一文掌握Kotlin集合

yuanzicheng 关注

0.624 2018.02.19 23:18:18 字数 1,139 阅读 1,890

Kotlin没有重新实现一套集合类，而是在Java的集合类基础上做了一些扩展，所以了解Java集合对掌握Kotlin集合大有帮助。Java集合的知识可以参考：Java集合总结。

1.List：有序可重复

Kotlin中的List分为：不可变 `List` （ReadOnly，Immutable）和可变 `MutableList` （Read&Write，Mutable）

1.1 创建List

创建不可变List

```
val list = listOf()
```

创建可变MutableList

```
val mutableList = mutableListOf()
```

List与MutableList互转

```
list.toMutableList()
mutableList.toList()
```

1.2 遍历List

使用Iterator迭代器

```
val iterator = list.iterator()
while(iterator.hasNext()){
    println(iterator.next())
}
```

使用forEach（以下3种写法都是支持的）

```
list.forEach{
    println(it)
}
list.forEach({
    println(it)
})
list.forEach(::println)
```

带元素索引和值的遍历forEachIndexed

```
//打印索引>0的元素索引和元素值
val list = listOf("aa","bb","cc")
list.forEachIndexed { index, s ->
    if(index > 0){
        println("$index : $s")
    }
}
```

1.3 元素操作

添加元素

```
//添加元素0至末尾
mutableList.add(0)
```

添加元素到指定索引位置

```
//在索引0位置添加元素100
mutableList.add(0,100)
```

移除元素

```
//移除元素0
mutableList.remove(0)
```

按索引移除元素

```
//移除索引0位置的元素
mutableList.removeAt(0)
```

更新元素

```
//更新索引0位置元素为200
 mutableList.set(0, 200)
 mutableList[0] = 200
```

查找元素

```
//查找索引0位置的元素
list[0]
list.elementAt(0)
```

判断是否包含指定元素

```
//判断是否包含元素0
list.contains(0)
```

查找元素索引位置

```
//查找元素0的位置
list.indexOf(0)
```

查找第一个（最后一个）满足条件的元素

```
//查找第一个偶数
list.first { it%2==0 }
//查找最后一个偶数
list.last { it%2==0 }
```

查找第一个（最后一个）满足条件的元素的索引位置

```
//查找第一个偶数位置
list.firstIndexOf { it%2==0 }
//查找最后一个偶数位置
list.lastIndexOf { it%2==0 }
```

查找符合条件的单个元素，元素不存在或超过1个会出现异常

```
//查找唯一的偶数
list.single({it%2==0})
```

1.4 基本操作

判断是否存在元素

```
//判断是否存在元素
list.any()
//判断是否不存在任何元素
list.none()
```

判断是否存在满足条件的元素

```
//判断是否包含偶数
list.any({ it%2==0 })
```

判断所有元素是否都满足条件

```
//判断所有元素是否均为偶数
list.all({ it%2==0 })
```

计算元素个数

```
//计算所有元素个数
list.count()
```

计算满足条件的元素个数

```
//计算偶数个数
list.count({ it%2==0 })
```

累加

```
//累加字符串
val list = listOf("aa","bb","cc")
//从前向后累加，累加时acc在前面
val s1 = list.reduce ({ acc, s -> acc+s })
//从后向前累加，累加时acc在后面
val s2 = list.reduceRight({ acc, s -> acc+s })
println(s1)//结果为aabbcc
println(s2)//结果为aabbcc
```

带初始值的累加fold

```
val s1 = list.fold("初始值", { acc, s -> acc+s })
val s2 = list.foldRight("初始值", { acc, s -> acc+s })
```

最大值/最小值

```
list.max()
list.min()
```

获取函数映射后结果最大/最小的元素

```
//获取平方值最大的元素
val list = listOf(10, -20, 30, -40)
val max = list.maxBy({ it*it })
//获取平方值最小的元素
val min = list.minBy({ it*it })
```

