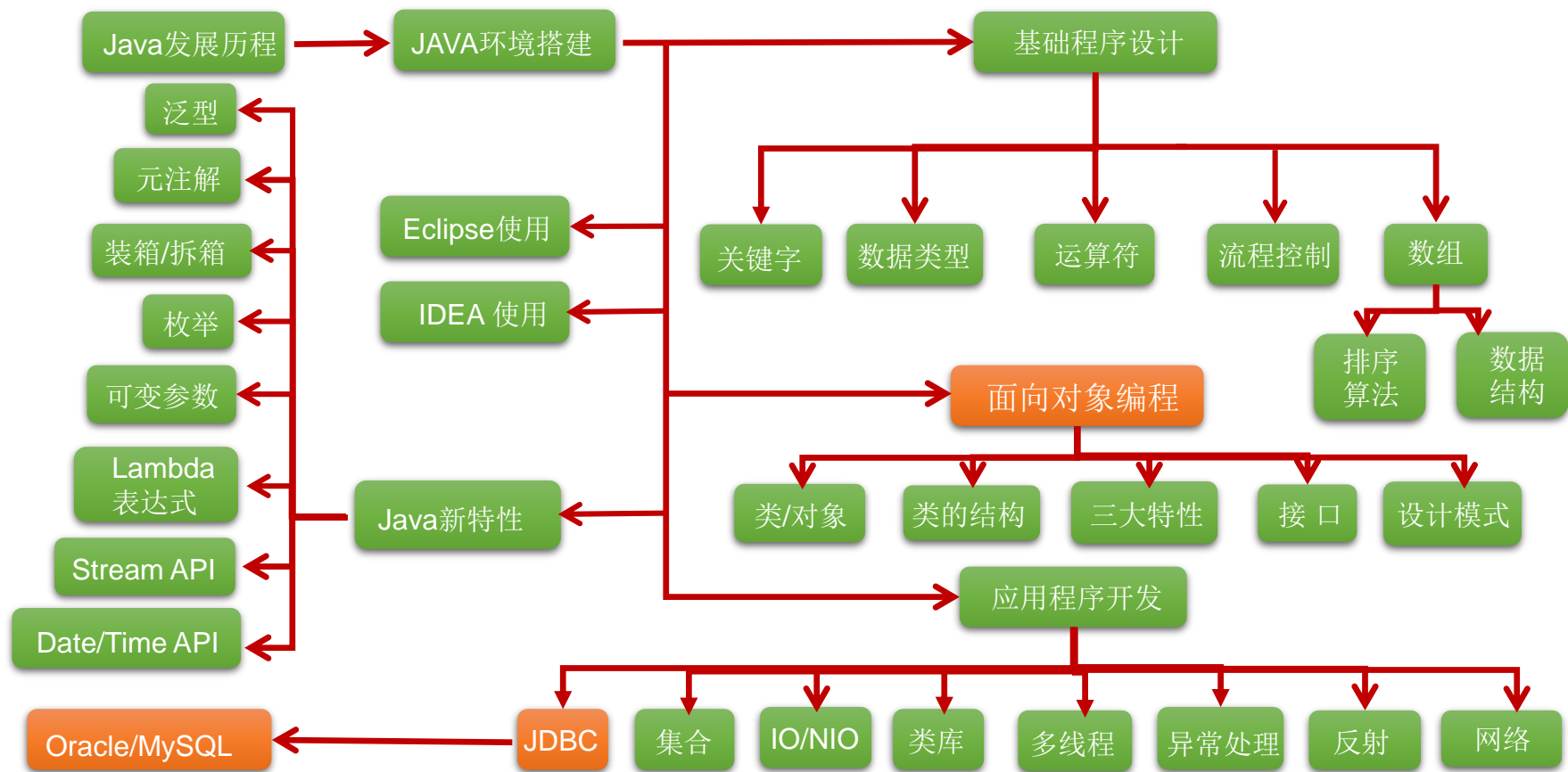


# 第3章 数 组



讲师：宋红康  
新浪微博：尚硅谷-宋红康



# 目录



1

数组的概述

2

一维数组的使用

3

多维数组的使用

4

数组中涉及到的常见算法

5

Arrays工具类的使用

6

数组使用中的常见异常



## 3-1 数组的概述



## 3.1 数组的概述

- 数组(Array), 是多个相同类型数据按一定顺序排列的集合, 并使用一个名字命名, 并通过编号的方式对这些数据进行统一管理。
- 数组的常见概念
  - 数组名
  - 下标(或索引)
  - 元素
  - 数组的长度

定价
20.32
43.21
43.22
54.21
64.21
33.87
65.33
78.32
56.98
22.65
62.32
71.99
44.55
39.21
29.43
37.24
48.21
50.62
71.42
89.49
50.26

菜名
拌海蜇
龙须菜
炆冬笋
玉兰片
烧鱼头
炸豆腐
拔丝山药
杏仁酪
烧萝卜
麻婆豆腐
红扒鱼翅
鱼香肉丝
叫花鸡
酸菜鱼
茄汁鱼卷
粉蒸鸡
酱茄子
佛跳墙
西湖醋鱼
太极虾
烤方



- 数组本身是引用数据类型，而数组中的元素可以是任何数据类型，包括基本数据类型和引用数据类型。
- 创建数组对象会在内存中开辟一整块连续的空间，而数组名中引用的是这块连续空间的首地址。
- 数组的长度一旦确定，就不能修改。
- 我们可以直接通过下标(或索引)的方式调用指定位置的元素，速度很快。
- 数组的分类：
  - 按照维度：一维数组、二维数组、三维数组、...
  - 按照元素的数据类型分：基本数据类型元素的数组、引用数据类型元素的数组(即对象数组)



## 3-2 一维数组的使用



- 一维数组的声明方式：

`type var[] 或 type[] var;`

➤ 例如：

```
int a[];
```

```
int[] a1;
```

```
double b[];
```

```
String[] c; //引用类型变量数组
```

- Java语言中声明数组时不能指定其长度(数组中元素的数)， 例如： `int a[5];` **//非法**





## 3.2 一维数组的使用：初始化

●**动态初始化：** 数组声明且为数组元素分配空间与赋值的操作分开进行

```
int[] arr = new int[3];  
arr[0] = 3;  
arr[1] = 9;  
arr[2] = 8;
```

```
String names[];  
names = new String[3];  
names[0] = "钱学森";  
names[1] = "邓稼先";  
names[2] = "袁隆平";
```

●**静态初始化：** 在定义数组的同时就为数组元素分配空间并赋值。

```
int arr[] = new int[]{ 3, 9, 8};  
或  
int[] arr = {3,9,8};
```

```
String names[] = {  
    "李四光" , "茅以升" , "华罗庚"  
}
```



## 3.2 一维数组的使用：数组元素的引用

- 定义并用运算符**new**为之分配空间后，才可以引用数组中的每个元素；
- 数组元素的引用方式：数组名[数组元素下标]
  - 数组元素下标可以是整型常量或整型表达式。如a[3] , b[i] , c[6\*i];
  - 数组元素下标从0开始；长度为n的数组合法下标取值范围: 0 —>n-1；如int a[]=new int[3]; 可引用的数组元素为a[0]、a[1]、a[2]
- 每个数组都有一个属性**length**指明它的长度，例如：**a.length** 指明数组**a**的长度(元素个数)
  - 数组一旦初始化，其长度是不可变的



## 3.2 一维数组的使用：数组元素的默认初始化值

- 数组是引用类型，它的元素相当于类的成员变量，因此数组一经分配空间，其中的每个元素也被按照成员变量同样的方式被隐式初始化。例如：

```
public class Test {  
    public static void main(String argv[]){  
        int a[]= new int[5];  
        System.out.println(a[3]);    //a[3]的默认值为0  
    }  
}
```

- 对于基本数据类型而言，默认初始化值各有不同
- 对于引用数据类型而言，默认初始化值为null(注意与0不同！)



## 3.2 一维数组的使用：数组元素的默认初始化值

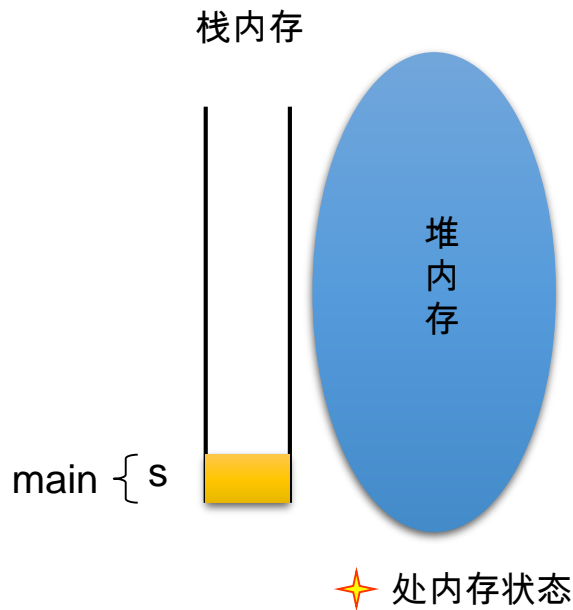
数组元素类型	元素默认初始化值
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0
char	0 或写为:'\u0000'(表现为空)
boolean	false
引用类型	null



