[

Java千百问_06数据结构（005）_数值中为什么会出现下划线

,

点击进入_更多_Java千百问

1、数值中为什么会出现下划线

这是Jdk 7之后的一个特性。
从Jdk 7开始，可以在数值型字面值（包括整型字面值和浮点字面值）插入一个或者多个下划线。例如：

```
int x = 123_456_789;
```

在编译的时候，下划线会自动去掉。但是下划线只能用于分隔数字，不能分隔字符与字符，也不能分隔字符与数字。

可以连续使用下划线，例如：

```
float f = 1.22___33__44;
```

二进制或者十六进制的字面值也可以使用下划线。
记住一点，下划线只能用于数字与数字之间，初次以外都是非法的。例如：1._23是非法的，_123、11000_L也都是非法的。

了解java8种基本数据类型：java有哪8种基本数据类型

]

[

Java千百问_06数据结构（006）_java基本数据类型如何转换

,

点击进入_更多_Java千百问

1、基本类型如何相互转换

我们看到，将一种类型的值赋给另一种类型是很常见的。在Java中，boolean类型与所有其他7种类型都不能进行转换，这一点很明确。
对于其他7中数值类型，它们之间都可以进行转换，但是可能会存在精度损失或者其他一些变化。
java中8种基本数据类型看这里：java有哪8种基本数据类型
转换分为自动转换和强制转换。对于自动转换（隐式），无需任何操作，而强制类型转换需要显式转换，即使用转换操作符（type）。

首先将7种类型按下面顺序排列一下：

byte < （short=char） < int < long < float < double

如果从小转换到大，可以自动完成，而从大到小，必须强制转换。short和char两种相同类型也必须强制转换。

2、如何自动转换类型

自动转换时会发生扩宽 （widening conversion）。因为较大的类型（如int）要保存较小的类型（如byte），内存总是足够的，不需要强制转换。
如果将字面值保存到byte、 short、char、long的时候，也会自动进行类型转换。
注意区别，此时从int（没有带L的整型字面值为int）到byte/short /char也是自动完成的，虽然它们都比int小。在自动类型转化中，除了以下几种情况可能会导致精度损失以外，其他的转换都不能出现精度损失。
1. int –> float
2. long –> float
3. long –> double
4. float –> double without strictfp （无符号double）
除了可能的精度损失外，自动转换不会出现任何运行时（run-time）异常。

3、如何强制转换类型

如果要把大的转成小的， 或者在short与char之间进行转换，就必须强制转换，也被称作缩小转换（narrowing conversion）。
因为必须显式地使数值更小以适应目标类型。强制转换采用转换操作符（type）。严格地说，将byte转为char不属于缩小转换，因为从byte到char的过程其实是byte–>int–>char，所以widening和narrowing都有。

强制转换除了可能的精度损失外，还可能使模（overall magnitude）发生变化。强制转换格式如下：

```
target-type val = (target-type) value;
```

例如：

```
int a=80;
byte b;
b = (byte)a;
```

但你，如果整数的值超出了byte所能表示的范围，结果将对byte类型的范围取余数。例如：

```
int a=257;
byte b;
b = (byte)a;
System.out.println(b);
```

结果如下：
1

因为a=257超出了byte的[-128,127]的范围，所以将257除以byte的范围（256）取余数得到：b=1。需要注意的是，当a=200时，此时除了256取余数应该为-56，而不是200。

将浮点类型赋给整数类型的时候，会发生截尾（truncation）。也就是把小数的部分去掉，只留下整数部分。此时如果整数超出目标类型范围，一样将对目标类型的范围取余数。

]

[

Java千百问_06数据结构（007）_String属于基本数据类型吗

,

点击进入_更多_Java千百问

1、String属于基本数据类型吗

首先要明确的是，在Java中，String不是基本数据类型，它继承于Object，是一个jdk提供的字符串类。
了解基本数据类型看这里：java有哪8种基本数据类型2
但是，String和其他对象相比，jdk做了很多特殊的处理。体现在如下几个方面：

- String可以通过new构造对象，也可以直接赋值。例如：

```
String str = new String( "abc" );//第一种
String str = "abc" ;//第二种
```

- 第一种用new()来新建对象，它会在存放于堆中，每调用一次就会创建一个新的对象。
  第二种是先在栈中创建一个对String类的对象引用变量str，然后查找栈中有没有存放”abc”，如果没有，则将”abc”存放进栈，并令str指 向”abc”，如果已经有”abc” 则直接令str指向”abc”。
  了解java内存堆栈看这里：

- 第二种方式（String str1 =”abc”）创建多个”abc”字符串，在内存中其实只存在一个对象而已。这种方式节省内存空间，同时它可以在一定程度上提高程序的运行速度，因为JVM会自动根据栈中数据的实际情况来决定是否有必要创建新对象。
  而对于String str = new String(“abc”)的代码，则一概在堆中创建新对象，而不管其字符串值是否相等，是否有必要创建新对象，从而加重了程序的负担。

- 比较类里面的数值是否相等时，用equals()方法；当比较两个引用变量是否指向同一个对象时，用==（可以理解为比较逻辑地址，实际是比较对象号）。 例如：

```
String str1 = "abc" ;
String str2 = "abc" ;
System.out.println(str1==str2);
```

运行结果：
true

可以看出str1和str2是栈中对象。

```
String str1 = new String ( "abc" );
String str2 = new String ( "abc" );
System.out.println(str1==str2);
```

运行结果：
false

了解String更多看这里：

]

[

Java千百问_06数据结构（008）_null属于哪种数据类型

,

点击进入_更多_Java千百问

1、null属于哪种数据类型

首先，在java中，null本身不是对象，也不是Objcet的实例。那么他属于哪种数据类型呢？
了解数据类型看这里：java中数据类型是什么
java基本数据类型看这里：[java有哪8种基本数据类型3
具体看下面这个例子：

```
if (null instanceof java.lang.Object) {
    System.out.println("null属于java.lang.Object类型");
} else {
    System.out.println("null不属于java.lang.Object类型");
}
```

结果：
null不属于java.lang.Object类型

结论：null本身虽然能代表一个不确定的对象。但就null本身来说，它不是对象，也不是java.lang.Object的实例。
null是一种特殊的type，但是你不能声明一个变量为null类型，null type的唯一取值就是null。

