

第3章 数 组

知识要点：

1. 数组概述
2. 一维数组
3. 多维数组
4. 不规则数组

学习目标：

通过本章的学习,要求读者掌握一维数组和 multidimensional 数组的定义、初始化与使用,数组的内存分配方式,不规则数组的使用。

3.1 数组概述

数组是一组具有相同类型或者类型兼容数据的集合,方便数据的管理和使用,是一种引用数据类型。

Java 语言中数组的使用非常灵活。按照所支持数据维度的不同,数组可以分为一维数组和 multidimensional 数组。按照每一维数据的个数是否相同,数组可以分为规则数组和非规则数组(锯齿数组)。

在数组中有 5 个重要概念,分别是：

- (1) 数组的名称(name),又称为数组名,代表一个数组。
- (2) 数组中的元素(element),数组中的每一个数据,数据在数组中有序排列。
- (3) 数组的类型(type),数组中每个数据的类型,通常是相同的类型。
- (4) 数组的索引(index),访问数组中元素的序号,序号的索引位置从 0 开始。
- (5) 数组的长度(length),数组中的元素个数,通过数组名.length 获取。

例如,现实生活中,一个班级有几十位学生,班级就可以作为学生的数组,班级名作为数组的名称,而每位学生作为数组中的一个元素,为了更好地管理学生,通常给每位学生分配一个学号,作为数组的索引,通过学号可以快速地找到对应的学生,班级的总人数作为数组的长度。

3.2 一维数组

3.2.1 一维数组的声明

类似于变量的声明,使用数组时,也要先声明数据类型和数组的名称。一维数组的声明格式如下：

数据类型[] 数组名;



一维数组

或者

```
数据类型 数组名[];
```

例如：声明整型一维数组 a 如下：

```
int[] a;
```

或者

```
int a[];
```

注意：以上两种声明方式没有任何区别。一般推荐使用第一种方式声明数组。从声明中可知数组的名称是 a，数组中元素的类型是 int。

数组中可以存放声明中的数据类型或者比声明数据类型小的类型，但是在使用数组中的元素时，都以声明中的数据类型操作数据。例如，在本例中，int 类型或者比 int 类型小的数据类型数据(byte, short, char)可以放在该数组中，但是在使用时会自动转型成 int 类型后使用。

数组的维度可以简单理解为有几对括号，例如：一维数组在声明的时候有一对括号[]，二维数组在声明的时候有两对括号[][]，以此类推。

声明数组时，不能指定数组的长度(即数组中元素的个数)，下面的写法是错误的。

例如：

```
int[3] a;
```

或者

```
int a[3];
```

都是错误的

3.2.2 一维数组的创建

声明数组之后，数组中能存放多少个数据还不能确定，因为 Java 虚拟机(JVM)还没有为存储数组中的元素分配内存空间，根据数组内存空间分配的不同方式，可以分为静态创建方式和动态创建方式两种。

1. 一维数组静态创建方式

```
数组名={元素 1, 元素 2, ..., 元素 n};
```

也可以在声明数组的同时创建数组。如下所示：

```
数据类型[] 数组名={元素 1, 元素 2, ..., 元素 n};
```

```
数据类型 数组名[]={元素 1, 元素 2, ..., 元素 n};
```

例如：

```
int[] a;
```

```
a=[10, 20, 30];
```

或者

```
int[] a={10,20,30};
```

2. 一维数组动态创建方式

数组名=new 数据类型 [数组的长度];

也可以在声明数组的同时创建数组。如下所示:

数据类型 [] 数组名=new 数据类型 [数组的长度];

数据类型 数组名 []=new 数据类型 [数组的长度];

例如:

```
int[] a;  
a=new int[3];
```

或者

```
int a=new int[3];
```

注意: 关键字 new 是内存分配符,可以动态地为数组开辟存储空间。

分配存储空间大小的公式是: 存储数据的类型 $\times$ 数组的长度。例如,一个 int 类型数据占 4 字节空间,则前面创建的数组 a 占用 $4 \times 3 = 12$ 字节的空间。

3.2.3 一维数组的使用

1. 一维数组的使用

一维数组元素的访问格式如下:

数组名 [索引]

注意: 索引是非负的整型常数或表达式,数组的索引从 0 开始,到数组的长度减 1 结束。如果索引小于 0 或者大于数组的长度减 1 时,则会出现运行时数组索引越界异常 ArrayIndexOutOfBoundsException。该索引也被称为元素在数组中的偏移位置。