### 创建基本数据类型数组 (1)

- Java中使用关键字`new`来创建数组
- 如下是创建基本数据类型元素的一维数组

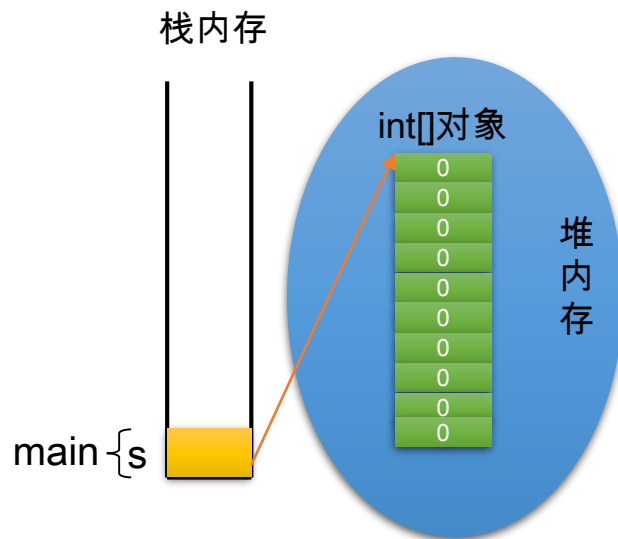
```
public class Test{  
    public static void main(String args[]){  
        int[] s; ✨  
        s = new int[10];  
        for ( int i=0; i<10; i++ ) {  
            s[i] =2*i+1;  
            System.out.println(s[i]);  
        }  
    }  
}
```





### 创建基本数据类型数组 (2)

```
public class Test{  
    public static void main(String args[]){  
        int[] s;  
        s = new int[10]; ✨  
        //int[] s=new int[10];  
        //基本数据类型数组在显式赋值之前，  
        //Java会自动给他们赋默认值。  
        for ( int i=0; i<10; i++ ) {  
            s[i] =2*i+1;  
            System.out.println(s[i]);  
        }  
    }  
}
```



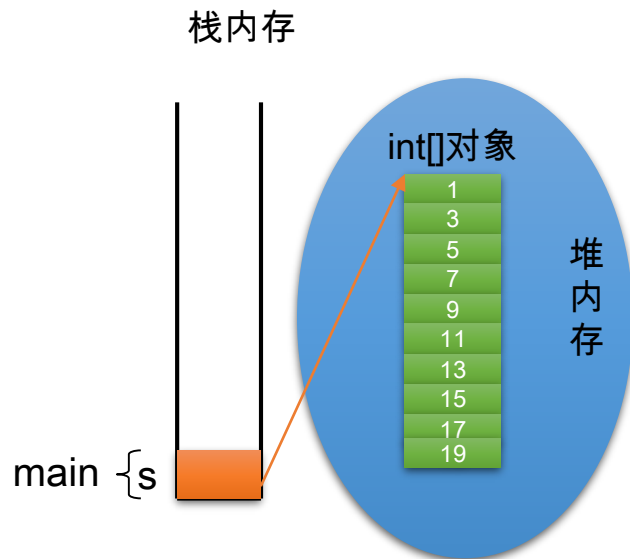
✨ 处内存状态

让天下没有难学的技术

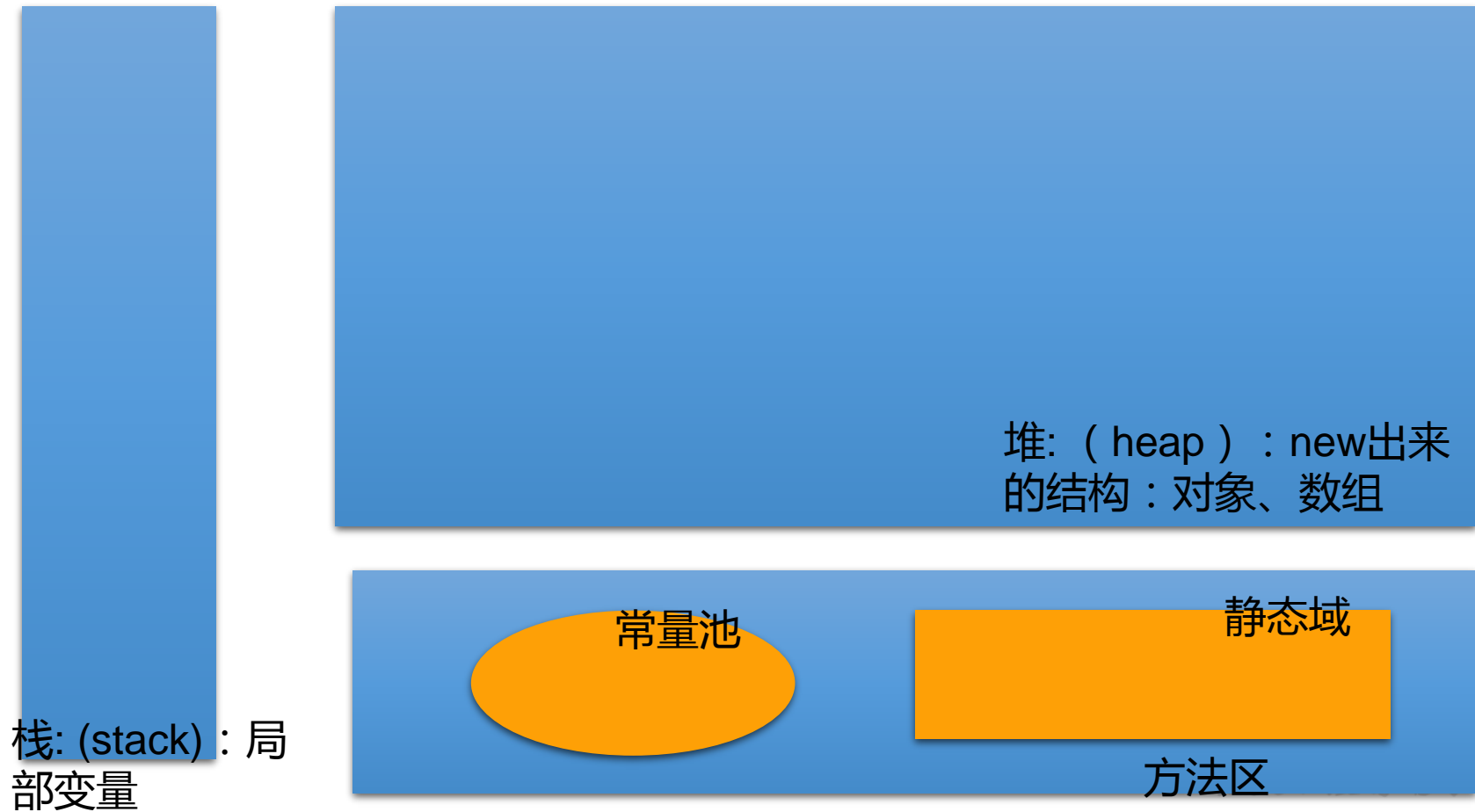


### 创建基本数据类型数组 (3)

```
public class Test{  
    public static void main(String args[]){  
        int[] s;  
        s = new int[10];  
        for ( int i=0; i<10; i++ ) {  
            s[i] = 2*i+1; ✨  
            System.out.println(s[i]);  
        }  
    }  
}
```



