# 数组

## 引入&概念

* 需求

现在需要统计班上每一个同学的名字

假设班上有30名同学，用前面所学的知识，首先需要声明30个变量来分别记录每一个同学的名字，这样做会显得很麻烦。

为了解决这种问题，Java就提供了数组供我们使用。

* 数组定义&特点
  + 数组是存储多个变量(元素)的容器
  + 这多个变量的**数据类型要一致**
  + 数组既可以存储基本数据类型，也可以存储引用数据类型。

## 定义&初始化

### 数组的定义格式

* + 格式1：数据类型[] 数组名;(推荐使用) 如: int[] nums ;
  + 格式2：数据类型 数组名[];

注意：这两种定义做完了，数组中是没有元素值的。如何对数组的元素进行初始化呢?

### 数组初始化

所谓初始化：就是为数组中的数组元素分配内存空间，并为每个数组元素赋值。

注意：Java中的数组必须先初始化,然后才能使用。

### 动态初始化：

初始化时只指定数组长度，由系统为数组分配初始值。

* + 格式：数据类型[] 数组名 = new 数据类型[数组长度]; 如: int[] nums = new int[30] //数组长度其实就是数组中元素的个数。
  + 举例：

int[] arr = new int[3]; //解释：定义了一个int类型的数组，这个数组中可以存放3个int类型的值。

### 静态初始化

初始化时指定每个数组元素的初始值，由系统决定数组长度。

* + 格式

数据类型[] 数组名 = new 数据类型[]{元素1,元素2,…}; 如: int[] nums = new int[] {1,2,3,4,5}; 或者简化为: int[] nums = {1,2,3,4,5};

* + 举例

int[] arr = new int[]{1,2,3}; //解释：定义了一个int类型的数组，这个数组中可以存放3个int类型的值，并且值分别是1,2,3。

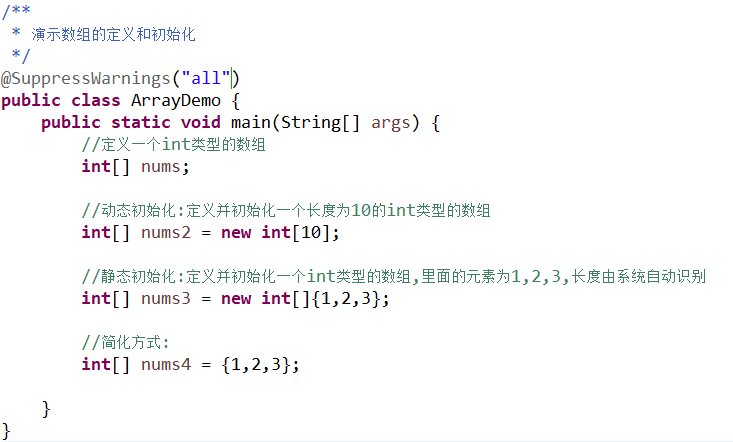
* + 简化格式

数据类型[] 数组名 = {元素1,元素2,...};

* + 举例

int[] arr = {1,2,3};

### 代码演示



## 访问数组元素

### 概念

* 索引

每一个存储到数组的元素，都会自动的拥有一个编号，从0开始，这个自动编号称为数组索引(index)（下标），可以通过数组的索引访问到数组中的元素。

* 格式

数组名[索引]

