第2章Scala基础

(源自:https://biglab.site)

(版本:Ver1.2-20230828)

学习目标

- 了解Scala基本特性并学会安装scala
- 掌握定义Scala的常量、变量和函数
- 掌握Scala的if判断和循环
- 掌握Scala的Collections操作
- 了解Scala类及读取文件

学习重点

理论学习

- （1）Scala简介与安装。
- （2）Scala编程学习。

实验学习

- （1）安装Scala编程环境。
- （2）定义Scala函数识别号码类型。
- （3）统计广州号码段数量。
- （4）根据归属地对手机号码段分组。
- （5）编写手机号码归属地查询程序。
- （6）编写函数过滤文本中的回文单词。
- （7）使用Scala编程实现杨辉三角。

学习视频

任务2.1环境安装

- Scala的简介与安装(20230909更新)

任务2.2Scala数组和函数定义

- Scala数组和函数定义(20230909更新)

任务2.3统计广州号码段数量

- 统计广州号码段数量 (20230912更新)

任务2.4根据归属地对手机号码段分组

- 根据归属地对手机号码段分组(循环&判断列表&集合&映射)(20230912更新)

任务2.5编写手机号码归属地信息查询

- (scala类&单例&模式匹配)，读写文件&编写手机号码归属地信息查询程序

任务2.1Scala的简介与安装

Scala简介

- Scalable Lanaguage简写
- 2001年由联邦理工学院洛桑EPFL的Martin Oderskey基于Funnel的工作开始设计的
- 是一种纯面向对象的语言，每个值都是对象
- 是一种函数式语言，其函数也能当成值也使用
- 被编译也Java字节码，可以运行在JVM上，可以调用Java类库

Scala特性

- 面向对象
  - 是一种纯面向对象的语言，每个值都是对象
- 函数式编程
  - 是一种函数式语言，其函数也能当成值也使用
- 静态类型
  - 以静态的方式进行抽象，以安全和连贯的方式进行使用
- 是扩展的
  - 提供独特语言机制，可以以库的形式轻易无缝添加新的语言结构

Scala的环境设置及安装

在网页上运行Scala

https://www.it1352.com/Onlinetools/details/21

需要java环境

- jdk下载： Java Downloads | Oracle
- 进入Java archive -> Java SE 8 (8u211 and later) -> jdk-8u333-windows-x64.exe 以及jdk-8u333-linux-x64.tar.gz
- 运行 jdk-8u333-windows-x64.exe进行安装jdk
- java -version 检查版本

Scala安装

Linux安装

运行批命令安装Scala

```
cd /root
 wget https://downloads.lightbend.com/scala/2.12.16/scala-2.12.16.tgz
 tar -xvf ./scala-2.12.16.tgz
 mv ./scala-2.12.16 /usr/local/scala
```

配置/etc/profile

```
export SCALA_HOME=/usr/local/scala
export PATH=$SCALA_HOME/bin:$PATH
```

测试是否安装完成

```
-bash-4.2# scala -version
 Scala code runner version 2.12.16 -- Copyright 2002-2022, LAMP/EPFL and Lightbend, Inc.
```

