007 数组

作者：pzs233
原文链接：https://ld246.com/article/1537282826432
来源网站：链滴

本文为《Java 语言程序设计》第十版 章节笔记

007 一维数组

Java 和许多高级语言都提供了一种称作 **数组（array）** 的数据结构，可以用它来存储一个元素个数定且元素类型相同的有序集。

7.1 数组的基础知识

一旦数组被创建，它的大小是固定的。使用一个数组引用变量，通过下标来访问数组中的元素。

声明数组变量的语法：

```
elementType[] arrayRefVar;//（元素类型[] 数组引用变量）
```

elementType可以是任意数据类型，但是数组中所有的元素都必须具有相同的数据类型。

不同于基本数据类型变量的声明，声明一个数组变量时并不在内存中给数组分配 任何空间。它只是创一个对数组的引用的存储位置。如果变量不包含对数组的引用，那么这个变量的值为null。除非数组经被创建，否则不能给它分配任何元素。

声明一个数组变量、创建数组、然后将数组引用赋值给变量这三个步骤可以合并在一条语句里，如下示：

```
elementType[] arrayReVar = new elementType[arraySize];
```

（元素类型[] 数组引用变量 = new 元素类型[数组大小]; ）

当创建数组后，它的元素被赋予默认值，数值型基本类型的默认值为0，char型的默认值为'\u0000'，oolean型的默认值为false。

数组的下标是基于0的，也就是说，其范围从0开始到arrayReVar.length-1结束。

使用数组初始化语法时，必须将声明、创建和初始化数组都放在一条语句中。

处理数组：

1. 使用输入值初始化数组

```
java.tuil.Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter" + myList.length + " values: ");
for (int i = 0; i < myList.length; i++){
    myList[i] = input.nextDouble();
}
```

2. 使用随机数初始化数组

```
for (int i = 0; i < myList.length; i++){
    myList[i] = Math.random() * 100;
}
```

3. 显示数组

```
for (int i = 0; i < myList.length; i++){
    System.out.print(myList[i] + " ");
```

```
}
```

对于char[]类型的数组，可以使用一条打印语句显示数组，例如：

```
char[] city = {'D', 'a', 'l', 'l', 'a', 's'};
System.out.print(city);
````
```

4. 对所有元素求和

```
double total = 0;
for (int i = 0; i < myList.length; i++){
    total += myList[i];
}
```

5. 找出最大元素

```
double max = myList[0];
for (int i = 0; i < myList.length; i++){
    if (myList[0] > max){
        max = myList[i];
    }
}
```

6. 找出最大元素的最小下标

```
double max = myList[0];
int indexOfMax = 0;
for (int i = 1; i < myList.length; i++){
    if (myList[i] > max) {
        max = myList[i];
        indexOfMax = i;
    }
}
```

7. 随机打乱(shuffling)

```
for (int i = myList.length - 1; i > 0; i--){
    // Generate an index j randomly with 0 <= j <= i
    int j = (int)(Math.random() * (i + 1));

    // Swap myList[i] with myList[j]
    double temp = myList[i];
    myList[i] = myList[j];
    myList[j] = temp;
}
```

8. 移动元素（左移或右移）

```
double temp = myList[0]; // Retain the first element

// Shift element left
for (int i = 1; i < myList.length; i++){
    myList[i - 1] = myList[i];
}

// Move the first element to fill the last position
myList[myList.length - 1] = temp;
```
```

9. 简化编码

例如，输入月份数字，获得月份名称：

```
String[] months = {"January", "February",..., "December"};
System.out.print("Enter a month number (1 to 12): ");
int minthNumber = input.nextInt();
System.out.print("The month is " + months[monthNumber - 1]);
```

foreach循环

Java支持一个简便的for循环，称为foreach循环，即不使用下标变量就可以顺序地遍历整个数组。例：

```
for (double e: myList) {
    System.out.println(e);
}
// "对myList中每个元素e 进行一下操作"。注意，变量 e 必须声明为与myList中元素相同的数据类型
```

语法：

```
for (elementType element: arrayReVar) {
    // Process the element
}
```

7.2 数组的复制

要将一个数组中的内容复制到另一个中，需要将数组的每个元素复制到另一个数组中。

在Java中，可以使用赋值语句复制基本数据类型的变量，但不能复制数组。

复制数组有三种方法：

1. 使用循环语句逐个地复制数组的元素。

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[soutceArray.length];
for (int i = 0; i < soutceArray.length; i++) {
    targetArray[i] = sourceArray[i];
}
```

2. 使用System类中的静态方法arraycopy。

```
arraycopy(sourceArray, srcPos, targetArray, tarPos, length);
// 参数srcPos 和 tarPos 分别为原数组 sourceArray 和
// 目标数组targetArray 中的起始位置，length为复制元素的个数
```

完整复制为：

```
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```

arraycopy方法没有给目标数组分配内存空间，复制前必须创建目标数组以及分配给它内存空间。复完成后，sourceArray和targetArray具有相同的内容，但占有独立的内存空间。

3. 使用clone方法复制数组。

13章介绍

7.3 将数组传递给方法

- 对于基本数据类型参数，传递的是实参的值。
- 对于数组类型参数，参数值是数组的引用，给方法传递的是这个引用。从语义上来讲，最好的描述是参数传递的是共享信息（pass-by-sharing），即方法中的数组和传递的数组是一样的。所以，如改变方法中的数组，将会看到方法外的数组也改变了。

