我们在第7章说scala没有内建很多控制抽象的语法,因为scala给用户一个创建自己各种函数的能力。

比如函数可以作为一个值,当作参数传递到函数中、比如currying,比如by-name parameters.

1. Reducing code duplication 减少代码的重复工作
2. 函数作为参数传递到函数中,减少代码

比如给用户提供一个API,

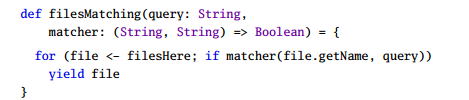
a.让用户给定一个字符串,返回目录下文件以该字符串结尾的文件集合

b.让用户给定一个字符串,返回目录下文件包含该字符串的文件集合

c.让用户给定一个字符串,返回目录下文件正则匹配该字符串的文件集合

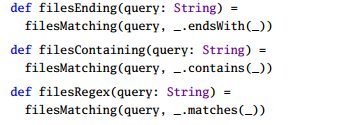
这样日后用户越来要求越多,越难满足

因此进行抽象,减少代码,方式是参数是一个函数即可



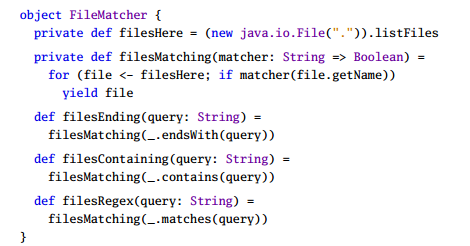
即用户传递一个字符串query,以及一个函数,其实可以优化函数只有一个参数即可,不管怎么样,反正这种用函数作为参数的方式可以节省代码。

前三个需求变成如下:



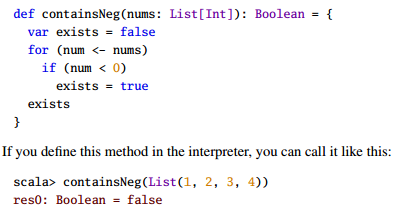
2．java中是使用接口定义方法,然后匿名函数实现,这样虽然也解决代码重复问题,但是代码复杂度高了,因此效果并不好

3．使用一个参数,简化刚刚的方法



这种方式跟上面的方式相比虽然简化了,但是函数使用的query是来自于非函数提供的参数.因此就涉及到闭环的问题了,因此最好也不要这么写,但是scala支持闭环,所以这么写也没问题。但是其他语言可能不支持,因此理论上传入两个参数在函数中正确的,因为这样就避免闭环了,函数body中用到的都是自己需要的参数。

1. Simplifying client code 简化客户端的调用
2. 复杂的问题



上面确定参数集合是否有负数,有则返回true

1. 简化



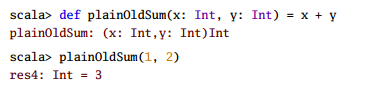
因此这样的方式也避免使用scala的for和while循环了,因此这也是scala为什么很少内建函数的原因。

或者方式

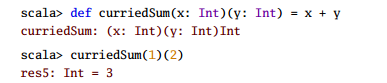
1. Currying柯里化

1.curried函数表示提供多个参数列表的函数,代替仅仅一个函数列表参数的方式

非curried版本



Curried版本



2. 内部逻辑是执行了两次函数,第一个函数是拿着第一个参数集合,返回一个函数对象,因为函数是一等公民,而返回的函数持有第二个参数集合





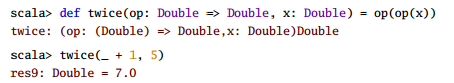


1. 上面讲述的是内部原理,但是要想拆分成多个函数的话,最好不要用上面的方式,而是用占位符的方式,即



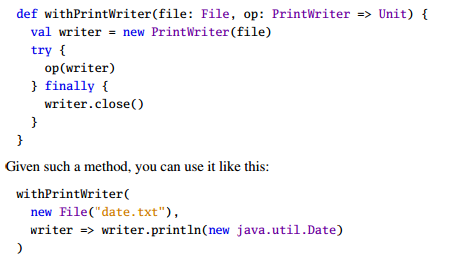
1. Writing new control structures 写一个新的控制对象