Windows安装

下载msi安装包

https://www.scala-lang.org/download/2.12.16.html

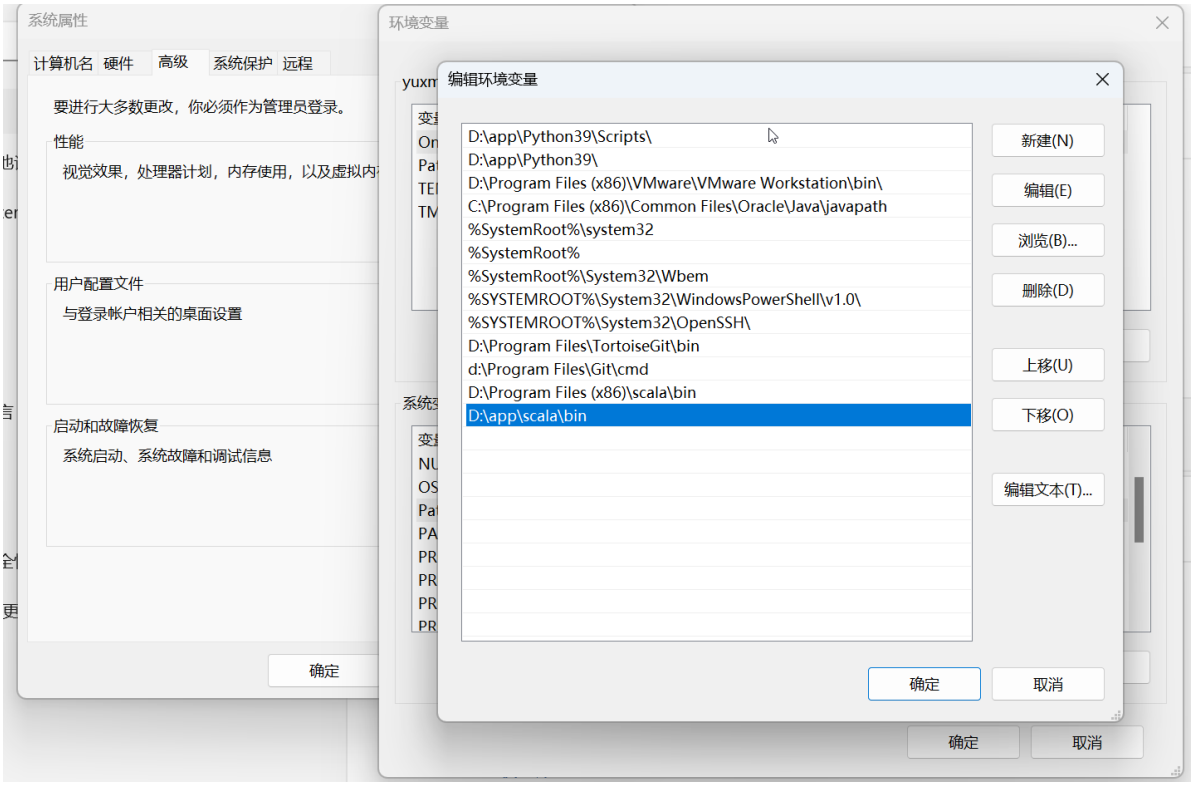
运行msi安装包

建议安装到D:\app\scala目录下

配置环境变量

右键单击“此电脑”图标，选择“属性”选项，在弹出的窗口中选择“高级系统设置”选项。在弹出的对话框中选择“高级”选项卡，并单击“环境变量”按钮，在环境变量对话框中，选择“Path”变量并单击“编辑”按钮，在Path变量中添加Scala安装目录的bin文件夹所在路径(如果没有的话)，如“D:\app\scala\bin”。

如下图:



测试是否安装完成

```
C:\Users\yuxm>scala -version
 Scala code runner version 2.12.16 -- Copyright 2002-2022, LAMP/EPFL and Lightbend, Inc.

 C:\Users\yuxm>scala
 Welcome to Scala 2.12.16 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_191).
 Type in expressions for evaluation. Or try :help.

 scala> 42*78
 res0: Int = 3276

 scala> :quit
```

运行Scala

运行scala

退出:quit

使用Paste模式:paste

Ctrl+D退出

进入:paste

粘贴测试类

```
object add2{
   def addInt(a:Int,b:Int):Int={
     var sum:Int=0
     sum = a+b
     return sum
   }
 }
```

Ctrl+D退出

测试方法

```
import add2.addInt;

addInt(2,3);
```

~运行视频

任务2.2定义函数识别号码类型

任务描述

使用数组存储各种类型的手机号段，并编写一个函数来识别手机号码类型

数据类型

- Scala数据类型与java数据类型相似，但不同
- Scala没有原生的数据类型，都是对象
- 第一个字母都必须大写
- 常用类型:Byte,Short,Int,Long,Float,Double,Char,String,Boolean,Unit,Null,Nothing,Any,AnyRef
- 特殊
  - Unit表示无值，和Java的void等同，用作不返回什么何结果的方法的结果类型
- 会基于使用值的方式来确定类型，称为类型推断

常量和变量

使用val定义常量

- 在程序运行过程中值不变也不可变
- 定义常量时可以不用显式说明类型，会有类型推断
  - val x=1+1
- 使用:type是显示说明类型
  - val x:Int=1+1
- 要求以字母或下划线开头
- 不能使用美元符号$

使用var定义变量

- 变量是在程序运行过程中其值可能发生改变的量
- 语法: var name:type=initialization
- 也可以不显式地说明数据类型
- 只能把同类型的值重新赋值给变量，否则报错

表达式

内置运行符

- 算术运算符
  - +, -, *, /, %求余
- 关系运算符
  - >,<,>=,<=,==判断相等,!=不等
- 逻辑运算符
  - &&与,||或, !取反
- 位运算符
  - &, |, ^
- 赋值运算符
  - =,+=,-=,*=,/=,%=求余后赋值,<<=,>>=,&=,|=,^=

数组

- 数组申明

```
var z:Array[String] = new Array[String](num)

var z = Array(元素1，元素2，元素3，...)
```

- 数组赋值

```
scala> var z:Array[String]=new Array[String](3)
z: Array[String] = Array(null, null, null)

scala> z(0)="baidu";z(1)="google";z(2)="biying"
```

- 数组操作
  - arr.length,返回数组的长度
  - arr.head,查看数组第一个元素
  - arr.tail,查看除第一个元素外的其他元素
  - arr.isEmpty,判断是否为空
  - arr.contains(x),判断是否包含元素x
  - ++可以连接两个数组

```
scala> var x:Array[String]=new Array[String](3)
x: Array[String] = Array(null, null, null)

scala> x(0)="abc";x(1)="cde";x(2)="def"
scala> z++x
res17: Array[String] = Array(baidu, google, biying, abc, cde, def)
```

