数组

一 内容回顾

（列举前一天重点难点内容）

1.1 教学重点:

```
1.掌握方法的语法结构,包括参数,返回值.
2.掌握方法的使用
3.掌握重载的原理以及使用
4.掌握递归的原理以及使用
```

1.2 教学难点:

```
1.递归的实现
```

二 教学目标

```
1.数组的基本使用
2.函数和数组的联合使用
3.理解址传递
4.熟练掌握冒泡排序,选择排序
5.熟练掌握二分查找
6.Arrays工具类
7.了解快速排序
8.了解归并排序
9.了解二维数组
```

三 教学导读

3.1. 为什么要使用数组?

我们来看一个现实当中的问题:

- 如何存储100名学生的成绩
  - 办法：使用变量存储，重复声明100个double类型的变量即可。
  - 缺点：麻烦，重复操作过多。
- 如何让100名学生成绩全部+1
  - 办法：100个变量重复相同操作，直到全部完毕。
  - 缺点：无法进行统一的操作。

3.2. 数组是什么?

数组， 是一个数据容器。 可以存储若干个相兼容的数据类型的数据。

在上述案例中， 存储100名学生的成绩， 可以用数组来完成。 将这100个成绩存入一个数组中。 此时对这些数据进行统一操作的时候， 直接遍历数组即可完成。

四 教学内容

4.1. 数组概述(会)

4.1.1. 数组定义

```
1.数组中可以存储基本数据类型的数据， 也可以存储引用数据类型的数据。
2.数组的长度是不可变的,数组的内存空间是连续的。 一个数组一旦实例化完成， 长度不能改变。
```

4.1.2. 比较简单和引用数据类型

```
1.引用数据类型里面存储的是地址,并且这个地址是十六进制的数.简单数据类型存储的是值,是十进制的
2.对于简单数据类型,直接在栈区的方法中开辟一块空间存储当前的变量,将要存储的数据直接放在这块空间里
```

4.2. 数组的声明(会)

4.2.1. 声明数组

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 数组的声明
 */
public class Test {
    public static void main(String[] args) {
        int a = 5;
        // 声明一个数组,  存储若干个double类型的数据
        double[] array1;
        // 声明一个数组,  存储若干个int类型的数据
        int[] array2;
        // 声明一个数组,  存储若干个String类型的数据
        String[] array3;

