Scala核心编程

-程序流程控制

尚硅谷研究院

- 程序流程控制介绍

在程序中，程序运行的流程控制决定程序是如何执行的，是我们必须掌握的，主要有三大流程控制语句。

温馨提示： Scala语言中控制结构和Java语言中的控制结构基本相同，在不考虑特殊应用场景的情况下，代码书写方式以及理解方式都没有太大的区别

1) 顺序控制
2) 分支控制
3) 循环控制

- 顺序控制

顺序控制介绍

程序从上到下逐行地执行，中间没有任何判断和跳转。

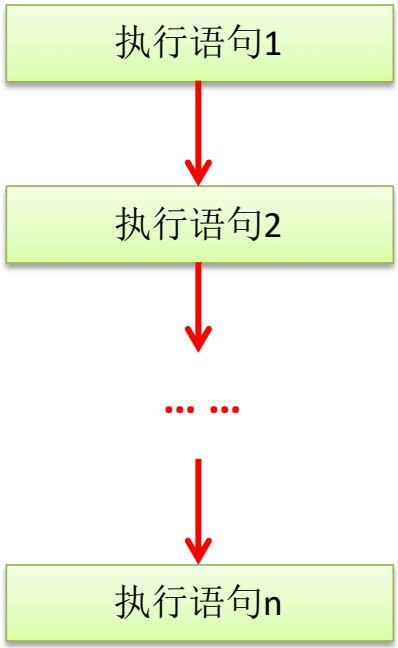
顺序控制举例和注意事项

Scala中定义变量时采用合法的前向引用。如：

```
def main(args : Array[String]) : Unit = {
    var num1 = 12
    var num2 = num1 + 2
}
```

错误形式：

```
def main(args : Array[String]) : Unit = {
    var num2 = num1 + 2
    var num1 = 12
}
```



分支控制if-else

- 分支控制if-else

分支控制if-else介绍

让程序有选择的的执行,分支控制有三种:

1) 单分支
2) 双分支
3) 多分支

- 分支控制if-else

单分支

➢ 基本语法

if (条件表达式) {
执行代码块
}

说明：当条件表达式为ture 时，就会执行 { } 的代码。

➢ 案例说明

请大家看个案例[IfDemo.scala]:

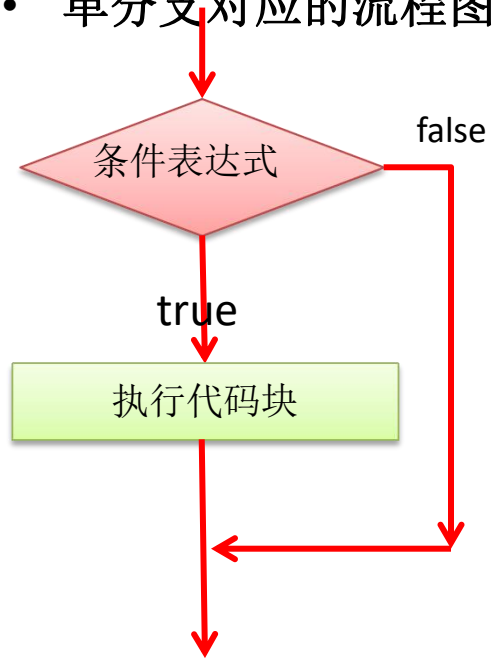
编写一个程序,可以输入人的年龄,如果该同志的年龄大于18岁,则输出 “age > 18”

```
val age = 20
if (age > 18) {
println("age > 18")
}
```

- 分支控制if-else

单分支

- 单分支对应的流程图



- 分支控制if-else

双分支

➢ 基本语法

```
if (条件表达式) {
        执行代码块1
} else {
        执行代码块2
}
```

说明：当条件表达式成立，即执行代码块1，否则执行代码块2.