- concat可以连接两个数组

```
import Array._

scala> import Array._
 import Array._

 scala> var y=concat(z,x)
 y: Array[String] = Array(baidu, google, biying, abc, cde, def)
```

- range可以创建区间数组

```
scala> var arr=range(1,10,2)
arr: Array[Int] = Array(1, 3, 5, 7, 9)
```

- 创建可变数组

```
import scala.collection.mutable.ArrayBuffer
```

函数

函数申明

def functionName (参数列表) : [return type] = {}

定义说明

- 参数列表中每个参数都有一个名字，参数名后跟着:和参数类型
- 返回类型可以是任意合法的数据类型，无返回则为Unit类型
- 可以不加return关键字来指时返回值

函数代码

有return函数

```
object Test {
 def add(a:Int,b:Int):Int={
     var sum=0;
     sum=a+b;
     sum
  }
 }

 scala> Test.add(100,200)
 res21: Int = 300
```

运行结果截图

无return函数

```
scala> def add(a:Int,b:Int):Int={a+b}
 add: (a: Int, b: Int)Int

 scala> add(100,200)
 res20: Int = 300
```

函数调用

- functionName(参数列表)
- 类方法：Test.addInt(a,b)

函数式语言

匿名函数

- 使用**"=>"定义，左边是参数列表，右边是表达式**
- 可以将**函数赋值给一个常量或变量**，然后通过常量名或变量名调用函数
- 可以**使用占位符_代替参数**
- 例如

```
scala> val add4=(x:Int,y:Int)=>x+y
 add4: (Int, Int) => Int = $Lambda$1229/1461381909@1919c877

 scala> add4(123,234)
 res22: Int = 357
```

- 占位符例如

```
scala> val add5=(_:Int)+(_:Int)
 add5: (Int, Int) => Int = $Lambda$1233/827294752@213eeb34

 scala> add5(234,345)
 res23: Int = 579
```

高阶函数-函数作为参数

- 高阶函数就是**操作其他函数的函数**，可以使用函数作为参数
- 例如

```
scala> def add6(f:(Int,Int)=>Int,a:Int,b:Int)=f(a,b)
 add6: (f: (Int, Int) => Int, a: Int, b: Int)Int

 scala> add6((a:Int,b:Int)=>a+b,1,2)
 res24: Int = 3

 scala> add6((a:Int,b:Int)=>a*b,3,4)
 res25: Int = 12
```

定义了f函数,f调用了a,b作为参数

定义了add6高阶函数,函数中使用了f函数作为参数1,a,b也作为参数

高阶函数-函数作为返回值

- 高阶函数可以产生新的函数，并将新的函数作为返回值
- 例如:

```
scala> def rectangle(length:Double)=(height:Double)=>(length+height)*2
 rectangle: (length: Double)Double => Double

 scala> var func=rectangle(4)
 func: Double => Double = $Lambda$1251/1175328696@5b724a76

 scala> func(5)
 res29: Double = 18.0
```

定义匿名函数
(height)=>(length+height)*2
将匿名函数赋给rectangle(length:Double)函数

将length赋值给rectange,则返回匿名函数
将匿名函数定义给变量func

调用 func函数，将height传入计算出结果

函数柯里化

- 柯里化是把接收多个参数的函数变换成接收一个单一参数的函数
- 例如:

```
scala> def add7(a:Int)(b:Int):Int=a+b
 add7: (a: Int)(b: Int)Int

 scala> add7(123)(456)
 res30: Int = 579
```

任务实现

任务目标:

使用数组存储各种类型的手机号段，并编写一个函数来识别手机号码类型

实现步骤

- 用数组分别存储各种类型的手机号段
- 定义函数识别手机号段
- 测试133号段属于哪家电信企业

源码

```
var yidong = Array(1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348)
yidong=yidong++Array(135, 136, 137, 138, 139, 150, 151, 152, 157)
yidong=yidong++Array(158, 159, 182, 183, 184, 187, 188, 178, 147, 1705)
val liantong = Array(130, 131, 132, 155, 156, 185, 186, 176, 145, 1709)
val dianxin = Array(133, 1349, 153, 180, 181, 189, 1700, 177)

def identify(x: Int) = {
  if (yidong.contains(x)) {
    println("这个号码是属于中国移动类型的")
  } else if (liantong.contains(x)) {
    println("这个号码是属于中国联通类型的")
  } else if (dianxin.contains(x)) {
    println("这个号码是属于中国电信类型的")
  } else {
    println("这个号码不属于中国号码")
  }
}

identify(133)
```

任务2.3统计广州号码段数量

任务描述

统计手机号段的归属地来反映该地区的人流量

if判断

使用if判断语法

```
# if…else if…else语句
if(布尔表达式1) {
   若布尔表达式1为true，则执行该语句块
} else if(布尔表达式2) {
      若布尔表达式2为true，则执行该语句块
} else if(布尔表达式3) {
      若布尔表达式3为true，则执行该语句块
}else {
      若以上布尔表达式都为false，则执行该语句块
}

