第四章 数组

1 数组概述

> 数组，是一组数据的集合，数组中的每个数据被称为元素。

如果有五个数据1 2 3 4 5，需要去接收、保存、操作这些数据，那么需要五个变量：

```
int a1 = 1;
int a2 = 2;
int a3 = 3;
int a4 = 4;
int a5 = 5;
```

除此之外，还可以选择使用一个组数来保存这五个数据：

```
int[] arr = {1,2,3,4,5};
//这里就是使用了一个数组，来保存这五个数据
```

在java中，数组也是对象。数组中的元素可以是任意类型（基本类型或引用类类型），但同一个数组里只能存放类型相同的元素。

main方法的参数，就是一个String类型的数组：

```
public static void main(String[] args){ //这个参数args的类型是字符串数组

    //控制循环输出的次数
    int num = 1;

    //如果main方法的参数args有接收到参数值的话
    if(args.length > 0){
        //把接收到的值转换为int类型，并赋值给变量num
        num = Integer.parseInt(args[0]);
    }
    //循环输出hello，默认输出的次数为1，如果用户给main方法传参了，则按照用户的要求的次数进行输出
    for(int i =0;i<num;i++){
        System.out.println("hello");
    }

}
```

> args.length 是获取这个数组对象中的元素个数
> args[0] 是获取这个数组中，第一个元素的值
> Integer.parseInt()方法是将一个字符串转换为int数字，例如，"109" ==> 109

使用java命令运行类中main方法，并给其传参：

`java com.briup.day05.PrintHello 20`

这里的这个20就是给类中的main方法进行参数的，这个参数会存放到args这个数组对象中。

2 数组类型

一般情况下，java中的类（class），是需要提前把代码写好，然后编译成class文件，再加载到内存中，最后在程序中就可以使用到这个类了。

而数组和类不同，数组是使用当前已经存在的某一个类型（任意类型都可以），然后再这个类型后面加上一对中括号（[]），这时就组成一个了新的类型：数组类型

例如，任意类型 + [] = 相应的数组类型

```
byte   + []  ---> byte[]
short  + []  ---> short[]
int    + []  ---> int[]
long   + []  ---> long[]

float  + []  ---> float[]
double + []  ---> double[]
char   + []  ---> char[]
boolean+[]   ---> boolean[]

String+ []   ---> String[]
Action+ []   ---> Action[] //Action假设是一个接口
int[] + []   ---> int[][]  //一维数组+[] 变为二维数组
```

> 数组中可以存放一组数据，要求这一组数据的类型是一样（或者兼容的，兼容就表示可以自动转换）。
> 例如，int类型数组（int[]）中，也是可以存放byte数据的，因为byte可以自动转换为int（兼容）

3 数组变量

> 先使用一个已有的类型，然后加上中括号，组合成一个数组类型，然后再声明这个数组类型的变量

俩种方式声明一个数组类型的变量：

1. `int[] a;`
2. `int a[];`

推荐使用第一种声明的方式

其他一些声明数组变量的示例：

```
String[] str;
Student[] stus;
Object[] objs;
long[] arr;
int[][] arrOfArr;
```

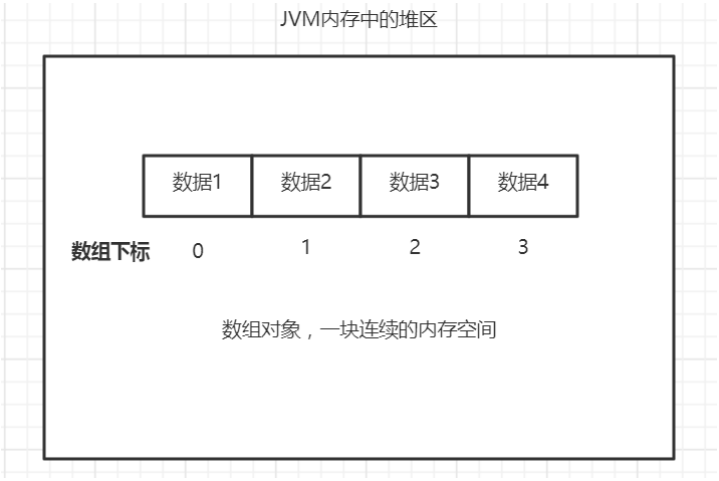
数组类型的变量，也是引用类型变量，简称引用，它是可以指向对象的（数组对象）

4 数组对象

使用new关键字，来创建数组对象，中括号里面的数字，就是数组的长度。

```
int[] a = new int[4];//数组对象a中，最多存放4个int类型的数据
String[] s = new String[2];//数组对象s中，最多存放2个String类型的数据
char[] c = new char[1];//数组对象c中，最多存放1个char类型的数据

