# day05

## 数组

### 数组的概述

1. 数组 : 数组就是一个容器, 存储很多的数据, 数组中存储的是相同数据类型的数据, 数组是定长容器, 定义数组就必须给定数组中存储的数据的个数
2. 数组存在意义:

a : 没有数组, 如果有很多的数据, 需要定义出很多变量, 但是多个变量彼此之间没有关联, 没有关系性, 如此这些数据无法进行排序, 定位

b : 有数组, 数组就是一个容器, 给这个容器起名, 可以向容器中存储很多数据, 数组容器对于存储的每一个数据都有对应的编号, 根据存入的顺序编号从0开始, 以此类推, 以后可以通过 容器名 + 数据编号, 定位每一个数据, 从而操作数据

### 数组的定义

1. 数组定义语法结构:

元素数据类型[] 数组名 = new 元素数据类型[数组容器大小];

1. 解释 :
2. 元素数据类型 : 表示数组中存储的元素是什么类型的数据
3. [] : 中括号表示是一维数组标识
4. 数组名 : 变量名, 符合标识符规范, 小驼峰原则
5. = : 赋值运算符, 表示将数组在内存中地址值赋值给等号左边的数组名
6. new : 关键字, 表示新建, 为数组容器在堆内存中开辟空间
7. 元素数据类型 : 与前面的数据类型保持一致即可
8. [数组容器大小] : 表示容器中可以存储多少个数据

举例 : int[] arr = new int[3]; // 表示创建出一个数组, 名字叫做arr , 这个数组中可以存储3个int类型整数数据

1. 数组变量结果打印: 不需要记忆地址具体是多少, 只需要知道数组结果代表的是一块内存空间地址即可

上述arr数组名打印结果: [I@15db9742,称为数组的内存地址

[ : 表示这是一个一维数组

I : 大写字母I, 表示数组中存储的数据类型是int类型整数, 大写字母I就是int类型的首字母大写表示

@ : 没有含义, 只是一个分隔符而已, 将结果分隔成前后两个部分

15db9742 : 十六进制数(数字中带有字母就是16进制数, 16进制逢16进1), 计算机世界中, 内存地址封装成人类可以识别的一种数字形式进行表现, 表现结果是一个16进制数

0 1 2 3 4 5 6 7 8 9 a b c d e f

代码

|  |
| --- |
| package com.ujiuye.array;  public class Demo01\_数组定义 {  public static void main(String[] args) {  int i = 15;    int[] arr = new int[3];  System.out.println(arr);//[I@15db9742 // 类比 : 1号楼2116    double[] arr1 = new double[5];  System.out.println(arr1);//[D@6d06d69c    String[] arrName = new String[63];  System.out.println(arrName);//[Ljava.lang.String;@7852e922  }  } |

### 数组的初始化

1. 数组初始化 : 表示给数组中的元素进行赋值(初始化就理解成赋值)
2. 动态初始化 : 随着代码的运行, 才能知道数组中的每一个元素值是多少
3. 数组元素的赋值:

数组名[索引] = 值;

索引 : 表示数组中对于每一个元素的编号, 从0开始, 到数组长度-1结束, 索引有其他名称, 角标, 脚标, 下标

int[] arr = new int[3]; // arr数组的索引范围0-2

1. 数组中元素获取:

变量 = 数组名[索引];

注意 : 创建出一个数组容器时, 如果没有给数组中的元素进行手动赋值, JVM虚拟机会默认给数组中的每一个元素进行默认的赋初值动作

根据每一个数组容器中存储的元素数据类型不同, 决定初始值

a : 整数 0

b : 浮点数 0.0

c : 字符 ‘ ’

d : 布尔类型 false

e : 引用数据类型 null(空)

1. 静态初始化 : 定义一个数组时, 已经知道了数组中的每一个元素值, 可以使用静态初始化方式

元素数据类型[] 数组名 = new 元素数据类型[]{罗列出数组中每一个元素};

举例 : int[] arr = new int[]{12,13,9,0,-7};

a : 静态初始化后面[]中空的, 不要写数组长度

b : 数组中的元素罗列在大括号中, 多个元素之间使用 , 进行分隔

c : 简写方式:

元素数据类型[] 数组名 = {罗列出数组中每一个元素};