* 数组的长度属性

每个数组都具有长度，而且是不可以改变的，Java中赋予了数组的一个属性，可以获取到数组的长度

语句为： 数组名.length ，属性length的执行结果是数组的长度，int类型结果。

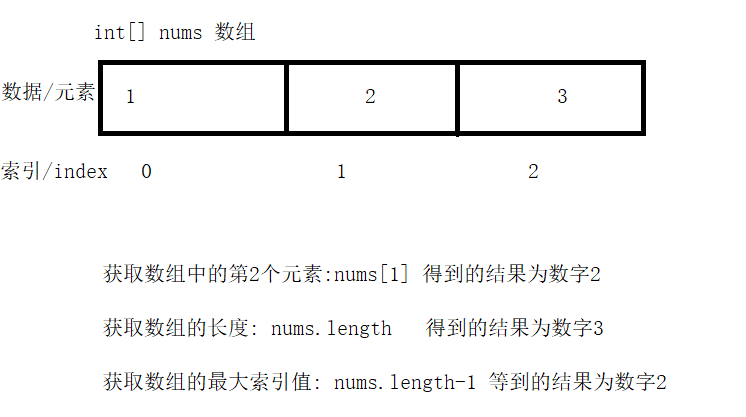
由次可以推断出，数组的最大索引值为数组名.length-1 。

数组的.length 与 String的.length()方法不同。

* 索引访问数组中的元素

数组名[索引]=数值，为数组中的元素赋值

变量=数组名[索引]，获取出数组中的元素



### 代码演示



## 练习-获取数组最值

### 需求

获取数组中的最大值最小值

### 代码实现



## 数组内存图(了解)

### 概念

内存是计算机中的重要原件，可以用来给程序临时存储数据。我们编写的程序，必须放进内存中才能运行，运行完毕后会清空内存。

Java虚拟机要运行程序，必须要对内存进行空间的分配和管理。

