第 4 章　数　　组

Java 语言中的数组是用来存放同一种数据类型数据集的特殊对象。在本章中，首先介绍数组的创建、初始化，然后是它们的基本使用情况。学习完这些基础知识后，本章会介绍几种基本的排序方法，然后是多维数组的使用。有些内容放在本章讲有些早，但为了使全书的内容组织有序，所以把数组相关的内容都放到了本章来讲解，如果读者觉得有的内容难于理解，可以先暂时放一下，等学习了以后的内容，再来回顾一下，一定会有更好的理解。

4.1　数组基础

数组保存的是一组有顺序的、具有相同类型的数据。在一个数组中，所有数据元素的数据类型都是相同的。可以通过数组下标来访问数组，数据元素根据下标的顺序，在内存中按顺序存放。本节的主要内容是介绍数组的基础知识。

4.1.1　为什么要使用数组

假设有一个程序要求输入一周内每天的天气情况，然后计算这一周内的平均气温。可以通过让用户输入一周七天的每天气温，然后来计算平均气温，最后显示出来。程序代码如下。

```
import java.util.*;
public class AverageTemperatures{
  public static void main(String args[ ]){
        int count;
        double next,sum,average;
        sum=0;
    //创建一个 Scanner 对象
    Scanner sc=new Scanner(System.in);
    System.out.println("请输入七天的温度：");
    for(count=0;count<7;count++)
    {
        //通过 Scanner 对象获得用户输入
        next=sc.nextDouble();
        sum+=next;
        }
        System.out.println(sum);
        average=sum/7;
        System.out.println("平均气温为："+average);
  }
}
```

程序的运行结果如下。

```
请输入七天的温度：
34.5
```

```
30.7
34
28.0
27.9
35.7
31.0
221.8
平均气温为：31.685714285714287
```

程序首先定义了一系列的变量，count 用来表示第几天，next 用来存放每天的气温，sum 用来存放气温的总和，而 average 是用气温的总和除以天数得到的平均值。下面定义了一个 Scanner 对象，Scanner 是一个使用正规表达式来解析基本类型和字符串的简单文本扫描器，可以用来读取用户输入的气温。在循环语句中调用该类的 nextDouble( )方法，读取用户输入的气温，把它放入 next 中，然后加入 sum 中。循环结束后求得气温的平均值 average。

假设程序有进一步的要求，要求记录每一天的气温，那么可以声明七个 double 类型的变量来存放气温。但是这样实现过于“笨拙”。这时候可以用数组来实现，用数组来存放统一类型的数据是十分方便的。

4.1.2 数组的创建与访问

Java 的数组可以看作一种特殊的对象，准确地说是把数组看作同种类型变量的集合。在同一个数组中的数据都有相同的类型，用统一的数组名，通过下标来区分数组中的各个元素。数组在使用前需要对它进行声明，然后对其进行初始化，最后才可以存取元素。下面是声明数组的两种基本形式。

```
ArrayType ArrayName[ ];
ArrayType [ ] ArrayName;
```

符号“[]”说明声明的是一个数组对象。这两种声明方式没有区别，但是第二种可以同时声明多个数组，使用起来较为方便，所以程序员一般习惯使用第二种方法。下面声明 int 类型的数组，格式如下。

```
int array1[ ];
int [ ] array2,array3;
```

在第一行中，声明了一个数组 array1，它可以用来存放 int 类型的数据。第二行中，声明了两个数组 array2 和 array3, 效果和第一行的声明方式相同。

上面的语句只是对数组进行了声明，还没有对其分配内存，所以不可以存放，也不能访问它的任何元素。这时候，可以用 new 对数组分配内存空间，格式如下。

```
array1=new int [5];
```

这时候数组就有了以下 5 个元素。

```
array[0]
array[1]
array[2]
array[3]
array[4]
```

注意：在 Java 中，数组的下标是从 0 开始的，而不是从 1 开始。这意味着最后一个索引号不是数组的长度，而是比数组的长度小 1。

数组是通过数组名和下标来访问的。例如下面的语句，把数组 array1 的第一个元素赋值给 int 型变量 a。

```
int a=array1[1];
```

Java 数组下标从 0 开始，到数组长度-1 结束，如果下标值超出范围，小于下界或大于上界，程序也能通过编译，但是在访问时会抛出异常。下面是一个错误的示例。

```
public class ArrayException
{
  public static void main(String args[ ])
{
  //创建一个容量为 5 的数组
    int[ ] array1=new int[5];
    //访问 array1[5]
    System.out.println(array1[5]);
  }
}
```

程序首先声明了一个大小为 5 的 int 类型数组，前面已经讲到，它的下标最大只能是 4。但在程序中却尝试访问 array1[5]，显然是不正确的。程序会正常通过编译，但是在执行时会抛出异常。程序的运行结果如才。

