一、 数组

1、数组的特点

1) 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。

2) 创建数组对象会在内存中开辟**一整块连续的空间**。占据的**空间的大小**，取决于数组的**长度**和数组中**元素的类型**。

3) 数组中的元素在内存中是**依次紧密排列**的，**有序的**。

4) 数组，一旦**初始化完成，其长度就是确定的**。数组的长度一旦确定，**就不能修改**。

5) 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。

6) **数组名**中**引用**的是这块连续空间**首地址**。

2、数组的分类

1）按照元素类型分：

- **基本数据类型元素**的数组：每个元素位置存储基本数据类型的值。
- **引用数据类型元素**的数组：每个元素位置存储对象（本质是存储对象的**首地址**）。

2）按照维度分：

- **一维数组**：存储一组数据。
- **二维数组**：存储多组数据，相当于二维表，一行代表一组数据，只是这里的二维表***每一行长度不要求一样***。

3、数组的声明需要明确：1）数组的**维度**；2）数组的**元素类型**；3）数组**名**。

一维数组的声明格式：

*1)元素的数据类型[ ] 一维数组名;**(int[ ] arr)**

2)元素的数据类型 一维数组名[];**(int arr[ ])**

(注：声明数组时**不能指定其长度**。例：int a[5];)

4、一维数组初始化

1）静态初始化：数组**变量的初始化**和数组**元素的赋值**操作**同时**进行。(数组的长度由**静态数据的个数**决定)

格式：

1>数据类型[ ] 数组名 = **new 数据类型[ ]{...};**

int[ ] arr = new int[ ]{1,2,3,4,5};

2>数据类型[ ] 数组名;

数组名 = new 数据类型[ ]{...};

int[ ] arr;

arr = new int[ ]{1,2,3,4,5};

3>数据类型[ ] 数组名 = {...};

int[ ] arr = {1,2,3,4,5};

(注：第 3 点不能分成两个语句写成这种：

int[ ] arr; arr = {1,2,3,4,5};)

2）动态初始化：数组**变量的初始化**和数组**元素的赋值**操作**分开**进行。(**只确定数组长度**，表示最多存储多少元素，**元素值为默认值**)

格式:(声明的时候不能写长度，**new 的时候写**)

1>数组存储的元素的数据类型[ ] 数组名字 = new 数组存储的元素的数据类型[长度];

int[ ] arr = new int[5];

2>数组存储的数据类型[ ] 数组名字;

数组名字 = new 数组存储的数据类型[长度];

int[ ] arr; arr = new int[5];

(注：{ }指定元素列表和[ ]指定元素个数选一；

例**错误**：int[ ] arr = new int[5]{1,2,3,4,5};)

5、数组的长度（属性 length；arr.length）

注：一旦**初始化**，其长度就是**确定，且不可变**。

6、数组元素的默认值

对于**基本**数据类型，默认**初始化值各有不同**。

对于**引用**数据类型，默认**初始化值为 null**。

byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0
char	0 或'\u0000'(表现为空)
boolean	false
引用类型	null

7、Java 虚拟机的内存划分为 5 部分：

程序计数器/虚拟机栈/本地方法栈/堆/方法区;

8、一维数组在内存中的存储

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

程序执行流程：

1)main 方法进入方法栈执行；

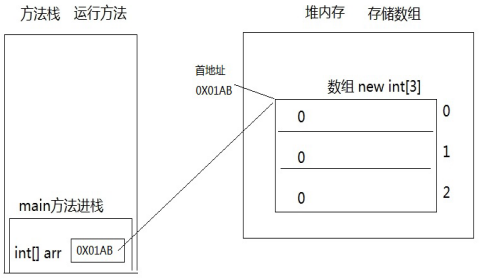
2)创建数组，JVM **在堆内存开辟空间存储数组**;

3)数组在内存中有内存地址(十六进制数表示)；

4)数组中有 3 个元素，**默认值为 0**；

5)JVM 将**内存首地址**赋值给**引用类型变量 arr;**

6)变量 arr 保存的是数组内存地址，而不是具体数值，因此称为引用数据类型。



9、数组下标从 0 开始的原因

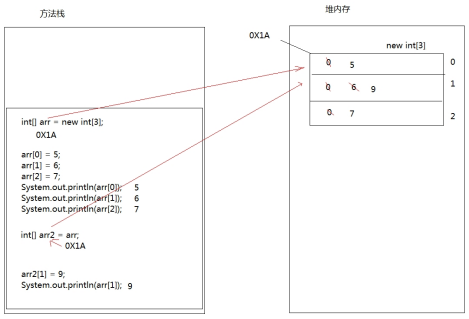
：因为第一个元素**距离数组首地址间隔 0 个单元格**。表示***偏移量***。

