数据结构面试专题

1、常用数据结构简介

数据结构是指相互之间存在着一种或多种关系的数据元素的集合和该集合中数据元素间的关系组成。常用的数据有：数组、栈、队列、链表、树、图、堆、散列表。

1）数组：在内存中连续存储多个元素的结构。数组元素通过下标访问，下标从 0 开始。优点：访问速度快；缺点：数组大小固定后无法扩容，只能存储一种类型的数据，添加删除操作慢。适用场景：适用于需频繁查找，对存储空间要求不高，很少添加删除。

2）栈：一种特殊的线性表，只可以在栈顶操作，先进后出，从栈顶放入元素叫入栈，从栈顶取出元素叫出栈。应用场景：用于实现递归功能，如斐波那契数列。

3）队列：一种线性表，在列表一端添加元素，另一端取出，先进先出。使用场景：多线程阻塞队列管理中。

4）链表：物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表的指针地址实现，每个元素包含两个结点，一个是存储元素的数据域，一个是指向下一个结点地址的指针域。有单链表、双向链表、循环链表。优点：可以任意加减元素，不需要初始化容量，添加删除元素只需改变前后两个元素结点的指针域即可。缺点：因为含有大量指针域，固占用空间大，查找耗时。适用场景：数据量小，需频繁增加删除操作。

5）树：由 n 个有限节点组成一种具有层次关系的集合。二叉树（每个结点最多有两个子树，结点的度最大为 2，左子树和右子树有顺序）、红黑树（HashMap 底层源码）、B+树（mysql 的数据库索引结构）

6）散列表（哈希表）：根据键值对来存储访问。

7）堆：堆中某个节点的值总是不大于或不小于其父节点的值，堆总是一棵完全二叉树。

8）图：由结点的有穷集合 V 和边的集合 E 组成。

2、并发集合了解哪些?

1）并发 List，包括 Vector 和 CopyOnWriteArrayList 是两个线程安全的 List，Vector 读写操作都用了同步，CopyOnWriteArrayList 在写的时候会复制一个副本，对副本写，写完用副本替换原值，读时不需要同步。

2）并发 Set，CopyOnWriteArraySet 基于 CopyOnWriteArrayList 来实现的，不允许存在重复的对象。

3）并发 Map，ConcurrentHashMap，内部实现了锁分离，get 操作是无锁的。

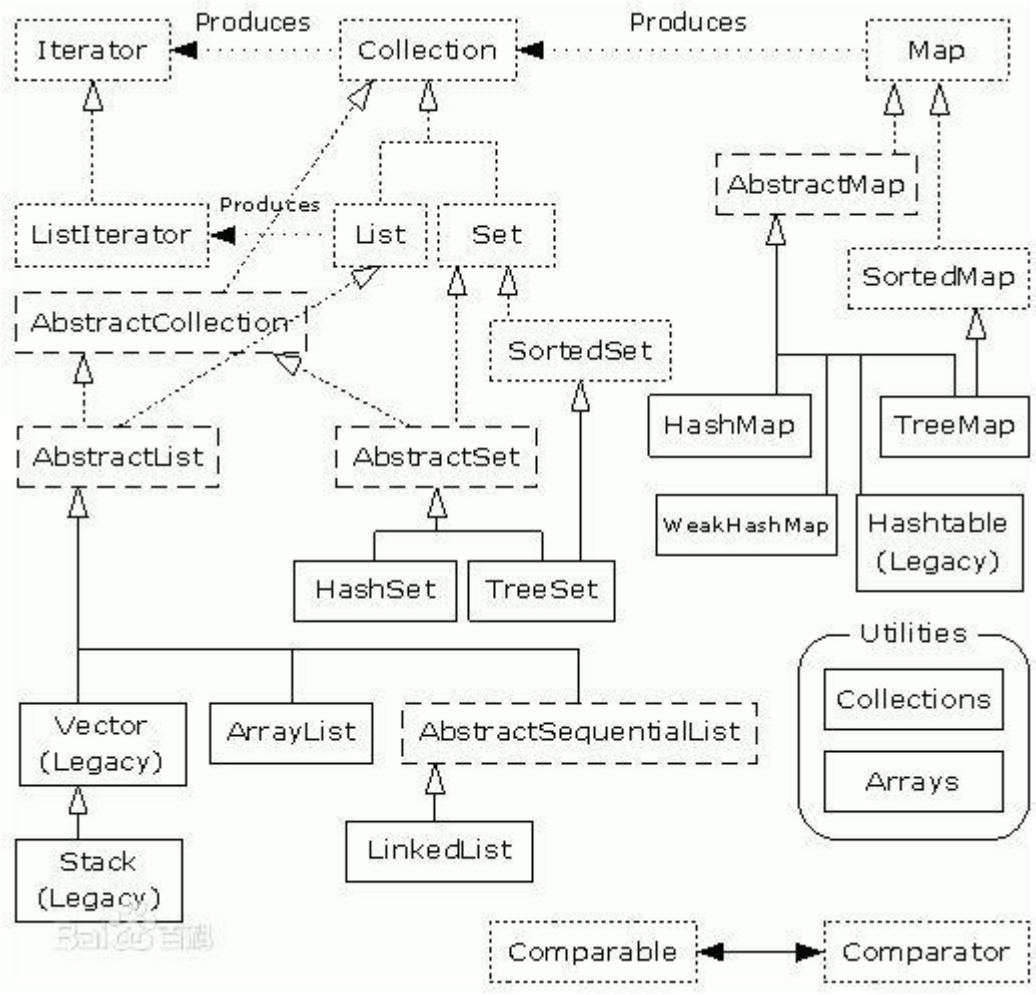
4）并发 Queue，ConcurrentLinkedQueue 适用于高并发场景下的队列，通过无锁方式实现。BlockingQueue 阻塞队列，应用场景，生产者-消费者模式，若生产快于消费，生产队列装满时会阻塞，等待消费。