2、null如何使用

Java中，null是一个关键字，用来标识一个不确定的对象。因此可以将null赋给引用类型变量，但不可以将null赋给基本类型变量。
比如：

```
int a = null;//错误
Ojbect o = null;//正确
```

null可以赋值给任意的类类型或者转化成任意的类类型。在实践中，一般把null当做字面值，这个字面值可以是任意的引用类型。
了解java字面值看这里：java数据类型的字面值是什么
Java中，变量都遵循一个原则，先定义，再初始化，才可以使用。我们不能int a后，不给a指定值，就去获取a的值。这条对于引用变量也是适用的。
例如：

```
        Connection conn = null;
        try {
            conn = DriverManager.getConnection("url", "user", "password");
        } catch (SQLException e) {
            e.printStackTrace();
        }

        String catalog = conn.getCatalog();
```

如果刚开始的时候不指定conn＝null，则会产生编译错误。

]

[

Java千百问_06数据结构（009）_void是什么

,

点击进入_更多_Java千百问

1、void是什么

java中还存在一种特殊的基本类型void。
了解数据类型看这里：java中数据类型是什么
java基本数据类型看这里：java有哪8种基本数据类型
void是指：无类型。在java中void表示方法无返回值。

void也有对应的包装类java.lang.Void，不过我们无法直接对它们进行操作。
它继承于Object，但不能扩展。如下：

```
public final class Void extends Object
```

Void类是一个不可实例化的占位符类，用来来保存一个引用代表了Java关键字void的Class对象。
但是，我们在开发过程中，不推荐使用Void，而是应该使用void。

]

[

Java千百问_06数据结构（010）_Class类型是什么

,

点击进入_更多_Java千百问

1、Class类型是什么

还有一种特殊的数据类型：class，用来表示某个类的类型。即，每一个类类型都是Class类的一个对象。
了解数据类型看这里：java中数据类型是什么
java基本数据类型看这里：java有哪8种基本数据类型
用type name加上.class表示，例如String.class。Class规则下：

1. 首先，String是类Class（java.lang.Class)的一个实例（对象），而”This is a string”是类String的一个对象。
2. 然后，class的字面值用于表示类Class的一个对象，比如String.class用于表示类Class的对象String。
   了解字面值看这里：java数据类型的字面值是什么
   简单地说，类的字面值（class literal）就是诸如String.class 、Integer.class这样的字面值，它所表示的就是类String、类Integer。

如果打印Integer.class，你会得到：

```
class java.lang.Integer;
```

打印List.class，会得到：

```
interface java.util.List。
```

总之，class字面值用于表示类型本身。

]

[

Java千百问_06数据结构（011）_java中的数组是什么

,

点击进入_更多_Java千百问

1、什么是数组

Java提供了一个用于存储相同类型的元素的，固定大小的连续集合数据结构：数组。

数组是用于存储数据的集合，储存相同类型数据的集合。

与单个变量相比（如number0, number1 … number99），数组变量需要使用下标索引来确定数组中某个数据的顺序（如numbers[0], numbers1 … numbers[99] ）。
了解变量看这里：局部变量、类变量、实例变量有什么区别2

2、如何声明数组

一个程序要使用数组，必须声明一个变量来引用数组，而且需要指定数组变量的引用类型。语法如下：

```
dataType[] arrayRefVar;   // preferred way.

dataType arrayRefVar[];  //  works but not preferred way.
```

注：dataType[] arrayRefVar这种写法是首选的。dataType arrayRefVar[]这种写法来自于C/C++语言。

数据中的数据类型可以是基本数据类型、类类型、自定义类类型等。
了解数据类型看这里：java中数据类型是什么
java基本数据类型看这里：java有哪8种基本数据类型
具体例子:

```
double[] myList;
double myList[];

MyClass[] myList;
MyClass myList[];
```

3、如何创建数组

可以通过new关键字创建一个数组，语法如下：

```
arrayRefVar = new dataType[arraySize];
```

上面的语句做了两件事：
1. 创建了一个数组: new dataType[arraySize];
2. 将新创建的数组变量分配至arrayRefVar变量。

声明数组变量，建立数组，并分配变量可以在一个语句中被组合，如下所示：

```
dataType[] arrayRefVar = new dataType[arraySize];
```

另外，可以直接通过数组的值来创建数组，如下所示：

```
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

这里的value0、value1对应的下标索引从0开始顺序排列，第一个值对应0下标，第二个值对应1下标。

4、如何访问数组

数组元素通过下标索引(int类型)访问。数组的下标索引是从0开始的。也就是说，它们从0开始到arrayRefVar.length-1。
例子：
下面的语句声明一个数组变量myList，创建double类型10个元素的数组，并把它的引用到 myList：

```
double[] myList = new double[10];
```

在这里，myList有10个double值，索引是从0到9。

访问数组中某一个值，可以通过变量加下标索引的方式，如下：

```
double a0 = myList[0];

int i = 2;
double a2 = myList[i];
```

]

[

Java千百问_06数据结构（012）_如何遍历数组

,

点击进入_更多_Java千百问

1、如何遍历数组

我们在处理数组时，经常使用for循环或foreach循环进行遍历，因为数组中的所有元素类型相同并且数组的大小是已知的。
了解什么是数组看这里：java中的数组是什么
了解for循环看这里：java中如何循环执行

使用for循环遍历

```
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (int i = 0; i < myList.length; i++) {
         System.out.println(myList[i] + " ");
      }
      // Summing all elements
      double total = 0;
      for (int i = 0; i < myList.length; i++) {
         total += myList[i];
      }
      System.out.println("Total is " + total);
      // Finding the largest element
      double max = myList[0];
      for (int i = 1; i < myList.length; i++) {
         if (myList[i] > max) max = myList[i];
      }
      System.out.println("Max is " + max);
   }
}
```

这将产生以下结果:
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5

使用foreach循环遍历

JDK 1.5引入了一个新的for循环被称为foreach循环或增强的for循环，它无需使用索引变量就能按顺序遍历数组。

```
public class TestArray {
```

```
   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (double element: myList) {
         System.out.println(element);
      }
   }
}
```

这将产生以下结果:

1.9
2.9
3.4
3.5

]

[

Java千百问_06数据结构（014）_java数组如何存储在内存中

,

点击进入_更多_Java千百问

1、数组的内存空间是何时分配的

java中的数组是用来存储同一种数据类型的数据结构，一旦初始化完成，即所占的空间就已固定下来，初始化的过程就是分配对应内存空间的过程。即使某个元素被清空，但其所在空间仍然保留，因此数组长度将不能被改变。
了解什么是数组看这里：java中的数组是什么
当仅定义一个数组变量(int[] numbers)时，该变量还未指向任何有效的内存，因此不能指定数组的长度，只有对数组进行初始化(为数组元素分配内存空间)后才可以使用。
数组初始化分为静态初始化(在定义时就指定数组元素的值，此时不能指定数组长度)和动态初始化(只指定数组长度，由系统分配初始值)。

