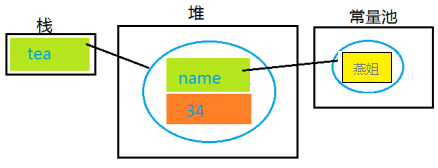
**你真的了解Java中的数组吗？**

数组是基本上所有语言都会有的一种数据类型，它表示一组相同类型的数据的集合，具有固定的长度，并且在内存中占据连续的空间。在C，C++等语言中，数组的定义简洁清晰，而在Java中确有一些会让人迷惑的特性，往往让初学者发蒙。本文就尝试分析这些特性。

**问题一：Java中的数组是对象吗？**

要判断数组是不是对象，那么首先明确什么是对象？在较高的层面上，对象是根据某个类创建出来的一个实例，表示某类事物中一个具体的个体。对象具有各种属性，并且具有一些特定的行为。而在较低的层面上，站在计算机的角度，对象就是堆内存中的一个区域，在这个内存块中封装了一些数据（即类中定义的各个属性值），所以，对象是用来封装数据的。以下为一个Teacher对象在内存中的表示：Teacher tea = new Teacher(“燕姐”,34);



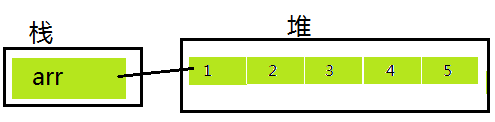
注意：

1）矩形表示一个引用（地址）或一个基本类型的数据，椭圆表示一个对象，多个矩形组合在一块，可组成一个对象。

2）name在对象中只表示一个引用， 也就是一个地址值，它指向一个真实存在的字符串对象。在这里严格区分了引用和对象。

那么在Java中，数组满足以上的条件吗？

在较高的层面上，数组不是某类事物中的一个具体的个体，而是多个个体的集合。那么它应该不是对象。而在计算机的角度，数组也是一个内存块，也封装了一些数据，这样的话也可以称之为对象。以下是一个数组在内存中的表示：int[] arr = {1,2,3,4,5};



这样的话， 数组既可以是对象， 也可以不是对象。至于到底是不是把数组当做对象，全凭Java的设计者决定。

|  |
| --- |
| int[] arr = {1,2,3,4,5};  int len = arr.length; //数组中保存一个字段, 表示数组的长度    //以下方法说明数组可以调用方法,java中的数组是对象.  //这些方法是Object中的方法,所以可以肯定,数组的最顶层父类也是Object  String str = arr.toString();  int[] clone = arr.clone(); |

在数组arr上， 可以访问它的属性，也可以调用一些方法。这基本上可以认定，java中的数组也是对象，它具有java中其他对象的一些基本特点：封装了一些数据，可以访问属性，也可以调用方法。**所以，在Java中数组是对象。**

而在C++中，数组虽然封装了数据，但数组名只是一个指针，指向数组中的首个元素，既没有属性，也没有方法可以调用。如下代码所示：

|  |
| --- |
| int main(){  int arr[] = {1, 2, 3, 4};  int\* pa = arr;  //无法访问属性，也不能调用方法。  return 0;  } |

**所以C++中的数组不是对象**，只是一个数据的集合，而不能当做对象来使用。

**问题二：Java中数组的类型**

Java是一种强类型的语言。既然是对象， 那么就必须属于一个类型，比如根据Person类创建一个对象，这个对象的类型就是Teacher。那么数组的类型是什么呢？看下面的代码：

|  |
| --- |
| int[] arr = {1, 2, 3, 4, 5};  System.out.println(arr.getClass().getName());  //打印出的数组类的名字为[I    String[] s = new String[2];  System.out.println(s.getClass().getName());  //打印出的数组类的名字为 [Ljava.lang.String;    String[][] ss = new String[2][3];  System.out.println(ss.getClass().getName());  //打印出的数组类的名字为 [[Ljava.lang.String; |

所以，**数组也是有类型的。**只是这个类型显得比较奇怪。而且我们没有自己创建这个类，也没有在Java的标准库中找到这个类。

|  |
| --- |
| public class int[] {  // ...  } |

这只能有一个解释，那就是虚拟机自动创建了数组类型，可以把数组类型和8种基本数据类型一样， 当做java的内建类型。这种类型的命名规则是这样的：

* 每一维度用一个[表示；开头两个[，就代表是二维数组。
* [后面是数组中元素的类型(包括基本数据类型和引用数据类型)

在java语言层面上,s是数组,也是一个对象,那么他的类型应该是String[]，这样说是合理的。但是在JVM中，他的类型为[java.lang.String。顺便说一句普通的类在JVM里的类型为“包名.类名”，也就是全限定名。同一个类型在java语言中和在虚拟机中的表示可能是不一样的。

