问题

(1) CopyOnWriteArraySet是用Map实现的吗?

(2) CopyOnWriteArraySet是有序的吗?

(3) CopyOnWriteArraySet是并发安全的吗?

(4) CopyOnWriteArraySet以何种方式保证元素不重复?

(5) 如何比较两个Set中的元素是否完全一致?

简介

CopyOnWriteArraySet底层是使用CopyOnWriteArrayList存储元素的，所以它并不是使用Map来存储元素的。

但是，我们知道CopyOnWriteArrayList底层其实是一个数组，它是允许元素重复的，那么用它来实现CopyOnWriteArraySet怎么保证元素不重复呢?

CopyOnWriteArrayList回顾请点击【死磕 java集合之CopyOnWriteArrayList源码分析】。

源码分析

Set类的源码一般都比较短，所以我们直接贴源码上来一行一行分析吧。

Set之类的简单源码适合泛读，主要是掌握一些不常见的用法，做到心里有说，坐个车三五分钟可能就看完了。

像ConcurrentHashMap、ConcurrentSkipListMap之类的比较长的我们还是倾向分析主要的方法，适合精读，主要是掌握实现原理以及一些不错的思想，可能需要一两个小时才能看完一整篇文章。

```
1.  public class CopyOnWriteArraySet<E> extends AbstractSet<E>
2.          implements java.io.Serializable {
3.      private static final long serialVersionUID = 5457747651344034263L;
4.
5.      // 内部使用CopyOnWriteArrayList存储元素
6.      private final CopyOnWriteArrayList<E> al;
7.
8.      // 构造方法
9.      public CopyOnWriteArraySet() {
10.         al = new CopyOnWriteArrayList<E>();
11.     }
12.
13.     // 将集合c中的元素初始化到CopyOnWriteArraySet中
14.     public CopyOnWriteArraySet(Collection<? extends E> c) {
15.         if (c.getClass() == CopyOnWriteArraySet.class) {
16.             // 如果c是CopyOnWriteArraySet类型，说明没有重复元素，
17.             // 直接调用CopyOnWriteArrayList的构造方法初始化
18.             @SuppressWarnings("unchecked") CopyOnWriteArraySet<E> cc =
19.                 (CopyOnWriteArraySet<E>)c;
20.             al = new CopyOnWriteArrayList<E>(cc.al);
21.         }
22.         else {
23.             // 如果c不是CopyOnWriteArraySet类型，说明有重复元素
24.             // 调用CopyOnWriteArrayList的addAllAbsent()方法初始化
25.             // 它会把重复元素排除掉
26.             al = new CopyOnWriteArrayList<E>();
27.             al.addAllAbsent(c);
28.         }
29.     }
30.
31.     // 获取元素个数
```

```
32.         public int size() {
33.             return al.size();
34.         }
35.
36.         // 检查集合是否为空
37.         public boolean isEmpty() {
38.             return al.isEmpty();
39.         }
40.
41.         // 检查是否包含某个元素
42.         public boolean contains(Object o) {
43.             return al.contains(o);
44.         }
45.
46.         // 集合转数组
47.         public Object[] toArray() {
48.             return al.toArray();
49.         }
50.
51.         // 集合转数组，这里是可能有bug的，详情见ArrayList中分析
52.         public <T> T[] toArray(T[] a) {
53.             return al.toArray(a);
54.         }
55.
56.         // 清空所有元素
57.         public void clear() {
58.             al.clear();
59.         }
60.
61.         // 删除元素
62.         public boolean remove(Object o) {
63.             return al.remove(o);
```

```
64.         }
65.
66.         // 添加元素
67.         // 这里是调用CopyOnWriteArrayList的addIfAbsent()方法
68.         // 它会检测元素不存在的时候才添加
69.         // 还记得这个方法吗？当时有分析过的，建议把CopyOnWriteArrayList拿出来再看看
70.         public boolean add(E e) {
71.             return al.addIfAbsent(e);
72.         }
73.
74.         // 是否包含c中的所有元素
75.         public boolean containsAll(Collection<?> c) {
76.             return al.containsAll(c);
77.         }
78.
79.         // 并集
80.         public boolean addAll(Collection<? extends E> c) {
81.             return al.addAllAbsent(c) > 0;
82.         }
83.
84.         // 单方向差集
85.         public boolean removeAll(Collection<?> c) {
86.             return al.removeAll(c);
87.         }
88.
89.         // 交集
90.         public boolean retainAll(Collection<?> c) {
91.             return al.retainAll(c);
92.         }
93.
94.         // 迭代器
95.         public Iterator<E> iterator() {
96.             return al.iterator();
```

```
        }

        // equals()方法
        public boolean equals(Object o) {
            // 如果两者是同一个对象，返回true
            if (o == this)
                return true;
            // 如果o不是Set对象，返回false
            if (!(o instanceof Set))
                return false;
            Set<?> set = (Set<?>)(o);
            Iterator<?> it = set.iterator();

            // 集合元素数组的快照
            Object[] elements = al.getArray();
            int len = elements.length;

            // 我觉得这里的设计不太好
            // 首先，Set中的元素本来就是不重复的，所以不需要再用个matched[]数组记录有没有出现过
            // 其次，两个集合的元素个数如果不相等，那肯定不相等了，这个是不是应该作为第一要素先检查
            boolean[] matched = new boolean[len];
            int k = 0;
            // 从o这个集合开始遍历
            outer: while (it.hasNext()) {
                // 如果k>len了，说明o中元素多了
                if (++k > len)
                    return false;
                // 取值
                Object x = it.next();
                // 遍历检查是否在当前集合中
                for (int i = 0; i < len; ++i) {
                    if (!matched[i] && eq(x, elements[i])) {
```

```
                        matched[i] = true;
                        continue outer;
                    }
                }
                // 如果不在当前集合中，返回false
                return false;
            }
            return k == len;
        }

        // 移除满足过滤条件的元素
        public boolean removeIf(Predicate<? super E> filter) {
            return al.removeIf(filter);
        }

        // 遍历元素
        public void forEach(Consumer<? super E> action) {
            al.forEach(action);
        }

        // 分割的迭代器
        public Spliterator<E> spliterator() {
            return Spliterators.spliterator
                (al.getArray(), Spliterator.IMMUTABLE | Spliterator.DISTINCT);
        }

        // 比较两个元素是否相等
        private static boolean eq(Object o1, Object o2) {
            return (o1 == null) ? o2 == null : o1.equals(o2);
        }
    }
```

