HashMap的工作原理是近年来常见的Java面试题。几乎每个Java程序员都知道HashMap，都知道哪里要用HashMap，知道[Hashtable和HashMap之间的区别](http://www.importnew.com/7010.html)，那么为何这道面试题如此特殊呢？是因为这道题考察的深度很深。这题经常出现在高级或中高级面试中。投资银行更喜欢问这个问题，甚至会要求你实现HashMap来考察你的编程能力。ConcurrentHashMap和其它同步集合的引入让这道题变得更加复杂。让我们开始探索的旅程吧！

### 先来些简单的问题

### ****“你用过HashMap吗？” “什么是HashMap？你为什么用到它？”****

几乎每个人都会回答“是的”，然后回答HashMap的一些特性，譬如HashMap可以接受null键值和值，而Hashtable则不能；HashMap是非synchronized;HashMap很快；以及HashMap储存的是键值对等等。这显示出你已经用过HashMap，而且对它相当的熟悉。但是面试官来个急转直下，从此刻开始问出一些刁钻的问题，关于HashMap的更多基础的细节。面试官可能会问出下面的问题：

### ****“你知道HashMap的工作原理吗？” “你知道HashMap的get()方法的工作原理吗？”****

你也许会回答“我没有详查标准的Java API，你可以看看Java源代码或者Open JDK。”“我可以用Google找到答案。”

但一些面试者可能可以给出答案，“HashMap是****基于hashing的原理****，我们****使用put(key, value)存储对象到HashMap中，使用get(key)从HashMap中获取对象****。当我们给put()方法传递键和值时，我们先对键调用hashCode()方法，返回的****hashCode用于找到bucket位置来储存Entry对象****。”这里关键点在于指出，****HashMap是在bucket中储存键对象和值对象，作为Map.Entry****。这一点有助于理解获取对象的逻辑。如果你没有意识到这一点，或者错误的认为仅仅只在bucket中存储值的话，你将不会回答如何从HashMap中获取对象的逻辑。这个答案相当的正确，也显示出面试者确实知道hashing以及HashMap的工作原理。但是这仅仅是故事的开始，当面试官加入一些Java程序员每天要碰到的实际场景的时候，错误的答案频现。下个问题可能是关于HashMap中的碰撞探测(collision detection)以及碰撞的解决方法：

### ****“当两个对象的hashcode相同会发生什么？”****

从这里开始，真正的困惑开始了，一些面试者会回答因为hashcode相同，所以两个对象是相等的，HashMap将会抛出异常，或者不会存储它们。然后面试官可能会提醒他们有equals()和hashCode()两个方法，并告诉他们两个对象就算hashcode相同，但是它们可能并不相等。一些面试者可能就此放弃，而另外一些还能继续挺进，他们回答“因为hashcode相同，所以它们的bucket位置相同，‘碰撞’会发生。因为****HashMap使用链表存储对象****，这个Entry(包含有键值对的Map.Entry对象)会存储在链表中。”这个答案非常的合理，虽然有很多种处理碰撞的方法，这种方法是最简单的，也正是HashMap的处理方法。但故事还没有完结，面试官会继续问：

### ****“如果两个键的hashcode相同，你如何获取值对象？”****

面试者会回答：当我们调用get()方法，HashMap会使用键对象的hashcode找到bucket位置，然后获取值对象。面试官提醒他如果有两个值对象储存在同一个bucket，他给出答案:将会遍历链表直到找到值对象。面试官会问因为你并没有值对象去比较，你是如何确定确定找到值对象的？除非面试者知道HashMap在链表中存储的是键值对，否则他们不可能回答出这一题。

其中一些记得这个重要知识点的面试者会说，找到bucket位置之后，会****调用keys.equals()方法去找到链表中正确的节点****，最终找到要找的值对象。完美的答案！

许多情况下，面试者会在这个环节中出错，因为他们混淆了hashCode()和equals()方法。因为在此之前hashCode()屡屡出现，而equals()方法仅仅在获取值对象的时候才出现。一些优秀的开发者会指出使用不可变的、声明作final的对象，并且采用合适的equals()和hashCode()方法的话，将会减少碰撞的发生，提高效率。不可变性使得能够缓存不同键的hashcode，这将提高整个获取对象的速度，使用String，Interger这样的wrapper类作为键是非常好的选择。

