写给Python程序员的Scala入门教程

随着业务和数据的需要，我们引入了`Spark`。`Spark`对`Python`的支持还是挺好的，但毕竟它还是使用`Scala`开发的，且现有的API并没有100%覆盖Python。所以就有了这篇文章，让Python程序员可以接触`Scala`这门更高（级）、更快（速）、更强（大）的（奥运精神）语言。

Scala兼具Python样的开发效率，但又有Java般的执行性能，真是一不可多得的神器！（当然，鱼和熊不可兼得，Scala的入门曲线相比Python是要那么陡峭一丢丢）

安装

一般Linux系统都自带`Python`环境，但Scala是没有的。这需要我们手动安装，还需要安装Java环境。Java环境的安装这里就不介绍了，网上很多。说说Scala的安装吧。下载地址在http://scala-lang.org/download/2.11.7.html。

```
wget -c http://downloads.typesafe.com/scala/2.11.7/scala-2.11.7.tgz
tar zxf scala-2.11.7
cd scala-2.11.7
./bin/scala
Welcome to Scala version 2.11.7 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_60).
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

我们可以看到终端出现了`scala>`提示符，这个就像`Python`的`REPL`一样，Scala也拥有一个`REPL`。我们可以在这里很方便的编写一些代码，测试一些想法。

对于Scala常用的IDE（集成开发环境），推荐使用IDEA for scala plugins和scala-ide。

Scala的强大，除了它自身对多核编程更好的支持、函数式特性及一些基于Scala的第3方库和框架（如：Akka、Playframework、Spark、Kafka……），还在于它可以无缝与Java结合。所有为Java开发的库、框架都可以自然的融入Scala环境。当然，Scala也可以很方便的Java环境集成，比如：**Spring**。若你需要第3方库的支持，可以使用**Maven**、**Gradle**、**Sbt**等编译环境来引入。

Scala是一个面向对象的函数式特性编程语言，它继承了Java的面向对特性，同时又从`Haskell`那里吸收了很多函数式特性并做了增强。

Hello, world.

通常情况下，Java系的程序都需要有一个main方法来执行。并需要放入一个`Web container`容器，或打成一个`jar`包来执行。但是Scala不一样，它除了支持传统的Java方式，它也可以像Python一样把代码保存到一个脚本文件里（.scala）执行，就像一个**Shell**脚本一样。

```
yangjing-mac-air:scala-2.11.7 jingyang$ cat test.scala
#!/bin/sh
exec scala "$0" "$@"
!#
// Say hello to the first argument
println("Hello, "+ args(0) +"!")
yangjing-mac-air:scala-2.11.7 jingyang$ ./test.scala 杨景
Hello, 杨景!
```

（注：需要把`$SCALA_HOME/bin`加入系统环境变量才能直接执行`scala`命令）

可以看到，使用Scala，你除了像传统的Java程序一样把它做为一个“服务”的方式来启动，你也可以把它像Python一样做为一个“脚本”来启动。

（注意：Scala不像Python一样通过代码缩进来表示代码的层次关系，而是和通常的语言一样使用`{}`来表示代码的层级。给程序员更多的自由）

变量、基础数据类型

Scala中变量不需要显示指定类型，但需要提前声明。这可以避免很多命名空间污染问题。Scala有一个很强大的类型自动推导功能，它可以根据右值及上下文自动推导出变量的类型。你可以通过如下方式来直接声明并赋值。

```
scala> val a = 1
a: Int = 1

scala> val b = true
b: Boolean = true

scala> val c = 1.0
c: Double = 1.0

scala> val a = 30 + "岁"
a: String = 30岁
```

Immutable

（注：函数式编程有一个很重要的特性：不可变性。Scala中除了变量的不可变性，它还定义了一套不可变集合`scala.collection.immutable._`。）

`val`代表这是一个final variable，它是一个常量。定义后就不可以改变，相应的，使用`var`定义的就是平常所见的变量了，是可以改变的。从终端的打印可以看出，Scala从右值自动推导出了变量的类型。Scala可以如动态语言似的编写代码，但又有静态语言的编译时检查，不会像Python一样留下很多陷阱在运行多时以后才被发现。

（注：在`RELP`中，`val`变量是可以重新赋值的，这是 ` RELP`的特性。在平常的代码中是不可以的。）

基础数据类型

Scala中基础数据类型有：Byte、Short、Int、Long、Float、Double，Boolean，Char、String。和Java不同的是，Scala中没在区分原生类型和装箱类型，如：`int`和`Integer`。它统一抽象成`Int`类型，这样在Scala中所有类型都是对象了。编译器在编译时将自动决定使用原生类型还是装箱类型。

字符串

Scala中单引号和双引号包裹是有区别的，单引号用于字符，双引号用于字符串。

