  
第8章 Scala—大數據的程式語言



### Scala程式手冊

* 請參考：

http://www.scala-lang.org/api/2.10.5/#package

### Scala REPL

|  |  |
| --- | --- |
| 指令 | 說明 |
| spark-shell | Read-Eval-Print Loop |

### 基礎

|  |  |
| --- | --- |
| 指令 | 說明 |
| println("Hello, world of Scala!") | 列印一行文字 |
| 1 + 1 | 整數相加 |
| res1 + 1 | 變數加1 |
| 1.0 + 1.0 | 浮點數相加 |
| "A" + "B" | 字串相加 |
| "1" + "2" | 字串相加 |
| 1.toString + 1.toString | 轉成字串再相加 |
| "1".toInt + "2".toInt  "1".toFloat + "2".toFloat | 將字串轉成整數或浮點數再運算 |
| 1.0 + 2.0  1.0.+(2.0) | 物件的方法 |
| 1.+(2)  (1).+(2) | 避免誤判為浮點數 |
| var a=5  a=6  val b=5  b=6 | var和val |
| val c={  val i=0; val j=i+5  val k=i+j  val m=5\*k+j  m  } | 程式區塊賦值 |

### 基本的資料型態

|  |  |
| --- | --- |
| 指令 | 說明 |
| val a1=5 // 32位元整數  val a2=5.0 // 64位元雙精度浮點數  val a3=6.23e10  val b=true // 布林邏輯值  val c='A' // 字元  val s="Hello!" // 字串 | 型態推斷（type inference） |
| val f1=5F // 32位元單精度浮點數  val f2:Float=5  val d1:Short=5 // 16位元短整數  val d2:Long=5 // 64位元長整數 | 用：來指定資料的型態 |
| println("Short: " + scala.Short.MinValue + " ~ " + scala.Short.MaxValue) // 短整數的範圍  println("Int: " + scala.Int.MinValue + " ~ " + scala.Int.MaxValue) // 整數的範圍  println("Long: " + scala.Long.MinValue + " ~ " + scala.Long.MaxValue) // 常整數的範圍  println("Float: " + scala.Float.MinValue + " ~ " + scala.Float.MaxValue) // 單精度浮點數的範圍  println("Double: " + scala.Double.MinValue + " ~ " + scala.Double.MaxValue) // 雙精度浮點數的範圍 | 數的範圍 |

