Scala Programming Learning

1. Scala介绍与安装

Scala编译源代码scala文件生成class字节码文件

安装Scala之后，需要配置环境变量C:\Program Files (x86)\scala\bin 到Path中，Linux中直接解压，添加到环境变量中即可。同时在Linux中有一个deb的安装包，因此我们可以使用dpkg安装此软件，但是不推荐。

1. Scala基础

命令行模式下变成，所见即所得

Java中常量使用的是 final

Scala声明变量使用的是var v=100, 常量使用val var=100

具体声明变量的类型使用 var c:Int=100, var v:Byte=128 (ERROR)

val a,b=100

Scala 数据类型：Byte Char Short Int Long Float Double Boolean

Scala中不区分基本类型和class，全都是类，可以直接访问每一个类型的所有方法，不存在拆箱和装箱的问题。

1. Scala操作符重载 包导入

在Scala中，所有的操作符都进行了重载

Scala中没有++ --之类的操作符，一般也不要使用+=之类的运算符，在2.10之下不支持。

包导入：

import scala.math.\_ 等价于Java中的\*，导入该目录下的所有类

sqrt(2)

pow(2,3)

min(3,5)

“Heollo”.distinct 去重 Hello(4) 类似操作数据，实际上调用的是apply(4)方法

BigInt(1000) 实际上调用的是BigInt.apply(1000)

1. Sccla 结构控制和函数

val s = if(x>0) 1 else -1

val s = if(x>0) 1 else “zhangsan”

Scala中的每一个语句都有值和类型，如果没有else后面的，则数据是AnyVal

命令行中输入粘贴的代码段使用 “:past” 之后粘贴代码，然后Ctrl +D 执行

块表达式的赋值代码是最后一句代码的值

不可以使用 x=y=1这种表达式，（空值不能给其他进行赋值）

输入：

val name=readLine(“input your name:”)

输出和C语言类似 println

循环

while(n>0){

n=n-1

println(n)

}

for( i<- 1 to 10){println (i)}

for(i<-1 until 10) println(i)

多级的嵌套循环

for(i<-1 to 3; j<-1 to 3; if(i==j)) { println(i\*10+j)}

函数与方法在Scala中

方法是在对象上，而函数不是

函数定义指定名称、参数、方法实现，且必须指定参数类型，最后一句是函数的返回值

**def** add(a:Int, b:Int):Int=a+b

参数的default值**def add(a:Int=10, b:Int=10):Int=a+b**

变长参数

**def** fun(names: String\*) {

**for** (name <- names) {

println("hello " + name)

}

}

val s = sum(1,2,3,4,5)

val s = sum(1 to 5 ) ERROR 1 to 5 is an object

val s = sum(1 to 5:\_\*) RIGHT

1. 异常处理和延迟初始化

lazy val v= 10 延迟初始化，在使用的时候在进行赋值，仅仅限于val常量

异常处理

try{

}catch {

case \_:Exception => println()

case ex:IOException=>println()

}finally{

}

1. 数组

定长数组

var nums = new Array[Int](10) 初始化全部是0

var strs = new Array[String](10) 初始化全部是null

var strss = Array(“zhangsa”,”lisi”) strss(0) 访问变量

变长数组

var arrbuf = ArrayBuffer[Int]()

arrbuf.+=(1,2,3,4,5,6)

arrbuf++=Array(7,8,9)

arrbuf.trimEnd(5)

var arrbuf = ArrayBuffer[Int]()

arrbuf.+=(1,2,3,4,5,6)

arrbuf++=Array(7,8,9)

arrbuf.insert(0, 0)

arrbuf.insert(0, -1,-2,-3)

arrbuf.remove(0)

arrbuf.remove(2,3)

var brr = arrbuf.toArray

println( brr.mkString("<", ";", ">"))

for(ele <- brr) println( ele)

var cbuf = brr.toBuffer

for(i<- 0 until brr.length){

println (brr(i))

}

for(i<- (0 until brr.length).reverse){

println (brr(i))

}

for(ele<-brr) println(ele)

var r = for(ele<-brr) yield ele\*2

brr.filter(\_%2==0).map(\_\*2)

多维数组

var arrdem = Array.ofDim[Int](3,4)

arrdem(0)(0)=100

for(r<-arrdem; c<-r) println(c)

for(i<- 0 until 3; j <- 0 until 4) {arrdem(i)(j)=(i+1)\*10+(j+1)}

for(r<-arrdem; c<-r) println(c)

