第 5 章　数组与字符串

本章主要内容：

- 一维数组和多维数组的定义；
- 数组元素的访问；
- 字符串及应用。

在程序设计中，数组和字符串是常用的数据结构。无论是在面向过程的程序设计中，还是面向对象的程序设计中，数组和字符串都起着重要的作用。

5.1　数组的基本概念

所谓数组就是若干个相同数据类型的元素按一定顺序排列的集合。在 Java 语言中数组元素可以由基本数据类型的量组成，也可以由对象组成。数组中的所有元素都具有相同的数据类型，用一个统一的数组名和一个下标来唯一地确定数组中的元素。从数组的构成形式上来分，数组可以分为一维数组和多维数组。

为了充分地理解数组的概念，首先介绍 Java 语言有关内存分配的知识。Java 语言把内存分为两种：栈内存和堆内存。

在方法中定义的一些基本类型的变量和对象的引用变量都在方法的栈内存中分配，当在一段代码块中定义一个变量时，Java 就在栈内存中为这个变量分配内存空间，当超出变量的作用域后，Java 会自动释放掉为该变量所分配的内存空间。

堆内存用来存放由 new 运算符创建的数组或对象，在堆中分配的内存，由 Java 虚拟机的垃圾回收器来自动管理。在堆中创建了一个数组或对象后，同时还在栈中定义一个特殊的变量，让栈中的这个变量的取值等于数组或对象在堆内存中的首地址，栈中的这个变量就成了数组或对象的引用变量，引用变量实际上保存的是数组或对象在堆内存中的首地址（也称为对象的句柄），以后就可以在程序中使用栈的引用变量来访问堆中的数组或对象。引用变量就相当于是为数组或对象起的一个名称。引用变量是普通的变量，定义时在栈中分配，引用变量在程序运行到其作用域之外后被释放。而数组或对象本身在堆内存中分配，即使程序运行到使用 new 运算符创建数组或对象的语句所在的代码块之外，数组或对象本身所占据的内存也不会被释放，数组或对象在没有引用变量指向它时，会变为垃圾，不能再被使用，但仍然占据内存空间不放，在随后一个不确定的时间被垃圾回收器收走（释放掉），这也是 Java 比较占内存的原因。

Java 有一个特殊的引用型常量 null，如果将一个引用变量赋值为 null，则表示该引用变

量不指向(引用)任何对象。

有了栈内存与堆内存的知识后,对下面要介绍的数组和后续章节中将要介绍的对象会有更深的了解。

数组主要有如下几个特点。

- 数组是相同数据类型元素的集合。
- 数组中的各元素是有先后顺序的,它们在内存中按照这个先后顺序连续存放在一起。
- 数组元素用整个数组的名字和它自己在数组中的顺序位置来表示。例如,a[0]表示名字为 a 的数组中的第一个元素,a[1]代表数组 a 的第二个元素,依次类推。

5.2 一维数组

一维数组是最简单的数组,其逻辑结构是线性表。要使用一维数组,需要经过定义、初始化和应用等过程。

5.2.1 一维数组的定义

要使用 Java 语言的数组,一般需经过三个步骤:一是声明数组;二是分配空间;三是创建数组元素并赋值。前两个步骤的语法如下:

```
数据类型[ ] 数组名;                    //声明一维数组
数组名 = new 数据类型[个数];            //分配内存给数组
```

在数组的声明格式里,“数据类型”是声明数组元素的数据类型,可以是 Java 语言中任意的数据类型,包括基本类型和引用类型。“数组名”是用来统一这些相同数据类型的名称,其命名规则和变量的命名规则相同。其中“[]”指明该变量是一个数组类型变量,Java 语言是将“[]”放到数组名的前面,但也可以像 C/C++语言的定义方式将“[]”放在数组名的后面来定义数组,如“数据类型 数组名[ ];”。与 C/C++语言不同,Java 语言在数组的定义中并不为数组元素分配内存,因此“[]”中不用给出数组中元素的个数(即数组的长度),但必须在为它分配内存空间后才可使用。

数组声明之后,接下来便是要分配数组所需的内存,这时必须用运算符 new,其中“个数”是告诉编译器,所声明的数组要存放多少个元素,所以 new 运算符是通知编译器根据括号里的个数,在内存中分配一块空间供该数组使用。利用 new 运算符为数组元素分配内存空间的方式称为动态内存分配方式。

下面举例来说明数组的定义。例如:

```
int[ ] x;          //声明名称为 x 的 int 型数组
x = new int[10];   //x 数组中包含有 10 个元素,并为这 10 元素分配内存空间
```

在声明数组时,也可以将两个语句合并成一行,格式如下:

```
数据类型[ ] 数组名 = new 数据类型[个数];
```

利用这种格式在声明数组的同时,也分配一块内存供数组使用。如上面的例子可以写成如下形式:

```
int[ ] x = new int[10];
```

等号左边的 int[] x 相当于定义了一个特殊的变量 x，x 的数据类型是一个对 int 型数组对象的引用，x 就是一个数组的引用变量，其引用的数组元素个数不定。等号右边的 new int[10]就是在堆内存中创建一个具有 10 个 int 型变量的数组对象。“int[] x = new int[10];”就是将右边的数组对象赋值给左边的数组引用变量。若利用两行的格式来声明数组，其意义也是相同的。例如：

```
int[ ] x;          //定义了一个数组 x
```

