Evan
Only let oneself become strong enough, good enough, can afford the life that you want to.

== 和 equals，equals 与 hashcode，HashSet 和 HashMap，HashMap 和 Hashtable

标签： equals 与 hashcode HashSet 和 HashMap HashMap 和 Hashtable 区别 和 equals

2017-06-07 23:35 53人阅读 评论(0) 收藏 举报

分类： Java (5)

一：== 和 equals

== 比较引用的地址

equals 比较引用的内容 (Object 类本身除外)

```
[java]
<span style="font-size:14px;">String obj1 = new String("xyz");
String obj2 = new String("xyz");
// If String obj2 = obj1, the output will be true
if(obj1 == obj2)
System.out.println("obj1==obj2 is TRUE");
else
System.out.println("obj1==obj2 is FALSE");
// It will print obj1==obj2 is False
// If String obj2 = obj1, the output will be true</span>
```

默认的, equals() 方法实际上和 “==” 在 object 类里是一样的. 但是这个方法在每一个子类里都

会被覆写用来比较引用的内容 (因为每个类都继承了 object 类并覆写了这个方法)

```
[java]
<span style="font-size:14px;">String obj1 = new String("xyz");
String obj2 = new String("xyz");
if(obj1.equals(obj2))
System.out.println("obj1==obj2 is TRUE");
else
System.out.println("obj1==obj2 is FALSE");
Resultat: obj1==obj2 is TRUE</span>
```

string1="aaa";

string2="aaa";

String string3=new String("aaa");

String string4=new String("aaa");

string1==string2 // true; .

string1.equals(string2);//true;

string3==string4;//false 因为用new创建了2个对象,所以是两个不同的内存地址

string3.equals(string4);//true 而String类的是不可改变的,所以会指向同一个内存地址,所以返回为true

equals()是object的方法,所以只是适合对象,不适合于基本类型,equals()默认是用"=="比较两个对象的内存地址,如果想要比较两个对象的[illegible]

equals()方法才可...而==可以比较两个基本类型,也可以是对象...

总而言之:在类对象中 equals()方法比较的是对象的值,==比较的是对象.即为对象的引用(即为内存地址)

二：equals 与 hashcode

如果需要比较对象的值，就需要equals方法了. 看一下JDK中equals方法的实现：

```
[java]
01. <span style="font-size:14px;">public boolean equals(Object obj) {
02. return (this == obj);
03. }</span>
```

也就是说，默认情况下比较的还是对象的地址. 所以如果把对象放入Set中等操作，就需要重写equals方法了

重写之后的 equals() 比较的就是对象的内容了

在Java中任何一个对象都具备equals(Object obj)和hashcode()这两个方法，因为他们是在Object类中定义的。

equals(Object obj)方法用来判断两个对象是否“相同”，如果“相同”则返回true，否则返回false。

hashcode()方法返回一个int数，在Object类中的默认实现是“将该对象的内部地址转换成一个整数返回”。

接下来有两个个关于这两个方法的重要规范(我只是抽取了最重要的两个,其实不止两个)：

规范1：若重写equals(Object obj)方法，有必要重写hashcode()方法，确保通过equals(Object obj)方法判断结果为true的两个对象具备相等的hashcode()返回值。说得简单点就是：“如果两个对象相同，那么他们的hashcode应该 相等”。不过请注意：这个只是规范，如果你非要写一个类让equals(Object obj)返回true而hashcode()返回两个不相等的值，编译和运行都是不会报错的。不过这样违反了Java规范，程序也就埋下了BUG。

规范2：如果equals(Object obj)返回false，即两个对象“不相同”，并不要求对这两个对象调用hashcode()方法得到两个不相同的数。说的简单点就是：“如果两个对象不相同，他们的hashcode可能相同”。

根据这两个规范，可以得到如下推论：

1、如果两个对象equals，Java运行时环境会认为他们的hashcode一定相等。

2、如果两个对象不equals，他们的hashcode有可能相等。

3、如果两个对象hashcode相等，他们不一定equals。

4、如果两个对象hashcode不相等，他们一定不equals。

三：HashSet 和 HashMap

HashSet和HashMap一直都是JDK中最常用的两个类，HashSet要求不能存储相同的对象，HashMap要求不能存储相同的键。

那么**Java**运行时环境是如何判断HashSet中相同对象、HashMap中相同键的呢？当存储了“相同的东西”之后Java运行时环境又将如何来维护呢？

根据上面的equals 与 hashcode分析我们就可以推断Java运行时环境是怎样判断HashSet和HastMap中的两个对象相同或不同了。我的推断是：先判断hashcode是否相等，再判断是否equals。