求和（List中的元素必须时数值类型：Byte、Double、Float、Int、Long、Short）

```
list.sum()
```

求元素映射之和

```
//求所有元素的平方之和
list.sumBy({ it*it })
```

1.5 过滤操作

筛选元素

```
//筛选前2个元素
list.take(2)
//筛选后2个元素
list.takeLast(2)
//正向筛选含有字符a的元素，元素不满足条件时终止循环并返回已筛选的子集合
list.takeWhile({it.contains('a')})
//反向筛选含有字符a的元素，元素不满足条件时终止循环并返回已筛选的子集合
list.takeLastWhile({it.contains('a')})
//筛选第0-3个元素
list.slice(0..3)
//筛选第1、3、5个元素
list.slice(listOf(1, 3, 5))
```

去除元素

```
//去除前2个元素
list.drop(2)
//去除后2个元素
list.dropLast(2)
//正向去除含有字符a的元素，元素不满足条件时终止循环并返回剩余元素的集合
list.dropWhile({it.contains('a')})
//反向去除含有字符a的元素，元素不满足条件时终止循环并返回剩余元素的集合
list.dropListWhile({it.contains('a')})
```

过滤元素

```
val list = listOf("aaa","abb","bc","ddd")
//筛选出含有字符a的元素
list.filter { it.contains('a') }
//筛选出不含有字符a的元素
list.filterNot({ it.contains('a') })
//过滤掉null元素
list.filterNotNull()
```

filter与takeWhile的区别：filter遇到不满足条件的元素不会终止循环

1.6 映射操作

转换：map()、flatMap()、mapIndexed()

```
var list = listOf("aaa","bbb","ccc","ddd")
//取字符串List中每个元素的前2个字符组成一个新的List
list = list.map { it.substring(0..1) }  //结果为：["aa","bb","cc"]
list = list.map({it->listOf(it,"===")})  //结果为：[[a, ===], [b, ===], [c, ===]]
list = list.flatMap({it->listOf(it,"===")}) //结果为：[a, ===, b, ===, c, ===]
//mapIndexed带下标的转换
list = list.mapIndexed { index, s -> "$index : $s".format(index,s) }
```

1.7 分组操作

对List中的元素按条件分组，分组条件作为key，满足条件的元素作为value，保存到Map中

```
val list = listOf("aaa", "bbb", "ccc", "ddd")
//按元素中是否含有a字符分组
val map1  = list.groupBy { it.contains("a") }
//按元素中是否含有a字符分组，并且对各个元素进行额外处理
val map2 = list.groupBy({ it.contains("a") }, { list.indexOf(it).toString() + ":" + it
//分组并折叠每个组，统计字符出现的次数
val group = list.groupingBy { it.first() }
group.eachCount().forEach(::println)
```

1.8 排序操作

倒序

```
ist.reversed()
```

升序 & 降序

```
list.sorted()
list.sortedDescending()
```

指定排序条件，升序 & 降序

```
//按字符串长度升序
list.sortedBy { it.length }
//按字符串长度降序
list.sortedByDescending { it.length }
```

1.9 生产操作

zip()：将2个List的元素按下标配对，组成一个Pair作为新List的元素，最终返回新的List（如果List长度不一致，以长度较短的为准）

```
val list1 = listOf(1, 2, 3)
val list2= listOf("aa", "bb", "cc", "dd")
val list3= list1.zip(list2)
```

unzip()：将以Pair为元素的List拆分成2个List为元素新的Pair返回

```
val pair: Pair<List<Int>,List<String>> = list3.unzip()
```

partition()：按元素是否满足条件拆分List，返回一个Pair

```
val list = listOf(1, 2, 3, 4, 5)
val pair = list.partition { it > 3 }
```

plus()：合并两个List，可以用+代替

```
val list1 = listOf(1, 2, 3)
val list2 = listOf(4, 5, 6)
val list3 = list1.plus(list2)
val list4 = list1 + list2
```

2.Set：无序不重复

Kotlin中的Set分为：不可变 `Set` 和可变 `MutableSet` 。

2.1 创建Set

创建可变Set

```
val set = setOf(1, 2, 3)
```

创建不可变MutableSet

```
val mutableSet = mutableSetOf(1, 2, 3)
```

Set与MutableSet互转

```
set.toMutableList()
mutableSet.toSet()
```

Kotlin中的HashSet、LinkedHashSet、SortedSet、TreeSet直接使用Java中的对应的集合类，没有另外实现一遍，只是提供了对应的创建函数。

HashSet

```
>>> val set = hashSetOf(1,2)
>>> set::class
class java.util.HashSet
```