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at ArrayException.main(ArrayException.java:4)
```

异常是 Java 中一种特殊的处理程序错误的方式，本书在后面章节会详细讲解该内容。读者这里只需要知道访问数组下标越界时会产生 ArrayIndexOutOfBoundsException 的异常即可。

4.1.3　数组初始化

数组在声明创建之后，数组中的各个元素就可以访问了，这是因为在数组创建时，自动给出了相应类型的默认值。默认值根据数组类型的不同而有所不同。下面的程序可以看到各类数组的默认值。

```
public class ArrayDefaultValue{
  public static void main(String args[ ]){
        //创建一个 byte 类型的数组
        byte [ ] byteArray=new byte[1];
        //创建一个 char 类型的数组
        char [ ] charArray=new char[1];
        //创建一个 int 类型的数组
        int    [ ] intArray=new int[1];
        //创建一个 long 类型的数组
         long [ ] longArray=new long[1];
        //创建。。。
        float [ ] floatArray=new float[1];
        double[ ] doubleArray=new double[1];
        String [ ] stringArray=new String[1];
        System.out.println("byte="+byteArray[0]);//打印出各个数组的默认初始化值
```

```
        System.out.println("char="+charArray[0]);
        System.out.println("int="+intArray[0]);
        System.out.println("long="+longArray[0]);
        System.out.println("float="+floatArray[0]);
        System.out.println("double="+doubleArray[0]);
       System.out.println("String="+stringArray[0]);
  }
}
```

程序的运行结果如下。

```
byte=0
char=
int=0
long=0
float=0.0
double=0.0
String=null
```

程序声明了各种类型的数组，并通过 new 来创建它们。然后访问它们的元素，可以看到在创建的时候数组元素会获得一个默认值，这个值跟前面讲解数据类型时各种类型的默认值是一致的。这样做可以避免程序编写中一些不必要的错误，但它没有其他意义，因为使用数组肯定是想存储一定的值，所以需要对它进行自己的初始化。一种方法是使用赋值语句来进行数组初始化，格式如下。

```
int [ ] array1=new int[5];
array1[0]=1;
array1[1]=2;
array1[2]=3;
array1[3]=4;
array1[4]=5;
```

通过上面的语句，数组的各个元素就会获得相应的值，如果没有对所有的元素进行赋值，它会自动被初始化为某个值（如前面所述）。另一种方式是在数组声明的时候直接进行初始化，格式如下。

```
int [ ] array1={1,2,3,4,5};
```

该语句跟上面的语句的作用是一样的。在数组声明的时候直接对其进行赋值，按括号内的顺序赋值给数组元素，数组的大小被设置成能容纳大括号内给定值的最小整数。下面的程序演示了数组的初始化。

```
public class ArrayInital{
  public static void main(String args[ ]){
    //创建一个 int 型数组
    int [ ] array1=new int[5];
    //对数组元素赋值
    array1[0]=1;
    array1[1]=2;
    array1[2]=3;
    array1[3]=4;
    array1[4]=5;
    //另一种数组创建方式
    int [ ] array2={1,2,3,4,5};
    //打印出数组元素
    for(int i;i<5;i++)
```

```
    System.out.println("array[i]="+array1[i]);
  for(int i;i<5;i++)
    System.out.println("array[i]="+array1[i]);
 }
}
```

程序的运行结果如下。

```
array1[0]=1
array1[1]=2
array1[2]=3
array1[3]=4
array1[4]=5
array2[0]=1
array2[1]=2
array2[2]=3
array2[3]=4
array2[4]=5
```

可以看到数组的两种初始化效果是相同的。

4.1.4　length 实例变量

Java 中的数组是一种对象，它会有自己的实例变量。事实上，数组只有一个公共实例变量，也就是 length 变量，这个变量指的是数组的长度。例如，创建下面一个数组：

```
int [ ] array1=new int [10];
```

那么 array1 的 length 的值就为 10。有了 length 属性，在使用 for 循环的时候就可以不用事先知道数组的大小，而写成如下形式。

```
for(int i=0;i<arrayName.length;i++)
```

通过这些基础知识的学习，已经可以解决前面那个求平均温度并得出哪些天高于平均温度，哪些天低于平均温度的程序编写问题了。程序代码如下。

```
import java.util.*;
public class AverageTemperaturesDemo
{
 public static void main(String args[ ])
{
 //声明用到的变量
   int count;
   double sum,average;
   sum=0;
   double [ ]temperature=new double[7];
   //创建一个 Scanner 类的对象，用它来获得用户的输入
   Scanner sc=new Scanner(System.in)；
   System.out.println("请输入七天的温度：");
   for(count=0;count<temperature.length;count++)
   {
       //读取用户输入
       temperature[count]=sc.nextDouble();
       sum+=temperature[count];
       }
      average=sum/7;
```

