hashcode（）和equals（）的作用、区别、联系

介绍一、

hashCode()方法和equal()方法的作用其实一样，在Java里都是用来对比两个对象是否相等一致，那么equal()既然已经能实现对比的功能了，为什么还要hashCode()呢？

因为重写的equal（）里一般比较的比较全面比较复杂，这样效率就比较低，而利用hashCode()进行对比，则只要生成一个hash值进行比较就可以了，效率很高，那么hashCode()既然效率这么高为什么还要equal()呢？

因为hashCode()并不是完全可靠，有时候不同的对象他们生成的hashcode也会一样（生成hash值得公式可能存在的问题），所以hashCode()只能说是大部分时候可靠，并不是绝对可靠，所以我们可以得出：

1.equal()相等的两个对象他们的hashCode()肯定相等，也就是用equal()对比是绝对可靠的。

2.hashCode()相等的两个对象他们的equal()不一定相等，也就是hashCode()不是绝对可靠的。

所有对于需要大量并且快速的对比的话如果都用equal()去做显然效率太低，**所以解决方式是，每当需要对比的时候，首先用hashCode()去对比，如果hashCode()不一样，则表示这两个对象肯定不相等（也就是不必再用equal()去再对比了）,如果hashCode()相同，此时再对比他们的equal()，如果equal()也相同，则表示这两个对象是真的相同了，这样既能大大提高了效率也保证了对比的绝对正确性！**

这种大量的并且快速的对象对比一般使用的hash容器中，比如hashset,hashmap,hashtable等等，比如hashset里要求对象不能重复，则他内部必然要对添加进去的每个对象进行对比，而他的对比规则就是像上面说的那样，先hashCode()，如果hashCode()相同，再用equal()验证，如果hashCode()都不同，则肯定不同，这样对比的效率就很高了。

然而hashCode()和equal()一样都是基本类Object里的方法，而和equal()一样，Object里hashCode()里面只是返回当前对象的地址，如果

我的标签

我的标签

java(1)

随笔分类

AJAX(18)
Android(1)
Apache(8)
Bootstrap(7)
DataTable(5)
docker(1)
DOM(8)
DWR(1)
easyUI(8)
EOS(4)
EXCEL(3)
Hibernate(15)
HTML(42)
HTTP(1)
Ibatis(8)
Java基础(125)
java设计模式(3)
jmx
jQuery(82)
JS(63)
JSON(4)
JSONP(11)
JSP(21)
Layer(1)
linux(15)
Maven(2)
myeclipse/eclipse(20)
MySql(9)
Netbeans(1)
Node.js
ORACLE(3)
redis(2)
select2(5)
Servlet(33)
Spring(8)
spring boot(1)
spring framework
springMVC(21)
Sql server 2008(3)
SSH(2)
SshClient(1)
Struts(35)
sudy(4)
SVN(11)
Swing(1)
Taglib(1)
tomcat(20)
web功能代码收录(4)
Web开发(22)
windows(3)
XML(9)
XPath
Ztree(1)
常用代码集合(11)
计算机英文
建模经验(1)
名人言录(2)
轻应用(1)
上传(1)
网络(13)
正则表达式(3)
总结(1)

是这样的话，那么我们相同的一个类，new两个对象，由于他们在内存里的地址不同，则他们的hashCode（ ）不同，所以这显然不是我们想要的，所以我们必须重写我们类的hashCode()方法，即一个类，在hashCode()里面返回唯一的一个hash值，比如下面：

自定义一个类

```
class Person{
  int num;
  String name;

  public int hashCode(){
    return num*name.hashCode();
}
}
```

由于标识这个类的是他的内部的变量num和name,所以我们就根据他们返回一个hash值，作为这个类的唯一hash值。

所以如果我们的对象要想放进hashSet，并且发挥hashSet的特性（即不包含一样的对象），则我们就要重写我们类的hashCode()和

equal()方法了。像String,Integer等这种类内部都已经重写了这两个方法。

当然如果我们只是平时想对比两个对象 是否一致，则只重写一个equal()，然后利用equal()去对比也行的。

介绍二、

哈希码(HashCode)