➢ 案例演示

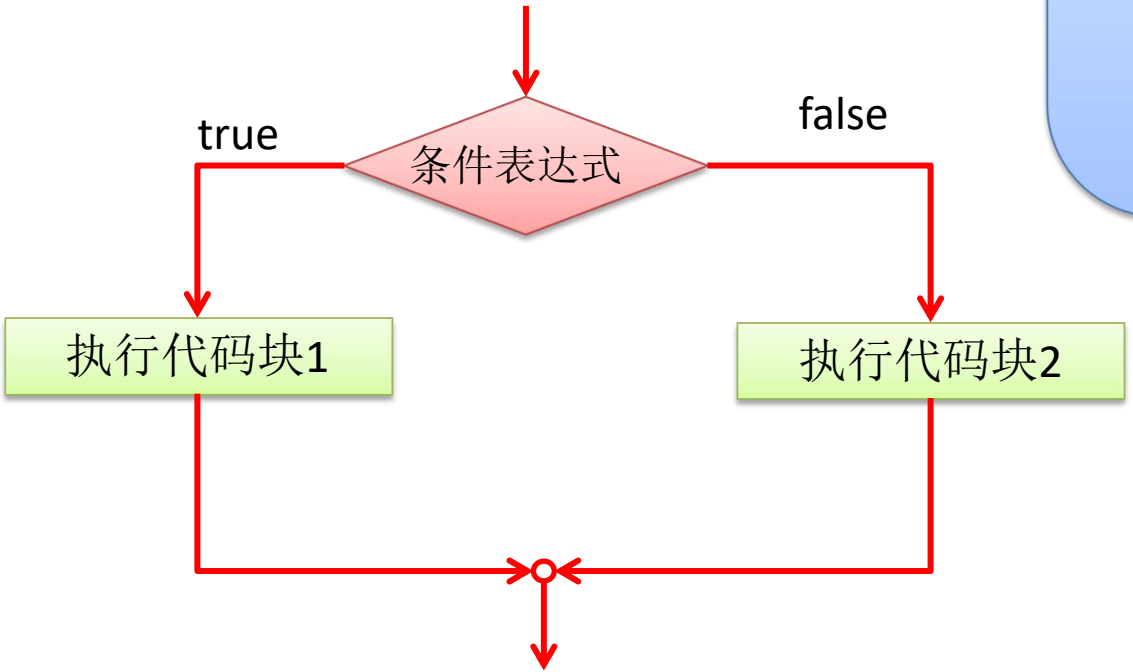
请大家看个案例[IfDemo2.scala]:

编写一个程序,可以输入人的年龄,如果该同志的年龄大于18岁,则输出 “age >18”。否则 ,输出 "age <= 18 "

- 分支控制if-else

双分支

- 双分支对应的流程图



- 分支控制if-else

单分支和双分支练习题

1) 对下列代码，若有输出，指出输出结果。

```
var x = 4
var y = 1
if (x > 2) {
      if (y > 2) {
            println(x + y)
     }
            println("atguigu")  // 输出内容 atguigu
} else
      println("x is " + x)
```

- 分支控制if-else

单分支和双分支课后题

2) 编写程序，声明2个Int型变量并赋值。判断两数之和，如果大于等于50，打印“hello world!”
3) 编写程序，声明2个Double型变量并赋值。判断第一个数大于10.0，且第2个数小于20.0，打印两数之和。
4) 【选作】定义两个变量Int，判断二者的和，是否既能被3又能被5整除，打印提示信息
5) 判断一个年份是否是闰年，闰年的条件是符合下面二者之一：(1)年份能被4整除，但不能被100整除；(2)能被400整除