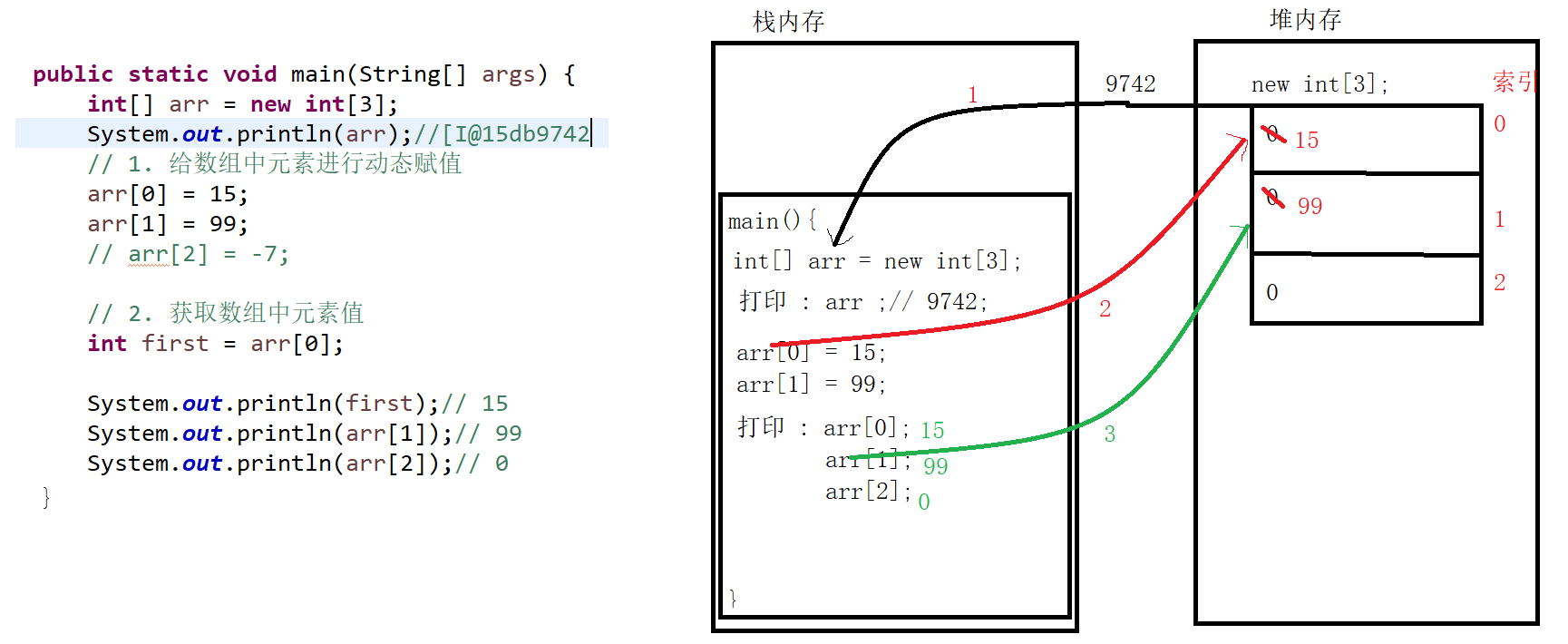
举例 : int[] arr1 = {12,13,9,0,-7}; // 0-4

代码

|  |
| --- |
| package com.ujiuye.array;  public class Demo02\_数组初始化 {  public static void main(String[] args) {  int[] arr = new int[3];  // 1. 给数组中元素进行动态赋值  arr[0] = 15;  arr[1] = 99;  // arr[2] = -7;    // 2. 获取数组中元素值  int first = arr[0];    System.out.println(first);// 15  System.out.println(arr[1]);// 99  System.out.println(arr[2]);// 0    double[] arr1 = new double[5];  System.out.println(arr1[2]);// 0.0    String[] arr2 = new String[2];  System.out.println(arr2[0]);// null    // 3. 数组静态初始化  int[] arr3 = new int[] {12,13,6,-7};  int[] arr4 = {12,13,6,-7};  System.out.println(arr4[2]);// 6    arr4[2] = 99;  System.out.println(arr4[2]);// 99  }  } |