5）并发 Deque，LinkedBlockingDueue 没有进行读写锁分离，同一时间只能有一个线程对其操作。

6）并发锁重入锁 ReentrantLock，互斥锁，一次最多只能一个线程拿到锁。

7）读写锁 ReadWriteLock，有读取和写入锁两种，读取允许多个读取线程同时持有，而写入只能有一个线程持有。

3、列举 java 的集合以及集合之间的继承关系



5、容器类介绍以及之间的区别

1）Collection 接口：集合框架的根接口，它是集合类框架中最具一般性的顶层接口。

2）Map 接口：提供了键值对的映射关系的集合，关键字不能有重复值，每个关键字至多可映射一个值。HashMap(通过散列机制，用于快速访问)，TreeMap（保持 key 处于排序状态，访问速度不如 hashmap），LinkedHashMap(保持元素的插入顺序)

3）Set 接口：可包含重复的元素，LinkedHashSet TreeSet(用红黑树来存储元素) HashSet

4）List 接口:可通过索引对元素进行精准的插入和查找，实现类有 ArrayList LinkedList

5）Queue 接口：继承自 Collection 接口，LinkedList 实现了 Queue 接口，提供了支持队列的行为。

6）Iterator 接口：为了迭代集合

7）Comparable 接口：用于比较

6、List,Set,Map 的区别
Set 是一个无序的集合，不能包含重复的元素；

list 是一个有序的集合可以包含重复的元素，提供了按索引访问的方式；

map 包含了 key-value 对，map 中 key 必须唯一，value 可以重复。

7、HashMap 的实现原理
1）数据结构

jdk1.7 及以前，HashMap 由数组+链表组成，数组 Entry 是 HashMap 的主体，Entry 是 HashMap 中的一个静态内部类，每一个 Entry 包含一个 key-value 键值对，链表是为解决哈希冲突而存在。

从 jdk1.8 起，HashMap 是由数组+链表/红黑树组成，当某个 bucket 位置的链表长度达到阀值 8 时，这个链表就转变成红黑树。

2）HashMap 是线程不安全的，存储比较快，能接受 null 值，HashMap 通过 put(key, value)来储存元素，通过 get(key)来得到 value 值，通过 hash 算法来计算 hashcode 值，用 hashcode 标识 Entry 在 bucket 中存储的位置。

3）HashMap 中为什么要使用加载因子，为什么要进行扩容

加载因子是指当 HashMap 中存储的元素/最大空间值的阀值，如果超过这个值，就会进行扩容。加载因子是为了让空间得到充分利用，如果加载因子太大，虽对空间利用更充分，但查找效率会降低；如果加载因子太小，表中的数据过于稀疏，很多空间还没用就开始扩容，就会对空间造成浪费。