【例 3-1】 一维数组的创建和使用。

```
1 package javaoo;  
2 public class Demo3_1 {  
3     public static void main(String[] args) {  
4         int[] a={10,20,30};  
5         System.out.println(a[1]); //20  
6         int[] b=new int[3];  
7         b[0]=8;  
8         System.out.println(b[0]); //8  
9         System.out.println(b[1]); //0  
10        System.out.println(b[5]); //ArrayIndexOutOfBoundsException  
11    }  
12 }
```

运行结果: 见单行代码注释。

代码解释：

第 4 行静态方式创建数组 a,并存储 3 个 int 类型元素,分别是 10,20,30。

第 5 行索引是 1,表示要访问数组中的第 2 个元素,数组 a 中的第 2 个元素是 20。

第 6 行动态方式创建数组时,系统会默认给数组的所有元素按照数据类型赋初值。本例中数据类型是 int 类型,所以该数组中所有的元素初值默认为 0。动态创建数组时元素的默认初值见表 3-1。

表 3-1 动态创建数组时元素的默认初值

数 据 类 型	默 认 值	备 注
byte	0	
short	0	
char	'\u0000'	
int	0	
long	0	实际为 0L
float	0.0	实际为 0.0F
double	0.0	实际为 0.0D
boolean	false	
引用类型	null	

第 7 行为数组 b 中的第 1 个元素重新赋值。

第 10 行无法访问到索引为 5 的元素,出现数组索引越界异常。

2. 一维数组的遍历

一维数组的遍历,就是对一维数组中的所有元素按照相同的规律获取,常采用 for 循环方式,其中用“数组名.length”表示一维数组的长度。

【例 3-2】 一维数组的遍历。

```
1 package javaoo;
2 public class Demo3_2 {
3     public static void main(String[] args) {
4         int[] a={10,20,30};
5         //普通 for 循环语句的写法
6         for(int i=0;i<a.length;i++) {
7             System.out.print(a[i]+" ");
8         }
9         System.out.println();
10        //增强 for 循环语句的写法
11        for(int i:a) {
12            System.out.print(i+" ");
13        }
14    }
15 }
```

运行结果：

```
10 20 30
10 20 30
```

代码解释：

由于数组的索引从 0 开始，循环变量也是从 0 开始，所以可以利用循环变量 `i` 作为索引。

第 6 行中的条件判断，推荐使用“数组名.length”的形式，而不是固定一个数值，因为“数组名.length”的形式会随着数组中元素个数的改变而动态变化，使用起来比较灵活。

第 11 行是增强的 for 循环语句写法，可以简化 for 循环操作。

增强的 for 循环语句格式如下：

```
for(数据类型 变量:集合类型){
    循环体语句;
}
```

说明：集合类型可以是数组，或者是其他集合类型（如 ArrayList 等）。变量前的数据类型要和集合中元素的数据类型一致。例如，例中 `a` 为 `int` 类型的数组，所以第 11 行变量 `i` 前的数据类型也是 `int` 类型。

增强的 for 循环语句会自动从集合中按顺序 1 次取 1 个的方式，把集合中的元素赋值给变量，并且能够自动判断循环的结束条件是否满足。

3.2.4 一维数组的内存分配

1. JVM 的内部体系结构

Java 程序都是在 JVM 中运行，JVM 为了运行一个程序，需要在内存中存储很多数据，如字节码文件、方法的参数、返回值、创建的对象，以及运算的中间结果等。JVM 把这些数据放在“运行时数据区”中统一管理。JVM 的内部体系结构如图 3-1 所示。