如果你认为到这里已经完结了，那么听到下面这个问题的时候，你会大吃一惊。

### ****“如果HashMap的大小超过了负载因子(load factor)定义的容量，怎么办？”****

除非你真正知道HashMap的工作原理，否则你将回答不出这道题。默认的负载因子大小为0.75，也就是说，当一个map填满了75%的bucket时候，和其它集合类(如ArrayList等)一样，将会创建****原来HashMap大小的两倍的bucket数组****，来重新调整map的大小，并****将原来的对象放入新的bucket数组中****。这个过程叫作rehashing，因为它调用hash方法找到新的bucket位置。

如果你能够回答这道问题，下面的问题来了：

### ****“你了解重新调整HashMap大小存在什么问题吗？”****

你可能回答不上来，这时面试官会提醒你当多线程的情况下，可能产生条件竞争(race condition)。

当重新调整HashMap大小的时候，确实存在条件竞争，因为如果两个线程都发现HashMap需要重新调整大小了，它们会同时试着调整大小。在调整大小的过程中，****存储在链表中的元素的次序会反过来****，因为移动到新的bucket位置的时候，HashMap并不会将元素放在链表的尾部，而是放在头部，这是为了避免尾部遍历(tail traversing)。如果条件竞争发生了，那么就死循环了。这个时候，你可以质问面试官，为什么这么奇怪，要在多线程的环境下使用HashMap呢？：）

### 热心的读者贡献了更多的关于HashMap的问题：

1. ****为什么String, Interger这样的wrapper类适合作为键？**** String, Interger这样的wrapper类作为HashMap的键是再适合不过了，而且String最为常用。因为String是不可变的，也是final的，而且已经重写了equals()和hashCode()方法了。其他的wrapper类也有这个特点。不可变性是必要的，因为为了要计算hashCode()，就要防止键值改变，****如果键值在放入时和获取时返回不同的hashcode的话，那么就不能从HashMap中找到你想要的对象****。不可变性还有其他的优点如线程安全。如果你可以仅仅通过将某个field声明成final就能保证hashCode是不变的，那么请这么做吧。因为获取对象的时候要用到equals()和hashCode()方法，那么键对象正确的重写这两个方法是非常重要的。如果两个不相等的对象返回不同的hashcode的话，那么碰撞的几率就会小些，这样就能提高HashMap的性能。
2. ****我们可以使用自定义的对象作为键吗？**** 这是前一个问题的延伸。当然你可能使用任何对象作为键，只要它遵守了equals()和hashCode()方法的定义规则，并且****当对象插入到Map中之后将不会再改变了****。如果这个自定义对象时不可变的，那么它已经满足了作为键的条件，因为当它创建之后就已经不能改变了。
3. ****我们可以使用CocurrentHashMap来代替Hashtable吗？****这是另外一个很热门的面试题，因为ConcurrentHashMap越来越多人用了。我们知道Hashtable是synchronized的，但是ConcurrentHashMap同步性能更好，因为它仅仅根据同步级别对map的一部分进行上锁。ConcurrentHashMap当然可以代替HashTable，但是HashTable提供更强的线程安全性。看看[这篇博客](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html)查看Hashtable和ConcurrentHashMap的区别。

我个人很喜欢这个问题，因为这个问题的深度和广度，也不直接的涉及到不同的概念。让我们再来看看这些问题设计哪些知识点：

* ****hashing的概念****
* ****HashMap中解决碰撞的方法****
* ****equals()和hashCode()的应用，以及它们在HashMap中的重要性****
* ****不可变对象的好处****
* ****HashMap多线程的条件竞争****
* ****重新调整HashMap的大小****

### 总结

### HashMap的工作原理

HashMap基于hashing原理，我们通过put()和get()方法储存和获取对象。

当我们将键值对传递给put()方法时，它调用键对象的hashCode()方法来计算hashcode，然后找到bucket位置来储存值对象。

当获取对象时，通过键对象的equals()方法找到正确的键值对，然后返回值对象。

HashMap使用链表来解决碰撞问题，当发生碰撞了，对象将会储存在链表的下一个节点中。 HashMap在每个链表节点中储存键值对对象。

当两个不同的键对象的hashcode相同时会发生什么？ 它们会储存在同一个bucket位置的链表中。键对象的equals()方法用来找到键值对。

因为HashMap的好处非常多，我曾经在电子商务的应用中使用HashMap作为缓存。因为金融领域非常多的运用Java，也出于性能的考虑，我们会经常用到HashMap和ConcurrentHashMap。你可以查看更多的关于HashMap的文章:

本文转载自：<http://www.importnew.com/7099.html>