```
//静态初始化
int[] numbers = new int[] { 3, 5, 12, 8, 7 };
String[] names = { "Miracle", "Miracle He" };//使用静态初始化的简化形式
//动态初始化
int[] numbers = new int[5];
String[] names = new String[2];
```

建议不要混用静态初始化和动态初始化，即不要既指定数组的长度的同时又指定每个元素的值。

当初始化完毕后，就可以按索引位置(0~array.length-1)来访问数组元素了。
当使用动态初始化时，如在对应的索引位未指定值的话，系统将指定相应数据类型对应的默认值(整数为0，浮点数为0.0，字符为'\u0000'，布尔类型为false，引用类型(包括String)为null)。

```
public class TestArray {
    public static void main(String[] args) {
        String[] names = new String[3];
        names[0] = "Miracle";
        names[1] = "Miracle He";
        /*
        for(int i = 0; i < names.length;i++) {
            System.out.print(names[i] + " ");
        }
        */
        //还可以使用foreach来遍历
        for(String name : names) {
            System.out.print(name + " ");
        }
    }
}
```

结果如下：
Miracle Miracle He null
Miracle Miracle He null

2、数组在内存中如何储存

首先给出数组(数组引用和数组元素)在内存中的存放形式，如图：

数组引用变量是存放在栈内存(stack)中，数组元素本质是一个对象，是存放在堆内存(heap)中。通过栈内存中的指针指向对应元素的在堆内存中的位置来实现访问。
了解堆和栈看这里：java堆和栈有什么区别
当数组在初始化时，就会在堆中分配对应的空间，这个大小是不会因为内部元素的变化而变化，也就是说，如果数组中某个元素被清空，数组占用的内存空间也不会缩小。

存放引用类型数组在内存中如何储存看这里：引用类型数组在内存中如何储存

]

[

Java千百问_06数据结构（015）_数组和普通对象的引用变量有什么区别

,

点击进入_更多_Java千百问

1、数组和普通对象的引用变量有什么区别

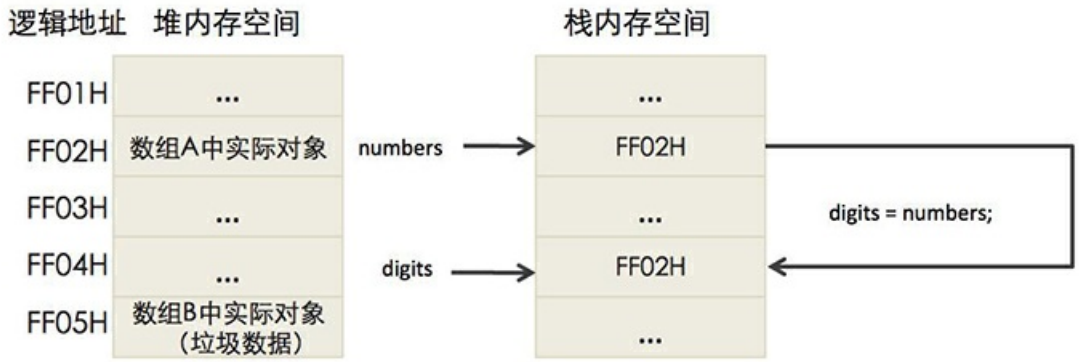
了解什么是数组看这里：java中的数组是什么
对于java的数组来说，只有类型兼容(即属于同一数据类型体系且遵守优先级由低到高原则)，才能将数组引用传递给另一数组引用，但仍然不能改变数组长度(仅仅只是调整数组引用指针的指向)。
了解数组传递看这里：数组如何传递

```
public class TestArrayLength {
    public static void main(String[] args) {
        int[] numbers = { 3, 5, 12 };
        int[] digits = new int[4];
        System.out.println("digits数组长度：" + digits.length);//4
        for(int number : numbers) {
            System.out.print(number + ",");//3,5,12,
        }
        System.out.println("");
        for(int digit : digits) {
            System.out.print(digit + ",");//0,0,0,0,
        }
        System.out.println("");
        digits = numbers;
        System.out.println("digits数组长度：" + digits.length);//3
    }
}
```

执行结果如下：
digits数组长度：4
3,5,12,
0,0,0,0,
digits数组长度：3

数组初始化之后在内存中的存储如下图，在堆中有2个新的数组对象，栈中有2个分别指向这两个数组对象的引用变量：

在执行digits = numbers赋值后，虽然看似digits的数组长度看似由4变成3，其实只是numbers和digits指向同一个数组对象而已。
而digits本身失去引用而变成垃圾，等待垃圾回收来回收(但其长度仍然为4)。如图：
了解java垃圾回收看这里：java垃圾回收机制是什么



看这篇文章就能清楚的明白数组在内存中的存储：java数组如何存储在内存中

]

[

Java千百问_06数据结构（016）_引用类型数组在内存中如何储存

,

点击进入_更多_Java千百问

1、存放基本类型数组在内存中如何储存

java的数组中可以存放引用类型。
存放引用类型的内存分布相比存放基本类型相对复杂。来看一段存储基本类型的程序:
了解什么是数组看这里：java中的数组是什么
了解数组在内存中的储存看这里：java数组如何存储在内存中

```
public class TestPrimitiveArray {
    public static void main(String[] args) {
        //1.定义数组
        int[] numbers;
        //2.分配内存空间
        numbers = new int[4];
        //3.为数组元素指定值
        for(int i = 1;i <= numbers.length;i++) {
            numbers[i] = i ;
        }
    }
}
```

内存分布如图：

从图中可看出数组元素直接存放在堆内存中，当操作数组元素时，实际上是操作基本类型的变量。

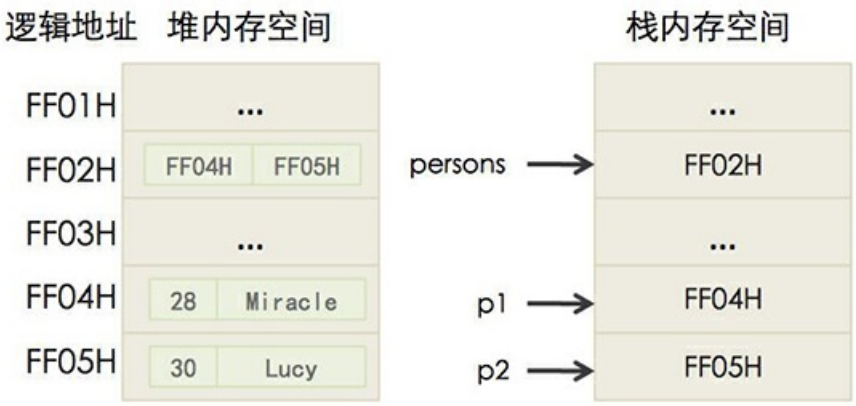
2、存放引用类型数组在内存中如何储存

先来再看一段存储引用类型数组的实例：

