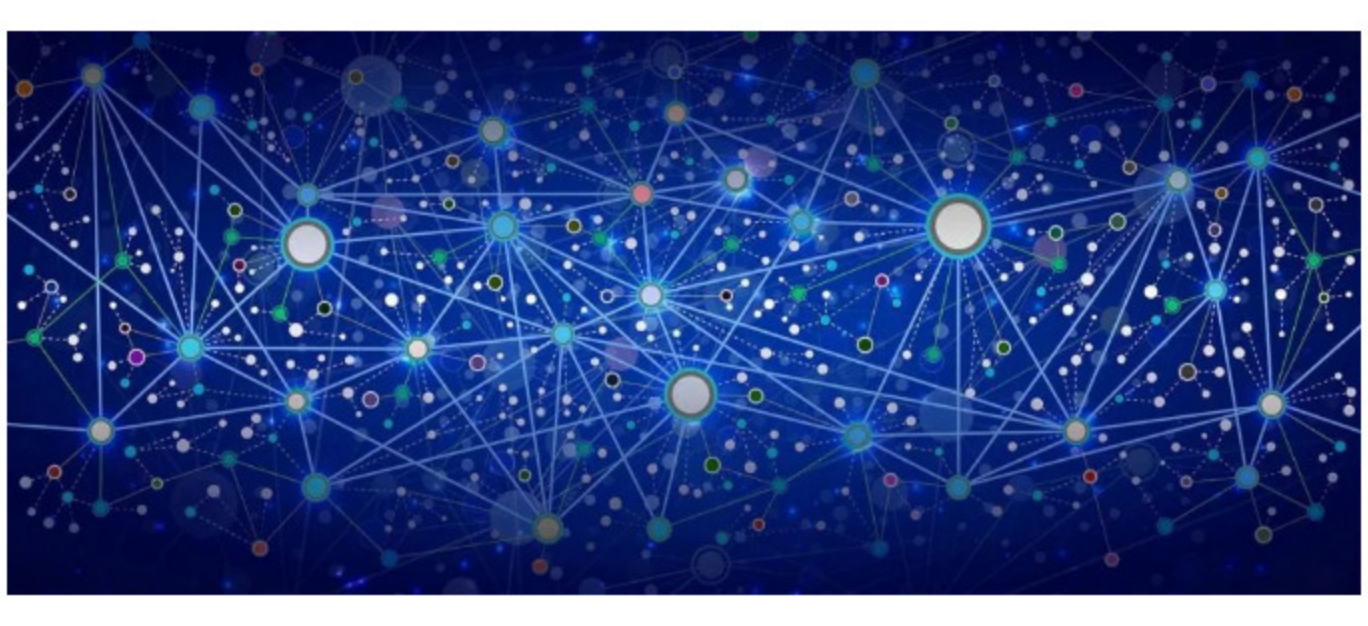
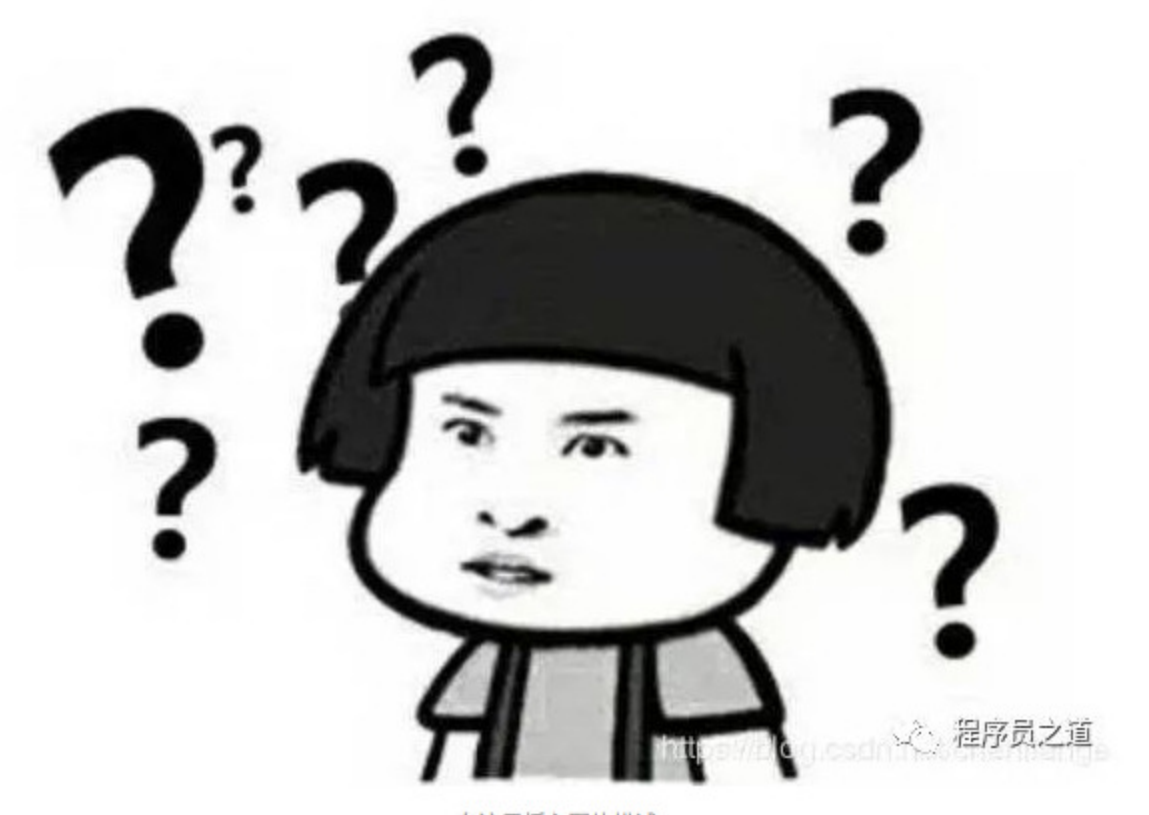
头条面试官问了几个equals的问题，我竟然没答上来！呜呜呜！

