

第 3 章

程序设计基础之流程控制和数组 ◀

从微观上看程序,程序有 3 种结构,即顺序结构、选择结构和循环结构。本章首先介绍 3 种结构的用法,并讲解了 break 和 continue 语句;然后讲解数组的作用、定义、性质和用法,以及二维数组的使用。

本章术语

if _____
switch _____
while _____
for _____
break _____
continue _____
array _____
引用 _____

3.1 判断结构

3.1.1 为什么需要判断结构

前面所列出的程序一般都是顺序结构,顺序结构就是程序从前到后一行一行执行,直到程序结束。

但是,计算机软件是个智能化产品,需要根据一定的情况进行判断。例如在聊天软件中,当客户收到聊天信息时必须进行提示,这就需要软件进行判断,因此要用判断结构来实现这个功能。

在 Java 中判断结构包括 if 和 switch 两种。

3.1.2 if 结构

if 是最常见的判断结构。

1. 最简单的 if 结构

if 结构最简单的格式如下:

```
if (条件表达式){
```

```
    代码块 A;  
}
```

以上结构表示如果条件成立执行代码块 A, 否则不执行。

例：输入一个客户的年龄, 如果在 0~100, 打印年龄, 否则不打印。

代码如下：

IfTest1.java

```
Public class IfTest1 {  
    public static void main(String[] args) {  
        String strAge =  
            javax.swing.JOptionPane.showInputDialog("输入客户年龄");  
        int age = Integer.parseInt(strAge);  
        if((age >= 0)&&(age <= 100)){  
            System.out.println("年龄为:" + age);  
        }  
    }  
}
```

运行, 效果如图 3-1 所示。

单击“确定”按钮, 控制台打印效果如图 3-2 所示。



图 3-1 IfTest1.java 的效果

年龄为:25

图 3-2 控制台打印效果

注意

(1) 由于键盘输入比较复杂, 在本代码中使用 `javax.swing.JOptionPane.showInputDialog` 函数进行输入, 返回一个字符串。在后面将详细讲解, 此处会用即可。

(2) 如果是一个 `boolean` 类型, 例如 `boolean b`, 若进行判断可写成 `if(b==true)` 或者 `if(b==false)`。但是为了防止将“==”写成“=”, 一般用 `if(b)` 代替 `if(b==true)`, 用 `if(!b)` 代替 `if(b==false)`。

(3) 如果 `if` 后面只跟一条语句, 大括弧可以省略。例如：

```
if((age >= 0)&&(age <= 100))  
    System.out.println("年龄为:" + age);
```

但是当程序复杂时可能会降低程序的可读性, 因此建议不管怎样都加上大括弧。

(4) 在 `if` 判断之后不要直接加分号, 这是初学者容易犯的错误：

```
if((age >= 0)&&(age <= 100)); { // if 后加分号, if 判断在分号处结束  
    System.out.println("年龄为:" + age);  
}
```

2. if-else 结构

if-else 结构的格式如下：

```
if (条件表达式 1){  
    代码块 A;  
}else{  
    代码块 B;  
}
```

以上结构表示如果条件成立执行代码块 A, 否则执行代码块 B。

例：输入一个客户的年龄, 如果在 0~100, 打印“正确”, 否则打印“错误”。

代码如下：

IfTest2. java

```
public class IfTest2 {  
    public static void main(String[] args) {  
        String strAge =  
        javax.swing.JOptionPane.showInputDialog("输入客户年龄");  
        int age = Integer.parseInt(strAge);  
        if ((age >= 0) && (age <= 100)) {  
            System.out.println("正确");  
        } else {  
            System.out.println("错误");  
        }  
    }  
}
```

正确

图 3-3 IfTest2. java 的效果

运行, 输入一个值, 例如 25, 单击“确定”按钮, 控制台打印效果如图 3-3 所示。

注意

另外还有一种和 if-else 类似但是更加紧凑的写法：

条件表达式?结果 1:结果 2

其意义是如果条件表达式成立, 返回结果 1, 否则返回结果 2。

在上面的例子中 if 语句也可以写成：

```
System.out.println((age >= 0) && (age <= 100) ? "正确" : "错误");
```

效果相同。该结构有时候很有用处, 例如：

```
x = x > 0 ? x : - x;
```

将 x 的值变为其绝对值。

3. if-else if-else 结构

if-else 结构只能判断“是否”的关系, if-else if-else 可以判断更加复杂的情况, 格式如下：

```
if (条件表达式 1){
    代码块 1;
}else if(条件表达式 2){
    代码块 2;
}...多个 else if
else{
    代码块 n;
}
```