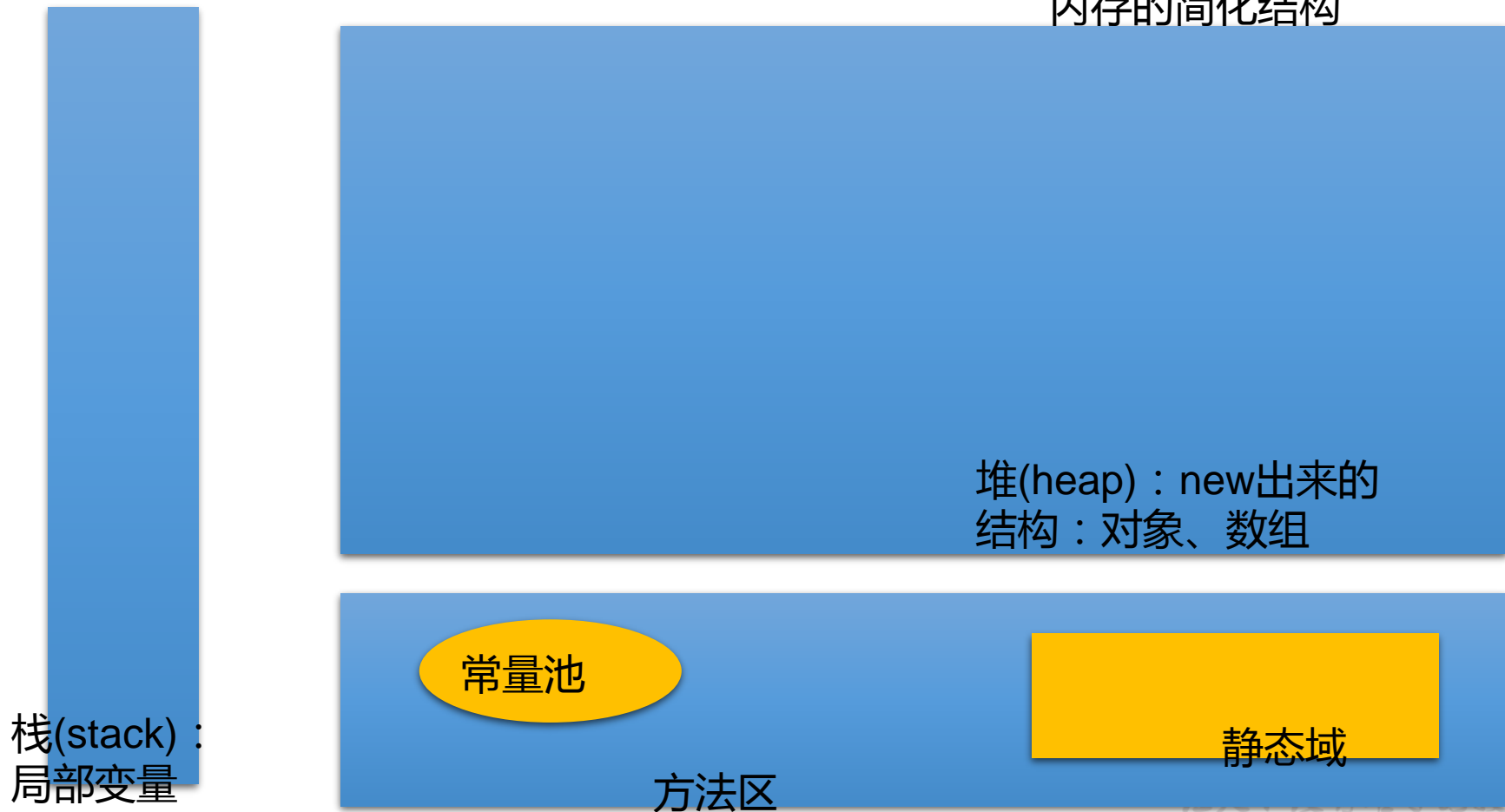
✨ 处内存状态







## 内存的简化结构





# 一维数组的内存解析

int[] arr1 = new int[4]; 硅谷

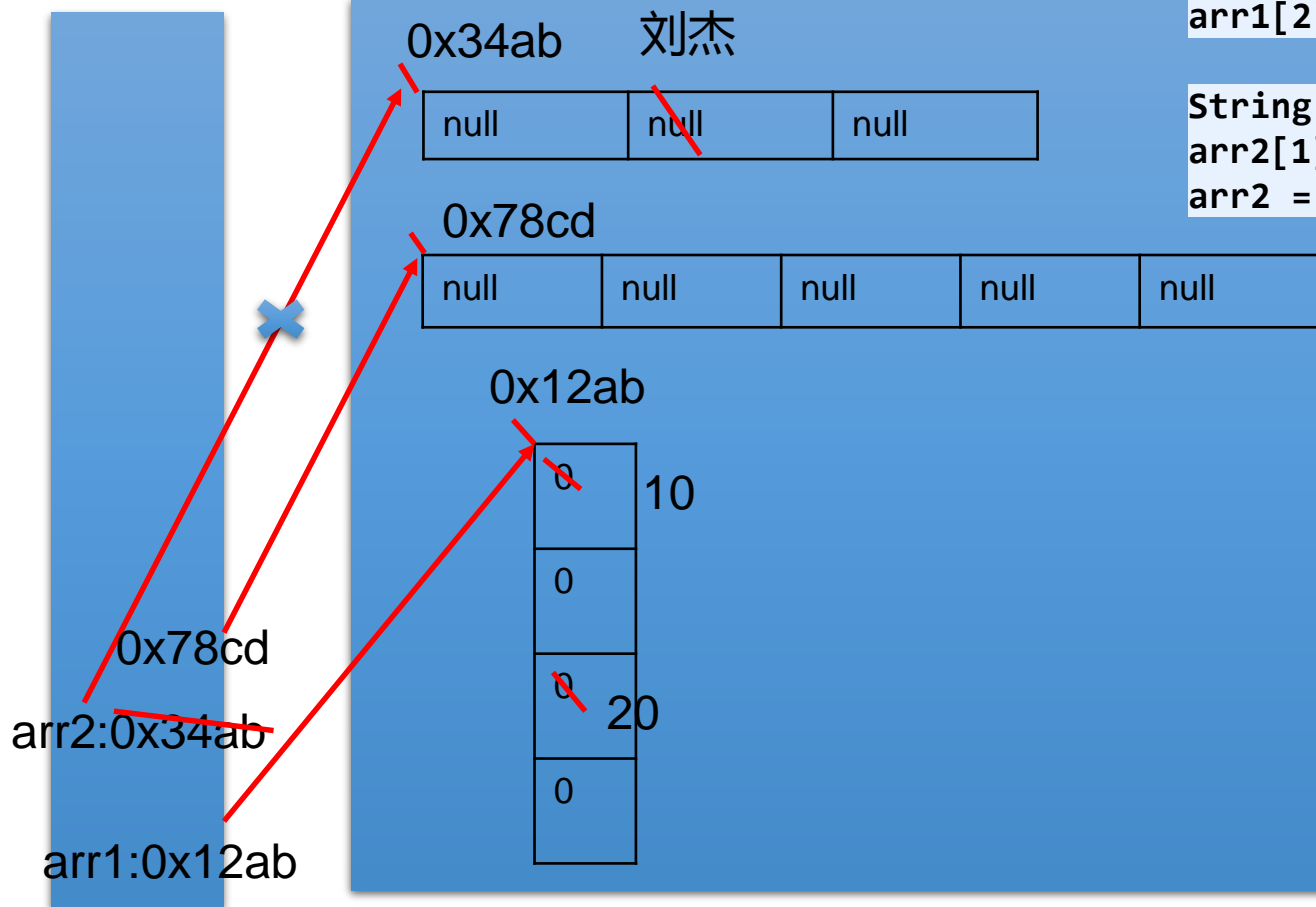
arr1[0] = 10;

arr1[2] = 20;

String[] arr2 = new String[3];

arr2[1] = "刘杰";

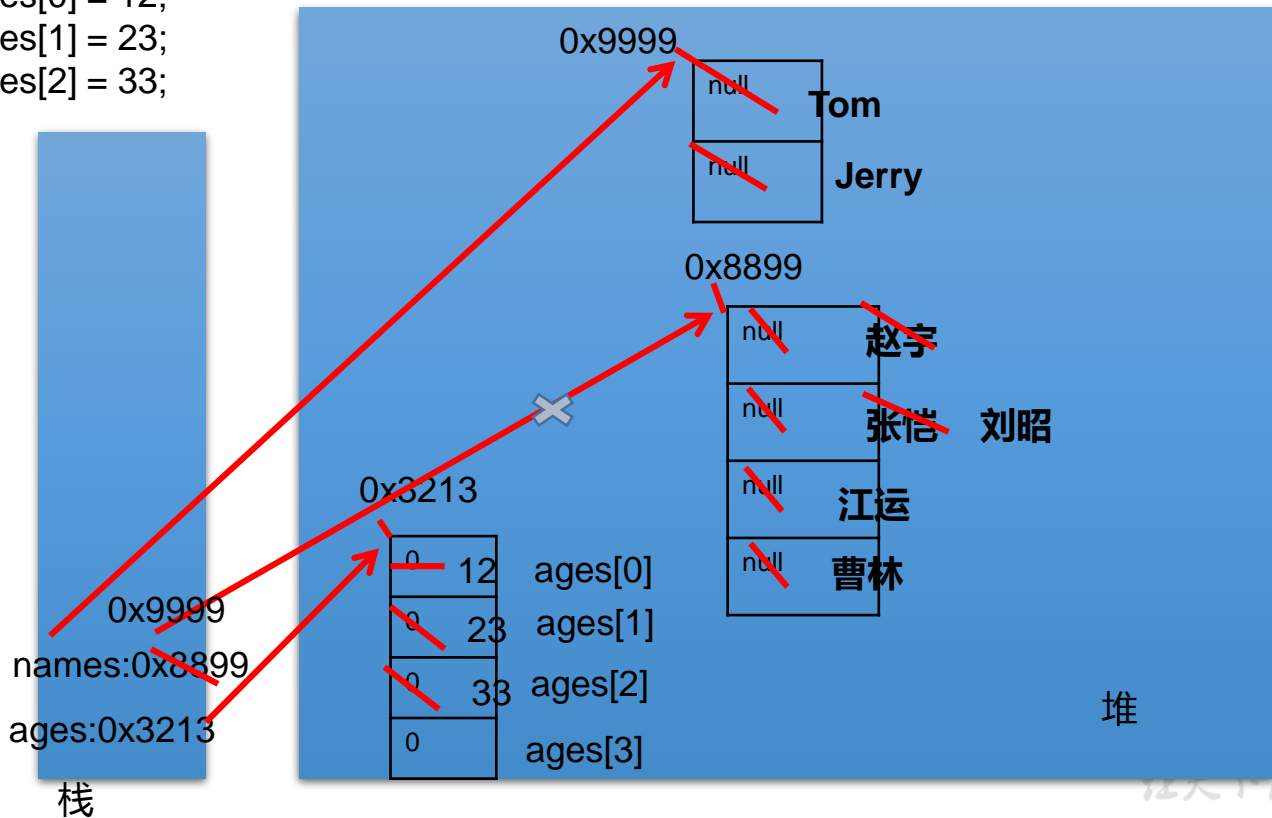
arr2 = new String[5];





```
int[] ages = new int[4];  
ages[0] = 12;  
ages[1] = 23;  
ages[2] = 33;
```

```
String[] names = new String[]{"赵宇", "张恺", "江运", "曹林"};  
names[1] = "刘昭";  
names = new String[]{"Tom", "Jerry"}; sysout(names[0]);
```





