**Scala光速入门**

杭州-Frank

天下武功，唯快不破！精通Spark，从Scala开始！

# Scala的重大价值

曾经有人问Java之父，“除了Java语言以外，您现在还使用JVM平台上的哪种编程语言?”**他毫不犹豫地说是Scala**！

Spark也是基于JVM，我们构筑分布式系统，借助JVM，而不一定是Java语言。 Spark和消息中间件KAFKA等都是用Scala编写的，学好Scala是掌握Spark的关键。

Scala和Java的关系：

* Scala和Java都是基于JVM的，Scala可以调用Java的任何功能。Spark运行在Hadoop上，Spark可以调用Hadoop的所有功能。
* 可以认为Scala就是升级版本的Java！有以下几个显著特性：

Scala是纯面向对象的语言

Scala是面向对象和函数式结合的语言。函数式编程用于实现具体的方法和功能，实现比Java更加简洁优雅，代码量只是Java的 1/5 ~ 1/10 。

# Scala基础语法入门实战

首先，参照相关攻略，在Linux下分别下载安装Java、Scala，然后配置Java和Scala环境变量。安装完毕，在终端敲入scala即可进入Scala命令行，如下所示：

root@Master:~# scala

Welcome to Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0\_66).

Type in expressions to have them evaluated.

Type :help for more information.

## 简单测试

scala> 1+2

res0: Int = 3

scala> 1.5\*2

res1: Double = 3.0

scala> 3\*res1

res2: Double = 9.0

**//按Tab键，命令自动补全**

scala> res2.to

toByte toChar toDouble toFloat toInt toLong toShort toString

## 变量

var声明可变变量；val声明不可变变量。

val声明的不可变变量，不希望数据被改变，RDD内部的数据都是不可变，所以在Spark中一般都是使用val。

**//下面声明了一个不可变变量result，result的值不可改变。**

scala> val result=2+10

result: Int = 12

**//假若修改result的值，会提示出错，如下：**

scala> result=13

<console>:8: error: reassignment to val

result=13

**//var声明可变变量：**

scala> var name="Spark"

name: String = Spark

scala> name="Scala" //可以修改变量name的值

name: String = Scala

scala> name //name的值现在为Scala

res4: String = Scala

**//val声明不可变变量**

//age被声明为Int类型的不可变变量

scala> val age: Int=0

age: Int = 0

//声明为String类型

scala> val name:String=null

name: String = null

## 一行代码声明多个变量

scala> val age1,age2,age3=0

age1: Int = 0

age2: Int = 0

age3: Int = 0

## 基本数据类型的自动转换操作

Scala自己可以完成基本数据类型的自动转换操作。

//输入整数10，按Tab键盘，可以看见它本身的很多方法：

scala> 10.to

toByte toChar toDouble toFloat toInt toLong toShort toString

scala> 10.toString

res5: String = 10

scala> 0.to(5)

res6: scala.collection.immutable.Range.Inclusive = Range(0, 1, 2, 3, 4, 5)

给了我们一个不可变的集合，获得的结果是：0~5，所以Scala一切皆对象！

## Scala隐私转换：

当类型本身没有没有这个方法，但是又需要调用这个方法时，内部就会自动触发隐式转换。刚才的示例中，Int本身没有to这样的方法，Scala引擎内部会隐式自动转换成RichInt，就像上面0.to(5)这样的调用，RichInt对象类型才有to这样的函数。

scala> 1+1

res7: Int = 2

//因为Scala一切皆对象，所以以上示例又可以写成：

scala> 1.+(1)

res9: Double = 2.0

这个示例，内部其实就是Int的一个隐式转换，+是类的一个方法。

## Scala没有++、---操作运算：

scala> var age=10

age: Int = 10

// **Scala没有++、---操作运算**

scala> age++

<console>:9: error: value ++ is not a member of Int

age++

^

//但是++运算可以用下面方式实现：

scala> age +=1

scala> age

res12: Int = 11

## 求最大、最小值

scala> min(20,4)

<console>:8: error: not found: value min

min(20,4)

^

该示例因为没有导入库，所以出错。

scala> import scala.math.\_ //导入math库

import scala.math.\_

scala> min(20,4)

res14: Int = 4

## apply工厂构造实现方法

在Spark中，大量的实例的构造都是使用了apply方式。

scala> Array(1,2,3,4)

res15: Array[Int] = Array(1, 2, 3, 4)

scala> val array=Array(1,2,3,4)

array: Array[Int] = Array(1, 2, 3, 4)