以上结构表示如果条件 1 成立,执行代码块 1,否则判断条件 2;如果条件 2 成立,执行代码块 2……如果条件都不成立,执行代码块 n。

例:输入一个月份,打印该月份对应的天数。1、3、5、7、8、10、12 月为 31 天,2 月份为 28 天,其他月份为 30 天,如果输入的月份超出范围,打印“错误”。

代码如下:

IfTest3.java

```
public class IfTest3 {
    public static void main(String[] args) {
        String strMonth =
            javax.swing.JOptionPane.showInputDialog("输入月份");
        int month = Integer.parseInt(strMonth);
        if(month == 1 || month == 3 || month == 5 ||
            month == 7 || month == 8 || month == 10 || month == 12){
            System.out.println(month + "月有 31 天");
        }else if(month == 2){
            System.out.println(month + "月有 28 天");
        }else if(month == 4 || month == 6 || month == 9 || month == 11){
            System.out.println(month + "月有 30 天");
        }else{
            System.out.println("错误");
        }
    }
}
```

运行,输入一个值,例如 6,单击“确定”按钮,控制台打印效果如图 3-4 所示。

6月有30天

图 3-4 IfTest3.java 的效果

注意

if 后面可以接多个“else if”,可以不接“else”。

4. if 嵌套使用

很显然,程序是丰富多彩的,在 if 中也可以包含 if,例如:

```
if (age >= 25){
    if(money > 100){
        System.out.println("可以登录");
    }else{
        System.out.println("不可以登录");
    }
}
```

🔔注意

大家要养成使用大括弧的好习惯,哪怕 if 成立后执行的只有一句代码,不要写成:

```
if (age >= 25)
    if (money > 100)
        System.out.println("可以登录");
else
    System.out.println("不可以登录");
```

实际上,最后一个 else 和最近的 if 配对,但是我们很难判断,影响了程序的可读性。

🔔阶段性作业

(1) 输入一个应收金额,输入一个实收金额,显示找零的各种纸币的张数,优先考虑面额大的纸币,显示各种人民币要多少张。假如现有 100、50、20、10、5、1 元的面额,如果实收金额小于应收金额将报错。

(2) 输入一个整数,如果是正数就减去 10,如果是负数就加上 10,然后显示。

3.1.3 switch 结构

switch 结构也可以进行判断,效果和 if-else if-else 类似,但是使用范围稍窄一些。其格式如下:

```
switch (变量名){
    case 值 1:
        代码块 1;
        break;
    case 值 2:
        代码块 2;
        break;
    ...
    default:
        代码块 n;
}
```

以上结构表示,如果变量等于值 1,执行代码块 1; 如果等于值 2,执行代码块 2……如果都不等于,执行代码块 n 。

例: 输入一个月份,打印该月份对应的天数。1、3、5、7、8、10、12 月为 31 天,2 月份为 28 天,其他月份为 30 天,如果输入的月份超出范围,打印“错误”。

代码如下:

SwitchTest1.java

```
public class SwitchTest1 {
    public static void main(String[] args) {
        String strMonth =
            javax.swing.JOptionPane.showInputDialog("输入月份");
        int month = Integer.parseInt(strMonth);
        switch(month){
```

```
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            System.out.println(month + "月有 31 天");
            break;
        case 2:
            System.out.println(month + "月有 28 天");
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            System.out.println(month + "月有 30 天");
            break;
        default:
            System.out.println("错误");
    }
}
```

运行,输入一个值,例如 6,单击“确定”按钮,控制台打印效果如图 3-5 所示。



图 3-5 SwitchTest1.java 的效果

注意

- (1) default 语句是可选的。
- (2) 在“switch(month)”中,被判断的变量只能是 byte、char、short、int 类型。
- (3) “break;”表示跳出这个 switch。如果没有 break,程序会在 switch 内继续向下运行,所以 case 与 else if 还不是等价的。else if 是一旦条件成立就不执行后面的其他 else if 语句;在 switch 中碰到第一个匹配的 case 就会执行 switch 剩余的所有,而不管后面的 case 条件是否匹配,直到碰到 break 语句为止。