1. Scala数组缓冲区

//关于数组缓冲区的学习

var buf = ArrayBuffer[AnyVal]()

buf.appendAll("abc123")

for(ele <- buf) print(ele + ",")

println

// a,b,c,1,2,3,

var arr = new Array[Int](10)

buf.appendAll(arr)

for(ele <- buf) print(ele + ",")

println

// a,b,c,1,2,3,0,0,0,0,0,0,0,0,0,0,

println (buf.count(\_=='a'))

//1

buf+=100 //在buff后面添加一个value为100 的元素

for(ele <- buf) print(ele + ",")

println

//a,b,c,1,2,3,0,0,0,0,0,0,0,0,0,0,100,

buf-=0 // 删除第一个值为0 的元素

for(ele <- buf) print(ele + ",")

println

//a,b,c,1,2,3,0,0,0,0,0,0,0,0,0,100,

//数组复制

var arr1 = new Array[AnyVal](10)

arr.copyToArray(arr1)

//缓冲数组的系列操作

var buf1 = new ArrayBuffer[Int]()

buf1+=1

buf1+=2

buf1.+=:(3) // add element at the begin

buf1.++=(Array(4,5,6)) // add elements at the end

buf1.++=:(Array(7,8,9)) // add elements at the begin

for(ele <- buf1) print(ele + ",")

println

//7,8,9,3,1,2,4,5,6,

1. 数组缓冲区操作符重载解析

操作符重载

+ //表示操作一个值

++ //表示操作一个集合

= //修改源缓冲区，否则原缓冲区不变，产生新的缓冲区

：//以冒号结尾的，通常是前导操作。

:+ //源缓冲区不变，产生新的缓冲区

下面的两个结果不一致，但是很少使用这种方式去进行编写代码，可读性太差

/: //左包含 (1,2,3) (3) => (\_-\_) 操作符

buf./:(3)(\_-\_)

:\ //右包含 (1,2,3) (3) => (\_-\_) 操作符

buf.:\(3)(\_-\_)

1. Eclipse插件安装以及Eclipse查看Scala源代码

建议使用Eclipse Keper版本，安装对应的Eclipse Scala插件

<http://download.scala-ide.org/sdk/helium/e38/scala210/stable/site>

因为Spark1.5.1使用的是Scala 2.10版本

我们不需要在系统中安装Scala环境，因为在插件中自己已经带了Scala的依赖包

在类中的main函数不可以运行，只有在Object中的main函数才可以运行Scala Application程序

1. Scala 与 Java互操作

javap \*\*\*.class 查看Class文件的签名

var list = scala.colection.JavaConversions.bufferAsJavaList(buf)

1. Scala 与 Java的互操作 双向处理

由于Scala数组是由Java数组实现的，因此可以在Java和Scala之间来回传递。如果你调用接受或者返回java.util.List的Java方法，则当然可以在Scala中使用ArrayList，但是没什么意思。所以你可以引入scala.collection.JavaConversions里的隐式转换方法，这样就可以在代码中使用Scala的Buffer，在调用Java方法的时候，这些对象会被自动包装成为Java列表。

import java.util.ArrayList

import scala.collection.JavaConversions.asScalaBuffer

import scala.collection.JavaConversions.bufferAsJavaList

import scala.collection.mutable.ArrayBuffer

import scala.collection.mutable.Buffer

//Scala to java

val command = ArrayBuffer("notepad", "d:\\note\\yang.txt")

val pb = new ProcessBuilder(command)

//这里的Scala的缓冲被包装成为一个实现了java.util.List接口的Java类的对象

pb.start

//java to scala

var jlist:ArrayList[String] = new ArrayList[String]()

jlist.append("zhang")

jlist.append("lisi")

jlist.append("wang")

var buff:Buffer[String] = jlist

//如果Java方法返回一个java.util.List的时候，也可以自动转化为Scala的Buffer。

Chapter 1 Scala基础知识

Chapter 2 控制结构和函数

Chapter 3 数组相关操作

Chapter 4 映射和元组

Chapter 5 类

Chapter 6 对象

Chapter 7 包和引入

Chapter 8 继承

Chapter 9 文件和正则表达式

Chapter 10 特质

Chapter 11 操作符

Chapter 12 高阶函数

Chapter 13 集合

Chapter 14 模式匹配和样例类

Match 是一个更好的switch，不会有意外掉入到下一个switch的问题

对于表达式的类型进行匹配，优先选择模式匹配，而不是is/asInstanceOf

使用Option来存放对于可能存在也可能不存在的值，这样比Null更加安全

Chapter 15 注解

Chapter 16 XML处理

Chapter 17 类型参数

Chapter 18 高级类型

Chapter 19 解析

Chapter 20 Actor

Chapter 21 隐式转换和隐式函数

Chapter 22 定界延续