第3章 运算符、表达式和语句

主要内容：

- 运算符与表达式；
- 语句概述；
- if 条件分支语句；
- switch 开关语句；
- 循环语句；
- break 和 continue 语句；
- 数组与 for 语句；
- 枚举类型与 for、switch 语句。

难点：

- 循环语句；
- 枚举类型与 for、switch 语句。

3.1 运算符与表达式

Java 提供了丰富的运算符，如算术运算符、关系运算符、逻辑运算符和位运算符等，本节将介绍这些运算符。

3.1.1 算术运算符与算术表达式

1. 加、减运算符

加、减运算符 + 、- 是二目运算符，即连接两个操作元的运算符。加、减运算符的结合方向是从左到右。例如 2 + 3 - 8，先计算 2 + 3，然后再将得到的结果减 8。加、减运算符的操作元是整型或浮点型数据，加、减运算符的优先级是 4 级。

2. 乘、除和求余运算符

乘、除和求余运算符 * 、/、% 是二目运算符，结合方向是从左到右。例如 2 * 3/8，先计算 2 * 3，然后再将得到的结果除以 8。乘、除和求余运算符的操作元是整型或浮点型数据，乘、除和求余运算符的优先级是 3 级。

用算术符号和括号连接起来的符合 Java 语法规则的式子称为算术表达式，如 $x + 2 * y - 30 + 3 * (y + 5)$。

3.1.2 自增、自减运算符

自增、自减运算符 ++、-- 是单目运算符，可以放在操作元之前，也可以放在操作元之后。操作元必须是一个整型或浮点型变量，作用是使变量的值增 1 或减 1。例如：

++x(--x)表示在使用 x 之前，先使 x 的值增(减)1。

x++(x--)表示在使用 x 之后，使 x 的值增(减)1。

粗略地看，++x 和 x++ 的作用相当于 $x = x + 1$。但 ++x 和 x ++ 的不同之处在于，++x 是先执行 $x = x + 1$ 再使用 x 的值，而 x++ 是先使用 x 的值再执行 $x = x + 1$。如果 x 的原值是 5，则：

对于“y = ++ x;”，y 的值为 6，

对于“y = x ++ ;”，y 的值为 5。

3.1.3 算术混合运算的精度

精度从“低”到“高”的排列顺序是 byte、short、char、int、long、float、double。

Java 在计算算术表达式的值时，使用下列计算精度规则：

(1) 如果表达式中有双精度浮点数(double 型数据)，则按双精度进行运算。例如，表达式 5.0/2 + 10 的结果 12.5 是 double 型数据。

(2) 如果表达式中最高精度是单精度浮点数(float 型数据)，则按单精度进行运算。例如，表达式 5.0F/2 + 10 的结果 12.5 是 float 型数据。

(3) 如果表达式中最高精度是 long 型整数，则按 long 精度进行运算。例如，表达式 12L + 100 + 'a'的结果 209 是 long 型数据。

(4) 如果表达式中的最高精度低于 int 型整数，则按 int 精度进行运算。例如，表达式 (byte)10 + 'a'和 5/2 的结果分别为 107 和 2，都是 int 型数据。

3.1.4 关系运算符与关系表达式

关系运算符是二目运算符，用来比较两个值的关系。关系运算符的运算结果是 boolean 型，当运算符对应的关系成立时，运算结果是 true，否则是 false。例如，10 < 9 的结果是 false，5 > 1 的结果是 true，3!= 5 的结果是 true，10 > 20 - 17 的结果是 true(因为算术运算符的级别高于关系运算符，10 > 20 - 17 相当于 10 >(20 - 17)，其结果是 true)。

结果为数值型的变量或表达式可以通过关系运算符(如表 3.1 所示)形成关系表达式。例如，4 > 8、($x + y$)> 80 等。

表 3.1 关系运算符

运 算 符	优 先 级	用 法	含 义	结 合 方 向
>	6	op1 > op2	大于	从左到右
<	6	op1 < op2	小于	从左到右
>=	6	op1 >= op2	大于等于	从左到右
<=	6	op1 <= op2	小于等于	从左到右
==	7	op1 == op2	等于	从左到右
!=	7	op1!= op2	不等于	从左到右

3.1.5 逻辑运算符与逻辑表达式

逻辑运算符包括 &&、‖、!。其中,&&、‖ 为二目运算符,实现逻辑与、逻辑或运算;! 为单目运算符,实现逻辑非运算。逻辑运算符的操作元必须是 boolean 型数据,逻辑运算符可以用来连接关系表达式。

表 3.2 给出了逻辑运算符的用法和含义。

表 3.2 逻辑运算符

运 算 符	优 先 级	用 法	含 义	结 合 方 向
&&	11	op1&&op2	逻辑与	从左到右
‖	12	op1‖op2	逻辑或	从左到右
!	2	!op	逻辑非	从右到左

结果为 boolean 型的变量或表达式可以通过逻辑运算符形成逻辑表达式。表 3.3 给出了逻辑运算符的用法和含义。

表 3.3 用逻辑运算符进行逻辑运算

op1	op2	op1&&op2	op1‖op2	!op1
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