哈希码产生的依据：哈希码并不是完全唯一的，它是一种算法，让同一个类的对象按照自己不同的特征尽量的有不同的哈希码，但不表示不同的对象哈希码完全不同。也有相同的情况，看程序员如何写哈希码的算法。

什么是哈希码(HashCode)

在Java中，哈希码代表对象的特征。
例如对象 String str1 = “aa”, str1.hashCode= 3104
String str2 = “bb”, str2.hashCode= 3106
String str3 = “aa”, str3.hashCode= 3104
根据HashCode由此可得出str1!=str2,str1==str3
下面给出几个常用的哈希码的算法。
1：Object类的hashCode.返回对象的内存地址经过处理后的结构，由于每个对象的内存地址都不一样，所以哈希码也不一样。
2：String类的hashCode.根据String类包含的字符串的内容，根据一种特殊算法返回哈希码，只要字符串所在的堆空间相同，返回的哈希码也相同。
3：Integer类，返回的哈希码就是Integer对象里所包含的那个整数的数值，例如Integer i1=new Integer(100),i1.hashCode的值就是100 。

由此可见，2个一样大小的Integer对象，返回的哈希码也一样。

equals方法在hibernate中的应用。

equals方法是默认的判断2个对象是否相等的方法，在Object类里有实现，判断的是2个对象的内存地址。在hibernate中，不允许存在同类对象中有2个一样的实例。hibernate通过equals方法做判断。如：

```
User u1 = new User("张三");
User u2 = new User("李四");
User u3 = new User("张三");
```

按照项目需求，用户只要名字相同，就表示同一个用户，所以我们认为，u1和u3是同一个人，同一个对象。但是因为u1,u2,u3三者的内存地址都各不相同，所以hibernate会认为这是3个不同的对象。这与我们假设的出了矛盾。 因此，我们将覆盖Object类中的equals方法。

```
public class User{
private String userName;
….//get ，set方法省
//覆盖Object里的equals方法
public boolean equals(Object arg0){
if (!(arg0 instanceof User)){
return false;
}
User user = (User)arg0;
//如果名字相同，则表示属于同一个对象。
if(user.getName().equals(this.getName)){
return true;
}else{
return false; }
}
```

这样hibernate在插入数据的时候，如果传过来一个叫"张三"的用户,hibernate会先判断有没有叫"张三"的用户，如果没有，就允许插入，如果有，就不允许插入。这样做可以保证数据的高度一致性，不同的项目有不同的需求，所以要根据自己的需求来覆盖equals方法。

equals和HashCode的关系

在hibernate中，它认为2个对象只要equals返回true,那么hashCode一定相等。 但是实际情况呢？

```
User u1 = new User("张三");
User u2 = new User("张三");
```

由于我们重写了User的equals方法，所以 u1.equals(u2);返回true 但是，User并没有重写hashCode方法，它用的是Object类的hashCode方法，所以 u1.hashCode = 31050006 u2.hashCode = 31587890 两者的hashCode并不相等。违背了hibernate的原则 由此hibernate会产生错误判断，又以为它们不是同一个对象，因此我们还得重写User 的hashCode方法。如何重写hashCode方法呢？

HashCode的重写

如第2节所讲，哈希码要完成这么一件事，首先要保证如果equlas出来的结果相等，那么hashCode也相等。像上面的u1和u2,由于名字都是"张三"，所以应该返回相同的hashCode。所以我们可以想一个办法。让User的哈希码返回User里面name字段的哈希码，这样就保证，名字相同的人，不但equlas方法相同，而且hashCode相等。 那么User类就变成

```
public class User{
private String userName;
//覆盖Object里的equals方法
public boolean equals(Object arg0){
if(!(arg0 instanceof User)){
return false;
```

}
User user = (User)arg0;
//如果名字相同，则表示属于同一个对象。
if (user.getName().equals(this.getName)){
return true;
}else{
return false;
}
}
//覆盖Object里的hashCode方法
public int hashCode() {
return name.hashCode();//返回名字的哈希码。
}
}
这样可以保证hibernate根据我们自己的需求来判断重复对象