这条语句执行完成后的内存状态如图 5.1 所示。

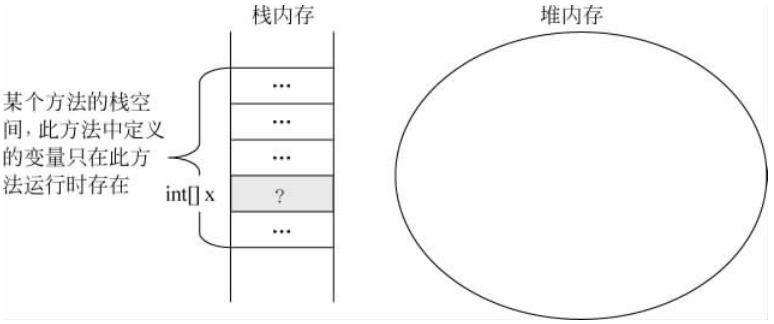

图 5.1　只声明了数组，而没有对其分配内存空间

```
x = new int[10];   //数组初始化
```

这条语句执行完后的内存状态如图 5.2 所示。

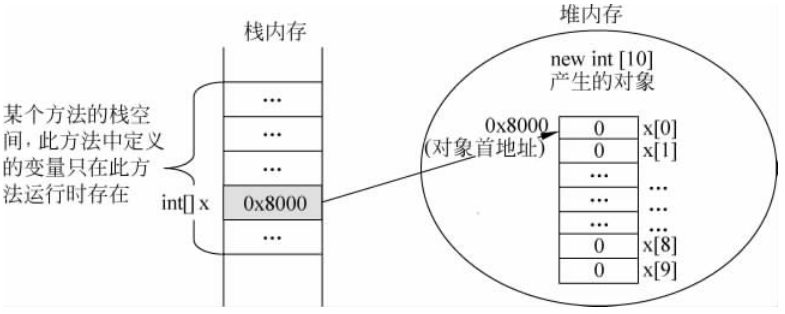

图 5.2　声明数组并分配相应的内存空间，引用变量指向数组对象

执行第 2 条语句“x=new int [10];”后，在堆内存里创建了一个数组对象，为这个数组对象分配了 10 个整数单元，并将数组对象赋给了数组引用变量 x。引用变量就相当于 C 语言中的指针变量，而数组对象就是指针变量指向的那个内存块。所以在 Java 内部还是有指针，只是把指针的概念对用户隐藏起来了，而用户所使用的是引用变量。

用户也可以改变 x 的值，让它指向另外一个数组对象，或者不指向任何数组对象。要想让 x 不指向任何数组对象，只需要将常量 null 赋给 x 即可。如“x=null;”这条语句执行完后的内存状态如图 5.3 所示。

执行完“x=null;”语句后，原来通过 new int [10]产生的数组对象不再被任何引用变量

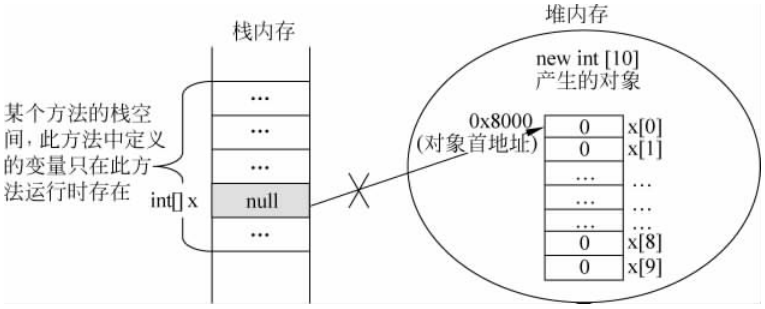

图 5.3 引用变量与引用对象断开

所引用，变成了“孤儿”，也就成了垃圾，直到垃圾回收器来将它释放掉。

说明：数组用 new 运算符分配内存空间的同时，数组的每个元素都会自动赋一个默认值：整数为 0，实数为 0.0，字符为“\0”，boolean 型为 false，引用型为 null。这是因为数组实际是一种引用型的变量，而其每个元素是引用型变量的成员变量。

Java 语言提供的 java.util.Arrays 类用于支持对数组的操作，如表 5.1 所示。

表 5.1 数组类 Arrays 的常用方法

常 用 方 法	功 能 说 明
public static intbinarySearch(X[] a,X key)	X 是任意数据类型。返回 key 在升序数组 a 中首次出现的下标，若 a 中不包含 key，则返回负值
public static void sort(X[] a)	X 是任意数据类型。对数组 a 升序排序后仍存放在 a 中
public static void sort(X[] a,int fromIndex,int toIndex)	对任意类型的数组 a 中从 fromIndex 到 toIndex－1 的元素进行升序排序，其结果仍存放在 a 数组中
public static X[] copyOf(X[] original,int newLength)	截取任意类型数组 original 中长度为 newLength 的数组元素复制给调用数组
public static boolean equals(X[] a,X[] a2)	判断同类型的两个数组 a 和 a2 中对应元素值是否相等。若相等则返 true，否则返回 false