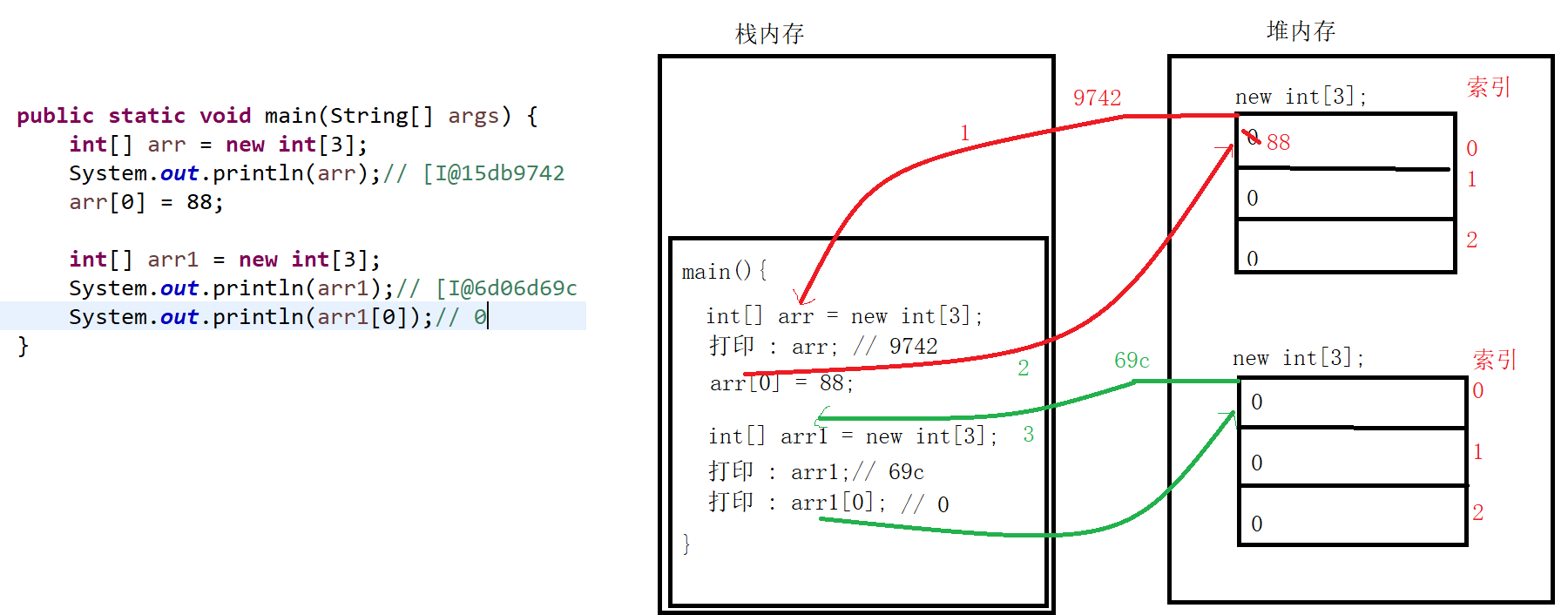
### 数组在内存中的理解

1. 数组本身也是一种数据类型, 引用数据类型: 例如 : int[] 表示整数类型数组
2. Java代码在JVM虚拟机中运行, 而JVM虚拟机进入到内存后, 占有很大一块内存空间, JVM虚拟机将其占有内存空间划分成5块区域:
3. 栈内存 : 代码中的方法运行, 进栈, 方法运行完毕释放掉占有的内存空间, 弹栈死亡
4. 堆内存 : 引用数据类型通过new关键字开辟空间, 在堆内存中, 引用数据类型中数据存储在堆内存中
5. 方法区(数据共享) : 一个java类使用, 对应的.class字节码文件存储在方法区中, 存储静态成员, 字符串常量
6. 一个数组在内存中理解

