目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题 最近阅读

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

07 List 源码会问哪些面试题

更新时间：2019-11-26 09:45:05

勤学如春起之苗，不见其增，日有所长。

——陶潜

引导语

List 作为工作中最常见的集合类型，在面试过程中，也是经常会被问到各种各样的面试题，一般来说，只要你看过源码，心中对 List 的总体结构和细节有所了解的话，基本问题都不大。

1 面试题

1.1 说说你自己对 ArrayList 的理解?

很多面试官喜欢这样子开头，考察面试同学对 ArrayList 有没有总结经验，介于 ArrayList 内容很多，建议先回答总体架构，再从某个细节出发作为突破口，比如这样：
ArrayList 底层数据结构是个数组，其 API 都做了一层对数组底层访问的封装，比如说 add 方法的过程是……（这里可以引用我们在 ArrayList 源码解析中 add 的过程）。

一般面试官看你回答得井井有条，并且没啥漏洞的话，基本就不会深究了，这样面试的主动权就掌握在自己手里面了，如果你回答得支支吾吾，那么面试官可能就会开启自己面试的套路了。

说说你自己对 LinkedList 的理解也是同样套路。

1.2 扩容类问题

1.2.1 ArrayList 无参数构造器构造，现在 add 一个值进去，此时数组的大小是多少，下一次扩容前最大可用大小是多少?

答：此处数组的大小是 1，下一次扩容前最大可用大小是 10，因为 ArrayList 第一次扩容时，是有默认值的，默认值是 10，在第一次 add 一个值进去时，数组的可用大小被扩容到 10 了。

目录

答：这里的考查点就是扩容的公式，当增加到 11 的时候，此时我们希望数组的大小为 11，但实际上数组的最大容量只有 10，不够了就需要扩容，扩容的公式是：oldCapacity + (oldCapacity>> 1)，oldCapacity 表示数组现有大小，目前场景计算公式是：10 + 10 / 2 = 15，然后我们发现 15 已经够用了，所以数组的大小会被扩容到 15。

1.2.3 数组初始化，被加入一个值后，如果我使用 addAll 方法，一下子加入 15 个值，那么最终数组的大小是多少？

答：第一题中我们已经计算出来数组在加入一个值后，实际大小是 1，最大可用大小是 10，现在需要一下子加入 15 个值，那我们期望数组的大小值就是 16，此时数组最大可用大小只有 10，明显不够，需要扩容，扩容后的大小是：10 + 10 / 2 = 15，这时候发现扩容后的大小仍然不到我们期望的值 16，这时候源码中有一种策略如下：

```
// newCapacity 本次扩容的大小，minCapacity 我们期望的数组最小大小
// 如果扩容后的值 < 我们的期望值，我们的期望值就等于本次扩容的大小
if (newCapacity - minCapacity < 0)
    newCapacity = minCapacity;
```

所以最终数组扩容后的大小为 16。

1.2.4 现在我有一个很大的数组需要拷贝，原数组大小是 5k，请问如何快速拷贝？

答：因为原数组比较大，如果新建新数组的时候，不指定数组大小的话，就会频繁扩容，频繁扩容就会有大量拷贝的工作，造成拷贝的性能低下，所以回答说新建数组时，指定新数组的大小为 5k 即可。

1.2.5 为什么说扩容会消耗性能？

答：扩容底层使用的是 System.arraycopy 方法，会把原数组的数据全部拷贝到新数组上，所以性能消耗比较严重。

1.2.6 源码扩容过程有什么值得借鉴的地方？

答：有两点：

- 是扩容的思想值得学习，通过自动扩容的方式，让使用者不用关心底层数据结构的变化，封装得很好，1.5 倍的扩容速度，可以让扩容速度在前期缓慢上升，在后期增速较快，大部分工作中要求数组的值并不是很大，所以前期增长缓慢有利于节省资源，在后期增速较快时，也可快速扩容。
- 扩容过程中，有数组大小溢出的意识，比如要求扩容后的数组大小，不能小于 0，不能大于 Integer 的最大值。

