Java 教程

Java 教程
Java 简介
Java 开发环境配置
Java 基础语法
Java 对象和类
Java 基本数据类型
Java 变量类型
Java 修饰符
Java 运算符
Java 循环结构
Java 条件语句
Java switch case
Java Number & Math 类
Java Character 类
Java String 类
Java StringBuffer
Java 数组
Java 日期时间
Java 正则表达式
Java 方法
Java Stream、File、IO
Java Scanner 类
Java 异常处理

Java 面向对象

Java 继承
Java Override/Overload

← Java StringBuffer 和 StringBuilder 类　　Java 日期时间 →

Java 数组

数组对于每一门编程语言来说都是重要的数据结构之一，当然不同语言对数组的实现及处理也不尽相同。

Java 语言中提供的数组是用来存储固定大小的同类型元素。

你可以声明一个数组变量，如 numbers[100] 来代替直接声明 100 个独立变量 number0，number1，....，number99。

本教程将为大家介绍 Java 数组的声明、创建和初始化，并给出其对应的代码。

声明数组变量

首先必须声明数组变量，才能在程序中使用数组。下面是声明数组变量的语法：

```
dataType[] arrayRefVar;   // 首选的方法

或

dataType arrayRefVar[];  // 效果相同，但不是首选方法
```

注意: 建议使用 **dataType[] arrayRefVar** 的声明风格声明数组变量。 dataType arrayRefVar[] 风格是来自 C/C++ 语言 ，在Java中采用是为了让 C/C++ 程序员能够快速理解java语言。

实例

下面是这两种语法的代码示例：

```
double[] myList;         // 首选的方法

或

double myList[];         //  效果相同，但不是首选方法
```

创建数组

Java语言使用new操作符来创建数组，语法如下：

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

上面的语法语句做了两件事：

分类导航

HTML / CSS
JavaScript
服务端
数据库
移动端
XML 教程
ASP.NET
Web Service
开发工具
网站建设

Java 多态
Java 抽象类
Java 封装
Java 接口
Java 枚举
Java 包(package)

Java 高级教程

Java 数据结构
Java 集合框架
Java ArrayList
Java LinkedList
Java HashSet
Java HashMap
Java Iterator
Java Object
Java 泛型
Java 序列化
Java 网络编程
Java 发送邮件
Java 多线程编程
Java Applet 基础
Java 文档注释
Java 实例
Java 8 新特性
Java MySQL 连接
Java 9 新特性
Java 测验

一、使用 dataType[arraySize] 创建了一个数组。

二、把新创建的数组的引用赋值给变量 arrayRefVar。

数组变量的声明，和创建数组可以用一条语句完成，如下所示：

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

另外，你还可以使用如下的方式创建数组。

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

数组的元素是通过索引访问的。数组索引从 0 开始，所以索引值从 0 到 arrayRefVar.length -1。

实例

下面的语句首先声明了一个数组变量 myList，接着创建了一个包含 10 个 double 类型元素的数组，并且把它的引用赋值给 myList 变量。

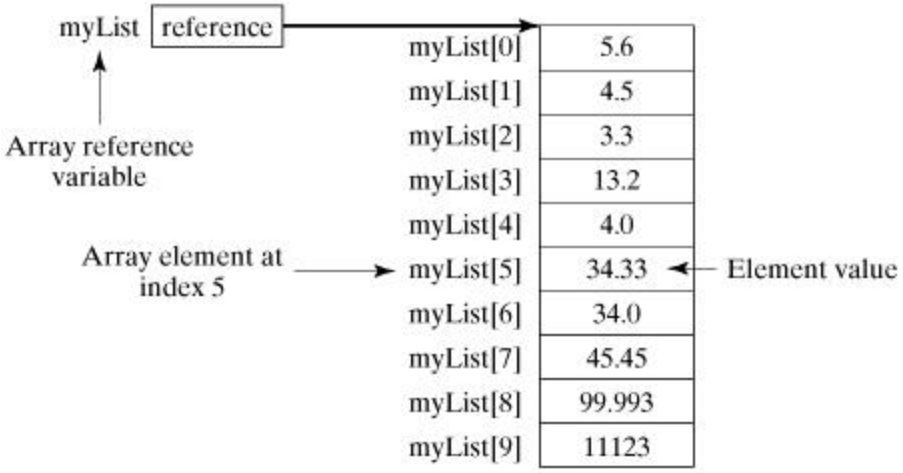
TestArray.java 文件代码：

```
public class TestArray {
   public static void main(String[] args) {
      // 数组大小
      int size = 10;
      // 定义数组
      double[] myList = new double[size];
      myList[0] = 5.6;
      myList[1] = 4.5;
      myList[2] = 3.3;
      myList[3] = 13.2;
      myList[4] = 4.0;
      myList[5] = 34.33;
      myList[6] = 34.0;
      myList[7] = 45.45;
      myList[8] = 99.993;
      myList[9] = 11123;
      // 计算所有元素的总和
      double total = 0;
      for (int i = 0; i < size; i++) {
         total += myList[i];
      }
      System.out.println("总和为： " + total);
   }
}
```

