第3章

Scala 控制结构

Scala 程序中的语句默认是按照书写顺序依次被执行的，这样的语句结构就是顺序结构。仅有顺序结构还是不够的，因为有时人们需要根据特定的情况有选择地执行某些语句，这时就需要一种选择结构的语句。另外，有时人们还需要在给定条件下重复执行某些语句，这时就需要循环结构的语句。有了顺序、选择和循环这三种基本的结构，就能构建复杂的程序了。本章主要介绍布尔表达式、选择结构、条件表达式、while 循环、for 循环、for 推导式、块表达式赋值、循环中的 break 和 continue 等。

3.1 布尔表达式

选择结构和循环结构都需要使用布尔表达式作为选择和循环的条件。布尔表达式是由表达式、关系运算符和逻辑运算符按一定的语法规则组成的式子。前文已介绍过 Scala 的关系运算符有<（小于）、<=（小于或等于）、==（等于）、>（大于）、>=（大于或等于）、!=（不等于），逻辑运算符有 &&（逻辑与）、||（逻辑或）、!（逻辑非）。

由于布尔数据类型（Boolean）的数据值只有两个（true 和 false），所以一个布尔类型的变量的值只能是 true 或 false，布尔表达式的值也只能取 true 或 false。布尔表达式举例如下。

```
scala>var a =true
a: Boolean =true
scala>var b =false
b: Boolean =false
scala>a&&b
res10: Boolean =false
scala>a||b
res11: Boolean =true
```

3.2 选 择 结 构

3.2.1 单向 if 选择语句

Scala 的 if 语句与其他语言的 if 语句相似，可以检测条件并根据其是否为真来决定是否执行对应的分支。if 语句由布尔表达式及之后的语句块组成，具体语

法格式如下所示。

```
if(布尔表达式)
{
    语句块                //如果布尔表达式为 true 则执行该语句块
}
```

举例如下所示。

```
scala>:paste              //进入代码块编写模式(paste 模式)
// Entering paste mode (ctrl-D to finish)

  var x =10
  if( x <20 ){
      println("x <20");
  }
// Exiting paste mode, now interpreting
```

代码编写完成后可按 Ctrl＋D 组合键退出 paste 模式并执行输入的代码，执行结果如下所示。

```
x <20
```

3.2.2　双向 if…else 选择语句

if…else 语句的语法格式如下所示：

```
if(布尔表达式){
    语句块 1              //如果布尔表达式为 true 则执行语句块 1
}else{
    语句块 2              //如果布尔表达式为 false 则执行语句块 2
}
```

举例如下所示。

```
scala>:paste
// Entering paste mode (ctrl-D to finish)
var x =30;
if (x <20 ){
   println("x 小于 20");
       }
else{
  println("x 大于或等于 20");
        }
// Exiting paste mode, now interpreting
```

退出 paste 模式并执行代码，结果如下所示。

```
x 大于 20
x: Int =30
```

3.2.3　嵌套 if…else 选择语句

if 语句后可以紧跟 else if…else 语句，在多个条件下判断语句的执行情况。嵌套 if…

else 语法格式如下。

```
if(布尔表达式 1){
    语句块 1          //如果布尔表达式 1 为 true 则执行语句块 1
}else if(布尔表达式 2){
    语句块 2          //如果布尔表达式 2 为 true 则执行语句块 2
}else if(布尔表达式 3){
    语句块 3          //如果布尔表达式 3 为 true 则执行语句块 3
}else {
    语句块 4          //如果以上条件都为 false 执行语句块 4
}
```

3.3 条件表达式

Scala 的 if…else 语法结构和 Java 一样。不过,在 Scala 中 if…else 表达式有值,这个值就是跟在 if 或 else 之后的表达式的值。在 Scala 中,条件表达式的用法举例如下所示。

```
if (x >0) 1 else -1
```

上述条件表达式的值是 1 或－1,具体是哪一个取决于当前程序流中 x 的值。用户可以将条件表达式赋值给变量,示例如下。

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

这与如下语句的效果一样。

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

不过,两者相较而言第一种写法更好,因为它可以用来初始化一个常量,而在第二种写法当中,s 必须是变量,如下所示。

```
scala>val x =3
x: Int =3
scala>val y =if ( x >1 ) 1 else -1
y: Int =1
```

3.4 while 循环

Scala 拥有与 Java 相同的 while 循环和 do…while 循环。while 循环的语法格式如下所示。

```
while(布尔表达式){
    循环体
}
```

while 循环举例如下所示。

```
scala>:paste
// Entering paste mode (ctrl-D to finish)
var i:Int =0                    //定义变量
while (i <5){
```

```
    print("hello" +i+" ")
    i +=1
    }
// Exiting paste mode, now interpreting
```

代码编写完成后按 Ctrl＋D 组合键退出 paste 模式并执行输入的代码，执行结果如下所示。

```
hello0 hello1 hello2 hello3 hello4
```

do…while 循环的用法也沿袭 Java，如下所示。

```
do{
    循环体
}
while(条件语句)
```

3.5　for 循环

Scala 的 for 循环的语法格式如下。

```
for(控制变量 <-可遍历序列){
    循环体}
```

这里的“<-”是 for 循环的组成部分。for 循环是一种遍历型的循环，它会依次遍历有序集合中的元素，每遍历一个元素就执行一次循环体，遍历完所有元素之后便退出循环。这种有序集合可以是数组、列表、字符串、元组、集、映射等。for 循环的使用方式主要有以下几种。