//array是一个声明整数类型的数组变量， 其实内部是自动调用了Array.apply方法，等同如下：

scala> val array = Array.apply(1,2,3,4)

array: Array[Int] = Array(1, 2, 3, 4)

## 条件控制、循环

**// if表达式示例：**

scala> if(age>=18) "成年人" else "小孩"

res16: String = 成年人

scala> val result=if(age>=18) "成年人" else "小孩"

result: String = 成年人

scala> result

res17: String = 成年人

scala> val result = if(age>=18){

| "adult"

| buffered=10

| buffered

| }

以上一个代码块，代码块后面有个返回值buffered，代码块的返回值就是最后一行的值。

## 打印值

scala> println("Spark") //输出一行字符串并换行

Spark

scala> println("\nSpark") //换行，输出一行字符串再换行。\n是换行转义符。

Spark

scala> print("Spark") //输出一行字符串，不换行

Spark

scala>

## 填充占位符

scala> printf("%s是大数据框架的未来", "Spark") //%s是占位符

Spark是大数据框架的未来

## 读取内容

readLine用于读取输入的内容

scala> readLine //此时敲入Scala之后，然后回车

res28: String = Scala

scala> res28

res29: String = Scala

补充说明，readLine是一个方法，如果方法如果没有参数，那么可以不带括号，readLine()跟readLine效果一样。

## 循环

**//声明一个可变变量，初始值为100**

scala> var element=100

element: Int = 100

**//while循环示例：**

scala> while(element>90){

| println(element)

| element -= 1

| }

100

99

98

97

96

95

94

93

92

91

scala> 0 to element

res32: scala.collection.immutable.Range.Inclusive = Range(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90)

**//for循环遍历并打印**

scala> for(i<-80 to element) println(i)

80

81

82

83

84

85

86

87

88

89

90

**//循环并增加条件判断**

scala> for(i<-0 to element if i%2==0) print(i+" ")

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90

**//for循环，并break退出**

scala> import scala.util.control.Breaks.\_ //添加break引用

import scala.util.control.Breaks.\_

scala> for(i<-1 to 10){

| if(i==4) break

| println(i)

| }

1

2

3

scala.util.control.BreakControl

**//循环，并return**

scala> val n=5

n: Int = 5

scala> def f1:Any = {

| for(i <-1 to 10){

| if(i==n) return i

| println(i)

| }

| }

f1: Any

scala> f1

1

2

3

4

res1: Any = 5

解释以上代码块，def是定义一个函数。f1就是一个函数。

# Scala函数入门实战

## 函数示例

函数的定义使用def关键字，并且函数体最后有返回值。

//声明了一个函数f3，两个参数：param1是String类型，param2为Int类型。param2默认值为30，因为有了默认值，那么在调用上可以不传param2。

scala> def f3(param1:String, param2:Int=30) = param1 + param2

f3: (param1: String, param2: Int)String

//调用函数f3,传入第一个参数param1值为Spark，没有传入第二个参数，默认为30。

scala> f3("Spark")

res4: String = Spark30

//带名参数调用，即在函数调用时，显示指定参数名，并不按顺序传入。

scala> f3(param2=100, param1="Scala")

res5: String = Scala100

//变长参数, 定义了一个sum函数，参数numbers是变成参数，即传入的Int变量个数不定。在函数体中，对传入的全部Int变量进行循环遍历并累计求和，最后把结果返回。

scala> def sum(numbers: Int\*)={var result=0; for(element<-numbers) result +=element; result}

sum: (numbers: Int\*)Int

scala> sum(1,2,3,4,5,6,7,8,9,10)

res1: Int = 55

//下面示例是一个非常经典的语法

scala> sum(1 to 10: \_\*) // \_\* 表示提取里面的每个元素, 然后作为变长参数传递

res3: Int = 55

## 过程

没有返回值的函数就是过程。

//这是一个函数

scala> def morning(content:String) = "Good" + content

morning: (content: String)String

//这是一个过程

scala> def morning(content:String) { println( "Good" + content)}

morning: (content: String)Unit

//强制声明一个过程

scala> def morning(content:String):Unit = "Good" + content

morning: (content: String)Unit

## 声明lazy类型

scala> import scala.io.Source.\_ //导入引用库

import scala.io.Source.\_

//声明一个lazy类型的变量content，打开一个不存在的文件。

scala> lazy val content = fromFile("/root/txt")

content: scala.io.BufferedSource = <lazy>

以上示例执行不会出错，表明content变量并没有执行。

//如果去掉lazy关键字，那么会出错，提示文件不存在。

scala> val content = fromFile("/root/txt")

java.io.FileNotFoundException: /root/txt (No such file or directory)

at java.io.FileInputStream.open0(Native Method)

at java.io.FileInputStream.open(FileInputStream.java:195)

at java.io.FileInputStream.<init>(FileInputStream.java:138)