**问题三：Java中数组的继承关系**

上面已经验证了，数组是对象，也就是说可以以操作对象的方式来操作数组。并且数组在虚拟机中有它特别的类型。既然是对象，遵循Java语言中的规则：Object是上帝， 也就是说所有类的顶层父类都是Object。数组的顶层父类也必须是Object，这就说明数组对象可以向上直接转型到Object，也可以向下强制类型转换，也可以使用instanceof关键字做类型判定。 这一切都和普通对象一样。如下代码所示：

|  |
| --- |
| //1 数组的顶层父类是Object, 所以可以向上转型  int[] a = {1, 2, 3, 4, 5};  Object obj = a ;    //2 那么能向下转型吗? 可以  int[] b = (int[])obj;    //3 能使用instanceof关键字判定吗? 可以  if(obj instanceof int[]){  System.out.println("obj的真实类型是int[]");  } |

**问题四：Java中数组的另一种“继承”关系**

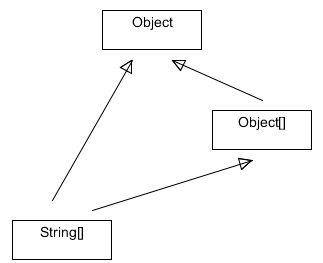
如下代码是正确的，却很容易让我们疑惑：

|  |
| --- |
| String[] s = new String[5];  Object[] os = s; //成立,说明可以用Object[]的引用来接收String[]的对象  String[] other = (String[]) os;//也可以向下转型 |

Object[]类型的引用可以指向String[]类型的数组对象？难道说String[]继承自Object[]，而Object[]又继承自Object? 让我们通过反射的方式来验证这个问题：

|  |
| --- |
| String[] s = new String[5];  System.out.println(s.getClass().getSuperclass().getName()); |

打印结果为java.lang.Object,说明String[] 的直接父类是 Object而不是Object[]。由代码可知，String[]的直接父类就是Object而不是Object[]。可是Object[]的引用明明可以指向String[]类型的对象。那么他们的继承关系有点像这样：



这样的话就违背了Java单继承的原则。String[]不可能即继承Object，又继承Object[]。上面的类图肯定是错误的。那么只能这样解释：数组类直接继承了Object，**关于Object[]类型的引用能够指向String[]类型的对象，这种情况只能是Java语法之中的一个特例，并不是严格意义上的继承。也就是说，String[]不继承自Object[]，但是我可以允许你向上转型到Object[]，这种特性是赋予你的一项特权。**

其实这种关系可以这样表述：如果有两个类A和B，如果B继承（extends）了A，那么A[]类型的引用就可以指向B[]类型的对象

|  |
| --- |
| public static class Father {    }    public static class Son extends Father {    }  //下面成立吗? Father是Son的直接父类  Son[] sons = new Son[3];  Father[] fa = sons; //成立    //那么Son[] 的直接父类是Father[] 还是Object[] 或者是Object?  System.out.println(sons.getClass().getSuperclass().getName());  //打印结果为java.lang.Object,说明Son[]的直接父类是Object |

上面的结论可以扩展到二维数组和多维数组：

|  |
| --- |
| Son[][] sonss = new Son[2][4];  Father[][] fathers = sonss; |

上面的代码可以这样理解：将Father[][]数组看成是一维数组, 这是个数组中的元素为Father[]，将Son[][]数组看成是一维数组, 这是个数组中的元素为Son[]，因为Father[]类型的引用可以指向Son[]类型的对象，所以，根据上面的结论，Father[][]的引用可以指向Son[][]类型的对象。

但是，数组的这种用法不能作用于基本类型数据：

|  |
| --- |
| int[] arr = {1,2,3,4,5};  Object[] objaa = arr; //错误的，不能通过编译 |

这是错误的, 因为int不是引用类型，Object不是int的父类

Java为什么会为数组提供这样一种语法特性呢？也就是说这种语法有什么作用？编写过Android中Sqlite数据库操作程序的同学可能发现过这种现象，用一个Object[]引用接收所有的数组对象，在编译SQL语句时，为SQL语句中的占位符提供对应的值。

|  |
| --- |
| db.execSQL("INSERT INTO person VALUES (NULL, ?, ?)", new Object[]{person.name, person.age}); |

所以这种特性主要是用于方法中参数的传递。如果不传递数组，而是依次传递各个值，会使方法参数列表变得冗长。如果使用具体的数组类型，如String[]，那么就限定了类型，失去了灵活性。所以传递数组类型是一种比较好的方式。但是如果没有上面的数组特性（即如果有两个类A和B，如果B继承（extends）了A，那么A[]类型的引用就可以指向B[]类型的对象），那么数组类型就只能通过Object类型接收，这样就无法在方法内部访问或遍历数组中的各个元素。如下代码：

|  |
| --- |
| private static void test3() {  String[] a = new String[3];  doArray(a);  }    private static void doArray(Object[] objs){    }    private static void doArray1(Object obj){  //不能用Object接收数组，因为这样无法对数组的元素进行访问  // obj[1] //错误    //如果在方法内部对obj转型到数组，存在类型转换异常的风险  // Object[] objs = (Object[]) obj;  }    private static void doArray2(String[] strs){  //如果适用特定类型的数组，就限制了类型，失去灵活性和通用性  }    private static void doArray3(String name, int age, String id, float account){  //如果不适用数组而是依次传递参数，会使参数列表变得冗长，难以阅读  } |

到此为止，数组的特性就总结完了。你明白了吗？