1.7. Множества, списки и ассоциативные массивы в Kotlin

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 1.7. Множества, списки и ассоциативные массивы в Kotlin

Напечатано:: Павел Степанов
Дата: вторник, 31 октября 2023, 13:07

Оглавление

1.7.1. Коллекции: списки, множества и ассоциативные массивы

До этого момента мы в основном работали с отдельными значениями разных типов. Зачастую приходится иметь дело с наборами значений, которые во многих языках программирования называются массивами. В Kotlin хранения для различных типов данных существуют коллекции.

Мы разделим коллекции на четыре типа:

Массивы и списки (Arrays, Lists). Массивы - это упорядоченные наборы данных одного типа, удобных для выборки по индексу (номеру) элемента.

Множества и ассоциативные массивы (Maps, Sets). В ряде случаев удобно обращаться к элементам не по индексу, а по произвольному ключу. Так организованы ассоциативные массивы (или словари), состоящие из пар ключ-значение. Ключи в ассоциативном массиве могут быть произвольного типа и не могут повторятся (уникальны), но необязательно упорядочены. Для множества характерны: уникальность элементов, наличие порядка зависит от реализации.

Интерфейсы коллекции и функции для работы с ними расположены в пакете `kotlin.collections`. Пара интерфейсов Для каждого типа коллекции определена пара интерфейсов: только для чтения и изменяемый (mutable).

Будьте внимательны при объявлении переменной для коллекции. Переменная указывает на один и тот же объект даже когда вы изменяете элементы коллекции, поэтому ссылка на саму коллекцию не изменяется. В то же время, если переменную, объявленную как val попытаться задать заново, получите ошибку компиляции.

```
// простой способ создать массив, при этом типы элементов определяются компилятором
val items = arrayOf(1, 2, 3.14, "36.6")
// так же, как в Java, существует специальный класс для работы с массивами
 print(Arrays.toString(items))
// создадим список изменяемого размера
val numbers = mutableListOf ("X", "Y", "Z", "?")
numbers.add ("Q") // можно добавлять и удалять элементы
```

Open in Playground → Target: JVM Running on v.1.9.10

Основа иерархии коллекций - `Collection<T>`. Этот интерфейс представляет собой обычное поведение коллекции только для чтения: получение размера, выборка, поиск элемента и пр. `Collection<T>` наследуется от интерфейса `Iterable <T>`. Удобно использовать `Collection` как параметр функции, которая применяется к разным типам коллекций. Для более конкретных случаев используйте наследников `Collection`: `List`, `Map` и `Set`.

1.7.2. Класс List и массивы

Массивы и списки удобны, если вы храните элементы в определённом порядке. Поиск в них может быть не очень быстрым, т.к. требует перебора всех элементов (в худшем случае) или предварительной сортировки элементов (что затратно).

```
// примеры создания массивов
val vowels = arrayOf("a", "i", "e", "o", "u") // неизменяемый массив
val allFives = Array(5, { 5 }) // пять пятёрок, мы изучим синтаксис подобных лямбда-выражений позже
```

Open in Playground → Target: JVM Running on v.1.9.10

Массивы, получаемые с помощью функции arrayOf являются наборами объектов, даже если хранить в них значения примитивных типов. Это требует расхода памяти и снижает производительности. Для оптимизации быстродействия существуют специализированные массивы, например:

```
val myBytes = byteArrayOf(1, 2, 3)
val toBeOrNotToBe = booleanArrayOf(true, false, false, true)
```

Open in Playground → Target: JVM Running on v.1.9.10

List хранит элементы в указанном порядке и предоставляет к ним доступ по индексу (начинаются с нуля). Существуют также удобные методы `last()`, `first()` и поле `lastIndex` (`firstIndex` = 0 по определению).

```
val numbers = listOf("one", "two", "three", "four")
println("Number of elements: ${numbers.size}")
println("Third element: ${numbers.get(2)}")
println("Fourth element: ${numbers[3]}")
println("Index of element \"two\" ${numbers.indexOf("two")}")
```

Open in Playground → Target: JVM Running on v.1.9.10

Элементы списка (включая нули) могут дублироваться: список может содержать любое количество одинаковых объектов или вхождений одного объекта. Два списка считаются равными, если они одинакового размера и имеют одинаковые элементы в одинаковых позициях.

```
val petya = User("Petya", 30)
val users = listOf(User("Vasya", 40), petya , petya )
val users2 = listOf(User("Vasya", 40), User("Bob", 30), petya)
println(users == users2) // true
petya.age = 123
```

```
println(users == users2) // false
```

Open in Playground → Target: JVM Running on v.1.9.10

Для добавления или удаления элементов существует отдельный интерфейс `MutableList<T>` список с возможностью записи, например, .

```
val numbers = mutableListOf(10, 20, 30, 40)
numbers.add(100)
numbers.removeAt(1)
numbers.shuffle()
numbers[0] = -10
println(numbers)
```

