知识点总结

1. ArrayList

- 底层数组，操作尾部速度快，操作中间速度慢。
- 并集、交集、差集
- 随机访问速度快
- 扩容，先1.5倍，不够再按需

2. LinkedList

- 底层双链表
- 不支持随机访问
- 具有队列、双端队列、栈的特性

3. CopyOnWriteArrayList

- 读写分离，读不加锁，写加锁，加ReentrantLock重入锁
- 写操作是复制一份数组，修改完后再更新回去，性能较低
- 底层是数组，支持随机读
- 只能最终一致性，不能实时一致性
- 适合读多写少场合

4. HashMap

- 数组 + 链表 + 红黑树
- 2倍扩容
- 桶>64，链表元素>8，树化
- 链表元素<6反树化
- hash值插入位置，所以无序

5. LinkedHashMap

- 继承HashMap，具有HashMap特性
- 内部维护双向链表，所以有序，可以按插入顺序或者访问顺序

6. TreeMap

- 红黑树
- 按key排序
- 访问速度比hashMap慢

7. WeakHashMap

- 数组 + 链表
- gc时会清除掉key
- 每次对map的操作都会剔除失效key对应的Entry；(关联队列里有gc掉的key)
- 使用String作为key时，一定要使用new String()这样的方式声明key，才会失效，其它的基本类型的包装类型是一样的；

8. ConcurrentHashMap

- 线程安全
- 数组 + 链表 + 红黑树
- java7分段锁，java8采用CAS

9. ConcurrentSkipListMap

- 线程安全
- 跳表，底层所有数据（有序），越往上数据越少（对半），二分法思想，查找数据块
- 插入时，随机看那一层，再插入到相应的位置

10. HashSet

- HashMap的key实现
- 无序，因为HashMap无序
- 可以一个null，因为HashMap允许key为null；

11. LinkedHashSet

- LinkedHashSet的底层使用LinkedHashMap存储元素。
- LinkedHashSet是有序的，它是按照插入的顺序排序的。

12. TreeSet

- TreeSet底层使用NavigableMap存储元素；
- 有序（key的自然排序）

13. CopyOnWriteArraySet

- 用CopyOnWriteArrayList实现的；
- 有序的，底层是数组
- 线程安全
- 读写分离

14. ConcurrentSkipListSet

- ConcurrentSkipListSet底层是使用ConcurrentNavigableMap实现的；
- ConcurrentSkipListSet有序的，基于元素的自然排序或者通过比较器确定的顺序；
- ConcurrentSkipListSet是线程安全的；

15. PriorityQueue

- 数组存储+堆排序
- 线程不安全
- 无序，只有堆顶是最大或最小

16. ArrayBlockingQueue

- 数组，指定容量，循环利用数组
- 只使用了一个锁来控制入队出队，效率较低。
- 线程安全

17. LinkedBlockingQueue

- 单链表
- 有界，不指定默认最大int值
- 两把锁的锁分离技术实现入队出队互不阻塞
- 线程安全

18. ArrayDeque

- 数组实现双端队列
- 出队入队是通过头尾指针循环利用数组实现的
- 可以直接作为栈使用
- 按一倍扩容

彩蛋

为什么CopyOnWriteArrayList没有size属性?

因为每次修改都是拷贝一份正好可以存储目标个数元素的数组，所以不需要size属性了，数组的长度就是集合的大小，而不像ArrayList数组的长度实际是要大于集合的大小的。

比如，add(E e)操作，先拷贝一份n+1个元素的数组，再把新元素放到新数组的最后一位，这时新数组的长度为len+1了，也就是集合的size了。