**耗时的操作，在大型分布式系统中，比较常见。声明lazy类型的变量，在需要的时候才去执行。**

## 异常

//首先导入相关引用包

scala> import java.io.\_

import java.io.\_

scala>

//示例打开一个存在的文件，使用try…catch捕获异常

scala> try{

| val content = fromFile("/root/.bashrc").mkString

| }catch{

| //case是一个偏函数

| case \_: FileNotFoundException => println("Oh, file not found!")

| }finally{

| println("Ok!")

| }

Ok!

scala>

//示例打开一个不存在的文件，使用try…catch捕获异常

scala> try{

| val content = fromFile("/root/.bashrc111").mkString

| }catch{

| //case是一个偏函数

| case \_: FileNotFoundException => println("Oh, file not found!")

| }finally{

| println("Ok!")

| }

Oh, file not found!

Ok!

# Scala中Array、Map、Tuple实战

## Array

**//声明变量arr为Array整数类型的数组，包含5个元素。**

scala> val arr= new Array[Int](5)

arr: Array[Int] = Array(0, 0, 0, 0, 0)

//访问第三个元素

scala> arr(2)

res15: Int = 0

//修改第三个元素

scala> arr(2)=8

//再次查看arr数组，发现第三个元素值已经变成8了。

scala> arr

res17: Array[Int] = Array(0, 0, 8, 0, 0)

**补充说明，刚才声明arr数组变量时，所以把它声明为val不可变变量，这只是表明arr的地址不可以变，但是数组里面的元素还是可以变化的。**

**//**在Spark中，更常见地创建数组是直接通过类名

scala> val arr1 = Array("Scala", "Spark")

arr1: Array[String] = Array(Scala, Spark)

该示例中，声明arr1为数组变量时，没有使用new关键字，也没有指定String类型，系统默认根据元素值，自动推导出元素的类型为String。

没有使用new关键字，其实它内部调用了apply方法， apply是工厂类构造器。等同于下面的写法：

scala> val arr1 = Array.apply("Scala", "Spark")

arr1: Array[String] = Array(Scala, Spark)

//给Array增加元素。下面写法会出错，给arr1数组增加一个元素，比如：

scala> arr1(2)="Hadoop"

java.lang.ArrayIndexOutOfBoundsException: 2

at .<init>(<console>:16)

at .<clinit>(<console>)

……

如果需要给Array增加元素，那么此时就应该使用ArrayBuffer类型。

## ArrayBuffer

//首先导入库

scala> import scala.collection.mutable.ArrayBuffer

import scala.collection.mutable.ArrayBuffer

//定义一个ArrayBuffer类型的变量arrbuffer

scala> val arrbuffer=ArrayBuffer[Int]()

arrbuffer: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer()

 //向arrbuffer中增加一个元素，值为10

scala> arrbuffer += 10

res23: arrbuffer.type = ArrayBuffer(10)

 //向arrbuffer中增加多个元素

scala> arrbuffer += (11,1,3,5)

res25: arrbuffer.type = ArrayBuffer(10, 11, 1, 3, 5)

//查看arrbuffer的内容

scala> arrbuffer

res26: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(10, 11, 1, 3, 5)

 //向arrbuffer中增加一个数组

scala> arrbuffer ++= Array(1,2,3,4)

res27: arrbuffer.type = ArrayBuffer(10, 11, 1, 3, 5, 1, 2, 3, 4)

//截掉arrbuffer后面的3个元素

scala> arrbuffer.trimEnd(3)

//再次查看arrbuffer的内容，发现元素：2, 3, 4被截掉

scala> arrbuffer

res29: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(10, 11, 1, 3, 5, 1)

//在第5个位置，插入元素值100

scala> arrbuffer.insert(5,100)

//查看arrbuffer的内容

scala> arrbuffer

res32: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(10, 11, 1, 3, 5, 100, 1)

//在第2个位置，插入多个元素：200,300,400

scala> arrbuffer.insert(2,200,300,400)

//查看arrbuffer的内容

scala> arrbuffer

res34: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(10, 11, 200, 300, 400, 1, 3, 5, 100, 1)

