

Data class Kotlin

equals and hashCode Java

Содержание

1. Сравнение объектов, equals, hashCode
2. Data class Kotlin
3. Коллизии хешкода

1. Сравнение объектов

Начнем с того, что рассмотрим экстеншн функцию которую мы написали в прошлый раз (или же класс SafeList из предыдущих лекций). Здесь мы не хотим иметь в списке дубликатов – для этого проверяем что уже содержится в списке подобный элемент и не даем ему туда добавиться второй раз. Вроде все прозрачно. Но нет. Давайте на деле напишем класс на джава с 2 полями

```
fun <T> MutableList<T>.addItem(item: T) {  
    if (!contains(item))  
        add(item)  
}
```

И давайте создадим список таких объектов и положим сначала один объект со значениями 1 и “1” и потом еще один объект. Посмотрим что будет

```
public class MyItem {  
  
    private final int i;  
    private final String s;  
  
    public MyItem(int i, String s) {  
        this.i = i;  
        this.s = s;  
    }  
  
    @Override  
    public String toString() {  
        return "MyItem{" +  
            "i=" + i +  
            ", s='" + s + '\'' +  
            '}';  
    }  
}
```

Как видите в логах консоли – у нас 2 объекта с одними и теми же данными.

```
@JvmStatic
fun main(args: Array<String>) {
    val list = mutableListOf(MyItem( i: 1, s: "1"))
    list.addItem(MyItem( i: 1, s: "1"))
    print(list)
}

Main > main()
Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
[MyItem{i=1, s='1'}, MyItem{i=1, s='1'}]
Process finished with exit code 0
```

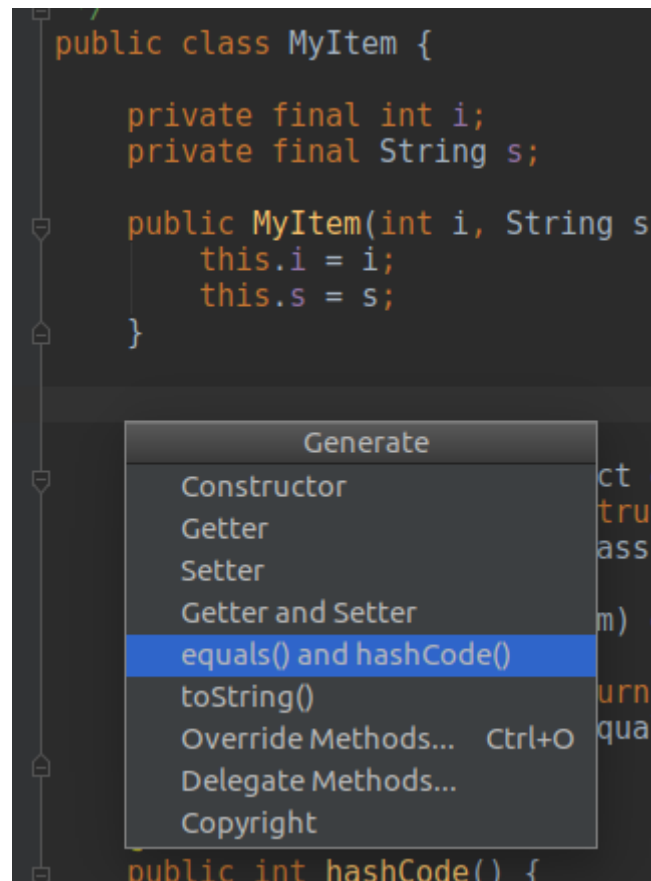
И кто-то скажет – ну так это разные объекты, мы 2 раза вызвали конструктор но с похожими данными. А чего вы ожидали? Давайте проверим так ли это

```
@JvmStatic
fun main(args: Array<String>) {
    val item = MyItem( i: 1, s: "1")
    val list = mutableListOf(item)
    list.addItem(item)
    print(list)
}

Main > main()
Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/b...
[MyItem{i=1, s='1'}]
Process finished with exit code 0
```

Да, если в список добавить ссылку на один и тот же объект, то в списке не будет более 1 элемента. Но секунду. А что если мы хотим считать одним и тем же объектом тот, который состоит из идентичных данных? Как нам это сделать?

Давайте посмотрим что может предложить нам идея для нашего класса в джава. Жмем Alt+Insert. Кроме генерации конструкторов, геттеров и сеттеров идея предлагает нам переопределить методы equals, hashCode. Как видите в одной строке. Одно без другого не работает. Но почему мы должны это сделать? Для этого давайте посмотрим на исходный код ArrayList как он проверяет содержится ли элемент в списке.