至于为什么要扩容，如果不扩容，HashMap 中数组处的链表会越来越长，这样查找效率就

会大大降低。

7.1 HashMap 如何 put 数据（从 HashMap 源码角度讲解）？

当我们使用 put(key, value)存储对象到 HashMap 中时，具体实现步骤如下：

1）先判断 table 数组是否为空，为空以默认大小构建 table，table 默认空间大小为 16

2）计算 key 的 hash 值，并计算 hash&(n-1)值得到在数组中的位置 index，如果该位置没值即 table[index]为空，则直接将该键值对存放在 table[index]处。

3）如果 table[index]处不为空，说明发生了 hash 冲突，判断 table[index]处结点是否是 TreeNode(红黑树结点)类型数据，如果是则执行 putTreeVal 方法，按红黑树规则将键值对存入；

4）如果 table[index]是链表形式，遍历该链表上的数据，将该键值对放在 table[index]处，并将其指向原 index 处的链表。判断链表上的结点数是否大于链表最大结点限制(默认为 8)，如果超过了需执行 treeifyBin()操作，则要将该链表转换成红黑树结构。

5）判断 HashMap 中数据个数是否超过了（最大容量*装载因子），如果超过了，还需要对其进行扩容操作。

7.2 HashMap 如何 get 数据?

get(key)方法获取 key 的 hash 值，计算 hash&(n-1)得到在链表数组中的位置 first=table[hash&(n-1)]，先判断 first（即数组中的那个）的 key 是否与参数 key 相等，不等的话，判断结点是否是 TreeNode 类型，是则调用 getTreeNode(hash, key)从二叉树中查找结点, 不是 TreeNode 类型说明还是链表型, 就遍历链表找到相同的 key 值返回对应的 value 值即可。

7.3 当两个对象的 hashcode 相同，即发生碰撞时，HashMap 如何处理

当两个对象的 hashcode 相同，它们的 bucket 位置相同，hashMap 会用链表或是红黑树来存储对象。Entry 类里有一个 next 属性，作用是指向下一个 Entry。第一个键值对 A 进来，通过计算其 key 的 hash 得到 index，记做 Entry[index]=A。一会又进来一个键值对 B，通过计算其 key 的 hash 也是 index, HashMap 会将 B.next=A, Entry[index]=B.如果又进来 C，其 key 的 hash 也是 index,会将 C.next=B, Entry[index]=C.这样 bucket 为 index 的地方存放了 A\B\C 三个键值对，它们能过 next 属性链在一起。数组中存储的是最后插入的元素，其他元素都在后面的链表里。

7.4 如果两个键的 hashcode 相同，如何获取值对象?

当调用 get 方法时，hashmap 会使用键对象的 hashcode 找到 bucket 位置，找到 bucket 位置后，会调用 key.equals()方法去找到链表中正确的节点，最终找到值对象。

7.5 hashMap 如何扩容

HashMap 默认负载因为是 0.75，当一个 map 填满了 75%的 bucket 时，和其他集合类一样，将会创建原来 HashMap 大小两倍的 bucket 数组，来重新调整 HashMap 的大小，并

将原来的对象放入新的 bucket 数组中。

在 jdk1.7 及以前，多线程扩容可能出现死循环。因为在调整大小过程中，存储在某个 bucket 位置中的链表元素次序会反过来，而多线程情况下可能某个线程翻转完链表，另外一个线程又开始翻转，条件竞争发生了，那么就死循环了。

而在 jdk1.8 中，会将原来链表结构保存至节点 e 中，将原来数组中的位置设为 null，然后依次遍历 e，根据 hash&n 是否为 0 分成两条支链，保存在新数组中。如果多线程情况可能会取到 null 值造成数据丢失。