    }
}
```

4.2.2. 数组的实例化

实例化数组： 其实就是在内存中开辟空间， 用来存储数据。

```java
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 数组的声明
 */
public class Test {
    public static void main(String[] args) {
        int a = 5;
        // 实例化了一个数组， 可以存储5个数据
        // 此时数组中的元素就是默认的5个0
        int[] array1 = new int[5];
        // 实例化了一个数组， 默认存储的是 1, 2, 3, 4, 5
        // 此时数组的长度， 由这些存储的数据的数量可以推算出来为5
        int[] array2 = new int[] { 1, 2, 3, 4, 5 };
        // 实例化了一个数组， 默认存储的是 1， 2， 3， 4， 5
        // 相比较于第二种写法， 省略掉了 new int[]
        int[] array3 = { 1, 2, 3, 4, 5 };
    }
}
```

4.2.3. 数组引用

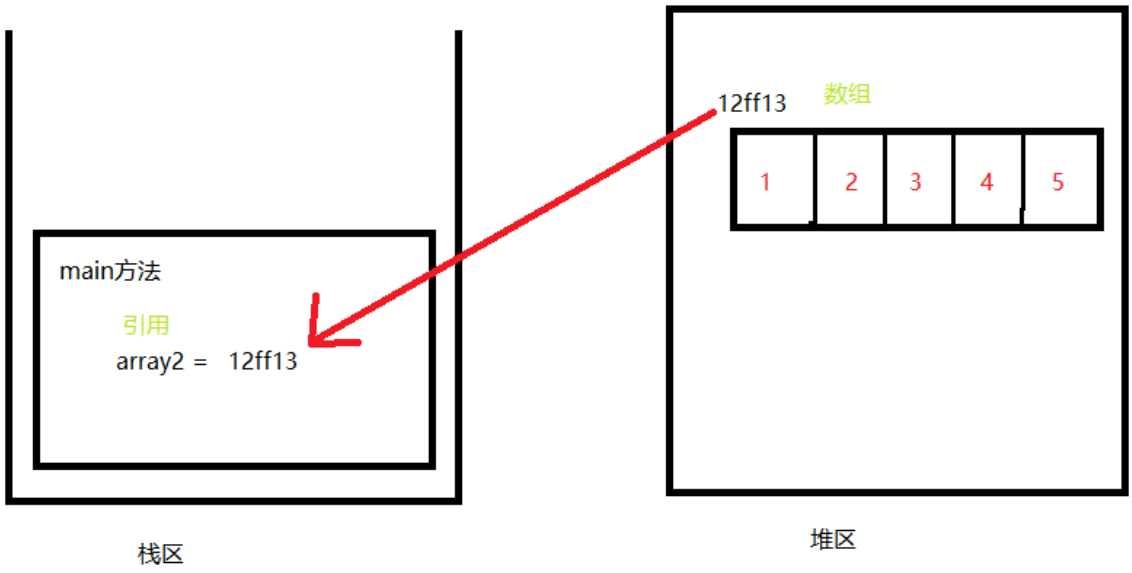
数组的实例化的时候， 需要使用到关键字**new**

以后但凡是遇到了new, 都表示在堆上开辟空间!

数组， 其实是在堆上开辟的连续的空间。 例如 `new int[5]` ， 就是在堆上开辟5个连续的4字节空间。

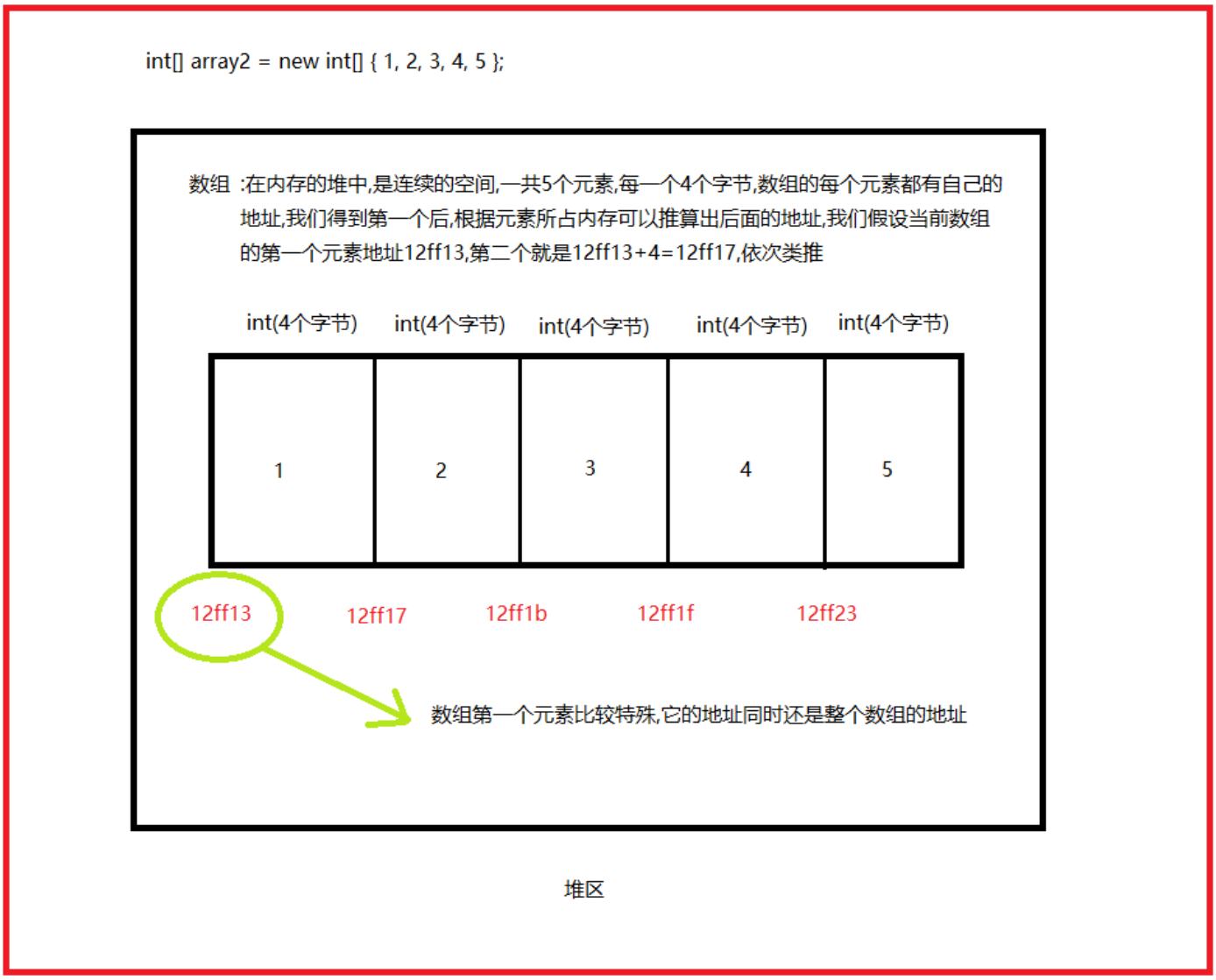
然后， 将堆上的内存地址， 赋值给栈上的变量array(引用)。

```
public class Test {
    public static void main(String[] args) {
        // 实例化了一个数组， 默认存储的是 1, 2, 3, 4, 5
        // 此时数组的长度， 由这些存储的数据的数量可以推算出来为5
        int[] array2 = new int[] { 1, 2, 3, 4, 5 };
        // 实例化了一个数组， 默认存储的是 1， 2， 3， 4， 5
    }
}
```



说明:图中的12ff13是数组的地址,main方法中的变量array2(引用)通过保存这个地址指向数组

- 关于内存地址的说明(扩展)



1. 1.引用地址(包括数组地址),是一个十六进制的数.
2. 2.在内存的堆中,是连续的空间,上图中的数组中一共5个元素,每一个4个字节,数组的每个元素都有自己的地址,我们得到第一个后,根据元素所占内存可以推算出后面的地址.
3. 3.数组第一个元素比较特殊,它的地址同时还是整个数组的地址

4.3. 数组的下标(会)

4.3.1. 下标的概念

下标， 就是数组中的元素在数组中存储的位置索引。

注意： 数组的下标是从0开始的， 即数组中的元素下标范围是 [0, 数组.length - 1]

4.3.2. 访问数组元素

访问数组中的元素， 需要使用下标访问。

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 数组的元素访问
 */
public class Test {
    public static void main(String[] args) {
        // 实例化一个数组
        int[] array = { 1, 2, 3, 4, 5 };
        // 访问数组中的元素
        array[2] = 300;          // 将数组中的第2个元素修改成300， 此时数组中的元素是 [ 1, 2, 300, 4, 5 ]
        System.out.println(array[2]); // 获取数组中的第2个元素， 此时的输出结果是 300
    }
}
```

4.2.3. 注意事项

在访问数组中的元素的时候， 注意下标的问题!