介绍三、

一、equals方法的作用

1、默认情况（没有覆盖equals方法）下equals方法都是调用Object类的equals方法，而Object的equals方法主要用于判断对象的内存地址引用是不是同一个地址（是不是同一个对象）。

2 、要是类中覆盖了equals方法，那么就要根据具体的代码来确定equals方法的作用了，覆盖后一般都是通过对象的内容是否相等来判断对象是否相等。

没有覆盖equals方法代码如下：

[java] view plain copy

```java
//学生类
public class Student {
    private int age;
    private String name;

    public Student() {
    }
    public Student(int age, String name) {
        super();
        this.age = age;
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public void setName(String name) {
        this.name = name;
    }
```

```
25.  }
```

测试 代码如下：

[java] view plain copy

```
1.  import java.util.HashSet;
2.  import java.util.LinkedList;
3.  import java.util.Set;
4.
5.
6.  public class EqualsTest {
7.      public static void main(String[] args) {
8.          LinkedList<Student> list = new LinkedList<Studen
    t>();
9.          Set<Student> set = new HashSet<Student>();
10.         Student stu1  = new Student(3,"张三");
11.         Student stu2  = new Student(3,"张三");
12.         System.out.println("stu1 == stu2 : "+
    (stu1 == stu2));
13.         System.out.println("stu1.equals(stu2) : "+stu1.e
    quals(stu2));
14.         list.add(stu1);
15.         list.add(stu2);
16.         System.out.println("list size:"+ list.size());
17.
18.         set.add(stu1);
19.         set.add(stu2);
20.         System.out.println("set size:"+ set.size());
21.     }
22.
23. }
```

运行结果：

stu1 == stu2：false
stu1.equals(stu2)：false
list size:2
set size:2

结果分析：Student类没有覆盖equals方法，stu1调用equals方法实际上调用的是Object的equals方法。所以采用对象内存地址是否相等来判断对象是否相等。因为是两个新对象所以对象的内存地址不相等，所以stu1.equals(stu2) 是false。

3、我们覆盖一下equals方法（age和name属性），让Student类其通过判断对象的内容是否相等来确定对象是否相等。

覆盖后的Student类：

[java] view plain copy

```java
//学生类
public class Student {
    private int age;
    private String name;

    public Student() {
    }
    public Student(int age, String name) {
        super();
        this.age = age;
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Student other = (Student) obj;
        if (age != other.age)
            return false;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }

}
```

运行结果：

stu1 == stu2：false
stu1.equals(stu2)：true
list size:2
set size:2

结果分析：因为Student两个对象的age和name属性相等，而且又是通过覆盖equals方法来判断的，所示stu1.equals(stu2) 为true。注意以上几次测试list和set的size都是2

二、HashCode

4、通过以上的代码运行，我们知道equals方法已经生效。接下来我们在覆盖一下hashCode方法（通过age和name属性来生成hashcode）并不覆盖equals方法，其中Hash码是通过age和name生成的。

覆盖hashcode后的Student类：

[java] view plain copy

```java
//学生类
public class Student {
    private int age;
    private String name;

    public Student() {
    }
    public Student(int age, String name) {
        super();
        this.age = age;
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + age;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }
}
```

运行结果：

stu1 == stu2：false
stu1.equals(stu2)：false
list size:2
hashCode :775943

hashCode :775943
set size:2

结果分析：我们并没有覆盖equals方法只覆盖了hashCode方法，两个对象虽然hashCode一样，但在将stu1和stu2放入set集合时由于equals方法比较的两个对象是false，所以就没有在比较两个对象的hashcode值。

