

# Java 基础语法

## 第一章 数组

在生活中，我们可能会碰到如下的场景。

现在需要统计某公司员工的工资情况，例如计算平均工资、最高工资等。假设该公司有 50 名员工，用前面所学的知识完成，那么程序首先需要声明 50 个变量来分别记住每位员工的工资，这样做会显得很麻烦。

其实在 Java 中，我们可以使用一个数组来记住这 50 名员工的工资。数组是指一组数据的集合，数组中的每个数据被称作元素。在数组中可以存放任意类型的元素，但同一个数组里存放的元素类型必须一致。

### JVM(虚拟机)内存模型:

JVM 内存划分:

程序计数器：当前线程所执行的字节码的行号指示器。

本地方法栈：为虚拟机使用的 native 方法服务。

**Java 虚拟机栈**：描述 Java 方法执行的内存模型，每个方法被执行的时候都会同时创建一个栈帧用于存储**局部变量**表、操作栈、动态链接、方法出口等信息。

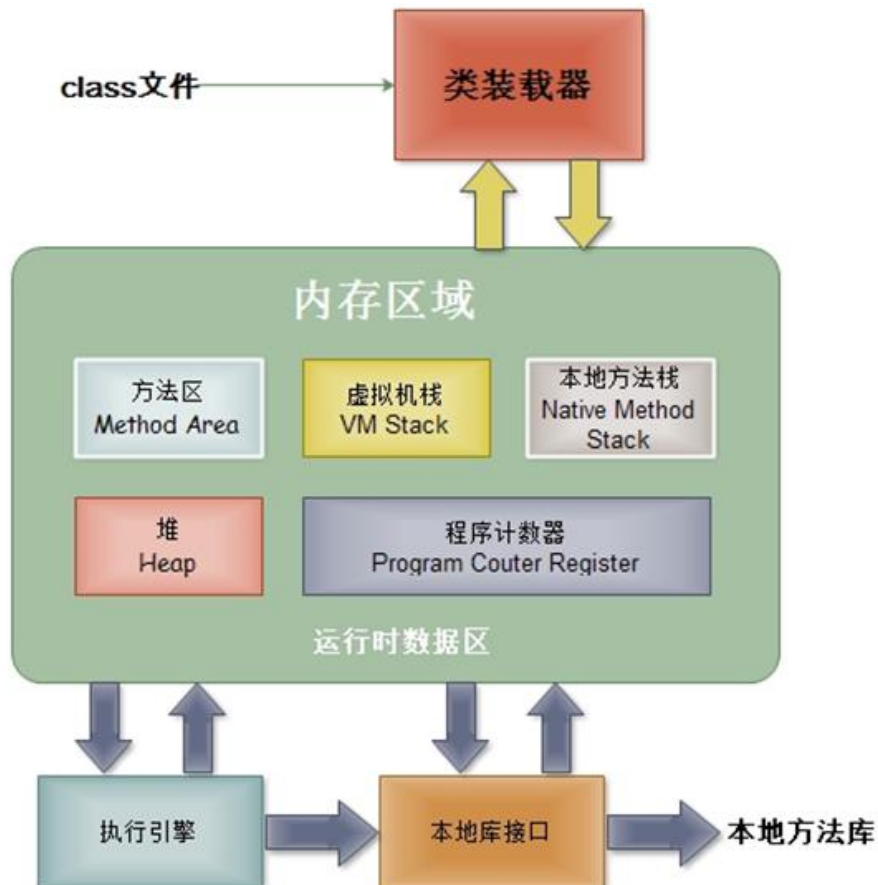
**Java 堆(heap)**：被所有线程共享的一块内存区域，在虚拟机启动时创建。所有的**对象实例**以及**数组**都要在堆上分配(存储对象数据,一般的,使用 new 出来的数据都在堆中)。

**方法区**：线程共享的内存区域，存储已被虚拟机加载的类信息、常量、静态变量即时编译器编译后的代码数据等(这个区域的内存回收目标主要是针对常量池的回收和对类型的卸载),面向对象再讲。

**GC(Garbage Collection):垃圾回收器。**

**Java 的自动垃圾回收机制:**简单理解为,

程序员就不需要再手动的去控制内存的释放。当 JVM 发觉内存资源紧张的时候,就会自动地去清理无用对象（**没有被引用到的对象**）所占用的内存空间。



## 数组的定义

数组								
索引(index)	0	1	2	3	4	5	6	7
元素	A	B	C	D	E	F	G	H

在 Java 中，可以使用以下格式来定义一个数组。如下

**数据类型[] 数组名 = new 数据类型[元素个数或数组长度];**

```
int[] x = new int[100];
```

上述语句就相当于在内存中定义了 100 个 int 类型的变量，第一个变量的名称为 x[0]，第二个变量的名称为 x[1]，以此类推，第 100 个变量的名称为 x[99]，这些变量的初始值都是 0。为了更好地理解数组的这种定义方式，可以将上面的一句代码分成两句来写，具体如下：

```
int[] x;           // 声明一个 int[] 类型的变量
x = new int[100];  // 创建一个长度为 100 的数组
```