阶段性作业

- (1) 删除上例中所有的 break,输入月份 6,看看打印什么。
- (2) 输入一个年份和月份,打印该年该月的天数。规定平年 2 月 28 天,闰年 2 月 29 天;年份能被 4 整除但不能被 100 整除为闰年,能被 400 整除的年份也是闰年。

3.2 认识循环结构

3.2.1 为什么需要循环结构

计算机和人相比优势有两个,第一是精度高,第二是运算快,但是运算再快也是按照人编制的程序进行的。如果要完成一个庞大的工作,例如计算 1~1000 各个整数的和,程序就不能写成:

```
sum = 1 + 2 + 3 + 4 + 5 + ... + 1000;
```

此时可以运用简单的分析设计简单的程序,让计算机通过重复工作帮我们完成复杂的计算。

如何进行分析呢?首先要让计算机进行重复工作代码一定要有重复性:

```
sum = 0, i = 1;  
sum = sum + i; i++;  
sum = sum + i; i++;  
...  
直到 i 加了 1000 次为止。
```

此时,“sum = sum + i; i++;”就是重复代码。不过应该告诉程序重复 1000 次就应该结束,因此还必须指定结束的条件。

在 Java 中可以通过 while、do-while 和 for 结构来实现循环。

3.2.2 while 循环

while 语句是常见的循环语句,结构如下:

```
while (条件表达式){  
    循环体;  
}
```

以上结构表示如果条件成立执行循环体,执行完毕后再判断条件是否成立;如果条件成立,执行循环体……周而复始,直到条件不成立为止。

例:计算 1~1000 各个整数的和。

代码如下:

WhileTest1.java

```
public class WhileTest1 {  
    public static void main(String[] args) {  
        int sum = 0, i = 1;  
        while(i <= 1000){  
            sum += i;  
            i++;  
        }  
        System.out.println("结果为:" + sum);  
    }  
}
```

运行,控制台打印效果如图 3-6 所示。

结果为:500500

☞注意

图 3-6 WhileTest1.java 的效果

(1) 在该循环中,i 称为控制变量,控制循环的运行。每循环一次,i 加 1,这个“1”也称为“步长”。

(2) 如果删掉“i++;”,每次循环时i的值均为1,“i≤1000”永远成立,循环将不会终止,这称为死循环。

(3) while 判断之后不要直接加分号,这是初学者容易犯的错误。

```
while(i <= 1000);{                                //while 后加分号,while 在此处不断空循环,造成死循环
    sum += i;
    i++;
}
```

(4) 如果循环体中只有一条语句,大括弧可以省略。但是当程序复杂时可能会降低程序的可读性,因此建议不管怎样都加上大括弧。

🔗 阶段性作业

(1) 在 WhileTest1 例子中“i++”执行多少次? 循环结束时i的值是多少?“i≤1000”判断了多少次?

(2) 打印0~127各个数字对应的字符。

3.2.3 do-while 循环

do-while 也比较常见,其结构如下:

```
do{
    循环体;
} while (条件表达式);
```

以上结构表示首先执行循环体,然后判断条件,如果条件成立,执行循环体,执行完毕后再判断条件是否成立;如果条件成立,执行循环体……周而复始,直到条件不成立为止。

从这里可以看出,和 while 循环不同的是,do-while 将首先执行循环体,再判断,所以循环体至少执行一次。

例: 计算1~1000各个整数的和。

代码如下:

DoWhileTest1.java

```
public class DoWhileTest1 {
    public static void main(String[] args) {
        int sum = 0, i = 1;
        do{
            sum += i;
            i++;
        }while(i <= 1000);
        System.out.println("结果为:" + sum);
    }
}
```

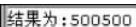
结果为:500500

图 3-7 DoWhileTest1.java 的效果

运行,控制台打印效果如图 3-7 所示。

注意

在 do-while 中,while 判断之后的分号不能丢:

```
do{
    sum += i;
    i++;
}while(i <= 1000)                //丢掉分号,出现语法错误
```

这是初学者容易犯的错误。

阶段性作业

在上面的例子中,i++ 执行多少次? 循环结束时 i 的值是多少? i<=1000 判断了多少次?

