lulipro

学习知识的最大价值在于愉悦自己的大脑，其次才是让自己生存。

一次性搞清楚equals和hashCode

前言

在程序设计中，有很多的“公约”，遵守约定去实现你的代码，会让你避开很多坑，这些公约是前人总结出来的设计规范。

Object类是Java中的万类之祖，其中，equals和hashCode是2个非常重要的方法。

这2个方法总是被人放在一起讨论。最近在看集合框架，为了打基础，就决定把一些细枝末节清理掉。一次性搞清楚！

下面开始剖析。

public boolean equals(Object obj)

Object类中默认的实现方式是： return this == obj 。那就是说，只有this 和 obj引用同一个对象，才会返回true。

而我们往往需要用equals来判断 2个对象是否等价，而非验证他们的唯一性。这样我们在实现自己的类时，就要重写equals.

按照约定，equals要满足以下规则。

自反性: x.equals(x) 一定是true

对null: x.equals(null) 一定是false

对称性: x.equals(y) 和 y.equals(x)结果一致

传递性: a 和 b equals , b 和 c equals，那么 a 和 c也一定equals。

一致性: 在某个运行时期间，2个对象的状态的改变不会不影响equals的决策结果，那么，在这个运行时期间，无论调用多少次equals，都返回相同的结果。

一个例子

```
class Test
{
    private int num;
    private String data;

    public boolean equals(Object obj)
    {
        if (this == obj)
            return true;

        if ((obj == null) || (obj.getClass() != this.getClass()))
            return false;

        //能执行到这里，说明obj和this同类且非null。
```


阅读排行榜

评论排行榜

```
14          Test test = (Test) obj;
15          return num == test.num&& (data == test.data || (data != null &&
data.equals(test.data)));
16      }
17
18      public int hashCode()
19      {
20          //重写equals，也必须重写hashCode。具体后面介绍。
24      }
25
26  }
```

equals编写指导

Test类对象有2个字段，num和data，这2个字段代表了对象的状态，他们也用在equals方法中作为评判的依据。

在第8行，传入的比较对象的引用和this做比较，这样做是为了 save time ，节约执行时间，如果this 和 obj是 对同一个堆对象的引用，那么，他们一定是qeuals 的。

接着，判断obj是不是为null，如果为null，一定不equals，因为既然当前对象this能调用equals方法，那么它一定不是null，非null 和 null当然不等价。

然后，比较2个对象的运行时类，是否为同一个类。不是同一个类，则不equals。getClass返回的是 this 和obj的运行时类的引用。如果他们属于同一个类，则返回的是同一个运行时类的引用。注意，一个类也是一个对象。

1、有些程序员使用下面的第二种写法替代第一种比较运行时类的写法。应该避免这样做。

3. 【C51】单片机定时器介绍(8)
4. 【C51】单片机芯片之——图解74HC595(7)
5. 用Arduino剖析PWM脉宽调制(6)

推荐排行榜

1. 一次性搞清楚equals和hashCode(13)
2. C语言指针详解(9)
3. 树莓派wiringPi库详解(5)
4. C++头文件，预处理详解(5)
5. Java中的异常和处理详解(5)

```
if((obj == null) || (obj.getClass() != this.getClass()))

     return false;


if(!(obj instanceof Test))

     return false; // avoid 避免！
```

它违反了公约中的对称原则。

例如：假设Dog扩展了Aminal类。

dog instanceof Animal　　得到true

animal instanceof Dog　　得到false

这就会导致

animal.equls(dog) 返回true
dog.equals(animal) 返回false

仅当Test类没有子类的时候，这样做才能保证是正确的。

2、按照第一种方法实现，那么equals只能比较同一个类的对象，不同类对象永远是false。但这并不是强制要求的。一般我们也很少需要在不同的类之间使用equals。

3、在具体比较对象的字段的时候，对于基本值类型的字段，直接用 == 来比较（注意浮点数的比较，这是一个坑）对于引用类型的字段，你可以调用他们的equals，当然，你也需要处理字段为null 的情况。对于浮点数的比较，我在看Arrays.binarySearch的源代码时，发现了如下对于浮点数的比较的技巧：

```
if ( Double.doubleToLongBits(d1) == Double.doubleToLongBits(d2) ) //d1 和 d2 是double类型

if(  Float.floatToIntBits(f1) == Float.floatToIntBits(f2)   )      //f1 和 f2 是d2是float类型
```