接下来，通过两张内存图来详细地说明数组在创建过程中内存的分配情况。

第一行代码 `int[] x;` 声明了一个变量 `x`，该变量的类型为 `int[]`，即一个 `int` 类型的数组。变量 `x` 会占用一块内存单元，它没有被分配初始值。内存中的状态如下图所示。



图 1-1 内存状态图

第二行代码 `x = new int[100];` 创建了一个数组，将数组的地址赋值给变量 `x`。在程序运行期间可以使用变量 `x` 来引用数组，这时内存中的状态会发生变化，如下图所示。

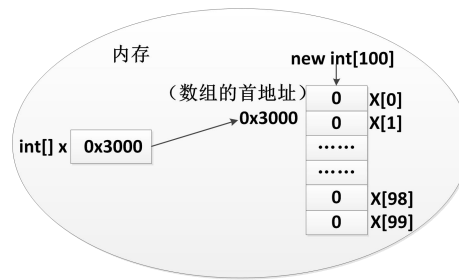


图 1-2 内存状态图

在上图中描述了变量 x 引用数组的情况。该数组中有 100 个元素，初始值都为 0。数组中的每个元素都有一个索引(也可称为角标)，要想访问数组中的元素可以通过“x[0]、x[1]、.....、x[98]、x[99]”的形式。需要注意的是，数组中最小的索引是 0，最大的索引是“数组的长度-1”。在 Java 中，为了方便我们获得数组的长度，提供了一个 length 属性，在程序中可以通过“数组名.length”的方式来获得数组的长度，即元素的个数。

接下来，通过一个案例来演示如何定义数组以及访问数组中的元素，如下所示。

#### ArrayDemo01.java

```
1 public class ArrayDemo01 {  
2     public static void main(String[] args) {  
3         int[] arr; // 声明变量  
4         arr = new int[3]; // 创建数组对象  
5         System.out.println("arr[0]= " + arr[0]); // 访问数组中的第一个元素  
6         System.out.println("arr[1]= " + arr[1]); // 访问数组中的第二个元素  
7         System.out.println("arr[2]= " + arr[2]); // 访问数组中的第三个元素  
8         System.out.println("数组的长度是： " + arr.length); // 打印数组长度  
9     }  
10 }
```

运行结果如下图所示。

```
D:\java>java ArrayDemo01  
arr[0]=0  
arr[1]=0  
arr[2]=0  
数组的长度是： 3
```

图 1-3 运行结果

在上述代码中声明了一个 `int[]` 类型变量 `arr`，并将数组在内存中的地址赋值给它。在 5~7 行代码中通过角标来访问数组中的元素，在第 8 行代码中通过 `length` 属性访问数组中元素的个数。从打印结果可以看出，数组中的三个元素初始值都为 0，这是因为当数组被成功创建后，数组中元素会被自动赋予一个默认值，根据元素类型的不同，默认初始化的值也是不一样的。具体如下表所示。

表 2-1 元素默认值

数据类型	默认初始化值
byte、short、int、long	0
float、double	0.0
char	一个空字符（空格），即 <code>'\u0000'</code>
boolean	false
引用数据类型	null，表示变量不引用任何对象

如果在使用数组时，不想使用这些默认初始值，也可以显式地为这些元素赋值。接下来通过一个程序来学习如何为数组的元素赋值，如下所示。ArrayDemo02.java

```
1 public class ArrayDemo02 {  
2     public static void main(String[] args) {  
3         int[] arr = new int[4]; // 定义可以存储 4 个整数的数组  
4         arr[0] = 1; // 为第 1 个元素赋值 1  
5         arr[1] = 2; // 为第 2 个元素赋值 2  
6         // 下面的代码是打印数组中每个元素的值  
7         System.out.println("arr[0]=" + arr[0]);  
8         System.out.println("arr[1]=" + arr[1]);  
9         System.out.println("arr[2]=" + arr[2]);  
10        System.out.println("arr[3]=" + arr[3]);  
11    }  
12 }
```

运行结果如下图所示。

```
D:\java>java ArrayDemo02
arr[0]=1
arr[1]=2
arr[2]=0
arr[3]=0
```

图 1-4 运行结果

在上述代码中，第 3 行代码定义了一个数组，此时数组中每个元素都为默认初始值 0。第 2、3 行代码通过赋值语句将数组中的元素 arr[0]和 arr[1]分别赋值为 1 和 2，而元素 arr[2]和 arr[3]没有赋值，其值仍为 0，因此打印结果中四个元素的值依次为 1、2、0、0。