```
        System.out.println("平均气温为："+average);
    //比较各天气温与平均气温
    for(count=0;count<temperature.length;count++)
    {
      if(temperature[count]<average)
          System.out.println("第"+(count+1)+"天气温低于平均气温");
       else if(temperature[count]>average)
           System.out.println("第"+(count+1)+"天气温高于平均气温");
         else
        System.out.println("第"+(count+1)+"天气温等于平均气温");
        }
  }
}
```

程序的运行结果如下。

```
请输入七天的温度：
32
30
28
34
27
29
35
平均气温为：30.714285714285715
第 1 天气温高于平均气温
第 2 天气温低于平均气温
第 3 天气温低于平均气温
第 4 天气温高于平均气温
第 5 天气温低于平均气温
第 6 天气温低于平均气温
第 7 天气温高于平均气温
```

程序声明一个 double 型数组来存放每天的温度，求得平均温度后，用每天的平均温度与平均温度比较，得到比较结果。

4.2 数组的深入使用

上一节讲解了数组的创建、初始化、访问等基本知识，及一个简单的小程序。在这一节里，将会涉及到数组复杂一点的应用：命令行参数和数组拷贝。命令行参数实际上也是以数组形式存在的，而数组拷贝是 Java 提供的一个方便的操作。

4.2.1 命令行参数

如果读者以前接触过 C 语言或其他参数语言，可能会知道命令行参数。命令行参数就是用户在执行程序时提供的一些参数，以供程序运行时使用，而不是每次都修改源程序中的数据或者通过标准输入输出读取用户输入的参数（现在比较流行使用 Scanner 类）。

仔细观察前面的程序，发现在所有的 Java 程序中都有一个 main 方法，而这个方法带有一个参数 String args[ ]。这个参数就是 main 方法接受的用户输入的参数列表，即命令行参数。程序代码如下。

```
public class ArgsDemo
{
 public static void main(String args[ ])
{
   System.out.println("共接受到"+args.length+"个参数");
   for(int i=0;i<args.length;i++)
     System.out.println("第"+i+"个参数"+args[i]);
}
}
```

编译完程序，输入如下命令执行程序。

```
java ArgsDemo name password email sex
```

程序的执行结果如下：

```
共接受到 4 个参数
第 0 个参数 name
第 1 个参数 password
第 2 个参数 email
第 3 个参数 sex
```

显然，参数列表数组 args 为：

```
args[0]=name
args[1]=password
args[2]=email
args[3]=sex
```

注意：执行程序的命令中，java 和程序名并不在参数列表之中。参数列表的第一个参数为程序名后面的第一个参数，各个参数之间用空格符隔开。

4.2.2　数组拷贝

数组拷贝可以直接把一个数组变量拷贝给另一个，这时候，数组都指向同一个数组。假如有两个数组 array1 和 array2，执行下面语句。

```
array1=array2;
```

这时候两个数组类型变量都指向同一个数组，即原来的 array2 所指向的数组。下面的程序说明了这个问题。

```
public class ArrayCopy{
  public static void main(String args[ ]){
    //创建两个数组
    int [ ] array1={1,2,3};
    int [ ] array2={4,5,6};
    System.out.println("两个数组的初值：");              //打印出两个数组的初值
    for(int i=0;i<array1.length;i++)
          System.out.println("array1["+i+"]="+array1[i]);
    for(int i=0;i<array2.length;i++)
          System.out.println("array2["+i+"]="+array2[i]);
```

```
    array1=array2;                                          //数组拷贝语句
    //打印出两个数组的元素
    System.out.println("执行数组拷贝后两个数组的值：");
       for(int i=0;i<array1.length;i++)
          System.out.println("array1["+i+"]="+array1[i]);
    for(int i=0;i<array2.length;i++)
          System.out.println("array2["+i+"]="+array2[i]);
    System.out.println("改变 array2[0]的值");
    array2[0]=10;                                           //改变 array2 的一个元素
    //打印出改变后的元素值
    System.out.println("array1[0]="+array1[0]);
    System.out.println("array2[0]="+array2[0]);
}
}
```

程序的执行结果如下。

```
两个数组的初值：
array1[0]=1
array1[1]=2
array1[2]=3
array2[0]=4
array2[1]=5
array2[2]=6
执行数组拷贝后两个数组的值：
array1[0]=4
array1[1]=5
array1[2]=6
array2[0]=4
array2[1]=5
array2[2]=6
改变 array2[0]的值
array1[0]=10
array2[0]=10
```

该程序首先声明两个数组 array1 和 array2，并对它们直接进行初始化，访问它们各个元素的值。然后执行下面语句。