```

> 数组对象，在内存中，就是一块连续的内存空间，在这个连续的空间中，可以存放多个类型相同的数据。



思考，数组对象中都有哪些属性和方法可以使用？

数组对象中只有一个属性（length），表示数组的长度。

数组对象中的方法，只有从父类型Object中继承过来的方法。

除此之外，数组对象中就没有其他属性和方法了。

5 数组长度

数组对象的长度：

1. 数组长度，是指在一个数组对象中，最多可以存放多少个同一类型的数据
2. 数组长度，必须在创建数组对象的时候就明确指定
3. 数组长度，一旦确定，就无法再改变
4. 数组长度，可以为0，但是不能为负数

例如，错误的一些情况

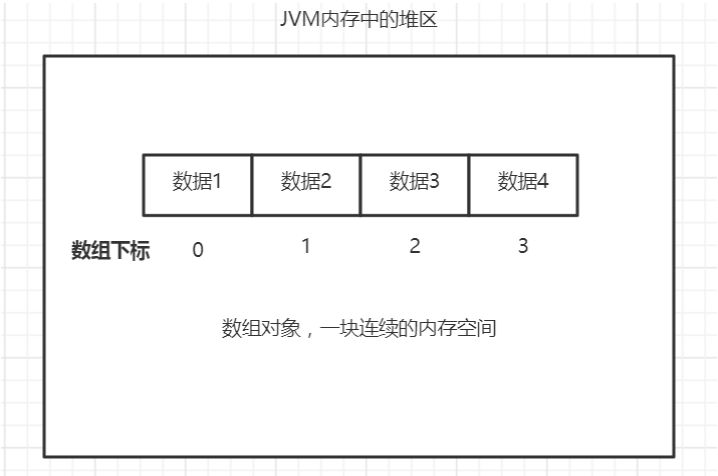
```
//编译报错，因为没指定数组的长度
byte[] b = new byte[];
//编译通过，但是运行报错，因为数组的长度不能为负数
byte[] b = new byte[-1];
```

例如，使用length属性，获取数组对象的长度

```
int[] a = new int[4];
System.out.println(a.length);
```

6 数组下标

> 数组对象在JVM的堆区中是一块连续的内存空间，并允许使用下标，设置或获取每一个内存空间的值



数组的下标的区间为，[0,arr.length-1]，假如数组的长度为length的话，那么数组下标的最小值为0，数组下标的最大值就是length-1

如果使用下标访问数组中元素的时候，下标的值超出了其可用范围，那么运行就会报错：

```java
int[] a = new int[4];//数组长度为4，那么其下标就是0~3
a[4] = 359;//这句代码运行时报错，下标超出了最大范围
```

可以通过数组下标，给数组某个下标位置上**赋值**：

```java
int[] a = new int[4];
a[0] = 337;
a[1] = 340;
a[2] = 348;
a[3] = 352;
```

可以通过数组下标，**获取**数组某个下标位置上的值：

```java
int[] a = new int[4];
System.out.println(a[0]):
int num = a[0]+a[1]+a[2]+a[3];
System.out.println(num);
```

可以结合循环来进行赋值或者取值：

```java
int[] a = new int[4];

//i从0开始，到a.lenght-1,那么i的取值刚好是数组a的下标可用范围
for(int i=0;i<a.length;i++){
    a[i] = 10+i;
}

//获取数组中每个下标的值，并且输出
for(int i=0;i<a.length;i++){
    System.out.println(a[i]);
}
```

7 默认值

> 一个数组对象在创建的时候，需要指定数组长度，表示数组中最多存放的元素个数，并且在数组对象创建完成之后，数组中每一个元素位置上，就已经有了相应的默认值，这个默认值和数组的类型有关。

```java
//byte、short、int、long类型数组中的默认值为 0
//例如，
int[] a = new int[4];//默认4个数据全是0

//float、double类型数组中的默认值为 0.0
```

```
//例如，
double[] d = new double[4];//默认4个数据全是0.0

//boolean类型数组中的默认值为 false
//例如，
boolean[] d = new boolean[4];//默认4个数据全是false

//char类型数组中的默认值为 '\u0000'
//例如，
char[] d = new char[4];//默认4个数据全是'\u0000'

//引用类型数组中的默认值为 null
//例如，
String[] d = new String[4];//默认4个数据全是null
```