在定义数组时只指定数组的长度，由系统自动为元素赋初值的方式称作动态初始化。

在初始化数组时还有一种方式叫做静态初始化，就是在定义数组的同时就为数组的每个元素赋值。数组的静态初始化有两种方式，具体格式如下：

- 1、类型[] 数组名 = new 类型[] {元素, 元素, .....};
- 2、类型[] 数组名 = {元素, 元素, 元素, .....};

上面的两种方式都可以实现数组的静态初始化，但是为了简便，建议采用第二种方式。接下来通过一段代码来演示数组静态初始化的效果，如下所示。ArrayDemo03.java

```
1 public class ArrayDemo03 {
2     public static void main(String[] args) {
3         int[] arr = { 1, 2, 3, 4 }; // 静态初始化
4         // 下面的代码是依次访问数组中的元素
5         System.out.println("arr[0] = " + arr[0]);
6         System.out.println("arr[1] = " + arr[1]);
7         System.out.println("arr[2] = " + arr[2]);
8         System.out.println("arr[3] = " + arr[3]);
9     }
10 }
```

运行结果如下图所示。

```
D:\java>java ArrayDemo03
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
```

图 1-5 运行结果

上述代码中采用静态初始化的方式为数组每个元素赋予初值，分别是 1、2、3、4。需要注意的是，第 3 行代码千万不可写成 `int[] arr = new int[4]{1,2,3,4};`，这样写编译器会报错。原因在于编译器会认为数组限定的元素个数[4]与实际存储的元素{1,2,3,4}个数有可能不一致，存在一定的安全隐患。

## 小结:

**数组的初始化操作:**开辟内存空间,存储数据.

**数组必须先初始化才能使用.**

当数组初始化之后,数组的长度就已经确定了,不能更改.

### 静态初始化:

程序员自己设置需要存储的数据(元素),而数组的长度由系统决定.

语法:

元素类型[] 数组名称 = new 元素类型[]{元素 1,元素 2,元素 3,...};

如:

```
int[] nums = new int[]{1,3,5,7,9};
```

更简单的语法:

```
int[] nums = {1,3,5,7,9};
```

### 动态初始化:

程序员指定数组的长度,而数组的元素的默认值由系统决定.

语法:

```
元素类型[] 数组名称 = new 元素类型[length];
```

如:

```
int[] nums = new int[5];
```

如果事先知道需要存储哪些数据,---->**使用静态初始化.**

若事先不知道存储哪些数据----->**动态初始化.**

注意:

**不能同时使用静态和动态初始化.**

```
int[] nums = new int[5]{1,3,5,7,9};
```

## 数组的遍历

在操作数组时，经常需要依次访问数组中的每个元素，这种操作称作数组的遍历。接下来通过一个案例来学习如何使用 for 循环来遍历数组，如下所示。ArrayDemo04.java

```
public class ArrayDemo04 {  
    public static void main(String[] args) {  
        int[] arr = { 1, 2, 3, 4, 5 }; // 定义数组  
        // 使用 for 循环遍历数组的元素  
        for (int i = 0; i < arr.length; i++) {  
            System.out.println(arr[i]); // 通过索引访问元素  
        }  
    }  
}
```

运行结果如下图所示。



```
D:\java>java ArrayDemo04
1
2
3
4
5
```

图 1-6 运行结果

上述代码中，定义一个长度为 5 的数组 `arr`，数组的角标为 0~4。由于 `for` 循环中定义的变量 `i` 的值在循环过程中为 0~4，因此可以作为索引，依次去访问数组中的元素，并将元素的值打印出来。

## 1.1 数组的常见问题

数组在编写程序时应用非常广泛，灵活地使用数组对实际开发很重要。接下来，本节将针对数组的常见操作进行详细地讲解，如数组的遍历、最值的获取、数组的排序等。

### 1.1.1 数组最值

在操作数组时，经常需要获取数组中元素的最值。接下来通过一个案例来演示如何获取数组中元素的最大值，如下所示。ArrayDemo05.java

```
public class ArrayDemo05 {
    public static void main(String[] args) {
        int[] arr = { 4, 1, 6, 3, 9, 8 }; // 定义一个数组
        int max = arr[0]; // 定义变量 max 用于记住最大数，首先假设第一个元素为最大值
        // 下面通过一个 for 循环遍历数组中的元素
        for (int x = 1; x < arr.length; x++) {
            if (arr[x] > max) { // 比较 arr[x] 的值是否大于 max
                max = arr[x]; // 条件成立，将 arr[x] 的值赋给 max
            }
        }
        System.out.println("max=" + max); // 打印最大值
    }
}
```