```
array1=array2;
```

再访问两个数组各个元素的值，发现现在 array1 的值跟 array2 的值是一样的。下面执行语句。

```
array2[0]=10;
```

改变 array2 的第一个元素的值，可以发现 array1 和 array2 的第一个元素的值都改变了。由此也证明了这一点，执行下面语句。

```
array1=array2;
```

该语句会把 array1 和 array2 都指向同一个数组。其过程示意图如图 4-1 所示。

这样的处理方式有些“粗暴”，如果程序只是想把一个数组的值拷贝给另一个数组，显然该方法并不合适。可以使用 System 类中的 arraycopy 方法。它的使用方式如下。

```
System.arraycopy(fromArray ,fromIndex,toArray,toIndex,length)
```

从指定源数组 fromArray 中复制一个数组，复制从指定的位置 fromIndex 开始，到目标数组 toArray，在指定位置 toIndex 结束，复制 length 个元素。注意目标数组必须有足够的空间

来存放复制的数据，如果空间不足的话，会抛出异常，并且不会修改该数组。

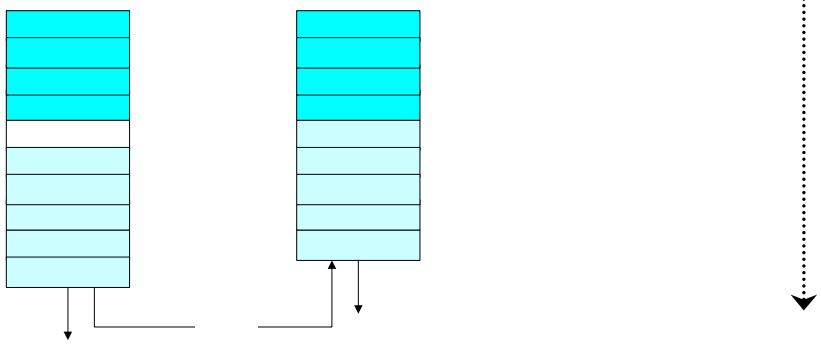

图 4-1　数组赋值

下面是一个具体的说明，例如声明并创建两个数组，把数组 away1 从第一个元素开始的五个元素拷贝到数组 array2 中，从第三个元素开始。

```
int [ ] array1={1,2,3,4,5,6,7,8,9};
int [ ] array2={10,20,30,40,50,60,70,80,90};
System.arraycopy(array1,0,array2,2,5);
```

执行程序后 array2 的值如下。

```
10        20        1        2        3        4        5        80        90
```

arraycopy 方法可以很广泛地使用，下面的程序就演示了它的一个使用，把一个数组的全部内容拷贝到另一个数组的尾部。

```
public class ArrayCopy3
{
  public static void main(String args[ ])
{
    //创建两个数组
    int [ ] array1={1,2,3,4,5,6,7,8,9};
    int [ ] array2=new int[20];
    for(int i=0;i<10;i++)
        array2[i]=10*i;
    System.arraycopy(array1,0,array2,10,array1.length);
    for(int i=0;i<array2.length;i++)
      System.out.println(array2[i]);
}
}
```

程序声明两个数组，对第一个数组 array1 直接进行初始化，里面有 9 个元素；声明 array2 是长度为 20 的数组，对其前 10 个元素进行赋值。假设这样只使用了 array2 的前 10 个元素，然后将 array1 的全部元素拷贝到数组 array2 的尾部。程序的运行结果如下。

```
0
10
20
30
40
50
```

```
60
70
80
90
1
2
3
4
5
6
7
8
9
0
```

由于最后一个元素一直未进行初始化，所以它仍为默认值 0。

4.3 数组排序

假设数组中已经有一些数组，有时候会要求对它们由高到低或者由低到高地进行排列。这时候就要用到数组排序算法，排序算法是算法和数据结构中的主要内容。本节将主要介绍选择排序、冒泡排序、快速排序 3 种算法。

4.3.1 选择排序

选择排序是一种比较简单的排序方法，非常容易理解。选择排序的基本思路是：对一个长度为 n 的数组进行 n 趟遍历，第一趟遍历选出最大（或者最小）的元素，将之与数组的第一个元素交换；然后进行第二趟遍历，再从剩余的元素中选出最大（或者最小）的元素，将之与数组的第二个元素交换。这样遍历 n 次后，得到的就为降序（或者升序）数组。

在数组的排序中需要对数组的两个元素进行交换，这时候使用赋值语句来实现，所以需要一个临时变量来存放数组元素，例如如果要交换数组元素 a[i]和 a[j]就需要使用如下操作。

```
int temp;
temp=a[i];
a[i]=a[j];
a[j]=a[i];
```

这种在排序过程中需要对数组元素进行交换的排序算法，称为“交换排序算法”。选择排序就是一种典型的交换排序算法。这里先用伪代码来表示选择排序算法的基本流程，伪代码就是用类似于书面语的形式来表示程序的基本过程，便于对算法的理解。下面是排序算法的伪代码。