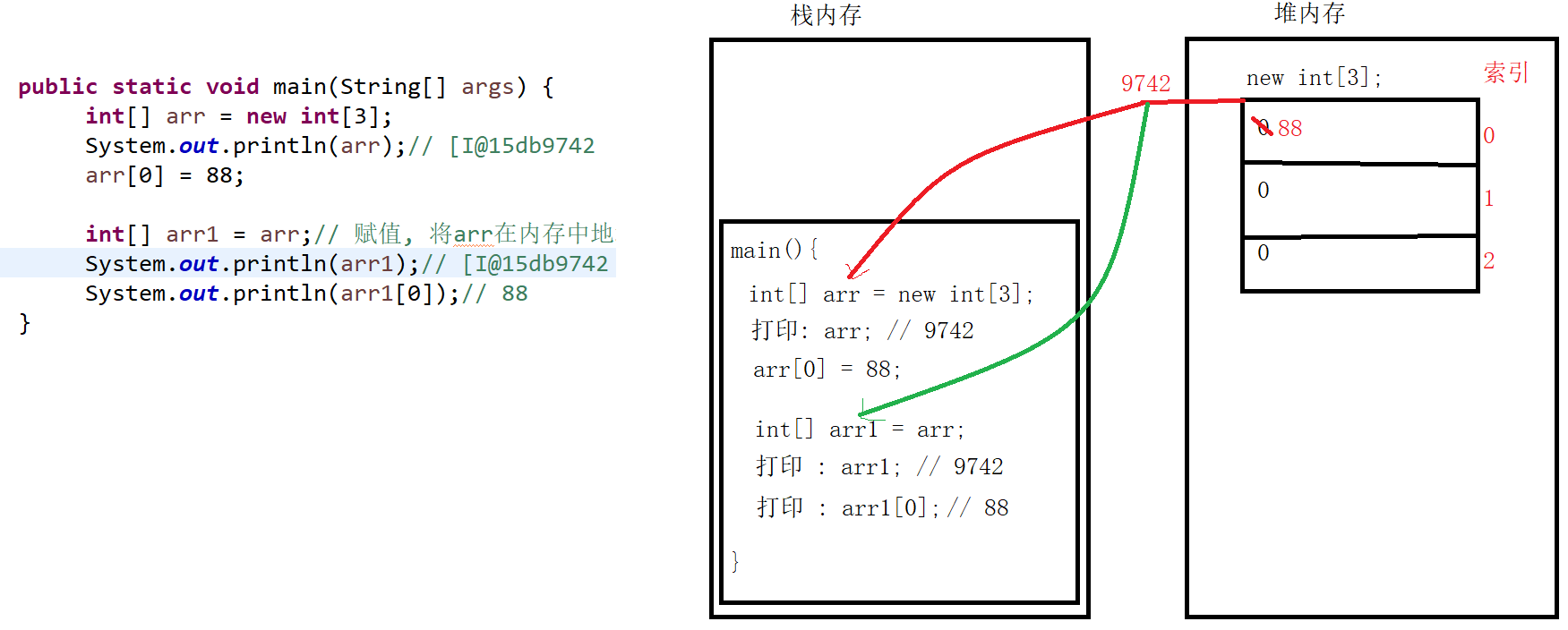


1. 两个数组在内存中理解:

引用数据类型每次通过new, 都是在内存中开辟新的地址空间



1. 两个数组引用指向同一块内存空间



### 数组中的异常

1. 异常(Exception) : 表示在Java代码编译和运行过程中, 出现了不正确的, 不正常的, 不符合实际场景情况, 统称为异常
2. 数组中常见的异常有两种 :

a : 数组索引越界异常

b : 空指针异常

#### 数组索引越界异常

1. 索引越界异常:

ArrayIndexOutOfBoundsException

数组索引 超出 边界 异常

1. 发生索引越界异常问题原因:

使用的索引在数组中不存在

1. 解决(避免)索引越界异常发生:
2. 操作的数组索引在正确范围内 : 0----数组长度-1
3. 因此使用数组索引时 : 需要验证, 索引 >= 0 && 索引 <= 数组长度-1,
4. 需要获取到数组的长度数据, 数组类型中, 有属性, length 可以获取到数组中的元素个数(长度)

int 数组长度 = 数组名.length;

代码

|  |
| --- |
| package com.ujiuye.array;  public class Demo05\_数组索引越界异常 {  public static void main(String[] args) {  int[] arr = {12,13,0,99,88,-7};// 元素有6个, 索引范围0-5  //ArrayIndexOutOfBoundsException  //数组 索引 超出 边界 异常  //arr[-1] = 72;    // arr[6] = 99; 越界  System.out.println(arr.length);// 6    int ele = getArrayElement(arr,5);  System.out.println(ele);// -7  }    // 定义出一个方法功能 : 获取到指定int[]中某一个索引位置元素值  public static int getArrayElement(int[] arr, int index) {  // 判断: 使用index索引在arr中存在才能使用  // index >= 0 index <= 5  // index >= 0 index < 6  if(index >= 0 && index < arr.length) {  return arr[index];  }else {  System.out.println(index + "索引不在arr数组正确索引范围之内, 无法操作, 返回默认值为0");  return 0;  }  }  } |

