Functional Android (II): Collection operations in Kotlin

by Antonio Leiva | Blog, Development, Kotlin | 4 comments

Lambdas are a great powerful tool to simplify code, but also to do things that were not possible before. We talked about them in the [first part of the series](Unleash functional power on Android (I): Kotlin lambdas).

In the end, lambdas are the basis to implement lots of functional features, such as the ones we are talking today: **Collection operations**. Kotlin provides an awesome set of operations that wouldn't be possible (or really verbose) when using a language which doesn't support lambdas.

This article is not Android specific, but it will boost Android Apps development in many different ways. Today I'll be talking about the different types of collections Kotlin provides, as well as the available operations over them.

Collections

Although we can just use Java collections, Kotlin provides a good set of native interfaces you will want to use:

- **Iterable**: The parent class. Any classes that inherit from this interface represent a sequence of elements we can iterate over.
- **MutableIterable**: Iterables that support removing items during iteration.
- **Collection**: This class represents a generic collection of elements. We get access to functions that return the size of the collection, whether the collection is empty, contains an item or a set of items. All the methods for this kind of collections are only to request data, because collections are immutable.
- **MutableCollection**: a `Collection` that supports adding and removing elements. It provides extra functions such as `add`, `remove` or `clear` among others.
- **List**: Probably the most used collection. It represents a generic ordered collection of elements. As it's ordered, we can request an item by its position, using the `get` function.
- **MutableList**: a `List` that supports adding and removing elements.
- **Set**: an unordered collection of elements that doesn't support duplicate elements.
- **MutableSet**: a `Set` that supports adding and removing elements.
- **Map**: a collection of key-value pairs. The keys in a map are unique, which means we cannot have two pairs with the same key in a map.
- **MutableMap**: a `Map` that supports adding and removing elements.

Collection Operations

This is the set of functional operations we have available over the different collections. I want to show you a little definition and example. It is useful to know what the options are, because that way it's easier to identify where these functions can be used. Please let me know if you miss any function from the standard library.

All this content and much more can be found in Kotlin for Android Developers book.

18.1 Aggregate operations

any

Returns `true` if at least one element matches the given predicate.

```
1 val list = listOf(1, 2, 3, 4, 5, 6)
2 assertTrue(list.any { it % 2 == 0 })
3 assertFalse(list.any { it > 10 })
```

all

Returns `true` if all the elements match the given predicate.

```
1 assertTrue(list.all { it < 10 })
2 assertFalse(list.all { it % 2 == 0 })
```

count

Returns the number of elements matching the given predicate.

```
1 assertEquals(3, list.count { it % 2 == 0 })
```

fold

Accumulates the value starting with an initial value and applying an operation from the first to the last element in a collection.

```
1 assertEquals(25, list.fold(4) { total, next -> total + next })
```

foldRight

Same as `fold`, but it goes from the last element to first.

```
1 assertEquals(25, list.foldRight(4) { total, next -> total + ne
```

forEach

Performs the given operation to each element.

```
1 list forEach { println(it) }
```

forEachIndexed

Same as `forEach`, though we also get the index of the element.

```
list forEachIndexed { index, value
       -> println("position $index contains a $value") }
```

max

Returns the largest element or `null` if there are no elements.

```
assertEquals(6, list.max())
```

maxBy

Returns the first element yielding the largest value of the given function or `null` if there are no elements.

```
// The element whose negative is greater
assertEquals(1, list.maxBy { -it })
```

min

Returns the smallest element or `null` if there are no elements.

```
assertEquals(1, list.min())
```

minBy

Returns the first element yielding the smallest value of the given function or `null` if there are no elements.

```
// The element whose negative is smaller
assertEquals(6, list.minBy { -it })
```

none

Returns `true` if no elements match the given predicate.

```
// No elements are divisible by 7
assertTrue(list.none { it % 7 == 0 })
```

reduce

Same as `fold`, but it doesn't use an initial value. It accumulates the value applying an operation from the first to the last element in a collection.

```
assertEquals(21, list.reduce { total, next -> total + next })
```

reduceRight

Same as `reduce`, but it goes from the last element to first.

```
1 assertEquals(21, list.reduceRight { total, next -> total + nex
```

sumBy

Returns the sum of all values produced by the transform function from the elements in the collection.

```
1 assertEquals(3, list.sumBy { it % 2 })
```