### Java的内存划分

为了提高运算效率，就对空间进行了不同区域的划分，因为每一片区域都有特定的处理数据方式和内存管理方式。

JVM的内存划分：

寄存器，给CPU使用，和我们开发无关。

本地方法栈：JVM在使用操作系统功能的时候使用，和我们开发无关。

方法区：存储可以运行的class文件。

堆内存：存储对象或者数组，new来创建的，都存储在堆内存。

方法栈：方法运行时使用的内存，比如main方法运行，进入方法栈中执行。

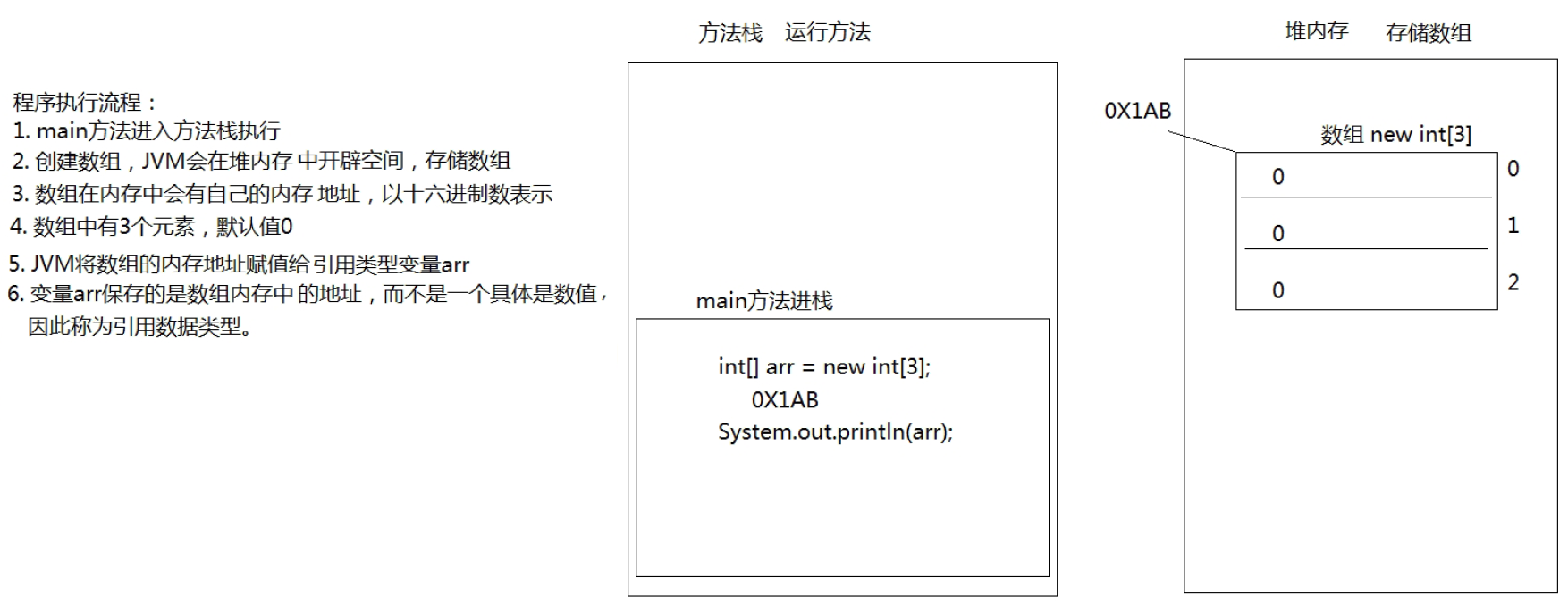
### 数组在内存中的存储

#### 一个数组的内存图

|  |
| --- |
| public static void main(String[] args) {  int[] arr = new int[3];  System.out.println(arr);//[I@5f150435  } |

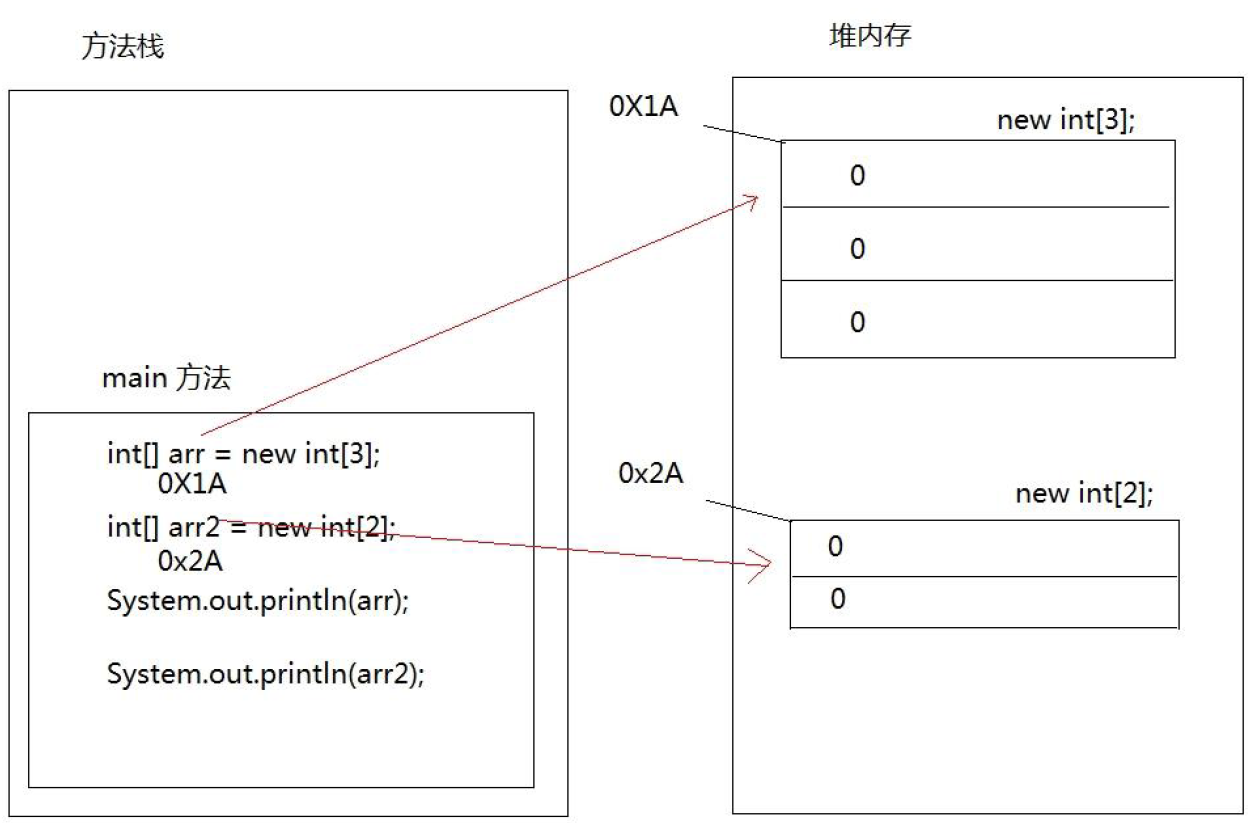
以上方法执行，输出的结果是[I@5f150435，这个是什么呢？是数组在堆内存中的地址。

