容器：

18.Java容器有哪些

分为Collection和Map两大类

Collection分为Queue,List,Set //都是接口，例如List<Integer> xx=new ArrayList<Integer>，子类必须实现接口所有功能才能实例化，不然看作抽象类

List：想想arrayList，特点就是有序，允许重复

下有ArrayList, LinkedList, Vector

Stack继承了Vector

Set: 集合，默认无序，不允许重复

下有HashSet, TreeSet（他是排序的，不是先进去Index就是0，而是561234这样插进去，会自动排成123456）

Queue: 单进单出，只能操作两端

常用实现类是LinkedList, Priority Queue

Map:特点就是Key,Value

常用实现类是HashMap,HashTable(别用，hashMap父类是AbstractMap,而HashTable父类是Dictionary,Dictionary已经被废弃了，所以它的子类也被废弃了),ConcurrentHashMap作为代替

2. 请谈一谈Java集合中的fail-fast和fail-safe机制

fail-fast是一种错误检测机制，Java在适合单线程使用的集合容器中很好地实现了fail-fast机制，举一个简单的例子：在多线程并发环境下，A线程在通过迭代器遍历一个ArrayList集合，B线程同时对该集合进行增删元素操作，这个时候线程A就会抛出并发修改异常，中断正常执行的逻辑。

java.util包下的集合类都是快速失败的，不能在多线程下发生并发修改（迭代过程中被修改）

而fail-safe机制更像是一种对fail-fast机制的补充，它被广泛地实现在各种并发容器集合中。回头看上面的例子，如果线程A遍历的不是一个ArrayList，而是一个CopyOnWriteArrayList，则符合fail-safe机制，线程B可以同时对该集合的元素进行增删操作，线程A不会抛出任何异常。

场景：java.util.concurrent包下的容器都是安全失败，可以在多线程下并发使用，并发修改。

一：快速失败（fail—fast）

         集合对象有一个modCount变量，如果遍历过程中对集合对象的内容进行了修改（增加、删除、修改），modCount自增，每当迭代器使用hashNext()/next()遍历下一个元素之前（每次获取下一个元素的时候），都会检测modCount变量是否为expectedmodCount值，如果不相等（也就是另外一个进程也改变了这个集合对象）就抛出异常。

二：安全失败（fail—safe）

      采用安全失败机制的集合容器，在遍历时不是直接在集合内容上访问的，而是先复制原有集合内容，在拷贝的集合上进行遍历。

      原理： 当使用迭代器遍历集合时，会基于原数组拷贝出一个新的数组（ArrayList的底层是数组），后续的遍历行为在新数组上进行。所以线程B同时进行增删操作不会影响到线程A的遍历行为。

### 3. 如何一边遍历一边删除Collection中的元素？

Java中的Iterator功能比较简单，并且只能单向移动：

(1) 使用方法iterator()要求容器返回一个Iterator。第一次调用Iterator的next()方法时，它返回序列的第一个元素。注意：iterator()方法是java.lang.Iterable接口,被Collection继承。

(2) 使用next()获得序列中的下一个元素。

**(3) 使用hasNext()检查序列中是否还有元素。**

**(4) 使用remove()将迭代器新返回的元素删除。**

Iterator是Java迭代器最简单的实现，为List设计的ListIterator具有更多的功能，它可以从两个方向遍历List，也可以从List中插入和删除元素。

使用集合迭代器自身的remove方法进行删除

Iterator<Integer> it = list.iterator();

while(it.hasNext()){

\*// do something\*

it.remove();

}

### 4. 谈谈ArrayList和LinkedList的区别

本质的区别来源于两者的底层实现：ArrayList的底层是数组，LinkedList的底层是双向链表。

数组拥有O(1)的查询效率，可以通过下标直接定位元素，支持随机访问；链表在查询元素的时候只能通过遍历的方式查询，是O(n)

数组增删元素的效率比较低，通常要伴随拷贝数组的操作；链表增删元素的效率很高，只需要调整对应位置的指针即可。

19. Collection 和 Collections 有什么区别？

Collection 是一个集合接口，它提供了对集合对象进行基本操作的通用接口方法，所有集合都是它的子类，比如 List、Set 等。

Collections 是一个包装类，包含了很多静态方法，不能被实例化，就像一个工具类，比如提供的排序方法： Collections. sort(list)。

20.List、Set、Map 之间的区别是什么？

****

21. HashMap 和 Hashtable 有什么区别？

hashMap允许空键值，而hashTable不允许。

在JDK1.7之前，两者的实现极为相似，最大的区别在于HashTable的方法都用synchronized关键字修饰起来了，表明它是线程安全的。

但是由于直接在方法上加synchronized关键字的同步效率较低，在并发情况下，官方推荐我们使用ConcurrentHashMap。

所以我们看到在JDK1.8中，官方甚至没有对HashTable进行链表转树这样的优化，HashTable已经不被推荐使用了。

**22. 如何决定使用 HashMap 还是 TreeMap？**

对于在Map中插入、删除和定位元素这类操作，HashMap是最好的选择。然而，假如你需要对一个有序的key集合进行遍历，TreeMap是更好的选择。基于你的collection的大小，也许向HashMap中添加元素会更快，将map换为TreeMap进行有序key的遍历。

**23. 说一下 HashMap 的实现原理？**

HashMap概述： HashMap是基于哈希表的Map接口的非同步实现。此实现提供所有可选的映射操作，并允许使用null值和null键。此类不保证映射的顺序，特别是它不保证该顺序恒久不变。