(1) 使用 for(x <- Range)的方式，用法如下。

```
for(x <-Range ){
    循环体}
```

其中，Range 可以是一个数字区间，如 i to j ，或者 i until j。左箭头“<-”用于为变量 x 赋值。i to j 表示的区间是[i, j]，i until j 表示的区间是[i, j)，两种形式举例如下。

```
scala>for (i <-1 to 3) {println(s"Day $i:")}
Day 1:
Day 2:
Day 3:
scala>for (i <-1 until 10) printf("%d ",i)
1 2 3 4 5 6 7 8 9
```

(2) 使用分号“;”设置多个区间，它将迭代给定区间所有可能的值，示例如下。

```
scala>for( a <-1 to 2; b <-1 to 2){
      println( "a: " +a +" b: " +b) }
```

运行上述代码得到的输出结果如下。

```
a: 1 b: 1
```

```
a: 1 b: 2
a: 2 b: 1
a: 2 b: 2
scala>for( i <-1 to 5 if i%2 !=0;j <-1 to 5 if j%2 ==0 ) println( i +" * " +j +" =" 
+i * j)
1 * 2 =2
1 * 4 =4
3 * 2 =6
3 * 4 =12
5 * 2 =10
5 * 4 =20
```

(3) 用 for 循环遍历数组、列表和集合,示例如下。

```
scala>val list1=List(3,5,2,1,7)          //创建列表
list1: List[Int] =List(3, 5, 2, 1, 7)
scala>for(x <-list1){print(" "+x)}
3 5 2 1 7
```

(4) 在 for 循环中使用过滤器,示例如下。

```
scala>for(x <-list1 if x%2==1){print(" "+x)}
3 5 1 7
scala>var bArray=Array("宋爱梅","王志芳","李涛","贾燕青","刘振杰","刘涛")
scala>for(file <-bArray if file.endsWith("涛")){println(file)}
李涛
刘涛
```

用户可以在 for 循环中引入已在循环中使用的变量,示例如下。

```
scala>for(i<-1 to 3; from =4-i; j<-from to 4) print((10 * i+j).toString+" ")
13 14 22 23 24 31 32 33 34
```

【例 3-1】 通过索引使用 for 循环和 until 关键字遍历数组。

下面需要在交互模式下使用代码块一次运行多条语句。先输入":paste"并回车进入 paste 模式,然后粘贴(写入)代码块。

```
scala>:paste          //进入粘贴模式
// Entering paste mode (ctrl-D to finish)

var a =Array("hello","Scala")
for (i <-0 until a.length)
  println(a(i))

// Exiting paste mode, now interpreting
```

之后按 Ctrl+D 组合键结束输入并运行代码块。

```
hello
Scala
```

如果想要在遍历元素时隔一个元素执行一下处理操作,则可以采用如下方式。

```
scala>val b=Array(1,2,3,4,5,6)
```

```
b: Array[Int] =Array(1, 2, 3, 4, 5, 6)
scala>for (i <-0 until (b.length,2))
     | printf("%d ",i)
0 2 4
```

从数组的尾端开始倒序遍历，语法格式如下。

```
for(i <-(0 until b.length).reverse)
    循环体语句
```

示例如下。

```
scala>for (i <-(0 until b.length).reverse)
     | printf("%d ",i)
5 4 3 2 1 0
```

不使用数组下标，直接遍历数组元素的语法格式如下。

```
for(e <-b)
    循环体语句
```

示例如下。

```
scala>for(e <-b)
       | printf("%d ",e)
1 2 3 4 5 6
```

注意：变量 e 将先后被设为 b(0)、b(1)，以此类推。

【例 3-2】　根据给定的整数数组创建一个新的数组，新数组中包括原数组中的全部值为正数的元素，并以原有顺序排列，之后的元素则是数组中的零或负值，这些元素也以原有顺序排列，如下所示。

```
scala>:paste
// Entering paste mode (ctrl-D to finish)
import scala.collection.mutable.ArrayBuffer //导入 ArrayBuffer 包
val arr =Array(1,3,-3,-5,-7,3,2)
val a =ArrayBuffer[Int]()
val b =ArrayBuffer[Int]()
arr.foreach(arg =>if(arg >0) a +=arg else b +=arg)
a ++=b
a.toArray
// Exiting paste mode, now interpreting
```

按 Ctrl＋D 组合键退出 paste 模式并执行代码，结果将如下所示。

```
a: scala.collection.mutable.ArrayBuffer[Int] =ArrayBuffer(1, 3, 3, 2, -3, -5, -7)
b: scala.collection.mutable.ArrayBuffer[Int] =ArrayBuffer(-3, -5, -7)
```

