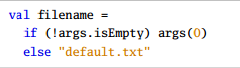
1. 因为scala提供了函数编程,因此提供了很少的控制语法,仅仅提供了if while等很少的。
2. Scala所有的控制函数都会返回一个值,这个与java不同。
3. If expressions
4. 默认其他语法的实现方式

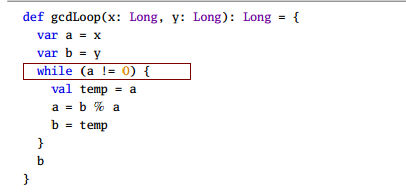


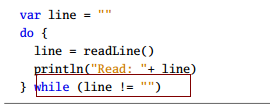
2.因为scala函数有返回值,因此可以优化



第2节 While loops

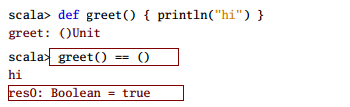
1.while和do while





2.为什么称之为loop呢.是因为他们的返回值类型是util,而不是一个value,即他们不是表达式.因为表达式是有返回值的,他们是没有返回值的。

3.unit返回值也是有值的,这个是与java的void不同的原因,unit的value是()



4.java与scala的=赋值区别

Java:

int a,b;

a=b=1;

scala:

var a: Int = 0  
var b: Int = 0  
a = b = 1

java中以上代码没问题,但是scala中会报错,说类型不匹配。

原因是java赋值语句返回被赋予的值.因此a = b = 1是先将b赋值1,然后返回1,又赋值给a.因此可以运行。

Scala中赋值语句返回的是Unit类型,技术()值。所以上面编译类型不兼容错误。

例如

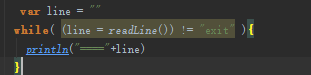
var line1 =""

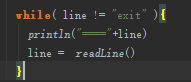
scala> val line = (line1 = readLine())

line: Unit = () 可以看到line的结果是一个unit结果,值是()

5．Scala中while无限循环bug原因分析

因为line = readLine()是赋值语句.返回(),因此永远不会等于exit,因此会无限循环。但是line已经被赋值了,不会有任何line的问题,只是会无限循环而已。

  
修改正确版本



6.将开始的while循环demo修改成函数式方式,省略var变量



第3节 For expressions

1.循环语法 for (I <- list) 循环list,每一个元素都是l



产生的file是一个val类型的,并且类型是File是推测出来的,因为集合的元素是有类型的,所以可以推测。

2. Ranges方式产生集合进行迭代

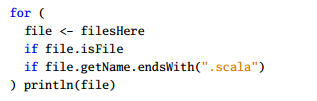
for (i <- 1 to 4) 结果是[1,4]

for (i <- 1 until 4) 结果是[1,4)

3. Filtering 不需要迭代全部元素,只是想要满足条件的元素

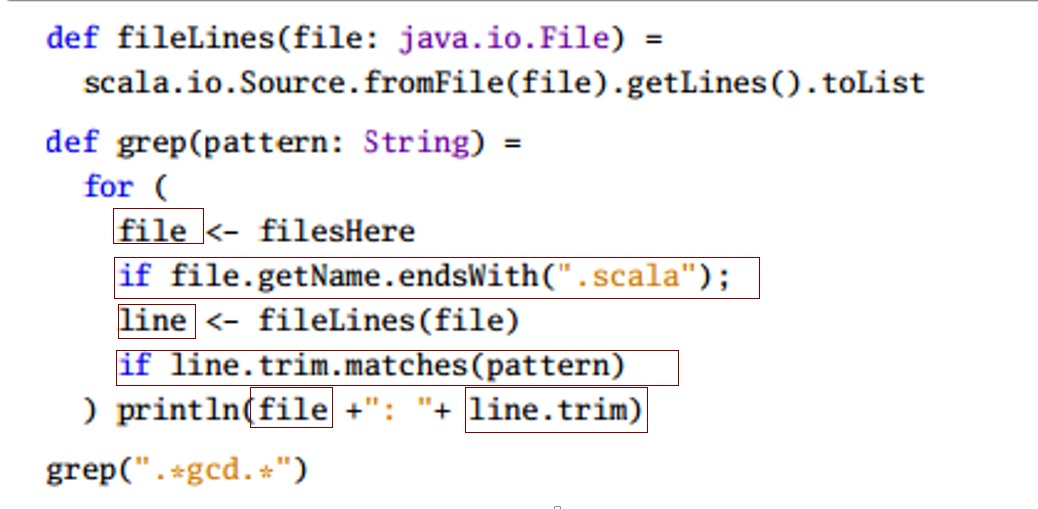
for (file <- filesHere if file.getName.endsWith(".scala"))