这两点在我们平时设计和写代码时都可以借鉴。

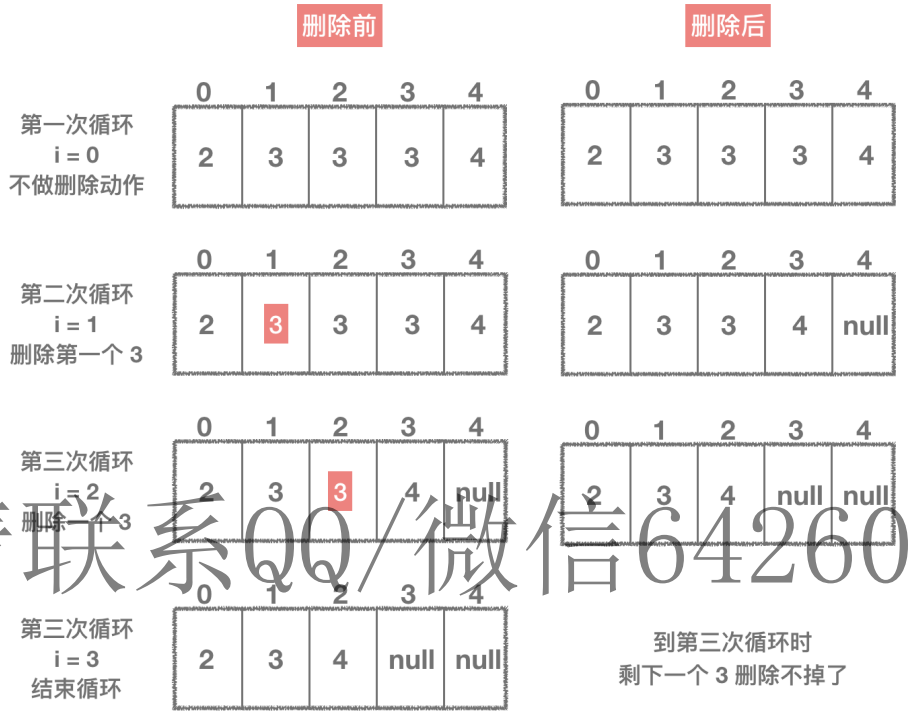
2 删除类问题

2.1 有一个 ArrayList，数据是 2、3、3、3、4，中间有三个 3，现在我通过 for (int i=0;i<list.size ();i++) 的方式，想把值是 3 的元素删除，请问可以删除干净么？最终删除的结果是什么，为什么？删除代码如下：

目录

```
  add("3");
  add("3");
  add("4");
}};
for (int i = 0; i < list.size(); i++) {
  if (list.get(i).equals("3")) {
    list.remove(i);
  }
}
```

答：不能删除干净，最终删除的结果是 2、3、4，有一个 3 删除不掉，原因我们看下图：



从图中我们可以看到，每次删除一个元素后，该元素后面的元素就会往前移动，而此时循环的 i 在不断地增长，最终会使每次删除 3 的后一个 3 被遗漏，导致删除不掉。

2.2 还是上面的 ArrayList 数组，我们通过增强 for 循环进行删除，可以么？

答：不可以，会报错。因为增强 for 循环过程其实调用的就是迭代器的 next () 方法，当你调用 list#remove () 方法进行删除时，modCount 的值会 +1，而这时候迭代器中的 expectedModCount 的值却没有变，导致在迭代器下次执行 next () 方法时，expectedModCount != modCount 就会报 ConcurrentModificationException 的错误。

2.3 还是上面的数组，如果删除时使用 Iterator.remove () 方法可以删除么，为什么？

答：可以的，因为 Iterator.remove () 方法在执行的过程中，会把最新的 modCount 赋值给 expectedModCount，这样在下次循环过程中，modCount 和 expectedModCount 两者就会相等。

2.4 以上三个问题对于 LinkedList 也是同样的结果么？

答：是的，虽然 LinkedList 底层结构是双向链表，但对于上述三个问题，结果和 ArrayList 是一致的。

目录

答：可以先从底层数据结构开始说起，然后以某一个方法为突破口深入，比如：最大的不同是两者底层的数据结构不同，ArrayList 底层是数组，LinkedList 底层是双向链表，两者的数据结构不同也导致了操作的 API 实现有所差异，拿新增实现来说，ArrayList 会先计算并决定是否扩容，然后把新增的数据直接赋值到数组上，而 LinkedList 仅仅只需要改变插入节点和其前后节点的指向位置关系即可。