HashMap的数据结构： 在java编程语言中，最基本的结构就是两种，一个是数组，另外一个是模拟指针（引用），所有的数据结构都可以用这两个基本结构来构造的，HashMap也不例外。HashMap实际上是一个“链表散列”的数据结构，**即数组和链表的结合体。**

当我们往Hashmap中put元素时,首先**根据key的hashcode重新计算hash值,根据hash值得到这个元素在数组中的位置(下标),**如果该数组在该位置上已经存放了其他元素,比较，如果key相同，替换value，如果key不同，在jdk1.8前插入链头，在1.8后插入链尾.如果数组中该位置没有元素,就直接将该元素放到数组的该位置上。

需要注意Jdk 1.8中对HashMap的实现做了优化,当链表中的节点数据超过八个之后,该链表会转为红黑树来提高查询效率,从原来的O(n)到O(logn)

24. 说一下 HashSet 的实现原理？

HashSet底层由HashMap实现

HashSet的值存放于HashMap的key上

HashMap的value统一为PRESENT

26. 如何实现数组和 List 之间的转换？

List转换成为数组：调用ArrayList的toArray方法。

数组转换成为List：调用Arrays的asList方法。

5. 谈谈ArrayList和Vector的区别

两者的底层实现相似，关键的不同在于Vector的对外提供操作的方法都是用synchronized修饰的，也就是说Vector在并发环境下是线程安全的，而ArrayList在并发环境下可能会出现线程安全问题。

由于Vector的方法都是同步方法，执行起来会在同步上消耗一定的性能，所以在单线程环境下，Vector的性能是不如ArrayList的

ArrayList更加通用，因为我们可以使用Collections工具类轻易地获取同步列表和只读列表。

### ****28. Array 和 ArrayList 有何区别？****

* Array可以容纳基本类型和对象，而ArrayList只能容纳对象，但是JAVA提供了封装类，通过自动封装你用arrayList.add(5)也是不会报错的。
* Array是静态的，初始时必须指定大小，但是arrayList是动态的，根据需求可以扩大内存，
* ArrayList提供了比Array更多的迭代方法例如Iterator
* ArrayList允许你删除元素，不是简单地把元素设置成null，而是帮你把后面的元素往前缩一格

### 6. 为什么ArrayList的elementData数组要加上transient修饰

由于ArrayList有自动扩容机制，所以ArrayList的elementData数组大小往往比现有的元素数量大，如果不加transient直接序列化的话会把数组中空余的位置也序列化了，浪费不少的空间。

ArrayList中重写了序列化和反序列化对应的writeObject和readObject方法，在遍历数组元素时，以size作为结束标志，只序列化ArrayList中已经存在的元素。

序列化：将一个对象转换成一串二进制表示的字节数组，通过保存或转移这些字节数据来达到持久化的目的。  
反序列化：将字节数组重新构造成对象。



给一个attribute加上transient以后不会序列化塔

### ****29. 在 Queue 中 poll()和 remove()有什么区别？****

poll() 和 remove() 都是从队列中取出一个元素，但是 poll() 在获取元素失败的时候会返回空，但是 remove() 失败的时候会抛出异常。

### ****30. 哪些集合类是线程安全的？****

* vector：就比arraylist多了个同步化机制（线程安全），因为效率较低，现在已经不太建议使用。在web应用中，特别是前台页面，往往效率（页面响应速度）是优先考虑的。
* statck：堆栈类，先进后出。
* hashtable：就比hashmap多了个线程安全。
* enumeration：枚举，相当于迭代器。

### 11. 为什么HashMap中适合用Integer，String这样的基础类型作为key？

因为这些基础类内部已经重写了hashCode和equals方法，遵守了HashMap内部的规范。

#### 追问：如果要用我们自己实现的类作为key，要注意什么？

一定要重写hashCode()和equals()方法，而且要遵从以下规则：

equals()是我们判断两个对象是否相同的依据，如果我们重写了equals方法，用自己的逻辑去判断两个对象是否相同，那么一定要保证：

两个equals()返回true的对象，一定要返回相同的hashCode。

这样，在HashMap的put方法中才能正确判断key是否相同。

不是经常有一个问题嘛，两个对象hashCode相同，equals一定返回true吗？答案肯定是否的，这和你的设计密切相关：如果在你的编程思路中这两个对象是不同的，那么就算恰巧两个对象的hashCode相同，equals也应该返回false。

### 10. 请说一下HashMap的扩容原理

1. 首先得到新的容量值和新的扩容阈值，默认都是原来大小的两倍。
2. 然后根据新容量创建新的数组
3. 最后把元素从旧数组中迁移到新数组中

### 9. 为什么要把链表转为红黑树，阈值为什么是8？

在极端情况下，比如说key的hashCode()返回的值不合理，或者多个密钥共享一个hashCode，很有可能会在同一个数组位置产生严重的哈希冲突。

这种情况下，如果我们仍然使用使用链表把多个冲突的元素串起来，这些元素的查询效率就会从O(1)下降为O(N)。为了能够在这种极端情况下仍保证较为高效的查询效率，HashMap选择把链表转换为红黑树，红黑树是一种常用的平衡二叉搜索树，添加，删除，查找元素等操作的时间复杂度均为O(logN)

至于阈值为什么是8，这是HashMap的作者根据概率论的知识得到的。当key的哈希码分布均匀时，数组同一个位置上的元素数量是成泊松分布的，同一个位置上出现8个元素的概率已经接近千分之一了，这侧面说明如果链表的长度达到了8，key的hashCode()肯定是出了大问题，这个时候需要红黑树来保证性能，所以选择8作为阈值。