18.2 Filtering operations

drop

Returns a list containing all elements except first n elements.

```
1 assertEquals(listOf(5, 6), list.drop(4))
```

dropWhile

Returns a list containing all elements except first elements that satisfy the given predicate.

```
1 assertEquals(listOf(3, 4, 5, 6), list.dropWhile { it < 3 })
```

dropLastWhile

Returns a list containing all elements except last elements that satisfy the given predicate.

```
1 assertEquals(listOf(1, 2, 3, 4), list.dropLastWhile { it > 4 }
```

filter

Returns a list containing all elements matching the given predicate.

```
1 assertEquals(listOf(2, 4, 6), list.filter { it % 2 == 0 })
```

filterNot

Returns a list containing all elements not matching the given predicate.

```
1 assertEquals(listOf(1, 3, 5), list.filterNot { it % 2 == 0 })
```

filterNotNull

Returns a list containing all elements that are not `null`.

```
assertEquals(listOf(1, 2, 3, 4), listWithNull.filterNotNull())
```

slice

Returns a list containing elements at specified indices.

```
assertEquals(listOf(2, 4, 5), list.slice(listOf(1, 3, 4)))
```

take

Returns a list containing first n elements.

```
assertEquals(listOf(1, 2), list.take(2))
```

takeLast

Returns a list containing last n elements.

```
assertEquals(listOf(5, 6), list.takeLast(2))
```

takeWhile

Returns a list containing first elements satisfying the given predicate.

```
assertEquals(listOf(1, 2), list.takeWhile { it < 3 })
```

18.3 Mapping operations

flatMap

Iterates over the elements creating a new collection for each one, and finally flattens all the collections into a unique list containing all the elements.

```
assertEquals(listOf(1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7), list.
```

groupBy

Returns a map of the elements in original collection grouped by the result of given function

```
assertEquals(mapOf("odd" to listOf(1, 3, 5), "even" to listOf(
            list.groupBy { if (it % 2 == 0) "even" else "odd"
```

map

Returns a list containing the results of applying the given transform function to each element of the original collection.

```
1 assertEquals(listOf(2, 4, 6, 8, 10, 12), list.map { it * 2 })
```

mapIndexed

Returns a list containing the results of applying the given transform function to each element and its index of the original collection.

```
1 assertEquals(listOf (0, 2, 6, 12, 20, 30), list.mapIndexed { i
2         -> index * it })
```

mapNotNull

Returns a list containing the results of applying the given transform function to each non-null element of the original collection.

```
1 assertEquals(listOf(2, 4, 6, 8), listWithNull mapNotNull { it
```

18.4 Elements operations

contains

Returns `true` if the element is found in the collection.

```
1 assertTrue(list.contains(2))
```

elementAt

Returns an element at the given index or throws an `IndexOutOfBoundsException` if the index is out of bounds of this collection.

```
1 assertEquals(2, list.elementAt(1))
```

elementAtOrElse

Returns an element at the given index or the result of calling the default function if the index is out of bounds of this collection.

```
1 assertEquals(20, list.elementAtOrElse(10, { 2 * it }))
```

elementAtOrNull

Returns an element at the given index or `null` if the index is out of bounds of this collection.

```
1 assertNull(list.elementAtOrNull(10))
```

first

Returns the first element matching the given predicate

```
1 assertEquals(2, list.first { it % 2 == 0 })
```

firstOrNull

Returns the first element matching the given predicate, or `null` if no element was found.

```
1 assertNull(list.firstOrNull { it % 7 == 0 })
```

indexOf

Returns the first index of element, or -1 if the collection does not contain element.

```
1 assertEquals(3, list.indexOf(4))
```

indexOfFirst

Returns index of the first element matching the given predicate, or -1 if the collection does not contain such element.

```
1 assertEquals(1, list.indexOfFirst { it % 2 == 0 })
```

indexOfLast

Returns index of the last element matching the given predicate, or -1 if the collection does not contain such element.

```
1 assertEquals(5, list.indexOfLast { it % 2 == 0 })
```

last

Returns the last element matching the given predicate.

```
1 assertEquals(6, list.last { it % 2 == 0 })
```

lastIndexOf

Returns last index of element, or -1 if the collection does not contain element.

```
1 val listRepeated = listOf(2, 2, 3, 4, 5, 5, 6)
2 assertEquals(5, listRepeated.lastIndexOf(5))
```

lastOrNull

Returns the last element matching the given predicate, or `null` if no such element was found.

```
val list = listOf(1, 2, 3, 4, 5, 6)
assertNull(list.lastOrNull { it % 7 == 0 })
```

single

Returns the single element matching the given predicate, or throws exception if there is no or more than one matching element.

```
assertEquals(5, list.single { it % 5 == 0 })
```

singleOrNull

Returns the single element matching the given predicate, or `null` if element was not found or more than one element was found.

```
assertNull(list.singleOrNull { it % 7 == 0 })
```

