java中equals和==的区别

值类型是存储在内存中的堆栈（以后简称栈），而引用类型的变量在栈中仅仅是存储引用类型变量的地址，而其本身则存储在堆中。  
==操作比较的是两个变量的值是否相等，对于引用型变量表示的是两个变量在堆中存储的地址是否相同，即栈中的内容是否相同。  
equals操作表示的两个变量是否是对同一个对象的引用，即堆中的内容是否相同。  
  
==比较的是2个对象的地址，而equals比较的是2个对象的内容。  
显然，当equals为true时，==不一定为true；

一、String中的equals和==  
1、  
public class TestString {  
     public static void main(String[] args) {  
         String s1 = "Monday";  
         String s2 = "Monday";  
     }  
}  
上面这段程序中，到底有几个对象呢？  
来检测一下吧，稍微改动一下程序  
public class TestString {  
     public static void main(String[] args) {  
         String s1 = "Monday";  
         String s2 = "Monday";  
         if (s1 == s2)  
             System.out.println("s1 == s2");  
         else  
             System.out.println("s1 != s2");  
     }  
}  
  
编译并运行程序，输出：s1 == s2  
说明：s1 与 s2 引用同一个 String 对象 -- "Monday"!  
  
2.  
再稍微改动一下程序，会有更奇怪的发现：  
public class TestString {  
     public static void main(String[] args) {  
         String s1 = "Monday";  
         String s2 = new String("Monday");  
         if (s1 == s2)  
             System.out.println("s1 == s2");  
         else  
             System.out.println("s1 != s2");  
         if (s1.equals(s2))  
             System.out.println("s1 equals s2");  
         else  
             System.out.println("s1 not equals s2");  
     }  
}  
我们将 s2 用 new 操作符创建  
程序输出：  
s1 != s2  
s1 equals s2  
说明：s1 s2分别引用了两个"Monday"String对象  
  
3. 字符串缓冲池  
  
原来，程序在运行的时候会创建一个字符串缓冲池  
当使用 s2 = "Monday" 这样的表达是创建字符串的时候，程序首先会  
在这个String缓冲池中寻找相同值的对象，在第一个程序中，s1先被  
放到了池中，所以在s2被创建的时候，程序找到了具有相同值的 s1  
将 s2 引用 s1 所引用的对象"Monday"  
第二段程序中，使用了 new 操作符，他明白的告诉程序：  
"我要一个新的！不要旧的！"于是一个新的"Monday"Sting对象被创  
建在内存中。他们的值相同，但是位置不同，一个在池中游泳  
一个在岸边休息。哎呀，真是资源浪费，明明是一样的非要分开做什么呢？  
4.   
再次更改程序：  
public class TestString {  
     public static void main(String[] args) {  
         String s1 = "Monday";  
         String s2 = new String("Monday");  
         s2 = s2.intern();  
         if (s1 == s2)  
             System.out.println("s1 == s2");  
         else  
             System.out.println("s1 != s2");  
         if (s1.equals(s2))  
             System.out.println("s1 equals s2");  
         else  
             System.out.println("s1 not equals s2");  
     }  
}  
这次加入：s2 = s2.intern();  
程序输出：  
s1 == s2  
s1 equals s2  
  
原来，（java.lang.String的intern()方法   
"abc".intern()方法的返回值还是字符串"abc"，表面上看起来好像这个方法没什么用处。但实际上，它做了个小动作：   
检查字符串池里是否存在"abc"这么一个字符串，如果存在，就返回池里的字符串；如果不存在，该方法会把"abc"添加到字符串池中，然后再返回它的引用。  
）  
更好的办法：  
把所有的String都intern()到缓冲池去吧  
最好在用到new的时候就进行这个操作  
String s2 = new String("Monday").intern();  
然后就可以用==比较两个字符串的值了  
  
  
二、简单数据类型和封装类中的equals和==  
Java为每一个简单数据类型提供了一个封装类，每个基本数据类型可以封装成对象类型。   
除int（Integer）和char（Character），其余类型首字母大写即成封装类类型名。double (Double), float(Float),long(Long), short(Short),byte(Byte),boolean(Boolean).  
    