5.2.2 一维数组元素的访问

当定义了一个数组，并用运算符 new 为它分配了内存空间以后，就可以引用数组中的每个元素了。要想使用数组里的元素，可以利用数组名和下标来实现。数组元素的引用方式为：

```
数组名[下标]
```

其中，“下标”可以是整型数或整型表达式，如 a[3＋i](i 为整数)。Java 语言数组的下标是从 0 开始的。例如：

```
int[ ] x = new int[10];
```

其中，x[0]代表数组中第 1 个元素，x[1]代表第 2 个元素，x[9]为第 10 个元素，也就是最后一个元素。另外，与 C/C++不同，Java 语言对数组元素要进行越界检查以保证安全性。同

时,对于每个数组都有一个属性 length 指明它的长度,如 x.length 指出数组 x 所包含的元素个数。

【例 5.1】 声明一个一维数组,其长度为 5,利用循环对数组元素进行赋值,然后再利用另一个循环逆序输出数组元素的内容。程序代码如下:

```
1   //filename: App5_1.java          一维数组
2   public class App5_1
3   {
4     public static void main(String[] args)
5     {
6       int i;
7       int[] a;                          //声明一个数组 a
8       a = new int[5];                   //分配内存空间供整型数组 a 使用,其元素个数为 5
9       for(i = 0;i < 5;i++)              //对数组元素进行赋值
10        a[i] = i;
11      for(i = a.length - 1;i >= 0;i -- )     //逆序输出数组的内容
12        System.out.print("a[" + i + "] = " + a[i] + ",\t");
13      System.out.println("\n 数组 a 的长度是: " + a.length);      //输出数组的长度
14    }
15  }
```

该程序的运行结果如下:

```
a[4] = 4,  a[3] = 3,  a[2] = 2,  a[1] = 1,  a[0] = 0
数组 a 的长度是: 5
```

该程序的第 7 行声明了一个整型数组 a,第 8 行为其分配包含 5 个元素的空间;第 9、10 行是利用 for 循环为数组元素赋值;第 11、12 行是利用 for 循环将数组 a 的各元素反序输出;第 13 行是利用数组的长度属性 length 输出元素个数。

5.2.3 一维数组的初始化及应用

对数组元素的赋值,既可以使用单独方式进行(如例 5.1),也可以在定义数组的同时就为数组元素分配空间并赋值,这种赋值方法称为数组的初始化。其格式如下:

```
数据类型[ ] 数组名 = {初值 0,初值 1,…,初值 n};
```

在花括号内的初值会依次赋值给数组的第 1,2,…,n+1 个元素。此外,在声明数组的时候,并不需要将数组元素的个数给出,编译器会根据所给的初值个数来设置数组的长度。如:

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

在上面的语句中,声明了一个整型数组 a,虽然没有特别指明数组的长度,但是由于花括号里的初值有 5 个,编译器会分别依次指定各元素存放,a[0]为 1,a[1]为 2,…,a[4]为 5。

注意:在 Java 程序中声明数组时,无论用何种方式定义数组,都不能指定其长度。如以“int[5] a;”方式定义数组将是非法的,该语句在编译时将出错。

【例 5.2】 设数组中有 n 个互不相同的数,不用排序求出其中的最大值和次最大值。

```
1   //filename: App5_2.java          比较数组元素值的大小
2   public class App5_2
3   {
4     public static void main(String[] args)
5     {
6       int i,max,sec;
7       int[] a = {8,50,20,7,81,55,76,93};       //声明数组 a,并赋初值
8       if(a[0]> a[1])
9       {
10        max = a[0];                            // max 存放最大值
11        sec = a[1];                            // sec 存放次最大值
12      }
13      else
14      {
15        max = a[1];
16        sec = a[0];
17      }
18      System.out.print("数组的各元素为: " + a[0] + "  " + a[1]);
19      for(i = 2;i < a.length;i++)
20      {
21        System.out.print("  " + a[i]);         //输出数组 a 中的各元素
22        if(a[i]> max)                          //判断最大值
23        {
24          sec = max;                           //原最大值降为次最大值
25          max = a[i];                          //a[i]为新的最大值
26        }
27        else if(a[i]> sec)          //即 a[i]不是新的最大值,但若 a[i]大于次最大值
28          sec = a[i];                          //a[i]为新的次最大值
29      }
30      System.out.print("\n 其中的最大值是: " + max);      //输出最大值
31      System.out.println("      次最大值是: " + sec);     //输出次最大值
32    }
33  }
```

该程序运行结果为：

```
数组的各元素为: 8  50  20  7  81  55  76  93
其中的最大值是: 93     次最大值是: 81
```