注意：new出来的内容，都是在堆内存中存储的，而方法中的变量arr保存的是数组的地址。



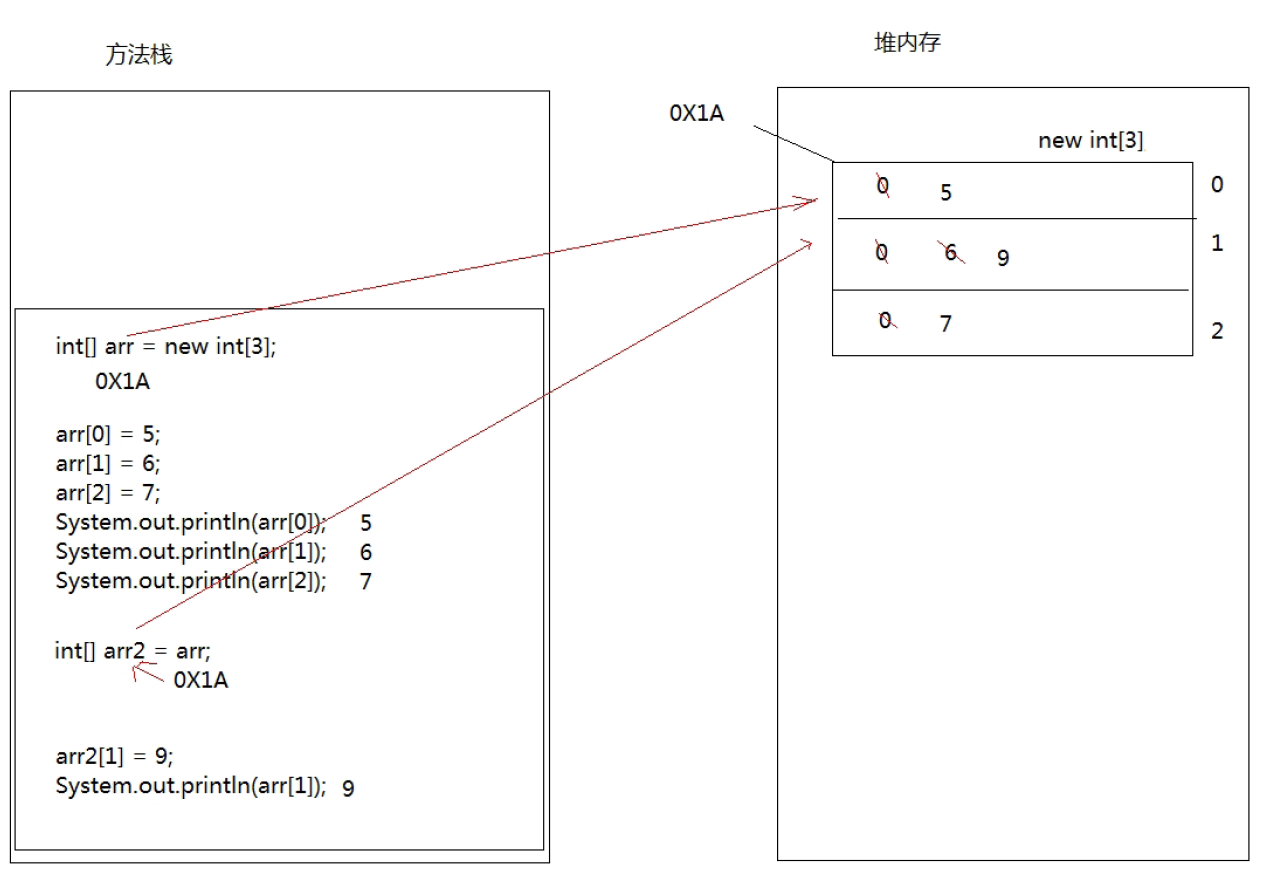
#### 两个数组的内存图

|  |
| --- |
| public static void main(String[] args) {  int[] arr = new int[3];  int[] arr2 = new int[2];  System.out.println(arr);  System.out.println(arr2);  } |