```
void sort()
{
 KeyType key;                    //记录最大或者最小值
 int   index;                    //记录最大或者最小值的位置所在
 for(int i=0;i<n-1;i++)          //从 a[i]，a[2]……a[n]中选择最大或最小值放入 a[i]中
 {
```

```
   index=i;
   key=a[i];                          //初始化关键字为 a[i]
   for(int j=i;j<n;j++)               //用关键字比较每个元素，如果满足条件对关键字 key 和 index 进行调整
    {
       if(a[j]<key){
         key=a[j];
         index=j
            }
}
Swap(a[i],a[index]);                  //进行交换操作
}
```

下面是完整的程序，代码如下。

```
public class SelectionSort{
   public static void main(String args[ ]){
   int [ ]intArray={12,11,45,6,8,43,40,57,3,20};
   int keyValue;
   int index;
   int temp;
   System.out.println("排序前的数组:");
   for(int i=0;i<intArray.length;i++)
          System.out.print(intArray[i]+" ");
   System.out.println();
   for(int i=0;i<intArray.length;i++)
    {
       index=i;
       keyValue=intArray[i];
       for(int j=i;j<intArray.length;j++)
         if(intArray[j]<keyValue)
          {
              index=j;
              keyValue=intArray[j];
              }
         temp=intArray[i];
         intArray[i]=intArray[index];
         intArray[index]=temp;
   }
   System.out.println("排序后的数组:");
   for(int i=0;i<intArray.length;i++)
     System.out.print(intArray[i]+"       ");
}
}
```

程序首先声明了一个数组，输出其排序前的内容。然后对数组进行选择排序，再输出排序后的数组内容。程序的运行结果如下。

```
排序前的数组:
12        11        45        6         8         43        40        57        3         20

排序后的数组:
3         6         8         11        12        20        40        43        45        57
```

选择排序的效率比较低，但它的实现是很简单的，有的算法尽管效率比较高，但是实现起来比较复杂。如果需要匆忙地写一个排序算法，选择排序是比较合适的，因为它简单且容易实现，出现编码错误的可能性很小。

4.3.2 冒泡排序

冒泡排序也是一种交换排序算法。冒泡排序的过程，是把数组元素中较小的看作是“较轻”的，对它进行“上浮”操作。从底部开始，反复地对数组进行“上浮”操作 n 次，最后得到有序数组。首先介绍它的伪代码。

```
void sort()
//冒泡排序，数组的长度为 n
{
  for(int i=0;i<n;i++)
    for(int j=i;j<n;j++)
      if(a[j]<a[i])
        swap(a[i],a[j]);//交换操作，如前面选择排序讲到的
}
```

具体的代码程序如下。

```
public class BubbleSort{
    public static void main(String args[ ]){
    int [ ]intArray={12,11,45,6,8,43,40,57,3,20};
    System.out.println("排序前的数组:");
        for(int i=0;i<intArray.length;i++)
            System.out.print(intArray[i]+" ");
    System.out.println();
    int temp;
    for(int i=0;i<intArray.length;i++)
    {
        for(int j=i;j<intArray.length;j++)
        {
                if(intArray[j]<intArray[i])
                {
                    temp=intArray[i];
                    intArray[i]=intArray[j];
                    intArray[j]=temp;
                }
        }
    }
    System.out.println("排序后的数组:");
    for(int i=0;i<intArray.length;i++)
      System.out.print(intArray[i]+"       ");
}
}
```

程序首先声明了一个数组，输出其排序前的内容。然后对数组进行冒泡排序，再输出排序后的数组内容。程序的运行结果如下。

```
排序前的数组:
12        11        45        6         8         43        40        57        3         20

排序后的数组:
3         6         8         11        12        20        40        43        45        57
```

冒泡排序的实现跟选择排序的思路差不多，但是冒泡排序的过程更简洁，由于它的这种

简单性和简洁性，而成为最常用的排序算法之一。

4.3.3　快速排序

快速排序是最有名的排序算法，它的效率很高，这也是它名字的由来。但是它实现起来是很复杂的，一般数据结构或者算法的书都会对它进行详细的讲解。由于这一节主要讨论数组的排序，尽管读者现在接触这么复杂的排序可能有困难，但笔者还是打算把它写出来，读者可以根据自己的实际情况来阅读。程序代码如下。

