## **说说常见的集合有哪些吧？**

《马士兵多线程与高并发》集合的那个图。二话不说，背过

# ===========HashMap=========

## **HashMap与HashTable的区别？**

HashMap允许K/V都为null；Hashtable 的K/V都不允许为null；

Hashtable是jdk1.0的，设计的有问题，方法都是synchronized的，后来jdk1.2加入了HashMap。

## **HashMap是怎么解决哈希冲突的？**

## **HashMap在JDK1.7和JDK1.8中有哪些不同？**

|  |  |  |
| --- | --- | --- |
| **不同** | **JDK 1.7** | **JDK 1.8** |
| 存储结构 | 数组 + 链表 | 数组 + 链表 + 红黑树 |
| 初始化方式 | 单独函数：inflateTable() | 直接集成到了扩容函数resize()中 |
| hash值计算方式 | 扰动处理 = 9次扰动 = 4次位运算 + 5次异或运算 | 扰动处理 = 2次扰动 = 1次位运算 + 1次异或运算 |
| 存放数据的规则 | 无冲突时，存放数组；冲突时，存放链表 | 无冲突时，存放数组；冲突 & 链表长度 < 8：存放单链表；冲突 & 链表长度 > 8：树化并存放红黑树 |
| 插入数据方式 | 头插法（先讲原位置的数据移到后1位，再插入数据到该位置） | 尾插法（直接插入到链表尾部/红黑树） |
| 扩容后存储位置的计算方式 | 全部按照原来方法进行计算（即hashCode ->> 扰动函数 ->> (h&length-1)） | 按照扩容后的规律计算（即扩容后的位置=原位置 or 原位置 + 旧容量） |

ArrayList 线程不安全，add数据时会参数java.util.concurrentModificationException（并发修改异常），可使用vector、集合工具类Collections.synchronizedList(new ArrayList) 、CopyOnWriteArrayList

HashSet 线程不安全，add数据时会参数java.util.concurrentModificationException（并发修改异常），集合工具类Collections.synchronizedSet(new HashSet) 、CopyOnWriteArraySet(public CopyOnWriteArraySet() {al = new CopyOnWriteArrayList<E>();})

HashMap 线程不安全，put数据时会参数java.util.concurrentModificationException（并发修改异常），ConcurrentHashMap(分段锁)集合工具类Collections.synchronizedMap(new HashMap)

ArrayList在jdk7中new时直接创建长度为10的数组，jdk8中在new时初始化长度为0，只是在第一次add时会创建长度为10的数组

HashSet:底层数组+链表，先用hashcode确定下标，再用eques确定是否一致

hashmap详细介绍

<https://blog.csdn.net/Yoga0301/article/details/84452104?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase>

HashMap可存储null的key和value，Hashtable 不能存储null的key，linkedHashMap在HashMap底层的基础上增加了一个指向前后元素的指针，便于遍历；treeMap 自然排序或key排序，底层使用红黑树

HashMap 在jdk7之前使用数组+链表，在jdk8以后使用数组+链表+红黑树

LinkedHashMap底层使用的是LinkedHashMap.Entry 继承了HashMap.Node，在原有数据结构基础上新增了befor和after节点，使LinkedHashMap有序

默认填充因子0.75，提前扩容，能尽量减少链表长度

CurrencyHashMap使用分段锁达到线程安全并提高效率

map数据结构：key无序不可重复，value无序可重复；一个key-value键值对组成一个entry对象，也是无序不可重复

底层构成：1、jdk7 Map map = new HashMap（）；创建一个初始大小为16的entry数组，当使用map.put(key，value) 存储数据时，首先调用key所在的类中的hashcode方法，计算key的hash值，

经过某种算法，得到entry数组中的下标位置，如果该下标已存在元素，再调用key所在类中的equals方法，返回false在链表中添加，返回true更新，如果不存在元素直接添加，在不断

添加过程中，超过16大小，需要扩容，默认扩容2倍，并将原有的数据复制过来

2、jdk8 Map map = new HashMap（）创建一个空的node{}数组，首次调用put方法，会初始化node数组大小为16，当数组某一索引上的元素以链表的形式存在的数量>8且数组大小>64(小于64扩容)时，此索引上的

所有数据采用红黑树方式存储

Arrays.asList();方法实现原理

new新的数组，把旧数组的元素放到新数组中，再把新数组循环放到new的ArrayList中，完毕

ConcurrentHashMap中的get没有加锁，怎么保证线程安全的？

答：因为Node中的val使用volatile修饰，保证了有序性和可见性；并不是因为数组用volatile修饰了，volatile修饰数组，仅仅保证引用的可见性，并不保证内容可见性

concurrenthashmap为什么放弃分段锁：效率问题，分段锁耗费性能

arraylist扩容

8、ConcurrentHashMap中size方法，1.7和1.8实现的区别

单链表倒叙：

package com.zjf;

public class SingleLinkedNode {

private Object data;

private SingleLinkedNode next;

public SingleLinkedNode(Object data, SingleLinkedNode next) {

this.data = data;

this.next = next;

}

public static void main(String[] args) {

SingleLinkedNode n5 = new SingleLinkedNode(5,null);

SingleLinkedNode n4 = new SingleLinkedNode(4,n5);

SingleLinkedNode n3 = new SingleLinkedNode(3,n4);

SingleLinkedNode n2 = new SingleLinkedNode(2,n3);

SingleLinkedNode n1 = new SingleLinkedNode(1,n2);

System.out.println(sortNode(n1));

}

@Override

public String toString() {

String s = this.data.toString();

SingleLinkedNode next = this.next;

while (next!=null){

s+=","+next.data;

next = next.next;

}

return s;

}

private static SingleLinkedNode sortNode(SingleLinkedNode n) {

SingleLinkedNode node = new SingleLinkedNode(n.data,null);

while (n.next!=null){

SingleLinkedNode temp = n.next;

node = new SingleLinkedNode(temp.data,node);

n = temp;

}

return node;

}

}