**Scala程序设计解析**

**配套课后习题**

**目 录**

[第1章 Scala基础 1](#_Toc492825621)

[第2章 控制结构和函数 3](#_Toc492825622)

[第3章 数据结构 6](#_Toc492825623)

[第4章 模式匹配 16](#_Toc492825624)

[第5章 高阶函数 19](#_Toc492825625)

[第6章 类 22](#_Toc492825626)

[第7章 对象 24](#_Toc492825627)

[第8章 包和引入 25](#_Toc492825628)

[第9章 继承 25](#_Toc492825629)

[第10章 特质 28](#_Toc492825630)

[第11章 类型参数 29](#_Toc492825631)

[第12章 文件和正则表达式 30](#_Toc492825632)

[第13章 高级类型 31](#_Toc492825633)

# Scala基础

**1、在Scala REPL中，计算3的平方根,然后再对该值求平方。现在，这个结果与3相差多少？**

解：

scala> scala.math.sqrt(3)

warning: there were 1 deprecation warnings; re-run with -deprecation for details

res5: Double = 1.7320508075688772

scala> res5\*res5

res6: Double = 2.9999999999999996

scala> 3 - res6

res7: Double = 4.440892098500626E-16

**2、如何检测一个变量是val还是var?**

解：  
val是不可变的，而var是可变的，只需要给res变量重新赋值就可以检测res是val还是var了

scala> res9 = 3

<console>:8: error: reassignment to val

       res9 = 3

            ^

**3、Scala允许你用数字去乘一个字符串，去REPL中试一下"crazy"\*3。这个操作做什么？在Scaladoc中如何找到这个操作?**

解：

scala> "crazy"\*3

res11: String = crazycrazycrazy

从代码可以推断,\*是"crazy"这个字符串所具有的方法，但是Java中的String可没这个方法，很明显。此方法在StringOps中。   
  
**4、10 max 2的含义是什么？max方法定义在哪个类中？**  
解：

scala> 10 max 2

res0: Int = 10

scala> 7 max 8

res1: Int = 8

scala> 0 max 0

res2: Int = 0

可以看出,此方法返回两个数字中较大的那个。此方法Java中不存在，所以在RichInt中。   
  
**5、用BigInt计算2的1024次方**

解：

scala> BigInt(2).pow(1024)

res4: scala.math.BigInt = 1797693134862315907729305190789024733617976978942306572734300811577326758055009631327084773224

075360211201138798713933576587897688144166224928474306394741243777678934248654852763022196012460941194530829520850057688

38150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216

**6、在Scala中如何获取字符串“Hello”的首字符和尾字符？**

解：

//获取首字符

"Hello"(0)

"Hello".take(1)

//获取尾字符

"Hello".reverse(0)

"Hello".takeRight(1)

# 控制结构和函数

**1、一个数字如果为正数，则它的signum为1;如果是负数,则signum为-1;如果为0,则signum为0.编写一个函数来计算这个值**

解：

简单的逻辑判断

def signum(num:Int){if(num>0)print(1)else if(num<0)print(-1)else print(0)}

Scala中已经有此方法了，刚才查找API的时候，应该能看到

BigInt(10).signum

**2、一个空的块表达式{}的值是什么？类型是什么？**

解：

在REPL中就能看出来了

scala> val t = {}

t: Unit = ()

可以看出，它的值是()类型是Unit

**3、针对下列Java循环编写一个Scala版本:**

**for(int i=10;i>=0;i–)System.out.println(i);**

解：

for(i <- 0 to 10 reverse)print(i)

**4、编写一个过程countdown(n:Int)，打印从n到0的数字**

解：

def countdown(n:Int){

(0 to n).reverse.foreach(print)

}

**5、编写一个for循环,计算字符串中所有字母的Unicode代码（toLong方法）的乘积。举例来说，"Hello"中所有字符串的乘积为9415087488L**

**解：**

scala> var t:Long = 1

t: Long = 1

scala> for(i <- "Hello"){

| t = t \* i.toLong

| }

scala> t

res57: Long = 9415087488

**6、同样是解决前一个练习的问题，请用StringOps的foreach方式解决。**

**解：**

scala> var t:Long = 1

t: Long = 1

scala> "Hello".foreach(t \*= \_.toLong)

scala> t

res59: Long = 9415087488

**7、编写一个函数product(s:String)，计算字符串中所有字母的Unicode代码（toLong方法）的乘积**

**解：**

def product(s:String):Long={

var t:Long = 1

for(i <- s){

t \*= i.toLong

}

t

}

**8、把7练习中的函数改成递归函数**

**解：**

def product(s:String):Long={

if(s.length == 1) return s.charAt(0).toLong

else s.take(1).charAt(0).toLong \* product(s.drop(1))

}

**9、编写函数计算,其中n是整数，使用如下的递归定义:**

* ** = x\*,如果n是正数的话**
* ** = 1**
* ** = 1/,如果n是负数的话**
* **不得使用return语句**

**解：**

def mi(x:Double,n:Int):Double={

if(n == 0) 1

else if(n>0) x \* mi(x,n-1)

else 1/mi(x,-n)

}

# 数据结构

**1、编写一段代码，将a设置为一个n个随机整数的数组，要求随机数介于0和n之间。**

|  |
| --- |
| scala> def makeArr(n : Int) : Array[Int] = {  | val a = new Array[Int](n);  | val rand = new scala.util.Random();  | for (i <- a) yield rand.nextInt(n);  | }  makeArr: (n: Int)Array[Int]  scala> makeArr(10).foreach(println)  9  8  3  5  9  6  8  8  0  1 |