如果使用错误的下标访问数组中的元素， 将会出现 **ArrayIndexOutOfBoundsException** 异常!

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 数组的元素访问
 */
public class Test {
    public static void main(String[] args) {
        // 实例化一个数组
        int[] array = { 1, 2, 3, 4, 5 };
        // 访问数组中的元素
        array[10] = 300;  // 使用下标10访问数组中的元素， 此时数组的最大下标为4， 就会出现 ArrayIndexOutOfBoundsException 异常
    }
}
```

4.4. 数组的遍历(会)

数组遍历： 其实就是按照数组中元素存储的顺序， 依次获取到数组中的每一个元素。

4.4.1. 下标遍历

思路： 循环依次获取数组中的每一个下标， 再使用下标访问数组中的元素

```java
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 下标遍历
 */
public class Test {
    public static void main(String[] args) {
        // 实例化一个数组
        int[] array = { 1, 2, 3, 4, 5 };
        // 使用下标遍历数组
        for (int i = 0; i < array.length; i++) {
            System.out.println(array[i]);
        }
    }
}
```

4.4.2. 增强for循环

思路： 依次使用数组中的每一个元素， 给迭代变量进行赋值。

```java
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 下标遍历
 */
public class Test {
    public static void main(String[] args) {
        // 实例化一个数组
```

```
9           int[] array = { 1, 2, 3, 4, 5 };
10          // 依次使用数组中的每一个元素， 给迭代变量进行赋值。
11          // 此时， 数组中的每一个元素依次给 element 进行赋值。
12          for (int element : array) {
13              System.out.println(element);
14          }
15      }
16  }
```

4.4.3. 两种方式的对比

- 如果需要在遍历的同时， 获取到数组中的元素下标， 需要使用下标遍历法。
- 如果需要在遍历的同时， 修改数组中的元素， 需要使用下标遍历法。
- 如果仅仅是想要获取数组中的每一个元素， 不需要下标， 也不需要修改数组中的元素， 使用增强for循环。 因为这种方式， 遍历的效率比下标遍历法高。

4.5. 函数和数组的联合应用(会)

4.5.1. 函数传参分类

值传递:将保存简单数据的变量作为参数传递

址传递:将保存地址的变量作为参数传递

址传递优点:让我们可以实现使用一个变量一次传递多个值

4.5.2. 示例代码

```java
public class Demo3 {
  public static void main(String[] args) {
    //求三个数的和
    //直接用数值作为参数传递-值传递
    int tmp1 = getMax(3, 4, 6);
    System.out.println(tmp1);
    //用数组实现求三个数的和-址传递
    int[] arr1 = new int[] {3,5,8};
    int tmp2 = getMax(arr1);
    System.out.println(tmp2);

  }

  public static int getMax(int a,int b,int c) {//值传递
    int tmp = a>b?a:b;
    return  c>tmp?c:tmp;
  }

  public static int getMax(int[] arr) {//地址传递   arr = arr1
    int max = arr[0];
    for (int i=0;i<arr.length-1;i++) {
      if (max < arr[i+1]) {
        max = arr[i+1];
      }
    }

