Kotlin 条件控制 Kotlin 类和对象

Kotlin 循环控制

For 循环

for 循环可以对任何提供迭代器（iterator）的对象进行遍历，语法如下:

```
for (item in collection) print(item)
```

循环体可以是一个代码块:

```
for (item: Int in ints) {
    // ……
}
```

如上所述，for 可以循环遍历任何提供了迭代器的对象。

如果你想要通过索引遍历一个数组或者一个 list，你可以这么做：

```
for (i in array.indices) {
    print(array[i])
}
```

注意这种"在区间上遍历"会编译成优化的实现而不会创建额外对象。

或者你可以用库函数 withIndex：

```
for ((index, value) in array.withIndex()) {
    println("the element at $index is $value")
}
```

实例

对集合进行迭代：

```
fun main(args: Array<String>) {
    val items = listOf("apple", "banana", "kiwi")
    for (item in items) {
        println(item)
    }

    for (index in items.indices) {
        println("item at $index is ${items[index]}")
```

```
    }
}
```

输出结果：

```
apple
banana
kiwi
item at 0 is apple
item at 1 is banana
item at 2 is kiwi
```

---

while 与 do...while 循环

while是最基本的循环，它的结构为：

```
while( 布尔表达式 ) {
  //循环内容
}
```

do…while 循环 对于 while 语句而言，如果不满足条件，则不能进入循环。但有时候我们需要即使不满足条件，也至少执行一次。

do…while 循环和 while 循环相似，不同的是，do…while 循环至少会执行一次。

```
do {
       //代码语句
}while(布尔表达式);
```

实例

```
fun main(args: Array<String>) {
    println("----while 使用-----")
    var x = 5
    while (x > 0) {
        println( x--)
    }
    println("----do...while 使用-----")
    var y = 5
    do {
        println(y--)
    } while(y>0)
}
```

输出结果：

```
5
4
3
2
1
----do...while 使用-----
5
4
3
2
1
```

返回和跳转

Kotlin 有三种结构化跳转表达式：

- *return*。默认从最直接包围它的函数或者匿名函数返回。
- *break*。终止最直接包围它的循环。
- *continue*。继续下一次最直接包围它的循环。

在循环中 Kotlin 支持传统的 break 和 continue 操作符。

```
fun main(args: Array<String>) {
    for (i in 1..10) {
        if (i==3) continue  // i 为 3 时跳过当前循环，继续下一次循环
        println(i)
        if (i>5) break   // i 为 6 时 跳出循环
    }
}
```

输出结果：

```
1
2
4
5
6
```

Break 和 Continue 标签

在 Kotlin 中任何表达式都可以用标签（label）来标记。 标签的格式为标识符后跟 @ 符号，例如：abc@、fooBar@都是有效的标签。 要为一个表达式加标签，我们只要在其前加标签即可。

```
loop@ for (i in 1..100) {
    // ……
}
```

现在，我们可以用标签限制 break 或者continue：

```
loop@ for (i in 1..100) {
    for (j in 1..100) {
        if (……) break@loop
    }
}
```

标签限制的 break 跳转到刚好位于该标签指定的循环后面的执行点。 continue 继续标签指定的循环的下一次迭代。

标签处返回

Kotlin 有函数字面量、局部函数和对象表达式。因此 Kotlin 的函数可以被嵌套。 标签限制的 return 允许我们从外层函数返回。最重要的一个用途就是从 lambda 表达式中返回。回想一下我们这么写的时候：

```
fun foo() {
    ints.forEach {
        if (it == 0) return
        print(it)
    }
}
```

这个 return 表达式从最直接包围它的函数即 foo 中返回。（注意，这种非局部的返回只支持传给内联函数的 lambda 表达式。）如果我们需要从 lambda 表达式中返回，我们必须给它加标签并用以限制 return。

```
fun foo() {
    ints.forEach lit@ {
        if (it == 0) return@lit
        print(it)
    }
}
```

现在，它只会从 lambda 表达式中返回。通常情况下使用隐式标签更方便。 该标签与接受该 lambda 的函数同名。

```
fun foo() {
    ints.forEach {
        if (it == 0) return@forEach
        print(it)
    }
}
```

或者，我们用一个匿名函数替代 lambda 表达式。 匿名函数内部的 return 语句将从该匿名函数自身返回

```
fun foo() {
    ints.forEach(fun(value: Int) {
        if (value == 0) return
        print(value)
    })
}
```

当要返一个回值的时候，解析器优先选用标签限制的 return，即

```
return@a 1
```

意为"从标签 @a 返回 1"，而不是"返回一个标签标注的表达式 (@a 1)"。

← Kotlin 条件控制 Kotlin 类和对象 →

1 篇笔记

写笔记

正常循环：

```
for (i in 1..4) print(i) // 打印结果为: "1234"
```

如果你需要按反序遍历整数可以使用标准库中的 downTo() 函数:

```
for (i in 4 downTo 1) print(i) // 打印结果为: "4321"
```

也支持指定步长：

```
for (i in 1..4 step 2) print(i) // 打印结果为: "13"

for (i in 4 downTo 1 step 2) print(i) // 打印结果为: "42"
```

如果循环中不要最后一个范围区间的值可以使用 until 函数:

```
for (i in 1 until 10) { // i in [1, 10), 不包含 10
     println(i)
}
```

小登 7个月前 (08-23)