# 一维数组的内存解析

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

```
String[] arr1 = new String[4];
```

```
arr1[1] = "刘德华";
```

```
arr1[2] = "张学友";
```

```
arr1 = new String[3];
```

```
sysout(arr1[1]); // null
```

0x5566

null	null	null
------	------	------

0x12ab

刘德华

张学友

null	<del>null</del>	<del>null</del>	null
------	-----------------	-----------------	------

0x34ab

<del>0</del>	<del>0</del>	<del>0</del>
--------------	--------------	--------------

1

2

3

0x5566

~~arr1:0x12ab~~

arr:0x34ab





### 练习1

升景坊单间短期出租4个月，550元/月（水电煤公摊，网费35元/月），空调、卫生间、厨房齐全。屋内均是IT行业人士，喜欢安静。所以要求来租者最好是同行或者刚毕业的年轻人，爱干净、安静。

- **public class ArrayTest {**
- **public static void main(String[] args) {**
- **int[] arr = new int[]{8,2,1,0,3};**
- **int[] index = new int[]{2,0,3,2,4,0,1,3,2,3,3};**
- **String tel = "";**
- **for(int i = 0;i < index.length;i++){**
- **tel += arr[index[i]];**
- **}**
- **System.out.println("联系方式: " + tel);**
- **}**
- **}**



## 3.2 一维数组的使用

2. 从键盘读入学生成绩，找出最高分，并输出学生成绩等级。

- 成绩  $\geq$  最高分 - 10 等级为 'A'
- 成绩  $\geq$  最高分 - 20 等级为 'B'
- 成绩  $\geq$  最高分 - 30 等级为 'C'
- 其余 等级为 'D'

提示：先读入学生人数，根据人数创建int数组，存放学生成绩。

请输入学生人数:5

请输入5个成绩

56

74

89

41

89

最高分是：89

student 0 score is 56 grade is D

student 1 score is 74 grade is B

student 2 score is 89 grade is A

student 3 score is 41 grade is D

student 4 score is 89 grade is A



## 3-3 多维数组的使用



### 3.3 多维数组的使用

- Java 语言里提供了支持多维数组的语法。
- 如果说可以把一维数组当成几何中的线性图形，那么二维数组就相当于是一个表格，像右图Excel中的表格一样。
- 对于二维数组的理解，我们可以看成是一维数组 `array1` 又作为另一个一维数组 `array2` 的元素而存在。其实，从数组底层的运行机制来看，其实没有多维数组。

姓名	联系电话
陈伟霆	13387654384
刘诗诗	13845678765
周笔畅	13012393458
鹿晗	13623490545
张艺兴	13334505689
杨幂	13623495439
宋茜	13723490897
李敏镐	13789459034
苍井空	13323409435
赵丽颖	13945893324
迪丽热巴	18623495489
周杰伦	18523489094
胡歌	13723409895
郑爽	13423409548
唐嫣	13721348984
戚薇	13909234895
王俊凯	13012390435
华晨宇	18723490438
佟丽娅	18823467128
刘亦菲	13834589533
邓紫棋	17623485739





## 3.3 多维数组的使用

### 二维数组`[][]`：数组中的数组

**格式1（动态初始化）：** `int[][] arr = new int[3][2];`

定义了名称为arr的二维数组

二维数组中有3个一维数组

每一个一维数组中有2个元素

一维数组的名称分别为arr[0], arr[1], arr[2]

给第一个一维数组1脚标位赋值为78写法是：arr[0][1] = 78;

**格式2（动态初始化）：** `int[][] arr = new int[3][];`

二维数组中有3个一维数组。

每个一维数组都是默认初始化值null (注意：区别于格式1)

可以对这个三个一维数组分别进行初始化

arr[0] = new int[3]; arr[1] = new int[1]; arr[2] = new int[2];

注：

`int[][] arr = new int[][3];` //非法



**格式3（静态初始化）：** `int[][] arr = new int[][]{{3,8,2},{2,7},{9,0,1,6}};`

定义一个名称为arr的二维数组，二维数组中有三个一维数组

每一个一维数组中具体元素也都已初始化

第一个一维数组 `arr[0] = {3,8,2};`

第二个一维数组 `arr[1] = {2,7};`

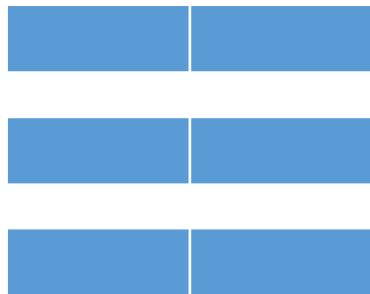
第三个一维数组 `arr[2] = {9,0,1,6};`

第三个一维数组的长度表示方式：`arr[2].length;`

- 注意特殊写法情况：`int[] x,y[];` x是一维数组，y是二维数组。
- Java中多维数组不必都是规则矩阵形式



### 3.3 多维数组的使用

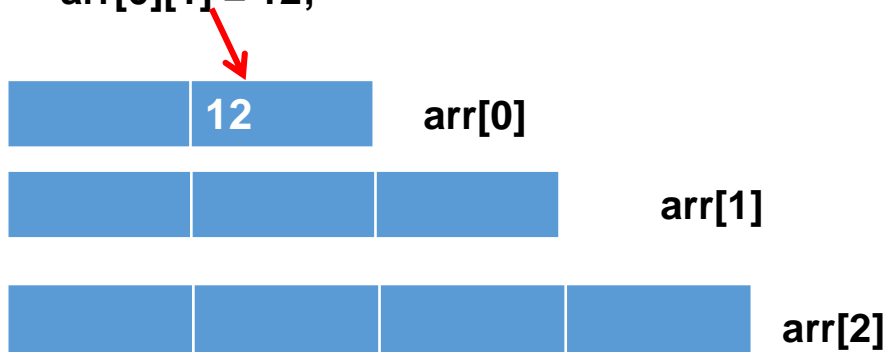


```
int[][] arr = new int[3][2];
```

或者

```
int[][] arr = new int[3][];  
arr[0] = new int[2];  
arr[1] = new int[2];  
arr[2] = new int[2];
```