```
class Person {
    public int age;
    public String name;
    public void display() {
        System.out.println(name + "的年龄是: " + age);
```

```
    }
}
public class TestReferenceArray {
    public static void main(String[] args) {
        //1.定义数组
        Person[] persons;
        //2.分配内存空间
        persons = new Person[2];
        //3.为数组元素指定值
        Person p1 = new Person();
        p1.age = 28;
        p1.name = "Miracle";
        Person p2 = new Person();
        p2.age = 30;
        p2.name = "Miracle He";
        persons[0] = p1;
        persons[1] = p2;
        //输出元素的值
        for(Person p : persons) {
            p.display();
        }
    }
}
```

元素为引用类型的数组，在内存中的存储与基本类型完全不一样。
此时数组元素存放引用，指向另一块内存，在其中存放有效的数据。如图：



了解数组和普通对象的引用变量：数组和普通对象的引用变量有什么区别

]

[

Java千百问_06数据结构（017）_什么是二维数组

,

点击进入_更多_Java千百问

1、二维数组如何定义

Java语言中，多维数组被看作数组的数组。
了解一维数组看这里：java中的数组是什么
定义方式和一维数组类似，如下：

```
type arrayName[ ][ ];
type [ ][ ]arrayName;
```

2、二维数组如何初始化

二维数组初始化和一维数组一样，分为静态初始化和动态初始化

静态初始化

Java语言中，由于把二维数组看作是数组的数组，数组空间不是连续分配的，所以不要求二维数组每一维的大小相同。初始化方式如下：

```
int intArray[ ][ ]={{1,2},{2,3},{3,4,5}};
```

动态初始化

二维数组可以直接为每一维分配空间，如下：

```
arrayName = new type[arrayLength1][arrayLength2];
int a[ ][ ] = new int[2][3];
```

也可以从最高维开始，分别为每一维分配空间，如下：

```
arrayName = new type[arrayLength1][ ];
arrayName[0] = new type[arrayLength20];
arrayName[1] = new type[arrayLength21];
…
arrayName[arrayLength1-1] = new type[arrayLength2n];
```

例如：

```
int a[ ][ ] = new int[2][ ];
a[0] = new int[3];
a[1] = new int[5];
```

特别的，对二维引用类型的数组，必须首先为最高维分配引用空间，然后再顺次为低维分配空间。而且，必须为每个数组元素单独分配空间。

例如：

```
String s[ ][ ] = new String[2][ ];
s[0]= new String[2];//为最高维分配引用空间
s[1]= new String[2]; //为最高维分配引用空间
s[0][0]= new String("Good");// 为每个数组元素单独分配空间
s[0][1]= new String("Luck");// 为每个数组元素单独分配空间
s[1][0]= new String("to");// 为每个数组元素单独分配空间
s[1][1]= new String("You");// 为每个数组元素单独分配空间
```

3、如何获取二维数组的元素

对二维数组中的每个元素，引用方式为：

```
arrayName[index1][index2]
```

例如：

```
num[1][0];
```

4、二维数组如何使用：

二维数组在数学中，可以认为是一个矩阵，我们来看一个两个矩阵相乘的例子：

```
public
class MatrixMultiply{
public static void main(String args[]){
  int i, j, k;

        int a[][] = new int[2][3]; // 动态初始化一个二维数组
        int b[][] = { { 1, 5, 2, 8 }, { 5, 9, 10, -3 }, { 2, 7, -5, -18 } };// 静态初始化一个二维数组
        int c[][] = new int[2][4]; // 动态初始化一个二维数组
        for (i = 0; i < 2; i++)
            for (j = 0; j < 3; j++)
                a[i][j] = (i + 1) * (j + 2);
        for (i = 0; i < 2; i++) {
            for (j = 0; j < 4; j++) {
                c[i][j] = 0;
                for (k = 0; k < 3; k++)
                    c[i][j] += a[i][k] * b[k][j];
            }
        }
        System.out.println("******* A********");
        for (int[] aa : a) {
            String rowStr = "";
            for (int aaa : aa) {
                rowStr += aaa + "  ";
            }
            System.out.println(rowStr);
        }
        System.out.println("******* B********");
        for (int[] bb : b) {
            String rowStr = "";
            for (int bbb : bb) {
                rowStr += bbb + " ";
            }
            System.out.println(rowStr);
        }
        System.out.println("*******Matrix C********");
        for (i = 0; i < 2; i++) {
            String rowStr = "";
            for (j = 0; j < 4; j++) {
                rowStr += c[i][j] + " ";
            }
            System.out.println(rowStr);
        }
}
}
```

执行结果如下：

* A********
2 3 4
4 6 8
* B********
1 5 2 8
5 9 10 -3
2 7 -5 -18
Matrix C

25 65 14 -65
50 130 28 -130

]

[

Java千百问_06数据结构（018）_多维数组如何储存在内存中

,

点击进入_更多_Java千百问

1、多维数组如何储存在内存中

了解一维数组如何储存看这里：java数组如何存储在内存中

从底层来看，数组元素可以存放引用类型，数组也是引用类型的一种。也就是说，在数组元素的内部还可以包含数组(如int[][] numbers = new int[length][])，即二维数组可当作一维数组(数组长度为length)来处理。

了解数组和普通引用对象在内存中的区别看这里：数组和普通对象的引用变量有什么区别

由此我们得出结论: 任何多维数组(维度为n，n>1)都可以当作一维数组，其数组元素为n-1维数组。多维数组在内存中的储存同引用类型数组在内存中的储存。

了解引用类型数组在内存中的储存看这里：引用类型数组在内存中如何储存
以二维数组为例：