运行结果如下图所示。

```
D:\java>java ArrayDemo05  
max=9
```

图 1-7 运行结果

上述代码中，定义了一个临时变量 `max`，用于记住数组的最大值。通过 `for` 循环获取数组中的最大值，赋值给 `max` 变量。

首先假设数组中第一个元素 `arr[0]` 为最大值，然后使用 `for` 循环对数组进行遍历，在遍历的过程中只要遇到比 `max` 值还大的元素，就将该元素赋值给 `max`。这样一来，变量 `max` 就能够在循环结束时记住数组中的最大值。需要注意的是，在 `for` 循环中的变量 `i` 是从 1 开始的，这样写的原因是程序已经假设第一个元素为最大值，`for` 循环中只需要从第二个元素开始比较，从而提高程序的运行效率。

### 1.1.1 数组异常

#### 1.1.1.1 数组越界异常

每个数组的索引都有一个范围，即  $0 \sim \text{length}-1$ 。在访问数组的元素时，索引不能超出这个范围，否则程序会报错，如下所示。ArrayDemo06.java

```
1 public class ArrayDemo06 {  
2     public static void main(String[] args) {  
3         int[] arr = new int[4]; // 定义一个长度为 4 的数组  
4         System.out.println("arr[0]=" + arr[4]); // 通过角标 4 访问数组元素  
5     }  
6 }
```

运行结果如下图所示。

```
D:\java>java ArrayDemo06  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4  
    at ArrayDemo06.main(ArrayDemo06.java:4)
```

上图运行结果中所提示的错误信息是数组越界异常 `ArrayIndexOutOfBoundsException` ,出现这个异常的原因是数组的长度为 4 , 其索引范围为 0~3 , 而上述代码中的第 4 行代码使用索引 4 来访问元素时超出了数组的索引范围。

所谓异常指程序中出现的错误 , 它会报告出错的异常类型、出错的行号以及出错的原因 , 关于异常在后面的章节会有详细地讲解。

## 空指针异常

在使用变量引用一个数组时 , 变量必须指向一个有效的数组对象 , 如果该变量的值为 `null` , 则意味着没有指向任何数组 , 此时通过该变量访问数组的元素会出现空指针异常 , 接下来通过一个案例来演示这种异常 , 如下所示。 `ArrayDemo07.java`

```
1 public class ArrayDemo07 {  
2     public static void main(String[] args) {  
3         int[] arr = new int[3]; // 定义一个长度为 3 的数组  
4         arr[0] = 5; // 为数组的第一个元素赋值  
5         System.out.println("arr[0]=" + arr[0]); // 访问数组的元素  
6         arr = null; // 将变量 arr 置为 null  
7         System.out.println("arr[0]=" + arr[0]); // 访问数组的元素  
8     }  
9 }
```

运行结果如下图所示。



```
D:\java>java ArrayDemo07  
arr[0]=5  
Exception in thread "main" java.lang.NullPointerException  
    at ArrayDemo07.main(ArrayDemo07.java:7)
```

图 1-8 运行结果

通过上图所示的运行结果可以看出 , 上述代码中第 4、5 行代码都能通过变量 `arr` 正常地操作数组。第 6 行代码将变量置为 `null` , 当第 7 行代码再次访问数组时就出现了空指针异常

NullPointerException。

### 小结:

#### 数组的基本操作:

0):打印数组对象,会打印出数组的 **hashCode 值**,看不出来数组元素值.

1):获取/设置/遍历元素元素:

获取数组元素值:      数组元素类型 变量 =    **数组名[index];**

设置数组元素值:      数组名[index] = 值;

迭代数组元素:          **使用循环,一般的,首选 for 循环.**

2):数组的长度(使用 length 属性):

**int len = 数组名.length;**//获取当前数组有几个元素个数

3):数组的索引从 0 开始,逐一递增.

**数组索引的范围:[0,数组长度-1]**

4):操作数组常见异常(错误):

**ArrayIndexOutOfBoundsException:**数组的索引越界,不在索引范围之内.

**NullPointerException:**空引用异常,操作了一个为 null 的数组变量.

5):获取数组最大最小元素(思路)

