Java 基础提升篇：equals()方法和“==”运算符

equals()

超类 Object 中有这个 equals()方法，该方法主要用于比较两个对象是否相等。该方法的源码如下：

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

我们知道所有的对象都拥有标识(内存地址)和状态(数据)，同时“==”比较两个对象的的内存地址，所以说使用 Object 的 equals()方法是比较两个对象的内存地址是否相等，即若 object1.equals(object2)为 true，则表示 equals1 和 equals2 实际上是引用同一个对象。虽然有时候 Object 的 equals()方法可以满足我们一些基本的要求，但是我们必须要清楚我们很大部分时间都是进行两个对象的比较，这个时候 Object 的 equals()方法就不可以了，实际上 JDK 中，String、Math 等封装类都对 equals()方法进行了重写。下面是 String 的 equals()方法：

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    if (anObject instanceof String) {
        String anotherString = (String)anObject;
        int n = count;
        if (n == anotherString.count) {
        char v1[] = value;
        char v2[] = anotherString.value;
        int i = offset;
        int j = anotherString.offset;
        while (n-- != 0) {
            if (v1[i++] != v2[j++])
            return false;
```

```
        }
        return true;
        }
    }
    return false;
}
```

对于这个代码段:if (v1[i++] != v2[j++])return false;我们可以非常清晰的看到 String 的 equals()方法是进行内容比较，而不是引用比较。至于其他的封装类都差不多。

equals()方法和“==”运算符比较

首先笼统的来讲“java 中 equals()方法和“==”运算符”都是比较的地址，那为什么我们在使用中总会出现混淆的情况呢老是弄错呢，这是因为“重写 equals（）方法”和一些“特殊情况”的存在。

有两种用法说明：

（1）对于字符串变量来说，使用“==”和“equals()”方法比较字符串时，其比较方法不同。

- “==”比较两个变量本身的值，即两个对象在内存中的首地址。
- “equals()”比较字符串中所包含的内容是否相同。

比如：

```
String s1,s2,s3 = "abc", s4 ="abc" ;
s1 = new String("abc");
s2 = new String("abc");
```

结果：

```
s1==s2 是 false    //两个变量的内存地址不一样，也就是说它们指向的对象不 一样，故不相等。
s1.equals(s2) 是 true     //两个变量的所包含的内容是 abc，故相等。
```

注意（1）：

如果：

```
StringBuffer s1 = new StringBuffer("a");
StringBuffer s2 = new StringBuffer("a");
```

结果：

```
s1.equals(s2) //是 false
```

解释：

StringBuffer 类中没有重新定义 equals 这个方法，因此这个方法就来自 Object 类，而 Object 类中的 equals 方法是用来比较“地址”的，所以等于 false.

注意（2）：

对于 s3 和 s4 来说，有一点不一样要引起注意，由于 s3 和 s4 是两个字符串常量所生成的变量，其中所存放的内存地址是相等的，所以 s3==s4 是 true（即使没有 s3=s4 这样一个赋值语句）

（2）对于非字符串变量来说，"=="和"equals"方法的作用是相同的都是用来比较其对象在堆内存的首地址，即用来比较两个引用变量是否指向同一个对象。

比如：

```
class A
{
      A obj1   =    new A();
      A obj2   = new A();
}
```

结果：

```
obj1==obj2 是 false
obj1.equals(obj2)是 false
```

但是如加上这样一句：obj1=obj2;

结果 ：

```
obj1==obj2 是 true
obj1.equals(obj2) 是 true
```

总结

equals 方法对于字符串来说是比较内容的，而对于非字符串来说是比较其指向的对象是否相同的。

== 比较符也是比较指向的对象是否相同的也就是对象在对内存中的的首地址。

String 类中重新定义了 equals 这个方法，而且比较的是值，而不是地址。所以是 true。

注意：

- 如果是基本类型比较，那么只能用==来比较，不能用 equals

```
public class TestEquals {
public static void main(String[] args)
{
int a = 3;
int b = 4;
 int c = 3;
 System.out.println(a == b);//结果是 false
 System.out.println(a == c);//结果是 true
```

```
 System.out.println(a.equals(c));//错误，编译不能通过，equals 方法
//不能运用与基本类型的比较
}
}
```

- 对于基本类型的包装类型 ,比如 Boolean、Character、Byte、Shot、Integer、Long、Float、Double 等的引用变量，==是比较地址的，而 equals 是比较内容的。

```
public class TestEquals {
 public static void main(String[] args)
 { Integer n1 = new Integer(30);
 Integer n2 = new Integer(30);
 Integer n3 = new Integer(31);
 System.out.println(n1 == n2);//结果是 false 两个不同的 Integer 对象，故其地址不同，
 System.out.println(n1 == n3);//那么不管是 new Integer(30)还是 new Integer(31) 结果都显示 false
 System.out.println(n1.equals(n2));//结果是 true 根据 jdk 文档中的说明，n1 与 n2 指向的对象中的内容是相等的，都是 30，故 equals 比较后结果是 true
 System.out.println(n1.equals(n3));//结果是 false 因对象内容不一样，一个是 30 一个是 31
}
}
```

这是 Integer 的实例，如果是其他的比如 Double、Character、Float 等也一样。