程序员面试 5月29日

以下文章来源于程序员之道，作者知行之道



擦，这两个值明明应该是相等的啊，为啥我用==判断的结果时不等于，真是活见鬼了。我来debug看看。关于对象、值一些相等的判断不知道你有没有踩过坑，或者面试的时候有没有被面试官坑过？我就被头条面试官坑过。相等判断几连问，直接晕倒。



在这里插入图片描述

如果你还没注意过这些问题，那好好看看下面的内容吧。

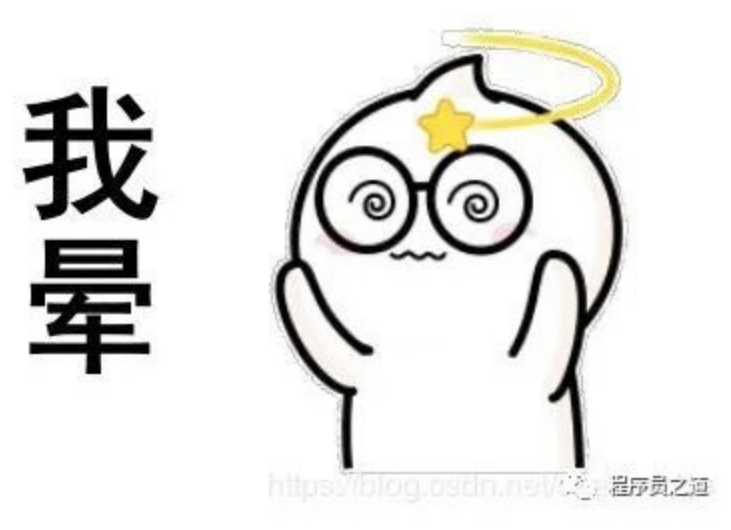
怎么判断相等

使用java编程时，经常写一些判断相等的代码，应该使用==还是equals呢？

1. 运算符==针对的是值的相等判断，应用类型是基本数据类型，说到基本数据类型，你应该不陌生吧，java里的基本数据类型有char、byte、short、int、long、double、float、bool，但这里针对的基本数据类型是char、byte、short、int、long，对于double、float的等于判断(涉及精度问题)，应该是二者的差值在一个区间内才认为是相等，bool一般只会判断true或false，很少会使用这个判等运算符。
2. equals针对的是引用类型，也就是实际地址里存储的对象内容相等，可以理解为c语言里指针内容判等。equals使用的场景是类的对象(包括包装类型Integer、Short等)，实际比较的是两个地址的内容是否相等一致。

这几个问题都能回答对吗？

但是在实际使用的时候，因为java jvm帮我们做了一些cache的优化，还是有一些坑点的，以下几个问题都可以回答对吗？



在这里插入图片描述

- 使用 == 对两个值为 99 的直接赋值的 Integer 对象判等；
- 使用 == 对两个值为 128 的直接赋值的 Integer 对象判等；
- 使用 == 对一个值为 99 的直接赋值的 Integer 和另一个通过 new Integer 声明的值为 99 的对象判等；
- 使用 == 对一个值为 128 的直接赋值的 Integer 和另一个通过 new Integer 声明的值为128 的对象判等；
- 使用 == 对两个通过 new Integer 声明的值为 99 的对象判等；
- 使用 == 对一个值为 128 的直接赋值的 Integer 对象和另一个值为 128 的 int 基本类型判等。

