第 5 章　数组、字符串和枚举

5.1　数　　组

在 Java 语言中，数组（Array）本身是一种引用数据类型，它是一组具有相同数据类型的数据的有序集合，该集合中的数据称为数组元素。数组元素可以由 8 种基本数据类型或引用类型（对象）组成，数组中的每个元素都具有相同的数据类型。数组具备如下特点。

（1）数组元素的下标从 0 开始。

（2）数组元素占用连续的内存。

（3）数组一旦用 new 分配好内存之后，就不能更改其长度，即不能更改数组所能包含的元素个数。

（4）可用一个统一的数组名和一个下标来唯一地确定其中的某个元素，例如，a[0]表示数组 a 的第一个元素；a[1]表示数组 a 的第二个元素，以此类推。

5.1.1　一维数组

1. 一维数组的声明

声明一维数组需要明确给出数组的名字、数组元素的数据类型，其格式可以是如下任意一种：

```
数组元素类型  数组名字[];
数组元素类型[] 数组名字;
```

例如：

```
int[ ] d;
String[ ] names;
char c[ ];
String s[ ];
```

例如，声明一个数据类型为 Student 类的数组：

```
Student[] s;        //等价于 Student s[]
```

2. 一维数组的实例化

声明数组仅仅是给出了数组名字和元素的数据类型，要想真正使用数组必须为它分配内存空间，即数组实例化。在为数组实例化时必须指明数组的长度，即该数组包含的元素个数。数组实例化使用关键词 new 来完成，格式如下：

```
数组名称=new 数组元素类型[数组长度]
```

例如：

```
int[ ] d;            //声明,不必指定数组的大小
d=new int[4];        //数组实例化,即分配内存
```

上面两条语句也可简化为如下一条语句：

```
int[ ] d=new int[4];
```

实例化时，数组 d 首先在堆内存中开辟 16B 的空间用于存放 int 数据，如图 5-1(b)所示，其中每 4B 构成一个单元，共 4 个单元，分别存储 d[0]、d[1]、d[2]、d[3]，这 4 个单元在内存中对应的地址分别为 0x7839E020、0x7839E024、0x7839E028、0x7839E02B(0x 表示十六进制)。然后在栈内存中分配一个单元用于存储第一个元素的地址，如图 5-1(a)所示。因此，数组首地址为 d[0]的地址，即 0x7839E020，也即数组 d 的地址。也就是说，数组名字 d 就表示数组的首地址。

注意：由于元素类型为 int，每个元素占 4B，因此各元素地址之间相差为 4。

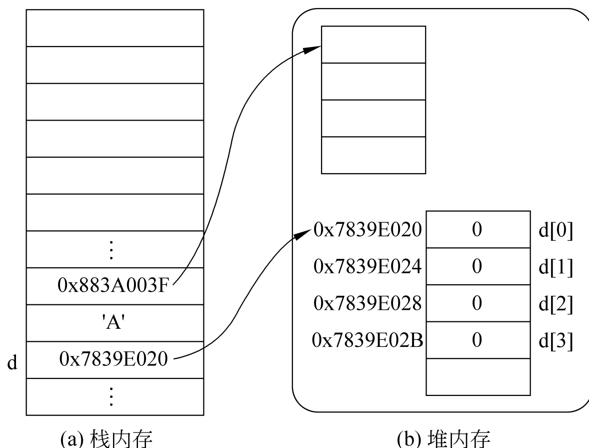

图 5-1 数组实例化时的内存分配

Java 把内存分成两种：栈内存和堆内存。在方法中定义的基本数据类型的变量和对象的引用变量都是在方法的栈内存中分配的，当在一段代码块中定义一个变量时，Java 就在栈中为这个变量分配内存空间，当超过变量的作用域后，Java 会自动释放掉为该变量分配的内存空间，该内存空间就可以立即被另做他用。

堆内存用来存放由 new 创建的对象和数组，在堆中分配的内存，由 Java 虚拟机的自动垃圾回收器来管理。在堆中产生了一个数组或者对象之后，还可以在栈中定一个特殊的变量，让栈中的这个变量的取值等于数组或对象在堆内存的首地址，栈中的这个变量就成了数组或对象的引用变量，以后就可以在程序中使用栈中的引用变量来访问堆中的数组或者对象，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，而数组和对象本身在堆中分配，即使程序运行到使用 new 产生数组或者对象的语句所在的代码块之外，数组和对象本身占据的内存不会被释放，数组和对象在没有引用变量指向它时，才变为垃圾，不能再被使用，但仍