```
public class SelectionSort
{
   public static void main(String args[ ])
{
   int [ ]intArray={12,11,45,6,8,43,40,57,3,20};
   int keyValue;
   int index;
   int temp;
   System.out.println("排序前的数组:");
   for(int i=0;i<intArray.length;i++)
          System.out.print(intArray[i]+" ");
   System.out.println();
   for(int i=0;i<intArray.length;i++)
    {
       index=i;
       keyValue=intArray[i];
       for(int j=i;j<intArray.length;j++)
        if(intArray[j]<keyValue)
         {
             index=j;
             keyValue=intArray[j];
             }
        temp=intArray[i];
        intArray[i]=intArray[index];
        intArray[index]=temp;
   }

   System.out.println("排序后的数组:");
   for(int i=0;i<intArray.length;i++)
    System.out.print(intArray[i]+"       ");
}
}
```

4.4　多维数组

前面介绍的都是一维数组，多维数组类似于空间表示中的一维空间、二维空间、多维空间。Java 是支持多维数组的，并利用多个下标来表示多维。本节的主要内容以二维数组为例进行介绍，多维数组的原理与此类似。

4.4.1 多维数组基础

多维数组用多个索引来访问数组元素，它适用于表示表或其他更复杂的内容。例如，当需要表示表 4-1 的时候，使用二维数组会比较方便。

表 4-1 各种利率投资增长

年数	5.00%	5.05%	6.00%	6.05%
1	1050	1055	1060	1065
2	1103	1113	1124	1134
.....	.....	.....	.....	.....

声明多维数组的时候需要一组方括号来制定他的下标。下面的语句是声明一个名为 twoD 的 int 型二维数组：

```
int [ ][ ]twoD=new int[5][5];
```

上面的语句声明了一个 5 行 5 列的二维数组，数组的初始化有以下两种方法。如一维数组直接赋值的方法。

```
twoD={
        {1,2,3,4,5},
        {6,7,8,9,10},
        {11,12,13,14,15},
        {16,17,18,19,20},
        {21,22,23,24,25}
};
```

也可以使用循环访问数组的每个元素的方法对数组元素进行赋值。

```
  for(int i=0;i<twoD2.length;i++)
     for(int j=0;j<twoD2[i].length;j++)
        twoD2[i][j]=k++;
```

在上面的两个 for 循环中，twoD2.length 表示的是数组的行数，而 twoD2[i].length 表示的则是数组的列数。这一点在后面会讲解，请读者记住。下面是完整的程序，通过运行结果可以看到，两种初始化方法效果相同。

```
public class TwoD{
 public static void main(String args[ ]){
  //创建一个二维 int 型数组
  int [ ][ ] twoD1={
        {1,2,3,4,5},
        {6,7,8,9,10},
        {11,12,13,14,15},
        {16,17,18,19,20},
        {21,22,23,24,25}
           };
  //另一种创建方式
  int [ ][ ]twoD2=new int[5][5];
  int k=1;
  for(int i=0;i<twoD2.length;i++)
     for(int j=0;j<twoD2[i].length;j++)
```

```
            twoD2[i][j]=k++;
    System.out.println("输出数组 twoD1:");
    //使用双重循环访问数组
    for(int i=0;i<twoD1.length;i++)
       {
        for(int j=0;j<twoD1[i].length;j++)
             System.out.print(twoD1[i][j]+"      ");
             System.out.println();
        }
    System.out.println("输出数组 twoD2:");
    for(int i=0;i<twoD2.length;i++)
       {
        for(int j=0;j<twoD2[i].length;j++)
             System.out.print(twoD2[i][j]+"      ");
             System.out.println();
    }
}
}
```

程序的运行结果如下。

```
输出数组 twoD1:
1        2        3        4        5
6        7        8        9        10
11       12       13       14       15
16       17       18       19       20
21       22       23       24       25
输出数组 twoD2:
1        2        3        4        5
6        7        8        9        10
11       12       13       14       15
16       17       18       19       20
21       22       23       24       25
```

4.4.2　多维数组的实现

在 Java 中实际上只有一维数组，多维数组可看作是数组的数组。例如，声明如下一个二维数组。

```
int [ ][ ]twoD=new int[5][6];
```

二维数组 twoD 的实现是数组类型变量指向一个一维数组，这个数组有 5 个元素，而这 5 个元素都是一个有 6 个整型数的数组。

twoD[i]表示指向第 i 个子数组，它也是一个数组类型，甚至可以将它赋值给另一个相同大小、相同类型的数组。

下面的程序首先声明了一个二维数组，然后把这个二维数组的第一行赋值给另一个数组，并且交换这个二维数组的第一行和最后一行。

```
public class TwoD2{
  public static void main(String args[ ]){
    //创建一个二维数组
    int [ ][ ] twoD1={
          {1,2,3,4,5},
          {6,7,8,9,10},
```