#### 两个变量指向一个数组

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // 定义数组，存储3个元素  **int**[] arr = **new** **int**[3];  // 数组索引进行赋值  arr[0] = 5;  arr[1] = 6;  arr[2] = 7;  // 输出3个索引上的元素值  System.***out***.println(arr[0]);//5  System.***out***.println(arr[1]);//6  System.***out***.println(arr[2]);//7  // 定义数组变量arr2，将arr的地址赋值给arr2  **int**[] arr2 = arr;  arr2[1] = 9;  System.***out***.println(arr2[1]);//9  System.***out***.println(arr[1]);//9  } |



## 数组常见错误

#### 数组越界异常(ArrayIndexOutOfBoundsException)

每个数组的索引都有一个范围，即0~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

|  |
| --- |
| 1. **public** **class** ArrayDemo { 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[4]=" + arr[4]); // 通过角标4访问数组元素 5. } 6. } |

上面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，关于异常在后面的课程中会有详细地讲解。

#### 空指针异常(NullPointerException)

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。

|  |
| --- |
| 1. **public** **class** ArrayDemo { 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表示没有执行任何内存空间 7. System.*out*.println("arr[0]=" + arr[0]); // 访问数组的元素 8. } 9. } |

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。

## 扩展阅读-数组反转

* 数组的反转：

数组中的元素颠倒顺序，例如原始数组为1,2,3,4,5，反转后的数组为5,4,3,2,1

* 实现思想：数组最远端的元素互换位置。

实现反转，就需要将数组最远端元素位置交换

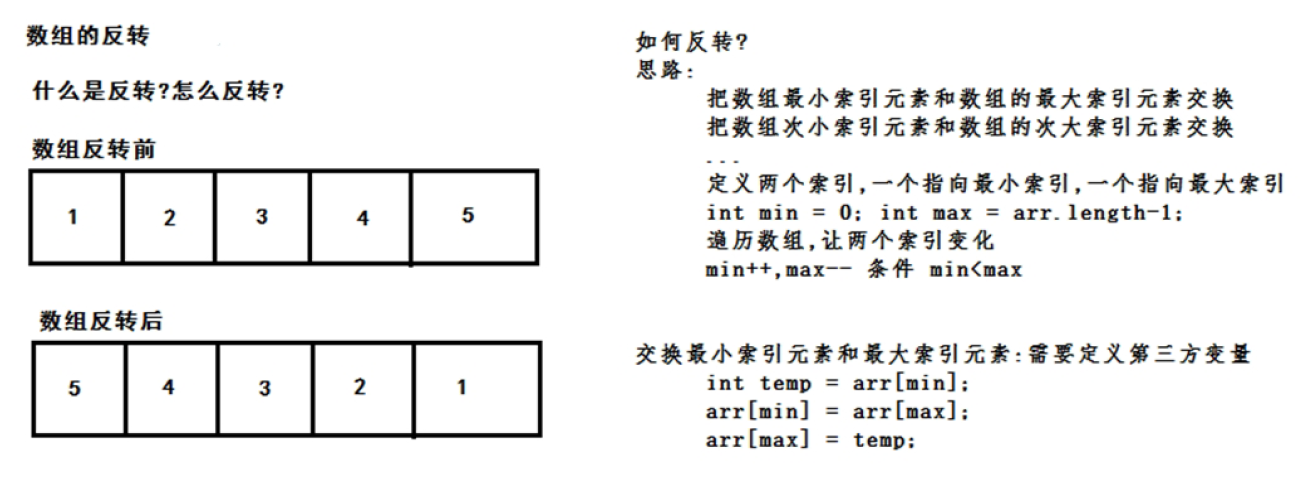
定义两个变量，保存数组的最小索引和最大索引

两个索引上的元素交换位置

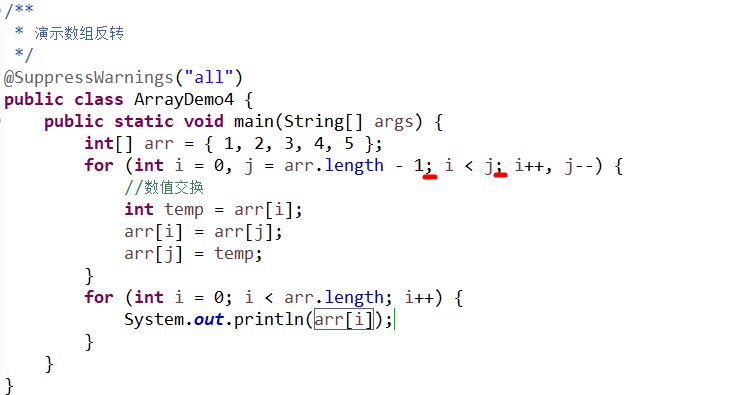
最小索引++，最大索引--，再次交换位置

最小索引超过了最大索引，数组反转操作结束

* 图解



* 代码实现



## 扩展阅读-数组排序

1. 冒泡排序法

重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。

步骤:

比较相邻的元素。如果第一个比第二个大，就交换他们两个。

对每一对相邻元素作同样的工作。这样，最后的元素会是最大的数。

针对所有的元素重复以上的步骤，除了最后一个。

持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

|  |
| --- |