3.6　for 推导式

for 推导式用来创建一个集合，其语法格式如下。

```
for () yield {}
```

从 for 推导式的语法格式可以看出,for 循环的循环体将以 yield 开始,每次迭代生成集合中的一个值,示例如下。

```
scala>val data =for( i <-1 to 5 ) yield {i}
data: IndexedSeq[Int] =Vector(1, 2, 3, 4, 5)
scala>print(data)
Vector(1, 2, 3, 4, 5)
scala>for(c<-"abcde";i<-0 to 1) yield c+i
res34: IndexedSeq[Int] =Vector(97, 98, 98, 99, 99, 100, 100, 101, 101, 102)
scala>for(c<-"abcde";i<-1 to 2) yield (c+i).toChar
res32: IndexedSeq[Char] =Vector(b, c, c, d, d, e, e, f, f, g)
scala>val data1=for(i<-1 to 5 if i%2 !=0;j <-10 until 11;sum=i+j) yield sum
data1: IndexedSeq[Int] =Vector(11, 13, 15)
scala>val names =Array("小丽", "唐诗涛", "刘涛","杨丽涛", "杨雪涛","杨鹏")
scala>val filteredNames=for(x <-names;if x.contains("涛") && !x.startsWith
("杨")) yield x
scala>filteredNames.foreach(println)
唐诗涛
刘涛
```

上面的“val filteredNames = for(x <- names; if x.contains("涛") && ! x.startsWith("杨")) yield x”的表达式还可以写成下面的式子。

```
val filteredNames =for {x <-names
if x.contains("涛") && !x.startsWith("杨")
} yield x
```

即将圆括号中的内容写到花括号中,注意这时要用换行的方式隔开它们,而不是使用分号(;)。

3.7 块表达式的赋值

在 Java 中,用花括号({})括起来的语句序列被称为块语句。当需要在选择分支或循环中放置多个语句时,就需要使用块语句。

在 Scala 中,块语句包含一系列表达式,其结果也是一个表达式。块中最后一个表达式的值就是块的值。这个特性对于那些需要分多步完成的初始化常量的操作很有用。如下所示。

```
scala>val x0,y0=1
x0: Int =1
y0: Int =1
scala>val x=4;val y=5;
x: Int =4
y: Int =5
scala>import scala.math._                    //Scala 中,字符 _ 是通配符,类似 Java 中的 *
scala>val distance ={val dx =x -x0; val dy =y -y0; sqrt(dx * dx +dy * dy);}
distance: Double =5.0
```

以上代码中块语句的值由最后一个表达式“sqrt(dx * dx + dy * dy)”的值决定,变量 dx 和 dy 仅为计算所需要的中间值。

以 scala 开头的包，在引入或使用时 scala 可以被省略。

```
scala>import math._
scala>sqrt(2)
res0: Double =1.4142135623730951
```

3.8　循环中的 break 和 continue 语句

在 Scala 中，类似 Java 的 break、continue 关键字被移除了。如果一定要实现 Java 中 break 和 continue 关键字的功能，就需要使用 scala.util.control 包中 Break 类的 breakable 静态类和 break()方法，具体使用方法如下。

(1) 导入 Breaks 包 import scala.util.control.Breaks._。

(2) 使用 breakable 静态类将 for 表达式包起来。

(3) 在 for 表达式中需要退出循环的地方添加 break()方法。

【例 3-3】　使用 for 循环打印 1～100 的数字，如果数字到达 50 则退出 for 循环。

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

import scala.util.control.Breaks._
breakable{
    for(i <-1 to 100) {
        if(i >=50) break()
        else print(i+",")
    }
}
// Exiting paste mode, now interpreting
```

按 Ctrl＋D 组合键退出 paste 模式并执行代码，结果如下所示。

```
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,
```

continue 功能的实现与 break 类似，但有一点不同：实现 break 功能是用 breakable 静态类将整个 for 表达式包起来，而实现 continue 功能只需要用 breakable 静态类将 for 表达式的循环体包起来就可以了，具体示例如下。

【例 3-4】　打印 1～100 的数字，使用 for 循环遍历这些数字，如果数字能整除 2，则将之跳过。

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

import scala.util.control.Breaks._
for(i <-1 to 100 ) {
    breakable{
        if(i %2 ==0) break()
        else print(i+",")
    }
}
```

```
// Exiting paste mode, now interpreting
```

按 Ctrl+D 组合键退出 paste 模式并执行代码,结果如下所示。

```
1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,
55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,99,
```

3.9 习　　题

1. 有一个包含 10 个元素的数组,第一个元素的值是 3,后面每个元素的值都是前面一个元素的两倍加 1,打印这个数组,然后将数组中奇数元素和偶数元素位置互换。

2. 输入一个整数,将这个整数的所有约数放入一个数组,打印此数组。

3. 请写出一个 for 循环,用 i 表示循环的变量,通过 Range 生成 0～20 的数字,并循环打印出 0～20 这些数字中的奇数。

4. 请写出一个 for 循环,用 i 表示循环的变量,通过 Range 生成 0～20 的数字,并循环打印出 0～20 这些数字中的偶数(包括 20)。

5. 请使用嵌套 for 循环打印九九乘法表。