```
public class TestMultiArray {
    public static void main(String[] args) {
        // 1.定义二维数组
        int[][] numbers;
        // 2.分配内存空间
        numbers = new int[2][];
        // 可以把numbers看作一维数组来处理
        for (int i = 0; i < numbers.length; i++) {
            System.out.print(numbers[i] + ",");// null,null,null
        }
        System.out.println("");
        // 3.为数组元素指定值
        numbers[0] = new int[2];
        numbers[0][1] = 1;
        numbers[1] = new int[2];
        numbers[1][0] = 11;
        numbers[1][1] = 15;
        for (int i = 0; i < numbers.length; i++) {
            for (int j = 0; j < numbers[i].length; j++) {
                System.out.print(numbers[i][j] + ",");
            }
            System.out.println("");
        }
    }
}
```

结果如下：
null,null,
0,1,
11,15,

以一个图展示这个数组在内存中的存储：


逻辑地址
堆内存空间
栈内存空间
FF01H
...
FF02H
FF04H
FF05H
numbers
FF02H
FF03H
...
...
FF04H
0
1
numbers[0]
FF04H
FF05H
11
15
numbers[1]
FF05H

]

[

Java千百问_06数据结构（019）_Arrays类有什么功能

,

点击进入_更多_Java千百问

1、Arrays类有什么功能

java.util.Arrays中的类包含了很多静态方法，用于排序数组、搜索数组、比较数组和填充数组元素等功能。

了解一维数组看这里：java中的数组是什么

常见方法如下：

1. public static int binarySearch(Object[] a, Object key)
   使用二分法搜索数组中指定值的位置（下标）。如果数组类型非基本数据类型，则需要类实现Comparable接口中的compareTo方法。

2. public static boolean equals(long[] a, long[] a2)
   比较两个数组是否相等。如果两个指定数组相等返回true。默认两个数组相等的判定方法：

   - 两个数组包含相同的元素数目，并在两个数组所有元素对应相等。
   - 同样的方法可以用于所有其它所有的数据类型（byte, short, int等等）

3. public static void fill(int[] a, int val)
   将数组种所有元素都填充为指定的int值。同样的方法可以用于所有其它的原始数据类型（byte, short, int等等）。还有一个指定填充位置的方法：public static void fill(byte[] a, int fromIndex, int toIndex, byte val)，可以指定某些元素进行填充。

4.public static String toString(Object[] a)
将每个元素的值按顺序拼装为一个String类型的字符串，同样的方法可以用于所有其它的数据类型（byte, short, int等等）

1. public static void sort(Object[] a)
   根据其元素的自然顺序，按升序排序指定的数组，同样的方法可以用于所有其它的数据类型（byte, short, int等等）

2、Arrays如何使用

下面通过实例展示：binarySearch、copyOf、copyOfRange、equals、fill、sort、toString等方法。如下：

```
import java.util.Arrays;
public class TestArrays {
    public static void main(String[] args) {
        int[] a = {3, 4, 5, 6};
        int[] b = {3, 4, 5, 6};
        System.out.println("a和b是否相等：" + Arrays.equals(a, b));//true
        System.out.println("5在a中的位置：" + Arrays.binarySearch(a, 5));//2
        int[] c = Arrays.copyOf(a, 6);
        System.out.println("a和c是否相等：" + Arrays.equals(a, c));//false
        System.out.println("c的元素：" + Arrays.toString(c));//3,4,5,6,0,0
        Arrays.fill(c, 2, 4, 1);//将c中第3个到第5个元素(不包含)赋值为1
        System.out.println("c的元素：" + Arrays.toString(c));//3,4,1,1,0,0
        Arrays.sort(c);
        System.out.println("c的元素：" + Arrays.toString(c));//0,0,1,1,3,4
    }
}
```

执行结果如下：
a和b是否相等：true
5在a中的位置：2
a和c是否相等：false
c的元素：[3, 4, 5, 6, 0, 0]
c的元素：[3, 4, 1, 1, 0, 0]
c的元素：[0, 0, 1, 1, 3, 4]

]

[

Java千百问_06数据结构（020）_String是什么

,

点击进入_更多_Java千百问

1、String是什么

java.lang.String类表示字符串常亮，所谓字符串，就是一组字符组成的字符集合。它有以下几个特点：

String是不可变的对象
每次对String进行改变的时候，其实都等同于生成了一个新的String对象，然后将指针指向新的String对象（若不使用new构造，实际是从堆中的String池查找是否已经再存该字符串，若有则直接指向；若没有则先将该字符串放入String池，然后在指向）。
所以经常改变内容的字符串最好不要用String，每次生成对象都会对系统性能产生影响（特别当堆中无引用对象多了以后，JVM的垃圾回收GC就会开始工作，性能会受影响）。

了解垃圾回收看这里：java垃圾回收机制是什么
了解String在内存中如何储存看这里：String在内存中如何存放
String与StringBuffer效率比较
而在某些特别情况下，String对象的改变速度并不会比StringBuffer对象慢，而特别是以下的字符串对象生成中，String效率是远要比StringBuffer快的：

```
String s1 = "hello " + "world" ;
StringBuffer sb = new StringBuffer( "hello " ) . append( "world" );
```

你会惊讶的发现，生成String对象的速度简直太快了，而这个时候StringBuffer居然速度上根本一点都不占优势。
其实这是JVM的一个把戏，在JVM眼里，这个

```
String s1 = "hello " + "world" ;
```

等效于：

```
String s1 = "hello world" ;
```

所以不需要太多的时间了。但这里要注意，如果你的字符串是来自其他String对象的话，这时候JVM会规规矩矩创建若干对象，例如：

```
String s2 = "hello " ;
String s3 = "world" ;
String s1 = "hello world" ;
```

String还有两个类似的常用类，StringBuffer/StringBuilder，了解他们的使用和区别看这里：StringBuffer/StringBuilder有什么区别

]

[

Java千百问_06数据结构（021）_StringBuffer/StringBuilder有什么区别

,

点击进入_更多_Java千百问

1、StringBuilder是什么

StringBuilder是非线程安全的可变字符串类。
java.lang.StringBuilder这个可变的字符序列类是5.0新增的。继承于AbstractStringBuilder（大部分方法都在该类实现），并不是线程安全的，当多个线程同时修改一个对象时很可能会冲突。

了解String是什么看这里：String是什么

部分AbstractStringBuilder、StringBuilder源码：

```
abstract class AbstractStringBuilder implements Appendable, CharSequence {

    public AbstractStringBuilder append(Object obj) {

        return append(String.valueOf(obj));