以上实例输出结果为：

```
总和为： 11367.373
```

下面的图片描绘了数组 myList。这里 myList 数组里有 10 个 double 元素，它的下标从 0 到 9。



处理数组

数组的元素类型和数组的大小都是确定的，所以当处理数组元素时候，我们通常使用基本循环或者 For-Each 循环。

示例

该实例完整地展示了如何创建、初始化和操纵数组：

TestArray.java 文件代码：

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

      // 打印所有数组元素
      for (int i = 0; i < myList.length; i++) {
         System.out.println(myList[i] + " ");
      }
      // 计算所有元素的总和
      double total = 0;
      for (int i = 0; i < myList.length; i++) {
         total += myList[i];
      }
      System.out.println("Total is " + total);
      // 查找最大元素
      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);
   }
}
```

以上实例编译运行结果如下：

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

For-Each 循环

JDK 1.5 引进了一种新的循环类型，被称为 For-Each 循环或者加强型循环，它能在不使用下标的情况下遍历数组。

语法格式如下：

```
for(type element: array)
{
    System.out.println(element);
}
```

实例

该实例用来显示数组 myList 中的所有元素：

TestArray.java 文件代码：

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

      // 打印所有数组元素
      for (double element: myList) {
         System.out.println(element);
      }
   }
}
```

以上实例编译运行结果如下：

```
1.9
2.9
3.4
3.5
```

数组作为函数的参数

数组可以作为参数传递给方法。

例如，下面的例子就是一个打印 int 数组中元素的方法:

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

下面例子调用 printArray 方法打印出 3，1，2，6，4 和 2：

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

数组作为函数的返回值

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--)
{
    result[j] = list[i];
  }
  return result;
}
```

以上实例中 result 数组作为函数的返回值。

多维数组

多维数组可以看成是数组的数组，比如二维数组就是一个特殊的一维数组，其每一个元素都是一个一维数组，例如：

```
String str[][] = new String[3][4];
```

多维数组的动态初始化（以二维数组为例）

1. 直接为每一维分配空间，格式如下：

```
type[][] typeName = new type[typeLength1][typeLength2];
```

type 可以为基本数据类型和复合数据类型，typeLength1 和 typeLength2 必须为正整数，typeLength1 为行数，typeLength2 为列数。

例如：

```
int a[][] = new int[2][3];
```

解析：

二维数组 a 可以看成一个两行三列的数组。

2. 从最高维开始，分别为每一维分配空间，例如：

```
String s[][] = new String[2][];
s[0] = new String[2];
s[1] = new String[3];
s[0][0] = new String("Good");
s[0][1] = new String("Luck");
s[1][0] = new String("to");
s[1][1] = new String("you");
s[1][2] = new String("!");
```

解析：

s[0]=new String[2] 和 **s[1]=new String[3]** 是为最高维分配引用空间，也就是为最高维限制其能保存数据的最长的长度，然后再为其每个数组元素单独分配空间 **s0=new String("Good")** 等操作。

多维数组的引用（以二维数组为例）

对二维数组中的每个元素，引用方式为 **arrayName[index1][index2]**，例如：

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

Arrays 类

java.util.Arrays 类能方便地操作数组，它提供的所有方法都是静态的。

具有以下功能：

给数组赋值：通过 fill 方法。

对数组排序：通过 sort 方法,按升序。

比较数组：通过 equals 方法比较数组中元素值是否相等。

查找数组元素：通过 binarySearch 方法能对排序好的数组进行二分查找法操作。

具体说明请查看下表：

序号	方法和说明
1	**public static int binarySearch(Object[] a, Object key)** 用二分查找算法在给定数组中搜索给定值的对象(Byte,Int,double等)。数组在调用前必须排序好的。如果查找值包含在数组中，则返回搜索键的索引；否则返回 (-(*插入点*) - 1)。
2	**public static boolean equals(long[] a, long[] a2)** 如果两个指定的 long 型数组彼此*相等*，则返回 true。如果两个数组包含相同数量的元素，并且两个数组中的所有相应元素对都是相等的，则认为这两个数组是相等的。换句话说，如果两个数组以相同顺序包含相同的元素，则两个数组是相等的。同样的方法适用于所有的其他基本数据类型（Byte，short，Int等）。
3	**public static void fill(int[] a, int val)** 将指定的 int 值分配给指定 int 型数组指定范围中的每个元素。同样的方法适用于所有的其他基本数据类型（Byte，short，Int等）。
4	**public static void sort(Object[] a)** 对指定对象数组根据其元素的自然顺序进行升序排列。同样的方法适用于所有的其他基本数据类型（Byte，short，Int等）。

练习

Java 数组测验

← Java StringBuffer 和 StringBuilder 类　　Java 日期时间 →