5、我们覆盖一下equals方法和hashCode方法。

Student代码如下：

[java] view plain copy

```
1.  //学生类
2.  public class Student {
3.      private int age;
4.      private String name;
5.      public Student() {
6.      }
7.      public Student(int age, String name) {
8.          super();
9.          this.age = age;
10.         this.name = name;
11.     }
12.     public int getAge() {
13.         return age;
14.     }
15.     public String getName() {
16.         return name;
17.     }
18.     public void setAge(int age) {
19.         this.age = age;
20.     }
21.     public void setName(String name) {
22.         this.name = name;
23.     }
24.     @Override
25.     public int hashCode() {
26.         final int prime = 31;
27.         int result = 1;
28.         result = prime * result + age;
29.         result = prime * result + ((name == null) ? 0 :
    name.hashCode());
30.         System.out.println("hashCode : "+ result);
31.         return result;
32.     }
33.     @Override
34.     public boolean equals(Object obj) {
35.         if (this == obj)
36.             return true;
37.         if (obj == null)
38.             return false;
39.         if (getClass() != obj.getClass())
40.             return false;
41.         Student other = (Student) obj;
42.         if (age != other.age)
43.             return false;
44.         if (name == null) {
```

```
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }

}
```

运行结果：

stu1 == stu2 : false

stu1.equals(stu2) :true

list size:2

hashCode :775943

hashCode :775943

set size:1

结果分析：stu1和stu2通过equals方法比较相等，而且返回的hashCode值一样，所以放入set集合中时只放入了一个对象。

6、下面我们让两个对象equals方法比较相等，但hashCode值不相等试试。

Student类的代码如下：

[java] view plain copy

```
//学生类
public class Student {
    private int age;
    private String name;
    <span style="color:#ff0000;">private static int index=5;</span>
    public Student() {
    }
    public Student(int age, String name) {
        super();
        this.age = age;
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public void setName(String name) {
        this.name = name;
    }
```

```
25.     @Override
26.     public int hashCode() {
27.         final int prime = 31;
28.         int result = 1;
29.         result = prime * result + <span style="color:#ff
0000;">(age+index++)</span>;
30.         result = prime * result + ((name == null) ? 0 :
name.hashCode());
31.         <span style="color:#ff0000;">System.out.println(
"result :"+result);</span>
32.         return result;
33.     }
34.     @Override
35.     public boolean equals(Object obj) {
36.         if (this == obj)
37.             return true;
38.         if (obj == null)
39.             return false;
40.         if (getClass() != obj.getClass())
41.             return false;
42.         Student other = (Student) obj;
43.         if (age != other.age)
44.             return false;
45.         if (name == null) {
46.             if (other.name != null)
47.                 return false;
48.         } else if (!name.equals(other.name))
49.             return false;
50.         return true;
51.     }
52.
53. }
```

运行结果：

stu1 == stu2：false
stu1.equals(stu2)：true
list size:2
hashCode :776098
hashCode :776129
set size:2

结果分析：虽然stu1和stu2通过equals方法比较相等，但两个对象的hashcode的值并不相等，所以在将stu1和stu2放入set集合中时认为是两个不同的对象。

7、修改stu1的某个属性值

Student代码如下：

[java] view plain copy

```
1.  //学生类
2.  public class Student {
3.      private int age;
4.      private String name;
5.      public Student() {
```

```
    }
    public Student(int age, String name) {
        super();
        this.age = age;
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + age;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        System.out.println("hashCode : "+ result);
        return result;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Student other = (Student) obj;
        if (age != other.age)
            return false;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }

}
```

测试代码如下：

[java] view plain copy

```
import java.util.HashSet;
```

```
2.  import java.util.LinkedList;
3.  import java.util.Set;
4.
5.
6.  public class EqualsTest {
7.      public static void main(String[] args) {
8.          LinkedList<Student> list = new LinkedList<Studen
    t>();
9.          Set<Student> set = new HashSet<Student>();
10.         Student stu1  = new Student(3,"张三");
11.         Student stu2  = new Student(3,"张三");
12.         System.out.println("stu1 == stu2 : "+
    (stu1 == stu2));
13.         System.out.println("stu1.equals(stu2) : "+stu1.e
    quals(stu2));
14.         list.add(stu1);
15.         list.add(stu2);
16.         System.out.println("list size:"+ list.size());
17.
18.         set.add(stu1);
19.         set.add(stu2);
20.         System.out.println("set size:"+ set.size());
21.         stu1.setAge(34);
22.         System.out.println("remove stu1 : "+set.remove(s
    tu1));
23.         System.out.println("set size:"+ set.size());
24.     }
25.
26. }
```

