Kotlin 变量和方法

jinkui 关注

2017.05.25 10:50:15 字数 754 阅读 5,020

变量

Kotlin 有两个关键字定义变量：var 和 val, 变量的类型在后面。
var 定义的是可变变量，变量可以被重复赋值。val 定义的是只读变量，相当于java的final变量。
变量的类型，如果可以根据赋值推测，可以省略。

```
var name: String = "jason"
name = "jame"

val max = 10
```

- **常量**

Java 定义常量用关键字 static final， Kotlin 没有static, 可以在命名对象里面用 const val 定义

```
object Config {
    const val TAG = "config"
}
```

或者伴随对象

```
class Config {
    companion object {
    const val TAG = "config"
}
}
```

方法

方法通过 fun 定义， 包含若干参数和返回值（可选）

```
fun log(msg: String): Unit {
    println(msg)
}
```

方法没有返回值用 Unit, 一般省略不写。

- **单表达式方法**

如果方法只有一个表达式，可以采用“=”的简明写法

```
fun square(x: Int) = x * x
```

- **顶层方法**

Java 的方法都在 class 里面， Kotlin 支持顶层方法，这些方法定义在 class 的外面。
顶层方法一般是一些工具方法，和 Java 将这些方法勉强放到一个不关联的类不同， 单独提供这些方法似乎更合理。

```
fun main(args: ArrayList<String>) {
    //
}

class Hello() {
    //
}
```

- **参数**

Java 不支持默认参数， 只能通过重载定义多个方法，Kotlin支持, 带默认值的参数放后面。

```
fun divide(divisor: BigDecimal, scale: Int = 0): BigDecimal

class Student2(val name: String, val registered: Boolean = false, credits: Int = 0)
```

当调用方法时，Kotlin支持命名参数，这有助于提高代码的可读性，尤其是多参数的方法。带名字的参数放后面。

```
fun deleteFiles(filePattern: String, recursive: Boolean, ignoreCase: Boolean,
        deleteDirectories: Boolean): Unit
{
    // do something
}

deleteFiles("*.jpg", true, true, false)

deleteFiles("*.jpg", recursive = true, ignoreCase = true, deleteDirectories = false)
```

- **不定长度参数**

例如 java `public void println(String.. args) { }` 可以这样定义不定个数参数的函数。
Kotlin 通过关键字 `vararg` 实现这个功能，如示例：

```
fun prints(vararg strings: String) {
    for (string in strings)
    println(string)
}

prints("a", "b", "c")
```

如果已经有一个数组， 可以通过关键字 `*` 传递数组。

```
val strings = arrayOf("a", "b", "c", "d", "e")
prints(*strings)
```

- **返回多个值**

要返回多个值，可以返回数组，或者自定义类型。Kotlin内置了 Pair 和 Triple， 返回2个值和3个值。

```
fun roots(k: Int): Pair<Double, Double> {
    require(k >= 0)
    val root = Math.sqrt(k.toDouble())
    return Pair(root, -root)
}

val (pos, neg) = roots(16)
```

- **尾递归函数(Tail recursive function)**

当一个函数标记为tailrec , 并且满足要求的形式, 编译器就会对代码进行优化, 消除函数的递归调用, 产生一段基于循环实现的, 快速而且高效的代码。

```
tailrec fun findFixPoint(x: Double = 1.0): Double = if (x == Math.cos(x)) x
        else findFixPoint(Math.cos(x))
```

上面的代码计算余弦函数的不动点(fixpoint), 结果应该是一个数学上的常数。这个函数只是简单地从 1.0 开始不断重复地调用 Math.cos 函数, 直到计算结果不再变化为止, 计算结果将是 0.7390851332151607。

要符合 tailrec 修饰符的要求, 函数必须在它执行的所有操作的最后一步, 递归调用它自身。不能将尾递归用在 try/catch/finally 结构内。 尾递归目前只能用在 JVM 环境内。

- **函数引用**

和 C 语言的函数指针类似， 函数可以赋值给变量， 也可以作为高阶函数的参数或者返回值，例如：

```
val printMessage = { message: String -> println(message) }

printMessage("hello")
printMessage("world")
```

参考

《Programming Kotlin》Stephen Samuel ， Stefan Bocutiu
《Kotlin in Action》Dmitry Jemerov，Svetlana Isakova