求最大值: getMax

求最小值: getMin

## 数组的排序

排序的分类：

选择排序（直接**选择排序**、堆排序）

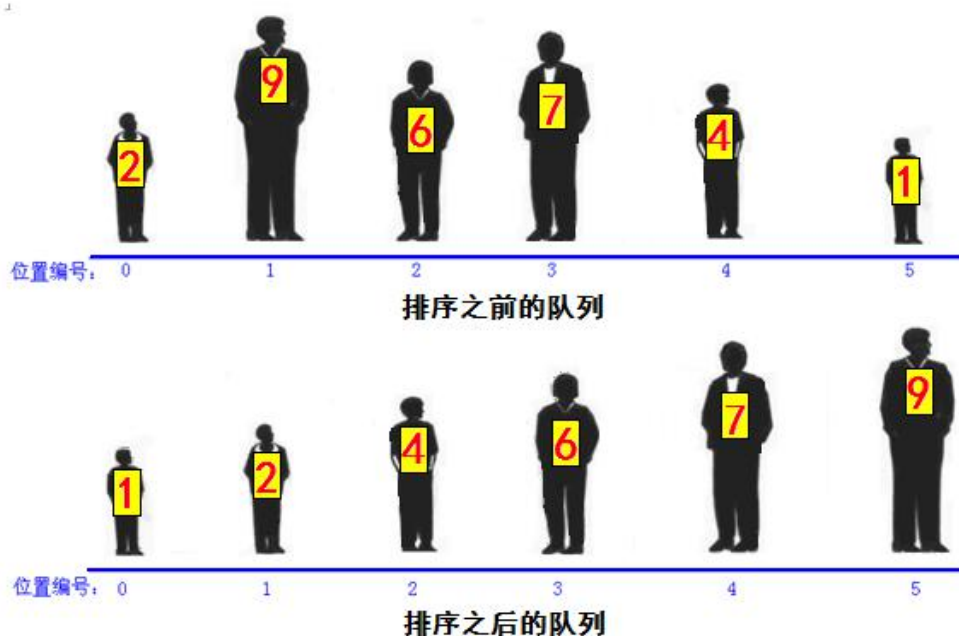
交换排序(**冒泡排序**、快速排序)

插入排序（直接插入排序、二分法插入排序、Shell 排序）

归并排序等。

排序有升序排列和降序排列之分，我们现在单讲**升序排列**：

我们主要讲解冒泡，选择，插入排序，当然在开发中因为性能问题，我们都不会自己写排序算法，**不过排序在笔试题里却是常客。**



若有下列 int 类型数组需要排序：

```
int[] arr = {2,9,6,7,4,1};
```

排序之后结果:

```
int[] arr = {1,2,4,6,7,9};
```

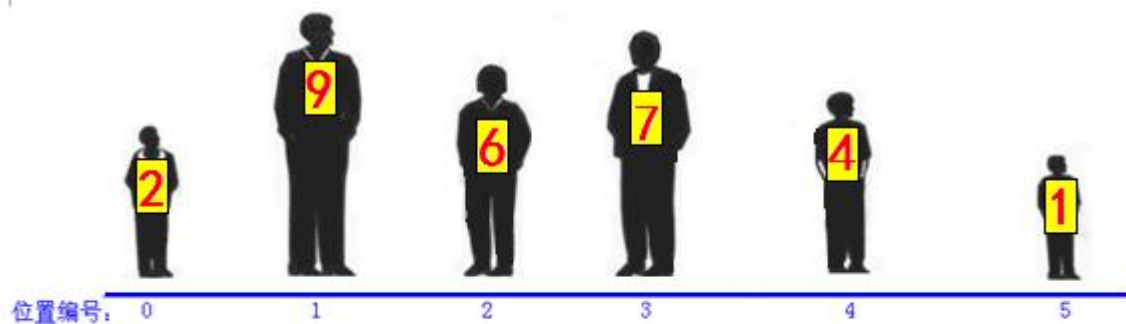
## 选择排序

### 选择排序(selection):

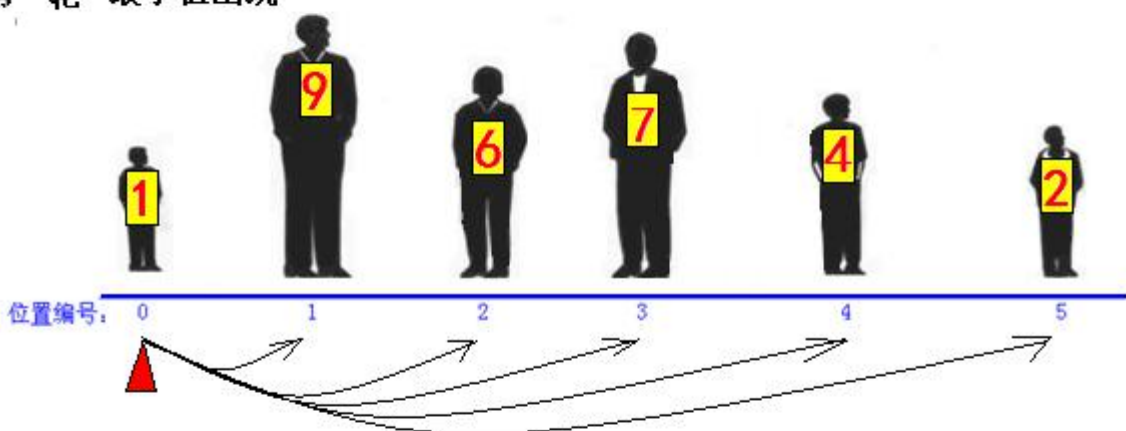
基本思路：**选定某个索引位置**，然后和后面元素依次比较，若大于则交换位置，经过第一轮比较排序后可得出最小值，然后使用同样的方法把剩下的元素逐个比较即可。