说明: 因为同学们都学习过Java 了，因此上面的题对大家应该很简单，自己课后练习下即可， 解题思路和Java完全一样。

- 分支控制if-else

多分支

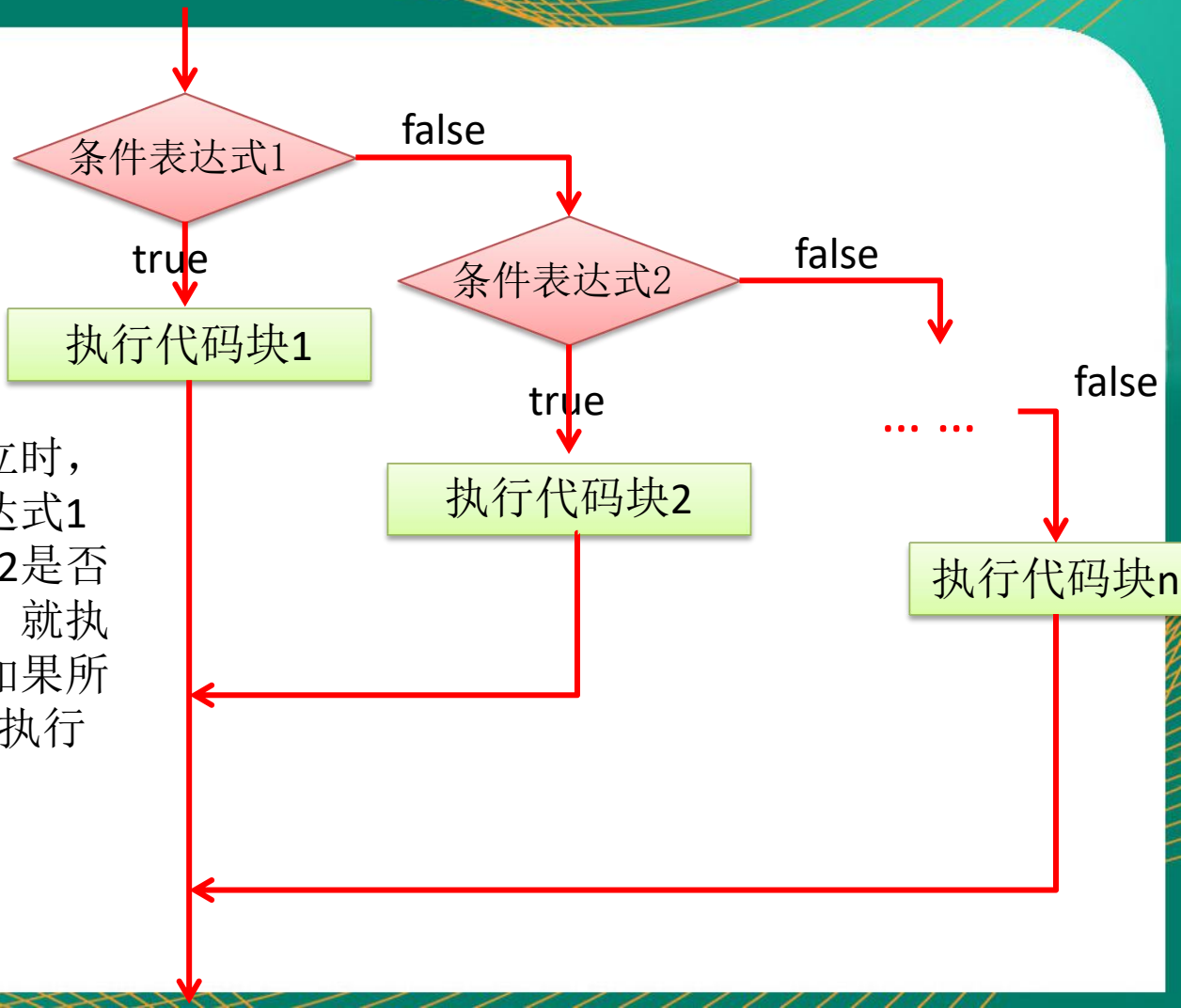
- 基本语法

```
if (条件表达式1) {
        执行代码块1
}
else if (条件表达式2) {
        执行代码块2
}
 ……
 else {
        执行代码块n
 }
```

- 分支控制if-else

多分支

➢ 多分支的流程图



说明：当条件表达式1成立时，即执行代码块1，如果表达式1不成立，才去判断表达式2是否成立，如果表达式2成立，就执行代码块2，以此类推，如果所有的表达式都不成立，则执行else 的代码块，注意，只能有一个执行入口。

- 分支控制if-else

多分支

➢ 案例演示

请大家看个案例[IfDemo3.scala]:

岳小鹏参加scala考试，他和父亲岳不群达成承诺：
如果：
成绩为100分时，奖励一辆BMW；
成绩为(80，99]时，奖励一台iphone7plus；
当成绩为[60,80]时，奖励一个 iPad；
其它时，什么奖励也没有。

说明: 成绩在控制台输入!