18.5 Generation operations

merge

Returns a list of values built from elements of both collections with same indexes using the provided transform function. The list has the length of shortest collection.

```
val list = listOf(1, 2, 3, 4, 5, 6)
val listRepeated = listOf(2, 2, 3, 4, 5, 5, 6)
assertEquals(listOf(3, 4, 6, 8, 10, 11), list.merge(listRepeat
        it1 + it2 })
```

partition

Splits original collection into pair of collections, where the first collection contains elements for which the predicate returned `true`,
while the second collection contains elements for which the predicate returned `false`.

```
assertEquals(Pair(listOf(2, 4, 6), listOf(1, 3, 5)),
        list.partition { it % 2 == 0 })
```

plus

Returns a list containing all elements of the original collection and then all elements of the given collection. Because of the name of the function, we can use the '+' operator with it.

```
assertEquals(listOf(1, 2, 3, 4, 5, 6, 7, 8), list + listOf(7,
```

zip

Returns a list of pairs built from the elements of both collections with the same indexes. The list has the length of the shortest collection.

```
1 assertEquals(listOf(Pair(1, 7), Pair(2, 8)), list.zip(listOf(7
```

18.6 Ordering operations

reverse

Returns a list with elements in reversed order.

```
1 val unsortedList = listOf(3, 2, 7, 5)
2 assertEquals(listOf(5, 7, 2, 3), unsortedList.reverse())
```

sort

Returns a sorted list of all elements.

```
1 assertEquals(listOf(2, 3, 5, 7), unsortedList.sort())
```

sortBy

Returns a list of all elements, sorted by the specified comparator.

```
1 assertEquals(listOf(3, 7, 2, 5), unsortedList.sortBy { it % 3
```

sortDescending

Returns a sorted list of all elements, in descending order.

```
1 assertEquals(listOf(7, 5, 3, 2), unsortedList.sortDescending()
```

sortDescendingBy

Returns a sorted list of all elements, in descending order by the results of the specified order function.

```
1 assertEquals(listOf(2, 5, 7, 3), unsortedList.sortDescendingBy
```

Author: Antonio Leiva

I'm in love with Kotlin. I've been learning about it for a couple of years, applying it to Android and digesting all

this knowledge so that you can learn it with no effort.

Twitter Google+ Linkedin Github

Share this:

More

Like this:

Loading...

Related

Kotlin awesome tricks for Android

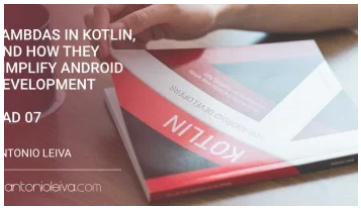
February 4, 2016

In "Blog"

Unleash functional power on Android (I): Kotlin lambdas

September 1, 2015

In "Blog"



Lambdas in Kotlin, and how they simplify Android development (KAD 07)

January 5, 2017

In "Blog"

4 Comments

Carles Lázaro Costa (@Carleslc) on June 20, 2017 at 14:37

Really useful

Reply

shefivp on July 21, 2017 at 19:28

Thanks Brother, Really helpful

Reply

Nameless on September 17, 2017 at 15:51

merge() was dropped as per Kotlin 1.0 so you can replace it with zip(). Also replace reverse() with reversed()

Reply

Antonio Leiva on September 26, 2017 at 23:07

Oh! Will need to review the list yeah. Thanks!

Reply

Trackbacks/Pingbacks

1. Kotlin for Android Developers - Java Advent - [...] may see the complete set of operations in this article. So a complex operation such as a filters, a...

2. Kotlin awesome tricks for Android - Antonio Leiva - [...] can check a complete list of operations for [...]

3. Ten Kotlin Features To Boost Android Development – Android Fan - [...] For an extensive list of functional operations that can be done on collections, check this blog post. [...]

4. Operaciones funcionales con colecciones en Kotlin (KDA 11) - DevExperto - [...] quieres un listado completo, puedes echar un vistazo a este artículo en inglés que escribí hace [...]

5. Functional operations with collections in Kotlin (KAD 11) - Antonio Leiva - [...] you want a more complete list, you can have a look at this article I wrote some time [...]

Home Contact Legal notice Privacy Policy Cookies policy
Terms and Conditions

Designed by **Elegant Themes** | Powered by **WordPress**