可以看出选择排序，第一轮会选出最小值，第二轮会选出第二小的值，直到最后。

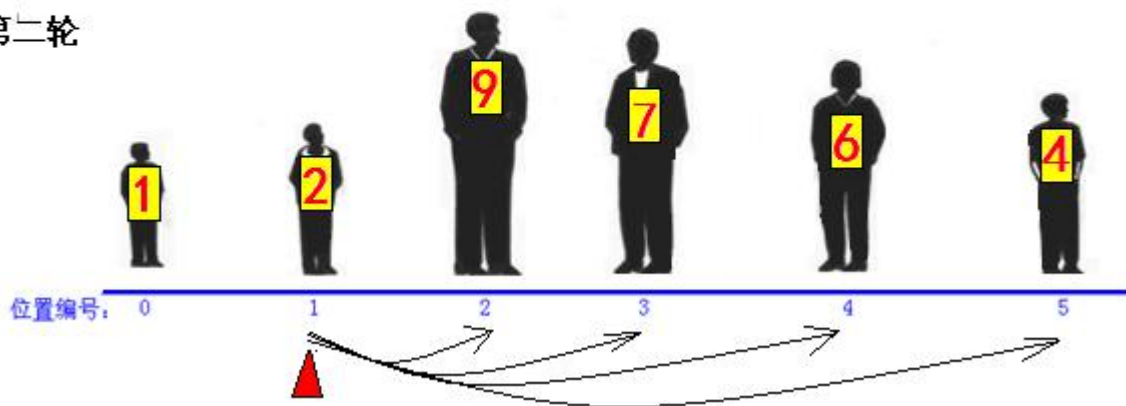
第一轮从 arr[0]和后面元素相比较，第二轮从 arr[1]和后面的元素相比较，依次类推。N 个数要进行 N-1 轮。选择排序每一轮只进行一次交换，相对于冒泡排序效率高一些。



第一轮 最小值出现



第二轮



## 冒泡排序

### 冒泡排序(bubble sort)：

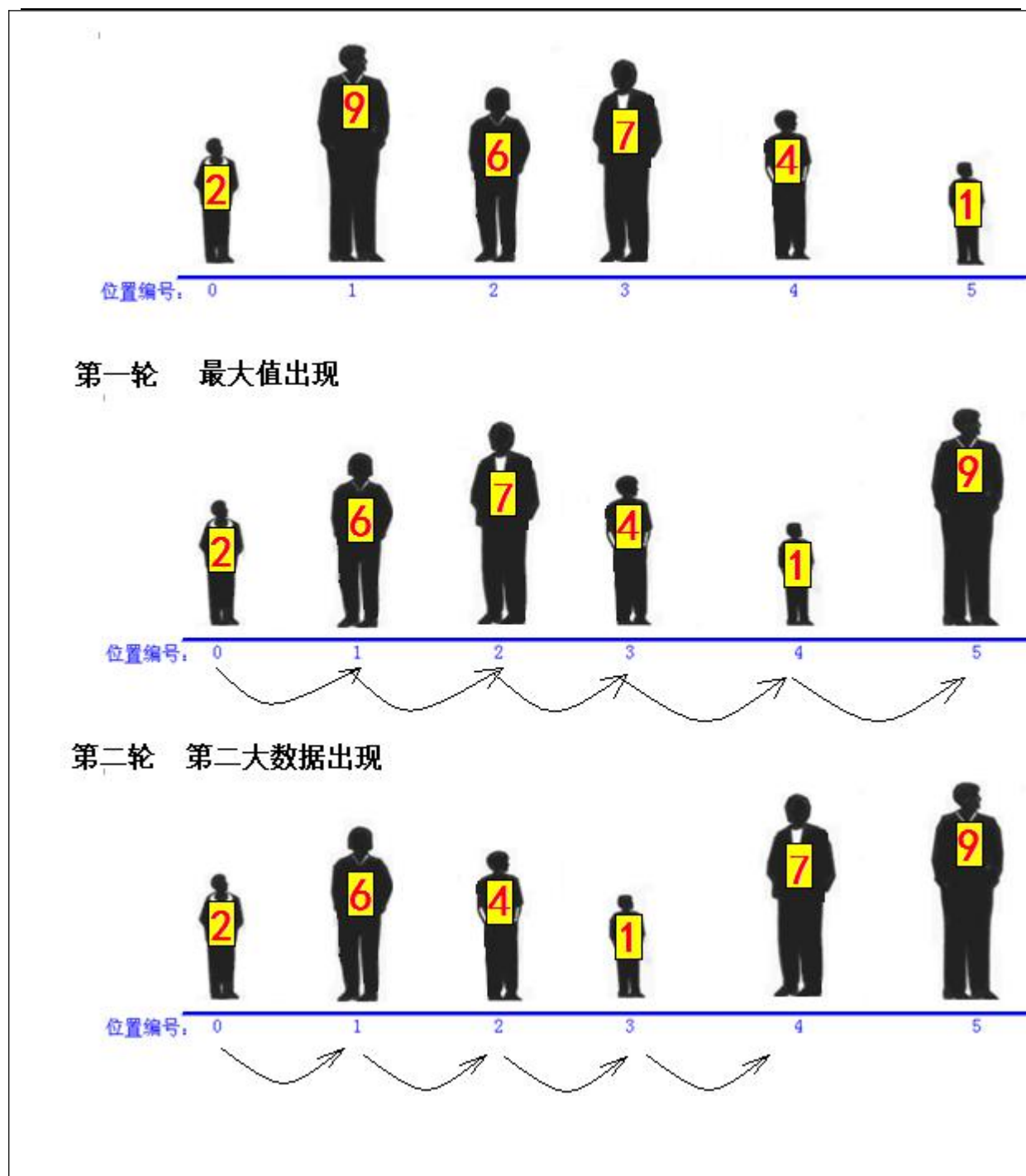
这是最简单的排序法，基本思路：对未排序的各元素从头到尾**依次比较相邻的两个元素**大小关系，若大于则交换位置，经过第一轮比较排序后可得出最大值，然后使用同样的方法把剩下的元素逐个比较即可。