正确结果

```
    public static void main(String[] args) {
        boolean result;
        Integer a = 99;
        Integer b = 99;
        result = a == b;
        System.out.println("Interger a = 99 is == Interger b = 99: " + result);

        Integer c = 128;
        Integer d = 128;
        result = c == d;
        System.out.println("Interger c = 128 is == Interger d = 128: " + result);

        Integer e = 99;
        Integer f = new Integer(99);
        result = e == f;
        System.out.println("Interger e = 99 is == Interger f = new Integer(99): " + res

        Integer g = 128;
        Integer h = new Integer(128);
        result = g == h;
        System.out.println("Interger g = 128 is == Interger h = new Integer(128): " + r

        Integer i = new Integer(99);
        Integer j = new Integer(99);
        result = i == j;
        System.out.println("Interger i = new Integer(99) is == j = new Integer(99): " +

        Integer k = 128;
        int l = 128;
        result = k == l;
        System.out.println("Integer k = 128 is == int l = 128: " + result);
    }
}
```

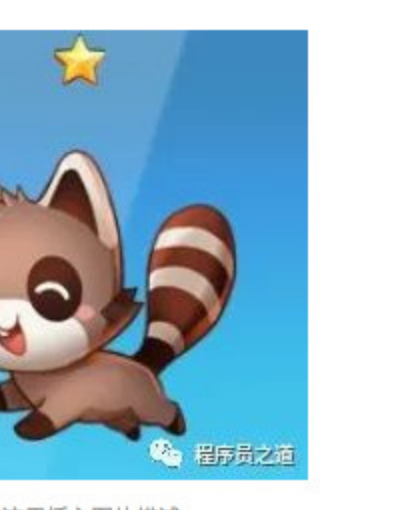
大家简单的在脑子里过一下程序，不知道这几个例子你都能回答对吗？

```
先来看下运行结果:
Interger a = 99 is == Interger b = 99: true
Interger c = 128 is == Interger d = 128: false
Interger e = 99 is == Interger f = new Integer(99): false
Interger g = 128 is == Interger h = new Integer(128): false
Interger i = new Integer(99) is == j = new Integer(99): false
Integer k = 128 is == int l = 128: true

Process finished with exit code 0
```

IntegerCache搞的鬼

首先，对于Integer a = 99的这种直接赋值，jvm会编译优化成Interger.valueOf(99)，而对于-128~127之间的值，对于Integer类型，为了优化内存使用，jvm对于这个范围内的值，初始化了内存cache，也可以加快访问速度，防止重复建立对象，可以看一下java的源码是怎么实现valueOf的：



在这里插入图片描述

```
    /**
     * Returns an {@code Integer} instance representing the specified
     * {@code int} value.  If a new {@code Integer} instance is not
     * required, this method should generally be used in preference to
     * the constructor {@link #Integer(int)}, as this method is likely
     * to yield significantly better space and time performance by
     * caching frequently requested values.
     *
     * This method will always cache values in the range -128 to 127,
     * inclusive, and may cache other values outside of this range.
     *
     * @param  i an {@code int} value.
     * @return an {@code Integer} instance representing {@code i}.
     * @since  1.5
     */
    public static Integer valueOf(int i) {
        if (i >= IntegerCache.low && i <= IntegerCache.high)
            return IntegerCache.cache[i + (-IntegerCache.low)];
        return new Integer(i);
    }
```

默认cache的区间是-128~127，也可以自定义cache的high区间。

```
    private static class IntegerCache {
        static final int low = -128;
        static final int high;
        static final Integer cache[];

        static {
            // high value may be configured by property
            int h = 127;
            String integerCacheHighPropValue =
                sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
            if (integerCacheHighPropValue != null) {
                try {
                    int i = parseInt(integerCacheHighPropValue);
                    i = Math.max(i, 127);
                    // Maximum array size is Integer.MAX_VALUE
                    h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
                } catch( NumberFormatException nfe) {
                    // If the property cannot be parsed into an int, ignore it.
                }
            }
            high = h;

            cache = new Integer[(high - low) + 1];
            int j = low;
            for(int k = 0; k < cache.length; k++)
                cache[k] = new Integer(j++);

            // range [-128, 127] must be interned (JLS7 5.1.7)
            assert IntegerCache.high >= 127;
        }

        private IntegerCache() {}
    }
```