    return max;
  }
}
```

4.5.3. 址传递的深入理解(扩展)

4.4.5.1 值传递和址传递比较

通过值作为参数传递,函数内部值的变量不会改变外部的值.

通过地址作为参数传递,函数内部值的变量可以直接改变外部值.

4.4.5.2. 示例代码

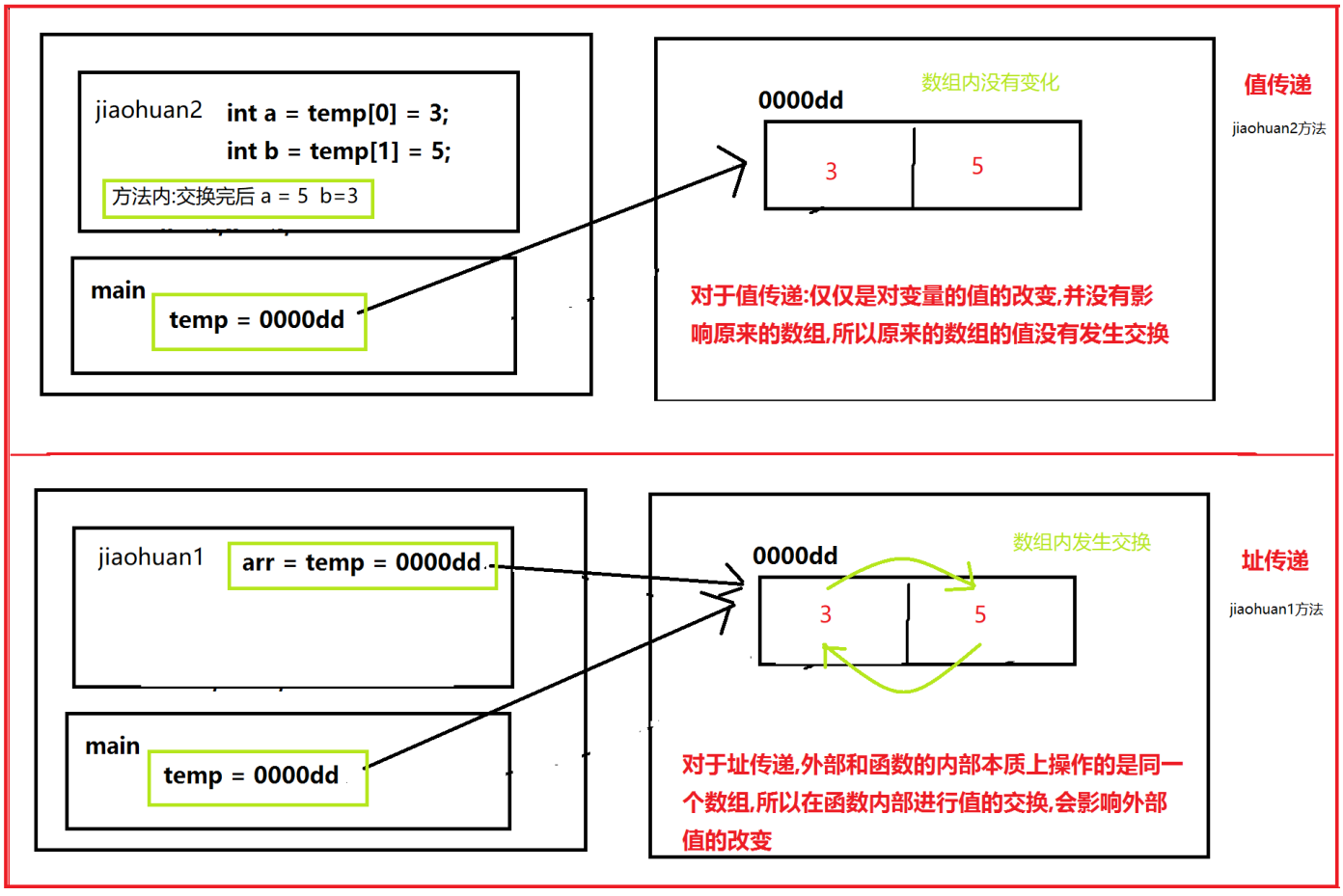
```
public class Demo4 {
  public static void main(String[] args) {
    //交换两个数的值
    int[] temp = {3,5};
    //地址传递
    jiaohuan1(temp);
    //我们发现通过址传递数组temp内的两个值发生了交换
    System.out.println("temp[0]:"+temp[0]+"
temp[1]:"+temp[1]);//  5   3

    //值传递
    int[] temp1 = {3,5};
    jiaohuan2(temp1[0], temp1[1]);
    //通过值传递数组temp内的两个值没有发生交换
    System.out.println("temp1[0]:"+temp1[0]+"
temp1[1]:"+temp1[1]);//  3    5
  }

  //地址传递
  public static void jiaohuan1(int[] arr) {
    arr[0] = arr[0] ^ arr[1];
    arr[1] = arr[0] ^ arr[1];
    arr[0] = arr[0] ^ arr[1];
  }
```

```
23    //值传递
24    public static void jiaohuan2(int a,int b) {
25        a = a ^ b;
26        b = a ^ b;
27        a = a ^ b;
28    }
29  }
30
```

4.4.5.3 内存分析