8 创建方式

> 创建数组对象的形式有多种，每一种都要求大家掌握

下面以创建int类型数组对象进行说明

1. int[] arr = new int[5];
   创建数组对象，没有还没有给其赋值
2. int[] arr = new int[]{1,3,5,7,9};
   创建数组对象的同时，并赋值
3. int[] arr = {1,3,5,7,9};
   java提供了创建数组对象的简便形式
4. int[] arr;
   arr = new int[]{1,3,5,7,9};
   显示声明数组类型变量，然后创建对象，并赋值

以下一些**错误**的数组创建方式：

错误方式**一**：

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

错误方式**二**：

int[] a;

a = {1,2,3};

9 数组拷贝

> 数组对象的长度确定之后便不能修改，但可以通过复制数组的内容变通实现改变数组长度

在java.lang.System类中提供一个名为arraycopy的方法可以实现复制数组中元素的功能

```
//该方法的声明
public static void arraycopy(Object src,
                             int srcPos,
                             Object dest,
                             int destPos,
                             int length)
//参数1,需要被复制的目标数组
//参数2,从目标数组的哪一个位置开始复制
//参数3,需要把数据复制到另外一个新的数组中
//参数4,把数据复制到新数组的时候，需要把数据从什么位置开始复制进去
//参数5,复制的目标数组的长度
```

例如，写一个方法,接收一个数组对象,把这个数组对象的长度扩大到原来的2倍并返回。

```
public int[] test(int[] a){
    int[] b = new int[a.length*2];
    System.arraycopy(a, 0, b, 0, a.length);
    return b;
}
```

10 工具类

> 由于数组对象本身并没有什么方法可以供我们调用，但API中提供了一个工具类Arrays供我们使用，从而可以对数据对象进行一些基本的操作。

java.util.Arrays类，是JAVASE API中提供给我们使用的一个工具类，这个类的作用就是在代码中辅助我们对数组对象进行操作的。里面有很多静态方法可以直接调用，主要的功能就是对数组对象进行操作，例如排序、查询、复制、填充数据等等。

Arrays中的常用方法：

- toString方法
  可以把一个数组变为对应的String形式
- copyOf方法

  可以把一个数组进行复制
  该方法中也是采用了arraycopy方法来实现的功能
- copyOfRange方法

  也是复制数组的方法，但是可以指定从哪一个下标位置开始复制
  该方法中也是采用了arraycopy方法来实现的功能
- sort方法
  可以对数组进行排序
- binarySearch方法
  在数组中，查找指定的值，返回这个指定的值在数组中的下标，但是查找之前需要在数组中先进行排序，可以使用sort方法先进行排序

- equals方法
  可以比较俩个数组是否相等,但是要求俩个数组中的值一一相等并且顺序也要一致才行，所以在比较之前，我们最好是用sort方法对俩个数组先进行排序
  第一个要求，俩个数组的元素个数相同
  第二个要求，俩个数组的每个下标对应的数据相同
- fill
  可以使用一个特定的值，把数组中的空间全都赋成这个值
- asList
  可以把一组数据，封装到一个List集合中，并且把list集合对象返回。

例如，

```
//注意，数组长度一旦确定，就不能再修改了。
//我们只是创建了一个新的数组，并且把老数组中的数据复制到了新数组中，并且把新数组返回给用户，这个效果让人感觉上像是老的数组的长度变长了，但其实并没有。
int[] a = {1,3,5,2,6,8};

System.out.println(Arrays.toString(a));

a = Arrays.copyOf(a,10);

System.out.println(Arrays.toString(a));

Arrays.sort(a);

System.out.println(Arrays.toString(a));

int index = Arrays.binarySearch(a,5);
System.out.println(index);

int[] arr1 = {1,2,3};
int[] arr2 = {3,2,1};
Arrays.sort(arr1);
Arrays.sort(arr2);
System.out.println(Arrays.equals(arr1,arr2));

Arrays.fill(a,100);
System.out.println(Arrays.toString(a));
```

11 案例

例如，实现一个方法，参数是int[]，该方法可以计算出数组中所有数据的平均值，并且把结果返回

```java
public double test(int[] arr){

    int length = arr.length;

    int sum = 0;

    for(int i=0;i<length;i++){
        sum = sum + arr[i];
    }

    return sum/length;
}
```

