常见面试题分析

== 和 equals 的区别是什么？

> 命题：*java*中其实只有 == 比较，没有*equals*比较，其实底层还是 ==。只不过*equals*是很多类定义个方法而已，而这个方法把内存地址的比较改成基础数据类型的比较。
>
> - 就算*String ,Integer,Long* 等都回归到基础数据类型的比较。

1、==

java中的数据类型，可分为两类：

1.基本数据类型，也称原始数据类型

`byte,short,char,int,long,float,double,boolean`　他们之间的比较，应用双等号（==）,比较的是他们的值。

- 在基础数据类型，就是字面值的比较，和数据类型无关，只要相同就是true
- 在封装数据类型中，它内存地址比较。在java中Short，Integer,Long都Byte范围的长度进来缓存，只要在-128-127都是相等的，其他的都是内存地址比较都是false

equals方式是来自Object方法，

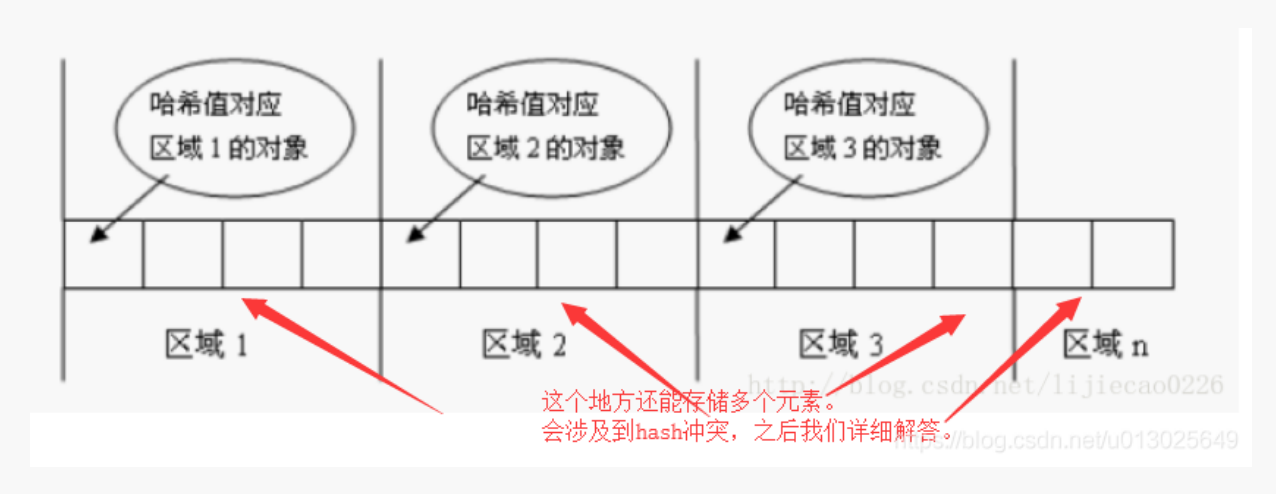
- 如果我们定义一个类，如果没有覆盖equals方法的话，全部都是内存地址比较。

```
Parent parent1 = new parent(1,2);
Parent parent2 = new parent(1,2);
parent1.equals(parent2);==相当于  parent1==parent2 == false
```

- 因为 String，Integer，Long、Double等是它们都重写了Object的equals方法，把equals方法中的内存地址比较改成了基础数据类型的字面值比较。根据基础数据类型的比较规则，只要字面值相同就是true。

.引用类型(类、接口、数组)

```
public class testDay {
    public static void main(String[] args) {
        String s1 = new String("11");
        String s2 = new String("11");
        System.out.println(s1 == s2);
        System.out.println(s1.equals(s2));
    }
}
结果是：false  true
```



s1和s2都分别存储的是相应对象的地址。所以如果用 s1== s2时，比较的是两个对象的地址值（即比较引用是否相同），为false。而调用equals方向的时候比较的是相应地址里面的值，所以值为true。这里就需要详细描述下equals()了。

2、equals()方法详解

equals()方法是用来判断其他的对象是否和该对象相等。其再Object里面就有定义，所以任何一个对象都有equals()方法。区别在于是否重写了该方法。

先看下源码：

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

很明显Object定义的是对两个对象的地址值进行的比较（即比较引用是否相同）。但是为什么String里面调用equals()却是比较的不是地址而是堆内存地址里面的值呢。这里就是个重点了，像String 、Math、Integer、Double等这些封装类在使用equals()方法时，已经覆盖了object类的equals()方法。看下String里面重写的equals()：

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

重写了之后就是这是进行的内容比较，而已经不再是之前地址的比较。依次类推Math、Integer、Double等这些类都是重写了equals()方法的，从而都会回归到基础数据类型的字面值比较。

需要注意的是当equals()方法被override时，hashCode()也要被override。按照一般hashCode()方法的实现来说，相等的对象，它们的hashcode一定相等。为什么会这样，这里又要简单提一下hashcode了。