3.2.4 for 循环

一个循环一般有以下要素:

- (1) 控制变量初始化,例如上面的“int i=1;”,表示 i 从 1 开始。
- (2) 循环执行的条件,例如上面的“i<=1000”,表示 i≤1000 才循环。
- (3) 循环运行,控制变量应该变化,例如上面的“i++”,表示每循环一次 i 加 1。

for 循环可以让用户将这 3 个语句写得更加紧凑,格式如下:

```
for (语句 1;语句 2;语句 3){
    循环体;
}
```

在以上结构中先运行初始化语句(语句 1),然后判断条件是否成立(语句 2),如果条件成立,执行循环体,执行完毕后运行语句 3,再判断条件是否成立(语句 2);如果条件成立,执行循环体……周而复始,直到条件不成立为止。

从这里可以看出 for 循环和 while 基本可以互相转换。

例:计算 1~1000 各个整数的和。

代码如下:

ForTest1.java

```
public class ForTest1 {
    public static void main(String[] args) {
        int sum = 0;
        for(int i = 1; i <= 1000; i++){
            sum += i;
        }
        System.out.println("结果为:" + sum);
    }
}
```

运行,控制台打印效果如图 3-8 所示。

结果为:500500

注意

(1) 在 for 循环之后不要直接加分号,这是初学者容易犯的错误:

图 3-8 ForTest1.java 的效果

```
for(int i = 1; i <= 1000; i++){  
    sum += i;  
}
```

//逻辑错误,for 循环运行完毕,没有执行 sum += i

(2) 只要了解了 for 循环中各语句的执行顺序就可以非常灵活地使用 for 循环,例如:

```
for(int i = 1; i <= 1000; sum += i, i++){
```

效果和本例相同。

阶段性作业

在 for 循环中去掉语句 1、语句 2 或者语句 3 会有什么效果? 请测试。

3.2.5 循环嵌套

很显然,程序是丰富多彩的,循环、if 可以嵌套,下面是一个嵌套的例子,用于打印一个九九乘法表:

NineX.java

```
public class NineX {  
    public static void main(String[] args) {  
        for(int r = 1; r <= 9; r++){  
            for(int c = 1; c <= r; c++){  
                System.out.print(r + " * " + c + " = " + r * c + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

//打印不换行
//换行

运行,控制台打印效果如图 3-9 所示。

```
1*1=1  
2*1=2 2*2=4  
3*1=3 3*2=6 3*3=9  
4*1=4 4*2=8 4*3=12 4*4=16  
5*1=5 5*2=10 5*3=15 5*4=20 5*5=25  
6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36  
7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49  
8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64  
9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

图 3-9 NineX.java 的效果

阶段性作业

用 for 循环实现,打印 0~127 各个数字对应的字符,每 8 个打印一行。

3.2.6 break 语句和 continue 语句

1. break 语句

有时候需要在某个时刻终止当前循环,此时可以使用 break 语句。看下面的例子:

BreakTest1.java

```
public class BreakTest1 {  
    public static void main(String[] args) {  
        for(int i = 1; i <= 1000; i++){  
            System.out.println(i);  
            if(i == 2){  
                break;  
            }  
        }  
    }  
}
```

运行,控制台打印效果如图 3-10 所示。

这是因为 i 等于 2 时跳出了 for 循环。



图 3-10 BreakTest1.java 的效果

注意

(1) break 语句还可以跳出 switch 语句。

(2) 在嵌套情况下,break 默认只能跳出当前循环,不能跳出外层循环。例如:

```
for( ... ){  
    for( ... ){  
        break;  
    }  
}
```

该 break 只能跳出内层循环,而不是整个大的循环。如果要解决这一问题,可以利用标号:

```
label:for( ... ){  
    for( ... ){  
        break label;  
    }  
}
```

这样 break 就可以跳出外层循环。

经验

break 和死循环配合使用可以很好地解决“循环次数不确定”的问题。请看下面的例子:输入一个年龄,如果不在 0~100,反复出现输入框,直到输入正确显示该年龄。

代码如下：

BreakTest2.java

```
public class BreakTest2 {
    public static void main(String[] args) {
        while(true){
            String strAge =
                javax.swing.JOptionPane.showInputDialog("输入客户年龄");
            int age = Integer.parseInt(strAge);
            if((age > 0)&&(age <= 100)){
                System.out.println("年龄为：" + age);
                break;
            }
        }
    }
}
```