可以看出若有 **N** 个元素，那么一共要进行 **N-1** 轮比较，第 M 轮要进行 N-M 次比较。（若 6 个元素，要进行 6-1 轮比较，第一轮比较 6-1 次，第三轮比较 6-3 次）。

第一轮比较之后:最大的数据浮现出来.

第二轮比较之后:第二大的数据浮现出来.





## 1.1 二维数组

在程序中可以通过一个数组来保存某个班级学生的考试成绩，试想一下，如果要统计一个学校

各个班级学生的考试成绩，又该如何实现呢？这时就需要用到多维数组，多维数组可以简单地理解为在数组中嵌套数组。在程序中比较常见的就是二维数组，接下来针对二维数组进行详细地讲解。

### 1.1.1 二维数组的定义格式

二维数组的定义有很多方式，接下来针对几种常见的方式进行详细地讲解，具体如下：

第一种方式：

```
int[][] arr = new int[3][4];
```

上面的代码相当于定义了一个 3\*4 的二维数组，即二维数组的长度为 3，二维数组中的每个元素又是一个长度为 4 的数组，接下来通过一个图来表示这种情况，如下图所示。

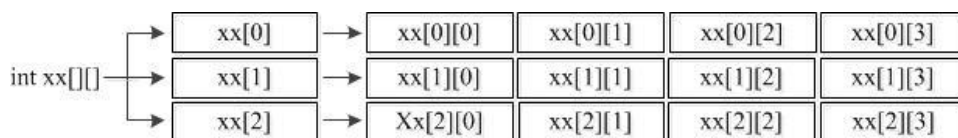


图 1-9 二维数组

第二种方式：

```
int[][] arr = new int[3][];
```

第二种方式和第一种类似，只是数组中每个元素的长度不确定，接下来通过一个图来表示这种情况，如下图所示。

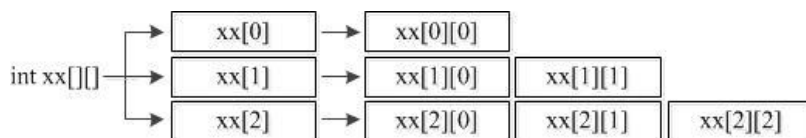


图 1-10 二维数组

第三种方式：

```
int[][] arr = {{1,2},{3,4,5,6},{7,8,9}};
```

上面的二维数组中定义了三个元素，这三个元素都是数组，分别为{1,2}、{3,4,5,6}、{7,8,9}，接下来通过一个图来表示这种情况，如图 2-54 所示。

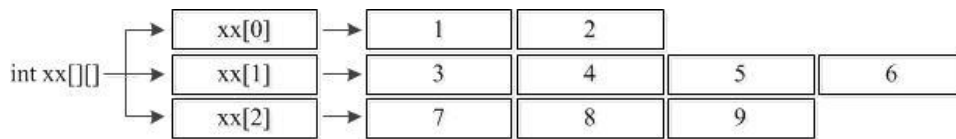


图 1-11 二维数组

对二维数组中元素的访问也是通过角标的方式，如需访问二维数组中第一个元素数组的第二个元素，具体代码如下：

```
arr[0][1];
```

### 1.1.1 二维数组元素的访问

操作二维数组时，经常需要获取数组中元素的值。接下来通过一个案例来演示如何获取数组中元素值，如下所示。ArrayDemo08.java

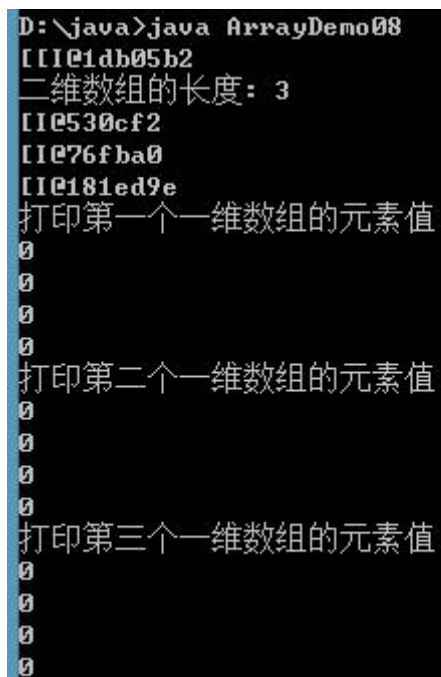
```
class ArrayDemo08 {  
    public static void main(String[] args){  
  
        //定义二维数组的方式  
        int[][] arr = new int[3][4];  
  
        System.out.println( arr );  
        System.out.println("二维数组的长度: " + arr.length);  
        //获取二维数组的 3 个元素  
        System.out.println( arr[0] );  
        System.out.println( arr[1] );  
        System.out.println( arr[2] );  
  
        System.out.println("打印第一个一维数组的元素值");  
        System.out.println( arr[0][0] );  
        System.out.println( arr[0][1] );//访问的为二维数组中第 1 个一维数组的第 2 个元素  
        System.out.println( arr[0][2] );  
        System.out.println( arr[0][3] );  
  
        System.out.println("打印第二个一维数组的元素值");  
        System.out.println( arr[1][0] );  
        System.out.println( arr[1][1] );  
        System.out.println( arr[1][2] );  
    }  
}
```

```
