**[（转）hashCode()、equals()以及compareTo()方法的理解 （整理）](http://www.cnblogs.com/kanliwei/p/4280425.html)**

http://crd1991.iteye.com/blog/1473108

http://blog.csdn.net/liushuai\_ly/article/details/8197508

 判断两个对象是否相等（是同一个对象），首先调用hashCode()方法得到各自的hashcode，

1、如果hashcode不相等，则表明两个对象不相等。

2、如果hashcode相等，继续调用equals方法进行判断

  2.1：equals()返回true,则对象相等

  2.2：equals()返回fasle,两对象不相等

所以，要求程序员在重写hashCode方法时尽量做到：不一样的对象，hashCode不一样，这样在判断两个对象是否是同一对象时可以提高效率。

根据这两点，我们可以看一道常见的JAVA面试题：

题目：对于两个对象A、B，A.equals(B)==true,不一定有相同的hashCode()； 这句话是错误的。

当然你自己定义的对象可以实现equals相同而hashCode不同（并不会报错，不知道JAVA为什么不限死），但java的Object类中规定相同的对象一定要有相同的hashCode：原话如下：

Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes.

compareTo()方法和equals()方法的关系：

对于某些对象如集合（TreeSet)需要实现内部排序，所以要实现Comparable接口，从而要实现里面的唯一方法compareTo();实现Comparable接口的对象表明遵循自然排序。从Comparable的API中可以看出：

 重写compareTo()方法，不要求必须重写equals()方法，但是却强烈推荐重写equals()，以使两个方法的比较结果在逻辑上是一致。原话如下：

It is strongly recommended (though not required) that natural orderings be consistent with equals. This is so because sorted sets (and sorted maps) without explicit comparators behave "strangely" when they are used with elements (or keys) whose natural ordering is inconsistent with equals. In particular, such a sorted set (or sorted map) violates the general contract for set (or map), which is defined in terms of the equals method.

    所以在编写需要用TreeSet添加的对象时，该对象一定要实现Comparable接口，并且重写compareTo()方法，并推荐同时重写equals()方法

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。   
那么我们怎么判断两个元素是否重复呢？ 这就是Object.equals方法了。

通常想查找一个集合中是否包含某个对象，就是逐一取出每个元素与要查找的元素进行比较，当发现某个元素与要查找的对象进行equals方法比较的结果相等时，则停止继续查找并返回肯定的信息，否则返回否定的信息，如果一个集合中有很多元素譬如成千上万的元素，并且没有包含要查找的对象时，则意味着你的程序需要从该集合中取出成千上万个元素进行逐一比较才能得到结论，于是，有人就发明了一种哈希算法来提高从集合中查找元素的效率，这种方式将集合分成若干个存储区域，每个对象可以计算出一个哈希码，可以将哈希码分组，每组分别对应某个存储区域，根据一个对象的哈希码就可以确定该对象应该存储的那个区域.

hashCode方法可以这样理解：它返回的就是根据对象的内存地址换算出的一个值。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，不相同就散列其它的地址。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。  
  
谈到hashcode()和equals()就不能不说到hashset,hashmap,hashtable中的使用，具体是怎样呢，请看如下分析：

Hashset是继承Set接口，Set接口又实现Collection接口，这是层次关系。那么hashset是根据什么原理来存取对象的呢？   
在hashset中不允许出现重复对象，元素的位置也是不确定的。在hashset中又是怎样判定元素是否重复的呢？判断两个对象是否相等的规则是：   
.1)，判断两个对象的hashCode是否相等   
如果不相等，认为两个对象也不相等，完毕,如果相等，转入2  
.2)，判断两个对象用equals运算是否相等   
如果不相等，认为两个对象也不相等   
如果相等，认为两个对象相等（equals()是判断两个对象是否相等的关键）   
  
小结：  
(1)只有类的实例对象要被采用哈希算法进行存储和检索时，这个类才需要按要求覆盖hashCode方法，即使程序可能暂时不会用到当前类的hashCode方法，但是为它提供一个hashCode方法也不会有什么不好，没准以后什么时候又用到这个方法了，所以，通常要求hashCode方法和equals方法一并被同时覆盖。

(2)equals()相等的两个对象，hashcode()一定相等；equals（）不相等的两个对象，却并不能证明他们的hashcode()不相等。换句话说，equals()方法不相等的两个对象，hashcode()有可能相等。反过来：hashcode()不等，一定能推出equals()也不等；hashcode()相等，equals()可能相等，也可能不等。

提示：  
(1)通常来说，一个类的两个实例对象用equal方法比较的结果相等时，它们的哈希码也必须相等，但反之则不成立，即equals方法比较结果不相等的对象可以有相同的哈希码，或者说哈希码相同的两个对象的equal方法比较的结果可以不等。

(2)当一个对象被存储进hashset集合中以后，就不能修改这个对象中的那些参与计算哈希值的字段了，否则，对象修改后的哈希值与最初存储进hashset集合时的哈希值就不同了，这种情况下，即使在contains方法使用该对象的当前引用作为的参数去hashset集合中检索对象，也将返回找不到对象的结果，这也会导致无法从hashset集合中单独删除当前对象，从而造成内存泄露，所谓的内存泄露也就说有一个对象不再被使用，但它一直占有内存空间，没有被释放。