- 分支控制if-else

多分支

➢ **案例演示2 [**课堂练习**]**

求$ax^2+bx+c=0$方程的根。a,b,c分别为函数的参数，如果：$b^2-4ac>0$，则有两个解；$b^2-4ac=0$，则有一个解；$b^2-4ac<0$，则无解； [a=3 b=100 c=6]

提示1：$x_1=(-b+sqrt(b^2-4ac))/2a$

$x_2=(-b-sqrt(b^2-4ac))/2a$

提示2：sqrt(num) 在 scala 包中(默认引入的) 的math 的包对象有很多方法直接可用.

- 分支控制if-else

分支控制if-else 注意事项

1) 如果大括号{}内的逻辑代码只有一行，大括号可以省略，这点和java 的规定一样。

2) Scala中任意**表达式都是有返回值的**，也就意味着if else表达式其实是有返回结果的，具体返回结果的值取决于满足条件的代码体的最后一行内容.[案例演示]

3) Scala中是没有三元运算符，因为可以这样简写

```
// Java
int result = flg ? 1 : 0
```

```
// Scala
val result = if (flg) 1 else 0   // 因为 scala 的if-else 是有返回值的，因此，本身这个语言也不需要三元运算符了(如图)，并且可以写在同一行，类似 三元运算
```

- 分支控制if-else

分支控制if-else 注意事项

4) 看下面案例，判断输出什么

```
object Hello01 {
  def main(args: Array[String]): Unit = {
    var sumVal = 9
    val result =
      if(sumVal > 20){
        "结果大于20"
      }
    println(result) //
  }
}
```

```
object Hello01 {
  def main(args: Array[String]): Unit = {
    var sumVal = 60
    val result =
      if(sumVal > 20){
        "结果大于20"
      }
    println(result) //
  }
}
```

嵌套分支

- 嵌套分支

基本介绍

在一个分支结构中又完整的嵌套了另一个完整的分支结构，里面的分支的结构称为内层分支外面的分支结构称为外层分支。嵌套分支不要超过3层

基本语法

```
if(){
        if(){
        }else{
        }
}
```

- 嵌套分支

应用案例1

参加百米运动会，如果用时8秒以内进入决赛，否则提示淘汰。并且根据性别提示进入男子组或女子组。【可以让学员先练习下5min】，输入成绩和性别，进行判断。
1分钟思考思路
double second; char gender;

- 嵌套分支

应用案例2

出票系统：根据淡旺季的月份和年龄，打印票价 [考虑学生先做5min]

4_10 旺季：

成人（18-60）： 60
儿童（<18）:半价
老人（>60）:1/3

淡季：

成人：40
其他：20

switch分支结构

- **switch分支结构**

在scala中没有switch,而是使用**模式匹配**来处理。
模式匹配涉及到的知识点较为综合，因此我们放在后面讲解。
match-case

for循环控制

- **for循环控制**

基本介绍

Scala 也为for 循环这一常见的控制结构提供了非常多的特性，这些for 循环的特性被称为**for 推导式**（for comprehension）或**for 表达式**（for expression）

范围数据循环方式1

- 基本案例

```
for(i <- 1 to 3){
  print(i + " ")
}
println()
```

说明

1) **i 表示循环的变量， <- 规定好 to 规定**
2) **i 将会从 1-3 循环， 前后闭合**

- 输出10句 "hello,尚硅谷!"

- **for循环控制**

范围数据循环方式2

➢ 基本案例

```
for(i <- 1 until 3) {
  print(i + " ")
}
println()
```

说明:

1) 这种方式和前面的区别在于 i 是从**1 到 3-1**

2) 前闭合后开的范围,和java的arr.length() 类似

```
for (int i = 0; i < arr.lenght; i++){}
```

➢ 输出10句 "hello,尚硅谷!"