4、并不总是要将对象的所有字段来作为equals 的评判依据，那取决于你的业务要求。比如你要做一个家电功率统计系统，如果2个家电的功率一样，那就有足够的依据认为这2个家电对象等价了，至少在你这个业务逻辑背景下是等价的，并不关心他们的价钱啊，品牌啊，大小等其他参数。

5、最后需要注意的是，equals 方法的参数类型是Object，不要写错！

public int hashCode()

这个方法返回对象的散列码，返回值是int类型的散列码。
对象的散列码是为了更好的支持基于哈希机制的Java集合类，例如 Hashtable, HashMap, HashSet 等。

关于hashCode方法，一致的约定是：

重写了euqls方法的对象必须同时重写hashCode()方法。

如果2个对象通过equals调用后返回是true，那么这个2个对象的hashCode方法也必须返回同样的int型散列码

如果2个对象通过equals返回false，他们的hashCode返回的值**允许相同**。(然而，程序员必须意识到，hashCode返回独一无二的散列码，会让存储这个对象的hashtables更好地工作。)

在上面的例子中，Test类对象有2个字段，num和data，这2个字段代表了对象的状态，他们也用在equals方法中作为评判的依据。那么， 在hashCode方法中，这2个字段也要参与hash值的运算，作为hash运算的中间参数。这点很关键，这是为了遵守：2个对象equals，那么 hashCode一定相同规则。

也是说，参与equals函数的字段，也必须都参与hashCode 的计算。

合乎情理的是：同一个类中的不同对象返回不同的散列码。典型的方式就是根据对象的地址来转换为此对象的散列码，但是这种方式对于Java来说并不是唯一的要求的
的实现方式。通常也不是最好的实现方式。

相比 于 equals公认实现约定，hashCode的公约要求是很容易理解的。有2个重点是hashCode方法必须遵守的。约定的第3点，其实就是第2点的
细化，下面我们就来看看对hashCode方法的一致约定要求。

第一：在某个运行时期间，只要对象的（字段的）变化不会影响equals方法的决策结果，那么，在这个期间，无论调用多少次hashCode，都必须返回同一个散列码。

第二：通过equals调用返回true 的2个对象的hashCode一定一样。

第三：通过equasl返回false 的2个对象的散列码不需要不同，也就是他们的hashCode方法的返回值允许出现相同的情况。

总结一句话：等价的(调用equals返回true)对象必须产生相同的散列码。不等价的对象，不要求产生的散列码不相同。

hashCode编写指导

在编写hashCode时，你需要考虑的是，最终的hash是个int值，而不能溢出。不同的对象的hash码应该尽量不同，避免hash冲突。

那么如果做到呢？下面是解决方案。

1、定义一个int类型的变量 hash,初始化为 7。

接下来让你认为重要的字段（equals中衡量相等的字段）参入散列运，算每一个重要字段都会产生一个hash分量，为最终的hash值做出贡献（影响）

运算方法参考表

重要字段var的类型	他生成的hash分量
byte, char, short ， int	(int)var
long	(int)(var ^ (var >>> 32))
boolean	var?1:0

float	Float.floatToIntBits(var)
double	long bits = Double.doubleToLongBits(var); 分量 = (int)(bits ^ (bits >>> 32));
引用类型	(null == var ? 0 : var.hashCode())

最后把所有的分量都总和起来，注意并不是简单的相加。选择一个倍乘的数字31，参与计算。然后不断地递归计算，直到所有的字段都参与了。

```
int hash = 7;

hash = 31 * hash + 字段1贡献分量;

hash = 31 * hash + 字段2贡献分量;

.....

return hash;
```

说明，以下的内容是我在google上找到并翻译整理的，其中加入了自己的话和一些例子，便于理解，但我能保证这并不影响整体准确性。

英文原文：http://www.javaranch.com/journal/2002/10/equalhash.html

分类: Java 基础

« 上一篇：Java迭代：Iterator和Iterable接口
» 下一篇：Tomcat安装和配置

posted @ 2016-07-01 21:49 lulipro 阅读(46677) 评论(13) 编辑 收藏

评论列表

#1楼 2017-08-02 10:51 **学计算机的人绝不认输**

楼主，写的不错！！！

支持(0) 反对(0)