什么是HashSet？

HashSet实现了Set接口，它不允许集合中有重复的值，当我们提到HashSet时，第一件事情就是在将对象存储在HashSet之前，要先确保对象重写equals()和hashCode()方法，这样才能比较对象的值是否相等，以确保set中没有储存相等的对象。如果我们没有重写这两个方法，将会使用这个方法的默认实现。

public boolean add(Object o)方法用来在Set中添加元素，当元素值重复时则会立即返回false，如果成功添加的话会返回true。

什么是HashMap？

HashMap实现了Map接口，Map接口对键值对进行映射。Map中不允许重复的键。Map接口有两个基本的实现，HashMap和TreeMap。TreeMap保存了对象的排列次序，而HashMap则不能。HashMap允许键和值为null。HashMap是非synchronized的，但collection框架提供方法能保证HashMap synchronized，这样多个线程同时访问HashMap时，能保证只有一个线程更改Map。

public Object put(Object Key,Object value)方法用来将元素添加到map中。

HashSet和HashMap的区别

HashMap	*HashSet*
HashMap实现了Map接口	HashSet实现了Set接口
HashMap储存键值对	HashSet仅仅存储对象（且无重复对象）
使用put()方法将元素放入map中	使用add()方法将元素放入set中
HashMap中使用键对象来计算hashcode值	HashSet使用成员对象来计算hashcode值，对于两个对象来说hashcode可能相同，所以equals()方法用来判断对象的相等性，如果两个对象不同的话，那么返回false
HashMap比较快，因为是使用唯一的键来获取对象	HashSet较HashMap来说比较慢

四：HashTable和HashMap

HashMap和Hashtable都实现了Map接口，但决定用哪一个之前先要弄清楚它们之间的分别。主要的区别有：**线程安全性**，**同步**(synchronization)，以及**速度**。

1. HashMap几乎可以等价于Hashtable，除了HashMap是非synchronized的，并可以接受null(`HashMap allows one null key and any number of null values.`，而Hashtable则不行)。这就是说，HashMap中如果在表中没有发现搜索键，或者如果发现了搜索键，但它是一个空的值，那么get()将返回null。如果有必要，用containKey()方法来区别这两种情况。
2. HashMap是非synchronized，而Hashtable是synchronized，这意味着Hashtable是线程安全的，多个线程可以共享一个Hashtable；而如果没有正确的同步的话，多个线程是不能共享HashMa的。 即是说，在多线程应用程序中，不用专门的操作就安全地可以使用Hashtable了；而对于HashMap，则需要额外的同步机制。但HashMap的同步问题可通过Collections的一个静态方法得到解决：
   Map Collections.synchronizedMap(Map m)
   这个方法返回一个同步的Map，这个Map封装了底层的HashMap的所有方法，使得底层的HashMap即使是在多线程的环境中也是安全的。
   而而且Java 5提供了ConcurrentHashMap，它是HashTable的替代，比HashTable的扩展性更好。
1. 另一个区别是HashMap的迭代器(Iterator)是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。所以当有其它线程改变了HashMap的结构（增加或者移除元素），将会抛出ConcurrentModificationException，但迭代器本身的remove()方法移除元素则不会抛出ConcurrentModificationException异常。但这并不是一个一定发生的行为，要看JVM。这条同样也是Enumeration和Iterator的区别。
2. 由于Hashtable是线程安全的也是synchronized，所以在单线程环境下它比HashMap要慢。如果你不需要同步，只需要单一线程，那么使用HashMap性能要好过Hashtable。
3. HashMap不能保证随着时间的推移Map中的元素次序是不变的。
4. 哈希值的使用不同，HashTable直接使用对象的hashCode，代码是这样的：
   ```
   int hash = key.hashCode();
   int index = (hash & 0x7FFFFFFF) % tab.length;
   ```
   而HashMap重新计算hash值，而且用与代替求模：
   ```
   int hash = hash(k);
   int i = indexFor(hash, table.length);
   ```

要注意的一些重要术语：

1) sychronized意味着在一次仅有一个线程能够更改Hashtable。就是说任何线程要更新Hashtable时要首先获得同步锁，其它线程要等到同步锁被释放之后才能再次获得同步锁更新Hashtable。

2) Fail-safe和iterator迭代器相关。如果某个集合对象创建了Iterator或者ListIterator，然后其它的线程试图“结构上”更改集合对象，将会抛出ConcurrentModificationException异常。但其它线程可以通过set()方法更改集合对象是允许的，因为这并没有从“结构上”更改集合。但是假如已经从结构上进行了更改，再调用set()方法，将会抛出IllegalArgumentException异常。

3) 结构上的更改指的是删除或者插入一个元素，这样会影响到map的结构。