例如，实现一个方法，参数是int[]，该方法可以计算出数组中所有数据的最大值，并且把结果返回。

```java
public int test(int[] arr){
    //max变量中的值，表示当前找到的最大值
    //假设数组中下标为0的位置是最大值
    //这步假设赋值的目的就是为了给局部变量max一个初始值
    int max = arr[0];

    for(int i=1;i<arr.length;i++){
        if(arr[i]>max){
            max = arr[i];
        }
    }
    return max;
}
```

例如，实现一个方法，参数是int[]，该方法可以对数组进行排序，使用【冒泡排序】，该方法不需要返回值。
规则：在一组数据中，从左到右，俩俩比较，然后把较大的数据往前推，一轮下来之后，最大的一个数据就被推到了最右边。

例如：2 1 4 9 8 5 3

第一轮第1次比较后 1 2 4 9 8 5 3
第一轮第2次比较后 1 2 4 9 8 5 3
第一轮第3次比较后 1 2 4 9 8 5 3
第一轮第4次比较后 1 2 4 8 9 5 3
第一轮第5次比较后 1 2 4 8 5 9 3
第一轮第6次比较后 1 2 4 8 5 3 9

```java
//冒泡排序
public void test(int[] arr){
    //获取到数组的长度
    int len = arr.length;

    //外层循环控制一共比较几轮
    for(int i=0;i<len-1;i++){
        //内层循环控制每轮比较多少次数，每轮比较的次数是有变化的
        for(int j=0;j<(len-i-1);j++){
            //比较俩个相邻数据的大小
            //如果前一个数据比后一个数据大
```

```
            if(arr[j]>arr[j+1]){
                //交互这个俩个数据的位置
                arr[j]   = arr[j] ^ arr[j+1];
                arr[j+1] = arr[j] ^ arr[j+1];
                arr[j]   = arr[j] ^ arr[j+1];
            }
        }
        //这个输出，可以看到每一轮比较结束后的结果是怎样的
        System.out.println("\t"+Arrays.toString(arr));
    }

}
```

例如，实现一个方法，参数是int[],该方法可以对数组进行排序，使用【**选择排序**】，该方法不需要返回值。

规则：每一轮在待排序的区域中比较找到一个最小值后，把这个最小值放到已经排好顺序的区域的末尾，剩下的部分，组成一个新的待排序部分，重复上面的步骤直到排序结束。

```
假设待排序的数组为：2 1 4 9 8 5 3

第一轮从待排序区中找到一个最小值，然后和第一个位置的数字交互位置
        1 2 4 9 8 5 3

第二轮从新的待排序区中找到一个最小值，然后和第二个位置的数字交互位置
        1 2 4 9 8 5 3

第三轮从新的待排序区中找到一个最小值，然后和第三个位置的数字交互位置
        1 2 3 9 8 5 4

第四轮从新的待排序区中找到一个最小值，然后和第四个位置的数字交互位置
        1 2 3 4 8 5 9

第五轮从新的待排序区中找到一个最小值，然后和第五个位置的数字交互位置
        1 2 3 4 5 8 9

第六轮从新的待排序区中找到一个最小值，然后和第六个位置的数字交互位置
        1 2 3 4 5 8 9


```

```
//注意，操作的核心目标：
//    1.每轮找到的最小值应该存放的下标位置是什么
//    2.每轮找到的最小值现在的下标位置是什么
//    3.找到之后，让这俩个位置的值进行交互就可以了
//    4.注意，最后一个数字就不用在比较了（前面都排好了，最后一个一定是最大的）
public void test(int[] arr){
    //数组的长度
    int len = arr.length;

    //min_now_index表示最小值【当前】在什么位置
    int min_now_index;
    //min_should_index表示最小值【应用】在什么位置
    int min_should_index;
```

```
14
15      //外层循环，控制一共要比较多少轮，同时这个变量i，刚好是每轮我们需要指定的最小值应该存放位置。
16      for(int i=0;i<len-1;i++){
17          //每一轮i的值，刚好就是本轮最小值应该存放的位置。
18          min_should_index = i;
19
20          //假设当前i的位置就是本轮最小值的实际位置
21          min_now_index = i;
22
23          //内层循环，负责每轮找出当前未排序区中的一个最小值的实际位置的下标
24          for(int j=i+1;j<len;j++){
25              //哪个数据小，就把这个数据下标赋值给min_now_index
26              //目的是让了min_now_index变量中始终保持当前未排序区中的最小值的位置
27              if(arr[j]<arr[min_now_index]){
28                  min_now_index = j;
29              }
30          }
31          //内层循环结束后，就明确了当前未排序区中的最新值的下标，以及这个最小值应该存放在什么位置
32
33          //接下来可以进行交互位置
34          if(min_now_index != min_should_index){
35              int temp = arr[min_now_index];
36              arr[min_now_index] = arr[min_should_index];
37              arr[min_should_index] = temp;
38          }
39
40      }
41
42  }
```