```

例如：代码2-11 if条件判断示例

```
// 定义一个变量x，值为10
var x = 10
// 使用if对变量的值进行判断并对应输出结果
if(x == 10) {
    println("x的值为10")
} else if(x == 20) {
    println("x的值为20")
} else{
    println("无法判断x的值")
}

```

循环

while循环

do...while循环

for循环

语法: for (变量 <- 集合) {循环语句}

使用a to b表示从a到b,包含b

例如：代码2-12 for循环嵌套打印九九乘法表

```
 var i, j = 0;
 for (i <- 1 to 9) {
   for (j <- 1 to 9)
     println("(" + i + "," + j + ")")
 }

```

```
// 循环嵌套打印九九乘法表优化后脚本
var i, j = 0;
for (i <- 1 to 9) {
  for (j <- 1 to i) {
    print(i + "*" + j + "=" + i * j + " ")
  }
  println("\n")
}

```

使用a until b 表示从a到b,不包含b

```
scala> for (i<- 2 until 10) {
   | print(i+" ")
   | }
 2 3 4 5 6 7 8 9
 scala>
```

多重循环

```
// for循环打印九九乘法表脚本2
for (i <- 1 to 9; j <- 1 to i) {
  print(i + "*" + j + "=" + i * j + " ")
  if (i == j) println()
}
```

运行结果：

```
1*1=1
 2*1=2 2*2=4
 3*1=3 3*2=6 3*3=9
 4*1=4 4*2=8 4*3=12 4*4=16
 5*1=5 5*2=10 5*3=15 5*4=20 5*5=25
 6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
 7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
 8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
 9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

for循环结合if判断，如：代码2-13 for循环嵌套if语句示例

```
for (i <- 2 until 10) {
  print (i + " ")
}
```

运行结果：

```
scala>   for (i <- 2 until 10) {
     |     print (i + " ")
     |   }
2 3 4 5 6 7 8 9
scala>
```

for循环结合yield

使用yield可以将返回值作为一个变量存储

```
scala> var even = for (i<- 1 to 10; if i%2==0 ; if i>6) yield i
 even: scala.collection.immutable.IndexedSeq[Int] = Vector(8, 10)
```

任务实现

源码实现

```
  def count(area: String) = {
    val arr = Array("115036,1477799,广东,广州,中国移动,020,510000",
      "115038,1477801,广东,东莞,中国移动,0769,511700",
      "115033,1477796,广东,广州,中国移动,020,510000",
      "115032,1477795,广东,广州,中国移动,020,510000")
    var sum = 0
    for (a <- arr; if a.contains(area)) {
      sum += 1
    }
    println(sum)
  }

  count("广州")
```

任务2.4根据归属地对手机号码段分组

任务描述

统计某个地区所有手机号码段

List

列表与数组相似，列表的所有元素都具有相同的类型。与数组不同的是列表是不可变的，即列表的元素不能通过赋值来更改

List定义实例

```
scala> val fruit:List[String]=List("apple","pears","oranges")
 fruit: List[String] = List(apple, pears, oranges)

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

 scala> val double_nums:List[Double]=List(1.0,2.5,3.3)
 double_nums: List[Double] = List(1.0, 2.5, 3.3)

 scala> val empty:List[Nothing]=List()
 empty: List[Nothing] = List()
```

构造列表的Nil和::

Nil表示空列表

::称为中缀操作符

实例

```
scala> val fruit1:List[String]="apple"::"pears"::"orange"::Nil
 fruit1: List[String] = List(apple, pears, orange)

 scala> val nums1:List[Int]=1::2::3::4::5::Nil
 nums1: List[Int] = List(1, 2, 3, 4, 5)
```

常用方法

def head:A

取列表第一个元素

```
scala> fruit1.head
 res24: String = apple
```

def init:List[A]

取所有元素，除最后一个

```
scala> fruit1.init
 res23: List[String] = List(apple, pears)
```

def last:A

取最后一个元素

```
scala> fruit1.last
 res22: String = orange
```

def tail:List[A]

取所有元素，除第一个

```
scala> fruit1.tail
 res21: List[String] = List(pears, orange)
```

def :::(prefix:List[A]):List[A]

添加元素

```
scala> val fruit2:List[String]=List("abc","bcd","cde")
fruit2: List[String] = List(abc, bcd, cde)

scala> fruit1.::::(fruit2)
res17: List[String] = List(abc, bcd, cde, apple, pears, orange)

scala> fruit1:::fruit2
res18: List[String] = List(apple, pears, orange, abc, bcd, cde)
```

def take(n:Int):List[A]

获取列表前n个元素

```
scala> fruit1.take(3)
 res20: List[String] = List(apple, pears, orange)
```

def contains(elem:Any):Boolean

判断列表是否包含指定元素

```
scala> fruit1.contains("apple")
 res25: Boolean = true
```

合并两个列表

除:::外，还可以使用List.concat

```
scala> List.concat(fruit1,fruit2)
 res26: List[String] = List(apple, pears, orange, abc, bcd, cde)
```

Set

Set是没有重复对象的集合，所有元素都是唯一

Set声明

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

常见方法

def head:A

获取第一个元素