**2. 编写一个循环，将整数数组中相邻的元素置换。比如**Array(1, 2, 3, 4, 5)置换后为Array(2, 1, 4, 3, 5)

scala> def revert(arr : Array[Int]) = {

| for (i <- 0 until (arr.length - 1, 2)) {

| val t = arr(i);

| arr(i) = arr(i + 1);

| arr(i + 1) = t;

| }

| }

revert: (arr: Array[Int])Unit

scala> val a = Array(1, 2, 3, 4, 5);

a: Array[Int] = Array(1, 2, 3, 4, 5)

scala> revert(a);

scala> a.foreach(println);

2

1

4

3

5

**3. 给定一个整数数组，产出一个新的数组，包含原数组中的所有正值，以原有顺序排列，之后的元素是所有零或负值，以原有顺序排列。**

scala> def sigNumArr(arr : Array[Int]) = {

| val buf = new ArrayBuffer[Int]();

| buf ++= (for (i <- arr if i > 0) yield i)

| buf ++= (for (i <- arr if i == 0) yield i)

| buf ++= (for (i <- arr if i < 0) yield i)

|

| buf.toArray

| }

sigNumArr: (arr: Array[Int])Array[Int]

scala> val a = Array(1, -2, 0, -3, 0, 4, 5);

a: Array[Int] = Array(1, -2, 0, -3, 0, 4, 5)

scala> val b = sigNumArr(a);

b: Array[Int] = Array(1, 4, 5, 0, 0, -2, -3)

scala> b.foreach(println);

1

4

5

0

0

-2

-3

**4. 创建一个由java.util.TimeZone.getAvailableIDs返回的时区集合，并只显示以America/前缀开头的时区，并且有序。**

|  |
| --- |
| scala> import scala.collection.mutable.ArrayBuffer  import scala.collection.mutable.ArrayBuffer  scala> import scala.collection.JavaConversions.asScalaBuffer  import scala.collection.JavaConversions.asScalaBuffer  scala>  scala> def timeZoneName() = {  | val arr = java.util.TimeZone.getAvailableIDs();  | val tmp = (for (i <- arr if i.startsWith("America/")) yield {  | i.drop("America/".length)  | })  | scala.util.Sorting.quickSort(tmp)  | tmp  | }  timeZoneName: ()Array[String]  scala> var c = timeZoneName()  scala> c.foreach(println) |

**5、设置一个映射，其中包含你想要的一些装备，以及它们的价格。然后根据这个映射构建另一个新映射，采用同一组键，但是价格上打9折。**

**解：**

|  |
| --- |
| scala> val map = Map("book"->10, "gun"->18, "ipad"->1000)  map: scala.collection.immutable.Map[String,Int] = Map(book -> 10, gun -> 18, ipad -> 1000)  scala> for((k,v) <- map) yield (k, v \* 0.9)  res0: scala.collection.immutable.Map[String,Double] = Map(book -> 9.0, gun -> 16.2, ipad -> 900.0) |

**6、编写一段WordCount函数，统计传入的字符串中单词的个数**

**解：**

|  |
| --- |
| scala> def wordCount(str:String)={  | val count = new scala.collection.mutable.HashMap[String, Int]  | for(word <- str.split("\\s+"))  | count(word) = count.getOrElse(word,0) + 1  | count  | }  wordCount: (str: String)scala.collection.mutable.HashMap[String,Int]  scala> wordCount("wo shi zhong guo ren, wo hen hao")  res1: scala.collection.mutable.HashMap[String,Int] = Map(hao -> 1, guo -> 1, wo -> 2, shi -> 1, ren, -> 1, hen -> 1, zhong -> 1) |

**7、重复上一个练习，使统计后的单词有序**

**解：**

|  |
| --- |
| scala> def wordCount(str:String)={  | var count = scala.collection.immutable.SortedMap[String, Int]()  | for(word <- str.split("\\s+"))  | count += (word -> (count.getOrElse(word,0) + 1))  | count  | }  wordCount: (str: String)scala.collection.immutable.SortedMap[String,Int]  scala> wordCount("wo shi zhong guo ren, wo hen hao")  res0: scala.collection.immutable.SortedMap[String,Int] = Map(guo -> 1, hao -> 1, hen -> 1, ren, -> 1, shi -> 1, wo -> 2, zhong -> 1) |

**8、 重复前一个练习，使用java.util.TreeMap进行实现，并使之适用于Scala API**

**解：**

|  |
| --- |
| scala> import scala.collection.JavaConversions.mapAsScalaMap  import scala.collection.JavaConversions.mapAsScalaMap  scala> def wordCount(str:String)={  | var count: scala.collection.mutable.Map[String, Int] = new java.util.TreeMap[String, Int]  | for(word <- str.split("\\s+"))  | count += (word -> (count.getOrElse(word,0) + 1))  | count  | }  wordCount: (str: String)scala.collection.mutable.Map[String,Int]  scala> wordCount("wo shi zhong guo ren, wo hen hao")  res1: scala.collection.mutable.Map[String,Int] = Map(guo -> 1, hao -> 1, hen -> 1, ren, -> 1, shi -> 1, wo -> 2, zhong -> 1) |

