关于Hashcode的作用

Hashcode是利用地址加类型加特定转换方法 转换出来的一串字符串

在对象地址与类型都不同的情况下也有可能出现相同的hashcode值

　　总的来说，Java中的集合（Collection）有两类，一类是List，再有一类是Set。前者集合内的元素是有序的，元素可以重复；后者元素无序，但元素不可重复。   
       要想保证元素不重复，可两个元素是否重复应该依据什么来判断呢？这就是Object.equals方法了。但是，如果每增加一个元素就检查一 次，那么当元素很多时，后添加到集合中的元素比较的次数就非常多了。也就是说，如果集合中现在已经有1000个元素，那么第1001个元素加入集合时，它 就要调用1000次equals方法。这显然会大大降低效率。   
       于是，Java采用了哈希表的原理。哈希算法也称为散列算法，是将数据依特定算法直接指定到一个地址上。这样一来，当集合要添加新的元素时，先调用这个元素的hashCode方法，就一下子能定位到它应该放置的物理位置上。如果这个位置上没有元素，它就可以 直接存储在这个位置上，不用再进行任何比较了；如果这个位置上已经有元素了，就调用它的equals方法与新元素进行比较，相同的话就不存了；不相同，也就是发生了Hash key相同导致冲突的情况,那么就在这个Hash key的地方产生一个链表,将所有产生相同hashcode的对象放到这个单链表上去,串在一起。所以这里存在一个冲突解决的问题（很少出现）。这样一来实际调用equals方法的次数就大大降低了，几乎只需要一两次。   
       所以，Java对于eqauls方法和hashCode方法是这样规定的：   
           1、如果两个对象相等，那么它们的hashCode值一定要相等；   
           2、如果两个对象的hashCode相等，它们并不一定相等。   
上面说的对象相等指的是用eqauls方法比较。   
  
============================================================   
如何理解hashCode的作用:   
============================================================   
以java.lang.Object来理解,JVM每new一个Object,它都会将这个Object丢到一个Hash哈希表中去,这样的话,下次做Object的比较或者取这个对象的时候,它会根据对象的hashcode再从Hash表中取这个对象。这样做的目的是提高取对象的效率。具体过程是这样:   
1.new Object(),JVM根据这个对象的Hashcode值,放入到对应的Hash表对应的Key上,如果不同的对象确产生了相同的hash值,也就是发生了Hash key相同导致冲突的情况,那么就在这个Hash key的地方产生一个链表,将所有产生相同hashcode的对象放到这个单链表上去,串在一起。   
2.比较两个对象的时候,首先根据他们的hashcode去hash表中找他的对象,当两个对象的hashcode相同,那么就是说他们这两个对象放在Hash表中的同一个key上,那么他们一定在这个key上的链表上。那么此时就只能根据Object的equal方法来比较这个对象是否equal。当两个对象的hashcode不同的话，肯定他们不能equal.   
  
============================================================     
改写equals时总是要改写hashCode   
============================================================   
java.lang.Object中对hashCode的约定：   
  
   1. 在一个应用程序执行期间，如果一个对象的equals方法做比较所用到的信息没有被修改的话，则对该对象调用hashCode方法多次，它必须始终如一地返回同一个整数。   
   2. 如果两个对象根据equals(Object o)方法是相等的，则调用这两个对象中任一对象的hashCode方法必须产生相同的整数结果。   
   3. 如果两个对象根据equals(Object o)方法是不相等的，则调用这两个对象中任一个对象的hashCode方法，不要求产生不同的整数结果。但如果能不同，则可能提高散列表的性能。

里面很多话都是参考其他博客，具体可以参考这篇文章，讲得很详细,有点长：   
Hashcode的作用:http://blog.csdn.net/chinayuan/article/details/3345559