例如，实现一个方法，参数是int[]，该方法可以对数组进行排序，例如使用**【插入排序】**，该方法不需要返回值。

规则：把数组分成俩部分，将后面部分的数据一个一个的和前面部分数据的元素进行比较，如果我们指定的元素比前面部分的元素小，那么就将前面的元素往前移动一步，直到没有比我们这个指定元素还小元素了，那么这个位置就是需要插入的地方。

思路：假设待排序的数组为 2 1 4 9 8 5 3

可以把这个数组，从2的位置进行分开，分成俩组，左边一组是2，右边一组是 1 4 9 8 5 3 ，左边一组是已经排好顺序的序列，右边一组是没有排好顺序的序列。

从右边的待排序序列中，依次拿出每一个值，和左边已经排好顺序的数字进行比较。

比较的目的，是为了给当前操作的右边数字，找一个合适的位置进行插入，注意这个操作数字的位置，有可能是不需要移动的，因为刚好不移动就保持了正常顺序。

```
1  举例说明：
2        |
3      2 | 1 4 9 8 5 3
```

```
        |
```

左边：已经排好顺序的区域
右边：待排序区域

从右边依次拿出一个数据，插入到左边已经排好顺序的序列中，但是需要我们找出一个合适的为，将这个值插入。

拿到右边一个值之后，怎么在左边找出一个合适的位置？
假设当前数组是处于排序中的这个状态：

```
            |
1 2 4 8 9 | 5 3
            |
```

从右边拿出一个值：5
依次和左边排好序的数字比较，从大到小比：先和9比 在和8比 ..直到找到一个比当前数字5要小的位置，那么我们就需要把5插入到这个位置的后面，在当前这个例子中，也就是要插入到8的位置（8和9都要往后依次移，把位置留出给5）

为了能成功的完成这个插入的操作，那么数字9就往前移动一个位置，数字8也需要往前移动一个位置，移动的过程如下：

当前是这种情况：（**特别注意，需要提前把5保存起来，以便后面的插入操作**）

```
            |
1 2 4 8 9 | 5 3
            |
```