`arr[0][1] = 12;`

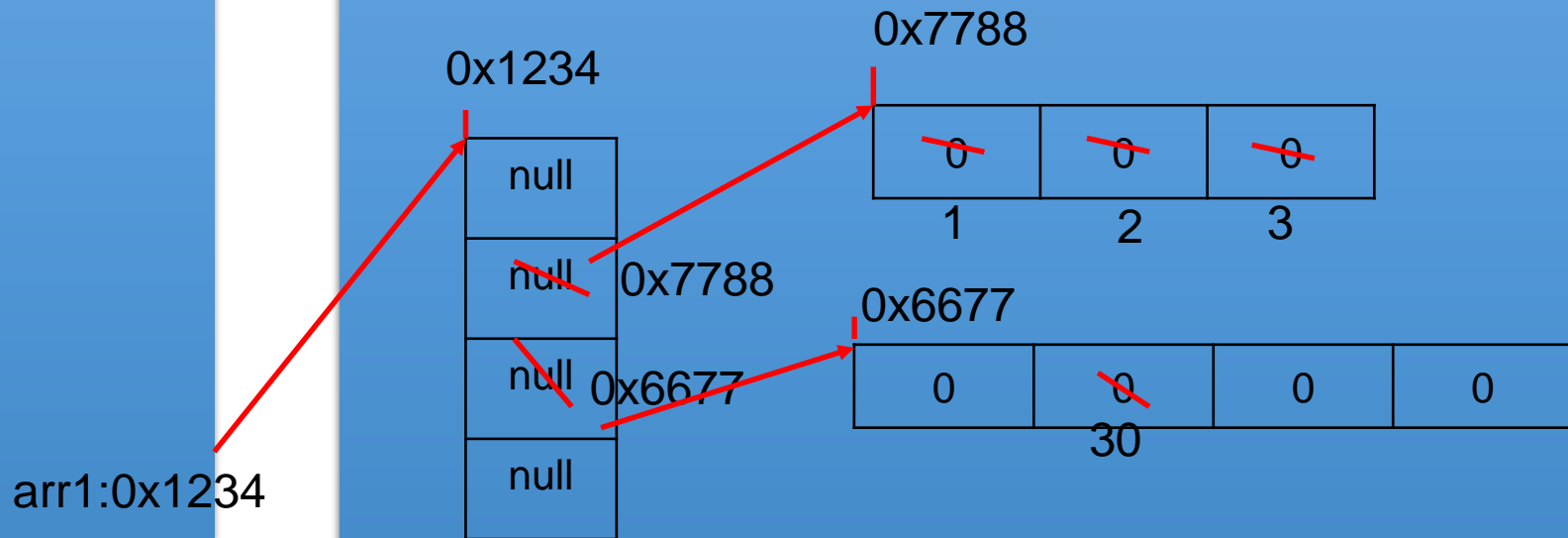


```
int[][] arr = new int[3][];  
arr[0] = new int[2];  
arr[1] = new int[3];  
arr[2] = new int[4];
```



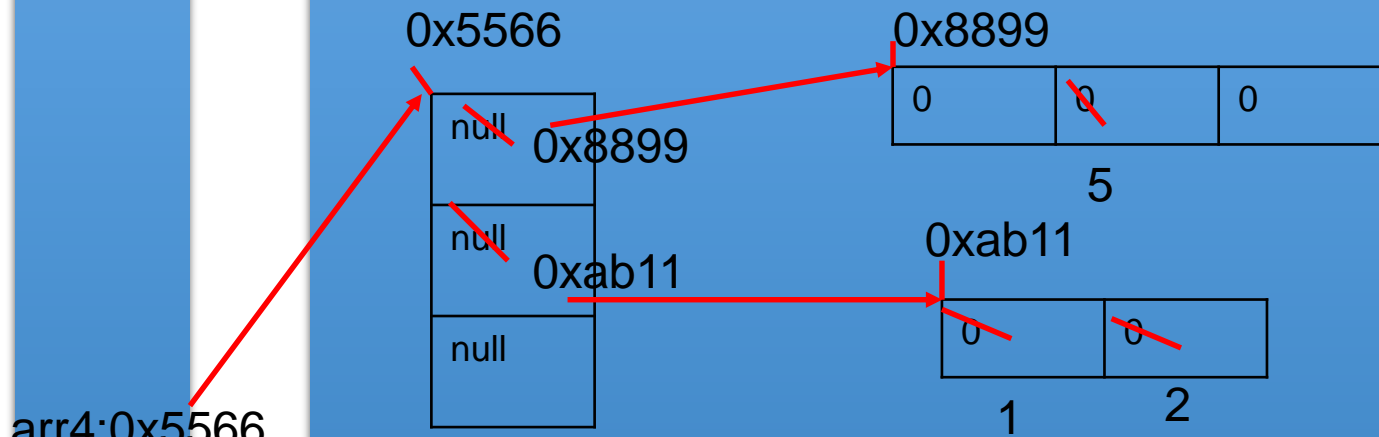
# 二维数组的内存解析

```
int[][] arr1 = new int[4][];  
arr1[1] = new int[]{1,2,3};  
arr1[2] = new int[4];  
arr1[2][1] = 30;
```





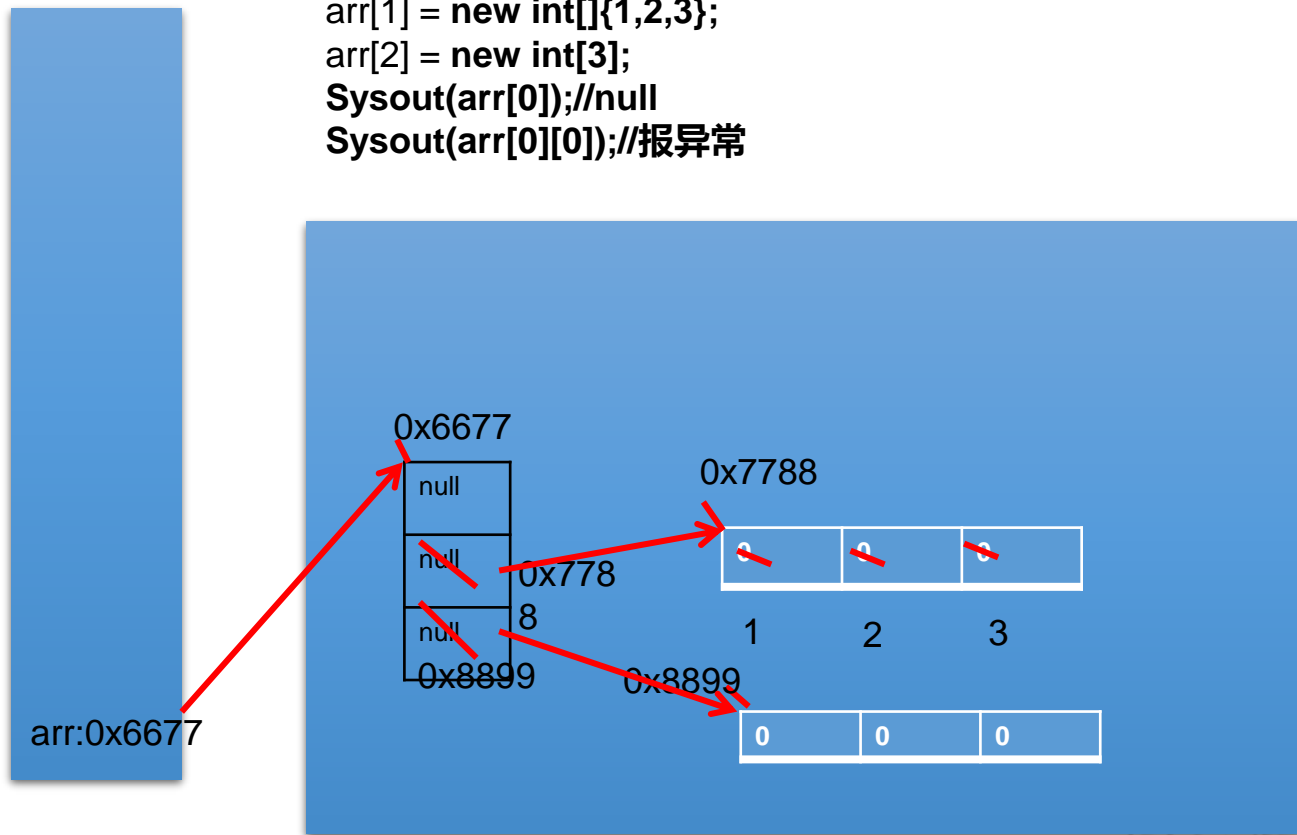
```
int[][] arr4 = new int[3][];  
sysout(arr4[0]); // null  
sysout(arr4[0][0]); // 报错  
arr4[0] = new int[3];  
arr4[0][1] = 5;  
arr4[1] = new int[]{1,2};
```