```
scala> set.head
 res27: Int = 5
```

def init:Set[A]

返回所有元素，除最后一个

```
scala> set.init
 res28: scala.collection.immutable.Set[Int] = Set(5, 1, 2, 3)
```

def last:A

获取最后一个元素

```
scala> set.last
 res29: Int = 4
```

def tail:Set[A]

返回所有元素除第一个

```
scala> set.tail
 res30: scala.collection.immutable.Set[Int] = Set(1, 2, 3, 4)
```

def ++(elems:A):Set[A]

合并两个集合

```
scala> val set1:Set[Int]=Set(5,6,7)
 set1: Set[Int] = Set(5, 6, 7)

 scala> set.++(set1)
 res31: scala.collection.immutable.Set[Int] = Set(5, 1, 6, 2, 7, 3, 4)
```

def take(n:Int):List[A]

获取列表前n个元素

```
scala> set.take(3)
 res32: scala.collection.immutable.Set[Int] = Set(5, 1, 2)
```

def contains(elem:Any):Boolean

判断是否包含批定元素

```
scala> set.contains(3)
 res33: Boolean = true
```

Map

Map(映射)是一种可迭代的键值对结构，所有值都可以通过键来获取，并且Map中的键都是唯一的

Map声明

```
scala> val person:Map[String,Int]=Map("John"->31,"Betty"->20,"Mike"->22)
 person: Map[String,Int] = Map(John -> 31, Betty -> 20, Mike -> 22)
```

常用方法

head,init,last,tail,++,take,contains

```
scala> person.head
 res34: (String, Int) = (John,31)

 scala> person.init
 res35: scala.collection.immutable.Map[String,Int] = Map(John -> 31, Betty -> 20)

 scala> person.last
 res36: (String, Int) = (Mike,22)

 scala> person.tail
 res37: scala.collection.immutable.Map[String,Int] = Map(Betty -> 20, Mike -> 22)
```

isEmpty

```
scala> person.isEmpty
 res38: Boolean = false
```

keys

```
scala> person.keys
 res39: Iterable[String] = Set(John, Betty, Mike)
```

values

```
scala> person.values
 res0: Iterable[Int] = MapLike.DefaultValuesIterable(31, 20, 22)
```

元组

元组是一种类似于列表的结构，不同的是元组可以包含不同类型的元素

元组的值是通过将单个的值包含在圆括号中的构成的

元组定义：

```
 val t = new Tuple3(1,3.14,"a")
```

Tuple3表示三元组

最大支持22个元素

元组定义2:

val t=(1,3.14,"a")

```
scala> var t=(1,3.14,"a")
 t: (Int, Double, String) = (1,3.14,a)
```

元组访问元素: 元组名._元素索引

```
scala> var t=(1,3.14,"a")
 t: (Int, Double, String) = (1,3.14,a)

 scala> t._2
 res1: Double = 3.14
```

函数组合器

运用函数组合器的操作会在集合中的每个元素上分别应用一个函数

List+map

通过List的map函数重新计算列表中的所有元素，并且返回一个相同数目元素的新列表

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

 scala> num.map(x=>x*x)
 res3: List[Int] = List(1, 4, 9, 16, 25)
```

List+foreach

通过List的foreach函数，对参数进行作用，没有返回值

```
scala> num.foreach(x=>print(x*x+"\t"))
 1    4    9    16    25
```

List+filter

filter函数移除传入函数的返回值为false的元素(过滤出来表达式为true的值)

```
scala> num.filter(x=>x%2==0)
 res5: List[Int] = List(2, 4)
```

List+flatten

把二维列表展开成一个一维列表

```
scala> val list=List(List(1,2,3),List(4,5,6))
 list: List[List[Int]] = List(List(1, 2, 3), List(4, 5, 6))

 scala> list.flatten
 res6: List[Int] = List(1, 2, 3, 4, 5, 6)
```

List+flatMap

结合map和flatten功有，接收一个可以处理嵌套列表的函数，然后把返回结果连接起来

```
scala> val str = List("a:b:c","d:e:f")
str: List[String] = List(a:b:c, d:e:f)

scala> str.flatMap(x=>x.split(":"))
res61: List[String] = List(a, b, c, d, e, f)

scala>
```

List+groupBy

是对集合中的元素进行分组操作，结果得到的是一个Map

```
scala> val num:List[Int]=List(1,2,3,4,5,6,7,9,10)
 num: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 9, 10)

 scala> num.groupBy(x=>x%2==0)
 res0: scala.collection.immutable.Map[Boolean,List[Int]] = Map(false ->
List(1, 3, 5, 7, 9), true -> List(2, 4, 6, 10))
```

