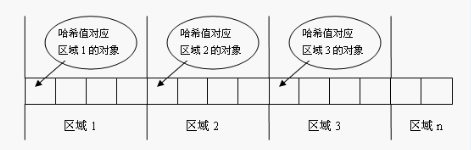
hashCode方法与HashSet类

如果想查找一个集合中是否包含有某个对象，大概的程序代码怎样写呢？

你通常是逐一取出每个元素与要查找的对象进行比较，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个元素，譬如一万个元素，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一万个元素进行逐一比较才能得到结论。有人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在哪个区域，如下图：



HashSet就是采用哈希算法存取对象的集合，它内部采用对某个数字n进行取余的方式对哈希码进行分组和划分对象的存储区域。Object类中定义了一个hashCode()方法来返回每个Java对象的哈希码，当从HashSet集合中查找某个对象时，Java系统首先调用对象的hashCode()方法获得该对象的哈希码，然后根据哈希码找到相应的存储区域，最后取出该存储区域内的每个元素与该对象进行equals方法比较，这样不能遍历集合中的所有元素就可以得到结论，可见，HashSet集合具有很好的对象检索性能。但是，HashSet集合存储对象的效率相对低些，因为向HashSet集合中添加一个对象时，要先计算出对象的哈希码和根据这个哈希码确定对象在集合中的存放位置。

为了保证一个类的实例对象能在HashSet中正常存储，要求这个类的两个实例对象用equals()方法比较的结果相等时，他们的哈希码也必须相等。也就是说，如果 obj1.equals(obj2)的结果为true,那么以下表达式的结果也要为true:

obj1.hashCode() == obj2.hashCode()

如果一个类的hashCode()方法没有遵循上述要求，那么，当这个类的两个实例对象用equals()方法比较的结果相等时，它们本来应该无法被同时存储进Set集合中，但是，如果将它们存储进HashSet集合中时，由于它们的hashCode()方法的返回值不同，第二个对象首先按哈希码计算可能会被放进与第一个对象不同的区域中，这样，它就不可能与第一个对象进行equals比较了，也就可能被存储进HashSet集合中了。Object类中的hashCode()方法不能满足对象被存入到HashSet中的要求，因为它的返回值是通过对象的内存地址推算出来的，同一个对象在程序运行期间的任何时候返回的哈希值都是始终不变的，所以，只要是两个不同的实例对象，即使它们的equals()方法比较结果相等，它们默认的hashCode方法的返回值是不同的。只要将15-3中使用的ArrayList集合改为使用HashSet集合就可以看到这种错误的效果了，修改后的代码如例程15-5所示：



例程15-5运行的结果为：在删除元素前打印出的集合中的元素个数为3,删除元素后再次打印出的集合中的元素个数还为3，这个结果显然是错误的。这是因为例程15-4中定义的User类虽然覆盖了equals()方法，但没有覆盖hashCode()方法，当两个User实例对象的equals方法比较的结果相等时，而这两个对象的哈希码却不同，所以导致它们都能存储进HashSet，这样我们使用HashSet存储不重复对象的初衷是相悖的。

**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。**

提示：

1. 通常来说，一个类的两个实例对象用equals()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equals方法比较的结果可以不等，例如，字符串“BB”和“Aa”的equals方法比较结果肯定不相等，但它们的hashCode方法返回值却相等。
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，在这种清况下，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。

示例代码：

ReflectTest2测试类：

**public** **class** ReflectTest2 {

**public** **static** **void** main(String[] args) **throws** Exception {

Collection collections = **new** HashSet();

ReflectPoint pt1 = **new** ReflectPoint(3, 3);

ReflectPoint pt2 = **new** ReflectPoint(5, 5);

ReflectPoint pt3 = **new** ReflectPoint(3, 3);

//pt1和pt3,两个对象采用默认equals比较时比较的是HashCode值,HashCode值通常是根据内存地址换算的.

collections.add(pt1);

collections.add(pt2);

collections.add(pt3);

collections.add(pt1);

//如果想要pt1和pt3相等，必须在ReflectPoint类中重写equals方法.

System.*out*.println(pt1.equals(pt3));

//如果想要pt1和pt3的哈希码相等，必须在ReflectPoint类中重写hashCode方法. System.*out*.println(pt1.hashCode()==pt3.hashCode()); System.*out*.println(collections.size());

//如果此处修改了pt1的属性，因为pt1的y属性参与了hashCode值得计算，所以pt1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，那么remove(pt1)就会失败， // pt1.y = 7 ;

collections.remove(pt1);

System.*out*.println(collections.size());

System.*out*.println(pt1.equals(pt3));

System.*out*.println(collections.contains(pt1));

}

}

ReflectPoint类代码：

**public** **class** ReflectPoint {

**private** **int** x;

**public** **int** y;

//alt+shift+s快捷键生成构造方法

**public** ReflectPoint(**int** x, **int** y) {

**super**();

**this**.x = x;

**this**.y = y;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + x;

result = prime \* result + y;

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

ReflectPoint other = (ReflectPoint) obj;

**if** (x != other.x)

**return** **false**;

**if** (y != other.y)

**return** **false**;

**return** **true**;

}

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

}