System.out.println( arr[1][3] );

System.out.println("打印第三个一维数组的元素值");
System.out.println( arr[2][0] );
System.out.println( arr[2][1] );
System.out.println( arr[2][2] );
System.out.println( arr[2][3] );

    }
}
```

运行结果如下图所示：



```
D:\java>java ArrayDemo08
[[I@1db05b2
二维数组的长度: 3
[[I@530cf2
[[I@76fba0
[[I@181ed9e
打印第一个一维数组的元素值
0
0
0
0
0
打印第二个一维数组的元素值
0
0
0
0
0
打印第三个一维数组的元素值
0
0
0
0
0
```

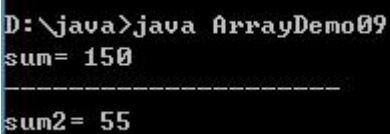
图 1-12 运行结果

## 二维数组元素遍历与数组元素累加和

学习完了数组元素的访问，我们来学习下数组的遍历及数组的元素累加和操作。

```
class ArrayDemo09 {  
    public static void main(String[] args){  
        //一维数组的求累加和并遍历  
        int[] arr = {10,20,30,40,50};  
        int sum = 0;  
        for (int i=0; i<arr.length; i++) {  
            //System.out.println(arr[i]);  
            sum += arr[i];  
        }  
        System.out.println("sum= " + sum);  
        System.out.println("-----");  
  
        //二维数组的求累加和并遍历  
        int[][] arr2 = { {1,2},{3,4,5},{6,7,8,9,10} };  
        int sum2 = 0;  
        for (int i=0; i<arr2.length; i++) {  
            for (int j=0; j<arr2[i].length; j++) {  
                //System.out.println(arr2[i][j])  
                sum2 += arr2[i][j];  
            }  
        }  
        System.out.println("sum2= " + sum2);  
    }  
}
```

运行结果如下图所示：



```
D:\java>java ArrayDemo09  
sum= 150  
-----  
sum2= 55
```

图 1-13 运行结果

## 知识点总结

- 数组

- 它是一个用来存储同一个数据类型多个元素的一个容器（数组长度是固定的，数组中存储的元素的数据类型要求一致）

- 格式：

格式 1：

数据类型[] 数组名 = new 数据类型[数组长度];

格式 2:

数据类型[] 数组名 = new 数据类型[]{元素值 1,元素值 2,...};

格式 3：

数据类型[] 数组名 = {元素值 1,元素值 2,...};

- 数组操作的常见问题：

NullPointerException: 空指针异常

ArrayIndexOutOfBoundsException: 数组越界异常

- 二维数组：

- 它是一个包含多个一维数组的数组
- 特点：二维数组中的每个元素都是一个一维数组
- 格式：

格式 1：

数据类型[][] 数组名 = new 数据类型[m][n];

m: 代表二维数组中一维数组的个数

n: 代表每个一维数组中元素的个数

格式 2 :

数据类型[][] 数组名 = new 数据类型[m][];

m: 代表二维数组中一维数组的个数

每一个一维数组通过赋值来确定数组长度

格式 3 :

数据类型[][] 数组名 = {{元素值 1 , 元素值 2,...},{元素值 1 , 元素值 2,...},...};