- **for循环控制**

循环守卫

➢ 基本案例

```
for(i <- 1 to 3 if i != 2) {
  print(i + " ")
}
println()
```

➢ 基本案例说明

1) 循环守卫，即循环保护式（也称条件判断式，守卫）。保护式为true则进入循环体内部，为false则跳过，类似于continue
2) 上面的代码等价

```
for (i <- 1 to 3) {
        if (i != 2) {
                println(i+"")
        }
}
```

- **for循环控制**

引入变量

➢ 基本案例

```
for(i <- 1 to 3; j = 4 - i) {
  print(j + " ")
}
```

➢ 对基本案例说明

1) 没有关键字，所以范围后一定要加；来隔断逻辑

2) 上面的代码等价

```
for ( i <- 1 to 3) {
        val j = 4 –i
        print(j+"")
}
```

- **for循环控制**

嵌套循环

➢ 基本案例

```
for(i <- 1 to 3; j <- 1 to 3) {
  println(" i =" + i + " j = " + j)
}
```

➢ 对基本案例说明

1) 没有关键字，所以范围后一定要加；来隔断逻辑

2) 上面的代码等价

```
for ( i <- 1 to 3) {
        for ( j <- 1to 3){
                println(i + " " + j + " ")
        }
  }
```

- **for循环控制**

循环返回值

➢ 基本案例

```
val res = for(i <- 1 to 10) yield i
println(res)
```

➢ 对基本案例说明

1) 将遍历过程中处理的结果返回到一个新Vector集合中，使用yield关键字

- **for循环控制**

使用花括号{}代替小括号()

➢ 基本案例

```
for(i <- 1 to 3; j =  i * 2) {
  println(" i= " + i + " j= " + j)
}
```

可以写成

```
for{
    i <- 1 to 3
    j = i * 2} {
    println(" i= " + i + " j= " + j)
 }
```

➢ 对基本案例说明

1) {}和()对于for表达式来说都可以
2) for 推导式有一个不成文的约定：当for 推导式仅包含单一表达式时使用圆括号，当其包含多个表达式时使用大括号
3) 当使用{} 来换行写表达式时，分号就不用写了

- **for循环控制**

注意事项和细节说明

1) scala 的for循环形式和java是较大差异，这点请同学们注意，但是基本的原理还是一样的。
2) scala 的for循环的步长如何控制! [for(i <- Range(1,3,2)]
3) 思考题：如何使用循环守卫控制步长

- **for循环控制**

for循环练习题(学员先做)

1) 打印1~100之间所有是9的倍数的整数的个数及总和.
2) 完成下面的表达式输出

```
0 + 6 = 6
1 + 5 = 6
2 + 4 = 6
3 + 3 = 6
4 + 2 = 6
5 + 1 = 6
6 + 0 = 6
```

- **while循环控制**

基本语法

循环变量初始化
while (循环条件) {
　　循环体(语句)
　　循环变量迭代
}

while循环应用实例

1) 画出流程图
2) 输出10句"你好,尚硅谷"


while.zip

- **while循环控制**

注意事项和细节说明

1) 循环条件是返回一个布尔值的表达式
2) while循环是先判断再执行语句
3) 与If语句不同，While语句本身没有值，即整个While语句的结果是Unit类型的()
4) **因为while中没有返回值,所以当要用该语句来计算并返回结果时,就不可避免的使用变量，而变量需要声明在while循环的外部，那么就等同于循环的内部对外部的变量造成了影响，所以不推荐使用，而是推荐使用for循环。**

```
var list = List(1,2,4)
```