   }

    public AbstractStringBuilder append(String str) {

        if (str == null ) str = "null" ;

        int len = str.length();

       ensureCapacityInternal(count + len);

       str.getChars( 0 , len, value, count);

       count += len;

        return this ;

   }

}

public final class StringBuilder extends AbstractStringBuilder implements java.io.Serializable, CharSequence

{

    public StringBuilder append(Object obj) {

        return append(String.valueOf(obj));

   }

    public StringBuilder append(String str) {

        super . append(str);

        return this ;

   }

}
```

2、StringBuffer是什么

StringBuffer是线程安全的可变字符串类

java.lang.StringBuffer继承于AbstractStringBuilder（同StringBuilder一样也继承于该类，并且部分方法都在该类实现），重写了其大部分方法，逻辑没变只是将其改为synchronized线程同步。

部分StringBuffer源码：

```
public final class StringBuffer extends AbstractStringBuilder implements java.io.Serializable, CharSequence

{

    public synchronized StringBuffer append(Object obj) {

        super . append(String.valueOf(obj));

        return this ;

   }

    public synchronized StringBuffer append(String str) {

        super . append(str);

        return this ;

   }

}
```

3、StringBuffer/StringBuilder有什么区别

由上面概念可以看到，StringBuffer是线程安全的，而StringBuilder并不是，StringBuilder在多线程中可能会发生冲突。

所以，当要保证线程安全的时候就需要用StringBuffer，然而在多线程时，StringBuffer比StringBuilder性能要差（synchronized相当于线程堵塞）。

]

[

Java千百问_06数据结构（022）_String在内存中如何存放

,

点击进入_更多_Java千百问

1、String在内存中如何存放

了解String是什么看这里：String是什么
String是一个特殊的包装类数据。
可以用两种的形式来创建：

```
String str = new String( "abc" );
String str = "abc" ;
```

了解java如何管理内存看这里：jvm是如何管理内存的
了解java堆和栈的区别看这里：java堆和栈有什么区别

第一种是用new()来新建对象的，它会在存放于堆中，每调用一次就会创建一个新的对象。
第二种是先在栈中创建一个对String类的对象引用变量str，然后查找运行时常量池中有没有存放”abc”，如果没有，则将”abc”存放进常量池，并令str指 向”abc”，如果已经有”abc” 则直接令str指向”abc”。

常量池在方法区内，用来存放基本类型包装类（包装类不管理浮点型，整形只会管理-128到127）和String（通过String.intern()方法可以强制将String放入常量池）。

2、String的equals方法如何使用

比较类里面的数值是否相等时，用equals()方法；当测试两个包装类的引用是否指向同一个对象时，用==（可以理解为比较逻辑地址，实际是比较对象号），下面用例子说明上面的理论。

```
String str1 = "abc" ;
String str2 = "abc" ;
System.out.println(str1==str2); //true
```

可以看出str1和str2是指向同一个对象的。

```
String str1 = new String ( "abc" );
String str2 = new String ( "abc" );
System.out.println(str1==str2); // false
```

- 对于第二种方式（String str1 =”abc”）创建多个”abc”字符串，在内存中其实只存在一个对象而已。这种方式节省内存空间，同时它可以在一定程度上提高程序的运行速度，因为JVM会自动根据常量池中数据的实际情况来决定是否有必要创建新对象。
- 而对于第一种方式，String str = new String(“abc”)，则一概在堆中创建新对象，而不管其字符串值是否相等，是否有必要创建新对象，从而加重了程序的负担。

另一个要注意的地方：我们在使用诸如String str = “abc”；的格式定义类时，总是想当然地认为，创建了String类的对象str。这里担心陷阱：对象可能并没有被创建！而可能只是指向一个先前已经创建的对象。只有通过new()方法才能保证每次都创建一个新的对象。

由于String类的immutable（不可变）性质，当String变量需要经常变换其值时，应该考虑使用StringBuffer/StringBuilder类，以提高程序效率。

StringBuffer/StringBuilder的区别看这里：StringBuffer/StringBuilder有什么区别

]

[

Java千百问_06数据结构（023）_基本数据类型在内存中如何存放

,

点击进入_更多_Java千百问

1、基本数据类型在内存中如何存放

了解基本数据类型看这里：java有哪8种基本数据类型
对于java中的8种基本数据类型，可以通过如下方式赋值给变量赋值。

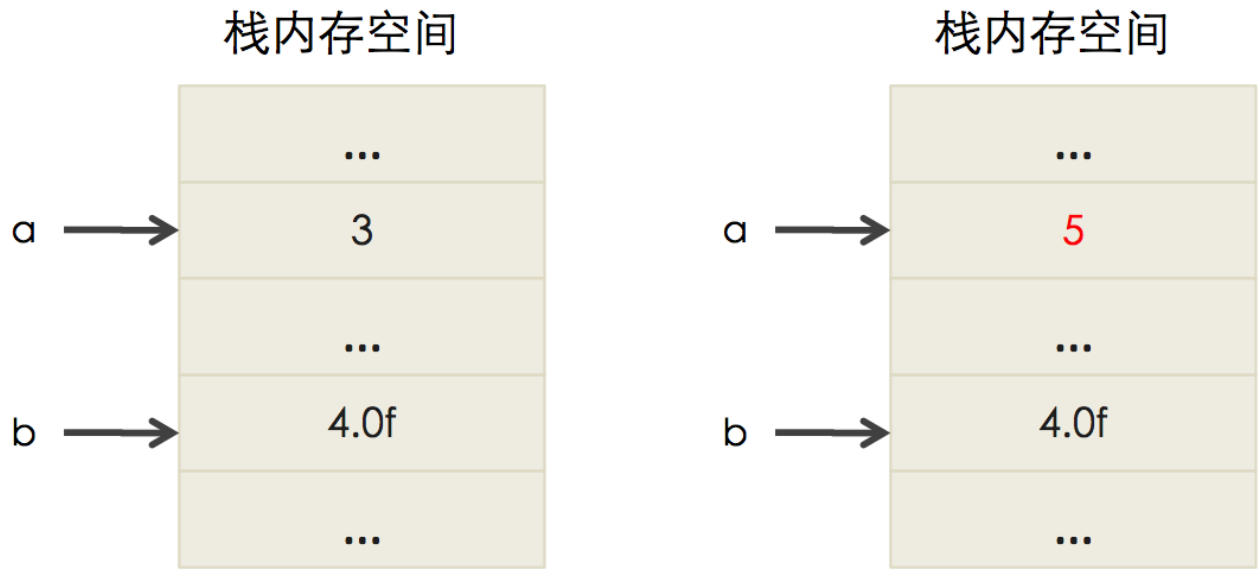
```
int a = 3;