**9、在REPL中打印出所有Java系统属性的表格，需要有格式，如下：**

****

**解：**

|  |
| --- |
| scala> import scala.collection.JavaConversions.\_  import scala.collection.JavaConversions.\_  scala> val props:scala.collection.Map[String,String] = System.getProperties()  props: scala.collection.Map[String,String] =  Map(env.emacs -> "", java.runtime.name -> Java(TM) SE Runtime Environment, sun.boot.library.path -> /Library/Java/JavaVirtualMachines/jdk1.8.0\_131.jdk/Contents/Home/jre/lib, java.vm.version -> 25.131-b11, user.country.format -> CN, gopherProxySet -> false, java.vm.vendor -> Oracle Corporation, java.vendor.url -> http://java.oracle.com/, path.separator -> :, java.vm.name -> Java HotSpot(TM) 64-Bit Server VM, file.encoding.pkg -> sun.io, user.country -> US, sun.java.launcher -> SUN\_STANDARD, sun.os.patch.level -> unknown, java.vm.specification.name -> Java Virtual Machine Specification, user.dir -> /Users/wuyufei, java.runtime.version -> 1.8.0\_131-b11, java.awt.graphicsenv -> sun.awt.CGraphicsEnvironment, java.endorsed.dirs -> /Library/Java/Jav...  scala> val keyLengths = for( key <- props.keySet ) yield key.length  keyLengths: scala.collection.Set[Int] = Set(10, 25, 14, 20, 29, 28, 21, 9, 13, 17, 12, 7, 18, 16, 11, 26, 23, 8, 19, 15)  scala> val maxKeyLength = keyLengths.max  maxKeyLength: Int = 29  scala> for(key <- props.keySet){  | print(key)  | print(" " \* (maxKeyLength - key.length))  | print(" | ")  | println(props(key))  | } |

**10、编写一个函数 minmax(values:Array[Int]), 返回数组中最小值和最大值的对偶**

**解：**

|  |
| --- |
| scala> def minmax(values:Array[Int])={  | (values.min,values.max)  | }  minmax: (values: Array[Int])(Int, Int)  scala> val ints = Array(1,2,3,4,5,6)  ints: Array[Int] = Array(1, 2, 3, 4, 5, 6)  scala> minmax(ints)  res4: (Int, Int) = (1,6) |

**11、编写一个函数indexes，给定字符串，产出一个包含所有字符下标的映射。举例来说：indexes(“Mississippi”)应返回一个映射，让’M’对应集{0}，‘i’对应集{1，4，7，10}，依次类推。使用字符到可变集的映射，注意下标的集应该是有序的。**

**解：**

|  |
| --- |
| scala> import scala.collection.mutable.\_  import scala.collection.mutable.\_  scala> def indexes(str:String):Map[Char,SortedSet[Int]]={  | var map = new HashMap[Char,SortedSet[Int]]()  | var i =0  | str.foreach{  | c =>  | map.get(c) match {  | case Some(result) => map(c) = result + i  | case None => map += (c-> SortedSet{i})  | }  | i+=1  | }  | map  | }  indexes: (str: String)scala.collection.mutable.Map[Char,scala.collection.mutable.SortedSet[Int]]  scala> indexes("Mississippi")  res5: scala.collection.mutable.Map[Char,scala.collection.mutable.SortedSet[Int]] = Map(M -> TreeSet(0), s -> TreeSet(2, 3, 5, 6), p -> TreeSet(8, 9), i -> TreeSet(1, 4, 7, 10)) |

**12、编写一个函数，从一个整型链表中去除所有的零值。**

**解：**

|  |
| --- |
| scala> import scala.collection.mutable.\_  import scala.collection.mutable.\_  scala> def removeZero(nums : List[Int]):List[Int]={  | nums.filter(\_ != 0)  | }  removeZero: (nums: List[Int])List[Int]  scala> removeZero(List(3,4,5,6,0,7,0,0))  res6: List[Int] = List(3, 4, 5, 6, 7) |

**13、编写一个函数，接受一个字符串的集合，以及一个从字符串到整数值的映射。返回整形的集合，其值为能和集合中某个字符串相对应的映射的值。举例来说，给定Array(“Tom”,”Fred”,”Harry”)和Map(“Tom”->3,”Dick”->4,”Harry”->5)，返回Array(3,5)。提示：用flatMap将get返回的Option值组合在一起。**

**解：**

|  |
| --- |
| scala> import scala.collection.mutable.\_  import scala.collection.mutable.\_  scala> def strMap(strArr: Array[String], map:Map[String,Int]): Array[Int] = {  | strArr.flatMap(map.get(\_))  | }  strMap: (strArr: Array[String], map: scala.collection.mutable.Map[String,Int])Array[Int]  scala> val a = Array("Tom","Fred","Harry")  a: Array[String] = Array(Tom, Fred, Harry)  scala> val m = Map("Tom"->3,"Dick"->4,"Harry"->5)  m: scala.collection.mutable.Map[String,Int] = Map(Tom -> 3, Dick -> 4, Harry -> 5)  scala> strMap(a,m).mkString(",")  res7: String = 3,5 |

**14、实现一个函数，作用与mkStirng相同，提示：使用reduceLeft实现试试**

**解：**

|  |
| --- |
| scala> import scala.collection.mutable.\_  import scala.collection.mutable.\_  scala> def myMakeString(set : Iterable[String]):String={  | if(set!=Nil) set.reduceLeft(\_ + \_) else ""  | }  myMakeString: (set: scala.collection.mutable.Iterable[String])String  scala> val setString = HashSet[String]("a","b","c")  setString: scala.collection.mutable.HashSet[String] = Set(c, a, b)  scala> myMakeString(setString)  res8: String = cab |