- **while循环控制**

练习题

1) 打印1—100之间所有能被3整除的数 [演示]
2) 打印40—200之间所有的偶数

- **do..while循环控制**

基本语法

```
循环变量初始化;
do{
        循环体(语句)
        循环变量迭代
} while(循环条件)
```

do...while循环应用实例

1) 画出流程图
2) 输入10 "你好, 尚硅谷"

dowhile.zip

- **do..while循环控制**

注意事项和细节说明

1) 循环条件是返回一个布尔值的表达式
2) do..while循环是先执行，再判断
3) 和while 一样，**因为do...while中没有返回值,所以当要用该语句来计算并返回结果时,就不可避免的使用变量，而变量需要声明在do...while循环的外部，那么就等同于循环的内部对外部的变量造成了影响，所以不推荐使用，而是推荐使用for循环**

课堂练习题【学员先做】

1) 计算1—100的和 【课后】
2) 统计1——200之间能被5整除但不能被3整除的个数 【课堂】

- 多重循环控制

介绍：

1) 将**一个循环放在另一个循环体内**，就形成了嵌套循环。其中，for ,while ,do...while均可以作为外层循环和内层循环。【建议一般使用两层，最多不要**超过3层**】

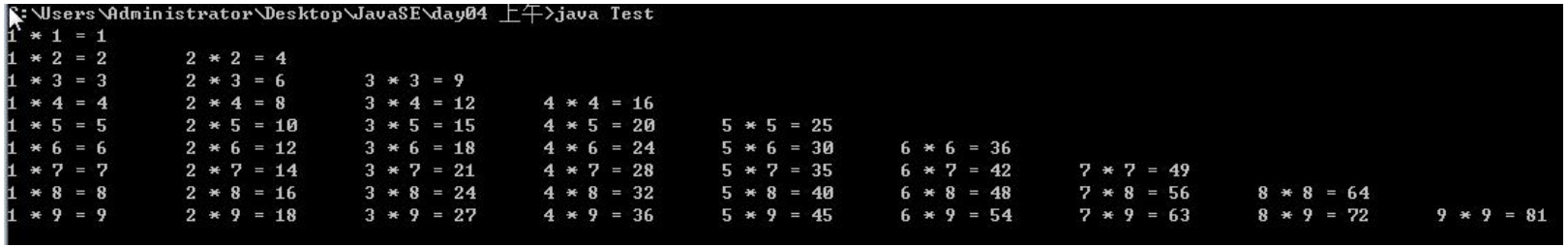
2) 实质上，**嵌套循环就是把内层循环当成外层循环的循环体**。当只有内层循环的循环条件为false时，才会完全跳出内层循环，才可结束外层的当次循环，开始下一次的循环。

3) 设外层循环次数为m次，内层为n次， 则内层循环体实际上需要执行m*n=mn次。

- 多重循环控制

应用实例：

1) 统计三个班成绩情况，每个班有5名同学，求出各个班的平均分和所有班级的平均分[学生的成绩从键盘输入]。
2) 统计三个班及格人数，每个班有5名同学。
3) 打印出九九乘法表



- **while循环的中断**

基本说明

Scala内置控制结构特地**去掉了break和continue**，是为了更好的适应函数化编程，推荐使用**函数式的风格**解决break和contine的功能，而不是一个关键字。

案例演示

```
import util.control.Breaks._
var n = 10
breakable{
  while(n <= 20){
    n += 1
    if(n == 18){
      break()
    }
  }
}
println("n=" + n)
```

代码说明：(看源码分析 重要)

- **while循环的中断**

如何实现continue的效果

Scala内置控制结构特地**也去掉了continue**，是为了更好的适应函数化编程，可以使用**if – else** 或是 循环守卫实现continue的效果

案例

```
for (i <- 1 to 10 if (i != 2 && i != 3)) {
println("i=" + i)
}
```

- 课后练习题

课后练习题：

1) 100以内的数求和，求出当和 第一次大于20的当前数【for】
2) 实现登录验证，有三次机会，如果用户名为”张无忌” ,密码”888”提示登录成功，否则提示还有几次机会，请使用for 循环完成

3）某人有100,000元,每经过一次路口， 需要交费,规则如下:
1. 当现金>50000时,每次交5%
2. 当现金<=50000时,每次交1000
编程计算该人可以经过多少次路口,使用 方式完成

谢谢！ 欢迎收看！