9往前移动一个位置：

```
            |
1 2 4 8 9 | 9 3
            |
```

8往前移动一个位置：

```
            |
1 2 4 8 8 | 9 3
            |
```

把我们【提前保存】下来的数字5，插入到原来8的位置：

```
            |
1 2 4 5 8 | 9 3
            |
```

这时候我们排序区的数字就又多了一个：

```
              |
1 2 4 5 8 9 | 3
              |
```

然后拿着3进行再重复上一轮的比较，最后就完成了排序的效果

```
//排序操作
//每轮操作完核心目标：
```

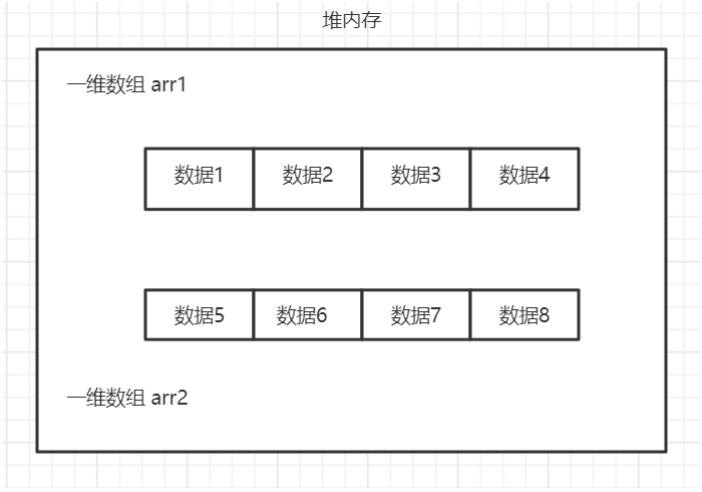
```java
// 1 拿到右边当前要操作的数据，这个值需要临时的保存起来，以便向左边插入合适的位置
// 2 找出左边一个合适的插入位置
// 3 最后把右边当前操作的数据，插入到左边合适的位置
// 4.注意，最后一个数字也需要进行比较，因为要插入到左边一个合适的位置
public void test(int[] arr){

    //输出排序前的数组的排列
    System.out.println(Arrays.toString(arr));


    //当前在右边拿到的第一个要进行操作的数据
    int currentValue;
    //需要把数据在左边插入的位置
    int insertPosition;

    //外层循环，控制比较的轮数
    //同时，变量i的值，还是每一轮我们要操作的右边第一个数字的下标
    for(int i=1;i<arr.length;i++){
        //提前保存好我们当前要操作的值
        currentValue = arr[i];
        //假设当前变量i的值就是要插入的位置，因为这个数据有可能是原位置不动的。
        insertPosition = i;

        //内存循环，控制每轮比较的次数,以及比较的顺序
        //同时，变量j的值，还是左边数据中从大到小的下标值
        //例如：1 2 4 8 9 | 5 3 这个时候
        //我们拿着数字5 要依次和左边的9 8 4 2 1 比较
        //9 8 4 2 1的下标顺序就是 3 2 1 0 ，这就是j的值的变化规律
        for(int j=i-1;j>=0;j--){
            //每次比较，如果发现arr[j]比当前要操作的数字大
            if(arr[j]>currentValue){
                //就把这个大的数字往后移动一个位置，就是往后赋值
                arr[j+1] = arr[j];
                //然后记录一下这个位置，因为这个位置很可能是要插入的位置，到底是不是这个位置，需要和下一个数字比较后才知道
                insertPosition = j;
            }else{
                //如果发现一个比currentValue值还小的值，那么这个值的上一个比较的位置就是我们要找的插入的位置，结束当前循环
                break;
            }
        }
        //内层循环结束后，把当前要操作的值currentValue，插入到指定位置insertPosition
        //如果insertPosition和当前i值相等，说明当前操作的这个值currentValue是不需要移动的。
        if(insertPosition != i){
            //进行值的插入
            //把当前右边第一个值（正在操作的值），插入到左边合适的位置
            arr[insertPosition] = currentValue;
        }
        //输出每一轮排序的过程
        System.out.println("\t"+Arrays.toString(arr));
    }
    //最后输出排序的结果
    System.out.println(Arrays.toString(arr));
}
```

12 二维数组

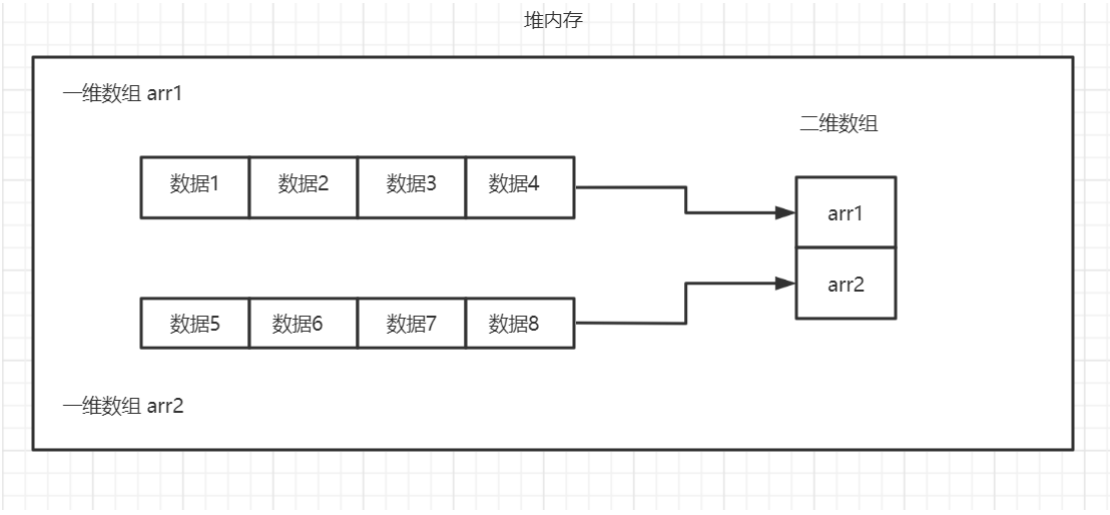
> 如果把普通的数组（一维数组），看作一个小盒子的话，盒子里面可以存放很多数据，那么二维数组就是像一个大点的盒子，里面可以存放很多小盒子（一维数组）

12.1 理解二维数组

例如，一维数组中存放的是普通数据



例如，二维数组中存放的是一维数组对象



> 可以看出，二维数组里面存放的是一维数组（array Of array）

例如，

```
int +   []  =   int[] ，所以一维数组int[]中存放的是数据是int类型数据

int[]   +   []  =   int[][] ，所以二维数组int[][]中存放的是数据是int[]类型数据，也就是一维数组对象
```