```
1  内存说明:
2  1.我们发现在值传递发生过程中,a和b的值在方法jiaohuan2中确实发生了交换,但是并没有影响到数组的值.
3  2.在址传递过程中,方法jiaohuan1中的变量arr和main方法中的变量temp保存的是同一个数组的地址,所以此时不管我们通过那个变量进行操作,都会对数组的值进行改变.
```

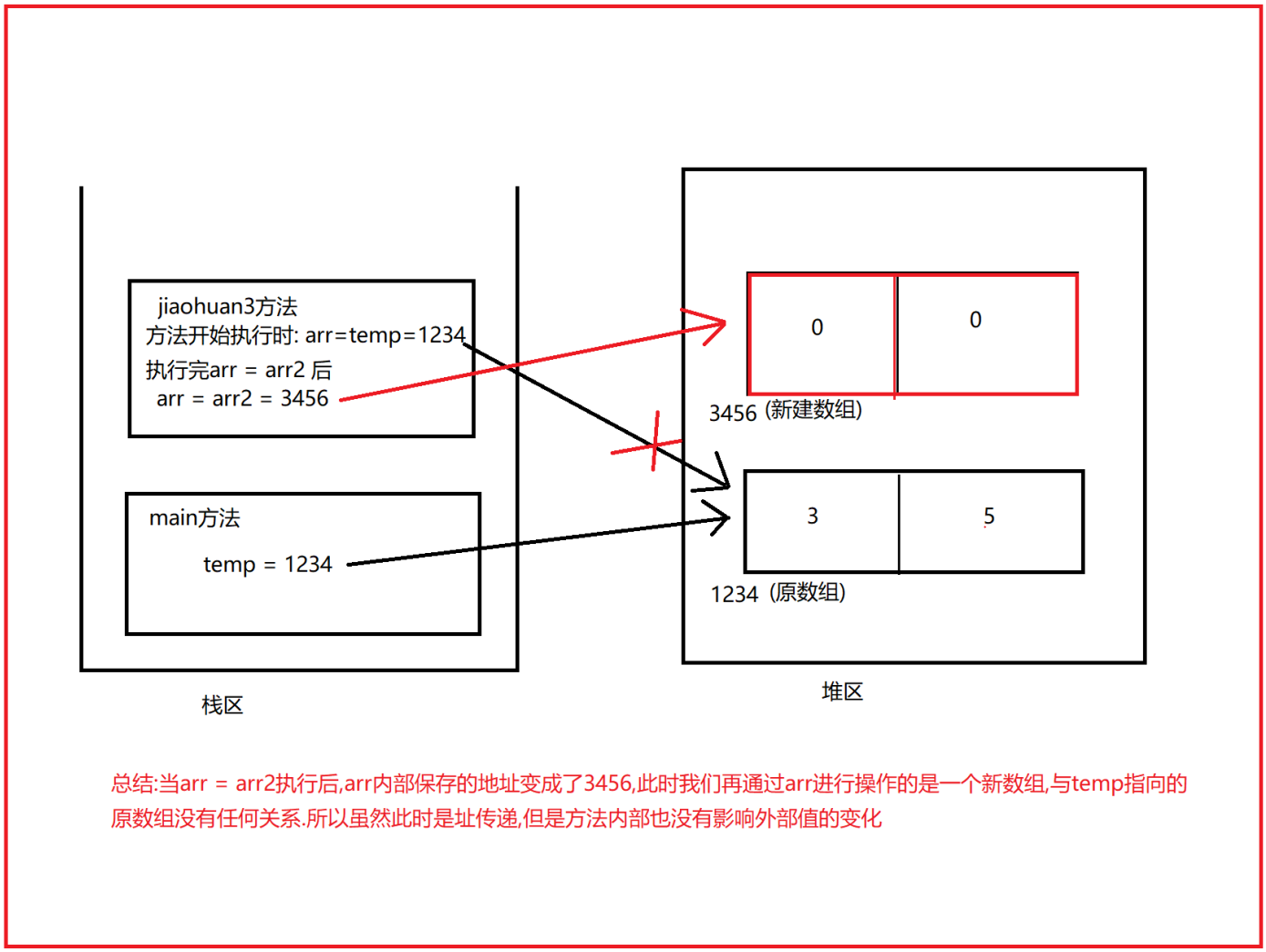
总结:址传递的最终原因是两个变量保存了同一个数组的地址,操作的是同一个数组.

4.4.5.4 案例分析

我们将上面例子中的址传递方法jiaohuan1替换成下面的方法jiaohuan3,再来观察数组temp的值,发现两个值并没有发生交换

```
1   public static void jiaohuan3(int[] arr) {//arr = temp
2     int[] arr2 = new int[2];
3     arr = arr2;
4
5     arr[0] = arr[0] ^ arr[1];
6     arr[1] = arr[0] ^ arr[1];
7     arr[0] = arr[0] ^ arr[1];
8
9   }
```

4.4.5.5 内存分析



原因总结:当arr = arr2执行后,arr内部保存的地址变成了3456,此时我们再通过arr进行操作的是一个新数组,与temp指向的原数组没有任何关系.所以虽然此时是址传递,但是方法内部也没有影响外部值的变化.

4.6. 数组的排序

4.6.1 时间复杂度和空间复杂度(了解)

讲解详情见文档---时间复杂度和空间复杂度

```
排序，即排列顺序，将数组中的元素按照一定的大小关系进行重新排列。
根据时间复杂度和空间复杂度选择排序方法

各算法的时间复杂度
平均时间复杂度
```

```
插入排序 O(n2)
冒泡排序 O(n2)
选择排序 O(n2)
快速排序 O(n log n)
堆排序 O(n log n)
归并排序 O(n log n)
基数排序 O(n)
希尔排序 O(n1.25)