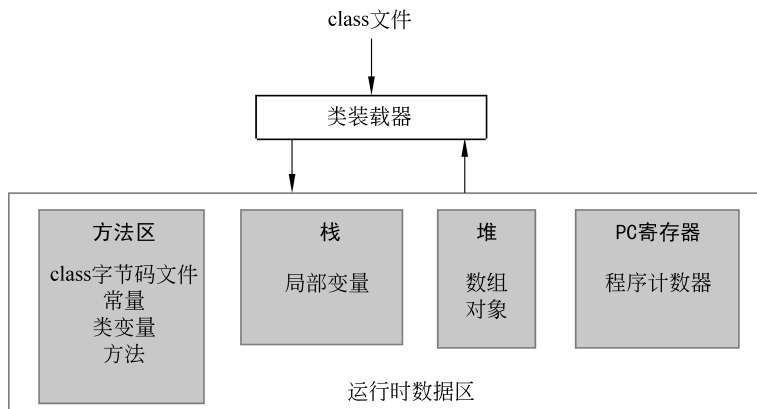


图 3-1 JVM 的内部体系结构

(1) 方法区。

方法区中存储 class 字节码文件、常量、类变量、方法等。当 JVM 装载 class 字节码文件

时,会从该文件中解析类型信息,并把 class 字节码文件以及类型信息存放在方法区中。

(2) 栈。

栈是一种数据结构,其特点是“先进后出”或者“后进先出”。该结构类似于游戏手枪中的弹夹,最后压入弹夹的子弹会第一个发射出去,而第一颗压入的子弹,会最后一个发射出去。本书中的栈特指方法栈,当运行方法时,JVM 会开辟一个方法栈空间,用来存储该方法中局部变量的值。

(3) 堆。

相对栈空间,系统为堆内存开辟的空间更大。在堆中主要存储引用类型数据,如数组或对象等。

(4) PC 寄存器。

PC 寄存器类似于程序计数器,是在多线程启动时创建的。

下面以例 3-1 为例,讲解一维数组的内存空间分配,如图 3-2 所示。

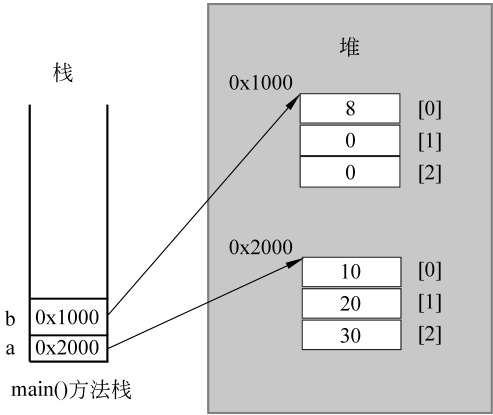


图 3-2 一维数组的内存空间分配示意图

解释说明:

第 3 行中,main()方法是程序运行的入口方法,JVM 为了运行该方法,会开辟一块方法栈空间。方法中定义的变量是局部变量(如,本例中的数组名 a 和 b),局部变量的值都保存在方法栈空间。无论是静态方式,还是动态方式创建的数组,由于数组中数据占用的空间比较大,都会放在堆空间中保存,并且系统会在堆空间中为数组分配一块连续的空间。

第 4 行 `int[] a = {10,20,30}`,通过静态方式创建数组。程序先执行“=”右边的代码,在堆内存中分配一块连续的空间,用来保存 3 个 int 类型数据的值,因为每个 int 类型的数据占 4 字节大小的空间,所以分配的连续空间大小是 12 字节。这块空间在内存中有一个首地址,如图 3-2 中假定首地址为 0x2000(内存地址由随机算法自动生成,通常用十六进制数表示),10、20、30 分别存储在这块空间中的特定位置上,因为 10 是数组中的第 1 个元素,所以它所在空间的内存偏移位置是 0,20 是数组中的第 2 个元素,内存偏移位置是 1,以此类推。“=”为赋值运算符,把首地址赋值给数组名 a,因此数组名中保存的是堆内存中的一个地址,从而建立起栈内存与堆内存之间的联系。

第 5 行通过“数组名[索引]”的方式访问堆内存中的元素。

第 6 行 `int[] b=new int[3]`,通过动态方式创建数组。动态方式创建类似于静态方式

创建,不同点是动态创建数组中的元素,由系统按照数据类型赋默认初值,int 类型数据的默认值是 0。

第 7 行改变 b[0]的值是 8,其他数据的值仍然是默认值 0。

第 10 行访问 b[5],因为在堆空间中没有为这个元素分配空间,所以出现数组索引越界异常。

注意: 掌握程序运行时的内存分配,对程序的理解大有益处。

2. 数组的复制

数组的复制按照分配内存空间的不同,可以分为浅拷贝和深拷贝。

(1) 浅拷贝。

浅拷贝中多个数组名共用同一块堆内存空间。

【例 3-3】 一维数组的浅拷贝。

```
1 package java00;  
2 public class Demo3_3 {  
3     public static void main(String[] args) {  
4         int[] a={10,20,30};  
5         int[] b=a;  
6         a[0]=88;  
7         System.out.println(b[0]); //88  
8     }  
9 }
```

运行结果:

88

代码解释:

第 5 行数组名 a 中保存的堆内存首地址,赋值给了数组名 b,因此 a 和 b 指向的是同一块堆内存空间。

第 6 行修改 a[0]的值为 88,使用数组名 b 访问第 1 个元素时也是 88,如图 3-3 所示。

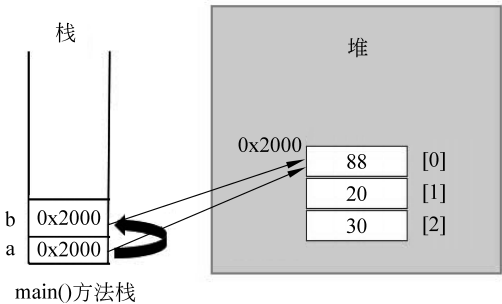


图 3-3 一维数组的浅拷贝示意图

(2) 深拷贝。

深拷贝中每个数组在堆内存中有不同的空间。深拷贝的方式有很多种,这里只讲解

System.arraycopy()方法,其具体定义如下:

```
System.arraycopy (dataType [] srcArray, int srcIndex, dataType [] destArray, int  
destIndex, int length)
```

说明: srcArray 表示原数组,srcIndex 表示原数组中的起始索引位置,destArray 表示目标数组,destIndex 表示目标数组中的起始索引位置,length 表示复制的数组长度。使用此方法复制数组时,length + srcIndex 必须小于或等于 srcArray.length,同时 length + destIndex 必须小于或等于 destArray.length。

【例 3-4】 一维数组的深拷贝。

```
1 package javaoo;  
2 public class Demo3_4 {  
3     public static void main(String[] args) {  
4         int[] a={10,20,30};  
5         int[] b=new int[3];  
6         System.arraycopy(a, 0, b, 0, a.length);  
7         a[0]=88;  
8         System.out.println(b[0]);  
9     }  
10 }
```

运行结果:

10

代码解释:

第 5 行动态方式创建数组 b,数组中的所有元素都是默认值 0。

第 6 行通过 System.arraycopy()方法,把数组 a 中的数据拷贝到数组 b 中对应的位置,即数组 b 中的元素分别是 10,20,30。

第 7 行只是修改数组 a[0]的值,数组 b[0]的值不变,如图 3-4 所示。

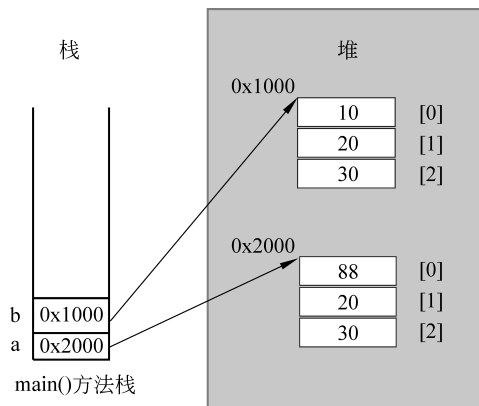


图 3-4 一维数组的深拷贝示意图

课堂练习 1

- 下列一维数组的声明中错误的是()。
A. `int[] a` B. `boolean a[ ]`
C. `double a[2]` D. `float[ ] d`
- 数组定义的代码如 `double a[ ] = new double[2];`; 下列选项中错误的是()。
A. 该一维数组的长度是 3
B. 该数组中的元素默认为 0.0
C. `a[1] = 3.0`
D. `a[0] = 1.0`
- 下列选项中正确的是()。

```
public class ArrayTest {
    public static void main(String[] args) {
        int f1[] =new int[5];
        int f2[]={ 1, 2, 3, 4 };
        f1=f2;
        System.out.println("f1[2] =" +f1[2]);
    }
}
```

- A. fl[2] = 0 B. fl[2] = 3
C. 编译错误 D. 运行无结果
4. 下列程序的运行结果是()。

```
int index = 2;
boolean[] test = new boolean[4];
boolean foo = test[index];
System.out.println("foo = " + foo);
```

- A. foo = 0; B. foo = null;
C. foo = true; D. foo = false;

3.3 多维数组

二维及以上维数组称为多维数组。在通常的编程中,二维数组使用得最频繁,因此以二维数组为例学习多维数组,其他多维数组的操作和二维数组类似。

二维数组和一维数组的使用基本一致,也有声明、创建、初始化和访问几个步骤。二维数组是一种特殊的一维数组,该一维数组的每个元素又是一维数组。因此,二维数组又称为数组的数组。