该程序的第 7 行定义并初始化了数组 a，第 8～17 行利用 if-else 语句将数组前两个元素中大的数保存在变量 max 中，将小的数存放在变量 sec 中；第 19～29 行利用 for 循环与 if 语句的结合，从数组的第三个元素开始到最后，对数组中的元素进行输出并检测，若检测到新的最大数，则将其保存到变量 max 中，次最大数保存到变量 sec 中；第 30、31 行将数组中的最大数和次最大数输出。

【例 5.3】 设有 N 个人围坐一圈并按顺时针方向从 1 到 N 编号，从第 S 个人开始进行 1 到 M 报数，报数到第 M 的人，此人出圈，再从他的下一个人重新开始从 1 到 M 报数，如此进行下去，每次报数到 M 的人就出圈，直到所有人都出圈为止。给出这 N 个人的出圈顺序。

```
1   //filename: App5_3.java                    "约瑟夫环"问题
2   public class App5_3
3   {
4     public static void main(String[ ] args)
5     {
6       final int N = 13,S = 3,M = 5;          //设有 13 个人,从第 3 个人开始 1 至 5 报数
7       int i = S - 1,j,k = N,g = 1;
8       int[ ] a = new int[N];
9       for(int h = 1;h <= N;h++)
10        a[h - 1] = h;                        //将第 h 人的编号存入下标为 h - 1 的数组元素中
11      System.out.println("\n 出圈的顺序为: ");
12      do
13      {
14        i = i + (M - 1);                     //计算出圈人的下标 i
15        while(i >= k)                        //当数组下标 i 大于等于圈中的人数 k 时
16          i = i - k;                         //将数组的下标 i 减去圈中的人数 k
17        System.out.print("    " + a[i]);     //输出出圈人的编号
18        for(j = i;j < k - 1;j++)
19          a[j] = a[j + 1];                   // a[i]出圈后,将后续人的编号前移
20        k -- ;                               //圈中的人数 k 减 1
21        g++;                                 //g 为循环控制变量
22      }while(g <= N);                        //共有 N 人,所以循环 N 次
23    }
24  }
```

程序运行后的输出结果如下:

```
出圈顺序为:
7  12  4  10  3  11  6  2  1  5  9  13  8
```

此题是著名的“约瑟夫环”问题。在第 9、10 行将每个人的编号 h 存入数组元素 a[h－1]中。因为要求从第 3 个人开始进行 1 到 5 报数,而代表第 3 个人的是 a[2],所以在第 7 行将控制数组下标的变量 i 赋值为 S－1 即 i＝2,表示从 i＝2 的下标开始报数。第 12～22 行的 do 循环共执行 N 次,其中的第 14 行用于计算出圈人的下标,因为 i＝i＋(M－1)就是下一个要出圈人的下标。由于变量 k 表示此时圈中剩余的人数,所当 i＞＝k 时,即是 i 已经超出剩下的人数,所以第 16 行是要重新计算数组的下标。第 17 行是输出出圈人的编号。第 18、19 行的循环是当下标为 i 的人出圈后,把后续人的编号前移。由于有一个人出圈,圈中的人数少 1,所以第 20 行将用于表示圈中剩余人数的变量 k 减 1。第 21 行的变量 g 是用于控制 do 循环次数的变量,所以每次加 1,每次循环找出一个出圈的人,所以循环共执行 N 次。

下面再给出该题的另一种算法。

```
1   //filename: App5_3.java
2   public class App5_3                  //例题 5.3 的另一种解法
3   {
4     public static void main(String[ ] args)
5     {
6       final int N = 13, S = 3, M = 5;   //N 为总人数,从第 S 个人开始报数,报数到 M 的为出圈
```

```
7       int[] p = new int[N];           //数组 p 用于标识已出圈的人
8       int[] q = new int[N];           //数组 q 存放出队顺序
9       int i,j,k,n = 0;
10      k = S - 2;                      //k 从 1 开始数出圈人的下标
11      for(i = 1;i <= N;i++)
12      {
13        for(j = 1; j <= M; j++)       //从 1 到 M 报数,计算出圈人的下标 k
14        {
15          if(k == N - 1)              //当出圈人的下标达到末尾时
16            k = 0;                    //出圈人的下标从 0 开始
17          else
18            k++;                      //否则下标加 1
19          if(p[k] == 1)               //若 p[k] = 1,说明下标为 k 的人已出圈
20            j -- ;          //由于让过已出圈的人,所以 j 要减 1,以保证每次数过 M 个人
21        }
22        p[k] = 1;                     //将下标为 k 的数组元素置 1,表示其出圈
23        q[n++] = k + 1;               //将下标为 k 的人的编号 k + 1 存入数组元素 q[n]中
24      }    //将上行改为 System.out.print((k + 1) + "   ");后可去掉下面三行输出语句
25      System.out.println("出圈顺序为: ");
26      for(i = 0; i < N; i++)
27        System.out.print(q[i] + "    ");
28    }
29  }
```