复杂度作用:了解了时间复杂度和空间复杂度,可以更好的选择算法,提高排序查找的效率.
```

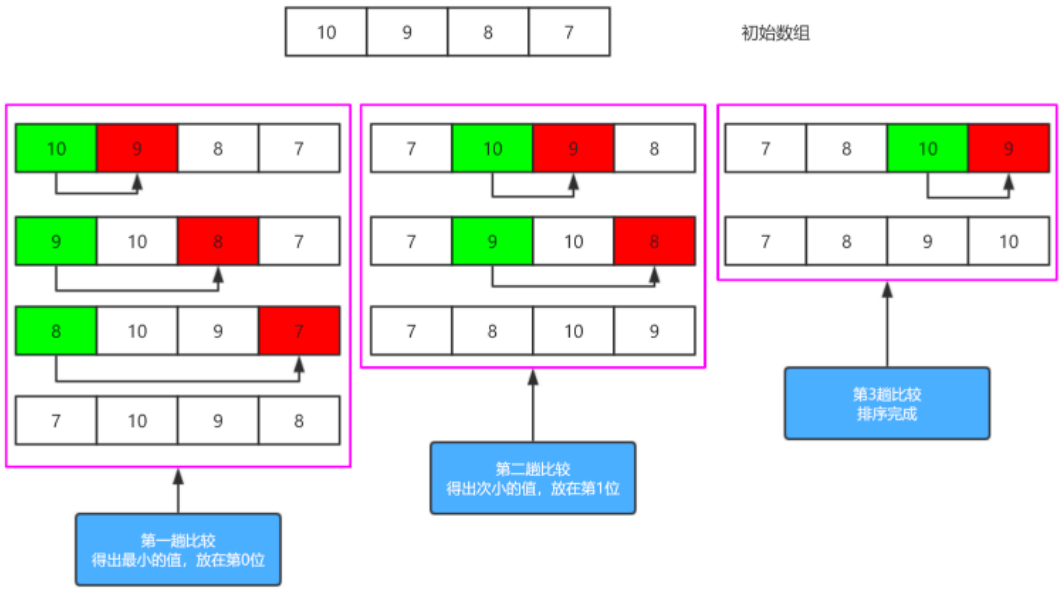
在Java中， 我们最常用的排序有:

- 选择排序：固定值与其他值依次比较大小，互换位置。
- 冒泡排序：相邻的两个数值比较大小，互换位置。

JDK提供默认的升序排序

- JDK排序：java.util.Arrays.sort(数组);

4.6.1. 选择排序(会)

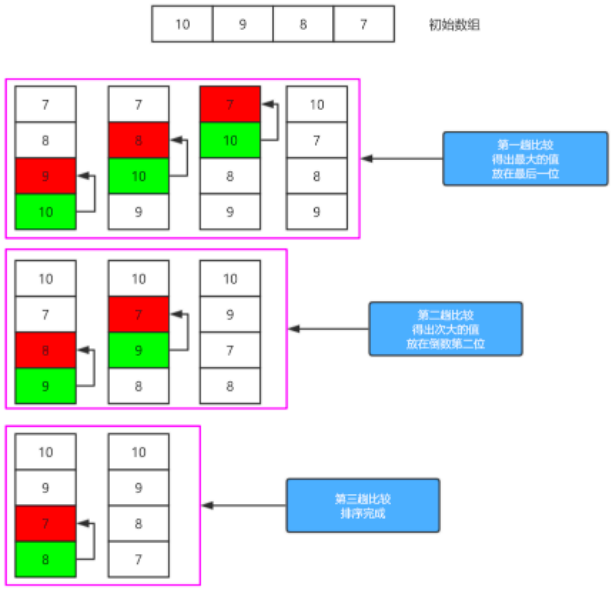



```
/**
```

```java
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 选择排序
 */
public class Test {
    public static void main(String[] args) {
        // 实例化一个数组
        int[] array = { 1, 2, 3, 4, 5 };
        // 选择排序
        sort(array);
    }
    /**
     * 使用选择排序，对数组进行排列
     * @param array 需要排序的数组
     */
    public static void sort(int[] array) {
        int times = 0;
        // 1. 固定下标，和后面的元素进行比较
        for (int i = 0; i < array.length - 1; i++) {
            // 2. 定义一个变量，用来记录最小值的下标
            int minIndex = i;
            // 3. 找出剩余元素中的最小值
            for (int j = i + 1; j < array.length; j++)
{
                if (array[j] < array[minIndex]) {
                    minIndex = j;
                }
            }
            // 4. 交换第i位和最小值位的元素即可
            if (minIndex != i) {
                int temp = array[i];
                array[i] = array[minIndex];
                array[minIndex] = temp;
```

```
                    times++;
                }
            }
            System.out.println(times);
        }
}
```

4.6.2. 冒泡排序(会)




```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 冒泡排序
 */
public class Test {
    public static void main(String[] args) {
        // 实例化一个数组
        int[] array = { 1, 2, 3, 4, 5 };
        // 冒泡排序
        sort(array);
    }
```

```
13      /**
14       * 使用冒泡排序进行升序排序
15       * @param array 需要排序的数组
16       */
17      public static void sort(int[] array) {
18          // 1. 确定要进行多少趟的比较
19          for (int i = 0; i < array.length - 1; i++) {
20              // 2. 每趟比较，从第0位开始，依次比较两个相邻的元
    素
21              for (int j = 0; j < array.length - 1 - i;
    j++) {
22                  // 3. 比较两个相邻的元素
23                  if (array[j] > array[j + 1]) {
24                      int temp = array[j];
25                      array[j] = array[j + 1];
26                      array[j + 1] = temp;
27                  }
28              }
29          }
30      }
31  }
```

4.6.3 快速排序(了解)

详情看文档---快速排序

4.6.4 归并排序(了解)

详情看文档---归并排序

4.7. 数组的查询(会)

数组查询， 即查询数组中的元素出现的下标。

4.7.1. 顺序查询

顺序查询， 即遍历数组中的每一个元素， 和要查询的元素进行对比。 如果是要查询的元素， 这个下标就是要查询的下标。

查询三要素:

1.我们只找查到的第一个与key相同的元素,查询结束.

2.当查询到与key相同的元素时,返回元素的下标

3.如果没有查询到与key相同的元素,返回-1

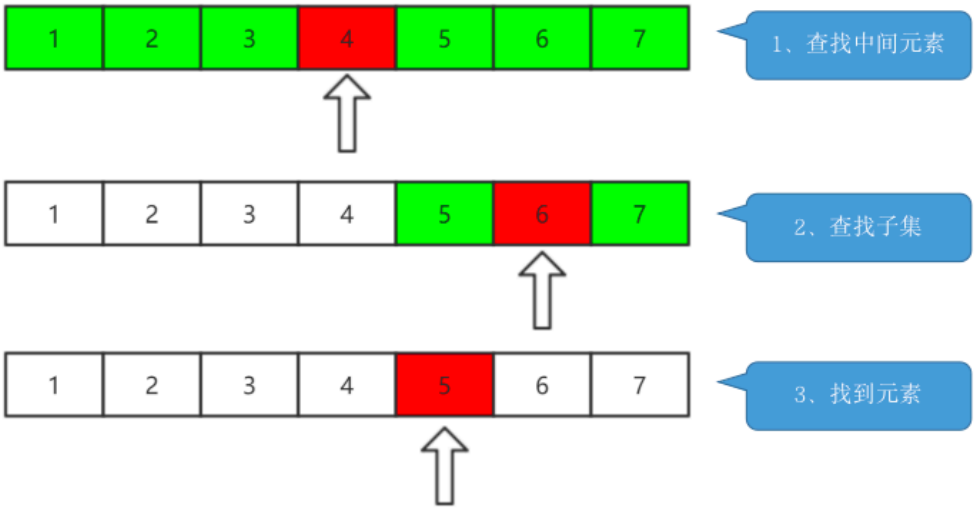
```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 顺序查询
 */
public class Test {
    public static void main(String[] args) {
        // 1. 实例化一个数组
        int[] array = { 1, 3, 5, 7, 9, 0, 8, 8, 8, 6, 4, 8,  2 };
        // 2. 从这个数组中查询元素8的下标
        System.out.println(indexOf(array, 80));
    }
    /**
     * 使用顺序查询法，从数组array中查询指定的元素
     * @param array 需要查询的数组
     * @param element 需要查询的元素
```