任务实现

源码实现

```
scala> val phone:List[String]=List("70999,1371001,广东,广州,中国移动,020,510000",
                                   "71000,1371002,广东,广州,中国移动,020,510000",
                                   "71348,1371350,广东,深圳,中国移动,0755,518000",
                                   "71349,1371351,广东,深圳,中国移动,0755,518000")
 phone: List[String] = List(70999,1371001,广东,广州,中国移动,020,510000, 71000,1371002,广东,广州,中国移动,020,510000, 71348,1371350,广东,深圳,中国移动,0755,518000, 71349,1371351,广东,深圳,中国移动,0755,518000)

 scala> phone.groupBy(x=>x.split(",")(3))
 res14: scala.collection.immutable.Map[String,List[String]] = Map(广州 -> List(70999,1371001,广东,广州,中国移动,020,510000, 71000,1371002,广东,广州,中国移动,020,510000), 深圳 -> List(71348,1371350,广东,深圳,中国移动,0755,518000, 71349,1371351,广东,深圳,中国移动,0755,518000)
```

任务2.5编写手机号码归属地信息查询

任务描述

读取所有数据，根据完整数据编写手机号码归属地查询程序

Scala类

Scala是面向对象的语言，有类和对象，类是对象的抽象，类是模板，对象才是真正的实体

定义 Class className(参数列表) extends t{}

- 类名第一个字母大写
- 不能申明为public
- 类可以有参数

类的定义实例

```
class Point(xc:Int,yc:Int){
 var x:Int =xc
 var y:Int =yc
 def move(dx:Int,dy:Int){
 x=x+dx
 y=y+dy
 println("x的坐标为:"+x)
 println("y的坐标为:"+y)
 }
 }
```

运行结果

```
defined class Point
```

类的实例化

```
scala> var p1=new Point(1,2)
 p1: Point = Point@e7afdd0

 scala> p1.move(2,3)
 x的坐标为:3
 y的坐标为:5
```

类的继承

继承特性

- 使用 extends来继承父类
- 父类中已有方法要用override

继承类

```
scala> class Location(val xc:Int,val yc:Int,val zc:Int) extends Point(xc,yc){
       var z:Int=zc
       def move(dx:Int,dy:Int,dz:Int){
           x=x+dx
           y=y+dy
           z=z+dz
           println("x的坐标"+x)
           println("y的坐标"+y)
           println("z的坐标"+z)
       }
  }
 defined class Location
```

继承类的实例化

```
scala> var loc = new Location(0,0,0)
 loc: Location = Location@7d352874

 scala> loc.move(2,3,4)
 x的坐标2
 y的坐标3
 z的坐标4

 scala> loc.move(2,3,4)
 x的坐标4
 y的坐标6
 z的坐标8
```

Override的使用

```
abstract class Father {
 def fun1 = 1
 def fun2: Int
}

class Child extends Father {
 override def fun1 = 2
 def fun2 = 1
}

var a=new Child()
a.fun1
a.fun2
```

运行结果：

```
//代码2-22 override关键字的使用
scala> abstract class Father {
     |     def fun1 = 1
     |     def fun2: Int
     | }
defined class Father

scala> class Child extends Father {
     |     override def fun1 = 2
     |     def fun2 = 1
     | }

scala> var a=new Child()
a: Child = Child@2095f1f

scala> a.fun1
res89: Int = 2

scala> a.fun2
res90: Int = 1
```

Scala 单例模式

使用单例模式时需要使用object定义一个单例对象

object特点

- object不能带参数
- 包含main方法的object对象可以作为程序的入口
- 单例对象与某个类共享同一个名称时，称伴生对象或伴生类；类和它的伴生对象可以互相访问私有成员

定义：object ObjectName{...}

实例

```
scala> object Person{
    val age=10
    def getAge=age
    }
 defined object Person
```

伴生类和伴生对象实例

```
:paste
 // Entering paste mode (ctrl-D to finish)

 class Person private(val name:String){
   private def getSkill()=name+"s skill is :"+Person.skill
 }
 object Person{
     private val skill="basketball"
     private val person = new Person("Tracy")
     def printSkill=println(person.getSkill())
     def main(args:Array[String]):Unit={
         Person.printSkill
    }
 }

 // Exiting paste mode, now interpreting.

 defined class Person
 defined object Person


scala> val para=new Array[String](0)
para: Array[String] = Array()

scala> Person.main(para)
Tracys skill is :basketball

scala> Person.printSkill
Tracys skill is :basketball
```

Scala模式匹配

一个模式匹配包含了一系列备选项，每个都开始于关键字case

模式匹配特点

- 备选项包含一个到多个表达式
- 模式和表达式之间用“=>"隔开
- match对应java的switch
- match不需要break,隐含break
- match选择器不仅可以是变量，还可以是列表
- case关键字定义的类称为样例类

模式匹配实例

变量模式

```
scala> def matchTest(x:Int)=x match{
    case 1=>println("one")
    case 2=>println("two")
    case _=>println("any")
    }
 matchTest: (x: Int)Unit

 scala> matchTest(3)
 any

 scala> matchTest(2)
 two
```

列表模式

```
def matchList(x:List[Int])=x match{
    case List(0,_,_) =>println("列表x有3个元素并且第一个元素是0")
    case List(1,_*) => println("列表x有任意个元素并且第一个元素是1")
    case List(_,1,_*) => println("列表至少有两个元素并且第二个元素是1")
    }
 matchList: (x: List[Int])Unit

scala> matchList(List(1,2,3))
```