### 資料集Collections

|  |  |
| --- | --- |
| 指令 | 說明 |
| 不可變陣列Array | |
| val arr1=new Array(5)  val arr2=new Array[Int](5)  val arr3=new Array[Float](5)  val arr4=Array(1,2,3,4,5)  val arr5=Array("A","B","C","D","E")  arr5(0)  arr5(0)="XYZ"  arr5 | 陣列長度固定，一經宣告建立無法改變  元素資料型態相同，但陣列元素資料的值可以改變 |
| val arr2Dim=Array(Array(1,2), Array(3,4))  arr2Dim(0)(0) // 使用二個位置索引值來指出陣列元素  arr2Dim(0)(1) | 二維陣列 |
| val arr6=Array.ofDim[Int](5) // 宣告一維整數陣列(5個元素)  val arr7=Array.ofDim[Int](2,2) // 宣告2\*2的二維整數陣列 | 使用陣列物件宣告陣列的方法 |
| val a=Array(1,2)  val b=Array(3,4)  a ++ b  val c= a ++ b | 二個 Array 的合併 |
| 可變陣列ArrayBuffer | |
| val arrBuff=scala.collection.mutable.ArrayBuffer(1,2,3) // 宣告可變陣列arrBuff，元素內容1,2,3 | 可變陣列宣告 |
| import scala.collection.mutable.\_  val arrBuff= ArrayBuffer(1,2,3) | 也可以這樣宣告 |
| arrBuff += 4 // 加入元素4  arrBuff -= 4 // 去掉元素4  arrBuff ++= Array(11,12,13,14,15) // 把不可變陣列併入到arrBuff  arrBuff.insert(0, 100,101,102) // 從0的索引位置插入100,101,102  arrBuff  arrBuff.remove(0, 3) // 從0的位置開始去除3個元素  arrBuff  arrBuff.trimEnd(5) // 去除arrBuff最後面的5個元素  arrBuff | 可變陣列操作範例 |
| 不可變映射Map | |
| val stdMap1=Map("ID01"->"曾子","ID02"->"子路","ID03"->"顏回","ID04"->"宰予","ID05"->"子貢")  stdMap1("ID01") | 不可變map的宣告與使用 |
| stdMap1("ID01")="子騫"  stdMap1("ID01") |  |
| 可變映射Map | |
| val stdMap2= scala.collection.mutable.Map(1->"曾子",2->"子路",3->"顏回",4->"宰予",5->"子貢")  stdMap2(1)  stdMap2(1)="蘇秦"  stdMap2(1)  stdMap2 += (6->"子騫") // 增加一組映射關係  stdMap2 -= 2 // 把鍵為2個關係刪除 | 可變map的宣告與使用 |
| stdMap2.getOrElse(5, "沒這個人") // 查詢鍵5的學生，可以查得到對應值  stdMap2.getOrElse(15, "沒這個人") // 查詢鍵15的學生，沒有對應值，返回 "沒這個人"，且不會產生程式錯誤 | 用getOrElse查詢Map |
| 不可變列表List | |
| val list1=List()  val list2=Nil  list1 == list2  val list3=List(6,7,8,9,10)  val list4=list3 :: 5  val list4=5 :: list3  val list5=1 :: 2 :: 3 :: 4 :: list4  list3.::(5) | 不可變列表的使用 |
| list5.head  list5.tail | 列表的頭、尾 |
| val list1=List(1,2,3)  val list2=List(4,5,6)  val list3=list1 :: list2  val list4=list1.foldRight(list2)((v1, v2)=>v1 :: v2) | 二個list的連接 |
| 可變列表ListBuffer | |
| val lstBuf=scala.collection.mutable.ListBuffer(1,2,3)  lstBuf += 4  lstBuf += (5,6)  val list1=List(7,8,9)  lstBuf ++= list1  lstBuf -= (4,5,6)  lstBuf --= list1 | 可變列表的宣告與使用 |
| 不可變Set | |
| val set1=Set(1,1,2,2,3,3)  val set2=set1 + 4  val set3=set2 + (5,6,7)  val set4=set3 ++ List(8,9,10)  val set5=set4 - 10  val set6=set5 - (9,8,7)  val set7=set6 -- List(6,5,4) | 元素無順序，且不可重複 |
| 可變Set | |
| val mutSet=scala.collection.mutable.Set(1,2,3)  mutSet += 4  mutSet += (5,6,7)  mutSet ++= Set(8,9,10)  mutSet -= 10  mutSet -= (9,8,7)  mutSet --= Set(6,5,4) | 可變Set的宣告與使用 |
| Range | |
| 0 to 10  0.to(10)  val range1=Range(0, 10) | 0~10 |
| val range4=Range(0, 10, 2)  val range5=Range(10, 0, -2) | 間隔2和-2 |
| 0 until 10 | 0～9 |
| tuple(a tuple is not a collection) | |
| val teacTpl=(0,"孔子",70,"老師")  teacTpl.getClass  teacTpl.\_1  teacTpl.\_2  teacTpl.\_3  teacTpl.\_4  val stdArr=Array((1,"曾子"),(2,"子路"),(3,"顏回"),(4,"宰予"),(5,"子貢"))  stdArr(0)  stdArr(0).\_1  stdArr(0).\_2 | tuple的建立和讀取 |

### 條件分支if

|  |  |
| --- | --- |
| 指令 | 說明 |
| val x=2  if(x % 2 == 0) println("x是偶數") else println("x是基數")  if(x % 2 == 0) {  println("x是偶數")  } else {  println("x是基數")  }  if(x % 2 == 0) "x是偶數" else "x是基數" | if的用法 |
| val rlt1= if(x % 2 == 0) "x是偶數" else "x是基數"  val rlt2 = rlt1 + "，這是真的！" | if不只分支，還可以返回值 |
| val y=0  val rlt2 = if(y == 0) {  val i=0; val j=i+5  val k=i+j  val m=5\*k+j  m  } else {  100  } | 複雜一點的例子 |