//从arrbuffer中移除第3个位置上的元素

scala> arrbuffer.remove(3)

res35: Int = 300 //被移除的值是300

//再次查看arrbuffer的内容，发现第3个位置上的元素300不见了。

scala> arrbuffer

res36: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(10, 11, 200, 400, 1, 3, 5, 100, 1)

//从arrbuffer中移除第2个位置开始的，3个元素，即：200, 400, 1

scala> arrbuffer.remove(2,3)

//再次查看arrbuffer的内容，发现三个元素：200, 400, 1 不见了。

scala> arrbuffer

res38: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(10, 11, 3, 5, 100, 1)

//可变数组变成不可变数组,此时arr2是一个不可变数组

scala> val arr2 = arrbuffer.toArray

arr2: Array[Int] = Array(10, 11, 3, 5, 100, 1)

// Array.toBuffer的结果变成一个ArrayBuffer

scala> arr2.toBuffer

res40: scala.collection.mutable.Buffer[Int] = ArrayBuffer(10, 11, 3, 5, 100, 1)

//遍历一个数组：

scala> for(elem <- arr2) println(elem)

10

11

3

5

100

1

//遍历数组时加上条件

scala> arr2

res42: Array[Int] = Array(10, 11, 3, 5, 100, 1)

//遍历时的条件，跳过偶数位上的元素

scala> for(i <- 0 until (arr2.length, 2)) println(arr2(i))

10

3

100

此时打印出来的结果，跳过了元素：11、5、1

//从尾部开始遍历

scala> for(i <- (0 until arr2.length).reverse) println(arr2(i))

1

100

5

3

11

10

**//对数组进行排序**

//导入排序包

scala> import scala.util.Sorting

import scala.util.Sorting

//排序之前

scala> arr2

res42: Array[Int] = Array(10, 11, 3, 5, 100, 1)

//对arr2进行排序

scala> Sorting.quickSort(arr2)

//排序之后

scala> arr2

res49: Array[Int] = Array(1, 3, 5, 10, 11, 100)

//显示arr2中内容

scala> arr2

res49: Array[Int] = Array(1, 3, 5, 10, 11, 100)

//拼接 arr2中的每个元素，用逗号拼接，生成一个字符串

scala> arr2.mkString(",")

res50: String = 1,3,5,10,11,100

//循环arr2里面的每个元素，对其进行自乘运算，并把结果收集起来，产生一个新的数组，赋给arr3

scala> val arr3 = for(i <- arr2) yield i\*i

arr3: Array[Int] = Array(1, 9, 25, 100, 121, 10000)

// for循环再加上if条件判断，仍然是循环arr2里面的每个元素，对其进行自乘运算，并把结果收集起来，产生一个新的数组，赋给arr3

scala> val arr3 = for(i <- arr2 if i%3==0) yield i\*i

arr3: Array[Int] = Array(9)

此时arr3中只有一个元素。

//在scala实际会用它的函数式编程来实现以上代码

scala> arr2.filter(\_%3 ==0).map(i => i\*i)

res61: Array[Int] = Array(9)

这种写法非常优雅简洁，生成的结果跟上面一样。map本身是一个函数，里面传入的参数仍然是一个函数。

//上一行示例中的括号，其实可以改成花括号。

scala> arr2.filter{\_%3 ==0}.map{i => i\*i}

res62: Array[Int] = Array(9)

//甚至可以省略map前面的点号

scala> arr2.filter{\_%3 ==0}map{i => i\*i}

res63: Array[Int] = Array(9)

效果跟上一行是一样的。

## Map

//Map里面的元素是Key、Value对，如下所示：

scala> val persons = Map("Spark" -> 6, "Hadoop"->11)

persons: scala.collection.Map[String,Int] = Map(Spark -> 6, Hadoop -> 11)

//访问Key为"Hadoop"的元素，获得该键值对中的Value

scala> persons ("Hadoop")

res65: Int = 11

//申明一个可变Map变量，Key是String，Value是Int类型

scala> val pesons = scala.collection.mutable.Map("Spark" -> 6, "Hadoop"->11)

pesons: scala.collection.mutable.Map[String,Int] = Map(Hadoop -> 11, Spark -> 6)

//对其进行增加元素操作

scala> persons += ("Flink" -> 5)

res67: persons.type = Map(Hadoop -> 11, Spark -> 6, Flink -> 5)

//对其进行减元素操作

scala> persons -= "Flink"

res68: persons.type = Map(Hadoop -> 11, Spark -> 6)