将输入的工作放入死循环中,只有当输入正确格式时才会跳出死循环。

如果用 for 循环构造死循环,只需要写成“for(;;){循环体;}”即可。

2. continue 语句

和 break 语句相比,continue 语句的使用少一些。continue 语句的作用是跳过当前循环的剩余语句块,接着执行下一次循环。

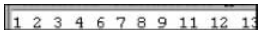
例：打印 1~100 的各个数字,不打印 5 的倍数。

代码如下：

ContinueTest1.java

```
public class ContinueTest1 {
    public static void main(String[] args) {
        for(int i = 1; i <= 100; i++){
            if(i % 5 == 0){
                continue;
            }
            System.out.print(i + " ");
        }
    }
}
```

运行,控制台打印效果如图 3-11 所示。



1 2 3 4 6 7 8 9 11 12 13

图 3-11 ContinueTest1.java 的效果

阶段性作业

(1) 制作一个猜数字游戏：系统随机产生一个 1~100 的整数,要求用户用输入框输入一个整数,如果数字小于随机值,系统提示“小了”;如果数字大于随机值,则提示“大了”;如果猜中,提示“成功”。若 3 次未猜中,则提示“游戏失败”。

(2) 制作一个模拟银行操作的流程。系统运行,出现输入框,让用户选择“0:退出 1:存

款 2:取款 3:查询余额:”。初始余额为 0。

用户选择 1,可以输入钱数,将款项存入余额;用户选择 2,可以输入钱数,将款项从余额中减去,但要保证余额足够;用户选择 3,可以打印当前余额;用户选择 0,程序退出。注意,只要没有退出,用户操作后选择菜单重新显示。

(3) 百鸡问题:公鸡一,值钱 3;母鸡一,值钱 2;小鸡三,值钱 1。今有百鸡百钱,问公鸡、母鸡、小鸡各多少只?

3.3 数 组

3.3.1 为什么需要数组

前面学习了变量,我们知道变量是在内存中存储数据的。如果要定义 100 个整数,保存 100 个用户的年龄,传统方法应该写成:

```
int age1,age2,age3, ...,age100;
```

可见非常麻烦。如果定义的是 1000 个变量,将是几乎无法实现的事情。那么能否一次性定义 100 个变量呢? 数组(Array)可以帮助用户完成。

3.3.2 如何定义数组

这里首先讲解最简单的数组——一维数组。一维数组的定义如下:

数据类型[]数组名 = new 数组类型[数组大小];

例如:

```
int[] age = new int[100];
```

上述语句定义了 100 个 int 变量,变量的名称分别为 age[0]、age[1]、...、age[98]、age[99]。

🔗说明

(1) 变量名是 age[0]~age[99],而不是 age[1]~age[100]。数组被定义之后,数组中的每一个变量叫数组的一个元素。

(2) 数组的大小也可以是整数变量,例如:

```
int size = 100;  
int[] age = new int[size]
```

(3) 数组的定义方法还可以写成:

```
int []age = new int[100];  
和  
int age[] = new int[100];
```

但这种情况使用相对较少。

(4) “`int[] age=new int[100];`”实际上可以看成两句：

```
int[] age
age = new int[100];
```

第一句相当于定义了一个变量名 `age`，是一个数组类型，或者称为数组引用，但是还没有给数组分配内存。第二句给数组分配了 100 个整数大小的内存。如果不分配内存，数组元素将无法访问。

(5) “`int[] age=new int[100];`”定义之后，数组中各元素的默认值为 0。例如：

```
int[] age = new int[100];
System.out.println(age[5]);
```

将打印 0。

(6) 在定义时，也可以给数组进行初始化，有以下两种方法：

```
int[] age1 = new int[] {1,2,3};
```

和

```
int[] age2 = {1,2,3};
```

不管采用哪一种方法，在定义时都不能给数组指定大小，大小由赋值个数决定。

🔗 阶段性作业

定义一个 `char` 数组、`float` 数组、`double` 数组、`boolean` 数组、`String` 数组，不给值，看看里面各个元素的默认值是多少？

3.3.3 如何使用数组

用户可以像使用变量一样来使用数组中的元素。例如：

```
int[] age = new int[100];
age[20] = 25;
System.out.println(age[20]);
```

将会打印 `age[20]` 的值。

为了方便对数组的访问，可以通过“数组名称.length”来获取数组长度。

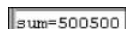
例：将 1~1000 的各个整数放入数组中，然后打印它们的和。

代码如下：

ArrayTest1.java

```
public class ArrayTest1 {  
    public static void main(String[] args) {  
        int[] arr = new int[1000];  
        int sum = 0;  
        for(int i = 0; i < arr.length; i++){  
            arr[i] = i + 1;  
        }  
        for(int i = 0; i < arr.length; i++){  
            sum += arr[i];  
        }  
        System.out.println("sum = " + sum);  
    }  
}
```