3.2 ArrayList 和 LinkedList 应用场景有何不同

答：ArrayList 更适合于快速的查找匹配，不适合频繁新增删除，像工作中经常会对元素进行匹配查询的场景比较合适，LinkedList 更适合于经常新增和删除，对查询反而很少的场景。

3.3 ArrayList 和 LinkedList 两者有没有最大容量

答：ArrayList 有最大容量的，为 Integer 的最大值，大于这个值 JVM 是不会为数组分配内存空间的，LinkedList 底层是双向链表，理论上可以无限大。但源码中，LinkedList 实际大小用的是 int 类型，这也说明了 LinkedList 不能超过 Integer 的最大值，不然会溢出。

3.4 ArrayList 和 LinkedList 是如何对 null 值进行处理的

答：ArrayList 允许 null 值新增，也允许 null 值删除。删除 null 值时，是从头开始，找到第一值是 null 的元素删除；LinkedList 新增删除时对 null 值没有特殊校验，是允许新增和删除的。

3.5 ArrayList 和 LinedList 是线程安全的么？为什么？

答：当两者作为非共享变量时，比如说仅仅是在方法里面的局部变量时，是没有线程安全问题的，只有当两者是共享变量时，才会有线程安全问题。主要的问题点在于多线程环境下，所有线程任何时刻都可对数组和链表进行操作，这会导致值被覆盖，甚至混乱的情况。

如果有线程安全问题，在迭代的过程中，会频繁报 ConcurrentModificationException 的错误，意思是在我当前循环的过程中，数组或链表的结构被其它线程修改了。

3.6 如何解决线程安全问题？

Java 源码中推荐使用 Collections#synchronizedList 进行解决，Collections#synchronizedList 的返回值是 List 的每个方法都加了 synchronized 锁，保证了在同一时刻，数组和链表只会被一个线程所修改，或者采用 CopyOnWriteArrayList 并发 List 来解决，这个类我们后面会说。

4 其它类型题目

4.1 你能描述下双向链表么？

答：如果和面试官面对面沟通的话，你可以去画一下，可以把《LinkedList 源码解析》中的 LinkedList 的结构画出来，如果是电话面试，可以这么描述：双向链表中双向的意思是说前后节点之间互相有引用，链表的节点我们称为 Node。Node 有三个属性组成：其前一个节点，本身节点的值，其下一个节点，假设 A、B 节点相邻，A 节点的下一个节点就是 B，B 节点的上一个节点就是 A，两者互相引用，在链表的头部节点，我们称为头节点。头节点的前一个节点是 null，尾部称为尾节点，尾节点的后一个节点是 null，如果链表数据为空的话，头尾节点是同一个节点，本身是 null，指向前后节点的值也是 null。

目录

合：如果是面对面沟通，最好可以直接画图，如果是电话面试，可以这么描述：

新增：我们可以选择从链表头新增，也可以选择从链表尾新增，如果是从链表尾新增的话，直接把当前节点追加到尾节点之后，本身节点自动变为尾节点。

删除：把删除节点的后一个节点的 prev 指向其前一个节点，把删除节点的前一个节点的 next 指向其后一个节点，最后把删除的节点置为 null 即可。

总结

List 在工作中经常遇到，熟读源码不仅仅是为了应对面试，也为了在工作中使用起来得心应手，如果想更深入了解 List，可以看一遍 ArrayList 源码之后，自己重新实现一个 List。这样的话，就会对 List 底层的数据结构和操作细节理解更深。

← 06 LinkedList 源码解析 08 HashMap 源码解析 →

精选留言 20

欢迎在这里发表留言，作者筛选后可公开显示

Aegon_Targaryen

其实要删干净很简单啊，直接 while(list.remove(elem));