#2楼 2017-08-10 13:35 **小姐姐爱糖糖棒**

你写的很棒，支持一下，也希望可以看到更多更棒的blog

支持(0) 反对(0)

#3楼 2017-08-18 10:29 黑胡渣

if (Double.doubleToLongBits(d1) == Double.doubleToLongBits(d2)) //d1 和 d2 是double类型

if(Float.floatToIntBits(f1) == Float.floatToIntBits(f2)) //f1 和 f2 是d2是float类型

关于浮点的比较 当超过浮点的精度时就不能获得正确的值了
请问有什么方法可以解决,目前只想到 了bigdecimal,但是必须要用String去构造创建,如果去讲浮点转化成String也会遇到超过精度的问题,只能把类定义为String去操作
谢谢

支持(0) 反对(0)

#4楼[楼主] 2017-08-18 17:04 lulipro

@ 黑胡渣
你的要求是对浮点数精度要求很高，那基本类型double是不满足的。你可以使用Java中的BigDecimal类，他支持以一个String类构造实例。

```
1  public static void main(String[] args)
2      {
3
4          BigDecimal num1 = new BigDecimal("123243.565364500");
5          BigDecimal num2 = new BigDecimal("123243.5653645");
6
7
8          /* 使用equals方法：必须在数值上相等，且在精度（也就是小数部分的位数）上也相等才会判定他们相等。
9           * 例如 211.122 和 211.1220 不是同的，equals会返回false。
10          * */
11
12         /*
13          * 如果你只需比较他们是否在数值上相等，就使用compareTo方法。
```

```
14          * 例如  211.122000 和  211.122 是相等的，compareTo会返回0
15          *
16          * */
17
18         System.out.println(num1.equals(num2));
19         System.out.println(num1.compareTo(num2));
20
21     }
```

支持(1) 反对(0)

#5楼 2017-08-18 17:06 **黑胡渣**

@ 代码钢琴家
恩,谢谢回复,看来跟我评论里想的一样只能把类型定义为String 走BigDecimal

支持(0) 反对(0)

#6楼 2017-09-21 07:18 **James.H.Fu**

很棒。楼主是如何学习这些基础的哦？有哪些好书，好博文推荐一下。

支持(0) 反对(0)

#7楼[楼主] 2017-09-22 17:36 **lulipro**

@ James.H.Fu
书本的话，都不会把单个基础知识点讲这么详细的，这些都是在墙外浏览的时候找到的，自己的翻译下，然后记下来。还有多去论坛逛逛。

支持(1) 反对(0)

#8楼 2017-09-25 10:35 **junxuelian**

好文啊！

支持(0) 反对(0)

#9楼 2017-11-16 16:05 Rookieek

感觉这里是不是写反了：
obj.getClass() != this.getClass()
而不用
obj instanceof Test
用getClass()的话Test的子类和Test才无论如何比较都是false，用instanceof的话不管Test类和Test的子类都是和Test相比的，
只要子类不覆写equals()方法就不会出现dog.equals(animal) 返回false的情况

支持(0) 反对(0)

#10楼 2018-03-14 15:31 qqqyyf

厉害了我的楼主

支持(0) 反对(0)

#11楼 2018-03-14 15:31 qqqyyf

厉害了我的楼主

支持(0) 反对(0)

#12楼 2018-03-28 09:03 parahaoer

楼主，为什么不同对象的hashcode可以相同呢？

支持(0) 反对(0)

#13楼[楼主] 2018-04-04 17:00 lulipro

@ parahaoer
比如我们规定Car类的hash算法为：hashCode = int字段值x2+short字段的值x1。
car.speed = 1 ; //int类型
car.acc = 2 ; //short类型
则car的hashCode = = 4

而User类型的hash算法为：hashCode = int字段值x4。
user.age = 1
则user的hashCode = 4

可见，不同对象的状态值，成员个数不同，而且不同类使用的hash算法也可能不同。所以这些变数使得2个不同对象的hashCode一样成为可能。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。