10、两个变量指向一个一维数组

int[ ] arr = new int[3];

//定义数组变量 arr2，将 arr 的地址赋值给 arr2

int[ ] arr2 = arr;



11、二维数组的声明格式

***1>元素的数据类型[ ][ ] 二维数组的名称;**

2>元素的数据类型 二维数组名[ ][ ];

3>元素的数据类型[ ] 二维数组名[ ];

(注：int[ ] x, y[ ];//x 是一维数组，y 是二维数组)

12、二维数组初始化

1）静态初始化：

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

2>int[ ][ ] arr;

arr = new int[ ][ ]{{1,2,3},{4,5,6},{7,8,9,10}};

3>int[ ][ ] arr = {{1,2,3},{4,5,6},{7,8,9,10}};

(注：第 3 点不能分成两个语句写)

2）动态初始化：

1>规则二维表：**每一行的列数是相同的**

a)确定**行数和列数**

元素的数据类型[][] 二维数组名 = new 元素的数据类型[m][n]; 例：int[ ][ ] arr = new int[3][2];

b)再**为元素赋新值**

二维数组名[行下标][列下标] = 值;

2>不规则：**每一行的列数不一样**

a)先确定**总行数**

元素的数据类型[ ][ ] 二维数组名 = new 元素的数据类型[m][ ]; **int[ ][ ] arr = new int[3][ ];**

//只是确定了总行数，每一行里面现在是 null;

b)再确定**每一行的列数，创建每行的一维数组**;

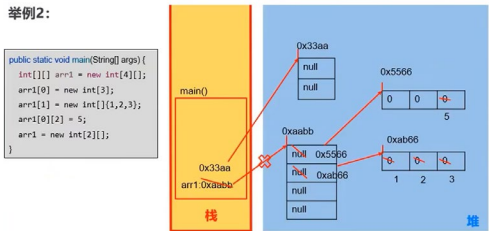
二维数组名[行下标] = new 元素的数据类型[该行的总列数]; arr[0] = new int[3]; arr[1] = new int[1]; arr[2] = new int[2]; //此时 new 完的行的元素有默认值，没有 new 的行还是 null;

c)再为**元素赋值**

二维数组名[行下标][列下标] = 值;

13、二维数组内存解析

二维数组本质上是元素类型是一维数组的一维数组。



14、类型相同，维数相同才能赋值操作。

15、冒泡排序优化：设置 flag 表示是否已有序。

16、内部排序性能比较与选择

1）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。

2）选择

1>若 **n 较小**(如 n≤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(nlgn)** 的排序方法：**快速**排序、**堆**排序或**归并**排序。

17、Arrays 工具类的使用

1）比较两个数组的元素是否依次相等

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

int[ ] arr2 = new int[ ]{1,2,3,4,5};

int[ ] arr3 = new int[ ]{1,2,3,5,4};

system.out.println(arr1 == arr2)//比较地址

boolean isEquals = **Arrays.equals(arr1,arr2)**True

boolean isEquals=**Arrays.equals(arr1,arr3)**False

2）输出数组元素信息

system.out.println(arr1);//输出地址

system.out.println(**Arrays.toString**(arr1));

3）将指定值填充到数组之中

Arrays.fill(arr1,**10**)//[10,10,10,10,10]

4）使用快速排序实现的排序

Arrays.sort(arr3);//**整体**排序

Arrays.sort(arr3, fromIndex, toIndex);

//索引[fromIndex, toIndex)**部分按照升序排列**

5）二分查找（前提：当前数组**必须有序**）

Int index = **Arrays.binarySearch**(arr3,**2**);

6)数组的复制

1>**Arrays. copyOf(srcArray, int length);**

//从原数组的第一个元素开始复制，目标数组的长度为 length。如果 length> srcArray.length，则目标数组中**采用默认值填充**（原数组不够长）；如果 length 小于 srcArray.length，则复制到第 length 个元素（索引值为 length-1）即止。

2>Arrays.copyOfRange(srcArray, int startIndex, int endIndex);//[startIndex, endIndex)

endIndex 必须大于等于 startIndex，如果大于 srcArray.length，则**目标数组中使用默认值填充**。

(注：**目标数组**如果已经存在，**将会被重构**。)

18、数组中的常见异常

1>**越界**异常:ArrayIndexOutOfBoundsException;

2>**空指针**异常：NullPointerException;

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

System.out.println(arr[0][0]);

//此时 arr[0]是 null 还未分配具体存储元素的空间。