3. 两个对象的 hashCode() 相同， 那么 equals() 也一定为 true 吗？

不对，两个对象的 hashCode() 相同，equals() 不一定 true。

代码示例：

```
String str1 = "keep";
String str2 = "brother";
System. out. println(String. format("str1：%d | str2：%d",  str1.
hashCode(),str2. hashCode()));
System. out. println(str1. equals(str2));
```

执行的结果：

```
str1：1179395 | str2：1179395

false
```

代码解读：很显然“keep”和“brother”的 hashCode() 相同，然而 equals() 则为 false，因为在散列表中，hashCode() 相等即两个键值对的哈希值相等，然而哈希值相等，并不一定能得出键值对相等。

3、hashcode()浅谈

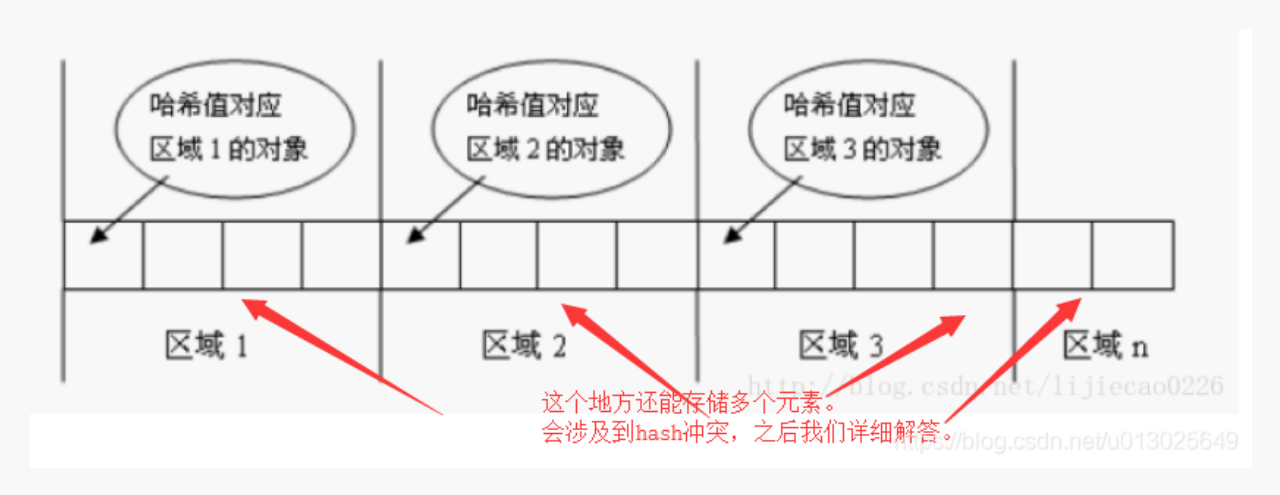
Hashcode覆盖的作用其实：未来这个对象要添加到java中数据结构比如：链表，map中，这个时候提供的数字。比如

明明是java中和equals和hashCode的区别问题，怎么一下子又扯到hashcode()上面去了。你肯定很郁闷，好了，我打个简单的例子你就知道为什么或者equals的时候会涉及到hashCode。

举例说明下：如果你想查找一个集合中是否包含某个对象，那么程序应该怎么写呢？不要用indexOf方法的话，就是从集合中去遍历然后比较是否想到。万一集合中有10000个元素呢，累屎了是吧。所以为了提高效率，哈希算法也就产生了。核心思想就是将集合分成若干个存储区域（可以看成一个个桶），每个对象可以计算出一个哈希码，可以根据哈希码分组，每组分别对应某个存储区域，这样一个对象根据它的哈希码就可以分到不同的存储区域（不同的区域）。

所以再比较元素的时候，实际上是先比较hashcode，如果相等了之后才去比较equal方法。

看下hashcode图解：



一个对象一般有key和value，可以根据key来计算它的hashCode值，再根据其hashCode值存储在不同的存储区域中，如上图。不同区域能存储多个值是因为会涉及到hash冲突的问题。简单如果两个不同对象的hashCode相同，这种现象称为hash冲突。简单来说就是hashCode相同但是equals不同的值。对于比较10000个元素就不需要遍历整个集合了，只需要计算要查找对象的key的hashCode，然后找到该hashCode对应的存储区域查找就over了。

所以判断相等的流程如图所示：

大概可以知道，先通过hashcode来比较，如果hashcode相等，那么就用equals方法来比较两个对象是否相等。再重写了equals最好把hashCode也重写。其实这是一条规范，如果不这样做程序也可以执行，只不过会隐藏bug。一般一个类的对象如果会存储在HashTable，HashSet,HashMap等散列存储结构中，那么重写equals后最好也重写hashCode。

总结：

1.hashCode是为了提高在散列结构存储中查找的效率，在线性表中没有作用。
2.equals重写的时候hashCode也跟着重写
3.两对象equals如果相等那么hashCode也一定相等，反之不一定。