以int和Integer为例说明   
Java中int和Integer区别如下：  
1.int是基本的数据类型,默认值可以为0；  
2.Integer是int的封装类,默认值为null；  
3.int和Integer都可以表示某一个数值；  
4.int和Integer不能够互用，因为他们两种不同的数据类型；  
int a1=1;  
int a2=1;  
Integer b1 =new Integer (1);  
Integer b2 =new Integer (1);  
------------------------------  
a1==a2 这个是成立的,很简单,都知道  
a1==b1 这个是不成立的.表达式的值为 false ,它们是不同的数据类型（在jdk1.5以上版本中为true）  
b1==b2 这个也是不成立的.表达式的值为 false,虽然是相同的数据类型,但是它们是两个对象,==比较的是2个对象的地址,它们的地址是不相等的,内容相等都是1;  
b1.equals(b2)==true 这个是成立的,表达式的值为 true. 相同数据类型,两个对象,地址不同,内容相同, quals比较的是2个对象的内容,所以成立。  
(a.equals(b),因为equals比较的是两个对象，所以a,b都不能为基本数据类型，否则会出编译错误。)（在jdk1.5以上版本中，b可以为基本数据类型，a不可以）  
同理,其它的封装类和基本类型也是这样的.  
java中equals和==的区别  
==比较的是2个对象的地址，而equals比较的是2个对象的内容。

在jdk1.5以上的版本中，基本类型和封装类能自动转化，与String类型的对象和字符串常量类似。  
        Integer i1 = 123;  
        Integer i2 = 123;  
          
        int i = 123;  
          
        Integer i3 = new Integer(123);  
        Integer i4 = new Integer(123);          
                 
        System.out.println("i1 == i2 = "+(i1 == i2));  
        System.out.println("i1.equals(i2) = "+(i1.equals(i2)));  
          
        System.out.println();  
        System.out.println("i3 == i4 = "+(i3 == i4));  
        System.out.println("i3.equals(i4) = "+(i3.equals(i4)));  
          
        System.out.println();  
        System.out.println("i2 == i4 = "+(i2 == i4));  
        System.out.println("i2.equals(i4) = "+(i2.equals(i4)));  
          
        System.out.println();  
        System.out.println("i == i2 = "+(i == i2));  
        System.out.println("i1.equals(i) = "+(i1.equals(i)));

        System.out.println();  
        System.out.println("i == i4 = "+(i == i4));  
        System.out.println("i4.equals(i) = "+(i4.equals(i)));

        ------------------------------   
       i1 == i2 = true  
      i1.equals(i2) = true

      i3 == i4 = false  
      i3.equals(i4) = true

      i2 == i4 = false  
      i2.equals(i4) = true

      i == i2 = true  
      i1.equals(i) = true

      i == i4 = true  
      i4.equals(i) = true

三、其他类怎么使用equals和==  
API里的类大部分都重写了equals方法，没有重写的一般是自己写的类，  
如果是你自己定义的一个类，比较自定义类用equals和==是一样的，都是比较句柄地址，  
因为自定义的类是继承于object，而object中的equals就是用==来实现的，你可以看源码。   
  
四、java里equals和hashCode之间什么关系  
只是为了维护 hashCode 方法的常规协定,才要求用equals比较的两个对象的hashCode相同.  
equals()和hashCode()都来自java.lang.Object.你当然可以重写.  
  
比如a.equals(b).仅当a的内存地址相等时,才返回true.当然如String等类已经对这个方法进行了重写,比较的就不再是内存地址了.  
hashCode()的值也是与内存地址相关的.所以仅当内存地址相等时,hashCode才相等.  
  
同样很多类也重写了这个方法,还是以String为例:  
public int hashCode() {  
int h = hash;  
if (h == 0) {  
      int off = offset;  
      char val[] = value;  
      int len = count;  
  
      for (int i = 0; i < len; i++) {  
         h = 31\*h + val[off++];  
      }  
      hash = h;  
   }  
   return h;  
}  
就不在与内存地址相关了.这样做是为了保证用equals比较返回为true的两个对象,他们的hashCode是相同的.  
  
所以一般重写equals的时候都会重写hashCode().  
当然,这个相当于一个约定,一个协议.你不这么做并不会错.   
  
五、hashCode  
在一般的应用中你不需要了解hashcode的用法，但当你用到hashmap，hashset等集合类时要注意下hashcode。  
  
你想通过一个object的key来拿hashmap的value，hashmap的工作方法是，  
通过你传入的object的hashcode在内存中找地址，  
当找到这个地址后再通过equals方法来比较这个地址中的内容是否和你原来放进去的一样，一样就取出value。   
  
所以这里要匹配2部分，hashcode和equals   
但假如说你new一个object作为key去拿value是永远得不到结果的，  
因为每次new一个object，这个object的hashcode是永远不同的，所以我们要重写hashcode，  
你可以令你的hashcode是object中的一个恒量，这样永远可以通过你的object的hashcode来找到key的地址，  
然后你要重写你的equals方法，使内存中的内容也相等。。。