
a(0)(1) = 2
```

Important thing is that Arrays are the same like Lists or other collection objects. To iterate over Arrays

```
for (i <- 1 to a.length) print(a(i))
for (arr <- a; elem <- arr) print(arr)
```

Lists


```
List.empty[Int]
List(1, 2, 3)
val a = 1 :: 2 :: 3 :: Nil

1 :: a
a.updated(1, 1)
a ::: a
```

Iterating over lists is the same as over arrays.

Tuples

Simple language feature to join few objects together:

```
def foo(x: Int): (Int, Int) = (x, x + 1)
val (a, b) = foo(1)
(a, b) match {case (x, y) if y - x == 1 => }
val first = foo(2)._1
```

Mostly useful for:

- Multiple return
- Multiple assign
- Pattern matching over multiple values

Access to tuple elements through _n methods.

(Int, Int) is syntactic sugar for Tuple2[Int, Int]

Additionally Scala has arrow syntax (1 –>”one”)

References

References

1. http://www.oracle.com/technetwork/java/javase/downloads/index.html
2. http://www.scala-lang.org/downloads
3. http://plugins.intellij.net/plugin/?ideapluginId=1347

Questions?