1.函数套着函数使用



说明op是一个函数,持有double参数,返回一个double的函数

2. 例子:打开一个资源,操作他.然后关闭该资源



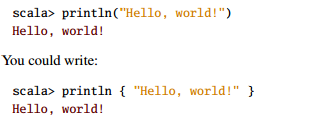
a.可以看到上面的代码逻辑虽然完美,但是也缺少问题,因为第二个参数是写入数据的函数,一行代码显然干不了什么大事儿,该函数需要用{}阔上,但是代码结构又很难看。

b.上面代码可以让用户不关注怎么取消资源,而关注更多的如何处理数据,在处理数据中要么完成,要么抛异常,都会关闭资源。

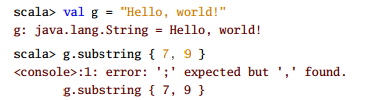
c.客户端代码看起来像内建函数一样使用如下方式:

使用{}代替()去包围参数集合, 因为在scala中任意方法执行时候,如果传递的是一个参数,那么都可以使用{}代替()。

例如

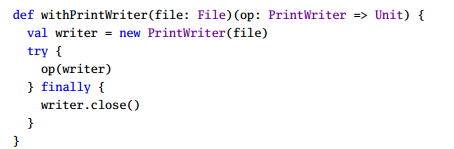


d. 第二个例子,使用{}代替（）包围参数去println,如果{}里面的参数不是一个参数,那么就会有问题,这个时候必须用(),比如



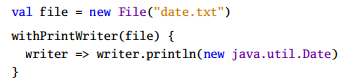


e. 因为withPrintWriter方法有两个参数,因此你不能使用{},但是最后一个参数是withPrintWriter,因此可以将该参数使用currying来拉出来,变成一个参数,因此产生新的形式,比如



这样产生一个方法调用感觉更像是一个控制逻辑,

f.调用新产生的函数



这个例子会看到我们更像标准控制逻辑了,即调用一个函数,传递一个参数,后面的用{}进行代码body控制。

第一个参数是一个文件对象,用()包上,第二个参数是一个函数对象,使用{}包上。

1. By-name parameters

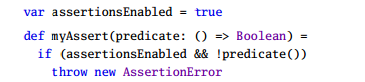
1．上面讲解的withPrintWriter函数的形式,与内置的控制逻辑代码不太一样,比如if(){},

区别是我们上面的代码中{}里面持有一个参数PrintWriter,参数的形式是writer =>.

2．如果我们想实现跟if一样,但是没有值被传递到代码中,怎么办?,即是引用传递进去,Scala提供了by-name parameters形式.

例子:假设要实现一个断言方法,称之为myAssert,该myAssert函数有一个function函数参数,和一个flag决定如何做的参数,

如果flag=false,则什么断言都不操作,如果flag=true,则进行断言,如果断言是false,则抛异常。



1. 上面的形式没问题,但是使用起来有一点尴尬,比如 myAssert(() => 5 > 3),

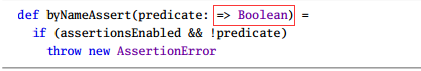
参数那么麻烦的原因是,函数Assert要求的参数是一个函数,而不是具体的boolean值.

你可能想取消() =>

即myAssert(5 > 3) 但是这种情况不能编译,因为参数传递的是一个boolean结果,因为先计算5>3的结果是true,然后传递到参数中,但是 myAssert要求的参数不是boolean,而是一个函数。

5. 使用by-name parameters形式进行修改

所以我们只要保证传入参数不是具体的值,而是一个name引用即可,scala让一个类型开始是=> 就表示该情况是一个引用传递,而不是值传递。

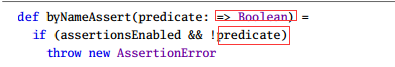


即参数predicate的类型是 => Boolean

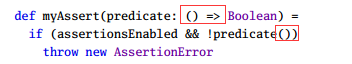
因此调用的时候使用byNameAssert(5 > 3)方式就可以调用了,就向内部函数一样,此时传递的5>3是一个函数,而不是具体的boolean值。即,因此你在参数里面打印一句话,也没问题.也不会立刻执行。

6.区别

新的



老的

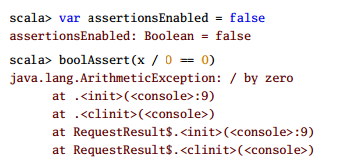


7.name的有什么好处



这样定义后,也是可以执行boolAssert(5 > 3)的,这样是看不到好处的.

但是当如下定义:



可以看到有异常,因为先执行参数内的函数.产生的结果再调用boolAssert函数,因此有问题。

而就不会有问题.因为传递进去的是一个函数,而函数此时在flag=false的时候是不会执行的,因此不会出错。

8.因为函数作为参数的时候,分为有参数的函数和无参数的函数,当有参数的函数时候,调用时候一定要加入参数,因此byname作为简写的方式就意义没有了,因此无参数的时候才有简写的意义,因此才有byname的需求

9.网上文章说明

