Scala核心编程

-使用递归的方式去思考,去编程

尚硅谷研究院

- 基本介绍

Scala 是运行在 Java 虚拟机（Java Virtual Machine）之上，因此具有如下特点:

1) 轻松实现和丰富的 Java 类库互联互通。
2) 它既支持面向对象的编程方式，又支持函数式编程。

3) 它写出的程序像动态语言一样简洁，但事实上它确是严格意义上的静态语言。
4) Scala 就像一位武林中的集大成者，将过去几十年计算机语言发展历史中的精萃集于一身，化繁为简，为程序员们提供了一种新的选择。设计者马丁·奥得斯基 希望程序员们将编程作为简洁，高效，**令人愉快**的工作。同时也让程序员们进行关于编程思想的新的思考。

- **Scala提倡函数式编程(递归思想)**

先说下编程范式:

1) 在所有的编程范式中，面向对象编程（Object-Oriented Programming）无疑是最大的赢家。
2) 但其实面向对象编程并不是一种严格意义上的编程范式，严格意义上的编程范式分为：命令式编程（Imperative Programming）、函数式编程（Functional Programming）和逻辑式编程（Logic Programming）。面向对象编程只是上述几种范式的一个交叉产物，更多的还是继承了命令式编程的基因。
3) 在传统的语言设计中，只有命令式编程得到了强调，那就是程序员要告诉计算机应该怎么做。而递归则通过灵巧的函数定义，告诉计算机做什么。因此在使用命令式编程思维的程序中，是现在多数程序采用的编程方式，递归出镜的几率很少，而在函数式编程中，大家可以随处见到递归的方式。

- 应用实例

scala中循环不建议使用while和do...while,而建议使用递归。

应用实例要求:
计算1-50的和

常规的解决方式

案例演示

```
val now: Date = new Date()
val dateFormat: SimpleDateFormat =
new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
val date = dateFormat.format(now)
println("date=" + date) //输出时间
var res = BigInt(0)
var num = BigInt(1)
var maxVal = BigInt(99999999l) //BigInt(99999999l)[测试效率大数]
while (num <= maxVal) {
res += num
num += 1
}
println("res=" + res)
```

- 应用实例

使用函数式编程方式-递归

函数式编程的重要思想就是尽量不要产生额外的影响,上面的代码就不符合函数式编程的思想, 下面我们看看使用函数式编程方式来解决(Scala提倡的方式)

测试：看看递归的速度是否有影响?

案例演示

```
// 递归的方式来解决
def mx(num:BigInt,sum:BigInt):BigInt = {
if(num <= 99999999l) return mx(num+1,sum + num)
else return sum
}

//测试
var num = BigInt(1)
var sum = BigInt(0)
var res = mx(num,sum)
println("res=" + res)
```

- 应用实例2

使用函数式编程方式-递归

求最大值

案例演示

```
//大话java数据结构
def max(xs: List[Int]): Int = {
    if (xs.isEmpty)
      throw new java.util.NoSuchElementException
    if (xs.size == 1)
      xs.head
    else if (xs.head > max(xs.tail)) xs.head else max(xs.tail)
  }
```

- 应用实例3

使用函数式编程方式-递归

翻转字符串

案例演示

```
//Str = "ab"
def reverse(xs: String): String =
    if (xs.length == 1) xs else reverse(xs.tail) + xs.head
```

- 应用实例4

使用函数式编程方式-递归

求阶乘

案例演示

```
def factorial(n: Int): Int =
    if (n == 0) 1 else n * factorial(n - 1)
```

谢谢！ 欢迎收看！