### 迴圈

|  |  |
| --- | --- |
| 指令 | 說明 |
| while | |
| var rlt=0  var i=1  while(i<=10){  rlt += i  i += 1  }  println(rlt) | 1加到10的總和，用while |
| var rlt=0  var i=1  do {  rlt += i  i += 1  } while(i<=10)  println(rlt) | 用do while |
| var rlt =0  (1 to 10).foreach(x=> rlt += x)  println(rlt) | 補充：用foreach做加總 |
| for | |
| var rlt1=0  for(i <- 1 to 10) rlt1 += i  println(rlt1) | 簡單迴圈：1加到10的總和 |
| var rlt2=0  for(i<- 1 until 11) rlt2 += i  println(rlt2) | 補充：1加到10的總和，用until |
| var rlt1=0  for(i<- 1 to 10 if i % 2 == 0) rlt1 += i  println(rlt1) | for的if（守衛），偶數的總和 |
| for(i<-1 to 2; j<-1 to 3){  println("第"+i+"章，"+"第"+j+"節")  } | 雙層for迴圈 |
| for(i<-1 to 2; j<-1 to 3 if i>=j){  println("第"+i+"章，"+"第"+j+"節")  } | 加入守衛子句 |
| val even=for(i<- 1 to 10 if i % 2 == 0) yield i  for(i<-1 to 10) yield {  val j=if(i>5) i-1 else i+2  val k=i+0.5\*j  val l=k\*k+6  l  } | for也可以產生輸出值，用yield |
| 用for訪問資料集（collections） | |
| val arr5=Array("A","B","C","D","E")  for(c <- arr5) println(c) | 訪問陣列元素 |
| for(c <- arr5 if c!="C") println(c) | 補充：加入守衛子句 |
| val stdMap1=Map(1->"曾子",2->"子路",3->"顏回",4->"宰予",5->"子貢")  for((k,v) <- stdMap1) println("學號 "+k+" 的學生是 "+v) | 用for訪問Map元素 |
| val stdArr=Array((1,"曾子"),(2,"子路"),(3,"顏回"),(4,"宰予"),(5,"子貢"))  for(s<-stdArr) println(s+" 第1個值："+s.\_1+"，第二個值："+s.\_2) | 用for訪問tuple元素 |

### 函數式程式設計

|  |  |
| --- | --- |
| 指令 | 說明 |
| def addOne(x:Int) = {  x+1  }  val rlt1=addOne(1)  def addTwo(x:Int):Float = {  x+2  }  val rlt2=addTwo(2) | addTwo用型態推斷 |
| def addOne(x:Int) {  x+1  }  val rlt1=addOne(1) | 沒有返回值，rlt1也沒有結果 |
| def printStar(x:Int) {  println("\*" \* x)  }  printStar(10) | 函數沒有返回值，只有「副作用」 |
| var mySum=0  def addToMySum(x:Int) {  mySum += x  }  addToMySum(50)  mySum  addToMySum(100)  mySum | 不被鼓勵的函數「副作用」 |
| def variSum(elems:Int\*)={  var s=0  for(e <- elems) s += e  s  }  variSum(1,1)  variSum(1,2,3,4,5) | 可變長度參數 |
| def f1(x:Int) {  println("我沒有返回值！"\*x)  }  f1(5)  def f2 {  println("我不需要參數，也沒有返回值")  }  f2 | 補充f1和f2的型態都是Unit |
| 匿名函數 | |
| (x:Int)=>x+10  val f=(x:Int)=>x+10  f(1)  val arr=Array(1,2,3,4,5)  arr.map((x:Int)=>x+10)  arr.map(f) | 使用匿名函數 |
| 函數當參數 | |
| def addOne(x:Int)={x+1}  def addTwo(x:Int)={x+2}  def whichAdd(func:Int=>Int, x:Int)={func(x)}  whichAdd(addOne, 9)  whichAdd(addTwo, 9) | 用函數當參數傳遞 |