该种解法的输出结果与前一解法的输出结果完全相同。各行代码的功能请读者自行分析解释。

5.3 foreach 语句与数组

在第 4 章中我们介绍过 for 循环,自 JDK 5 开始引进了一种新的 for 循环,它不用下标就可遍历整个数组,这种新的循环称为 foreach 语句。foreach 语句只需提供三个数据:元素类型、循环变量的名字(用于存储连续的元素)和用于从中检索元素的数组。foreach 语句的语法如下:

```
for(type element : array)
{
  System.out.println(element);
  ⋮
}
```

其功能是每次从数组 array 中取出一个元素,自动赋给变量 element,用户不用判断是否超出了数组的长度。需要注意的是 element 的类型必须与数组 array 中元素的类型相同。例如:

```
int[] arr = {1,2,3,4,5};
for(int element : arr)
  System.out.println(element);        //输出数组 arr 中的各元素
```

5.4 多维数组

虽然一维数组可以处理一般简单的数据，但是在实际的应用中仍显不足，所以 Java 语言提供了多维数组，但在 Java 语言中并没有真正的多维数组。所谓多维数组，就是数组元素也是数组的数组。

5.4.1 二维数组

二维数组的声明方式与一维数组类似，内存的分配也一样是用 new 运算符。其声明与分配内存的格式如下所示：

```
数据类型[][] 数组名;
数组名 = new 数据类型[行数][列数];
```

二维数组在分配内存时，要告诉编译器二维数组行与列的个数。因此在上面格式中，“行数”是告诉编译器所声明的数组有多少行，“列数”则是声明每行中有多少列。例如：

```
int[][] a;          //声明二维整型数组 a
a = new int[3][4];  //分配一块内存空间,供 3 行 4 列的整型数组 a 使用
```

同样地，也可以用较为简洁的方式来声明数组，其格式如下：

```
数据类型[][] 数组名 = new 数据类型[行数][列数];
```

以该种方式声明的数组，在声明的同时，就分配一块内存空间，供该数组使用。如

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

虽然在应用上很像 C 语言中的多维数组，但还是有区别的，在 C 语言中定义一个二维数组，必须是一个 m×n 的矩形，如图 5.4 所示。Java 语言的二维数组不一定是规则的矩形，如图 5.5 所示。

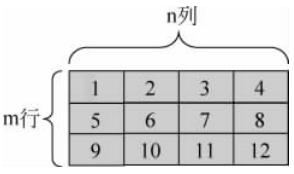

图 5.4 C 语言中二维数组必须是矩形

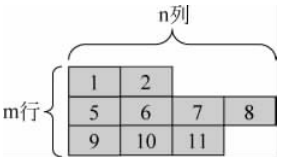

图 5.5 Java 语言中的二维数组不一定是矩形

如，定义一个如下的数组：

```
int[][] x;
```

它表示定义了一个数组引用变量 x，第一个元素为 x[0]，第 n 个元素变量为 x[n－1]。x 中从 x[0]到 x[n－1]的每个元素变量正好又是一个整数类型的数组引用变量。需要注意的是，这里只是要求每个元素都是一个数组引用变量，并没有要求它们所引用数组的长度是多少，也就是每个引用数组的长度可以不一样。例如：

```
int[][] x;
```

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

这两行代码表示数组 x 有三个元素，每个元素都是 int[]类型的一维数组。该语句相当于定义了三个数组引用变量，分别是 int[] x[0]，int[] x[1]和 int[] x[2]，完全可以把x[0]、x[1]和 x[2]当成普通变量名来理解。

由于 x[0]、x[1]和 x[2]都是数组引用变量，因此，必须对它们赋值，指向真正的数组对象，才可以引用这些数组中的元素。例如：

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

由此可以看出，x[0]和 x[1]的长度可以是不一样的，数组对象中也可以只有一个元素。程序运行到这之后的内存分配情况如图 5.6 所示。

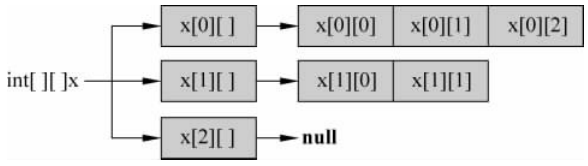

图 5.6 Java 中的二维数组可以看成是多个一维数组

x[0]中的第二个元素用 x[0][1]来表示，如果要将整数 100 赋给 x[0]中的第二个元素，写法如下：

```
x[0][1] = 100;
```

如果数组对象正好是一个 m×n 形式的规则矩阵，可不必向上面代码一样，先创建高维的数组对象后，再逐一创建低维的数组对象。完全可以用一条语句在创建高维数组对象的同时，创建所有的低维数组对象。例如：

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

该语句表示创建了一个 2×3 形式的二维数组，其内存布局如图 5.7 所示。

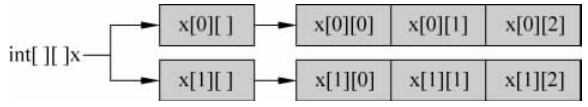

图 5.7 规则的二维数组内存分配

在二维数组中，若要取得二维数组的行数，只要在数组名后加上.length 属性即可；若要取得数组中某行元素的个数，则须在数组名后加上该行的下标，再加上.length。例如：

```
x.length;        //计算数组 x 的行数
x[0].length;     //计算数组 x 的第 1 行元素的个数
x[2].length;     //计算数组 x 的第 3 行元素的个数
```