然占据内存空间不放，在随后的一个不确定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。实际上，栈中的变量指向堆内存中的变量，这就是 Java 中的“指针”。

3. 一维数组的初始化

数组实例化以后，在内存中实际存储的数据是什么呢？在图 5-1 的实例中，由于数据类型为 int，因此实例化时各单元默认分配数据为 0，即 d[0]、d[1]、d[2]、d[3]都为 0。对于 8 种基本数据类型或引用类型而言，在数组实例化时，其默认值如表 5-1 所示。

表 5-1　8 种基本数据类型的数组实例化时的默认值

数组声明与实例化	数组元素默认值
boolean b[]=new boolean[10];	false
char c[]=new char[10];	'\0'（若用 println 输出，无显示）
int i[]=new int[10];	0
byte by[]=new byte[10];	0
short sh[]=new short[10];	0
long l[]=new long[10];	0
float f[]=new float[10];	0.0
double d[]=new double[10];	0.0
引用类型	null

除了默认值外，也可以在声明的同时进行初始化。

例如：

```
int a[]={1, 2, 3, 4};                    //注意没有 new
char[] c={'A', 'B', 'C', 'D'};
String[] str={"How", "are", "you"};
```

其中，数组 c 初始化时的内存分配如图 5-2 所示。

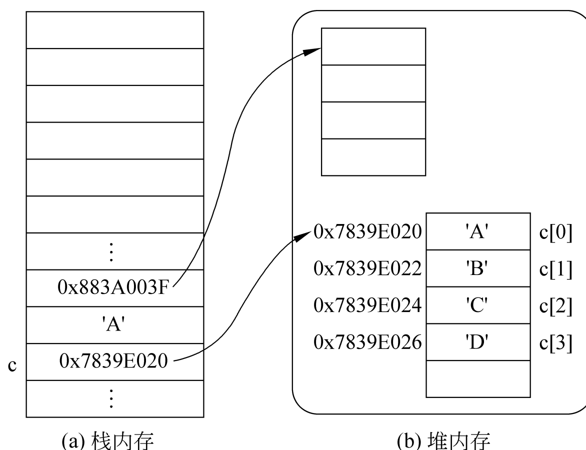

图 5-2　数组 c 初始化时的内存分配

注意：由于元素类型为char，每个元素占2B，因此各元素地址之间相差为2。

这种初始化方法仅限于8种基本数据类型和String型，若为引用类型，则需要采用new关键词对每个对象进行实例化。

例如：

```
Student st[];
st=new Student[3];          //或者把这两句合成一句：Student st[]=new Student[3];
st[0]=new Student();
st[1]=new Student();
st[2]=new Student();
```

5.1.2 二维数组

1. 二维数组的声明

二维数组的声明格式可以是如下任意一种：

```
数组元素类型  数组名字[][];
数组元素类型[][] 数组名字;
```

例如：

```
int[][] d;
String[][] names;
char c[][];
String s[][];
```

例如，声明一个数据类型为Student类的数组：

```
Student[][] s;            //等价于 Student s[][]
```

2. 二维数组的实例化

二维数组的实例化仍然使用new来实现。

例如：

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

或者

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

这样为二维数组t进行了实例化，分配了内存空间，它相当于定义了一个长度为2的一维数组，而该一维数组的每个元素又是一个长度为3的一维数组。二维数组可理解成如下的表格。

t[0][0]	**t[0][1]**	**t[0][2]**
t[1][0]	t[1][1]	t[1][2]

二维数组还可以动态实例化。

例如:

```
int a[][]=new int[2][];            //分配两行,即两个一维数组
a[0]=new int[3];                   //第一个一维数组有 3 个元素
a[1]=new int[5];                   //第二个一维数组有 5 个元素
```