8、ConcurrentHashMap 的实现原理

1) jdk1.7 及以前：一个 ConcurrentHashMap 由一个 segment 数组和多个 HashEntry 组成，每一个 segment 都包含一个 HashEntry 数组，Segment 继承 ReentrantLock 用来充当锁角色，每一个 segment 包含了对自己的 HashEntry 的操作，如 get\put\replace 操作，这些操作发生时，对自己的 HashEntry 进行锁定。由于每一个 segment 写操作只锁定自己的 HashEntry，可以存在多个线程同时写的情况。

jdk1.8 以后：ConcurrentHashMap 取消了 segments 字段，采用 transient volatile HashEntry<K, V> table 保存数据，采用 table 数组元素作为锁，实现对每一个数组数据进行加锁，进一小减少并发冲突概率。ConcurrentHashMap 是用 Node 数组+链表+红黑树数据结构来实现的，并发制定用 synchronized 和 CAS 操作。

2）Segment 实现了 ReentrantLock 重入锁，当执行 put 操作，会进行第一次 key 的 hash 来定位 Segment 的位置，若该 Segment 还没有初始化，会通过 CAS 操作进行赋值，再进行第二次 hash 操作，找到相应的 HashEntry 位置。

9、ArrayMap 和 HashMap 的对比

1)存储方式不一样，HashMap 内部有一个 Node<K,V>[]对象，每个键值对都会存储到这个对象里，当用 put 方法添加键值对时，会 new 一个 Node 对象，tab[i] = newNode(hash, key, value, next);

ArrayMap 存储则是由两个数组来维护，int[] mHashes; Object[] mArray; mHashes 数组中保存的是每一项的 HashCode 值，mArray 存的是键值对，每两个元素代表一个键值对，前面保存 key，后面保存 value。mHashes[index]=hash; mArray[index<<1]=key; mArray[(index<<1)+1]=value;

ArrayMap 相对于 HashMap，无需为每个键值对创建 Node 对象，且在数组中连续存放，更省空间。

2）添加数据时扩容处理不一样，进行了 new 操作，重新创建对象，开销很大；而 ArrayMap 用的是 copy 数据，所有效率相对高些；

3）ArrayMap 提供了数组收缩功能，在 clear 或 remove 后，会重新收缩数组，释放空间；

4）ArrayMap 采用二分法查找，mHashes 中的 hash 值是按照从小到大的顺序连续存放的，通过二分查找来获取对应 hash 下标 index，去 mArray 中查找键值对。mHashes 中的 index*2 是 mArray 中的 key 下标，index*2+1 为 value 的下标，由于存在 hash 碰撞情况，二分查找到的下标可能是多个连续相同的 hash 值中的任意一个，此时需要用 equals 比对命中的 key 对象是否相等，不相等，应当从当前 index 先向后再向前遍历所有相同 hash 值。

5）sparseArray 比 ArrayMap 进一步优化空间，SparseArray 专门对基本类型做了优化，Key 只能是可排序的基本类型，如 int\long，对 value，除了泛型 Value，还对每种基本类型有单独实现，如 SparseBooleanArray\SparseLongArray 等。无需包装，直接使用基本类型值，无需 hash，直接使用基本类型值索引和判断相等，无碰撞，无需调用 hashCode 方法，无需 equals 比较。SparseArray 延迟删除。

10、HashTable 实现原理

Hashtable 中的无参构造方法 Hashtable()中调用了 this(11, 0.75f)，说明它默认容量是 11，加载因子是 0.75,在构造方法上会 new HashtableEntry<?, ?>[initialCapacity]；会新建一个容量是初始容量的 HashtableEntry 数组。HashtableEntry 数组中包含 hash\Key\Value\next 变量，链表形式，重写了 hashCode 和 equals 方法。Hashtable 所有 public 方法都在方法体上加上了 synchronized 锁操作，说明它是线程安全的。它还实现了 Serializable 接口中的 writeObject 和 readObject 方法，分别实现了逐行读取和写入的功能，并且加了 synchronized 锁操作。

（1） put(Key, Value)方法

1）先判断 value 是否为空，为空抛出空指针异常；

2）根据 key 的 hashCode()值，计算 table 表中的位置索引(hash&0x7FFFFFFF)%tab.length 值 index，如果该索引处有值，再判断该索引处链表中是否包含相同的 key，如果 key 值相同则替换旧值。

3）如果没有相同的 key 值，调用 addEntry 方法，在 addEntry 中判断 count 大小是否超过了最大容量限制，如果超过了需要重新 rehash()，容量变成原来容量*2+1，将原表中的值都重新计算 hash 值放入新表中。再构造一个 HashtableEntry 对象放入相应的 table 表头，如果原索引处有值，则将 table[index].next 指向原索引处的链表。