```
        {11,12,13,14,15},
        {16,17,18,19,20},
        {21,22,23,24,25}
          };
    //创建一个一维数组作为中间变量
    int [ ]array1=new int[5];
    //把 twoD 的第一行赋值给 array1
    array1=twoD1[0];
    //交换二维数组的两行
    twoD1[0]=twoD1[4];
    twoD1[4]=array1;
    System.out.println("得到的一维数组 array1");
    for(int i=0;i<array1.length;i++)
       System.out.print(array1[i]+"      ");
    System.out.println();
    System.out.println("交换后的二维数组 twoD1");
    for(int i=0;i<twoD1.length;i++)
     {
      for(int j=0;j<twoD1[i].length;j++)
          System.out.print(twoD1[i][j]+"      ");
          System.out.println();
       }
}
}
```

程序的运行结果如下。

```
得到的一维数组 array1
1          2          3          4          5
交换后的二维数组 twoD1
21         22         23         24         25
6          7          8          9          10
11         12         13         14         15
16         17         18         19         20
1          2          3          4          5
```

4.4.3 不规则数组

既然 Java 中的多维数组实质上都是一维数组，那么每个数组的数组是否可以大小不同？答案是肯定的。这一节将会介绍这类不规则数组。例如，可以如下声明一个二维数组：

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

数组的行数在数组声明时必须确定，列数可以再确定。上面的数组就可以按如下的声明确定各个行的列数。

```
twoD[0]=new int[1];
twoD[1]=new int[2];
twoD[2]=new int[3];
twoD[3]=new int[4];
```

经过上面的声明，在二维数组 twoD 中，第一行有一个元素，第二行有两个元素，第三行有三个元素，第四行有四个元素。这个数组是一个不规则的数组。下面的程序完整地演示了不规则数组的声明和使用。

```
public class TwoD3{
 public static void main(String args[ ]){
 //创建一个二维数组，指定它的行数
 int [ ][ ]twoD=new int[4][ ];
 //指定各行的列数
 twoD[0]=new int[1];
 twoD[1]=new int[2];
 twoD[2]=new int[3];
 twoD[3]=new int[4];
 int k=1;
 //对数组元素进行赋值
 for(int i=0;i<twoD.length;i++)
    for(int j=0;j<twoD[i].length;j++)
       twoD[i][j]=k++;
 System.out.println("得到的不规则二维数组为：");
 for(int i=0;i<twoD.length;i++)
 {
    for(int j=0;j<twoD[i].length;j++)
       System.out.print(twoD[i][j]+"     ");
    System.out.println();
 }
}
}
```

程序的运行结果如下。

```
得到的不规则二维数组为：
1
2        3
4        5        6
7        8        9        10
```

需要注意的一点是，在程序中 twoD.length 表示二维数组 twoD 的行数，而 twoD[i].length 表示第 i 行的列数。

4.4.4　用二维数组来表示银行账单

这一节里，主要讲解一个简单实例——用二维数组来表示银行账单。如表 4-2 所示，需要首先有一个一维数组来记录各种不同的利率，初始化第一年相同的金额为 1000，然后计算不同年份的额度。

表 4-2　各种利率投资增长

年数	5.00%	5.05%	6.00%	6.05%
1	1050	1055	1060	1065
2	1103	1113	1124	1134
……	……	……	……	……

程序代码如下。

```
public class BankBalance{
```

```
public static void main(String args[ ]){
  //用一个一维数组来表示利率
  double rate[ ]={5.00/100,5.05/100,6.00/100,6.05/100};
  //表示账单的二维数组
  int[ ][ ] balance=new int[10][4];
  for(int i=0;i<balance[0].length;i++)
  balance[0][i]=1000;
  //计算账单的值
  for(int i=1;i<balance.length;i++)
     for(int j=0;j<rate.length;j++)
      {
            double inc=balance[i-1][j]*rate[j];
        balance[i][j]=(int)(balance[i-1][j]+inc);
}
  //打印出结果
  System.out.print("years"+"     ");
  System.out.println("5.00%"+" "+"5.05%"+"      "+"6.00%"+"      "+"6.05%");
  for(int i=0;i<balance.length;i++)
  {
      System.out.print(i+"  ");
     for(int j=0;j<balance[i].length;j++)
       System.out.print(balance[i][j]+" ");
    System.out.println();
  }
}
}
```

程序首先定义了一个一维的 double 型数组 rate，用来存储不同的利率，然后定义了一个描述 10 行 4 列账单用的数组，然后把该数组的第一行初始化为 1000，表示本金。计算每年在不同的利率下本金利息总额，并且放入相应的数组位置中存储，最后取出数组输出。程序的运行结果如下。