注意: Java 语言中,二维数组被看作是数组的数组,数组空间不是连续分配的,所以不要求二维数组每一维的大小相同。

二维数组 t 在实例化时,首先在堆内存中分配一维数组的空间,即 t[0]、t[1],其对应的内存地址分别为 0x7839E020 和 0x7839E024;然后,在堆内存中开辟 24B 的内存空间用于存放 int 数据,如图 5-3(b)所示,其中每 4B 构成一个单元,共 6 个单元,分别存储 t[0][0]、t[0][1]、t[0][2]、t[1][0]、t[1][1]、t[1][2],这 6 个单元在内存中对应的地址分别为 0x8849F080、0x8849F084、0x8849F088、0x9458B050、0x9458B054、0x9458B058(0x 表示十六进制)(多维数组中维与维之间的内存空间分配是不连续的);其次,将 t[0][0]的地址存入 t[0]单元中,即将 0x8849F080 存入 0x7839E020 表示的内存单元,将 t[1][0]的地址存入 0x7839E024 表示的内存单元;最后,在栈内存中分配一个单元用于存储 t[0]元素的地址,如图 5-3(a)所示。因此,数组 t 的地址即 0x7839E020。也就是说,数组名字 t 就表示数组的入口地址。

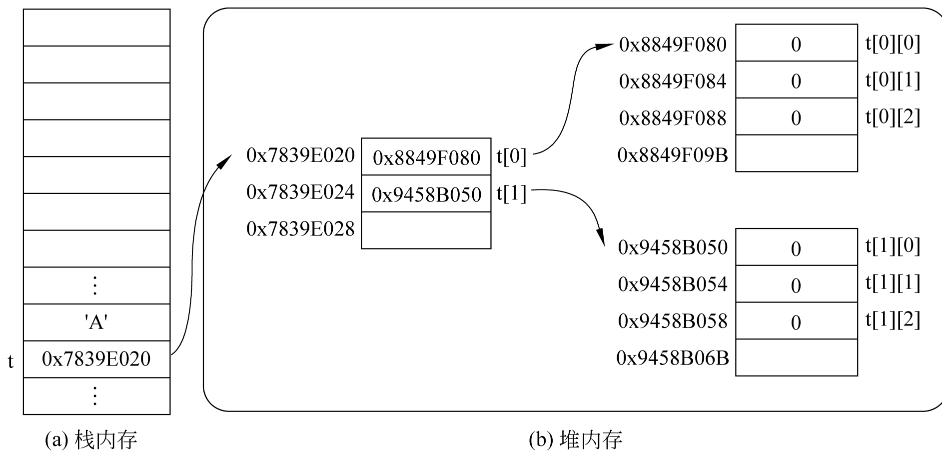

图 5-3　二维数组实例化时的内存分配

注意: 由于元素类型为 int,每个元素占 4B,因此各元素地址之间相差为 4。

3. 二维数组的初始化

1) 直接初始化

对于 8 种基本数据类型和 String 类型的数组,可进行直接初始化。

例如:

```
int iArray[][]={{1, 2}, {3, 4}, {5, 6}};      //表示 3 行 2 列的二维数组,相当于一个表格
char cArray[][]={{'A', 'B'}, {'C'}, {'D', 'E', 'F'}};
String sArray[][]={{"Hello", "World"}, {"How", "Are", "You"}};
```

在上例中，数组 cArray 有 3 行，而每一行的元素个数分别为 2 个、1 个、3 个。也就是说，cArray 由 3 个一维数组构成，即 cArray[0]、cArray[1]、cArray[2]，而这 3 个一维数组的元素分别为{cArray[0][0]，cArray[0][1]}、{cArray[1][0]}、{cArray[2][0]，cArray[2][1]，cArray[2][2]}。

2）动态初始化

对于引用类型的二维数组，必须首先为最高维分配引用空间，然后再顺次为低维分配空间，而且必须为每个数组元素单独分配空间（否则将不能使用）。

例如：

```
Student[][] s=new Student[2][];          //为最高维分配引用空间
s[0]=new Student[2];                     //为低维分配引用空间
s[1]=new Student[2];                     //为低维分配引用空间
s[0][0]=new Student();                   //为每个数组元素单独分配引用空间
s[0][1]=new Student();                   //为每个数组元素单独分配引用空间
s[1][0]=new Student();                   //为每个数组元素单独分配引用空间
s[1][1]=new Student();                   //为每个数组元素单独分配引用空间
```