float b = 4.0f;

a = a + 2;
```

8中基本数据是将具体值直接存放在栈中，在发生变更时，将具体值替换为新的值。具体如下：

了解java如何管理内存看这里：jvm是如何管理内存的
了解java堆和栈的区别看这里：java堆和栈有什么区别



对于基本数据类型，并没有基本数据池的概念，每次赋值并不会在栈中进行任何查询，而是直接存储值。

但是，对于基本数据类型的包装器，在一定数值范围内是存放在运行时常量池中的。

了解基本类型包装器看这里：什么是基本类型包装器
了解运行时常量池看这里：运行时常量池是什么

顺便提一句，对于基本数据类型，在内存中都是以二进制储存（当然，内存是以8位二进制作为一个存储单元，也就是一个字节），不同的类型所占用的内存空间（体现为存储单元）也不同。在java中，数值类型都是有符号存储（二进制首位为符号位），浮点类型也是遵循IEEE754、854标准。

了解二进制表示整型看这里：[用二进制如何表示整型数值][7]
了解二进制表示浮点型看这里：[用二进制如何表示浮点型数值][8]

对于一些不靠谱的资料或博客会混淆这个概念，认为基本数据类型在存储时首先回去看看栈中是否有该值，如果没有则放入，如果有则指向。这种说法纯属瞎扯，一个简单的道理，如果我分配了2G的栈内存，是否每次简单的int a = 1的赋值时，难道都会去排查一下这2g的空间？java没有这些人想象的这么无聊。

]

[

Java千百问_06数据结构（024）_用二进制如何表示整型数值

,

点击进入_更多_Java千百问

1、用二进制如何表示整型数值

我们都知道，计算机只认识0、1二进制，我们一般操作的寄存器和存储单元也都只认识二进制，我们称一个二进制为一个bit（位），一般32位计算机的寄存器允许操作32bit的数据，即32个0或1，由于书写过长，我们一般使用十六进制表示（每两个十六进制成为一个byte字节，即8bit=1byte）。例如：

1111 1111 1111 1111 1111 1111 1111 1111 = ffffffff

了解32位和64位计算机看这里：32位和64位计算机有什么区别

Java中也是一样，以上整型数值最终都会被解释为二进制机器码，具体规则如下：

1. 首位均是符号位，1代表负，0代表正。
2. 正值的二进制补码，即对应的负值。
3. 不同类型的范围均是-2^(n-1)到2^(n-1)-1，其中n为类型所占用的bit数。

其中所提到的补码，即反码+1，反码也就是对原码的每一个二进制位取反，以长度8bit整型为例：

原码：0100 1010

反码：1011 0101

补码：1011 0110

2、Java中整型如何用二进制表示

在Java语言中，整型数值分为4种：byte、short、int、long，均是带符号整型。

了解基本数据类型看这里：java有哪8种基本数据类型
这些基本类型除了长度不一致外，其他规则均按照以上规则，具体如下：

byte
内存中占用1个字节，8bit。由于是带符号数值，所以只能使用7bit的二进制来表示大小，最大只能表示111 1111，最小即它的补码1000 0000，所以他的范围是：

1 000 0000 到 0 111 1111

其中，首位符号位，1代表负，0代表正，故转换为10进制即-128到127。

例如：

```
public class Test {

    public static void main(String[] args) throws UnsupportedEncodingException {
        byte b1 = 100;
        byte b2 = -100;

        System.out.println("正byte===" + Integer.toBinaryString(b1));// 转二进制
        System.out.println("负byte===" + Integer.toBinaryString(b2));
```

```
        System.out.println("正byte===" + Integer.toHexString(b1));// 转十六进制
        System.out.println("负byte===" + Integer.toHexString(b2));
}
}
```

结果如下：
正byte==1111111
负byte==11111111111111111111111110000000
正byte==7f
负byte==ffffff80

这里我们看到，byte通过Integer类的工具转为二进制拥有32bit长度，实际上在java中，byte和short都是通过int进行赋值操作，正值会在前端补零，但真正压入栈或作为对象的属性存入堆时，只会占用对应长度：byte占用8bit，short占用16bit。例如：

```
byte b = 0x81;//编译错误
byte b = 0xffffff81;//正确
```

了解byte、short到内存占用看这里：[byte、short到底占用多少内存][4]

short
内存中占用2个字节，16bit。同byte，带符号数值，使用15bit的二进制来表示大小，最大只能表示111 1111 1111 1111，最小即它的补码1000 0000 0000 0000，所以他的范围是：

1 000 0000 0000 0000 到 0 111 1111 1111 1111

其中，首位符号位，1代表负，0代表正，故转换为10进制即-32768到32767。

例如：

```
public class Test {

    public static void main(String[] args) throws UnsupportedEncodingException {
        short s1 = 23235;
        short s2 = -23235;

        System.out.println("正short===" + Integer.toBinaryString(s1));// 转二进制
        System.out.println("负short===" + Integer.toBinaryString(s2));
        System.out.println("正short===" + Integer.toHexString(s1));// 转十六进制
        System.out.println("负short===" + Integer.toHexString(s2));
}
}
```

结果如下：
正short==101101011000011
负short==11111111111111111010010100111101
正short==5ac3
负short==ffffa53d

同样，在java中short只能通过int进行赋值。

int
内存中占用4个字节，32bit。带符号数值，使用31bit的二进制来表示大小，规则同上，他的范围是：

1 000 0000 0000 0000 0000 0000 0000 0000 到 0 111 1111 1111 1111 1111 1111 1111 1111

其中，首位符号位，1代表负，0代表正，故转换为10进制即-2,147,483,648到2,147,483,648。

例如：

```
public class Test {

    public static void main(String[] args) throws UnsupportedEncodingException {
        int i1 = 328999123;
        int i2 = -328999123;

        System.out.println("正int===" + Integer.toBinaryString(i1));// 转二进制
        System.out.println("负int===" + Integer.toBinaryString(i2));
        System.out.println("正int===" + Integer.toHexString(i1));// 转十六进制
        System.out.println("负int===" + Integer.toHexString(i2));
}
}
```

结果如下：

正int===10011100111000010000011010011
负int===11101100011000111101111100101101
正int===139c20d3
负int===ec63df2d

long

内存中占用8个字节，64bit。带符号数值，使用63bit的二进制来表示大小，规则同上，他的范围是：

1 000 0000 0000 0000 0000 0000 0000 0000 到 0 111 1111 1111 1111 1111 1111 1111 1111

其中，首位符号位，1代表负，0代表正，故转换为10进制即-2,147,483,648到2,147,483,648。

例如：