5. Java 中的 Math. round(-1. 5) 等于多少？

Math.round(-1.5)的返回值是-1。四舍五入的原理是在参数上加0.5然后做向下取整。

我们可以通过大量实验看下结果

```
public class test {
    public static void main(String[] args){
        System.out.println(Math.round(1.3));   //1
        System.out.println(Math.round(1.4));   //1
        System.out.println(Math.round(1.5));   //2
        System.out.println(Math.round(1.6));   //2
        System.out.println(Math.round(1.7));   //2
        System.out.println(Math.round(-1.3));  //-1
        System.out.println(Math.round(-1.4));  //-1

        System.out.println(Math.round(-1.5));  //-1 --------这里要注意

        System.out.println(Math.round(-1.6));  //-2
        System.out.println(Math.round(-1.7));  //-2
    }
}
```

6. String 属于基础的数据类型吗？

String是final修饰的java类，java中的基本类型一共有8个，它们分别为：

- 字符类型：byte，char
- 基本整型：short，int，long
- 浮点型：float，double
- 布尔类型：boolean

此外需要说明 有的文章中吧void也算是一种基本的数据类型

7. Java 中操作字符串都有哪些类？它们之间有什么区别？

操作字符串的类有：String、StringBuffer、StringBuilder。

String 和 StringBuffer、StringBuilder 的区别在于 String 声明的是不可变的对象，每次操作都会生成新的 String 对象，然后将指针指向新的 String 对象，而 StringBuffer、StringBuilder 可以在原有对象的基础上进行操作，所以在经常改变字符串内容的情况下最好不要使用 String。

StringBuffer 和 StringBuilder 最大的区别在于，StringBuffer 是线程安全的，而 StringBuilder 是非线程安全的，但 StringBuilder 的性能却高于 StringBuffer，所以在单线程环境下推荐使用 StringBuilder，多线程环境下推荐使用 StringBuffer。

8. String str="i"与 String str=new String("i")一样吗？

不一样，因为内存的分配方式不一样。String str="i"的方式，Java 虚拟机会将其分配到常量池中；而 String str=new String("i") 则会被分到堆内存中。

代码示例：

```
String x = "叶痕秋";
String y = "叶痕秋";
String z = new String("叶痕秋");
System.out.println(x == y); // true
System.out.println(x == z); // false
```

String x = "叶痕秋" 的方式，Java 虚拟机会将其分配到常量池中，而常量池中没有重复的元素，比如当执行“叶痕秋”时，java虚拟机会先在常量池中检索是否已经有“叶痕秋”,如果有那么就将“叶痕秋”的地址赋给变量，如果没有就创建一个，然后在赋给变量；而 String z = new String(“叶痕秋”) 则会被分到堆内存中，即使内容一样还是会创建新的对象。

11. 抽象类必须要有抽象方法吗？

不需要，抽象类不一定非要有抽象方法。

示例代码：

```
abstract class Cat {
    public static void sayHi() {
        System. out. println("hi~");
    }
}
```

上面代码，抽象类并没有抽象方法但完全可以正常运行。

12. 普通类和抽象类有哪些区别？

- 普通类不能包含抽象方法，抽象类可以包含抽象方法。
- 抽象类不能直接实例化，普通类可以直接实例化。

13. 抽象类能使用 **final** 修饰吗？

不能，定义抽象类就是让其他类继承的，如果定义为 final 该类就不能被继承，这样彼此就会产生矛盾，所以 final 不能修饰抽象类，如下图所示，编辑器也会提示错误信息：

14. 接口和抽象类有什么区别？

- 实现：抽象类的子类使用 extends 来继承；接口必须使用 implements 来实现接口。
- 构造函数：抽象类可以有构造函数；接口不能有。
- 实现数量：类可以实现很多个接口；但是只能继承一个抽象类。
- 访问修饰符：接口中的方法默认使用 public 修饰；抽象类中的方法可以是任意访问修饰符。

比较点	抽象类	接口
默认方法	抽象类可以有默认的方法实现	java 8之前,接口中不存在方法的实现
实现方式	子类使用extends关键字来继承抽象类.如果子类不是抽象类,子类需要提供抽象类中所声明方法的实现	子类使用implements来实现接口,需要提供接口中所有声明的实现.
构造器	抽象类中可以有构造器	接口中不能
和正常类区别	抽象类不能被实例化	接口则是完全不同的类型
访问修饰符	抽象方法可以有public,protected和default等修饰	接口默认是public,不能使用其他修饰符
多继承	一个子类只能存在一个父类	一个子类可以存在多个接口
添加新方法	抽象类中添加新方法,可以提供默认的实现,因此可以不修改子类现有的代码	如果往接口中添加新方法,则子类中需要实现该方法

静态变量和实例变量的区别?

静态变量存储在方法区,属于类所有.实例变量存储在堆当中,其引用存在当前线程栈.需要注意的是从JDK1.8开始用于实现方法区的PermSpace被MetaSpace取代了.

Object中有哪些公共方法?

```
equals()`,`clone()`,`getClass()`,`notify(),notifyAll(),wait()`,`toStri
ng
```

java 创建对象的几种方式

java中提供了以下四种创建对象的方式:

- new创建新对象
- 通过反射机制
- 采用clone机制
- 通过序列化机制