3.3.1 多维数组的声明

二维数组的声明格式如下:



数据类型 [] [] 数组名;

或者

数据类型 数组名 [] [];

或者

数据类型 [] 数组名 [];

例如: 声明整型二维数组 a 如下:

```
int [ ] [ ] a;
```

或者

```
int a [ ] [ ];
```

或者

```
int [ ] a [ ];
```

注意: 以上 3 种声明方式没有任何区别。推荐使用第 1 种方式声明二维数组, 对第 3 种方式只了解。从声明中可以知道二维数组的名称为 a, 数据类型是 int。

声明二维数组时, 不能指定数组的长度。下面的写法是错误的。

例如:

```
int [2] [3] a;
```

或者

```
int a [2] [3];
```

或者

```
int [2] a [3];
```

3.3.2 多维数组的创建

多维数组的创建分为静态创建方式和动态创建方式两种。

1. 二维数组的静态创建方式

数组名 = { {元素 1}, {元素 2}, ..., {元素 n} };

说明: 元素 1, 元素 2, ..., 元素 n 都是一维数组。

也可以在声明数组的同时创建数组, 如下所示。

数据类型 [] [] 数组名 = { {元素 1}, {元素 2}, ..., {元素 n} };

数据类型 数组名 [] [] = { {元素 1}, {元素 2}, ..., {元素 n} };

数据类型 [] 数组名 [] = { {元素 1}, {元素 2}, ..., {元素 n} };

例如:

```
int [ ] [ ] a;
```

```
a = [[1, 2, 3], [10, 20, 30]];
```

或者

```
int[][] a = [[1, 2, 3], [10, 20, 30]];
```

2. 二维数组的动态创建方式

数组名=new 数据类型[一维长度][二维长度];

也可以在声明数组的同时创建数组,如下所示。

数据类型[][] 数组名=new 数据类型[一维长度][二维长度];

数据类型 数组名[][]=new 数据类型[一维长度][二维长度];

数据类型[] 数组名[]=new 数据类型[一维长度][二维长度];

例如:

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

或者

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

注意: 该二维数组在内存中的存储空间大小等于“数据类型×一维长度×二维长度”。

例如,int 类型数据占 4 字节空间,则数组 a 占 $4 \times 2 \times 3 = 24$ 字节的空间。

3.3.3 多维数组的使用

1. 二维数组的使用

访问二维数组中的元素格式如下:

数组名[索引 1][索引 2];

说明: 索引 1 和索引 2 都是非负的整型常数或表达式,索引值从 0 开始,到对应维度的长度减 1 为止。如果索引小于 0 或者大于对应维度的长度减 1 时,则会出现数组索引越界异常 `ArrayIndexOutOfBoundsException`。

【例 3-5】 二维数组的创建和使用。

```
1 package javaoo;  
2 public class Demo3_5 {  
3     public static void main(String[] args) {  
4         int[][] a = {{1, 2, 3}, {10, 20, 30}};  
5         System.out.println(a[1][2]); //30  
6         int[][] b = new int[2][3];  
7         b[0][0] = 8;  
8         System.out.println(b[0][0]); //8  
9         System.out.println(b[1][2]); //0  
10        System.out.println(b[0][5]); //ArrayIndexOutOfBoundsException  
11    }
```

```
12 }
```

运行结果：见单行代码注释。

代码解释：

第 5 行 `a[1][2]` 表示访问二维数组中第 2 个一维数组中的第 3 个元素。`a[1]` 是 {10, 20, 30}, 是一维数组, 它的第 3 个元素是 30。

第 6 行动态方式创建数组时, 系统会默认给数组的所有元素按照数据类型赋初值。因为本例中数据类型是 `int` 类型, 所以该数组中所有的元素初值默认是 0。

第 7 行给二维数组 `b` 中的第 1 个一维数组中的第 1 个元素赋值 8。

第 10 行 `b[0]` 中没有第 6 个元素, 所以运行时出现异常。

2. 二维数组的遍历

二维数组的遍历是按照相同规律对数组中所有元素进行获取的过程, 常采用 `for` 循环方式。注意以下两个区别：

数组名.`length` 表示二维数组中的元素个数, 即一维数组的个数。

数组名[索引].`length` 表示二维数组中索引值所对应的一维数组的长度。

【例 3-6】 二维数组的遍历。

```
1 package javaoo;
2 public class Demo3_6 {
3     public static void main(String[] args) {
4         int[][] a = {{1, 2, 3}, {10, 20, 30}};
5         //普通 for 循环语句的写法
6         for(int i=0; i<a.length; i++) {
7             for(int j=0; j<a[i].length; j++) {
8                 System.out.print(a[i][j]+" ");
9             }
10            System.out.println();
11        }
12        //增强 for 循环语句的写法
13        for(int[] i:a) {
14            for(int j:i) {
15                System.out.print(j+" ");
16            }
17            System.out.println();
18        }
19    }
20 }
```