Как видите когда мы вызываем contains он проверяет через метод indexOf что тот содержится. И если внимательно посмотреть на самом метод indexOf то он во второй ветке там где аргумент не null смотрит в цикле не равен ли элемент аргументу через метод equals.

```
public boolean contains(Object var1) {  
    return this.indexOf(var1) >= 0;  
}  
  
public int indexOf(Object var1) {  
    int var2;  
    if (var1 == null) {  
        for(var2 = 0; var2 < this.size; ++var2) {  
            if (this.elementData[var2] == null) {  
                return var2;  
            }  
        }  
    } else {  
        for(var2 = 0; var2 < this.size; ++var2) {  
            if (var1.equals(this.elementData[var2])) {  
                return var2;  
            }  
        }  
    }  
    return -1;  
}
```

Да, как видите это метод из класса Object. Если вам надо сравнить 2 объекта на факт того, смотрят ли они на одну и ту же ссылку, то вам ничего не надо делать, а если вы хотите проверить что содержимое 2 объектов идентично, то вы должны переопределить методы equals, hashCode. Я вам предлагаю самостоятельно прочитать про эти методы и как они работают. А пока давайте просто с помощью идеи добавим эти методы в класс джава.

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    MyItem myItem = (MyItem) o;

    if (i != myItem.i) return false;
    return s != null ? s.equals(myItem.s) : myItem.s == null;
}

@Override
public int hashCode() {
    int result = i;
    result = 31 * result + (s != null ? s.hashCode() : 0);
    return result;
}
```

Как видите хешкод это арифметическая сумма числа умноженного на 31 и хешкода строки если она не нул. Т.е. по сути это некая константа. Далее в методе equals мы смотрим на то, что ссылки смотрят на одно и тоже и потом смотрим что объект нул или же их классы различаются. Далее сравниваем поочередно поля класса – сначала число, после строку. Готово. Каждый раз если что-то различается – вернем false. Просто и понятно.

Теперь вернемся в наш мейн и проверим что теперь в списке 1 элемент.

```
fun main(args: Array<String>) {
    val list = mutableListOf(MyItem( i: 1, s: "1"))
    list.addItem(MyItem( i: 1, s: "1"))
    print(list)
}

Main > main()
Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
[MyItem{i=1, s='1'}]
Process finished with exit code 0
```

Да, теперь мы проверяем по содержанию и не кладем в список дубликат. Все просто.

И вернемся к теме джава против котлин. В джава мы часто пишем классы и переопределяем его методы equals, hashCode, toString. В котлин подумали о нас и решили сократить нам время написания кода.

2. Data class Kotlin

Просто пишем ключевое слово data перед объявлением класса и вуаля

```
data class MyItem(private val i: Int, private val s: String)
```

То же самое в джава будет так

```
public class MyItem {

    private final int i;
    private final String s;

    public MyItem(int i, String s) {
        this.i = i;
        this.s = s;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        MyItem myItem = (MyItem) o;

        if (i != myItem.i) return false;
        return s != null ? s.equals(myItem.s) : myItem.s == null;
    }

    @Override
    public int hashCode() {
        int result = i;
        result = 31 * result + (s != null ? s.hashCode() : 0);
        return result;
    }


    @Override
    public String toString() {
        return "MyItem{" +
            "i=" + i +
            ", s='" + s + '\'' +
            '}';
    }
}
```

И это еще 2 поля, а представьте что их 5. В Андроид к этому всему еще прибавляется десятка 2 линий для Parcelable и вуаля – ваш класс на джава сложно прочитать, в нем более 100 линий кода. Просто сравните 36 линий в джава и 1 в котлин.

Так же в котлин доступна функция сору – она упрощает вам жизнь если вам нужен такой же объект, но например в котором 1 поле класса отличается.

3. Коллизии хешкода

Есть одна “маленькая” проблема с этим хешкодом и я вам ее сейчас покажу