#### 空指针异常

1. 引用数据类型的默认值可以设置为null, null表示空

栈内存中的变量引用, 没有指向任何堆内存中地址空间

1. 空指针异常

NullPointerException

1. 发生空指针异常的原因:

当引用数据类型值设置为null(前提), 证明这个变量引用没有记录任何堆内存地址, 但仍然想通过这个变量引用访问堆内存中数据, 就会报出空指针异常

1. 解决(避免)空指针异常方案:

引用数据类型的值不要设置为null, 使用数组时, 验证数组的值不为null然后再继续使用

代码

|  |
| --- |
| package com.ujiuye.array;  public class Demo06\_空指针异常 {  public static void main(String[] args) {  int[] arr = null;  int[] arr1 = {};  // NullPointerException  // 空 指针 异常  // System.out.println(arr.length);    /\* if(arr != null) {  System.out.println(arr[0]);  }else {  System.out.println("数组不能设置为null,因为null数组无法操作");  }\*/  getArrayEle(arr,1);  }    // 定义出一个方法功能 : 获取到指定int[]中某一个索引位置元素值  public static int getArrayEle(int[] arr, int index) {  // 实际开发中, 会先验证数组存在, 再验证数组中有元素  if(arr != null && arr.length > 0) {  if(index >= 0 && index < arr.length) {  return arr[index];  }else {  System.out.println(index + "索引不在arr数组正确索引范围之内, 无法操作, 返回默认值为0");  return 0;  }  }else {  System.out.println("数组不能设置为null,因为null数组无法操作");  return 0;  }  }  } |

### 数组相关操作

#### 数组遍历

1. 数组遍历 : 将数组中的元素一个一个获取到
2. 分析 :
3. 使用 数组名[索引]--->获取到数组中的某一个元素
4. 思考 : 如果可以将一个数组中每一个索引都获取到(利用循环从0开始,到数组长度-1结束, 循环中的每一个数据就表示数组中每一个索引), 再结合数组名[索引]方法, 就可以获取到数组中的每一个元素