运行结果：

```
1 2 3
10 20 30
1 2 3
10 20 30
```

代码解释：

第 6 行 `a.length` 表示二维数组的长度,值是 2,`a[0]`表示{1,2,3},因此 `a[0].length` 的值是 3,`a[1]`表示{10,20,30},`a[1].length` 的值也是 3。`a[0][0]`表示{1,2,3}中的第 1 个元素值 1,类似地,`a[1][2]`表示{10,20,30}中的第 3 个元素值 30。

第 13~18 行为二维数组增强的 `for` 循环写法,因为二维数组中的每个元素都是一维数组,所以第 13 行冒号前用 `int[] i`,又因为 `i` 为一维数组,它的每个元素都是 1 个 `int` 值,所以第 14 行冒号前用 `int j`。

注意：对动态方式创建数组的遍历方式同本例。

3.3.4 多维数组的内存分配

二维数组在内存分配时,也涉及栈内存和堆内存,但是内存分配的情况较一维数组更复杂。下面以例 3-5 为例,讲解二维数组的内存空间分配,如图 3-5 所示。

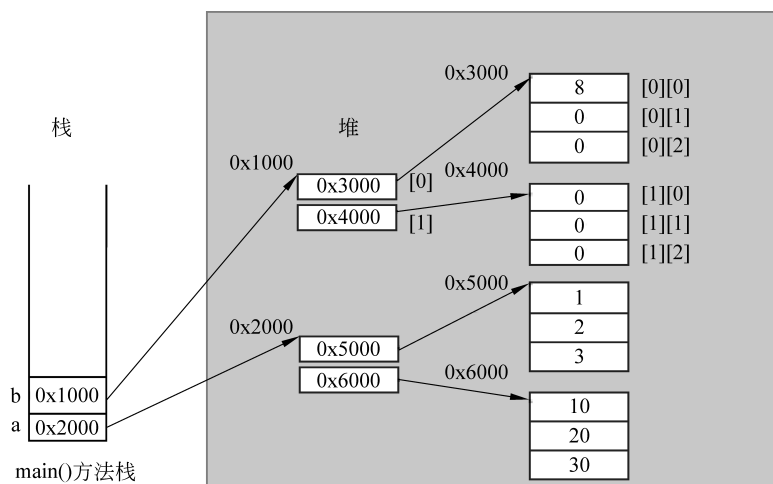


图 3-5 二维数组的内存空间分配示意图

解释说明：

系统在堆内存分配时,为二维数组中不同维度的数据开辟多块内存空间。因为二维数组中每个元素都是一维数组,所以分配的第一块空间用来保存一维数组的名字,而一维数组的名字对应的是内存中的 1 个地址,第二块空间用来保存数组中的数据。

内存地址用十六进制的整数表示,格式是“数组类型@内存地址”,图 3-5 中给出的是简略写法。

输出 `a`,得到如“`[[I@2f92e0f4`”的结果(实际情况下,会有不同)。“`[[`”表示 `a` 是二维数组,“`I`”表示数据类型是整型,“`@`”后的数据是十六进制数,“`f`”表示十六进制中的 15,“`a`”表示 10,“`b`”表示 11,以此类推。

输出 `a[0]`,得到如“`[I@28a418fc`”的结果。“`[`”表示 `a[0]`是一维数组,“`I`”表示数据类型是整型。

课堂练习 2

1. 下列二维数组的创建,错误的是()。
A. `double[][] a = new double[][];`
B. `int a[][] = new int[3][];`
C. `int a[][] = { { 1, 2 }, { 3, 4 } };`
D. `int[][] a = new int[3][4];`
2. 下面能引起编译器错误的语句是()。
A. `int[] a = { 1, 2, 3, 4 };`
B. `int a[][] = { 1, 2 }, { 3, 4 };`
C. `int a[] = new int[4];`
D. `String a[] = { "1", "2", "3" };`
3. 下列数组初始化形式正确的是()。
A. `int t1[][] = { { 1, 2 }, { 3, 4 }, { 5, 6 } };`
B. `int t2[][] = { 1, 2, 3, 4, 5, 6 };`
C. `int t3[3][2] = { 1, 2, 3, 4, 5, 6 };`
D. `int t4[][]; t4 = { 1, 2, 3, 4, 5, 6 };`