```
years    5.00%    5.05%    6.00%    6.05%
0        1000     1000     1000     1000
1        1050     1050     1060     1060
2        1102     1103     1123     1124
3        1157     1158     1190     1192
4        1214     1216     1261     1264
5        1274     1277     1336     1340
6        1337     1341     1416     1421
7        1403     1408     1500     1506
8        1473     1479     1590     1597
9        1546     1553     1685     1693
```

4.5 For-Each 循环语句

For-Each 循环是 J2SE 5 中引入的，它是 for 循环的一种缩略形式，通过它可以简化复杂的 for 循环结构。For-Each 循环主要用在集合（如数组）中，按照严格的方式，从开始到结束循环，它的使用是非常方便的。

4.5.1　For-Each 循环的一般使用

在前面获取数组中的所有元素时，通常使用 for 循环来获取，可以看出是比较麻烦的。在新版本的 Java 中就可以使用 For-Each 循环来进行获取，其相对简单得多。For-Each 循环的一般格式如下。

```
for(数据类型 变量 : 集合)
语句块
```

在 for 关键字后面的括号里先是集合的数据类型，接着是一个元素用于进行操作，它代表了当前访问的集合元素，然后是一个冒号，最后是要访问的集合。一般访问一个数组使用的格式如下。

```
int sum=0;
int [ ]nums={1,2,3,4,5,6,7,8,9,0};
for(int i=0;i<nums.length;i++)
    sum=+nums[i];
```

这种方式是比较复杂的，如果使用 For-Each 循环来重写该段代码是非常简单的，它的格式如下。

```
        int sum=0;
        int [ ]nums={1,2,3,4,5,6,7,8,9,0};
        for(int i:nums)
            sum+=i;
```

下面是一个完整的示例程序。

```
public class ForEach
{
    public static void main(String[ ] args)
    {
        int sum=0;
        int [ ]nums={1,2,3,4,5,6,7,8,9,0};
        for(int i:nums)
        {
            System.out.println("数组元素:"+i);
            sum+=i;
        }
        System.out.println("数组元素和:"+sum);
    }
}
```

程序的运行结果如下。

```
数组元素:1
数组元素:2
数组元素:3
数组元素:4
数组元素:5
数组元素:6
数组元素:7
数组元素:8
数组元素:9
数组元素:0
```

```
数组元素和:45
```

4.5.2 For-Each 循环访问多维数组

在前面的学习中，并没有使用 for 循环来获取一个多维数组的每一个元素，这是因为该方法是很复杂的。在学习了 For-Each 循环后，该操作就相对简单化了。在 For-Each 循环中是可以循环访问多维数组的，它的格式如下。

```
int nums[ ][ ]=new int[5][5];
for(int x[ ]:nums)
  for(int y:x)
//对元素进行操作
```

首先在第一个 for 循环中，每次访问得到一个一维数组，而在第二次循环中把第一次得到数组作为其集合，就可访问数组元素。下面是一个完整的程序。

```
public class ForEach
{
	public static void main(String[ ] args)
	{
		int sum=0;
		//定义二维数组
		int nums[ ][ ]=new int[4][4];
		int k=0;
		for(int i=0;i<4;i++)
			for(int j=0;j<4;j++)
				nums[i][j]=k++;
		//用双重循环来访问二维数组
		for(int x[ ]:nums)
			for(int y:x)
			{
				System.out.println("数组元素:"+y);
				sum+=y;
			}
		System.out.println("数组元素和:"+sum);
	}
}
```

程序首先声明了一个二维数组，并通过普通的嵌套循环对其赋值，然后通过 For-Each 循环对其进行访问求和操作。程序的运行结果如下。

```
数组元素:0
数组元素:1
数组元素:2
数组元素:3
数组元素:4
数组元素:5
数组元素:6
数组元素:7
数组元素:8
数组元素:9
数组元素:10
数组元素:11
```

```
数组元素:12
数组元素:13
数组元素:14
数组元素:15
数组元素和:120
```

4.6　小结

在本节中主要介绍了数组的内容，分别介绍了数组的创建和使用、数组拷贝、数组排序、以及多维数组的使用。

数组拷贝的讲解使用的是 Java 本身提供的数组拷贝方法，当然读者可以使用自己的方法。读者可以阅读数组拷贝的 Java 源码来养成优秀的编程习惯。数组排序是很重要的一节，但是一般应该把它归类为数据结构和算法的知识。本书作为一本介绍 Java 语言的书，重点不放在这方面，但是本书还是提供了常用的三种算法：选择排序、冒泡排序、快速排序。一般情况下快速排序是效率最高的排序算法，它的实现也比较复杂，读者如果感觉阅读有难度，可以参考相关的数据结构和算法书籍。多维数组的内容主要通过二维数组的形式来讲解，多维数组也是使用类似的形式，只不过是增加下标索引而已。

结束本章的学习，Java 的基础知识已经基本学完，从下一章开始将会学习 Java 面向对象的相关内容。面向对象是 Java 的主要特性，希望大家认真学习接下来的内容。