> 任何一个一维数组，再加一对中括号[]，就变成了二维数组

例如，

```
//三个一维数组
int[] a1 = {1,2,3};
int[] a2 = {2,3,4};
int[] a3 = {3,4,5};

//二维数组中，存放三个一维数组对象
int[][] arr = {
    a1,
    a2,
    a3
};
//等价于上面的代码
int[][] arr = {
    {1,2,3},
    {2,3,4},
    {3,4,5}
};

```

12.2 声明和创建

```
//第一个中括号中的4，代表这个二维数组对象里面，最多可以存放4个一维数组
//第二个中括号中的3，代表这个二维数组中，每个存放的一维数组对象，都可以存放存放3个int数据
int[][] a = new int[4][3];

//这句代码等价于以下代码

int[][] a = new int[4][];
a[0] = new int[3];
a[1] = new int[3];
a[2] = new int[3];
a[3] = new int[3];

//这句代码等价于以下代码

int[][] a = {
        {0,0,0},
        {0,0,0},
        {0,0,0},
        {0,0,0}
};
```

> 可以把这个二维数组，理解为一栋大厦，共有4层楼，每层楼有3间房，每个房间可以存放一个int数据，现在每个房间的默认值都是0

思考：把二维数组看作一栋大厦的话，假设共4层楼，那么每层楼的房间数是否可以不同？

每层楼的房间数，可以不同，使用代码可以如下表示：

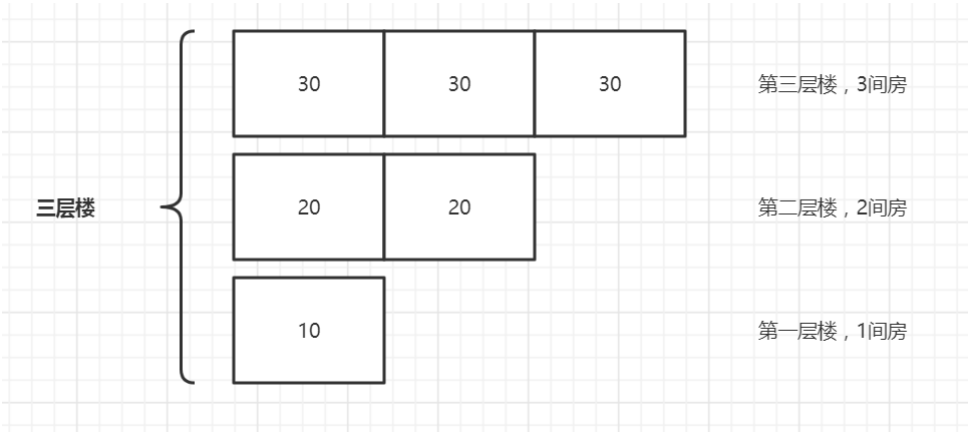
```
int[][] a = new int[4][]; //共4层楼
//每次楼的房间数不同
a[0] = new int[3];
a[1] = new int[5];
a[2] = new int[2];
a[3] = new int[4];

//也可以这样来表示
int[][] a = {
    {1,2,3},
    {1,2,3,4,5},
    {1,2},
    {1,2,3,4}
};