### 3.3 多维数组的使用：二维数组内存解析

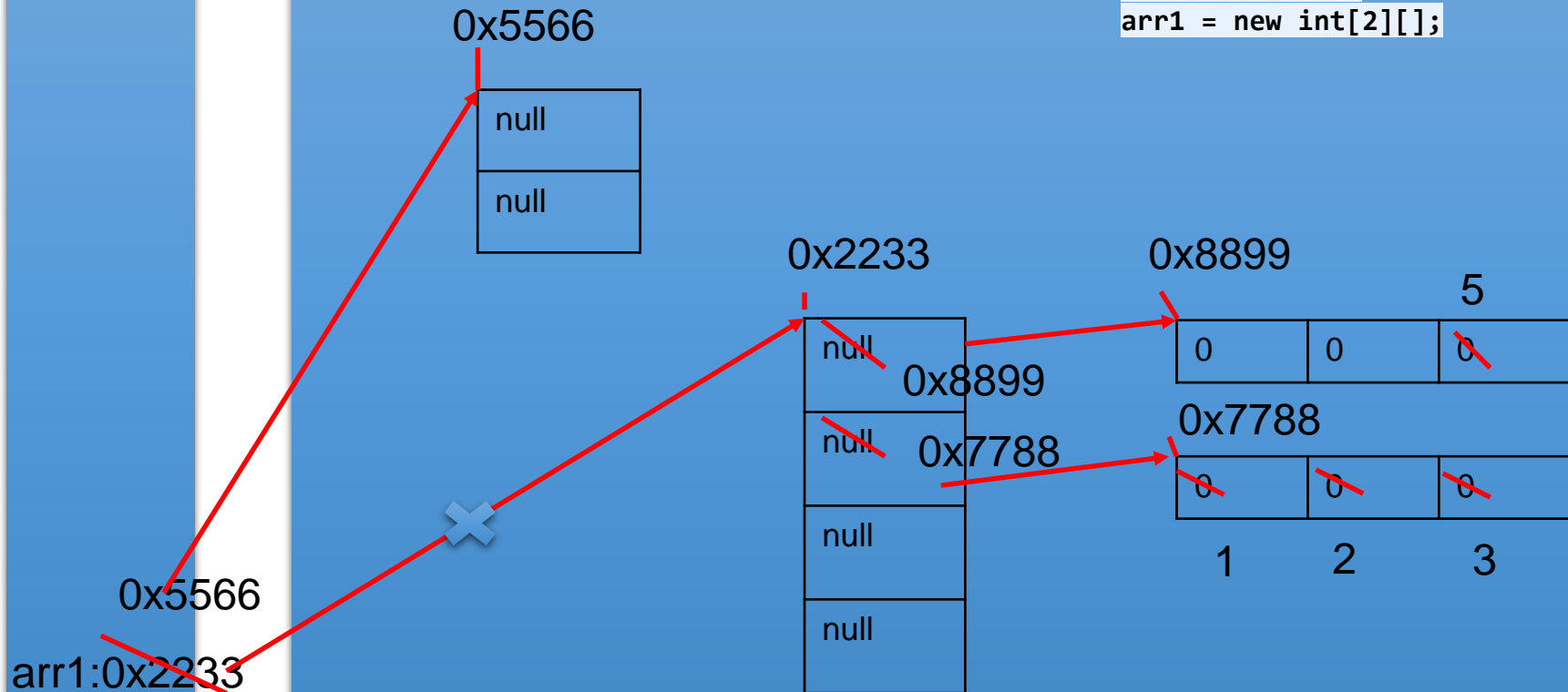
```
int[][] arr = new int[3][];  
arr[1] = new int[]{1,2,3};  
arr[2] = new int[3];  
Sysout(arr[0]); // null  
Sysout(arr[0][0]); // 报异常
```





### 3.3 多维数组的使用：二维数组内存解析

```
int[][] arr1 = new int[4][];  
arr1[0] = new int[3];  
arr1[1] = new int[]{1,2,3};  
arr1[0][2] = 5;  
arr1 = new int[2][];
```





### 练习 2

获取arr数组中所有元素的和。

提示：使用for的嵌套循环即可。

$\begin{matrix} j \\ i \end{matrix}$	j = 0	j = 1	j = 2	j = 3
i = 0	3	5	8	-
i = 1	12	9	-	-
i = 2	7	0	6	4

```
int[][] arr = new int[3][];  
arr[0] = new int[3];  
arr[0][0] = 3;  
...
```





### 练习3

声明： `int[] x,y[]`; 在给`x,y`变量赋值以后， 以下选项允许通过编译的是：

- a) `x[0] = y;` no
- b) `y[0] = x;` yes
- c) `y[0][0] = x;` no
- d) `x[0][0] = y;` no
- e) `y[0][0] = x[0];` yes
- f) `x = y;` no

提示：

一维数组： `int[] x` 或者 `int x[]`

二维数组： `int[][] y` 或者 `int[] y[]` 或者 `int y[][]`



### 练习4

使用二维数组打印一个 10 行杨辉三角。

#### 【提示】

1. 第一行有 1 个元素, 第 n 行有 n 个元素
2. 每一行的第一个元素和最后一个元素都是 1
3. 从第三行开始, 对于非第一个元素和最后一个元素的元素。即:  
$$\text{yanghui}[i][j] = \text{yanghui}[i-1][j-1] + \text{yanghui}[i-1][j];$$

[0]	1									
[1]	1	1								
[2]	1	2	1							
[3]	1	3	3	1						
[4]	1	4	6	4	1					
[5]	1	5	10	10	5	1				
[6]	1	6	15	20	15	6	1			
[7]	1	7	21	35	35	21	7	1		
[8]	1	8	28	56	70	56	28	8	1	
[9]	1	9	36	84	126	126	84	36	9	1
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

#### 【拓展之笔试题】

创建一个长度为6的int型数组，要求数组元素的值都在1-30之间，且是随机赋值。同时，要求元素的值各不相同。



## 3-4 数组中涉及到的常见算法



### 3.4 数组中涉及的常见算法

1. 数组元素的赋值(杨辉三角、回形数等)
2. 求数值型数组中元素的最大值、最小值、平均数、总和等
3. 数组的复制、反转、查找(线性查找、二分法查找)
4. 数组元素的排序算法



### 练习5

定义一个int型的一维数组，包含10个元素，分别赋一些随机整数，然后求出所有元素的最大值，最小值，和值，平均值，并输出出来。

要求：所有随机数都是两位数。

提示：

$[0,1) * 90 \rightarrow [0,90) + 10 \rightarrow [10,100) \rightarrow [10,99]$

`(int)(Math.random() * 90 + 10)`



### 练习 6

使用简单数组

- (1) 创建一个名为ArrayTest的类，在main()方法中声明array1和array2两个变量，他们是int[]类型的数组。
- (2) 使用大括号{}，把array1初始化为8个素数：2,3,5,7,11,13,17,19。
- (3) 显示array1的内容。
- (4) 赋值array2变量等于array1，修改array2中的偶索引元素，使其等于索引值（如array[0]=0,array[2]=2）。打印出array1。