```
scala> val c1 = 'c'
c1: Char = c

scala> val 字符2 = '杨'
字符2: Char = 杨

scala> val s1 = "杭州誉存科技有限公司"
s1: String = 杭州誉存科技有限公司

scala> val s2 = s"杭州誉存科技有限公司工程师${c2}景"
s2: String = 杭州誉存科技有限公司工程师杨景

scala> val s3 = s"""杭州誉存科技有限公司"工程师"\n${c2}景是江津人"""
s3: String =
杭州誉存科技有限公司"工程师"
杨景是江津人
```

Scala基于JVM平台，默认使用unicode，所以变量名是可以直接用中文的。而在Scala中，中文也是直接显示的，不像Python2一样会输出成unicdoe编码形式：\uxxxx。

Scala还支持**String Interpolation**（“字符串插值”）的特性，可以使用`${variable name}`这样的形式引用变量，并将值插入。就像示例`s2`一样。

而连线3个双引号在Scala中也有特殊含义，它代表被包裹的内容是原始字符串，可以不需要字符转码。这一特性在定义正则表达式时很有优势。

运算符

Scala中的运算符其实是定义在对象上的方法（函数），你看到的诸如：`3 + 2`其实是这样子的：`3.+(2)`。`+`符号是定义在`Int`对象上的一个方法。支持和Java一至的运算符（方法）：

（注：在Scala中，方法前的`.`号和方法两边的小括号在不引起歧义的情况下是可以省略的。这样我们就可以定义出很优美的`DSL`）

- `==`、`!=`：比较运算
- `!`、`|`、`&`、`^`：逻辑运算
- `>>`、`<<`：位运算

在Scala中，修正（算更符合一般人的常规理解吧）`==`和`!=`运算符的含义。在Scala中，`==`和`!=`是执行对象的值比较，相当于Java中的`equals`方法（实际上编译器在编译时也是这么做的）。而对象比较需要使用`eq`和`ne`两个方法来实现。

控制语句

Scala中支持`if`、`while`、`for comprehension`（for表达式)、`match case`（模式匹配）四大主要控制语句。Scala不支持`switch`和`? :`两种控制语句，但它的`if`和`match case`会有更好的实现。

`if`

Scala支持`if`语句，其基本使用和`Java`、`Python`中的一样。但不同的时，它是有返回值的。

（注：Scala是函数式语言，函数式语言还有一大特性就是：表达式。函数式语言中所有语句都是基于“表达式”的，而“表达式”的一个特性就是它会有一个值。所有像`Java`中的`? :`3目运算符可以使用`if`语句来代替）。

```
scala> if (true) "真" else "假"
res0: String = 真

scala> val f = if (false) "真" else "假"
f: String = 假

scala> val unit = if (false) "真"
unit: Any = ()

scala> val unit2 = if (true) "真"
unit2: Any = 真
```

可以看到，`if`语句也是有返回值的，将表达式的结果赋给变量，编译器也能正常推导出变量的类型。`unit`和`unit2`变量的类型是`Any`，这是因为`else`语句的缺失，Scala编译器就按最大化类型来推导，而`Any`类型是Scala中的根类型。`()`在Scala中是`Unit`类型的实例，可以看做是`Java`中的`Void`。

`while`

Scala中的`while`循环语句：

```
while (条件) {
  语句块
}
```

`for comprehension`

Scala中也有`for`表达式，但它和`Java`以及`Python`中的`for`不太一样，它具有更强大的特性。通常的`for`语句如下：

```
for (变量 <- 集合) {
  语句块
}
```

Scala中`for`表达式除了上面那样的常规用法，它还可以使用`yield`关键字将集合映射为另一个集合：

```
scala> val list = List(1, 2, 3, 4, 5)
list: List[Int] = List(1, 2, 3, 4, 5)

scala> val list2 = for (item <- list) yield item + 1
list2: List[Int] = List(2, 3, 4, 5, 6)
```

还可以在表达式中使用`if`判断：

```
scala> val list3 = for (item <- list if item % 2 == 0) yield item
list3: List[Int] = List(2, 4)
```

还可以做`flatMap`操作，解析2维列表并将结果摊平（将2维列表拉平为一维列表）：