LinkedHashSet

```
>>> val set = mutableSetOf(1,2)
>>> set::class
class java.util.LinkedHashSet
>>> val set = linkedSetOf(1,2)
>>> set::class
class java.util.LinkedHashSet
```

TreeSet

```
>>> val set = sortedSetOf(1,2)
>>> set::class
class java.util.TreeSet
```

2.2 Set的常用操作

并集、交集

```
val set1 = setOf(1, 2, 3)
val set2 = setOf(2, 3, 4)
val set3 = set1.plus(set2)
val set4 = set1 + set2
val set5 = set1.minus(set2)
val set6 = set1 - set2
```

Set中的其它操作如增加、移除等与List几乎一致。

3.Map：无序、key-value对存储、key不重复

Kotlin中的Map分为：只读 `Map` 和可变 `MutableMap` 。

3.1 创建Map

```
>>> val map1 = mapOf(Pair(1, "a"), Pair(2, "b"))
>>> val map2 = mutableMapOf(Pair(1, "a"), Pair(2, "b"))
>>> val map3 = hashMapOf(Pair(1, "a"), Pair(2, "b"))
>>> val map4 = linkedMapOf(Pair(1, "a"), Pair(2, "b"))
>>> val map5 = sortedMapOf(Pair(1, "a"), Pair(2, "b"))
>>> map1::class
class java.util.LinkedHashMap
>>> map2::class
```

```
class java.util.LinkedHashMap
>>> map3::class
class java.util.HashMap
>>> map4::class
class java.util.LinkedHashMap
>>> map5::class
class java.util.TreeMap
```

3.2 访问Map元素

entries属性

```
map1.entries.forEach({println("${it.key} : ${it.value}")})
```

keys属性

```
map1.keys.forEach(::println)
```

values属性

```
map1.values.forEach(::println)
```

通过key获取value

```
//以下两种写法效果是一致的
map1.get(1)
map1[1]
```

通过key获取value，如果不存在返回默认值

```
//返回默认值
map1.getOrDefault(0,"x")
//返回传入函数的默认值
map1.getOrElse(0,{maxOf("aa","bb")})
```

通过key获取value，如果不存在直接抛出异常

```
//抛出java.util.NoSuchElementException
map1.getValue(0)
//返回null
map1.get(0)
```

通过key获取value，如果不存在则添加该元素，其值为传入的函数返回值，并且返回该值（注意：该函数只支持可变的Map）

```
map2.getOrPut(0,{"x"})
```

3.3 其它操作

添加、移除、清空（仅适用可变MutableMap）

```
//设置key=0，value="x"的元素，如果key已存在，则覆盖该元素
map2.put(0, "x")
//移除key=0的元素
map2.remove(0)
//移除key=0，value="x"的元素
map2.remove(0, "x")
//清空元素
map2.clear()
```

是否包含key

```
map1.containsKey(1)
```

是否包含value

```
map1.containsValue("a")
```

Map.Entry<K，V>的操作符函数 component1() 、 component2() ，分别用来直接访问key和value

```
map1.entries.forEach({println("${it.component1()} : ${it.component2()}")})
```

Map.Entry<K，V>.toPair()将key-value对转成Pair

```
map1.entries.forEach({println(it.toPair())})
```

将key映射作为新的key，value不变，返回一个新的Map（ 注意：如果存在key映射后相同，则后面的元素会覆盖前面的元素 ）

```
//将key增大为10倍
val map = map1.mapKeys { it.key * 10 }
```

将value映射后作为新的value，key不变，返回一个新的Map

```
//在value的前后添加#
val map = map1.mapValues ({ "#" + it.value + "#" })
```

过滤满足条件的key对应的元素，组成新的Map返回

```
val map = map1.filterKeys ({ it>0 })
```

过滤满足条件的value对应的元素，组成新的Map返回

```
val map = map1.filterValues ({ it.length>1 })
```

过滤满足条件的Entry，组成新的Map返回

```
val map = map1.filter ({ it.key>0 && it.value.length>1 })
```

Map中的+：如果key不重复，则添加元素；如果key重复，则覆盖该ke y对应的value

```
/*
结果为
1=aa
2=b
3=b
*/
val map2 = mutableMapOf(Pair(1, "a"), Pair(2, "b"))
map2 += mapOf(Pair(1, "aa"), Pair(3, "b"))
map2.entries.forEach(::println)
```