//通过条件判断来获取map元素的值，判断该元素是否存在

scala> val sparkValue = if(persons.contains("Spark")) persons("Spark") else 1000

sparkValue: Int = 6

//Map自带getOrElse函数用于获取某个元素

//首先查看persons的内容

scala> persons

res70: scala.collection.mutable.Map[String,Int] = Map(Hadoop -> 11, Spark -> 6)

//访问时存在Spark元素

scala> val sparkValue = persons.getOrElse("Spark",1000)

sparkValue: Int = 6

//访问时不在Flink元素

scala> val sparkValue = persons.getOrElse("Flink",1000)

sparkValue: Int = 1000

//循环遍历Map中的元素

scala> for((key,value) <-persons) println("key:"+key+",value:"+value)

key:Hadoop,value:11

key:Spark,value:6

//注意，此时(key,value)其实是一个Tuple

//遍历Map中的全部的Key

scala> for(key <-persons.keySet) println("key:"+key)

key:Hadoop

key:Spark

//SortedMap

scala> val persons = scala.collection.immutable.SortedMap("Spark" ->6, "Hadoop" -> 11)

persons: scala.collection.immutable.SortedMap[String,Int] = Map(Hadoop -> 11, Spark -> 6)

此时Hadoop元素排在Spark元素的前面

## Tuple

一个元组里面有很多不同的类型的元素，接收函数的多个参数时，Tuple特别有用！

//定义了一个Tuple，里面有三个不同类型的元素

scala> val tuple = ("Spark",6,99.01)

tuple: (String, Int, Double) = (Spark,6,99.01)

//访问Tuple变量的第1个元素，注意是顺序从1开始！

scala> tuple.\_1

res72: String = Spark

//访问Tuple变量的第2个元素。

scala> tuple.\_2

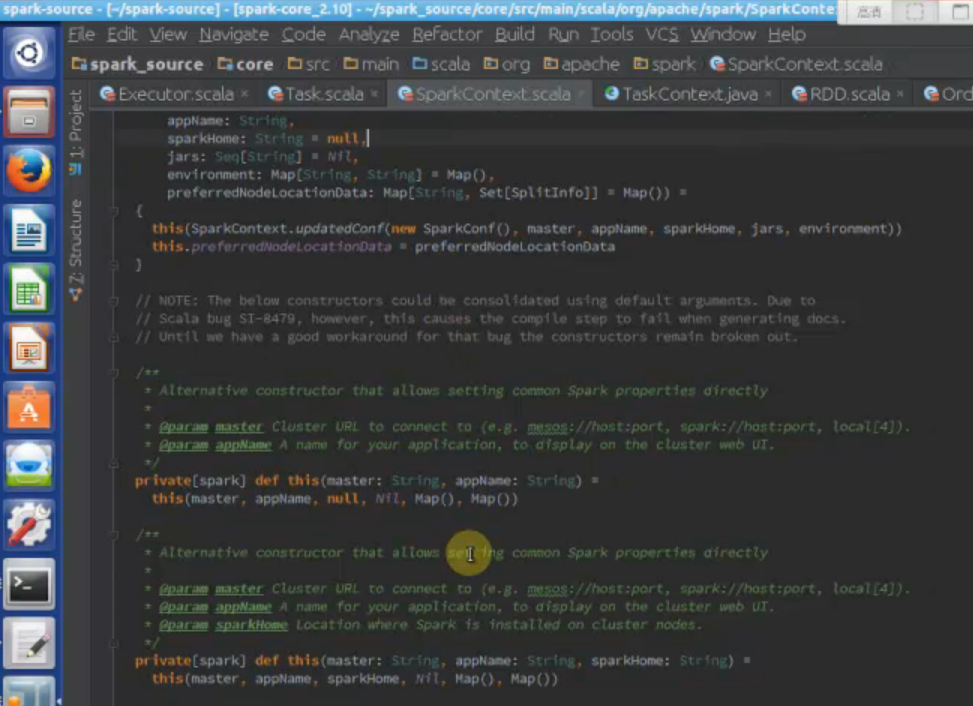
res73: Int = 6

# 综合案例及Spark源码解析

在您安装了Spark代码开发神器：IDEA-的前提下，通过IDEA，就可以欣赏Spark源码了。王老师给我们欣赏了SparkContext的源码，您可以在下面目录下找到：

您的Spark源码目录\......\org\apache\spark\SparkContext.scala

源码截图如下：



在该文件源码中，您几乎可以看到本章节所讲的全部知识点。

# 作业

1、移除一个数组中第一个负数后的所有负数