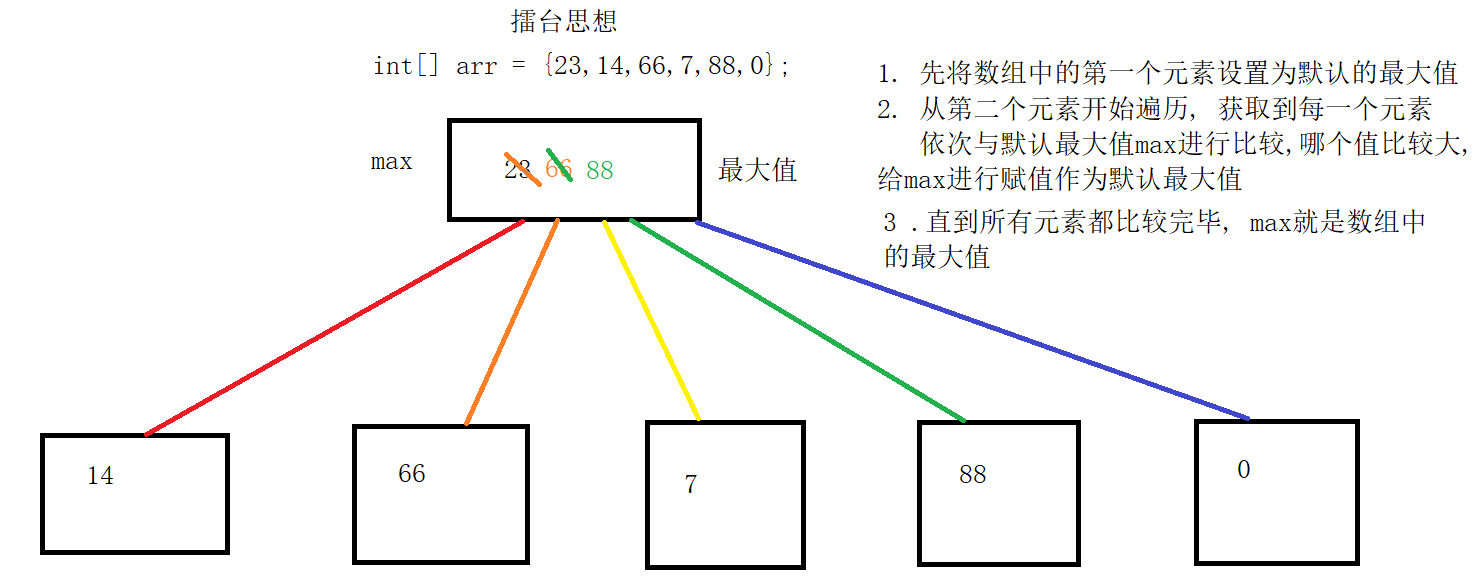
代码

|  |
| --- |
| package com.ujiuye.array;  public class Demo07\_数组遍历 {  public static void main(String[] args) {  int[] arr = {66,1234,0,-98};  getAllElementFromArray(arr);  }    // 定义出一个方法 : 功能就是可以获取到int[]数组中的每一个元素(遍历)  public static void getAllElementFromArray(int[] arr) {  if(arr != null && arr.length > 0) {  // 1. 将数组索引位作为循环依据  for(int index = 0; index <= arr.length-1; index++) {  // 2. 在循环内部, 可以使用数组名[index]获取数组元素  int ele = arr[index];  System.out.println(ele);  }  }else {  System.out.println("数组为null或者数组中没有元素, 那么无效遍历");  }  }  } |

#### 数组求最值

需求 : 定义出一个int[]数组, 求出数组中的最大值

分析 :



代码

|  |
| --- |
| package com.ujiuye.array;  public class Demo08\_数组求最值 {  public static void main(String[] args) {  int[] arr = {112,13,55,7,66,88};  // int[] arr = {};    if(arr != null && arr.length > 0) {  // 1. 设置一个默认最大值, 设置0索引位默认值  int max = arr[0];  // 2. 获取到除了0索引之外的剩余所有元素  for(int index = 1; index < arr.length; index++) {  int ele = arr[index];  // 3. 将数组中的每一个元素与默认最大值max进行比较, 将较大值赋值给max  if(ele > max) {  max = ele;  }  }  System.out.println(max);    }else {  System.out.println("数组为null或者数组中没有元素, 不能求最值");  }  }  } |

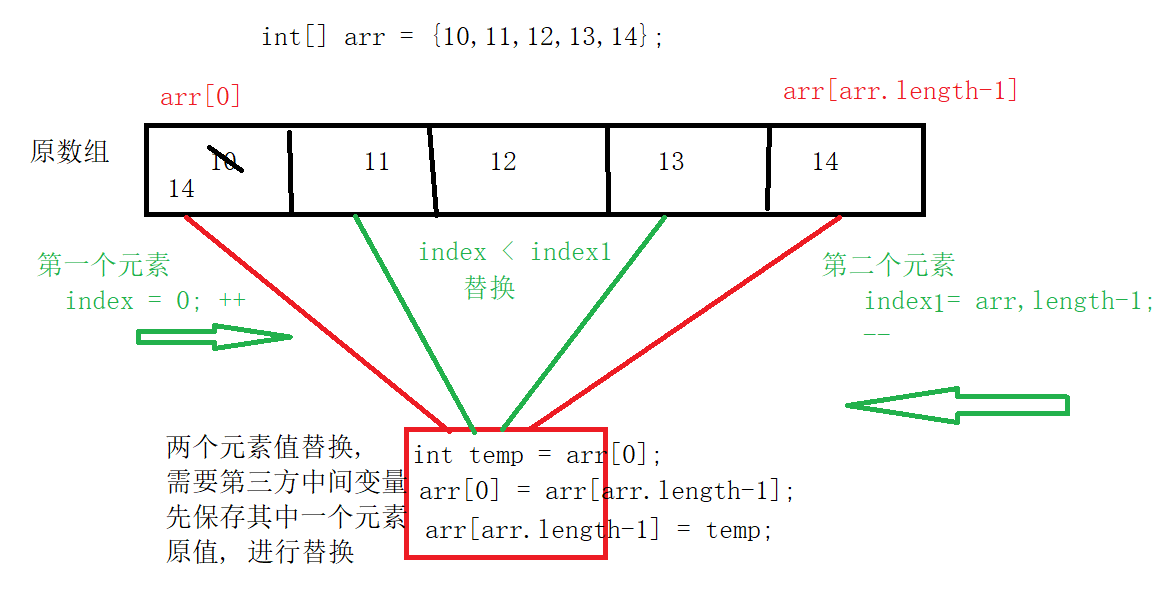
#### 数组的反转

定义一个int[]数组, 将数组中的元素进行逆序的排列(即 : 第一个元素与最后一个元素位置交换, 第二个与倒数第二个元素交换...)