不规则数组

3.4 不规则数组

在定义二维数组的时候,组成二维数组的各个一维数组的长度相同,该数组称为规则数组。如果组成二维数组的一维数组的长度不同,则称之为不规则数组(或锯齿数组),其他多维数组中出现的不规则数组与二维数组的定义类似。

【例 3-7】 不规则二维数组的使用。

```
1 package javaoo;
2 public class Demo3_7 {
3     public static void main(String[] args) {
4         int[][] a={1},{2,3},{10,20,30}};
5         int[][] b=new int[3][];
6         b[0]=new int[1];
7         b[1]=new int[2];
8         b[2]=new int[3];
9         a[0][0]=66;
10        b[2][1]=88;
11        for(int i=0;i<a.length;i++) {
12            for(int j=0;j<a[i].length;j++) {
13                System.out.print(a[i][j]+" ");
14            }
15            System.out.println();
16        }
```

```

17         for(int[] i:b) {
18             for(int j:i) {
19                 System.out.print(j+" ");
20             }
21             System.out.println();
22         }
23     }
24 }

```

运行结果：

```

66
2 3
10 20 30
0
0 0
0 88 0

```

代码解释：

第4行静态方式创建不规则数组 `int[][] a= {{1},{2,3},{10,20,30}}`；

二维数组 `a` 有3个元素，分别是一维数组 `a[0]` 是{1}，`a[1]` 是{2,3}，`a[2]` 是{10,20,30}，3个一维数组的长度不相同。

第5行动态方式创建不规则数组，表示该二维数组中可以包含3个一维数组，并且只为存储这3个一维数组名分配了空间，默认空间里保存的值都是 `null`。

第6~8行在堆内存中二次分配内存空间，分别为3个一维数组分配了4字节、8字节及12字节的空间。由于是动态方式创建，因此会按照数据类型 `int` 赋初值0。

第9~10行修改二维数组中元素的值。

第11~16行使用普通 `for` 循环输出二维数组 `a` 的所有元素。

第17~22行使用增强 `for` 循环输出二维数组 `b` 的所有元素。

注意：静态方式创建数组会一次性为所有元素分配内存空间，而动态方式创建数组时，会多次分配内存空间。

例3-7 不规则二维数组的内存空间分配如图3-6所示。

【例3-8】 编程实现杨辉三角形，输出效果如下所示。

```

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

```

提示：假设 `i` 表示行，`j` 表示列，该三角形腰上的数都是1，其他位置的数值等于其上一行相邻两个数之和，公式为：`a[i][j]=a[i-1][j-1]+a[i-1][j]`。

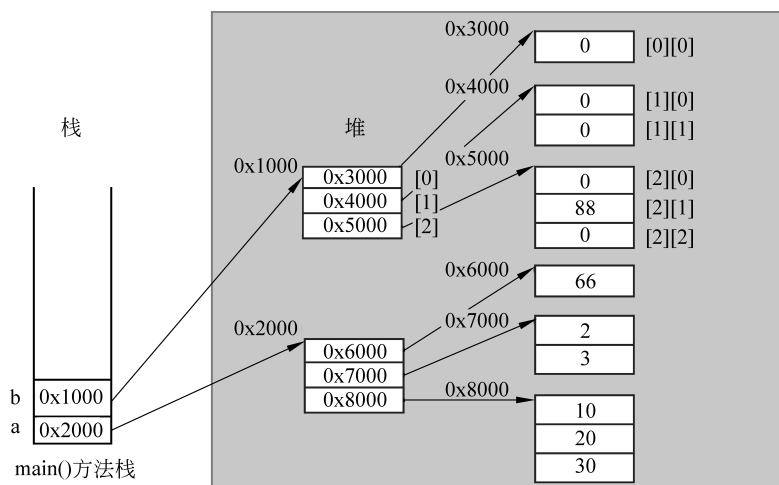


图 3-6 不规则二维数组的内存空间分配示意图

```

1 package java00;
2 public class Demo3_8 {
3     public static void main(String[] args) {
4         int a[][] = new int[5][];
5         for (int i = 0; i < a.length; i++) {
6             a[i] = new int[i + 1];
7             for (int j = 0; j < a[i].length; j++) {
8                 if (j == 0 || i == j) {
9                     a[i][j] = 1;
10                } else {
11                    a[i][j] = a[i - 1][j - 1] + a[i - 1][j];
12                }
13                System.out.print(a[i][j] + " ");
14            }
15            System.out.println();
16        }
17    }
18 }