**15、给定整型列表lst，(lst :\ List[Int]())(\_ :: \_)得到什么? (List[Int]() /: lst)(\_ :+ \_)又得到什么？如何修改他们中的一个，以对原列表进行反向排列?**

**解：**

|  |
| --- |
| scala> import scala.collection.mutable.\_  import scala.collection.mutable.\_  scala> val lst = List(1,2,3,4,5)  lst: List[Int] = List(1, 2, 3, 4, 5)  scala> println((lst:\ List[Int]())(\_ :: \_))  List(1, 2, 3, 4, 5)  scala> println((List[Int]() /: lst)(\_ :+ \_))  List(1, 2, 3, 4, 5)  scala> println((List[Int]() /: lst)((a,b)=> b :: a))  List(5, 4, 3, 2, 1) |

**16、编写一个函数，将Double数组转换成二维数组。传入列数作为参数。距离来说，传入Array(1,2,3,4,5,6)和3列，返回Array(Array(1,2,3), Array(4,5,6))。**

**解：**

|  |
| --- |
| scala> def divArr(arr:Array[Int], i:Int)={  | arr.grouped(i).toArray  | }  divArr: (arr: Array[Int], i: Int)Array[Array[Int]]  scala> val arr = Array(1,2,3,4,5,6,7)  arr: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7)  scala> divArr(arr,3).foreach(a => println(a.mkString(",")))  1,2,3  4,5,6  7 |

**17【选做】\*通过par编写一个并行的wordcount程序，统计字符串中每个字符的出现频率。可以通过并行集合的aggregate函数试试**

**解：**

|  |
| --- |
| scala> import scala.collection.mutable.\_  import scala.collection.mutable.\_  scala> var str = "aaabbccdddaaa"  str: String = aaabbccdddaaa  scala> val parWordCount = str.par.aggregate(Map[Char,Int]())(  | { (a,b) => a + (b -> (a.getOrElse(b,0)+1)) },  | { (map1,map2)=>  | (map1.keySet ++ map2.keySet).foldLeft(Map[Char,Int]()){  | (result,k) => result + (k -> (map1.getOrElse(k,0) + map2.getOrElse(k,0)))  | }  | }  | )  parWordCount: scala.collection.mutable.Map[Char,Int] = Map(b -> 2, d -> 3, a -> 6, c -> 2) |

# 模式匹配

**1、利用模式匹配，编写一个swap函数，接受一个整数的对偶，返回对偶的两个组成部件互换位置的新对偶**

|  |
| --- |
| scala> def swap[S,T](tup: (S,T))={  | tup match{  | case (a,b) => (b,a)  | }  | }  swap: [S, T](tup: (S, T))(T, S)  scala> println(swap[String,Int](("1",2)))  (2,1) |

**2、利用模式匹配，编写一个swap函数，交换数组中的前两个元素的位置，前提条件是数组长度至少为2**

|  |
| --- |
| scala> def swap(array:Array[Any])={  | array match{  | case Array(first,second,rest @\_\*)=> Array(second,first)++rest  | case \_ => array  | }  | }  swap: (array: Array[Any])Array[Any]  scala> println(swap(Array("1","2","3","4")).mkString)  2134 |

**3、编写计算Item价格的通用函数。**

**abstract class** Item  
**case class** Article(description: String, price: Double) **extends** Item  
**case class** Bundle(description: String, discount: Double, items: Item\*) **extends** Item  
  
**val** special = *Bundle*(**"Father's day special"**, 20.0,  
 *Article*(**"Scala for the Impatient"**, 39.95),  
 *Bundle*(**"Anchor Distillery Sampler"**, 10.0,  
 *Article*(**"Old Potrero Straight Rye Whiskey"**, 79.95),  
 *Article*(**"Junípero Gin"**, 32.95)))

**def** price(it: Item): Double = it **match** {  
 **case** *Article*(\_, p) => p  
 **case** *Bundle*(\_, disc, its @ \_\*) => its.map(price \_).sum - disc  
}

**4、通过case类制作一颗二叉树。我们不妨从二叉树开始。**  
**sealed abstract class BinaryTree  
case class Leaf(value : Int) extends BinaryTree  
case class Node(left : BinaryTree,right : BinaryTree) extends BinaryTree**  
**编写一个通用函数计算所有叶子节点中的元素之和。**

