Scala

Scala和java有什么不同？

Scala和java都是跑在JVM上的语言，Java能实现的功能Scala都能实现，Scala能实现的功能理论上Java也都可以实现，他们只是两种不同的编程语言。有道是，条条大路通罗马，Scala和Java只是你通向罗马不同的工具。

Scala是面向对象和面向函数的完美结合，它介于Java和Haskell中间的一种语言，是两个的一种折中。所以用scala，更多的是用它的函数式编程，对于javaer也更加友好。Scala比其他FP更好的一点是，它跑在JVM上，所以所有的java的库包，Scala都可以拿过来用。

什么是函数式编程呢？

这里的函数指的是数学上的函数概念，就是从一种东西到另外一种东西的对应关系。

举个例子：

输出从1到100

Java：

For(int i=1; i<=100; i++)

System.out.println(i)

这是一种面向计算机硬件抽象的命令式编程，有变量，赋值语句，表达式，控制语句，一句话，命令式程序，就是一个冯诺依曼的指令序列。

Scala：

(1 to 100) map (println)

这是两个集合的映射关系，从一个（1,2,3,4 …100） 到（unit，unit…unit）,我们只是描述了两个集合之间的映射关系。

通过上面的例子，我们可以看出来，函数式编程中的函数，不是指计算机中的函数，而是数学中的函数，即自变量的映射。也就是说一个函数的值，仅仅取决于函数的参数值，不依赖于其他的状态。

纯函数编程语言中的变量也不是命令式语言中的变量，即存储状态的单元，而是代数中的变量，即一个值的名字，变量的值是不可变的。也就是不允许像命令式编程语言中那样多次给一个变量赋值。比如“x=x+1”，这依赖于可变状态。

但最终，函数式的程序还是会被编译成（冯诺依曼机）机器语言去执行。

函数式编程的好处：

我觉得函数式的最主要好处是由不可变性带来的。

1. **class** Account (val balance: Integer) {
2. def getBalance() = balance
3. }
5. object Account {
6. def increment(account: Account): Account {
7. **new** Account(account.getBalance() + 1)
8. }
9. }

假设两个人都去get Account，取到了同一个Account，然后increment得到的值应该是相同的，但是他们的本意是加两次，最终得到的却是相同的结果

Actor是一个比Thread更小的并发单位，那么定义在一个Actor内部的状态最多只能同时被一个线程执行，这样如何提高效率？那么我们定义在一个actor中的装填最多只能同时被一个thread操作，只能是并发，而不是真正意义上的并行。而java中，多核情况下，thread会有真正意义上的并行存在。