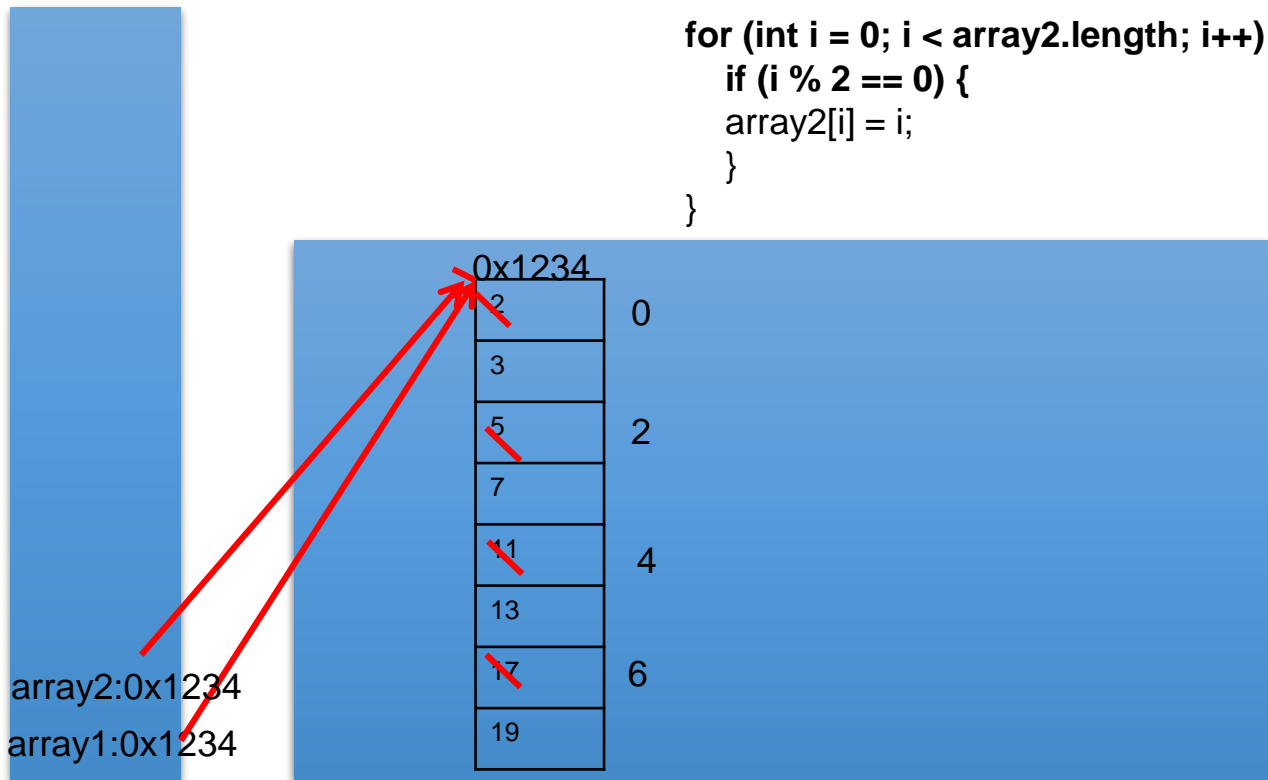
**思考：** array1和array2是什么关系？

**拓展：** 修改题目，实现array2对array1数组的复制



## 3.4 数组中涉及到的常见算法

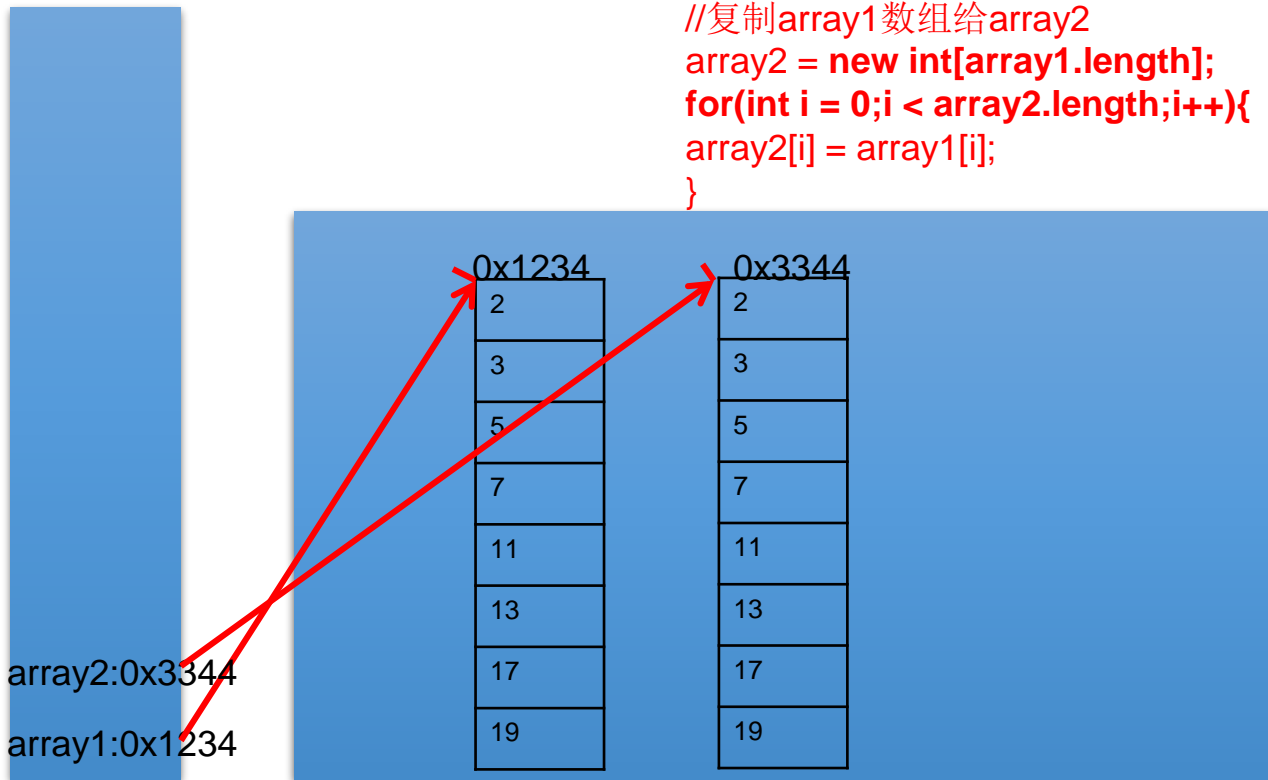
```
int[] array1, array2;  
array1 = new int[] { 2, 3, 5, 7, 11, 13, 17, 19 };  
array2 = array1;  
for (int i = 0; i < array2.length; i++) {  
    if (i % 2 == 0) {  
        array2[i] = i;  
    }  
}
```





## 3.4 数组中涉及到的常见算法

```
int[] array1, array2;  
array1 = new int[] { 2, 3, 5, 7, 11, 13, 17, 19 };  
  
//复制array1数组给array2  
array2 = new int[array1.length];  
for(int i = 0;i < array2.length;i++){  
    array2[i] = array1[i];  
}
```

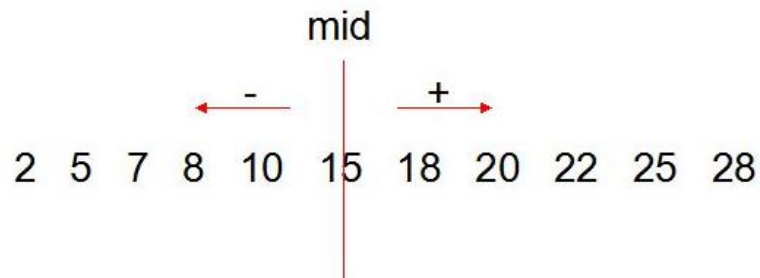




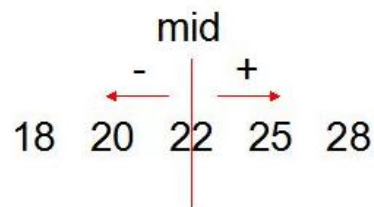


### 3.4 数组中涉及到的常见算法：二分法查找算法

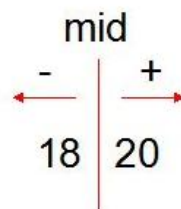
step 1:



step 2:



step 3:





## 3.4 数组中涉及到的常见算法：二分法查找算法

//二分法查找：要求此数组必须是有序的。

```
int[] arr3 = new int[]{-99,-54,-2,0,2,33,43,256,999};
```

```
boolean isFlag = true;
```

```
int number = 256;
```

```
//int number = 25;
```

```
int head = 0; //首索引位置
```

```
int end = arr3.length - 1; //尾索引位置
```

```
while(head <= end){
```

```
    int middle = (head + end) / 2;
```

```
    if(arr3[middle] == number){
```

```
        System.out.println("找到指定的元素，索引为：" + middle);
```

```
        isFlag = false;
```

```
        break;
```

```
    }else if(arr3[middle] > number){
```

```
        end = middle - 1;
```

```
    }else{//arr3[middle] < number
```

```
        head = middle + 1;
```

```
    }
```

```
}
```

```
if(isFlag){
```

```
    System.out.println("未找到指定的元素");
```

```
}
```



# 排序算法

**排序：**假设含有 $n$ 个记录的序列为 $\{R_1, R_2, \dots, R_n\}$ ,其相应的关键字序列为 $\{K_1, K_2, \dots, K_n\}$ 。将这些记录重新排序为 $\{R_{i1}, R_{i2}, \dots, R_{in}\}$ ,使得相应的关键字值满足条 $K_{i1} \leq K_{i2} \leq \dots \leq K_{in}$ ,这样的一种操作称为排序。

➤ 通常来说，排序的目的是快速查找。

### 衡量排序算法的优劣：

- 1.时间复杂度：分析关键字的比较次数和记录的移动次数
- 2.空间复杂度：分析排序算法中需要多少辅助内存
- 3.稳定性：若两个记录A和B的关键字值相等，但排序后A、B的先后次序保持不变，则称这种排序算法是稳定的。



排序算法分类：内部排序和外部排序。

- 内部排序：整个排序过程不需要借助于外部存储器（如磁盘等），所有排序操作都在内存中完成。
- 外部排序：参与排序的数据非常多，数据量非常大，计算机无法把整个排序过程放在内存中完成，必须借助于外部存储器（如磁盘）。外部排序最常见的是多路归并排序。可以认为外部排序是由多次内部排序组成。



# 十大内部排序算法

- 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、折半插入排序、Shell排序
- 归并排序
- 桶式排序
- 基数排序

详细操作，见《附录：尚硅谷\_宋红康\_排序算法.pdf》

让天下没有难学的技术



### 算法的5大特征

输入 ( Input )	有0个或多个输入数据，这些输入必须有清楚的描述和定义
输出 ( Output )	至少有1个或多个输出结果，不可以没有输出结果
有穷性 ( 有限性 , Finiteness )	算法在有限的步骤之后会自动结束而不会无限循环，并且每一个步骤可以在可接受的时间内完成
确定性 ( 明确性 , Definiteness )	算法中的每一步都有确定的含义，不会出现二义性
可行性 ( 有效性 , Effectiveness )	算法的每一步都是清楚且可行的，能让用户用纸笔计算而求出答案