0 回复 2019-11-29

文贺 回复 **Aegon_Targaryen**

很有道理，给你点赞。

回复 2019-11-30 13:12:58

慕粉4318313

老师你好，链表可以从指定位置插入吗

0 回复 2019-10-28

文贺 回复 **慕粉4318313**

可以的哈，add 方法可以指定插入的位置的。

回复 2019-10-31 12:52:01

威先森

2.3使用iterator迭代删除能把3都删除，因为在iterator的remove方法中，删除元素后，又将指针指回上一次的位置了。next方法中将cursor变量+1，remove里面--cursor了，所以迭代器能把3删除干净。

2 回复 2019-10-14

目录

回复 2019-10-22 15:18:34

锋Ginger 回复 **慕粉3445147**

for (String string : list) { list.remove(string) } 老师说的是这种情况

回复 9天前

音殿

请教一下上面所说的arrayList 无参构造 第一次add一个进去 此时数组长度是1 然后add的时候就是第一次扩容的时候 第一次扩容默认值是10 就是说现在还可以add9个值进去的意思 请问我理解的对吗?

0 回复 2019-10-11

文贺 回复 **音殿**

同学你好，是的是的。

回复 2019-10-12 18:49:56

buxingshi

严格来说，2.2 用增强for循环为删除元素，可以删一次，因为第一次删除之前modCount还未改变。

0 回复 2019-09-24

文贺 回复 **buxingshi**

不好意思同学，漏掉了你的留言，有且仅当 list 大小是2时，第一次删除时才会通过，其余的都会报错哈，场景有丢丢极限。

回复 2019-10-12 19:10:00

Seed2009

3.3 里一个LinkedList拼错了，无伤大雅

0 回复 2019-09-22

文贺 回复 **Seed2009**

谢谢同学提醒，3.3 中木有找到拼错的单词呢。

回复 2019-09-23 10:45:59

licly

老师，这里说的不太全面。如果是顺序插入的话，ArrayList是比linkedlist快的，即调用无参的add方法；linkedlist只有随机插入的时候可能快于ArrayList，即调用add(int index, E element)方法插入，这个可能因素决定于ArrayList移动的元素个数。一般情况下，都是使用add（）无参方法，所以ArrayList使用的多。

0 回复 2019-09-20

文贺 回复 **licly**

是的，add 方法有两个，我们基本用的是无参的 add

回复 2019-09-23 10:44:52

目录

ADC之父

纠正一点，ArrayList并不擅长查询O(n)，而是根据索引随机读写O(1)。

0 回复 2019-09-19

小明12345 回复 **ADC之父**

arraylist为什么不擅长查询?

回复 2019-11-11 17:52:16

温柔的微笑

老师您好我有个问题：当指定addAll操作时，源码中会将参数集合调用toArray方法，此时内存复制一个新的数组吗，这样内部占用就会加倍

1 回复 2019-09-18

文贺 回复 **温柔的微笑**

toArray() 底层使用的是 Arrays 的 copyOf 方法，底层会返回新的数组。 如果可以预见 addAll 的集合特别大，可以进行分段插入，防止一次性 addAll 耗时久的话，引起 JVM 的 full gc。日常工作中，一般来说，无需担忧这个问题哈，主要是因为 addAll 执行很快，在 addAll 执行完成之后，数组很快就会被 jvm 回收掉了。

回复 2019-09-18 21:12:20

bb111323

您好，我想问一下那个ArrayList无产构造后，add一个值，此时list的size为1，但对于存储数据的数组为什么长度是1，不应该是10吗？ 数组的长度和可用大小难道不是一个概念吗?

1 回复 2019-09-13

文贺 回复 **bb111323**

文章没有描述过数组的长度是1哈，一般List 大小有两种称呼方式，第一种就是直接叫做数组的大小，指的是数组实际有多少个的元素，就是源码中的 size，第二种叫做数组的可用大小，或数组的最大可用大小，或数组的容量，就是源码中的 capacity。

回复 2019-09-15 09:53:59

大胖睛

老师，我想问下，5k初始化的话，是不是放的字节的大小，5 * 1024 * 8嘛?

0 回复 2019-09-10

文贺 回复 **大胖睛**

点击展开剩余评论

千学不如一看，千看不如一练

目录