```
17          * @return 下标
18          */
19         public static int indexOf(int[] array, int element)
{
20             // 1. 使用下标遍历法，依次获取数组中的每一个元素
21             for (int i = 0; i < array.length; i++) {
22                 // 2. 依次用每一个元素和element进行比较
23                 if (array[i] == element) {
24                     // 说明找到了这个元素，这个下标就是想要的下标
25                     return i;
26                 }
27             }
28             // 约定俗成：如果要查询的元素，在数组中不存在，一般情况
下，得到的结果都是-1
29             return -1;
30         }
31     }
```

4.7.2. 二分查询

二分查询， 即利用数组中间的位置， 将数组分为前后两个字表。 如果中间位置记录的关键字大于查找关键字，则进一步查找前一子表，否则进一步查找后一子表。重复以上过程，直到找到满足条件的记录，使查找成功，或直到子表不存在为止，此时查找不成功。

二分查询， 要求数组必须是排序的， 否则无法使用二分查询。




```java
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 二分查询
 */
public class Test {
    public static void main(String[] args) {
        int[] array = { 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 };
        System.out.println(binarySearch(array, 14));
    }

    /**
     * 二分查询，查询数组中element出现的下标
     * @param array 需要查询的数组
     * @param element 需要查询的元素
     * @return 元素出现的下标，如果数组中不包含这个元素，返回-1
     */
    public static int binarySearch(int[] array, int element) {
        // 1. 定义两个变量，用来标记需要查询范围的上限和下限
        int max = array.length - 1, min = 0;

```

```
22          while (max >= min) {
23              // 2. 计算中间下标
24              int mid = (max + min) / 2;
25
26              // 3. 使用中间下标的元素和要查询的元素进行比较
27              if (array[mid] == element) {
28                  // 说明找到了
29                  return mid;
30              }
31              else if (array[mid] > element) {
32                  // 中间位比要查询的元素大
33                  max = mid - 1;
34              }
35              else {
36                  // 中间位比要查询的元素小
37                  min = mid + 1;
38              }
39          }
40
41          // 如果循环走完了，依然没有结果返回，说明要查询的元素在数组中不存在
42          return -1;
43      }
44  }
45
```

4.8. 可变长参数(了解)

4.8.1. 概念

可以接收多个类型相同的实参，个数不限，使用方式与数组相同。

在调用方法的时候， 实参的数量可以写任意多个。

作用:简化代码,简化操作等

4.8.2. 语法

`数据类型... 形参名` （必须放到形参列表的最后位，且只能有一个）

```
// 这里的参数paramters其实就是一个数组
static void show(int... parameters) {
  //
}
```

4.8.3. 使用

```
public class Demo12 {
    public static void main(String[] args) {
        //求两个数的和
        sum(3,5);//值传递
        int [] arr1 = {3,5};
        sum(arr1);//地址传递
        sum1(arr1);
        //可变参数的特点
        //1.给可变参数传值的实参可以直接写,个数不限制,内部会自动的将他们放入可变数组中.
//          sum({3,4,5};
        sum1(3,4,5,6);
        //2.当包括可变参数在内有多个参数时,可变参数必须放在最后面,并且一个方法中最多只能有一个可变参数
        sum2(4,5,6,7);
        //3.当可变参数的方法与固定参数的方法是重载关系时,调用的顺序,固定参数的优先于可变参数的.
        sum3(3,5);
```

```
16          //4.如果同时出现两个重载方法,一个参数是可变参数,一个是第一个是固定参数,后面是可变参数.
17          //我们就不能使用下面的方法调用.两个方法都不能使用
18  //          sum4(4,5,6,7,8);
19      }
20
21      public static  int sum(int a,int b){
22          return  a+b;
23      }
24
25      public static  int sum(int[] a){
26          int sum=0;
27          for (int i = 0; i < a.length; i++) {
28              sum+=a[i];
29          }
30          return  sum;
31      }
32
33      //用可变参数实现求和
34      //构成:数据类型+...    实际上就是数据类型[]  即:int[]
35      public static  int sum1(int... a){
36          int sum=0;
37          for (int i = 0; i < a.length; i++) {
38              sum+=a[i];
39          }
40          return  sum;
41      }
42      //2.当包括可变参数在内有多个参数时,可变参数必须放在最后面,并且一个方法中最多只能有一个可变参数
43      public static  int sum2(float b,int... a){
44          int sum=0;
45          for (int i = 0; i < a.length; i++) {
46              sum+=a[i];
```