### 常用資料集處理方法

|  |  |
| --- | --- |
| 指令 | 說明 |
| val arr1=Array(1,2,3,4,5,6)  arr1.length  arr1.sum  arr1.max  arr1.min | 陣列統計量 |
| arr1.last  arr1.init  arr1.head  arr1.tail | 取陣列的頭尾部份 |
| val arr1=Array(1,2,3,4,5,6)  val arr5=arr1.slice(0,3) // 取陣列0～2（不含3）位置的元素  val arr6=arr1.drop(3) // 去掉陣列前3個元素  val arr7=arr5 ++ arr6 // 將arr5與arr6合併放入arr7  arr1.take(3) // 取陣列前三個元素  arr1.takeRight(3) // 取陣列後三個元素 | 陣列切割與合併 |
| val arr2=Array(2,2,2,1,1,1,3,3,3)  val arr3=arr2.distinct  val arr4=arr3.sortBy(x=>x)  arr4.reverse | 陣列元素的去重和排序 |
| val arr1=Array(1,2,3,4,5,6)  arr1.map(\_ \* 10)  arr1.filter(\_ >= 4)  arr1.count(\_ >= 4)  arr1.reduce(\_+\_)  arr1.reduce(\_\*\_)  arr1.filter(x => x % 2 == 0).map(\_ \* 10).reduce(\_+\_) | 陣列元素的資料操作 |
| val arr1=Array(1,2,3,4,5)  val arr2=Array("A","B","C","D","E")  val arr3=arr2.zip(arr1) | 二個array的zip |
| val list1=arr1.toList  val set1=arr1.toSet  val map1=arr3.toMap  val list2=map1.toList  val arr4=map1.toArray | collection型態的轉換 |
| val arr5=Array(1,2,3)  val arr6=Array(4,5,6)  val arr7=Array(arr5, arr6)  arr7.flatten | 平坦化 |
| arr7  arr7.map(\_.map(\_\*2)) | 複雜集合的map |
| foreach | |
| val arr8=Array(1,2,3,4,5)  arr8.foreach(println)  arr8.foreach(x=>println("\*" \* x))  arr8.foreach(x=>if(x<=3) println("@" \* x) else println("#" \* x)) | foreach範例 |
| var mySum=0  (1 to 10).foreach((x:Int)=> mySum += x)  mySum  val calSum=(x:Int)=> mySum += x  mySum=0  (1 to 10).foreach(calSum)  mySum | foreach一種副作用：將匿名函數指定給常量 |
| var mySum=0  def calSumFunc(x:Int)={mySum += x}  (1 to 10).foreach(calSumFunc)  mySum  (1 to 10).foreach(x=>calSumFunc(x))  mySum | 補充：foreach一種副作用：定義一個函數 |

### 模式匹配Pattern Matching

|  |  |
| --- | --- |
| 指令 | 說明 |
| def noteMatch(note: String) {  note match {  case "Do" => println("doe a deer a female deer")  case "Re" => println("ray a drop of golden sun")  case "Mi" => println("me a name I call myself")  case \_ => println("沒有對應資料喔！")  }  }  noteMatch("Re")  noteMatch("GG") | 簡單字串match |
| def noteMatch(note: Any) {  note match {  case x:String if x.length != 2 => println("長度錯誤喔!")  case x:Int => println("我不要整數，請輸入音符字串！")  case x:Array[String] => println("我不要陣列，請輸入音符字串！")  case "Do" => println("doe a deer a female deer")  case "Re" => println("ray a drop of golden sun")  case "Mi" => println("me a name I call myself")  }  }  noteMatch("Ree")  noteMatch(1)  noteMatch(Array("Do", "Re"))  noteMatch("Mi") | match的守衛（）  匹配型態 |
| val tpl1=(0,"傑克",20,"male")  val tpl2=(1,"珍妮佛",21,"female")  def opening(tpl: Any):String={  tpl match {  case (id, name, age, sex) => {  val title=if(sex=="male") " 先生" else " 小姐"  "親愛的 "+name+title+" 您好："  }  case \_ => "資料格式有問題"  }  }  opening(tpl1)  opening(tpl2)  opening("林先生") | 匹配複雜的資料格式 |

### 補充：物件

|  |  |
| --- | --- |
| 指令 | 說明 |
| class Adder {  private var v=0  def addOne() { v += 1 }  def addTen() { v += 10 }  def addWhatever(w:Int) { v += w }  def output = v  }  val myAdder = new Adder  myAdder.addOne  println(myAdder.output)  myAdder.addTen  println(myAdder.output)  myAdder.addWhatever(100)  println(myAdder.output) | 建立一個Adder 物件，及其方法 |