```

代码解释：

第 4 行 `int a[][] = new int[5][];` 使用动态方式创建由 5 个一维数组组成的二维数组。用每个一维数组保存三角形的一行数据，但是每个一维数据的组成情况目前未知。

第 6 行 `a[i] = new int[i + 1];` 通过循环变量 `i` 创建不同长度的一维数组，分别是 `a[0] = new int[1]`，`a[1] = new int[2]`， $\dots$ ，`a[4] = new int[5]`，这时该二维数组是不规则数组。不规则二维数组的使用是为了更节省内存空间的消耗。

假设第 4 行使用 `int a[][] = new int[5][5]` 创建规则二维数组时，保存所有数据需要 $4 \times 5 \times 5 = 100$ 字节空间，而不规则数组只需要 $4 \times 1 + 4 \times 2 + 4 \times 3 + 4 \times 4 + 4 \times 5 = 60$ 字节空间，节省了 40% 的内存空间。

第 8 行判断数组中的元素是否在三角形的腰上,条件为 $j == 0$ 或 $i == j$,其他元素的赋值都满足公式 $a[i][j] = a[i-1][j-1] + a[i-1][j]$ 。

本章小结

本章主要讲述了数组的基本概念;一维数组和二维数组的声明、创建、使用,其中需要重点掌握数组静态创建方式和动态创建方式的区别;结合内存分配图,方便读者更好地了解数组内部的运行原理,因为数组是一种引用类型,所以对后续章节中对象的内存分配理解有指导作用。本章最后讲解了不规则数组的声明、创建、使用。不规则数组满足“按需分配”的要求,能更好地节省内存空间。

习 题 3

一、单选题

1. 下列程序的运行结果是()。

```
public class Test {
    public static void main(String args[]) {
        String foo = "blue";
        boolean[] bar = new boolean[1];
        if (bar[0]) {
            foo = "green";
        }
        System.out.println("foo = " + foo);
    }
}
```

A. foo = B. foo = blue C. foo = null D. foo = green

2. 下列程序的运行结果是()。

```
public class Test {
    public static void main(String args[]) {
        int index = 1;
        int[] foo = new int[3];
        int bar = foo[index];
        int baz = bar + index;
        System.out.println(" baz = " + baz);
    }
}
```

A. baz = 0 B. baz = 1 C. baz = 2 D. 编译错误

3. 下列程序的运行结果是()。

```
int index;
char[] array = {'x', 'y', 'z'};
```

```
System.out.println(array[index]);
```

A. x

B. y

C. z

D. 编译错误

4. 下列二维数组的声明中,错误的是()。

A. `int a[][];`

B. `int[][] a;`

C. `int[] a[];`

D. `int a[5][6];`

5. 下列二维数组的创建中,错误的是()。

A. `int a[][]=new int[3][];`

B. `int[][] a=new int[3][4];`

C. `int a[][]=new int[1][4];`

D. `int [][] a=new int[][];`

二、简答题

1. 简述数组中的 5 个重要概念。
2. 简述创建一维数组的两种方式。
3. 简述栈内存和堆内存的区别。
4. 如何遍历二维数组?
5. 简述不规则数组和规则数组的区别。

三、编程题

1. 正序和倒序输出如下数组。

```
a[] = {1, 2, 3, 4, 5}
```

2. 找出如下数组中的最大元素和最小元素。

```
a[][] = { {3, 2, 6}, {6, 8, 2, 10}, {5}, {12, 3, 23} }
```

3. 编写主类,在主方法中定义大小为 10×10 的二维字符型数组,数组名为 y,正反对角线上存的是 '*',其余位置存的是 '#';请输出这个数组中的所有元素。

4. 编写类,在主方法中定义大小为 50 的一维整型数组,数组中存放 {1, 3, 5, ..., 99}, 输出这个数组中的所有元素,每输出 10 个数换行,相邻元素之间用空格隔开。