运行,控制台打印效果如图 3-12 所示。



☞注意

图 3-12 ArrayTest1.java 的效果

(1) 代码:

```
for(int i = 0; i < arr.length; i++){  
    arr[i] = i + 1;  
}
```

在运行时每次循环都要求 arr.length, 可以对其进行优化:

```
int length = arr.length;  
for(int i = 0; i < length; i++){  
    arr[i] = i + 1;  
}
```

这样, arr.length 就只要求一次了。

(2) 在高版本的 JDK 中, 对数组(乃至集合)进行循环还有一种简化写法:

```
for(数组元素类型 变量名: 数组名称){  
    //使用变量  
}
```

在循环时, 数组中的元素依次放在变量中, 变量类型必须和数组元素类型相同。例如, ArrayTest1 的代码可以改为:

ArrayTest1.java

```
public class ArrayTest1 {  
    public static void main(String[] args) {  
        int[] arr = new int[1000];  
        int sum = 0;  
        for(int i = 0; i < arr.length; i++){  
            arr[i] = i + 1;  
        }  
        for(int e: arr){
```

```
        sum += e;  
    }  
    System.out.println("sum = " + sum);  
}  
}
```

3.3.4 数组的引用性质

简单数据类型变量名表示一个个的内存单元。例如：

```
int a = 5;  
int b = 6;  
a = b;
```

a 和 b 在内存中代表不同的整数空间。如果执行“a=b;”，相当于将变量 b 内存中的值赋给 a。

但是，数组名称赋值却不是将数组中的内容进行赋值，只是将引用赋值。看下面的代码：

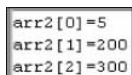
ArrayTest2.java

```
public class ArrayTest2 {  
    public static void main(String[] args) {  
        int[] arr1 = new int[] {1,2,3};  
        int[] arr2 = new int[] {100,200,300};  
        arr1 = arr2;  
        arr1[0] = 5;  
        System.out.println("arr2[0] = " + arr2[0]);  
        System.out.println("arr2[1] = " + arr2[1]);  
        System.out.println("arr2[2] = " + arr2[2]);  
    }  
}
```

运行，控制台打印效果如图 3-13 所示。

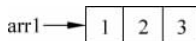
下面分析一下运行过程：

(1) “int[] arr1=new int[] {1,2,3};”表示在内存中开辟一片空间，保存一个数组，如图 3-14 所示。



```
arr2[0]=5  
arr2[1]=200  
arr2[2]=300
```

图 3-13 ArrayTest2.java 的效果

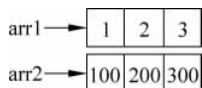


```
arr1 -> [ 1 | 2 | 3 ]
```

图 3-14 保存 arr1

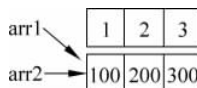
(2) “int[] arr2=new int[] {100,200,300};”表示在内存中开辟一片空间，保存一个数组，如图 3-15 所示。

(3) “arr1=arr2;”表示将 arr2 引用赋值给 arr1，内存变成如图 3-16 所示的状态。



```
arr1 -> [ 1 | 2 | 3 ]  
arr2 -> [ 100 | 200 | 300 ]
```

图 3-15 保存 arr2



```
arr1 -> [ 100 | 200 | 300 ]  
arr2 -> [ 100 | 200 | 300 ]
```

图 3-16 将 arr2 引用赋值给 arr1

此时 arr1 和 arr2 表示同一个数组,因此 arr1[0]变成了 5,arr2[0]也变成了 5。这就是数组的引用性质。

问答

问: arr1 原先指向的{1,2,3}到哪里去了?

答: arr1 原先指向的{1,2,3}在内存中成了“散兵游勇”,最后被当成垃圾搜集。

3.3.5 数组的应用

1. 使用命令行参数

主函数的定义如下:

```
public static void main(String[] args)
```

其中的“String[] args”是什么意思呢?