| /\*\*  \* **@author** liulei  原来数组：  4 3 2 1  ---------  3 4 2 1  3 2 4 1 -------> 进行了3次比较  3 2 1 4  ---------  2 3 1 4 -------> 进行了2次比较  2 1 3 4  ---------  1 2 3 4 -------> 进行了1次比较  \*/  **public** **class** Demo11\_冒泡排序 {  **public** **static** **void** main(String[] args) {  **int**[] aa = {4,3,2,1};  **for**(**int** i = 0; i < aa.length - 1;i ++) {//控制轮数(轮数=长度-1,比如:4个数只需要把最大的3个排好)  **for**(**int** j = 0;j < aa.length - 1 - i;j ++) {//控制每轮比较次数 -1保证j+1不越界;-i是优化,因为排完i轮后,后面的i个是已经排好序的不用比了  **if**(aa[j] > aa[j + 1]) {  **int** temp = aa[j];  aa[j] = aa[j + 1];  aa[j + 1] = temp;  }  }  }    **for**(**int** i = 0;i < aa.length;i ++) {  System.***out***.println(aa[i]);  }  }  } |

1. 选择排序法

每一次从待排序的数据元素中选择出最小（或最大）的一个元素，存放在序列的起始位置，直到全部待排序的数据元素排完

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **int**[] aa = {4,3,2,1};  **for** (**int** i = 0; i < aa.length-1; i++) {//轮数  **for** (**int** j = i+1; j < aa.length; j++) {  **if** (aa[i] > aa[j]) {//拿当前位置元素和后面的比较,如果比后面的大,互换位置,这样可以选择出较小的放到当前位置  **int** temp = aa[i];  aa[i] = aa[j];  aa[j] = temp;  }  }  }  **for**(**int** i = 0;i < aa.length;i ++) {  System.***out***.println(aa[i]);  }  } |

1. 插入排序法

从第一个元素开始，该元素可以认为已经被排序

取出下一个元素，在已经排序的元素序列中从后向前扫描

如果原元素大于取出的元素，将取出的元素插入该位置,原有元素后移

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **int**[] aa = {4,3,2,1};  **for** (**int** i = 1; i < aa.length; i++) {  **int** j = i - 1;  **int** temp = aa[i];  **for** (; j >= 0 && temp < aa[j]; j--) {//只要没比完并且取出的元素比该位置元素小,就一直循环,目的是将大于取出元素的全部后移  aa[j + 1] = aa[j]; // 将大于temp的值整体后移一个单位  }  aa[j + 1] = temp;//一轮比较之后,再把取出的元素放到合适的位置  }  **for**(**int** i = 0;i < aa.length;i ++) {  System.***out***.println(aa[i]);  }  } |

## 数组操作工具类Arrays

|  |
| --- |
| **int**[] aa = {1,4,3,5,2};  Arrays.*sort*(aa); // 数组排序，默认升序  String str = Arrays.*toString*(aa); // 把数组转换成字符串  System.***out***.println(str); |