5.1.3 数组的注意事项

（1）当通过循环遍历数组时，下标永远不要小于 0，下标永远要比数组元素个数小。

（2）当数组下标出错（小于 0、大于或等于数组元素个数），Java 产生 ArrayIndexOutOfBoundsException 异常。

（3）一旦创建数组后，不能调整数组大小，但可使用相同的引用变量来引用一个全新的数组。

例如：

```
int[]a=new int[6];
a=new int[10];
```

（4）数组是一种特殊的对象，其长度可以用“数组名.length”来引用，如 c.length，对于二维数组，如前述例子中的数组 cArray，cArray.length 返回最高维的长度（二维数组的行数），值为 3；cArray[2].length 表示第三行低维数组的长度，值为 3。

5.1.4 数组的应用

1. 数组应用举例

【程序 5-1】 AvgScores.java。

```
/*
已知 3 位同学的高等数学、汇编语言以及 Java 程序设计三门课程的成绩，计算每个同学的平均分数并输出到屏幕。
*/
import java.text.DecimalFormat;
class AvgScores{
    public static void main(String args[]){
```

```
        String names[]={"丁一","丁二","丁三"};                //三位同学的姓名
        DecimalFormat df=new DecimalFormat("#");             //用于输出时的格式化
        int[][] iScores={{89,98,87},{90,88,95},{86,92,93}}; //三门课程的成绩
        int i, j;
        double avg;                            //临时变量,用于保存平均成绩
        for(i=0;i<iScores.length;i++){
            avg=0.0;                           //初始化每个同学的平均成绩
            for(j=0;j<iScores[i].length;j++)
                avg+=iScores[i][j];            //先用 avg 保存三门课程的总和

            avg=avg/j;                         //此时的 j 表示一共有几门课
            System.out.println(names[i]+":"+df.format(avg));        //格式化输出
        }
    }
}
```

输出结果如图 5-4 所示。

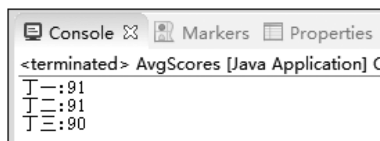

图 5-4　输出结果

2. 数组作为方法参数及返回值

数组可以作为参数传递给方法,也可以作为方法的返回值。在调用的方法中的数组对象与调用者中的是同一个。如果在方法中修改了任何一个数组元素,因为这个数组与方法之外的数组对象是同一个,所以方法之外的数组也将发生改变。例如程序 5-2。

【程序 5-2】　CallArray.java。

```
/*数组作为调用方法的传递参数*/
public class CallArray{
    static void updateArray (int[ ] arrays){
          arrays[3]=10;
    }
    public static void main(String []args){
        int [ ]hold={0,1,2,3,4,5,6,7,8,9};
        updateArray(hold);
        for(int i=0;i<hold.length;i++)
            System.out.print(hold[i]+" ");
    }
}
```

运行时,hold 数组的内容如下：0，1，2，10，4，5，6，7，8，9,可见 arrays[3]中的元素已经被修改了。

【程序 5-3】　ReturnArray.java。

```
/* 数组作为方法的返回结果 */
public class ReturnArray{
    static int[ ] updateArray(int[] arrays){
        for(int i=0;i<arrays.length;i++){
            arrays[i]=i;
        }
        return arrays;
    }
    public static void main(String []args){
        int[] hold={9,8,7,3,5,6,4,2,1,0};
        hold=updateArray(hold);
        for(int i=0;i<hold.length;i++)
            System.out.println("hold["+i+"]="+hold[i]);
    }
}
```

运行之后，输出结果如下：

```
hold[0]=0
hold[1]=1
hold[2]=2
hold[3]=3
hold[4]=4
hold[5]=5
hold[6]=6
hold[7]=7
hold[8]=8
hold[9]=9
```

3. main 方法的数组参数获取

在 Java 程序的主方法 public static void main (String args[ ]) 中，args[ ]是一个字符串数组，用来接收应用程序被调用时由用户直接从键盘输入的参数。

【程序 5-4】 MyFriend.java。

```
/* 获取 main 方法的数组参数 */
public class MyFriend{
    public static void main (String arg[]){
        if(arg.length>=2)                    //判断输入参数是否多于两个
            System.out.println(arg[0]+" and "+arg[1]+" are my good friends! ");
    }
}
```