```

12.3 赋值

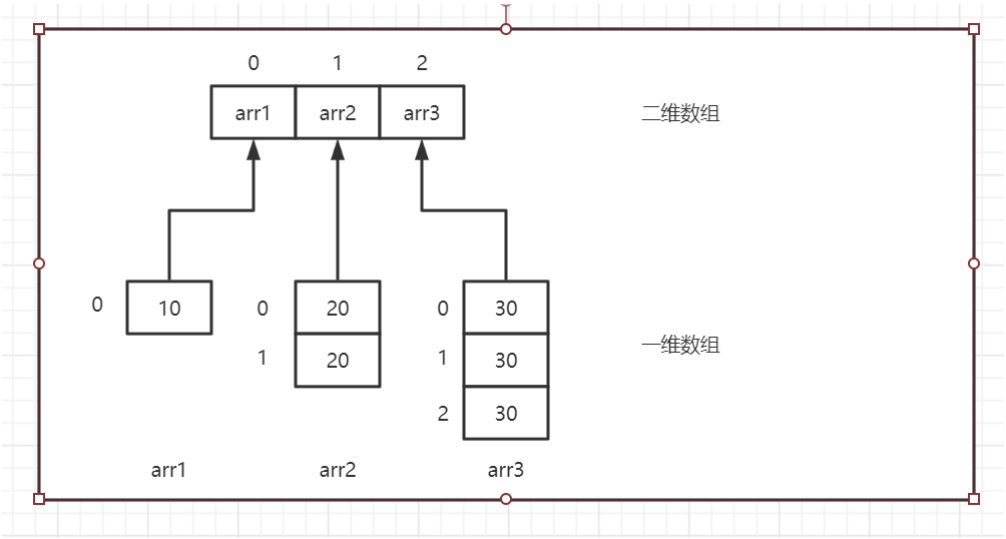
例如，

```
int[][] a = new int[3][];//三层楼
a[0] = new int[1];//第一层楼，1间房
a[1] = new int[2];//第二层楼，2间房
a[2] = new int[3];//第三层楼，3间房

a[0][0] = 10; //第一层楼，第一间房，存放数据

a[1][0] = 20; //第二层楼，第一间房，存放数据
a[1][1] = 20; //第二层楼，第二间房，存放数据

a[2][0] = 30; //第三层楼，第一间房，存放数据
a[2][1] = 30; //第三层楼，第二间房，存放数据
a[2][2] = 30; //第三层楼，第三间房，存放数据
```

可以想象为以下情况：



在JVM内存中的情况如下：



12.4 取值

```
String[][] str = {
    {"test1"},
    {"hello1","hello2"},
    {"world1","world2","world3"}
};

//循环三次,因为str中有三个元素
//只不过这三个元素每一个都是个一维数组
for(int i=0;i<str.length;i++){

    //第一个一维数组中有1个元素,元素是String类型的
    //第二个一维数组中有2个元素,元素是String类型的
    //第三个一维数组中有3个元素,元素是String类型的
    //所以内层循环每次打印的次数是不一样的
    for(int j=0;j<str[i].length;j++){
        System.out.println(str[i][j]);
    }
}

```

12.5 案例

使用二维数组，打印输出杨辉三角，效果如下：

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

代码实现：

```
//参数line，表示需要输出的行数
public void test(int line){

    int[][] arr = new int[line][];

    //根据规律，构造出二维数组并且赋值
    for(int i=0;i<arr.length;i++){

        arr[i] = new int[i+1];
        //循环给二维数组中的每个位置赋值
        for(int j=0;j<arr[i].length;j++){
            if(j==0 || j==arr[i].length-1){
                arr[i][j] = 1;
            }
            else{
                //除了下标中的0和最后一个，其他的元素都具备相同的规律
                //这个位置的值=上一层和它相同下标的值+前一个元素的值
                arr[i][j] = arr[i-1][j] + arr[i-1][j-1];
            }
        }

    }

    //把赋值完成的二维数组按要求进行输出
    for(int i=0;i<arr.length;i++){
        //控制每行开始输出的空格
        for(int j=0;j<(arr.length-i-1);j++){
            System.out.print(" ");
        }
        //控制输出二维数组中的值，记得值和值之间有空格隔开
        for(int k=0;k<arr[i].length;k++){
            System.out.print(arr[i][k]+" ");
        }
        //当前行输出完，再单独输出一个换行
        System.out.println();
    }

}
```

13 可变参数

> JDK1.5或者以上版本中，可以使用可变参数

例如，假设有这样一个方法，参数 int[] 类型

```
public void test(int[] a){
    //...
}

./在调用这个方法的时候，【只能】传一个数组对象作为参数
int[] arr = {1,2,3};
t.test(arr);
```

例如，如果使用可变参数的语法，上面的方法就可以变成下面的情况：

```
 public void test(int... a){

 }

 //在调用这种方法的时候，我们所传的参数情况就可以有多种选择了
 int[] arr = {1,2,3};

 t.test();
 t.test(1);
 t.test(1,2,3,4);
 t.test(arr);
```

在test方法中，这个可变参数a，其实就是一个int类型的数组，在方法中可以直接把a当做数组来使用。

如果没有传任何参数，那么这个数组a的长度就是0

如果传1个参数，那么数组a的长度是1，数组里面的数据就是所传的参数。

依次类推，传多个参数的情况也是类似的。

额外的，还可以把一个数组当做参数传进来，因为这个可变参数a，本来就是一个数组。

例如，方法中有一个可变参数同时，还可以有其他普通的参数

```
public void test(String str,int... a){

}
```

> 注意，可变参数和普通参数共存的时候，可边参数必须放到最后一个参数的位置