说明：满足确定性的算法也称为：确定性算法。现在人们也关注更广泛的概念，例如考虑各种非确定性的算法，如并行算法、概率算法等。另外，人们也关注并不要求终止的计算描述，这种描述有时被称为过程（procedure）。



### 冒泡排序

#### 介绍：

冒泡排序的原理非常简单，它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。

#### 排序思想：

1. 比较相邻的元素。如果第一个比第二个大（升序），就交换他们两个。
2. 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。
3. 针对所有的元素重复以上的步骤，除了最后一个。
4. 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为止。



# 快速排序

### 介绍：

快速排序通常明显比同为 $O(n\log n)$ 的其他算法更快，因此常被采用，而且快排采用了分治法的思想，所以在很多笔试面试中能经常看到快排的影子。可见掌握快排的重要性。

快速排序（Quick Sort）由图灵奖获得者Tony Hoare发明，被列为**20世纪十大算法之一**，是迄今为止所有内排序算法中速度最快的一种。冒泡排序的升级版，交换排序的一种。**快速排序的时间复杂度为 $O(n\log(n))$ 。**



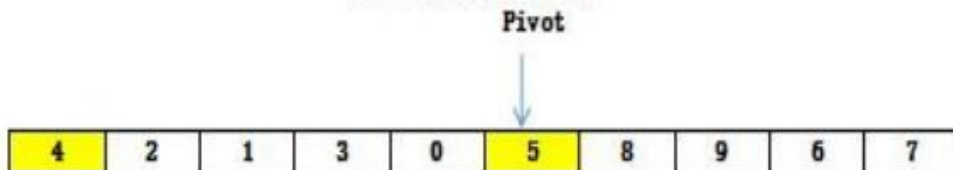
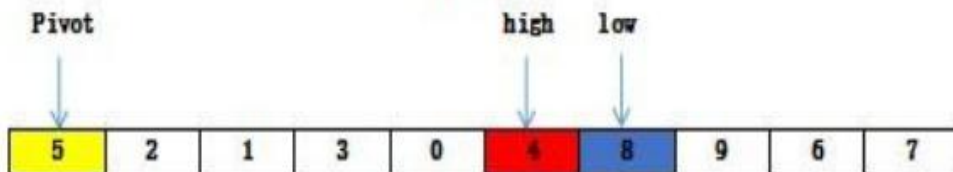
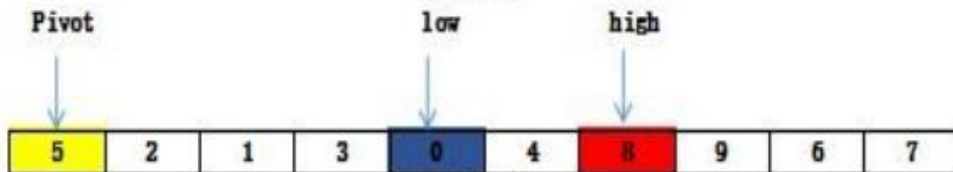
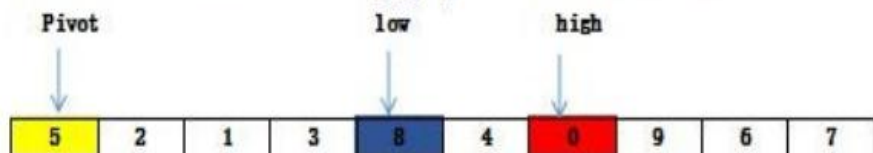
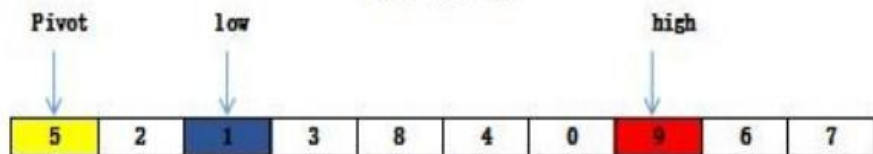
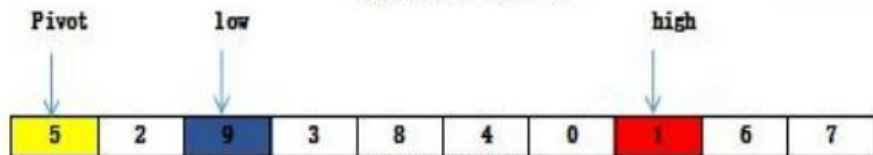
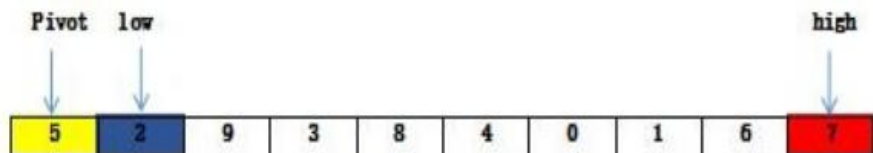


# 快速排序

### 排序思想：

1. 从数列中挑出一个元素，称为"基准"（**pivot**），
2. 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区结束之后，该基准就处于数列的中间位置。这个称为分区（**partition**）操作。
3. 递归地（**recursive**）把小于基准值元素的子数列和大于基准值元素的子数列排序。
4. 递归的最底部情形，是数列的大小是零或一，也就是永远都已经被排序好了。虽然一直递归下去，但是这个算法总会结束，因为在每次的迭代（**iteration**）中，它至少会把一个元素摆到它最后的位置去。

## 快速排序演示





### 3.4 数组中涉及到的常见算法：排序算法

假设: [ 49 38 65 97 76 13 27 49]

第1趟 [ 27 38 13 ] 49 [ 76 97 65 49]

第2趟 [ [13] 27 [38] ] 49 [ [ 49' 65 ] 76 [97] ]

第3趟 [ [13] 27 [38] ] 49 [ [ 49'[65]] 76 [97] ]

最后结果 13 27 38 49 49' 65 76 97



### 3.4 数组中涉及到的常见算法：排序算法性能对比

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	$O(n\log_2 n)$	不稳定
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定
桶排序	$O(n+k)$	$O(n^2)$	$O(n)$	$O(n+k)$	稳定
基数排序	$O(n*k)$	$O(n*k)$	$O(n*k)$	$O(n+k)$	稳定



### 各种内部排序方法性能比较

1. **从平均时间而言**：快速排序最佳。但在最坏情况下时间性能不如堆排序和归并排序。
2. **从算法简单性看**：由于直接选择排序、直接插入排序和冒泡排序的算法比较简单，将其认为是简单算法。对于Shell排序、堆排序、快速排序和归并排序算法，其算法比较复杂，认为是复杂排序。
3. **从稳定性看**：直接插入排序、冒泡排序和归并排序时稳定的；而直接选择排序、快速排序、Shell排序和堆排序是不稳定排序
4. **从待排序的记录数 $n$ 的大小看**， $n$ 较小时，宜采用简单排序；而 $n$ 较大时宜采用改进排序。



### 排序算法的选择

(1)若 $n$ 较小(如 $n \leq 50$ )，可采用直接插入或直接选择排序。

当记录规模较小时，直接插入排序较好；否则因为直接选择移动的记录数少于直接插入，应选直接选择排序为宜。

(2)若文件初始状态基本有序(指正序)，则应选用直接插入、冒泡或随机的快速排序为宜；

(3)若 $n$ 较大，则应采用时间复杂度为 $O(n \lg n)$ 的排序方法：快速排序、堆排序或归并排序。



## 3-5 Arrays工具类的使用



`java.util.Arrays`类即为操作数组的工具类，包含了用来操作数组（比如排序和搜索）的各种方法。

1	<b><code>boolean equals(int[] a,int[] b)</code></b>	判断两个数组是否相等。
2	<b><code>String toString(int[] a)</code></b>	输出数组信息。
3	<b><code>void fill(int[] a,int val)</code></b>	将指定值填充到数组之中。
4	<b><code>void sort(int[] a)</code></b>	对数组进行排序。
5	<b><code>int binarySearch(int[] a,int key)</code></b>	对排序后的数组进行二分法检索指定的值。





### 数组排序

- java.util.Arrays类的**sort()**方法提供了数组元素排序功能:

```
import java.util.Arrays;
public class SortTest {
    public static void main(String[] args) {
        int [] numbers = {5,900,1,5,77,30,64,700};
        Arrays.sort(numbers);
        for(int i = 0; i < numbers.length; i++){
            System.out.println(numbers[i]);
        }
    }
}
```



## 3-6 数组使用中的常见异常



编译时，不报错！！

### 数组脚标越界异常(ArrayIndexOutOfBoundsException)

```
int[] arr = new int[2];  
System.out.println(arr[2]);  
System.out.println(arr[-1]);
```

访问到了数组中不存在的脚标时发生。

### 空指针异常(NullPointerException)

```
int[] arr = null;  
System.out.println(arr[0]);
```

arr引用没有指向实体，却在操作实体中的元素时。

让天下没有难学的技术



尚硅谷