```
public class Test {

    public static void main(String[] args) throws UnsupportedEncodingException {
        int i1 = 328999123;
        int i2 = -328999123;

        System.out.println("正int===" + Integer.toBinaryString(i1));// 转二进制
        System.out.println("负int===" + Integer.toBinaryString(i2));
        System.out.println("正int===" + Integer.toHexString(i1));// 转十六进制
        System.out.println("负int===" + Integer.toHexString(i2));
}
}
```

结果如下：

正int===10011100111000010000011010011
负int===11101100011000111101111100101101
正int===139c20d3
负int===ec63df2d

这里要说的是，对于32位机，由于寄存器指令只能接受32bit的值，所以64bit的long在进行运算时只能分段处理，性能相对较慢。所以在32位机最好不要使用long和double等64bit的类型。而64位计算机可以完美使用。

]

[

Java千百问_06数据结构（025）_用二进制如何表示浮点型数值

,

点击进入_更多_Java千百问

1、用二进制如何表示浮点型数值

我们再了解二进制如何表达浮点型数值前，需要先了解用二进制如何表示整型数值：用二进制如何表示整型数值
由于计算机只认识0、1二进制，所以与表示整数一样，浮点数值最终也都会被解释为二进制机器码，与整型不同的是，所有由计算机储存的浮点类型，都是通过运算转换为十进制的，所以都是高度近似值，并不可能100%精确。具体规则如下：

1. 遵循Ieee754标准（IEEE二进位浮点数算术标准）
2. 首位均是符号位，1代表负，0代表正。

3.除去首位，用来表示浮点型的二进制要需要划分为指数位和尾数位（也称作小数位）。

1. 不同浮点类型的指数位和尾数位占用长度不一样。
2. 二进制转换十进制的指数偏差为：2^(指数位长度-1)-1。

这里所说的指数位、尾数位是十进制转二进制运算的关键，以32位浮点为例，它由1位符号位，8位指数位以及23位尾数位组成，例下面这个32位浮点数：

0100 0010 1101 0110 0101 0001 1100 1111

其中：指数位：100 0010 1；尾数位：101 0110 0101 0001 1100 1111。

从上面可知一下结论：

1. 符号位为：0，即正值。
2. 32位浮点，故指数偏差值：2^(8-1)-1 = 127。
3. 指数位为：100 0010 1，即十进制133。
4. 尾数位为：101 0110 0101 0001 1100 1111。

下面我们根据以上结论来运算十进制值，步骤如下：

1. 计算指数，指数位减去指数偏差值，即133-127=6
2. 计算小数，首先为尾数位前面补充小数点以及隐藏位1得：1.101 0110 0101 0001 1100 1111，而后右移指数6位得：1101 011.0 0101 0001 1100 1111
3. 逐位运算，逐位求2的乘方得：
   1*(2^6)+1*(2^5)+0*(2^4)+1*(2^3)+0*(2^2)+1*(2^1)+1*(2^0)+小数点+0*(2^-1)+0*(2^-2)+1*(2^-3)+0*(2^-4)+1*(2^-5)+0*(2^-6)+0*(2^-7)+0*(2^-8)+1*(2^-9)+1*(2^-10)+1*(2^-11)+0*(2^-12)+0*(2^-13)+1*(2^-14)+1*(2^-15)+1*(2^-16)+1*(2^-17)
   =107.1597824
4. 添加符号位得：+107.1597824

由此可知，浮点类型进制转换需要耗费一定的cpu运算，而且并不精确，如果想尽量精确，需要提升浮点类型的位数，例如64位。而且在一定范围外，不同的十进制浮点数可能会转换为相同的二进制浮点数，例如：

```
        float f3 = 423.15243f;
        float f4 = 423.15244f;
        System.out.println(f3 == f4);
```

结果为：

true

也就是说我们只能判断两个浮点数的精度差，一般使用f4 - f3 < 0.0001方式来判断两个浮点数是否相等（近似相等）。

2、Java中浮点型如何用二进制表示

在Java语言中，浮点数值分2种：float、double，均是带符号整型。

了解基本数据类型看这里：java有哪8种基本数据类型
这些类型除了长度不一致外，其他规则均按照以上规则，具体如下：

float
内存中占用8个字节，32bit。其中符号位1位，指数位8位，尾数位23位。指数偏差值：2^(8-1)-1 = 127

例如：

```
public class Test {

    public static void main(String[] args) throws UnsupportedEncodingException {
        float f1 = 423.1594f;
        float f2 = -423.1594f;
        int floatToIntBits1 = Float.floatToIntBits(f1);// 根据IEEE754规则，得到浮点的表示值。
        int floatToIntBits2 = Float.floatToIntBits(f2);

        System.out.println("正float===" + Integer.toBinaryString(floatToIntBits1));// 转二进制
        System.out.println("负float===" + Integer.toBinaryString(floatToIntBits2));
        System.out.println("正float===" + Integer.toHexString(floatToIntBits1));// 转十六进制
        System.out.println("负float===" + Integer.toHexString(floatToIntBits2));
}
}
```

结果如下：
正float==1000011110100111001010001100111
负float==11000011110100111001010001100111
正float==43d39467
负float==c3d39467

double
内存中占用16个字节，64bit。其中符号位1位，指数位11位，尾数位52位。指数偏差值：2^(8-1)-1 = 1023

例如：

```
public class Test {

    public static void main(String[] args) throws UnsupportedEncodingException {
        double d1 = 423453.1597824345;
        double d2 = -423453.1597824345;
        long doubleToLongBits1 = Double.doubleToLongBits(d1);// 根据IEEE754规则，得到浮点的表示值。
        long doubleToLongBits2 = Double.doubleToLongBits(d2);

        System.out.println("正double===" + Long.toBinaryString(doubleToLongBits1));// 转二进制
        System.out.println("负double===" + Long.toBinaryString(doubleToLongBits2));
        System.out.println("正double===" + Long.toHexString(doubleToLongBits1));// 转十六进制
```

```
        System.out.println("负double===" + Long.toHexString(doubleToLongBits2));
}
}
```

结果如下：

正double==1000001000110011101100001110100101000111001111000000000110101011

负double==1100000100011001110110000111010010100011100111100000000110101011

正double==4119d874a39e01ab

负double==c119d874a39e01ab

]