结果分析

知道了这些，我们来分析下刚才的几个例子运行结果为什么是那样的？首先明确，==比较的是地址，而非地址的内容，其次有IntegerCache在搞怪。

- 值为99的两个Integer对象直接赋值，编译优化成valueOf(99)，都是IntegerCache的同一个对象，所以结果为true。
- 两个值为 128 的直接赋值的 Integer 对象超出了IntegerCache的范围，会直接new一个Integer对象，所以不等。设置 JVM 参数加上 -XX:AutoBoxCacheMax=2000，保证128在cache的范围区间内，返回结果就会是true了。
- 一个值为 99 的直接赋值的 Integer 和另一个通过 new Integer 声明的值为 99 的对象，一个是IntegerCache里的对象，一个是new出的Integer对象，地址肯定不同嘛，所以为false。
- 一个值为 128 的直接赋值的 Integer 和另一个通过 new Integer 声明的值为128 的对象，这个跟例子2是一样的，相当于都是new了两个Integer对象，当然是不同的，返回false。
- 对两个通过 new Integer 声明的值为 99 的对象，还是一样的嘛，都是new出的对象，比较必然不等，返回false。
- 对一个值为 128 的直接赋值的 Integer 对象和另一个值为 128 的 int 基本类型，这里要注意Integer对象和int型的值比较，java会进行自动拆箱，都退化成int型的值比较，所以最终结果肯定是相等的了。

所以对于对象相等的比较请使用equals，而非==，==是用于值比较的。
在我们的平时开发中，也要注意类中有用到Integer声明的变量，比较相等时，一定要使用equals，如果使用==比较，可能会出现莫名其妙的错误(-128~127之间的比较正常，其他值的比较就不符合预期了)。

对于使用==判断对象相等，Intellij的插件也会提示我们应该使用equals。

在这里插入图片描述

String对象又怎么判断相等呢？

String是对象，那判断相等的方式肯定是使用equal了。但如果用==判断会有什么后果呢？这里也一起看一下。

```
    public static void stringEqual() {
        boolean result;
        String a = "this";
        String b = "this";
        result = a == b;
        System.out.println("String a = this is == String b = this: " + result);

        String c = new String("is");
        String d = new String("is");
        result = c == d;
        System.out.println("String c = new String(is) is == String d = new String(is):

        String e = new String("a").intern();
        String f = new String("a").intern();
        result = e == f;
        System.out.println("String e = new String(a).intern() is == String f = new Stri

        String g = new String("test");
        String h = new String("test");
        result = g.equals(h);
        System.out.println("String g = new String(test) is == String h = new String(tes
    }

    public static void main(String[] args) {
        stringEqual();
    }
```

先看下运行结果：

```
String a = this is == String b = this: true
String c = new String(is) is == String d = new String(is): false
String e = new String(a).intern() is == String f = new String(a).intern(): true
String g = new String(test) is == String h = new String(test): true

Process finished with exit code 0
```

String类的设计，也借鉴了Integer的cache缓存，java的设计之初就是为了节省内存的，所以String对象也是有常量池的。

解释一下刚才几个例子的运行结果：

- 通过String="xxx"常量赋值的方式，jvm会检查当前常量池里有没有这个字符"xxx"，如果没有，会新建一个对象，放到常量池里，如果已经存在就会返回常量池里的对象。所以这种赋值的方式，返回的是常量池的同一个对象，所以返回的结果就是true。
- 通过new对象的方式，是在jvm堆上重新建立一个对象，所以返回的对象地址是不同的，这也就解释了这个例子返回的是false。
- 通过String.intern()这种方式，是强制将对象放到字符串常量池里，所以通过这种方式，对于同一个字符串，返回的结果一定是true。可以参见另一个链接：java基础面试题-String深入理解，但这里要注意string.intern()也不能滥用，因为字符串常量表是用map来维护的，而且map是有固定容量的，所以对象如果太多的话，map中每个index下的值会退化成一个比较长的链表，查询效率大大下降。所以使用intern的字符串对象太多的话，效率反而不高。
- 通过equals方式的比较的是对象的值，而非两个对象是同一个对象，是正确的对象比较方式，所以这个例子会返回true。

> 写在后面
> 关于equals到这里还没有完，java是如何判断两个对象的equals，set是怎么去重的？砥砺前行，永不停止，我们下篇见！
> 坚持和习惯是学习中的两大绊脚石，先从养成点赞和评论开始吧，😁!