追加多个if



4. Nested iteration

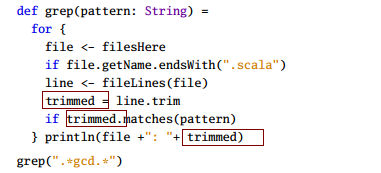
嵌套迭代



读取一个目录下所有scala文件,然后循环每一个文件的所有匹配模式的行,

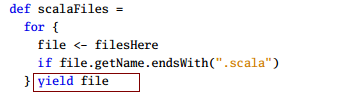
可以看到每一个for循环都可以进行if过滤。

5. Mid-stream variable bindings



产生中间为val的变量,后续可以使用

6. Producing a new collection 产生新的集合



每次body的表达式产生一个新的value,当for表达式完成后,yield关键字将所有产生的新的value结果组成一个新的集合。



有问题的写法



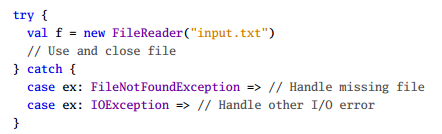
7. Exception handling with try expressions 异常处理表达式

a.异常的抛出,代表返回值是Nothing



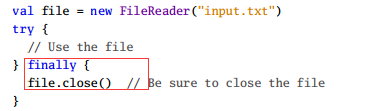
可以看到返回值是int,抛异常是Nothing,他是Int的子类。

b.catch使用的是模式匹配方式



c. scala不要求一定要声明异常,即scala都是类似java的运行期异常，@throws annotation可以产生java的声明异常方式

8.finally

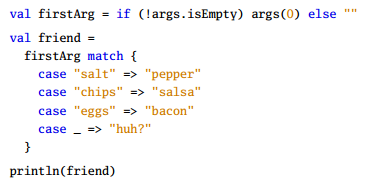


9.try catch finally也是有返回值的

scala的try和java的语法一样,区别就是有返回值以及catch使用模式匹配语法。

第5节 Match expressions

1.匹配表达式



与java不同的是:

a.java的swith只支持枚举和Integer类型

b.不需要java的break退出

c.match…case 也是返回值的表达式

第6节 Living without break and continue如何不适用break和continue进行编程

1.break和continue在函数式编程里面不是很合适,因此应该少用

很明显continue是在while的里面,scala支持函数式编程和命令行方式编程,但是更倾向于函数式编程。

有很多方式不写break and continue也可以编写程序的.如果熟悉了函数式的优点,他可以提供更短的代码方式。

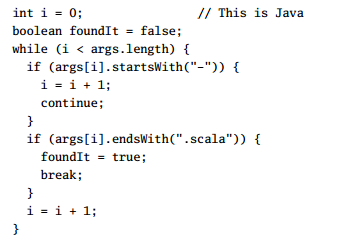
2.最简单的方式代替

a.通过if代替continue

b.通过boolean的变量代替break

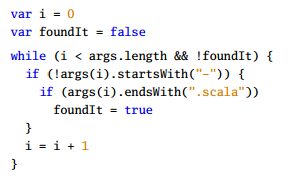
3.例子

Java版本



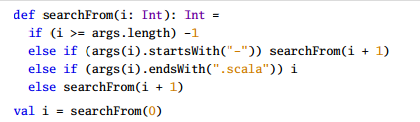
循环所有参数集合,如果是-开头的,则continue,如果是scala文件,则找到合适的文件,因此退出程序,返回第几个参数是找到的数据

Scala版本,简单的逻辑,与java一样,定义了两个var



优化scala版本,取消两个var

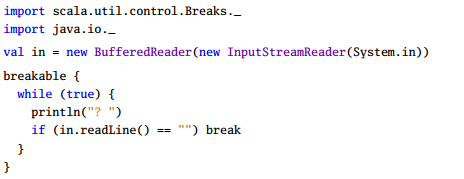
写一个递归函数



3.scala的编译器不会真的去将递归函数去递归执行的,因为所有的递归函数都可以做成尾递归(tail-call position),即8章9节的内容.

4.非要使用break语法--- scala.util.control包下提供了Breaks方法

在breakable的语法块中可以使用break进行控制逻辑关系



Breaks类实现break方法是通过抛出一个exception实现的

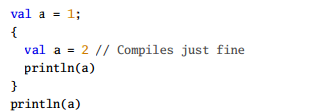
第7节 Variable scope 变量范围

1.作用于范围在{}内

2.相同作用于下不能有同名的



3.在内部作用于下优先级更大



先打印2,再打印1

第8节 Refactoring imperative-style code 重构命令行方式代码风格

暂时没看懂有什么意义