```
scala> val llist = List(List(1, 2, 3), List(4, 5, 6), List(7, 8, 9))
llist: List[List[Int]] = List(List(1, 2, 3), List(4, 5, 6), List(7, 8, 9))

scala> for {
     |   l <- llist
     |   item <- l if item % 2 == 0
     | } yield item
res3: List[Int] = List(2, 4, 6, 8)
```

看到了，Scala中`for comprehension`的特性是很强大的。它和`Python`中的`list comprehension`很类似，但不同的是在Scala中这一特性并不只限于`list`中，而是整个集合库都支持这一特性，包括：`Seq`、`Map`、`Set`、`Array`……

Scala和Python一样，也没有C-Like语言里的`for (int i = 0; i < 10; i++)`语法，但和Python类似的它也有`xrange`或`range`函数样的效果。在Scala中的使用方式如下：

```
scala> for (i <- (0 until 10)) {
     |   println(i)
     | }
0
1
2
3
4
5
6
7
8
9
```

比Python更好的一点时，Scala中还有一个`to`方法（Scala中去处符其实都是定义在对象上的方法/函数）：

```
scala> for (i <- (0 to 10)) print(" " + i)
 0 1 2 3 4 5 6 7 8 9 10
```

match case

模式匹配，是函数式语言很强大的一个特性。它比命令式语言里的`switch`更好用，表达性更强。

```
scala> def level(s: Int) = s match {
     |   case n if n >= 90 => "优秀"
     |   case n if n >= 80 => "良好"
     |   case n if n >= 70 => "良"
     |   case n if n >= 60 => "及格"
     |   case _ => "差"
     | }
level: (s: Int)String

scala> level(51)
res28: String = 差

scala> level(93)
res29: String = 优秀

scala> level(80)
res30: String = 良好
```

可以看到，模式匹配可以使用`switch`相同的功能。但也`switch`需要使用`break`明确告知终止之后的判断不同，Scala中的`match case`是默认**break**的，只要其中一个`case`语句匹配，就终止之后的所以比较。且对应`case`语句的表达式值将作为整个`match case`表达式的值返回。

Scala中的模式匹配还有类型匹配、数据抽取、谓词判断等其它有用的功能。这里只做简单介绍，之后会单独一个章节来做较详细的解读。

集合

在Python中，常用的集合类型有：`list`、`tuple`、`set`、`dict`。Scala中对应的有：`List`、`Tuple[X]`、`Set`、`Map`。

List

Scala中`List`是一个递归不可变集合，它很精妙的使用递归结构定义了一个列表集合。除了之前使用`List`object来定义一个列表，还可以使用如下方式：

```
scala> val list = 1 :: 2 :: 3 :: 4 :: 5 :: Nil
list: List[Int] = List(1, 2, 3, 4, 5)
```

`List`采用前缀操作的方式（所有操作都在列表顶端（开头））进行，`::`操作符的作用是将一个元素和列表连接起来，并把元素放在列表的开头。这样`List`的操作就可以定义成一个递归操作。添加一个元素就是把元素加到列表的开头，List只需要更改下头指针，而删除一个元素就是把List的头指针指向列表中的第2个元素。这样，`List`的实现就非常的高效，它也不需要对内存做任何的转移操作。`List`有很多常用的方法：

```
scala> list.indexOf(3)
res6: Int = 2

scala> 0 :: list
res8: List[Int] = List(0, 1, 2, 3, 4, 5)

scala> list.reverse
res9: List[Int] = List(5, 4, 3, 2, 1)

scala> list.filter(item => item == 3)
res11: List[Int] = List(3)

scala> list
res12: List[Int] = List(1, 2, 3, 4, 5)

scala> val list2 = List(4, 5, 6, 7, 8, 9)
list2: List[Int] = List(4, 5, 6, 7, 8, 9)

scala> list.intersect(list2)
res13: List[Int] = List(4, 5)

scala> list.union(list2)
res14: List[Int] = List(1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9)

scala> list.diff(list2)
res15: List[Int] = List(1, 2, 3)

```

Scala中默认都是**Immutable collection**，在集合上定义的操作都不会更改集合本身，而是生成一个新的集合。Python中只有`set`上有求交、并、差积运算，Scala中将其范化到所以序列集合上（`Seq`、`List`、`Set`、`Array`……）都可以支持。

Tuple

Scala中也支持**Tuple**（元组）这种集合，但最多只支持22个元素（事实上Scala中定义了`Tuple0`、`Tuple1`……`Tuple22`这样22个`TupleX`类，实现方式与`C++ Boost`库中的`Tuple`类似）。和Python中类似，Scala也采用小括号来定义元组。