举例 : int[] arr = {1,2,3,4,5};

反转后的效果 : {5,4,3,2,1};

分析:

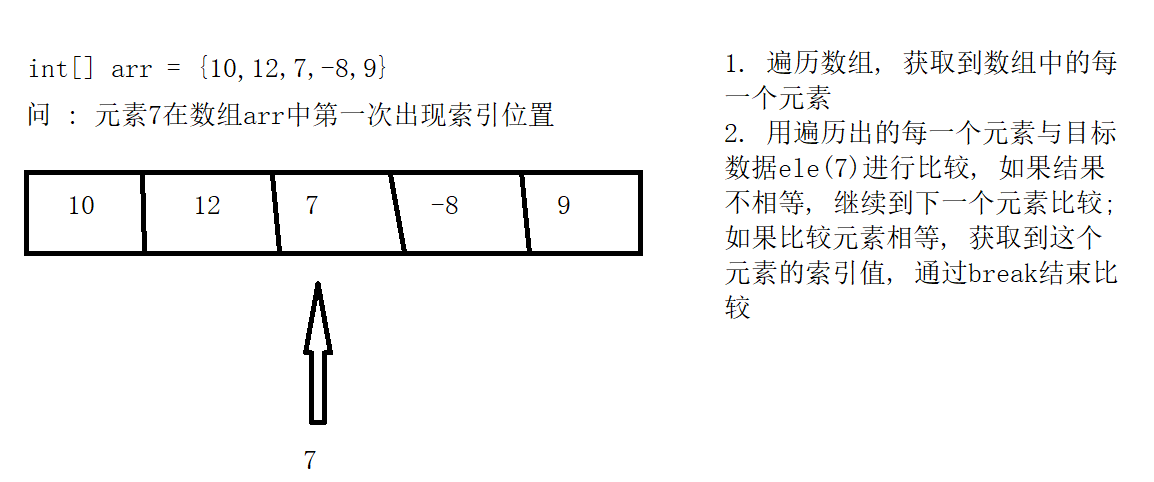


代码

|  |
| --- |
| package com.ujiuye.array;  public class Demo09\_数组反转 {  public static void main(String[] args) {  int[] arr = {1,2,3,4,5};    for(int beginIndex = 0, endIndex = arr.length-1 ;  beginIndex < endIndex; beginIndex++, endIndex--) {  // 两个元素替换  int temp = arr[beginIndex];  arr[beginIndex] = arr[endIndex];  arr[endIndex] = temp;  }  getAllArrayEle(arr);  }    public static void getAllArrayEle(int[] arr) {  if(arr != null && arr.length > 0) {  for(int index = 0 ; index < arr.length ; index ++) {  System.out.println(arr[index]);  }  }  }  } |

#### 数组元素的查找

需求: 定义出一个int[]数组, 问元素ele在数组中第一次出现索引位置, 如果数组中没有ele元素,返回-1



代码

|  |
| --- |
| package com.ujiuye.array;  public class Demo10\_数组元素的查找 {  public static void main(String[] args) {  int[] arr = {12,99,88,7,88,7};  //查找元素7在数组arr中第一次出现的索引位置  int ele = 7;  if(arr != null && arr.length > 0) {  // endIndex表示元素ele在数组中出现的索引位置,初始值给-1,初始认为找不到  // int endIndex = -1;  // 1. 遍历数组arr, 获取到数组中的每一个元素  for(int index = 0; index < arr.length; index++) {  int arrayEle = arr[index];  // 2. 将数组中元素与目标数据进行比较  if(arrayEle == ele) {// 找到了  //endIndex = index; 给endIndex赋值  System.out.println(index);  break;  }  // 3. 当循环执行到最后一次, 还没有结束循环,证明没有找到元素ele  if(index == arr.length-1) {  System.out.println("-1");  }  }    //System.out.println(ele + "在数组arr中第一次出现索引位置为: "+endIndex);    }  }  } |