**sealed abstract class BinaryTree**

**case class Leaf(value : Int) extends BinaryTree**

**case class Node(left: BinaryTree, right:BinaryTree) extends BinaryTree**

**def leafSum(tree:BinaryTree):Int={**

**tree match {**

**case Node(a,b) => leafSum(a) + leafSum(b)**

**case Leaf(v) => v**

**}**

**}**

**val r = Node(Leaf(3),Node(Leaf(3),Leaf(9)))**

**println(leafSum(r))**

**5、扩展前一个练习中的树，使得每个节点可以有任意多的后代，并重新实现计算函数。完成下面这棵树的叶子节点元素统计：Node(Node(Leaf(3),Leaf(8)),Leaf(2),Node(Leaf(5)))**

**sealed abstract class BinaryTree**

**case class Leaf(value: Int) extends BinaryTree**

**case class Node(tr: BinaryTree\*) extends BinaryTree**

**def leafSum(tree:BinaryTree):Int={**

**tree match {**

**case Node(r @\_\*) => r.map(leafSum).sum**

**case Leaf(v) => v**

**}**

**}**

**val r = Node(Node(Leaf(3), Leaf(8)), Leaf(2), Node(Leaf(5)))**

**println(leafSum(r))**

**6、扩展前一个练习中的树，使得每个非叶子节点除了后代之外，能够存放一个操作符。然后编写一个eval函数来计算它的值。举例来说：**

**上面这棵树的值为(3 \* 8) + 2 + (-5) = 21**  
**新的case类如下：**

**sealed abstract class BinaryTree**

**case class Leaf(value: Int) extends BinaryTree**

**case class Node(op: Char, leafs: BinaryTree\*) extends BinaryTree**

**完成下面的树的计算：**

**Node('+', Node('\*', Leaf(3), Leaf(8)), Leaf(2), Node('-', Leaf(5)))**

**sealed abstract class BinaryTree**

**case class Leaf(value: Int) extends BinaryTree**

**case class Node(op: Char, leafs: BinaryTree\*) extends BinaryTree**

**def eval(tree:BinaryTree):Int= {**

**tree match {**

**case Node(op, leafs@\_\*) => op match {**

**case '+' => leafs.map(eval \_).sum**

**case '-' => -leafs.map(eval \_).sum**

**case '\*' => leafs.map(eval \_).product**

**}**

**case Leaf(x) => x**

**}**

**}**

**val x = Node('+', Node('\*', Leaf(3), Leaf(8)), Leaf(2), Node('-', Leaf(5)))**

**println(x)**

**println(eval(x))**

**7、编写一个函数，计算List[Option[Int]]中所有非None值之和。不得使用match语句。**

**def sum(lst: List[Option[Int]]) = lst.map(\_.getOrElse(0)).sum**

**val x = List(Some(1), None, Some(2), None, Some(3))**

**println(sum(x))**

# 高阶函数

**1、编写一个compose函数，将两个类型为Double=>Option[Double]的函数组合在一起，产生另一个同样类型的函数。如果其中一个函数返回None，则组合函数也应返回None。例如：**  
**def f(x : Double) = if ( x >= 0) Some(sqrt(x)) else None  
def g(x : Double) = if ( x != 1) Some( 1 / ( x - 1)) else None  
val h = compose(f,g)  
h(2)将得到Some(1)，而h(1)和h(0)将得到None**

**def compose(f:Double=>Option[Double],g:Double=>Option[Double])={**

**(x : Double) =>**

**if (f(x) == None || g(x) == None) None**

**else g(x)**

**}**

**import scala.math.sqrt**

**def f(x : Double) = if ( x >= 0) Some(sqrt(x)) else None**

**def g(x : Double) = if ( x != 1) Some( 1 / ( x - 1)) else None**

**val h = compose(f,g)**

**println(h(2))**

**2、编写函数values(fun:(Int)=>Int,low:Int,high:Int),该函数输出一个集合，对应给定区间内给定函数的输入和输出。比如，values(x=>x\*x,-5,5)应该产出一个对偶的集合(-5,25),(-4,16),(-3,9),…,(5,25)**

**def values(fun:(Int)=>Int,low:Int,high:Int) ={**

**var array = List[(Int,Int)]()**

**low to high foreach {**

**x =>**

**array = (x, fun(x)) :: array**

**}**

**array**

**}**

**println(values(x => x \* x, -5, 5).mkString)**

**3、如何用reduceLeft得到数组Array(1,333,4,6,4,4,9,32,6,9,0,2)中的最大元素?**

**val arr = Array(1,333,4,6,4,4,9,32,6,9,0,2)**

**print(arr.reduceLeft((l,r)=>if(l>=r) l else r))**

**4、用to和reduceLeft实现阶乘函数,不得使用循环或递归**

**def factorial(n:Int): Unit ={**

**1 to n reduceLeft(\_ \* \_)**

**}**

**5、编写函数largest(fun:(Int)=>Int,inputs:Seq[Int]),输出在给定输入序列中给定函数的最大值。举例来说，largest(x=>10x-xx,1 to 10)应该返回25.不得使用循环或递归**

**def largest1(fun:(Int)=>Int, inputs:Seq[Int]) = inputs.foldLeft(1)((a,b)=> if(fun(b)>a) fun(b) else a)**

**def largest2(fun:(Int)=>Int, inputs:Seq[Int]) = inputs.map(fun(\_)).max**

**println(largest1(x => 10 \* x - x \* x, 1 to 10))**

**println(largest2(x => 10 \* x - x \* x, 1 to 10))**

**6【选做】、修改前一个函数，返回最大的输出对应的输入。举例来说,largestAt(fun:(Int)=>Int,inputs:Seq[Int])应该返回5。不得使用循环或递归**

**def largestAt1(fun:(Int)=>Int, inputs:Seq[Int]) = inputs.reduce((a,b)=> if(fun(b)>fun(a)) b else a)**

**def largestAt2(fun: (Int) => Int, inputs: Seq[Int]) = inputs.map(x => (x, fun(x))).reduceLeft((x,y) => if (x.\_2 > y.\_2) x else y).\_1**