注意：与一维数组相同，用 new 运算符来为数组申请内存空间时，很容易在数组各维数的指定中出现错误，二维数组要求必须指定高层维数。例如，下面是正确的申请方式：

```
int[][] myArray = new int[10][];     //只指定数组的高层维数
int[][] myArray = new int[10][3];    //指定数组的高层维数和低层维数
```

下面是错误的申请方式：

```
int[][] myArray = new int[ ][5];     //只指定数组的低层维数
int[][] myArray = new int[ ][ ];     //没有指定数组的任何维数
```

如果想直接在声明二维数组时就给数组赋初值，可以利用花括号实现，只要在数组的声明格式后面再加上初值的赋值即可。其格式如下：

```
数据类型[][] 数组名 = { {第 1 行初值},
                        {第 2 行初值},
                        { … },
                        {第 n + 1 行初值} };
```

同样需要注意的是，用户并不需要定义数组的长度，因此在数据类型后面的方括号里并不必填写任何的内容。此外，在花括号内还有几组花括号，每组花括号内的初值会依次赋值给数组的第 1，2，…，n＋1 行元素。例如：

```
int[][] a = { {11,22,33,44},        //二维数组的初始赋值
              {66,77,88,99} };
```

该语句中声明了一个整型数组 a，该数组有 2 行 4 列共 8 个元素，花括号里的两组初值会分别依次指定给各行里的元素存放，a[0][0]为 11，a[0][1]为 22，…，a[1][3]为 99。

注意：与一维数组一样，在声明多维数组并初始化时不能指定其长度，否则出错。如“int[2][3] b＝{{1,2,3},{4,5,6}};”语句在编译时将出错。

【例 5.4】 计算并输出杨辉三角形。

```
1   //filename: App5_4.java          二维数组应用的例子：显示杨辉三角形
2   public class App5_4
3   {
4     public static void main(String[] args)
5     {
6       int i,j;
7       int level = 7;
8       int[][] iaYong = new int[level][]; //声明 7 行二维数组，存放杨辉三角形的各数
9       System.out.println("杨辉三角形");
10      for(i = 0;i < iaYong.length;i++)
11        iaYong[i] = new int[i + 1];       //定义二维数组的第 i 行有 i + 1 列
12      iaYong[0][0] = 1;
13      for(i = 1;i < iaYong.length;i++)   //计算杨辉三角形
14      {
15        iaYong[i][0] = 1;
16        for(j = 1;j < iaYong[i].length - 1;j++)
17          iaYong[i][j] = iaYong[i - 1][j - 1] + iaYong[i - 1][j];
18        iaYong[i][ iaYong[i].length - 1] = 1;
19      }
20      for(int[] row : iaYong)            //利用 foreach 语句显示出杨辉三角形
21      {
```

```
22          for(int col : row)
23            System.out.print(col + "  ");
24          System.out.println();
25        }
26      }
27    }
```

该程序的运行结果为：

```
杨辉三角形
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5  10  10   5   1
1   6  15  20  15   6   1
```

该程序的第 8 行声明了一个 7 行的二维数组 iaYong，每行的列数由第 10、11 行的 for 循环来定义；第 13～19 行的 for 循环用于计算杨辉三角形并存入数组 iaYong 的相应元素中，其中的第 15、18 行分别是将第 i 行的第一个元素和最后一个元素置 1；第 16 行定义的内层 for 循环用于计算第 i 行的其他元素，其计算方法是第 17 行的循环体；第 20～25 行利用 foreach 循环将杨辉三角形输出。

5.4.2 三维以上的多维数组

通过对二维数组的介绍，不难发现，要想提高数组的维数，只要在声明数组的时候将下标与中括号再加一组即可，所以三维数组的声明为“int[][][] a;”，而四维数组为“int[][][][] a;”，依次类推。

使用多维数组时，输入输出的方式和一、二维数组相同，但是每多一维，嵌套循环的层数就必须多一层，所以维数越高的数组其复杂度也就越高。

【例 5.5】 声明三维数组并赋初值，然后输出该数组的各元素，并计算各元素之和。

```
1   //filename: App5_5.java              三维数组应用
2   public class App5_5
3   {
4     public static void main(String[] args)
5     {
6       int i,j,k,sum = 0;
7       int[][][] a = {{{1,2},{3,4}},{{5,6},{7,8}}}; //声明三维数组并赋初值
8       for(i = 0;i < a.length;i++)
9         for(j = 0;j < a[i].length;j++)
10          for(k = 0;k < a[i][j].length;k++)
11          {
12            System.out.println("a[" + i + "][" + j + "][" + k + "] = " + a[i][j][k]);
13            sum += a[i][j][k];                    //计算各元素之和
14          }
15      System.out.println("sum = " + sum);
16    }
```

```
17  }
```

程序运行时的结果如下：

```
a[0][0][0] = 1
a[0][0][1] = 2
a[0][1][0] = 3
a[0][1][1] = 4
a[1][0][0] = 5
a[1][0][1] = 6
a[1][1][0] = 7
a[1][1][1] = 8
sum = 36
```