Open in Playground → Target: JVM Running on v.1.9.10

Для выборки нескольких значений (диапазона) удобно использовать метод `slice()`, в качестве параметра которому передаётся диапазон (включая его границы), например:

```
val bookIds = arrayListOf(10, 20, 30, 40)
println(bookIds.slice(1..2))
```

Open in Playground → Target: JVM Running on v.1.9.10

Проверить принадлежность элемента к коллекции можно оператором `in` (инверсия - `!in`) или с помощью метода `contains()`. Во многом списки очень похожи по смыслу на массивы, однако есть важные отличия. Размер массива определяется при его создании далее не изменяется; в свою очередь, размер списка не задан заранее и может быть изменен в результате операций записи: добавления, изменения или удаления элементов. Данные в массиве хранятся в памяти единым блоком, для списка это зависит от реализации (`List<T>` может быть реализован как `ArrayList<T>` или `LinkedList<T>`)

В Kotlin реализацией List по умолчанию является ArrayList, который можно рассматривать как массив с изменяемым размером.

1.7.3. Класс Set

Для работы с множествами существует интерфейс `Set<T>`. В такой коллекции все элементы уникальные, но их порядок в общем случае не определен (для `TreeSet` элементы упорядочены). Значение null считается уникальным и множество может содержать только один null. Множества считаются равным, если содержат идентичный набор элементов .

```
fun main() {
    val intSet = setOf(1, 2, 3, 4)
    println("Number of elements: ${intSet .size}")
    if (intSet .contains(3)) println("3 есть во множестве intSet ")
```

Open in Playground → Target: JVM Running on v.1.9.10

```
val intSetInverted = setOf(4, 3, 2, 1)
println("Множества идентичны: ${numbers == intSetInverted }")
```

По умолчанию множество реализуется через `LinkedHashSet`, в котором порядок добавления элементов сохраняется. Функции, знакомые вам по работе со списками, такие как `first()` или `last()`, работают аналогично и возвращают первый и последний элементы соответственно.

```
val intSet = setOf(1, 2, 3, 4)  // LinkedHashSet is the default implementation
val intSetInverted = setOf(4, 3, 2, 1)
```

Open in Playground → Target: JVM Running on v.1.9.10

println(intSet .first() == intSetInverted.first()) println(intSet .first() == intSetInverted.last())

Альтернативная реализация - `HashSet` - ничего не говорит о порядке элементов, поэтому на вызов функций `first()` или `last()` полагаться не стоит. Однако `HashSet` требует меньше памяти для хранения того же количества элементов.

1.7.4. Класс Map

Часто бывает удобно обращаться к элементам не по номеру (индексу), а по ключу. Для этой цели существует интерфейс `Map<K, V>`. Хотя он не является наследником интерфейса `Collection`, его относят к коллекциям в Kotlin. Ассоциативный массив (словарь) хранит пары ключ-значение (пары или записи); ключи уникальны, но разные ключи могут иметь одинаковые значения. Важным элементом языка являются пары (`Pair`), из которых удобно формировать словари:

```
val countryCodes = mapOf("RU" to "Russia", "FR" ro "France", "PL" to "Poland")
val codes = countryCodes.keys
```

Open in Playground → Target: JVM Running on v.1.9.10

Наличие ключа проверяем знакомым уже оператором `in`.

```
val players = mapOf ("Peter" to 1, "Paul" to 2, "Tom" to 3, "Bob" to 1)
println ("Все ключи: $ {players.keys}")
println ("Все значения: $ {players.values}")
if ("Peter" in players ) println ("Значение по ключу \" Peter\ ": $ {players [" Peter"]}")
if (1 in players.values) println («Значение 1 есть в словаре»)
if (players.containsValue (1)) println ("Значение 1 есть в словаре") // то же, что и предыдущее
```

Open in Playground → Target: JVM Running on v.1.9.10

Два словаря, содержащие идентичные пары, равны независимо порядка этих пар.

```
val moneyMap = mapOf ("USD" to "Dollar", "RUR" to "Ruble", "EUR" to "Euro")
val anotherMap = mapOf ("RUR" to "Ruble", "USD" to "Dollar", "EUR" to "Euro")
println ("Коллекции равны: ${moneyMap == anotherMap}")
```

Open in Playground → Target: JVM Running on v.1.9.10

Стандартная библиотечная функция mapOf создаёт неизменяемую коллекцию. Для добавления/удаления пар и изменения значений используйте `MutableMap` - словарь с операциями записи, например, вы можете добавить новую пару ключ-значение или обновить значение, связанное с данным ключом.

```
val moneyMap = mutableMapOf ("USD" to "Dollar", "RUR" to "Ruble", "EUR" to "Euro")
moneyMap.put ("USD", "Доллар")
moneyMap["RUR"] = "Рубль" // аналогично
```

```
println (moneyMap)
```

По умолчанию словари (ассоциативные массивы) реализуются классом `LinkedHashMap`, который сохраняет порядок вставки элементов. Другая доступная реализация - `HashMap` - не отслеживает порядок элементов.

Начать тур для пользователя на этой странице