**println(largestAt1(x => 10 \* x - x \* x, 1 to 10))**

**println(largestAt2(x => 10 \* x - x \* x, 1 to 10))**

**7、要得到一个序列的对偶很容易，比如:**  
**val pairs = (1 to 10) zip (11 to 20)**

**编写函数adjustToPair,该函数接受一个类型为(Int,Int)=>Int的函数作为参数，并返回一个等效的, 可以以对偶作为参数的函数。举例来说就是:adjustToPair(\_\*\_)((6,7))应得到42。然后用这个函数通过map计算出各个对偶的元素之和**

**def ajustToPair(fun: (Int, Int) => Int) = (x: (Int, Int)) => fun(x.\_1, x.\_2)**

**val x = ajustToPair(\_ \* \_)((6, 7))**

**println(x)**

**val pairs = (1 to 10) zip (11 to 20)**

**println(pairs)**

**val y = pairs.map(ajustToPair(\_ + \_))**

**println(y)**

**8、实现一个unless控制抽象，工作机制类似if,但条件是反过来的。**

**def unless(condition: => Boolean)(block: => Unit) { if (!condition) { block } }**

**unless (0 > 1) { println("Unless!") }**

# 类

**1、编写一个Time类，加入只读属性hours和minutes，和一个检查某一时刻是否早于另一时刻的方法before(other:Time):Boolean。Time对象应该以new Time(hrs,min)方式构建。**

**class Time(val hrs:Int, val min:Int){**

**val hours = hrs**

**val minutes = min**

**def before(other:Time):Boolean = {**

**if(hours<other.hours)**

**true**

**if(hours==other.hours)**

**if(minutes<other.minutes)**

**true**

**false**

**}**

**}**

**2、创建一个Student类，加入可读写的JavaBeans属性name(类型为String)和id(类型为Long)。有哪些方法被生产？(用javap查看。)你可以在Scala中调用JavaBeans的getter和setter方法吗？**

