## JDK8中的HashMap有哪些改动?

1. JDK7中的底层实现是数组+链表，JDK8中使用的是数组+链表+红黑树。
2. JDK7中扩容时有可能出现死锁，JDK8中通过算法优化不会出现死锁了。
3. JDK8中对算哈希值的哈希算法进行了简化以提高运算效率。

## JDK8中为什么要使用红黑树?

因为JDK7中是用数组+链表来作为底层的数据结构的，但是如果数据量较多，或者hash算法的散列性不够，可能导致链表上的数据太多，导致链表过长，考虑一种极端情况：如果hash算法很差，所有的元素都在同一个链表上。那么在查询数据的时候的时间复杂度和链表查询的时间复杂度差不多是一样的，我们知道链表的一个优点是插入快，但是查询慢，所以如果HashMap中出现了很长的链表结构会影响整个HashMap的查询效率，我们使用HashMap时插入和查询的效率是都要具备的，而红黑树的插入和查询效率处于完全平衡二叉树和链表之间，所以使用红黑树是比较合适的。

## HashMap扩容机制是怎么样的，JDK7与JDK8有什么不同吗?

首先，我们需要知道HashMap为什么需要扩容，道理很简单，HashMap底层是用数组+链表实现的，而数组是预先就已经分配好内存的，如果需要对数组进行扩容，需要重新开辟一个新的数组再将旧数组上的元素进行转移，如果不进行扩容，那么会导致HashMap的链表过长，查询效率降低，所以需要对数组进行扩容。

在JDK7中，HashMap扩容的条件是 `(size >= threshold) && (null != table[bucketIndex])` ， `size` 为HashMap当前的容量， `threshold` 初始化值为12， `table[bucketIndex]` 代表所put进来的key所对应的数组上的元素，所以在JDK7中扩容条件是**当Put操作传入的Key值所对应的数组位置上不为空时并且当前容量大于等于了扩容的阈值时才进行扩容**，JDK7中的扩容思路是：开辟一个新的数组，数组大小为原数组的两倍，然后再将数组上的链表与元素转移到新数组上，此过程可能会出现死锁，具体可参考：http://www.importnew.com/22011.html。

JDK8中的扩容条件比JDK7中要少，只有**当前容量大于等于了扩容的阈值时才进行扩容**，并且扩容的思路也发生了变化，思路比较复杂具体看课程视频内容的讲解。

## 为什么重写对象的Equals方法时，要重写HashCode方法，跟HashMap有关系吗?为什么?

跟HashMap有关系，或者说因为HashMap中用到了对象的hashcode方法所以会有关系，因为我们如果在设计两个对象相等的逻辑时，如果只重写Equals方法，那么一个类有两个对象A1，A2，他们的A1.equals(A2)为true，A1.hashcode和A2.hashcode不一样，当将A1和A2都作为HashMap的key时，HashMap会认为它两不相等，因为HashMap在判断key值相不相等时会判断key的hashcode是不是一样，hashcode一样相等，所以在这种场景下会出现我们认为这两个对象相等，但是hashmap不这么认为，所以会有问题。

## HashMap是线程安全的吗？遇到过ConcurrentModificationException异常吗？为什么会出现？如何解决？

HashMap不是线程安全的，**ConcurrentModificationException**这个异常通常会出现在多线程环境中，比如两个线程共享一个hashmap，一个线程在遍历，一个线程在删除，那么就有可能出现**ConcurrentModificationException**异常，假设如果不出现这个异常，那么则可能出现并发问题，可能遍历的线程发现hashmap存的元素少了，HashMap为了防止这种情况出现，所以直接会抛出**ConcurrentModificationException**异常，这是Fast-Fail机制，让错误尽快出现，不让用户继续“错下去”。具体的原因需要参考源码实现，想了解的同学也可以参看教学视频的讲解。

## 在使用HashMap的过程中我们应该注意些什么问题？

1. HashMap的扩容机制是很影响效率的，所以如果事先能确定有多少个元素需要存储，那么建议在初始化HashMap时对数组的容量也进行初始化，防止扩容。
2. HashMap中使用了对象的hashcode方法，而且很关键，所以再重写对象的equals时建议一定要重写hashcode方法。
3. 如果是用对象作为HashMap的key，那么请将对象设置为final，以防止对象被重新赋值，因为一旦重新赋值其实就代表了一个新对象作为了key，因为两个对象的hashcode可能不同。

## HashMap 和 Hashtable 的区别

1. HashMap 是非线程安全的，HashTable 是线程安全的；HashTable 内部的方法基本都经过 synchronized 修饰
2. 因为线程安全的问题，HashMap 要比 HashTable 效率高一点。另外，HashTable 基本被淘汰，不要在代码中使用它
3. HashMap 中，null 可以作为键，这样的键只有一个，可以有一个或多个键所对应的值为 null。但是在 HashTable 中 put 进的键值只要有一个 null，直接抛出 NullPointerException。
4. JDK1.8 以后的 HashMap 在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为8）时，将链表转化为红黑树，以减少搜索时间。Hashtable 没有这样的机制。



## 参考资料


1. 死锁场景：http://www.importnew.com/22011.html