```
47          }
48          return  sum;
49      }
50
51      //3.当可变参数的方法与固定参数的方法是重载关系时,调用的顺序,
    固定参数的优先于可变参数的.
52      public static  int sum3(int a,int b){
53          System.out.println("haha");
54          return  a+b;
55      }
56      public static  int sum3(int... b){
57          System.out.println("hehe");
58          int sum=0;
59          for (int i = 0; i < b.length; i++) {
60              sum+=b[i];
61          }
62          return  sum;
63      }
64
65      //4.如果同时出现两个重载方法,一个参数是可变参数,一个是第一个
    是固定参数,后面是可变参数.
66      //我们就不能使用下面的方法调用.两个方法都不能使用
67      public static int sum4(int... a){
68          int mysum = 0 ;
69          for (int i:a) {
70              mysum+=i;
71          }
72          return  mysum;
73      }
74      public static int sum4(int b,int... a){
75          int mysum = 0 ;
76          for (int i:a) {
77              mysum+=i;
```

```
78          }
79          return  mysum;
80      }
81  }
```

4.9. 二维数组(了解)

4.9.1. 概念

二维数组， 其实就是数组中嵌套数组。

二维数组中的每一个元素都是一个小的数组。

> 理论上来讲， 还可以有三维数组、四维数组， 但是常用的其实就是二维数组。

4.9.2. 定义与使用

```
1  /**
2   * @Author 千锋大数据教学团队
3   * @Company 千锋好程序员大数据
4   * @Description 二维数组的定义与使用
5   */
6  public class Array3 {
7      public static void main(String[] args) {
8          // 1. 实例化一个二维数组
9          //    第一个中括号中的3：二维数组中包含了三个一维数组
10         //    第二个中括号中的5：二维数组中的每一个一维数组长度为5
11         int[][] array = new int[3][5];
```

```
12          // 使用双下标访问数组中的元素
13          array[0][3] = 10;
14
15          // 这里得到的，是二维数组的长度，3
16          System.out.println(array.length);
17
18
19          // 2. 实例化一个二维数组
20          //    第一个中括号中的3：二维数组中包含了三个一维数组
21          //    第二个中括号中什么都没有，代表现在二维数组中的三个元素是 null
22          int[][] array2 = new int[3][];
23          array2[0] = new int[] { 1, 2, 3 };
24
25          // 3. 通过初始值实例化一个二维数组
26          int[][] array3 = { {1, 2, 3}, {1, 2, 3, 4, 5}, {2, 3, 4} };
27      }
28  }
```

4.10. Arrays工具类(会)

4.10.1. 常用方法

方法	描述
copyOf(int[] array, int newLength)	从原数组中拷贝指定数量的元素，到一个新的数组中，并返回这个新的数组
copyOfRange(int[] array, int from, int to)	从原数组中拷贝指定范围 [from, to) 的元素，到一个新的数组中，并返回这个新的数组
equals(int[] array1, int[] array2)	判断两个数组是否相同
fill(int[] array, int element)	使用指定的数据，填充数组
sort(int[] array)	对数组进行排序（升序）
binarySearch(int[] array, int element)	使用二分查找法，到数组中查询指定的元素出现的下标
toString(int[] array)	将数组中的元素拼接成字符串返回

4.10.2. 示例代码

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 二维数组的定义与使用
 */
public class ArraysUsage {
    // Arrays 工具方法：可以便捷的实现指定操作的方法
    // Arrays 工具类：若干个工具方法的集合
    public static void main(String[] args) {
```

```java
        // 1. 实例化一个数组
        int[] array = { 1, 3, 5, 7, 9, 0, 8, 6, 4, 2 };

        // 从原数组中拷贝指定数量的元素，到一个新的数组中，并返回这个新的数组
        // 第一个参数：源数组
        // 第二个参数：需要拷贝的元素数量，如果这个数量比源数组长，目标数组剩余的部分补默认值
        int[] ret1 = Arrays.copyOf(array, 13);

        // 从原数组中拷贝指定范围 [from, to) 的元素，到一个新的数组中，并返回这个新的数组
        // 第一个参数：源数组
        // 第二个参数：起始下标，从这个下标位的元素开始拷贝
        // 第三个参数：目标下标，拷贝到这个下标位截止
        int[] ret2 = Arrays.copyOfRange(array, array.length, array.length + 10);

        // 判断两个数组是否相同
        // 判断的逻辑：长度、每一个元素依次都相等
        boolean ret3 = Arrays.equals(ret1, ret2);

        // 使用指定的数据，填充数组
        // 第一个参数：需要填充的数组
        // 第二个参数：填充的数据
        Arrays.fill(ret2, 100);

        // 对数组进行排序（升序）
        Arrays.sort(array);

        // 使用二分查找法，到数组中查询指定的元素出现的下标
        // 第一个参数：需要查询的数组
        // 第二个参数：需要查询的数据
```

```
39          // 返回：这个元素出现的下标，如果不存在，返回-1
40          int index = Arrays.binarySearch(array, 4);
41
42          // 将数组中的元素拼接成字符串返回
43          String str = Arrays.toString(array);
44          System.out.println(str);
45      }
46  }
```