**import scala.beans.BeanProperty**

**class Student{**

**@BeanProperty var name:String = \_**

**@BeanProperty var id:Long = \_**

**}**

**/\*\***

**javap -c Student 后显示如下**

**Compiled from "Student.scala"**

**public class Student extends java.lang.Object implements scala.ScalaObject{**

**public java.lang.String name();**

**Code:**

**0: aload\_0**

**1: getfield #13; //Field name:Ljava/lang/String;**

**4: areturn**

**public void name\_$eq(java.lang.String);**

**Code:**

**0: aload\_0**

**1: aload\_1**

**2: putfield #13; //Field name:Ljava/lang/String;**

**5: return**

**public void setName(java.lang.String);**

**Code:**

**0: aload\_0**

**1: aload\_1**

**2: putfield #13; //Field name:Ljava/lang/String;**

**5: return**

**public long id();**

**Code:**

**0: aload\_0**

**1: getfield #19; //Field id:J**

**4: lreturn**

**public void id\_$eq(long);**

**Code:**

**0: aload\_0**

**1: lload\_1**

**2: putfield #19; //Field id:J**

**5: return**

**public void setId(long);**

**Code:**

**0: aload\_0**

**1: lload\_1**

**2: putfield #19; //Field id:J**

**5: return**

**public long getId();**

**Code:**

**0: aload\_0**

**1: invokevirtual #25; //Method id:()J**

**4: lreturn**

**public java.lang.String getName();**

**Code:**

**0: aload\_0**

**1: invokevirtual #28; //Method name:()Ljava/lang/String;**

**4: areturn**

**public Student();**

**Code:**

**0: aload\_0**

**1: invokespecial #34; //Method java/lang/Object."<init>":()V**

**4: return**

**}**

**\*\*/**

# 对象

**1、编写一个Conversions对象，加入 inchesToCentimeters,gallonsToLiters 和 milesToKilometers 方法**

**object Conversions{**

**def inchesToSantimeters(value: Double) = value \* 2.54**

**def gallonsToLiters(value: Double) = value \* 3.78541178**

**def milesToKilometers(value: Double) = value \* 1.609344**

**}**

**2、定义一个 Point 类和一个伴生对象,使得我们可以不用 new 而直接用 Point(3,4)来构造 Point 实例 apply 方法的使用**

**class Point(x: Int = 0, y: Int = 0) extends java.awt.Point(x, y)**

**object Point {**

**def apply(x: Int = 0, y: Int = 0) = new Point(x, y)**

**}**

**3、编写一个 Scala 应用程序,使用 App 特质,以反序打印命令行参数,用空格隔开。举例来说,scala Reverse Hello World 应该打印 World Hello**

**object Reverse extends App{**

**args.reverse.mkString(" ")**

**}**

**4、编写一个扑克牌 4 种花色的枚举,让其 toString 方法分别返回♣,♦,♥,♠，并实现一个函数,检查某张牌的花色是否为红色**

**object Suits extends Enumeration{**

**type Suits = Value**

**val Spade = Value("♠")**

**val Club = Value("♣")**

**val Heart = Value("♥")**

**val Diamond = Value("♦")**

**override def toString():String={**

**Suits.values.mkString(",")**

**}**

**def isRed(card: Suits) = card == Heart || card == Diamond**

**}**

# 包和引入

**1、练习使用包的各种声明方式，并查看他们的不同**

**2、编写一段程序,将Java哈希映射中的所有元素拷贝到Scala哈希映射。用引入语句重命名这两个类。**

**object Q6 extends App{**

**import java.util.{HashMap => JavaHashMap}**

**import collection.mutable.{HashMap => ScalaHashMap, Map => ScalaMap}**

**val javaMap = new JavaHashMap[Int,String]**

**javaMap.put(1, "One");**

**javaMap.put(2, "Two");**

**javaMap.put(3, "Three");**

**javaMap.put(4, "Four");**

**val scalaMap = new ScalaHashMap[Int,String]**

**for(key <- javaMap.keySet().toArray){**

**scalaMap += (key.asInstanceOf[Int] -> javaMap.get(key))**

**}**

**println(scalaMap.mkString(" "))**

**}**

# 继承

**1、扩展如下的BankAccount类，新类CheckingAccount对每次存款和取款都收取1美元的手续费**  
**class BankAccount(initialBalance:Double){  
 private var balance = initialBalance  
 def deposit(amount:Double) = { balance += amount; balance}  
 def withdraw(amount:Double) = {balance -= amount; balance}  
}**

**class BankAccount(initialBalance:Double){**

**private var balance = initialBalance**

**def deposit(amount:Double) = { balance += amount; balance}**

**def withdraw(amount:Double) = {balance -= amount; balance}**

**}**

**class CheckingAccount(initialBanlance:Double) extends BankAccount(initialBanlance){**

**override def deposit(amount:Double) = super.deposit(amount-1)**

**override def withdraw(amount:Double) = super.withdraw(amount+1)**

**}**

**2、扩展前一个练习的BankAccount类，新类SavingsAccount每个月都有利息产生(earnMonthlyInterest方法被调用)，并且有每月三次免手续费的存款或取款。在earnMonthlyInterest方法中重置交易计数。**

**class BankAccount(initialBalance:Double){**

**private var balance = initialBalance**

**def deposit(amount:Double) = { balance += amount; balance}**

**def withdraw(amount:Double) = {balance -= amount; balance}**

**}**

**class SavingsAccount(initialBalance:Double) extends BankAccount(initialBalance){**

**private var num:Int = \_**

**def earnMonthlyInterest()={**

**num = 3**

**super.deposit(1)**

**}**

**override def deposit(amount: Double): Double = {**

**num -= 1**

**if(num < 0) super.deposit(amount - 1) else super.deposit(amount)**

**}**

**override def withdraw(amount: Double): Double = {**

**num -= 1**

**if (num < 0) super.withdraw(amount + 1) else super.withdraw(amount)**

**}**

**}**

**3、定义一个抽象类Item,加入方法price和description。SimpleItem是一个在构造器中给出价格和描述的物件。利用val可以重写def这个事实。Bundle是一个可以包含其他物件的物件。其价格是打包中所有物件的价格之和。同时提供一个将物件添加到打包当中的机制，以及一个适合的description方法**

**import collection.mutable.ArrayBuffer**

**abstract class Item{**

**def price():Double**

**def description():String**

**override def toString():String={**

**"description:" + description() + " price:" + price()**

**}**

**}**

**class SimpleItem(val price:Double,val description:String) extends Item{**

**}**

**class Bundle extends Item{**

**val items = new ArrayBuffer[Item]()**

**def addItem(item:Item){**

**items += item**

**}**

**def price(): Double = {**

**var total = 0d**

**items.foreach(total += \_.price())**

**total**

**}**

**def description(): String = {**

**items.mkString(" ")**

**}**

**}**

**4、设计一个Point类，其x和y坐标可以通过构造器提供。提供一个子类LabeledPoint，其构造器接受一个标签值和x,y坐标,比如:new LabeledPoint(“Black Thursday”,1929,230.07)**

|  |
| --- |
| **class Point(x:Double, y:Double)**  **class LabeledPoint(x:Double, y:Double, tag:String) extends Point(x,y)** |

**5、定义一个抽象类Shape，一个抽象方法centerPoint，以及该抽象类的子类Rectangle和Circle。为子类提供合适的构造器，并重写centerPoint方法**

**abstract class Shape{**

**def centerPoint()**

**}**

**class Rectangle(startX:Int,startY:Int,endX:Int,endY:Int) extends Shape{**

**def centerPoint() {}**

**}**

**class Circle(x:Int,y:Int,radius:Double) extends Shape{**

**def centerPoint() {}**

**}**

**6、提供一个Square类，扩展自java.awt.Rectangle并且是三个构造器：一个以给定的端点和宽度构造正方形，一个以(0,0)为端点和给定的宽度构造正方形，一个以(0,0)为端点,0为宽度构造正方形**

**import java.awt.{Point, Rectangle}**

**class Square(point:Point,width:Int) extends Rectangle(point.x,point.y,width,width){**

**def this(){**

**this(new Point(0,0),0)**

**}**

**def this(width:Int){**

**this(new Point(0,0),width)**

**}**

**}**

# 特质

**1、java.awt.Rectangle类有两个很有用的方法translate和grow,但可惜的是像java.awt.geom.Ellipse2D这样的类没有。在Scala中，你可以解决掉这个问题。定义一个RenctangleLike特质,加入具体的translate和grow方法。提供任何你需要用来实现的抽象方法,以便你可以像如下代码这样混入该特质:  
val egg = new java.awt.geom.Ellipse2D.Double(5,10,20,30) with RectangleLike  
egg.translate(10,-10)  
egg.grow(10,20)**