（2）get 方法

根所 key.hashCode()，计算它在 table 表中的位置，(hash&0x7FFFFFFF)%tab.length，遍历该索引处表的位置中是否有值，是否存在链表，再判断是 key 值和 hash 值是否相等，相等则返回对应的 value 值。

11、HashMap 和 HashTable 的区别

1）Hashtable 是个线程安全的类，在对外方法都添加了 synchronized 方法，序列化方法上也添加了 synchronized 同步锁方法，而 HashMap 非线程安全。这也导致 Hashtable 的读写等操作比 HashMap 慢。

2）Hashtable 不允许值和键为空，若为空会抛出空指针。而 HashMap 允许键和值为空；

3）Hashtable 根据 key 值的 hashCode 计算索引，(hash&0x7FFFFFFF)%tab.length，保证 hash 值始终为正数且不超过表的长度。而 HashMap 中计算索引值是通过 hash(key)&(tab.length-1)，是通过与操作，计算出在表中的位置会比 Hashtable 快。

4) Hashtable 容量能为任意大于等于 1 的正数，而 HashMap 的容量必须为 2^n，Hashtable 默认容量为 11，HashMap 初始容量为 16

5）Hashtable 每次扩容，新容量为旧容量的 2 倍+1，而 HashMap 为旧容量的 2 倍。

12、HashMap 与 HashSet 的区别

HashSet 底层实现是 HashMap,内部包含一个 HashMap<E, Ojbect> map 变量

```
private transient HashMap<E,Object> map;
```

一个 Object PRESENT 变量（当成插入 map 中的 value 值）

```
private static final Object PRESENT = new Object();
```

HashSet 中元素都存到 HashMap 键值对的 Key 上面。具体可以查看 HashSet 的 add 方法，直接调用了 HashMap 的 put 方法，将值作为 HashMap 的键，值用一个固定的 PRESENT 值。

```
public boolean add(E e) {
        return map.put(e, PRESENT)==null;
    }
```

HashSet 没有单独的 get 方法，用的是 HashMap 的。HashSet 实现了 Set 接口，不允许集合中出现重复元素，将对象存储进 HashSet 前，要先确保对象重写了 hashCode()和 equals 方法，以保证放入 set 对象是唯一的。

13、HashSet 与 HashMap 怎么判断集合元素重复?

HashMap 在放入 key-value 键值对是，先通过 key 计算其 hashCode()值，再与 tab.length-1 做与操作，确定下标 index 处是否有值，如果有值，再调用 key 对象的 equals 方法，对象不同则插入到表头，相同则覆盖；

HashSet 是将数据存放到 HashMap 的 key 中，HashMap 是 key-value 形式的数据结构，它的 key 是唯一的，HashSet 利用此原理保证放入的对象唯一性。

14、集合 Set 实现 Hash 怎么防止碰撞

HashSet 底层实现是 HashMap，HashMap 如果两个不同 Key 对象的 hashCode()值相等，会用链表存储，HashSet 也一样。

15、ArrayList 和 LinkedList 的区别，以及应用场景

ArrayList 底层是用数组实现的，随着元素添加，其大小是动态增大的；在内存中是连续存

放的；如果在集合末尾添加或删除元素，所用时间是一致的，如果在列表中间添加或删除元素，所用时间会大大增加。通过索引查找元素速度很快。适合场合：查询比较多的场景

LinkedList 底层是通过双向链表实现的，LinkedList 和 ArrayList 相比，增删速度快，但查询和修改值速度慢。在内存中不是连续内存。场景：增删操作比较多的场景。

-二叉树的深度优先遍历和广度优先遍历的具体实现
-堆的结构
-堆和树的区别
-堆和栈在内存中的区别是什么(解答提示：可以从数据结构方面以及实际实现方面两个方面去回答)?
-什么是深拷贝和浅拷贝
-手写链表逆序代码
-讲一下对树，B+树的理解
-讲一下对图的理解
-判断单链表成环与否?
-链表翻转（即：翻转一个单项链表）
-合并多个单有序链表（假设都是递增的）