```
scala> val tuple1 = (1, 2, 3)
tuple1: (Int, Int, Int) = (1,2,3)

scala> tuple1._2
res17: Int = 2

scala> val tuple2 = Tuple2("杨", " )
tuple2: (String, String) = (杨,景)
```

可以使用`xxx._[X]`的形式来引用`Tuple`中某一个具体元素，其`_[X]`下标是从1开始的，一直到22（若有定义这么多）。

Set

`Set`是一个不重复且无序的集合，初始化一个`Set`需要使用`Set`对象：

```
scala> val set = Set("Python", "Scala", "Java", "C++", "Javascript", "C#", "PHP")
set: scala.collection.immutable.Set[String] = Set(Scala, C#, Python, Javascript, PHP, C++, Java)

scala> set + "Go"
res21: scala.collection.immutable.Set[String] = Set(Scala, C#, Go, Python, Javascript, PHP, C++, Java)

scala> set filterNot (item => item == "PHP")
res22: scala.collection.immutable.Set[String] = Set(Scala, C#, Python, Javascript, C++, Java)
```

Map

Scala中的`Map`是一个**HashMap**，其`key`也是无序的。Python中的`dict`对应些类型（可以使用`TreeMap`来让Map有序）。与Python中不一样，Scala并没有提供一个`{}`来定义`Map`，它还是很统一的采用相关类型的object来定义：

```
scala> val map = Map("a" -> "A", "b" -> "B")
map: scala.collection.immutable.Map[String,String] = Map(a -> A, b -> B)

scala> val map2 = Map(("b", "B"), ("c", "C"))
map2: scala.collection.immutable.Map[String,String] = Map(b -> B, c -> C)
```

Scala中定义`Map`时，传入的每个`Entry`（**K**、**V**对）其实就是一个`Tuple2`（有两个元素的元组），而`->`是定义`Tuple2`的一种便捷方式。

```
scala> map + ("z" -> "Z")
res23: scala.collection.immutable.Map[String,String] = Map(a -> A, b -> B, z -> Z)

scala> map.filterNot(entry => entry._1 == "a")
res24: scala.collection.immutable.Map[String,String] = Map(b -> B)

scala> map
res25: scala.collection.immutable.Map[String,String] = Map(a -> A, b -> B)
```

Scala的immutable collection并没有添加和删除元素的操作，其定义`+`（`List`使用`::`在头部添加）操作都是生成一个新的集合，而要删除一个元素一般使用`.filterNot`函数来映射一个新的集合实现。

（注：Scala中也`scala.collection.mutable._`集合，它定义了不可变集合的相应可变集合版本。一般情况下，除非一性性能优先的操作（其实Scala集合采用了共享变量的优化，生成一个新集合并不会生成所有元素的副本，它将会和老的集合共享大元素。因为Scala中变量默认都是不可变的），推荐还是采用不可变集合。因为它更直观、线程安全，你可以确定你的变量不会在其它地方被不小心的更改。）

函数（初级）

在Scala中，函数是一等公民。函数可以像类型一样被赋值给一个变量，也可以做为一个函数的参数被传入，甚至还可以做为函数的返回值返回（这就是函数式编程）。

他Python一样，Scala也使用`def`关键词来定义一个函数：

```
scala> def calc(n1: Int, n2: Int): (Int, Int) = {
     |   (n1 + n2, n1 * n2)
     | }
calc: (n1: Int, n2: Int)(Int, Int)

scala> val (add, sub) = calc(5, 1)
add: Int = 6
sub: Int = 5
```

这里定义了一个函数：`calc`，它有两个参数：`n1`和`n2`，其类型为：`Int`。`cala`函数的返回值类型是一个有两个元素的元组，在Scala中可以简写为：`(Int, Int)`。在Scala中，代码段的最后一句将做为函数返回值，所以这里不需要显示的写`return`关键字。

而`val (add, sub) = calc(5, 1)`一句，是Scala中的抽取功能。它直接把`calc`函数返回的一个`Tuple2`值赋给了`add`他`sub`两个变量。

总结

本篇文章简单的介绍了Scala的语言特性，本文并不只限于Python程序员，任何有编程经验的程序员都可以看。现在你应该对Scala有了一个基础的认识，并可以写一些简单的代码了。之后会分享一些《Scala实战（系列）》，介绍更**函数式**的写法及与实际工程中结合的例子。

#python #scala

分享到