该程序利用三层循环来输出三维数组的各元素并计算各元素之和。

5.5 字符串

字符串就是一系列字符的序列。在 Java 语言中字符串是用一对双引号(" ")括起来的字符序列，在前几章的例子中已多次用到，如"你好"、"Hello"等。字符串也是编程中经常要使用的数据结构，从某种程度上说字符串有些类似于字符数组。在 Java 语言中无论是字符串常量还是字符串变量，都是用类来实现的。程序中用到的字符串可以分为两大类：一类是创建之后不会再做修改和变动的字符串变量；另一类是创建之后允许再做修改的字符串变量。对于前一种字符串变量，由于程序中经常需要对它做比较、搜索之类的操作，所以通常把它放在一个具有一定名称的对象之中，由程序完成对该对象的上述操作，在 Java 程序中存放这种字符串的变量是 String 类对象；对于后一种字符串变量，由于程序中经常需要对它做添加、插入、修改之类的操作，所以这种字符串变量一般都存放在 StringBuilder 类的对象之中。本书只讨论 String 类型的串变量。

5.5.1 字符串变量的创建

首先再强调一下字符串常量与字符常量的不同，字符常量是用单引号(')括起来的单个字符，而字符串常量是用双引号(")括起来的字符序列。

声明字符串变量的格式与其他变量一样，分为对象的声明与对象创建两步，这两步可以分成两个独立的语句，也可以在一个语句中完成。

格式一：

```
String 变量名;
变量名 = new String("字符串");
```

如：

```
String s;                  //声明字符串型引用变量 s,此时 s 的值为 null
s = new String("Hello"); //在堆内存中分配空间,并将 s 指向该字符串首地址
```

第一个语句只声明了字符串引用变量 s，此时 s 的值为 null；第二个语句则在堆内存中分配了内存空间，并将 s 指向了字符串的首地址。

上述的两个语句也可以合并成一个语句。其格式如下。

格式二：

```
String 变量名 = new String("字符串");
```

如：

```
String s = new String("Hello");
```

还有一种非常特殊而常用的创建 String 对象的方法，这种方法就是直接利用双引号括起来的字符串为新建的 String 对象赋值，即在声明字符串变量时直接初始化。

格式三：

```
String 变量名 = "字符串";
```

如：

```
String s = "Hello";
```

由于字符串是引用型变量，所以其存储方式与数组的存储方式基本相同。

程序中可以用赋值运算符为字符串变量赋值，除此之外，Java 语言定义“+”运算符可用于两个字符串的连接操作(关于字符串的运算符在 3.7.7 中已讲述过)。例如：

```
str = "Hello" + "Java";   //str 的值为"HelloJava"
```

如果字符串与其他类型的变量进行“+”运算，系统自动将其他类型的数据转换为字符串型。例如：

```
int i = 10;
String s = "i = " + i;      //s 的值为"i = 10"
```

前面说过，利用 String 类创建的字符串变量，一旦被初始化或赋值，它的值和所分配的内存内容就不可再改变。如果硬要改变它的值，它会产生一个新的字符串。例如：

```
String str1 = "Java";
str1 = str1 + " Good";
```

这看起来像是一个简单的字符串重新赋值，实际上在程序的解释过程中却不是这样的。程序首先产生 str1 的一个字符串对象并在内存中申请了一段空间，由于发现又需要重新赋值，在原来的空间已经不可能再追加新的内容，系统不得不将这个对象放弃，再重新生成第二个新的对象 str1 并重新申请一个新的内存空间。虽然 str1 指向的内存地址(句柄)是同一个，但对象已经不再是同一个了。

5.5.2 String 类的常用方法

Java 语言为 String 类定义了许多方法。可以通过下述格式调用 Java 语言定义的方法：

```
字符串变量名.方法名();
```

表 5.2 列出了 String 类的常用方法。

表 5.2 String 类的常用方法

常 用 方 法	功 能 说 明
public int length()	返回字符串的长度
public boolean equals(Object anObject)	将给定字符串与当前字符串相比较,若两字符串相等,则返回 true,否则返回 false
public String substring(int beginIndex)	返回字符串中从 beginIndex 开始到字符串末尾的子串
public String substring(int beginIndex,int endIndex)	返回从 beginIndex 开始到 endIndex－1 的子串
public char charAt(int index)	返回 index 指定位置的字符
public int indexOf(String str)	返回 str 在字符串中第一次出现的位置
public int compareTo(String anotherString)	若调用该方法的字符串大于参数字符串,则返回大于 0 的值;若相等则返回数 0;若小于参数字符串,则返回小于 0 的值
public String replace(char oldChar,char newChar)	以 newChar 字符替换字符串中所有 oldChar 字符
public String trim()	去掉字符串的首尾空格
public String toLowerCase()	将字符串中的所有字符都转换为小写字符
public String toUpperCase()	将字符串中的所有字符都转换为大写字符

