# 集合

## Java中常用的集合有哪些？

**常用的集合主要在java.util包下面：**

主要接口有List,Set,Map,Queue,Deque,SortedMap,SortedSet

List:

线程不安全：ArrayList, LinkedList, AttributeList, RoleList

线程安全：CopyOnWriteArraylist ,Stack, Vector

Set:

线程不安全：EnumSet, HashSet, LinkedHashSet, TreeSet

线程安全：ConcurrentSkipListSet, CopyOnWriteArraySet

Map:

线程不安全：EnumMap, HashMap, IdentityHashMap, LinkedHashMap, Properties, TreeMap

线程安全：ConcurrentHashMap, ConcurrentSkipListMap, Hashtable

SortedMap继承于map:

ConcurrentSkipListMap, TreeMap

SortedSet继承于set:

ConcurrentSkipListSet, TreeSet

queue:

1.ArrayDeque, （数组双端队列）

2.PriorityQueue, （优先级队列）

3.ConcurrentLinkedQueue, （基于链表的并发队列）

4.DelayQueue, （延期阻塞队列）（阻塞队列实现了BlockingQueue接口）

5.ArrayBlockingQueue, （基于数组的并发阻塞队列）

6.LinkedBlockingQueue, （基于链表的FIFO阻塞队列）

7.LinkedBlockingDeque, （基于链表的FIFO双端阻塞队列）

8.PriorityBlockingQueue, （带优先级的无界阻塞队列）

9.SynchronousQueue （并发同步阻塞队列）

Deque:

ArrayDeque, LinkedBlockingDeque, LinkedList

## Java的数据结构相关的类实现原理？

Java常用的数据包括数组，链表，树。有些类是基于单一的数据结构实现的，比如ArrayList，Vector是基于数组实现的，LinkedList是基于链表实现的。有些类是基于复合数据结构实现的，比如HashMap是基于数组+链表或数组+树的数据结构实现的，LinkedHashMap更是在HashMap的基础上再加上一层链表数据结构来维护集合的顺序。

## 讲讲自己对HashMap的理解？

HashMap是以key-value键值对存储数据的集合，JDK1.7以前，HashMap底层使用的是数组+单向链表的数据结构（在数据结构中，一般称之为“链表散列“），其主要包含put，get方法，put方法用于存储数据，get用于获取数据。当使用put方法时，首先会根据key的hashcode与数组大小长度-1进行逻辑与运算，也就是通过求模计算出该数据在数组中存储的位置。遍历该位置下的链表，使用equals判断这个链表是否包含这个key，如果存储则替换该entry对象的值，并将旧的value返回。如果不存在则将新数据放在链表的首位。存储完成以后会判断数组的大小是否超过阀值，超过则对数组进行扩容，并遍历整个hashmap，将数据放入新的数组中。get方法则相对简单，只是通过hashcode找到数组对应位置，遍历数组，使用equals方法查找是否包含对应的值，有则返回，无则返回null。

JDK1.8之后HashMap数据结构做了改动，在链表数据达到8个时会转换为使用红黑树的数据结构存储，put，get方法也做了相应的改动。put方法存储数据时会先判断当前数据结构是否为红黑树，是则通过红黑树的方式存储数据，而且扩容的时候也有些改变，1.7是遍历整个数组，并将链表的数据一个一个重新计算hashcode到新的数组中，但1.8则是先生成新的两个链表，再放到新的数组中。

**hashmap扩容过程，1.7 1.8的区别**

1.7

Index的计算方式：return h & (length-1);

扩容方式：循环遍历所有的node节点，再一个个获取位置存入新的数组，扩容时也是扩为原来的两倍。

1.8

Index的计算方式 : h & (n-1)

扩容时方式：遍历数组，将每个桶先分成两个桶再插入到新的数组中

1.8新增：h = (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);将高位参与运算是为了在table长度比较小时，减少碰撞。



如果还是产生了频繁的碰撞，会发生什么问题呢？作者注释说，他们使用树来处理频繁的碰撞

## HashMap是否线程安全？

HashMap非线程安全的，如何线程安全的使用 HashMap。这个无非就是以下三种方式：Hashtable ConcurrentHashMap Synchronized Map