**import java.awt.geom.Ellipse2D**

**trait RectangleLike{**

**this:Ellipse2D.Double=>**

**def translate(x:Double,y:Double){**

**this.x = x**

**this.y = y**

**}**

**def grow(x:Double,y:Double){**

**this.x += x**

**this.y += y**

**}**

**}**

**object Test extends App{**

**val egg = new Ellipse2D.Double(5,10,20,30) with RectangleLike**

**println("x = " + egg.getX + " y = " + egg.getY)**

**egg.translate(10,-10)**

**println("x = " + egg.getX + " y = " + egg.getY)**

**egg.grow(10,20)**

**println("x = " + egg.getX + " y = " + egg.getY)**

**}**

# 类型参数

**1、定义一个不可变类Pair[T,S], 带一个swap方法，返回组件交换过位置的新对偶**

**class Pair[T,S](val t:T,val s:S){**

**def swap() = new Pair(s,t)**

**}**

2、**定义一个可变类Pair[T]，带一个swap方法，交换对偶中组件的位置。**

**class Pair[T](val s:T,val t:T){**

**def swap() = new Pair(t,s)**

**}**

**3、给定类Pair[T, S] ，编写一个泛型方法swap，接受对偶作为参数并返回组件交换过位置的新对偶。**

**class Pair[T,S](val t:T, val s:S){**

**def swap[T,S](t:T,s:S) = new Pair(s,t)**

**}**

**4、编写一个泛型方法middle，返回任何Iterable[T]的中间元素。举例来说，middle(“World”)应得到’r’。**

**def middle[T](iter:Iterable[T]):T={**

**val seq = iter.toArray**

**seq(seq.length/2)**

**}**

**5、给定可变类Pair[S,T]，使用类型约束定义一个swap方法，当类型参数相同时可以被调用。**

**class Pair[S,T](val s:S, val t:T){**

**def swap(implicit env: S =:= T) = new Pair(t,s)**

**}**

# 文件和正则表达式

**1、编写一小段Scala代码，将某个文件中的行倒转顺序(将最后一行作为第一行,依此类推)**

**val path = "./exercise01.txt"**

**val file = Source.fromFile(path)**

**val reverseLines = file.getLines().toArray.reverse**

**val pw = new PrintWriter(path)**

**reverseLines.foreach (line => pw.write(line+"\n"))**

**pw.close()**

**2、编写Scala程序打印出某个网页中所有img标签的src属性。使用正则表达式和分组**

**val pattern = """<img[^>]+(src\s\*=\s\*"[^>^"]+")[^>]\*>""".r**

**val source = scala.io.Source.fromURL("http://www.vernonzheng.com","utf-8").mkString**

**for (pattern(str) <- pattern.findAllIn(source)) println(str)**

# 高级类型

**1、实现一个Bug类，对沿着水平线爬行的虫子建模。move方法向当前方向移动，turn方法让虫子转身，show方法打印出当前的位置。让这些方法可以被串接调用。例如：  
bugsy.move(4).show().move(6).show().turn().move(5).show()  
上述代码应显示4 10 5。**

**package \_1801 {**

**class Bug(var pos: Int = 0) {**

**var forword: Int = 1**

**def move(up: Int):this.type = {**

**pos += forword \* up**

**this**

**}**

**def show():this.type = {**

**print(pos + " ")**

**this**

**}**

**def turn():this.type = {**

**forword = -forword**

**this**

**}**

**}**

**class Test extends App {**

**val bugsy = new Bug**

**bugsy.move(4).show().move(6).show().turn().move(5).show()**

**}**

**}**

2、**为前一个练习中的Bug类提供一个流利接口，达到能编写如下代码的效果：  
bugsy move 4 and show and then move 6 and show turn around move 5 and show**

**package \_1802 {**

**//非动词 non-verb**

**object then**

**object show**

**object around**

**class Bug(var pos: Int = 0) {**

**var forword: Int = 1**

**def move(num: Int): this.type = { pos += num; this }**

**def and(obj: then.type): this.type = this**

**def and(obj: show.type): this.type = { print(pos + " "); this}**

**def turn(obj: around.type): this.type = { pos = 0; this}**

**}**

**class Test extends App {**

**val bugsy = new Bug**

**bugsy move 4 and show and then move 6 and show turn around move 5 and show**

**}**

**}**

**3、实现一个方法，接受任何具备如下方法的类的对象和一个处理该对象的函数。  
调用该函数，并在完成或有任何异常发生时调用close方法。**  
**def close(): Unit**

**def tryWithClose[T<:{def close():Unit}](obj:T,func: T => Unit)={**

**try{**

**func(obj)**

**}finally {**

**obj.close()**

**}**

**}**

4、**编写一个函数printValues，带有三个参数f、from和to，打印出所有给定区间范围内的输入值经过f计算后的结果。这里的f应该是任何带有接受Int产出Int的apply方法的对象。例如：  
printValues((x: Int) => x\*x, 3, 6) //将打印 9 16 25 36  
printValues(Array(1, 1, 2, 3, 5, 8, 13, 21, 34, 55), 3, 6) //将打印 3 5 8 13**

**def printValues(f:{def apply(param:Int):Int}, from:Int, to:Int)={**

**for(i <- from to to) {**

**print(f.apply(i) + " ")**

**}**

**}**

**printValues((x: Int) => x\*x, 3, 6) //将打印 9 16 25 36**

**printValues(Array(1, 1, 2, 3, 5, 8, 13, 21, 34, 55), 3, 6) //将打印 3 5 8 13**