【强制】关于hashCode和equals的处理，遵循如下规则：

1. 只要重写equals，就必须重写hashCode()
2. 因为Set存储的是不重复的对象，依据hashCode和equals进行判断,所以Set存储的对象必须重写这两个方法。
3. 如果自定义对象为Map的键，那么必须重写hashCode和equals。

说明：String重写了hashCode和equals方法，所以我们可以非常愉快地使用String对象作为key来使用。

#### 介绍

equals()判断两个对象是否相等。

hashCode()获取哈希码，也称为散列码；它实际上是返回一个int整数。这个哈希码的作用是确定该对象在哈希表中的索引位置。

#### 关系

以“类的用途”来将“hashCode()和equals()的关系”分2中情况来说明。

1. **不会创建“类对应的散列表”**

这里所说的“不会创建类对应的散列表”是说：我们不会在HashSet，Hashtable，HashMap等等这些本质是散列表的数据结构中，用到该类。例如，不会创建该类的HashSet集合。

在这种情况下，该类的“hashCode()和equals()”没有半毛钱关系的！equals()用来比较该类的两个对象是否相等。而hashCode()根本没有任何作用。

下面通过示例查看类的两个对象相等以及不等时hashCode()的取值。

public static void main(String[] args) {

//新建2个相同内容的Person对象，  
 //再用两个equals比较它们是否相等。  
  
 Person p1 = new Person("eee", 100);  
 Person p2 = new Person("eee", 100);  
 Person p3 = new Person("aaa", 200);  
  
 System.*out*.printf("p1.equals(p2) : %s; p1(%d) p2(%d)\n", p1.equals(p2), p1.hashCode(), p2.hashCode());  
 System.*out*.printf("p1.equals(p3) : %s; p1(%d) p3(%d)\n", p1.equals(p3), p1.hashCode(), p3.hashCode());  
}

运行结果：

p1.equals(p2) : true; p1(1639705018) p2(1627674070)

p1.equals(p3) : false; p1(1639705018) p3(1300109446)

p1和p2相等的情况下，hashCode也不一定相等。

**2）会创建“类对应的散列表”**

这里所说的“会创建类对应的散列表”是说：我们会在HashSet，HashTable，HashMap，等等这些本质是散列表的数据结构中，用到该类。例如，会创建该类的HashSet集合。

在这种情况下，该类的“hashCode()和equals()”是有关系的：

* 如果两个对象相等，那么它们的hashCode()值一定相同。这里的相等是指，通过equals()比较两个对象时返回true。
* 如果两个对象hashCode()相等，它们并不一定相等。因为在散列表中，hashCode()相等，即两个键值对的哈希值相等。然而哈希值相等，并不一定能得出键值对相等。“两个不同的键值对，哈希值相等”，这就是哈希冲突。

此外，在这种情况下。若要判断两个对象是否相等，除了要覆盖equals之外，也要覆盖hashCode()函数。否则，equals()无效。

举例，创建Person类的HashSet集合，必须同时覆盖Person类的equals()和hashCode()方法。如果单单只是覆盖equals()方法。我们会发现，equals()方法没有达到我们想要的效果。

HashSet<Person> personSet = new HashSet<>();

personSet.add(p1);  
personSet.add(p2);  
personSet.add(p3);  
  
//比较p1和p2  
System.*out*.printf("p1.equals(p2) : %s; p1(%d) p2(%d)\n", p1.equals(p2), p1.hashCode(), p2.hashCode());  
//打印set  
System.*out*.printf("set:%s\n", personSet);

运行结果：

p1.equals(p2) : true; p1(1639705018) p2(1627674070)

set:[{name='eee', weight=100.0}, {name='eee', weight=100.0}, {name='aaa', weight=200.0}]

重写了Person的equals()。但是，HashSet中仍然有重复元素，p1和p2。为什么会出现这种情况呢？

这是因为虽然p1和p2的内容相等，但是它们的hashCode()不等；所以HashSet在添加p1和p2的时候，认为它们不相等。

同时覆盖equals()和hashCode()方法呢？

*/\*\**

*\* 重写hashCode方法  
 \*/*@Override  
public int hashCode() {  
 int nameHash = name.toUpperCase().hashCode();  
 //maneHash和weight按位异或运算  
 return nameHash ^ weight;  
}

运行结果：

p1.equals(p2) : true; p1(68545) p2(68545)

p1.equals(p3) : false; p1(68545) p3(64745)

set:[{name='eee', weight=100}, {name='aaa', weight=200}]

这下，equals()生效了，HashCode中没有重复元素。比较p1和p2，我们发现：它们的hashCode()相等，通过equals()比较它们也返回true。所以，p1和p2被视为不相等。

比较p1和p3，我们发现：虽然它们的hashCode()相等；但是，通过equals()比较它们返回false。所以，p1和p3被视为不相等。

#### 原则

1. 同一个对象（没有发生过修改）无论何时调用hashCode()得到的返回值必须一样。

如果一个key对象在put的时候调用hashCode()决定了存放的位置，而在get的时候调用hashCode()得到不一样的返回值，这个值映射到了一个和原来不一样的地方，那么肯定就找不到原来那个键值对了。

1. hashCode()的返回值相等的对象不一定相等，通过hashCode和equals必须能够唯一确定一个对象。不相等的对象的hashCode()的结果可以相等。hashCode()在注意关注碰撞问题的时候，也要关注速度问题，完美hash不识闲。
2. 一旦重写了equals()函数（重写equals的时候还要主要满足自反省、对称性、传递性、一致性），就必须重写hashCode()函数。而且hashCode()的生成哈希值的依据应该是equals()中用来比较是否相等的字段。

如果两个由equals()规定相等的对象生成的hashCode不等，对于hashMap来说，他们很可能分别映射到不同位置，没有调用equals()比较是否相等的机会，两个实际上相等的对象可能被插入不同位置，出现错误。其他一些基于哈希方法的集合类可能也会有这个问题。