列表x有任意个元素并且第一个元素是1

```
 scala> matchList(List(5,1,3))
```

列表至少有两个元素并且第二个元素是1

```
 scala> matchList(5::1::3::Nil)
```

列表至少有两个元素并且第二个元素是1

样例类

使用case关键字定义的类称为样例类

```
case class Person(name: String,age: Int)
val alice = new Person("Alice", 25)
val bob = new Person("Bob", 22)
val mike = new Person("mike", 24)
for(person <- List(alice, bob, mike)) {
    person match {
        case Person("Alice", 25) => println("Hi, Alice!")
        case Person("Bob", 22) => println("Hi, Bob!")
        case Person(name, age) => println("name:" + name + "\t" + "age:" +
age)
    }
}

```

Scala读取文件

写文件

使用Java的I/O类的PrintWriter

win上测试

```
//请先在D盘中创建 tmp 目录
import java.io._
 val pw=new PrintWriter(new File("D:\\tmp\\scala1.txt"))
 pw.write("I am learning Scala")
 pw.close
```

注意文件名中的双斜线

linux上测试

```
//请先在root目录下创建scala子目录： mkdir /root/scala
scala> :quit
You have new mail in /var/spool/mail/root
[root@master ~]# mkdir /root/scala
[root@master ~]#
[root@master ~]# scala
Welcome to Scala 2.12.16 (OpenJDK 64-Bit Server VM, Java 1.8.0_322).
Type in expressions for evaluation. Or try :help.

scala>
scala> :paste
 // Entering paste mode (ctrl-D to finish)

 import java.io._
 val pw=new PrintWriter(new File("/root/scala/scala1.txt"))

```

```
pw.write("I am learning Scala")
pw.close


// Exiting paste mode, now interpreting.

import java.io._
pw: java.io.PrintWriter = java.io.PrintWriter@4567e53d

scala> :quit
[root@c22 ~]# more /root/scala/scala1.txt
I am learning Scala
[root@c22 ~]#
```

屏幕输入

```
scala> val line=io.StdIn.readLine
line: String = abcde

scala>
```

注意readLine 大小写

abcdef为屏幕输入

读文件

```
import scala.io.Source
 Source.fromFile("D:\\tmp\\scala1.txt").foreach{print}

import scala.io.Source
 Source.fromFile("/root/scala/scala1.txt").foreach{print}

import scala.io.Source
 Source.fromFile("/root/scala/scala1.txt","UTF-8").foreach{print} //带编码
```

任务实现

windows上测试

数据准备

通过浏览器将文件下载到d:\tmp目录下
http://bigdata.hddly.cn/b46488/file/2016phone.txt

win执行

```
import scala.io.Source
import scala.io.StdIn

 object Phone{
     def checkPhone(){
         val phone=for(line<-Source.fromFile("D:\\tmp\\2016phone.txt","UTF-8").getLines) yield line
         val phoneList:List[String]=phone.toList
         println("请输入要查询的号段：")
         val num:String= StdIn.readLine()
         //var num:String="130"
         for(line <- phoneList;if line.contains(num)){println(line)}
     }
 }
```

scala执行

```
import Phone.checkPhone
 checkPhone
```

linux上测试

数据准备

```
mkdir /root/spark
 cd /root/spark
 wget http://bigdata.hddly.cn/b46488/file/2016phone.tar.gz
 tar -xvzf ./2016phone.tar.gz
 ls /root/spark/2016phonelocation.txt
```

scala类

```
import scala.io.Source
 import scala.io.StdIn
 object Phone{
     def checkPhone(){
         val phone=for(line<-Source.fromFile("/root/spark/2016phonelocation.txt","UTF-8").getLines) yield line
         val phoneList:List[String]=phone.toList
         println("请输入要查询的号段：")
         val num:String= StdIn.readLine()
         for(line <- phoneList;if line.contains(num)){println(line)}
     }
 }
```

scala执行

```
import Phone.checkPhone
 checkPhone
```

在云课上

准备数据

```
mkdir /home/shiyanlou/spark
 cd /home/shiyanlou/spark
 wget http://bigdata.hddly.cn/b46488/file/2016phone.tar.gz
 tar -xvzf ./2016phone.tar.gz
 ls /home/shiyanlou/spark/2016phonelocation.txt
```

脚本

```
import scala.io.Source
 import scala.io.StdIn
 object Phone{
     def checkPhone(){
         val phone=for(line<-
Source.fromFile("/home/shiyanlou/spark/2016phonelocation.txt","UTF-
8").getLines) yield line
         val phoneList:List[String]=phone.toList
         println("请输入要查询的号段：")
         val num:String= StdIn.readLine()
         for(line <- phoneList;if line.contains(num)){println(line)}
     }
 }
```

~ 运行结果截图

实训课堂练习

定义函数识别号码类型

任务目标:

使用数组存储各种类型的手机号段，并编写一个函数来识别手机号码类型

任务实现

实现步骤

- 用数组分别存储各种类型的手机号段
- 定义函数识别手机号段
- 测试133号段属于哪家电信企业

源码

```
var yidong = Array(1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348)
  yidong=yidong++Array(135, 136, 137, 138, 139, 150, 151, 152, 157)
  yidong=yidong++Array(158, 159, 182, 183, 184, 187, 188, 178, 147, 1705)
  val liantong = Array(130, 131, 132, 155, 156, 185, 186, 176, 145, 1709)
  val dianxin = Array(133, 1349, 153, 180, 181, 189, 1700, 177)

  def identify(x: Int) = {
    if (yidong.contains(x)) {
      println("这个号码是属于中国移动类型的")
    } else if (liantong.contains(x)) {
      println("这个号码是属于中国联通类型的")
    } else if (dianxin.contains(x)) {
      println("这个号码是属于中国电信类型的")
    } else {
      println("这个号码不属于中国号码")
    }
  }

  identify(133)
```

统计广州号码段数量

任务描述

统计手机号段的归属地来反映该地区的人流量

任务实现

源码实现

```
  def count(area: String) = {
    val arr = Array("115036,1477799,广东,广州,中国移动,020,510000",
      "115038,1477801,广东,东莞,中国移动,0769,511700",
      "115033,1477796,广东,广州,中国移动,020,510000",
      "115032,1477795,广东,广州,中国移动,020,510000")
    var sum = 0
    for (a <- arr; if a.contains(area)) {
      sum += 1
    }
    println(sum)
  }

  count("广州")
```

根据归属地对手机号码段分组

任务描述

统计某个地区所有手机号码段

任务实现

源码实现

```
val phone:List[String]=List("70999,1371001,广东,广州,中国移动,020,510000",
                            "71000,1371002,广东,广州,中国移动,020,510000",
                            "71348,1371350,广东,深圳,中国移动,0755,518000",
                            "71349,1371351,广东,深圳,中国移动,0755,518000")

phone.groupBy(x=>x.split(",")(3))

```

运行结果：

```
scala> val phone:List[String]=List("70999,1371001,广东,广州,中国移动,020,510000",
     |                             "71000,1371002,广东,广州,中国移动,020,510000",
     |                             "71348,1371350,广东,深圳,中国移动,0755,518000",
     |                             "71349,1371351,广东,深圳,中国移动,0755,518000")
phone: List[String] = List(70999,1371001,广东,广州,中国移动,020,510000, 71000,1371002,广东,广州,中国移动,020,510000, 71348,1371350,广东,深圳,中国移动,0755,518000, 71349,1371351,广东,深圳,中国移动,0755,518000)

scala>

scala> phone.groupBy(x=>x.split(",")(3))
res0: scala.collection.immutable.Map[String,List[String]] = Map(广州 -> List(70999,1371001,广东,广州,中国移动,020,510000, 71000,1371002,广东,广州,中国移动,020,510000), 深圳 -> List(71348,1371350,广东,深圳,中国移动,0755,518000, 71349,1371351,广东,深圳,中国移动,0755,518000))

scala>
```

编写手机号码归属地信息查询程序

任务描述

读取所有数据，根据完整数据编写手机号码归属地查询程序

任务实现

数据准备

```
mkdir /root/spark
 cd /root/spark
 wget http://bigdata.hddly.cn/b46488/file/2016phone.tar.gz
 tar -xvzf ./2016phone.tar.gz
 ls /root/spark/2016phonelocation.txt
```

源码实现

```
import scala.io.Source
 import scala.io.StdIn
 object Phone{
 def checkPhone(){
 val phone=for(line<-
Source.fromFile("/root/spark/2016phonelocation.txt","UTF-8").getLines) yield
line
 val phoneList:List[String]=phone.toList
 println("请输入要查询的号段：")
 val num:String= StdIn.readLine()
 for(line <- phoneList;if line.contains(num)){println(line)}
 }
 }
```

//注意添加"UTF-8"编码

scala执行

```
import Phone.checkPhone
 checkPhone
```

编写函数过滤文本中的回文单词

实训目的

- (1)掌握Scala的REPL使用。
- (2)掌握Scala的Array、List、Map等创建与使用。
- (3)掌握Scala循环与判断的使用。
- (4)掌握Scala函数式编程。

训练要点

- 函数的定义。
- 方法的调用

需求说明

回文是指正向和逆向读起来相同的词。例如“mom”和“dad”。编写一个函数用来测试单词是否是回文，若是则打印出该单词。提示: String的revere方法在此处会很有用

思路及步骤

- (1)定义函数isPalindrom(word:String)。
- (2)在丽数中判断单词正向与逆向是否样，若是则打印。
- (3)测试函数。

参考

```
scala> def isPalindrom(word:String):Unit={
    val word2=word.reverse
     if(word==word2){
      println(word+"  is palindrom!")
     }
    }

 scala> isPalindrom("aba")
```

版本历史

- Ver1.1-20220121
    - 初始版本
- Ver1.2-20230828
    - 增加markdown版本

上一章节 下一章节