Synchronized Map：

SynchronizedMap：synchronizedMap() 方法后会返回一个 SynchronizedMap 类的对象，而在 SynchronizedMap 类中使用了 synchronized 同步关键字来保证对 Map 的操作是线程安全的。

## HashMap和HashTable的区别

Hashtable是线程安全的，基本上每个方法都使用了synchronized来保证线程安全性。

1、hashMap去掉了HashTable?的contains方法，但是加上了containsValue()和containsKey()方法

2、Hashtable不允许 null 值(key 和 value 都不可以)，HashMap允许 null 值(key和value都可以)。

3、两者的遍历方式大同小异，Hashtable仅仅比HashMap多一个elements方法。

4、HashTable使用Enumeration，HashMap使用Iterator

## ConcurrentHashMap 是如何实现的？

**1.7、1.8 实现有何不同？为什么这么做？**

base1.7

1.7仍然是数组加链表，相比于HashMap，ConcurrentHashMap数组与entry很多核心数据都使用了volatile修饰.原理上来说：ConcurrentHashMap 采用了分段锁技术，其中 Segment 继承于 ReentrantLock。不会像 HashTable 那样不管是 put 还是 get 操作都需要做同步处理。因为volatile无法保证原子性， put 操作时仍然需要加锁处理。首先第一步的时候会尝试获取锁，如果获取失败肯定就有其他线程存在竞争，则利用 scanAndLockForPut() 自旋获取锁。如果重试的次数达到了 MAX\_SCAN\_RETRIES 则改为阻塞锁获取，保证能获取成功。

由于 HashEntry 中的 value 属性是用 volatile 关键词修饰的，保证了内存可见性，所以每次获取时都是最新值。ConcurrentHashMap 的 get 方法是非常高效的，因为整个过程都不需要加锁。

base1.8

1.8抛弃了原有的 Segment 分段锁，而采用了 CAS + synchronized 来保证并发安全性。也将 1.7 中存放数据的 HashEntry 改为 Node，但作用都是相同的。其中的 val next 都用了 volatile 修饰，保证了可见性。

## ConcurrentHashMap能完全替代HashTable吗？

HashTable虽然性能上不如ConcurrentHashMap，但并不能完全被取代，两者的迭代器的一致性不同的，HashTable的迭代器是强一致性的，而ConcurrentHashMap是弱一致的。 ConcurrentHashMap的get，clear，iterator 都是弱一致性的。 Doug Lea 也将这个判断留给用户自己决定是否使用ConcurrentHashMap。

那么什么是强一致性和弱一致性呢？

get方法是弱一致的，是什么含义？可能你期望往ConcurrentHashMap底层数据结构中加入一个元素后，立马能对get可见，但ConcurrentHashMap并不能如你所愿。换句话说，put操作将一个元素加入到底层数据结构后，get可能在某段时间内还看不到这个元素，若不考虑内存模型，单从代码逻辑上来看，却是应该可以看得到的。

下面将结合代码和java内存模型相关内容来分析下put/get方法。put方法我们只需关注Segment#put，get方法只需关注Segment#get，在继续之前，先要说明一下Segment里有两个volatile变量：count和table；HashEntry里有一个volatile变量：value。

总结:ConcurrentHashMap的弱一致性主要是为了提升效率，是一致性与效率之间的一种权衡。要成为强一致性，就得到处使用锁，甚至是全局锁，这就与Hashtable和同步的HashMap一样了。

## HashMap和WeakHashMap的区别

HashMap的key是对实际对象的强引用,而WeakHashMap的key是对实际对象的弱引用。弱引用（WeakReference）的特性是：当gc线程发现某个对象只有弱引用指向它，那么就会将其销毁并回收内存。WeakReference也会被加入到引用队列queue中。

理解了相关概念之后，对WeakHashMap的实际应用感到很好奇。然后发现tomcat的源码里，实现缓存时会用到WeakHashMap。

## ArrayList与Vector的区别

ArrayList 是线程不安全的， Vector 是线程安全的，这两个类底层都是由数组实现的。