数组在 Java 中是对象，JVM 将对象存储在一个称为堆（heap）的内存区域中，堆用于动态内存分配（方法在栈中）

7.4 可变长参数列表

具有同类型的可变长度的参数可以传递给方法，并将作为数组对待。

方法声明如下：

```
typeName...parameterName //  （类型名...参数名）
```

e.g:

```
public class VarArgsDemo{
    public static void main(String[] args) {
        printMax(34, 3, 3, 2, 56.5);
        printMax(new double[]{1, 2, 3});
    }

    public static void printMax(double... number) {
        if (number.length == 0) {
            System.out.println("No argument passed");
            return;
        }

        double result = numbers[0];

        for (int i = 1; i < numbers.length; i++){
            result = numbers[i];
        }

        System.out.println("The max value is " + result);
    }
}
```

7.5 数组的查找

如果一个数组排好序了，对于寻找数组中的一个元素，二分查找比线性查找更加高效。

两种常用的方法：

- 线性查找（linear searching）

```
public class LinearSearch {
    /** The method for finding a key in the list */
    public static int linearSearch(int[] list, int key) {
```

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

如果匹配成功，返回与关键字匹配的元素在数组中的下标。如果没有匹配成功，则返回-1。

数组中的元素可以按任意顺序排列。

- 二分查找（binary searching）（使用二分查找法的前提条件是数组中的元素只需已经排好序。）

```
public class BinarySearch {
    /** Use binary search to find the key in the list */
    public static int binarySearch(int[] list, int key) {
        int low = 0;
        int high = list.length - 1;

        while (hgih >= low) {
            int mid = (low + high) / 2;

            if (key < list[mid]) {
                high = mid - 1;
            } else if (key == list[mid]) {
                return mid;
            } else {
                low = mid + 1;
            }
        }

        return -low - 1; // Now hig < low, key not found
    }
}
```

此方法数组按升序排好。匹配成功，返回下标。如果关键字不在该序列中，方法返回-low-1，不仅表关键字不在序列中，而且还给了关键字应该插入的位置（index = low)。

7.6 数组的排序

选择排序：假设要按升序排列一个数列。先找到数列中最小的数，然后将它第一个元素交换。接下来在剩下的数中找到最小数，将它和第二个元素交换，以此类推，直到数列中仅剩一个数为止。

```
public class SelectionSort {
    /** The method for sorting the numbers */
    public static void selectionSort(double[] list) {
        for (int i = 0; i < list.length - 1; i++) {
            // Find the minimum in the list[i...list.length-1]
            double currentMin = list[i];
            int currentMinIndex = i;

            for (int j = i + 1; j < list.length; j++) {
                if (currentMin > list[j]) {
                    currentMin = list[j];
```

```
                currentMinIndex = j;
            }
        }

        // Swap list[i] with list[currentMinIndex] if necessary
        if (currentMinIndex != i) {
            list[currentMinIndex] = lsit[i];
            lsit[i] = current;
        }
    }
  }
}
```

7.6 Arrays类

java.tuil.Arrays类包含一些实用的方法用于常见的数组操作，比如排序和查找。

1. 使用 sort 或者 parallelSort 方法对整个数组或部分数组进行排序。

```
double[] numbers = {6.0, 4.4, 1.9, 3.4, 3.5};
java.util.Arrays.sort(numbers); // Sort the whole array
java.uril.Arrays.parallelSort(numbers); // Sort the whole array

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars, 1, 3); // Sort part of the array
java.tuil.Arrays.parallelSort(chars, 1, 3); // Sort part of the array
```

可以调用sort(chars, 1, 3)对从chars[1] 到 chars[3-1] 的部分数组排序。

2. 采用二分查找法在数组中查找关键字。

```
int[] lsit = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("1. Index is " + java.tuil.Arrays.binarySearch(list, 11));
System.out.println("2. Index is " + java.util.Arrays.binarySearch(lsit, 12));

char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("3. Index is " + java.util.Arrays.binarySearch(chars, 'a'));
System.out.println("4. Index is " + java.util.Arrays.binarySearch(chars, 't'));
/** Output:
*1. Index is 4
*2. Index is -6
*3. Index is 0
*4. Index is -4
*/
```

3. 可以采用 equals方法检测两个数组是否相等

```
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 10};
int[] list3 = {4, 2, 7, 10};
System.out.println(java.util.Arrays.equals(list1, list2));
System.out.println(java.util.Arrays.equals(list1, list3));
```

4. 可以使用 fill方法填充整个数组或部分数组

5. Arrays.toString()

008 多维数组

二维数组的基础知识

语法

数据类型 [][] 数组名;

二维数组装的两个下标，一个表示行，另一个表示列。

Java 中每个下标必须放在一对方括号中。

获取二维数组的长度

分别获取每行的一维数组的长度

x[0].length、x[1].length...

锯齿数组

二维数组的每行的长度不一样。

处理二维数组

嵌套的 for 循环 常用来处理二维数组

将二维数组传递给方法

将一个二维数组传递给方法的时候，数组的引用传递给了方法。

多维数组

二维数组由一个一维数组的数组组成，三维数组则可以认为是由一个二维数组的数组组成。