```
@JvmStatic
fun main(args: Array<String>) {
    val set = HashSet<Int>()
    val size = 50_000_000
    for(i in 0 until size) {
        set.add(MyItem(i, s: "$i").hashCode())
    }
    print(set.size - size)
}
```

Main > main()

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ..
-470070
Process finished with exit code 0

Кстати, ваш компьютер может не выдержать столько повторений, возьмите поменьше.

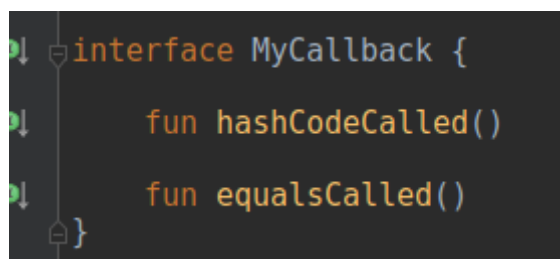
Для начала скажу что HashSet это класс который имплементирует интерфейс Set – тот же лист, но без порядка и там нельзя иметь дубликатов. Да, то что мы хотели сделать в классе SafeList но это не список, так что все элементы хранятся хаотично.

Итак, создаем сет, берем 50 миллионов повторов и в цикле в этот сет кладем хешкоды от объекта с 2 полями. Если бы все хешкоды генерировались по истине разными, то размер сета должен был бы совпасть с количеством повторов. Согласны? По факту в нашем сете на 470070 штук меньше уникальных хешкодов.

Вся пролема с хешкодами в том, что для действительно разных объектов хешкоды могут совпасть.

Где это видно? В коллекциях и в частности в Map. До сих пор мы хранили списки объектов и кучи объектов. Теперь же мы рассмотрим такую коллекцию, где объекты можно получить по ключу. Т.е. у вас теперь не лист, где можно получить объект по индексу и не сет, где объект можно получить в цикле например, а мапа – где мы кладем объект по ключу и достаем по этому же ключу. Считайте что лист это мап с ключом индекс инт.

Давайте напишем интерфейс чтобы посчитать вызовы методов equals и hashCode



```
interface MyCallback {
    fun hashCodeCalled()
    fun equalsCalled()
}
```

Закинем в наш класс и вызовем его.

```
public class MyItem {  
  
    private final int i;  
    private final String s;  
    private final MyCallback callback;  
  
    public MyItem(int i, String s, MyCallback callback) {  
        this.i = i;  
        this.s = s;  
        this.callback = callback;  
    }  
  
    @Override  
    public boolean equals(Object o) {  
        callback.equalsCalled();  
        if (this == o) return true;  
        if (o == null || getClass() != o.getClass()) return false;  
  
        MyItem myItem = (MyItem) o;  
  
        if (i != myItem.i) return false;  
        return s != null ? s.equals(myItem.s) : myItem.s == null;  
    }  
  
    @Override  
    public int hashCode() {  
        callback.hashCodeCalled();  
        int result = i;  
        result = 31 * result + (s != null ? s.hashCode() : 0);  
    }  
}
```

И снаружи будем считать сколько раз вызвали один метод и сколько раз другой.

```
hashCodecalls 15800000  
hashCodecalls 15900000  
equalsscalled 1  
equalsscalled 2  
equalsscalled 3
```

```
equalsscalled 3071  
equalsscalled 3072  
hashCodecalls 16000000  
hashCodecalls 16100000  
hashCodecalls 16200000
```

```
hashCodecalls 20000000  
hashcodes 20000000 equals 3072  
Process finished with exit code 0
```

```

@JvmStatic
fun main(args: Array<String>) {

    val map = HashMap<MyItem, Int>()

    val size = 20_000_000

    var hashCodeCalls= 0
    var equalsCalls = 0

    val callback = object : MyCallback {
        override fun hashCodeCalled() {
            hashCodeCalls++
            if (hashCodeCalls %100000 == 0)
                println("hashcodecalls $hashCodeCalls")
        }

        override fun equalsCalled() {
            equalsCalls++
            println("equalsscalled $equalsCalls")
        }
    }

    for (i in 0 until size) {
        map[MyItem(i, s: "$i",callback)] = i
    }
    print("hashcodes $hashCodeCalls equals $equalsCalls")
}

```

Итак, мы создаем мапу где ключем будет наш класс, а значение инт (неважно). Мы хотим проверить что коллизия хешкодов приводит к тому, что сравнение объектов переходит в ответственность метода equals. Т.е. что происходит когда вы в мапу кладете объект по хешкоду кладется в мапу объект

если хешкоды совпали (а это случается иногда)

мы вызываем метод equals

Как можете видеть из логов – где-то спустя 16 миллионов повторений хешкоды стали коллизировать с существующими. Итого 3072 раза вызывался метод equals. Так что если вы не переопределили в вашем классе equals наравне с hashCode то вы получите проблемы если будете использовать ваш объект как ключ в коллекциях.

То же самое можете проверить и для других коллекций если интересно.

Но опять же предупрежу – ваша машина может испытывать трудности при миллионах повторов.