运行及输出如图 5-5 所示，其中的 Tom 对应于 arg［0］，Alice 对应于 arg［1］，命令行中 MyFriend Tom Alice 各单词之间用空格隔开。

```
G:\java1\chap5>javac MyFriend.java

G:\java1\chap5>java MyFriend Tom Alice
Tom and Alice are my good friends!

G:\java1\chap5>
```

图 5-5　main 方法的数组参数输入实例

5.2 字 符 串

字符串是字符的序列。在 C/C++ 语言中,把字符串作为字符数组来处理,明确以字符'\0'作为字符串结束的标志,因此在进行字符串处理时比较容易发生错误。

在 Java 语言中,字符串是用双引号分隔的一系列字符,如"Java is very interesting and easy to learn!"。在 Java 中字符串是作为对象处理的,在对象中封装了一系列方法进行字符串处理,不仅减少了程序设计的工作量,而且规范了程序编制,减少了错误的发生。

Java 提供了 String 和 StringBuffer 两个字符串类,它们均在 java.lang 包中。String 表示的字符串在初始化之后,不能被修改;StringBuffer 中的字符串内容可以被动态修改。

字符串中每个字符的位置是从左边开始,第 1 个字符的位置标号为 0,第 2 个字符的位置标号为 1,以此类推。

5.2.1 不可变字符串 String

1. String 的声明与实例化

对字符串的声明与实例化,可如下:

```
String name;
name=new String("Latte" );
```

或

```
String name;
name="Latte";
```

或

```
String name="Latte";
```

或

```
String name=new String("Latte");
```

上面最后一行是调用了 String 的构造方法进行初始化。在 Java SDK 1.6.0 中,String 常用的构造方法如表 5-2 所示。

表 5-2 String 常用的构造方法

构造方法	说明
String()	初始化一个新创建的空字符序列的 String 对象
String(byte[] bytes)	用一个 byte 数组构造一个新的 String
String(byte[] bytes, int offset, int length)	利用 bytes 数组中从 offset 开始的 length 个字符构造一个新的 String

续表

构造方法	说明
String(char[] value)	用 char 数组分配一个新的 String
String(char[] value, int offset, int count)	利用 value 数组中从 offset 开始的 count 个字符构造一个新的 String
String(String original)	初始化一个新创建的 String 对象,使其表示一个与参数相同的字符序列;换句话说,新创建的字符串是该参数字符串的副本
String(StringBuffer buffer)	分配一个新的字符串,它包含字符串缓冲区参数中当前包含的字符序列

例如:

```
char[] cArray1={'B', 'C', 'D', 'E'};
char[] cArray2={'A', 'B', 'C', 'D', 'E'};
String s1=new String(cArray1);        //用 char 数组初始化字符串 s1
String s2=new String(cArray2, 1, 4); //用 char 数组的下标为 1 开始的 4 个字符初始化 s2
```

则生成的 s1 与 s2 的内容均为"BCDE"。

例如:

```
byte[] cArray3={66, 67, 68};
byte[] cArray4={65, 66, 67, 68};      //分别为'A'、'B'、'C'、'D'的十进制 ASCII 码表示
String s3=new String(cArray3);        //用 byte 数组初始化 s3
String s4=new String(cArray4, 1, 3); //用 byte 数组的下标为 1 开始的 3 个字符初始化 s4
```

则生成的 s3 与 s4 的内容均为"BCD"。

2. String 字符串的内存分配

String 字符串的内存分配根据不同的情况在栈内存或堆内存中进行,在栈内存的空间通常又称为字符串常量池(Constant Pool)。常量池指的是在编译期被确定,并被保存在已编译的.class 文件中的一些数据,它包括关于类、方法、接口等中的变量,也包括字符串常量。首先,要知道 Java 会确保一个字符串常量只有一个副本。当一个字符串由多个字符串常量连接而成时,它自己肯定也是字符串常量。用 new String()创建的字符串不是常量,不能在编译期就确定,所以 new String()创建的字符串不放入常量池中,它们有自己的地址空间,即堆空间。下面举例说明其详细的内存分配。

例如:

```
String s1="hello";
String s2="hello";
String s3="he"+"llo";
String s4=new String("hello");
String s5=new String("hello");

s1=s1+" world";
```