运行结果：

stu1 == stu2：false
stu1.equals(stu2)：true
list size:2
hashCode：775943
hashCode：775943
set size:1
hashCode：776904
remove stu1：false
set size:1

结果分析：

当我们将某个对象存到set中时，如果该对象的属性参与了hashcode的计算，那么以后就不能修改该对象参与hashcode计算的那些属性了，否则会引起意向不到的错误的。正如测试中，不能够移除stu1对象。
总结：

1、equals方法用于比较对象的内容是否相等（覆盖以后）

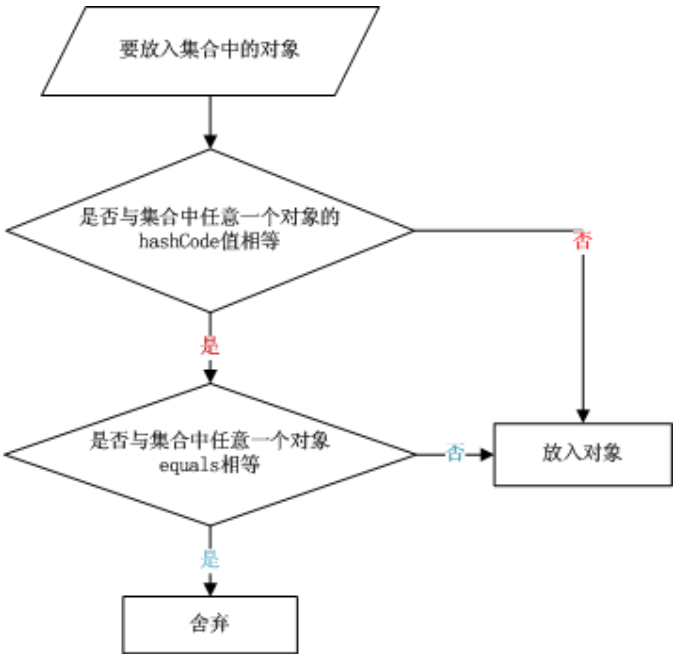
2、hashcode方法只有在集合中用到

3、当覆盖了equals方法时，比较对象是否相等将通过覆盖后的equals方法进行比较（判断对象的内容是否相等）。

4、将对象放入到集合中时，首先判断要放入对象的hashcode值与集合中的任意一个元素的hashcode值是否相等，如果不相等直接将该对象放入集合中。如果hashcode值相等，然后再通过equals方法判断要放入对象与集

合中的任意一个对象是否相等，如果equals判断不相等，直接将该元素放入到集合中，否则不放入。

5、将元素放入集合的流程图：



6、HashSet中add方法源代码：

[java] view plain copy

```
1. public boolean add(E e) {
2.     return map.put(e, PRESENT)==null;
3.     }
```

map.put源代码:

[java] view plain copy

```
1. <pre name="code" class="java"> public V put(K key, V val
   ue) {
2.         if (key == null)
3.             return putForNullKey(value);
4.         int hash = hash(key.hashCode());
5.         int i = indexFor(hash, table.length);
6.         for (Entry<K,V> e = table[i]; e != null; e = e.n
   ext) {
7.             Object k;
8.             if (e.hash == hash && ((k = e.key) == key ||
    key.equals(k))) {
9.                 V oldValue = e.value;
10.                e.value = value;
11.                e.recordAccess(this);
12.                return oldValue;
13.            }
14.        }
15.
```

```
16.             modCount++;
17.             addEntry(hash, key, value, i);
18.             return null;
19.         }</pre>
20.     <pre></pre>
21.     <pre></pre>
22.
```

顶

79

踩

分类: Java基础

十年磨一剑_卧薪尝胆
关注 - 24
粉丝 - 56
+加关注

0 0

« 上一篇：关于JSP页面中的pageEncoding和contentType两种属性的区别

» 下一篇：iterator与iterable的区别和联系

posted on 2017-07-05 10:13 十年磨一剑_卧薪尝胆 阅读(1670) 评论(0) 编辑 收藏