实际上这是一个数组,表示在运行命令提示符时可以通过命令提示符给主函数一些参数。

如果编写了以下代码:

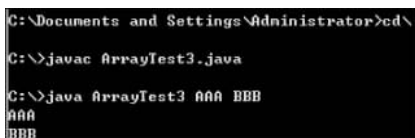
ArrayTest3.java

```
public class ArrayTest3 {  
    public static void main(String[] args) {  
        for(String arg:args){  
            System.out.println(arg);  
        }  
    }  
}
```

在命令行下编译、运行该程序,如图 3-17 所示。

可见,系统将“AAA”和“BBB”放入了 args 数组中。

这有什么用呢?有时候,我们需要让程序运行时还进行一些参数输入,例如编写一个复制文件的类(如 FileCopy),将源文件复制到目的地,运行时就可以写成如图 3-18 所示。



```
C:\Documents and Settings\Administrator>cd\  
C:\>javac ArrayTest3.java  
C:\>java ArrayTest3 AAA BBB  
AAA  
BBB
```

图 3-17 编译、运行程序



```
C:\>java FileCopy file1.txt file2.txt
```

图 3-18 复制文件

系统能根据参数来读写文件,让程序更加灵活。

2. 数组中元素的排序

用户可以用“java.util.Arrays.sort”对数组进行排序(此处只需要了解即可),其代码如下:

ArrayTest4.java

```
public class ArrayTest4 {  
    public static void main(String[] args) {  
        int[] arr = new int[] {5,3,7,2,8,3};  
        java.util.Arrays.sort(arr);  
        for(int a:arr){  
            System.out.print(a + " ");  
        }  
    }  
}
```

```
    }  
}
```

运行,控制台打印效果如图 3-19 所示。

```
2 3 3 5 7 8
```

图 3-19 ArrayTest4.java 的效果

3.3.6 多维数组

以上讲解的是一维数组,是线性的。在 Java 中其实没有多维数组,所谓的多维数组实际上是一维数组中的各个元素又是数组。看下面的代码:

ArrayTest5.java

```
public class ArrayTest5 {  
    public static void main(String[] args) {  
        int[][] arr = new int[][]{{1,2,3},  
                                   {100},  
                                   {15,26}};  
    }  
}
```

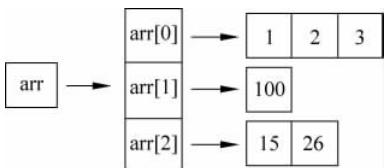


图 3-20 多维数组示例

代码“int[][] arr”实际上定义了一个一维数组 arr,只不过其中的每个元素是“int[]”类型,是一个个小的 一维数组。第一个一维数组名为 arr[0],第二个为 arr[1],第三个为 arr[2]。如果要访问 arr[0]中的第一个元素,当然就是 arr[0][0]了。

示例如图 3-20 所示。

下面的代码是打印各个元素:

ArrayTest5.java

```
public class ArrayTest5 {  
    public static void main(String[] args) {  
        int[][] arr = new int[][]{{1,2,3},  
                                   {100},  
                                   {15,26}};  
        for(int i = 0; i < arr.length; i++){  
            for(int j = 0; j < arr[i].length; j++){  
                System.out.print(arr[i][j] + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

运行,控制台打印效果如图 3-21 所示。

注意

(1) 以上二维数组的定义方法还可以写成:

```
1 2 3  
100  
15 26
```

图 3-21 ArrayTest5.java 的效果

```
int[][] arr = new int[3][];  
arr[0] = new int[]{1,2,3};
```

```
arr[1] = new int[]{100};  
arr[2] = new int[]{15,26};
```

其中的第一句还可以写成“`int [][] arr = new int[3][];`”“`int [] arr[] = new int[3][];`”“`int arr[][] = new int[3][];`”等。

(2) 用户也可以确定二维数组中每个一维数组的大小相同。例如：

```
int [][] arr = new int[3][5];
```

表示定义一个二维数组,其中 3 个一维数组,每个一维数组中含有 5 个元素,实际上就是一个 3 行 5 列的方阵。

🔗 阶段性作业

- (1) 定义一个一维数组,不排序,求出里面所有元素的最大值和最小值。
- (2) 定义一个二维数组,将每一行进行排序,然后输出所有元素。
- (3) 判断一个整型数组中是否存在负数,如果存在,打印相应消息。

本章知识体系

知 识 点	重要等级	难度等级
if 结构	★★★★	★★★
switch 结构	★★	★★
while 结构	★★★★	★★★
do-while 结构	★★★	★★
for 结构	★★★★	★★★
break 和 continue	★★★	★★★
数组的定义	★★★	★★★
数组的性质	★★★	★★
数组的使用	★★★	★★★
二维数组	★★	★★★★★