【例 5.6】 判断回文字符串。

回文是一种“从前往后读”和“从后往前读”都相同的字符串,例如,"rotor"就是一个回文字符串。在本例中使用两种算法来判断回文字符串。

程序中比较两个字符时,使用关系运算符“==”,而比较两个字符串时,需使用 equals() 方法。程序代码如下:

```
1   //filename: App5_6.java              字符串应用: 判断回文字符串
2   public class App5_6
3   {
4     public static void main(String[] args)
5     {
6       String str = "rotor";
7       int i = 0,n;
8       boolean yn = true;
9       if(args.length > 0)
10        str = args[0];
11      System.out.println("str = " + str);
12      n = str.length();
13      char sChar,eChar;
14      while(yn && (i < n/2))   //算法 1
15      {
16        sChar = str.charAt(i);//返回字符串 str 正数第 i + 1 个位置的字符
17        eChar = str.charAt(n - i - 1);        //返回字符串 str 倒数第 i + 1 个位置的字符
18        System.out.println("sChar = " + sChar + "   eChar = " + eChar);
19        if(sChar == eChar)                    //判断两个字符是否相同使用运算符" == "
20          i++;
21        else
22          yn = false;
23      }
```

```
24        System.out.println("算法 1: " + yn);
25        String temp = "",sub1 = "";            //算法 2
26        for(i = 0;i < n;i++)
27        {
28          sub1 = str.substring(i,i + 1);       //将 str 的第 i + 1 个字符截取下来赋给 sub1
29          temp = sub1 + temp;                  //将截下来的字符放在字符串 temp 的首位置
30        }
31        System.out.println("temp = " + temp);
32        System.out.println("算法 2: " + str.equals(temp));  //判断 str 与 temp 是否相等
33     }
34  }
```

该程序运行时可以带命令行参数。若在命令行方式下输入 java App5_6 hello,则程序的运行结果如下:

```
sChar = h     eChar = o
算法 1: false
temp = olleh
算法 2: false
```

该程序的第 9 行用于判断是否带命令行参数,在执行程序时,若带有参数,则第一个参数 args[0]赋值给字符串变量 str,否则,将 str 仍取程序中设定的值"rotor"; 第 14～24 行是算法 1,分别从前向后和从后向前依次获得源串 str 的一个字符 sChar 和 eChar,比较 sChar 和 eChar,如果不相等,则 str 肯定不是回文字符串,所以 yn=false,立即退出循环; 否则,继续比较,直到 str 的所有字符全部比较完,若 yn 值仍为 true,才能肯定 str 是回文字符串; 第 25～32 行是算法 2,将源串 str 反转存入字符串变量 temp 中,再比较两个字符串,如果相等则是回文字符串。

本章小结

1. 数组是由若干个相同类型的变量按一定顺序排列所组成的数据结构,它们用一个共同的名字来表示。数组的元素可以是基本类型或引用类型。数组根据存放元素的复杂程度,分为一维及多维数组。

2. 要使用 Java 语言的数组,必须经过两个步骤: ①声明数组; ②分配内存给数组。

3. 在 Java 语言中要取得数组的长度,即数组元素的个数,可以利用数组的.length 属性来完成。

4. 如果想直接在声明时就给数组赋初值,则只要在数组的声明格式后面加上元素的初值即可。

5. Java 语言允许二维数组中每行的元素个数不相同。

6. 在二维数组中,若要想获得整个数组的行数,或者是某行元素的个数时,也可以利用.length 属性来取得。

7. 字符串可以分为两大类: 一类是创建之后不会再做修改和变动的字符串变量; 另一类是创建之后允许再做修改的字符串变量。

8. 字符串常量与字符常量的不同,字符常量是用单引号(')括起来的单个字符,而字符串常量是用双引号(")括起来的字符序列。

第 5 章习题

5.1 从键盘输入 n 个数,输出这些数中大于其平均值的数。

5.2 从键盘输入 n 个数,求这 n 个数中的最大数与最小数并输出。

5.3 求一个 3 阶方阵的对角线上各元素之和。

5.4 找出 4×5 矩阵中值最小和最大元素,并分别输出其值及所在的行号和列号。

5.5 产生 0～100 的 8 个随机整数,并利用冒泡排序法将其升序排序后输出(冒泡排序算法:每次进行相邻两数的比较,若次序不对,则交换两数的次序)。

5.6 有 15 个红球和 15 个绿球排成一圈,从第 1 个球开始数,当数到第 13 个球时就拿出此球,然后再从下一个球开始数,当再数到第 13 个球时又取出此球,如此循环进行,直到仅剩 15 个球为止,问怎样排才能使每次取出的球都是红球?

5.7 编写 Java 应用程序,比较命令行中给出的两个字符串是否相等,并输出比较的结果。

5.8 从键盘上输入一个字符串和子串开始的位置与长度,截取该字符串的子串并输出。

5.9 从键盘上输入一个字符串和一个字符,从该字符串中删除给定的字符。

5.10 编程统计用户从键盘输入的字符串中所包含的字母、数字和其他字符的个数。

5.11 将用户从键盘输入的每行数据都显示输出,直到输入"exit"字符串,程序运行结束。