可以看到，在添加元素时调用了CopyOnWriteArrayList的addIfAbsent()方法来保证元素不重复。

还记得这个方法的实现原理吗？点击直达【死磕 java集合之CopyOnWriteArrayList源码分析】。

总结

（1）CopyOnWriteArraySet是用CopyOnWriteArrayList实现的；

（2）CopyOnWriteArraySet是有序的，因为底层其实是数组，数组是不是有序的？！

（3）CopyOnWriteArraySet是并发安全的，而且实现了读写分离；

（4）CopyOnWriteArraySet通过调用CopyOnWriteArrayList的addIfAbsent()方法来保证元素不重复；

彩蛋

（1）如何比较两个Set中的元素是否完全相等？

假设有两个Set，一个是A，一个是B。

最简单的方式就是判断是否A中的元素都在B中，B中的元素是否都在A中，也就是两次两层循环。

其实，并不需要。

因为Set中的元素并不重复，所以只要先比较两个Set的元素个数是否相等，再作一次两层循环就可以了，需要仔细体味。代码如下：

```
public class CopyOnWriteArraySetTest {

    public static void main(String[] args) {
        Set<Integer> set1 = new CopyOnWriteArraySet<>();
        set1.add(1);
        set1.add(5);
        set1.add(2);
        set1.add(7);
//        set1.add(3);
        set1.add(4);

        Set<Integer> set2 = new HashSet<>();
        set2.add(1);
        set2.add(5);
        set2.add(2);
        set2.add(7);
        set2.add(3);

        System.out.println(eq(set1, set2));

        System.out.println(eq(set2, set1));
    }

    private static <T> boolean eq(Set<T> set1, Set<T> set2) {
        if (set1.size() != set2.size()) {
            return false;
        }

        for (T t : set1) {
            // contains相当于一层for循环
            if (!set2.contains(t)) {
                return false;
            }
        }

        return true;
    }
```

```
    }

```

（2）那么，如何比较两个List中的元素是否完全相等呢?

我们知道，List中元素是可以重复的，那是不是要做两次两层循环呢?

其实，也不需要做两次两层遍历，一次也可以搞定，设定一个标记数组，标记某个位置的元素是否找到过，请仔细体味。代码如下:

```
public class ListEqTest {
    public static void main(String[] args) {
        List<Integer> list1 = new ArrayList<>();
        list1.add(1);
        list1.add(3);
        list1.add(6);
        list1.add(3);
        list1.add(8);
        list1.add(5);

        List<Integer> list2 = new ArrayList<>();
        list2.add(3);
        list2.add(1);
        list2.add(3);
        list2.add(8);
        list2.add(5);
        list2.add(6);

        System.out.println(eq(list1, list2));
        System.out.println(eq(list2, list1));
    }

    private static <T> boolean eq(List<T> list1, List<T> list2) {
        if (list1.size() != list2.size()) {
            return false;
        }

        // 标记某个元素是否找到过，防止重复
        boolean matched[] = new boolean[list2.size()];

        outer: for (T t : list1) {
            for (int i = 0; i < list2.size(); i++) {
                // i这个位置没找到过才比较大小
                if (!matched[i] && list2.get(i).equals(t)) {
                    matched[i] = true;
                    continue outer;
                }
            }
            return false;
        }

        return true;
    }
}

```

这种设计是不是很巧妙？^^