例如,2>8&&9>2 的结果为 false,2>8‖9>2 的结果为 true。由于关系运算符的级别高于 &&、‖ 的级别,2>8&&9>2 相当于(2>8)&&(9>2)。

逻辑运算符 && 和 ‖ 也称为短路逻辑运算符,这是因为当 op1 的值是 false 时,&& 运算符在进行运算时不再去计算 op2 的值,直接得出 op1&&op2 的结果是 false;当 op1 的值是 true 时,‖ 运算符在进行运算时不再去计算 op2 的值,直接得出 op1‖op2 的结果是 true。

3.1.6 赋值运算符与赋值表达式

赋值运算符“=”是二目运算符,左面的操作元必须是变量,不能是常量或表达式。设 x 是一个整型变量,y 是一个 boolean 型变量,$x=20$ 和 $y=$ true 都是正确的赋值表达式,赋值运算符的优先级较低,是 14 级,结合方向为从右到左。

赋值表达式的值就是“=”左边变量的值。假如 a、b 是两个 int 型变量,那么表达式 $b=12$ 和 $a=b=100$ 的值分别是 12 和 100。

注意,不要将赋值运算符“=”与等号关系运算符“==”混淆。例如,12=12 是非法的表达式,而表达式 12==12 的值是 true。

3.1.7 位运算符

整型数据在内存中以二进制的形式表示，比如一个 int 型变量在内存中占 4 个字节，共 32 位。int 型数据 7 的二进制表示为：

```
00000000 00000000 00000000 00000111
```

左边最高位是符号位，最高位是 0 表示正数，是 1 表示负数。负数采用补码表示，比如-8 的二进制表示为：

```
111111111 111111111 1111111 11111000
```

这样就可以对两个整型数据实施位运算，即对两个整型数据对应的位进行运算得到一个新的整型数据。

1. “按位与”运算符

“按位与”运算符“&”是双目运算符，用于对两个整型数据 a、b 按位进行运算，运算结果是一个整型数据 c。运算法则是，如果 a、b 两个数据的对应位都是 1，则 c 的该位是 1，否则是 0。如果 b 的精度高于 a，那么结果 c 的精度和 b 相同。

例如：

```
     a:  00000000  00000000  00000000  00000111
&    b:  10000001  10100101  11110011  10101011
     c:  00000000  00000000  00000000  00000011
```

2. “按位或”运算符

“按位或”运算符“|”是二目运算符，用于对两个整型数据 a、b 按位进行运算，运算结果是一个整型数据 c。运算法则是，如果 a、b 两个数据的对应位都是 0，则 c 的该位是 0，否则是 1。如果 b 的精度高于 a，那么结果 c 的精度和 b 相同。

3. “按位非”运算符

“按位非”运算符“~”是单目运算符，用于对一个整型数据 a 按位进行运算，运算结果是一个整型数据 c。运算法则是，如果 a 的对应位是 0，则 c 的该位是 1，否则是 0。

4. “按位异或”运算符

“按位异或”运算符“^”是二目运算符，用于对两个整型数据 a、b 按位进行运算，运算结果是一个整型数据 c。运算法则是，如果 a、b 两个数据的对应位相同，则 c 的该位是 0，否则是 1。如果 b 的精度高于 a，那么结果 c 的精度和 b 相同。

由异或运算法则可知：

```
a^a=0
a^0=a
```

因此，如果 $c = a \wedge b$，那么 $a = c \wedge b$，也就是说，“^”的逆运算仍然是“^”，即 $a \wedge b \wedge b$ 等于 a。

使用位运算符也可以操作逻辑型数据，法则如下：

(1) 当 a、b 都是 true 时，$a\&b$ 是 true，否则 $a\&b$ 是 false。

(2) 当 a、b 都是 false 时，$a|b$ 是 false，否则 $a|b$ 是 true。

(3) 当 a 是 true 时,$\sim a$ 是 false;当 a 是 false 时,$\sim a$ 是 true。

位运算符在操作逻辑型数据时,与逻辑运算符 &&、‖、! 不同的是,位运算符要在计算完 a 和 b 之后再给出运算的结果。比如,x 的初值是 1,那么经过下列逻辑比较:

```
((y = 1) == 0))&&((x = 6) == 6));
```

运算后,x 的值仍然是 1。但是,如果经过下列位运算:

```
((y = 1) == 0))&((x = 6) == 6));
```

之后,x 的值将是 6。

在下面的例 3.1 中,利用“异或”运算的性质,对几个字符进行加密并输出密文,然后再解密,运行效果如图 3.1 所示。

```
C:\chapter3>java Example3_1
密文:昌嘈襖燮
原文:中国科大
```

图 3.1 异或运算

例 3.1

Example3_1.java

```
public class Example3_1 {
    public static void main(String args[]) {
        char a1 = '中',a2 = '国',a3 = '科',a4 = '大';
        char secret = 'A';
        a1 = (char)(a1 ^secret);
        a2 = (char)(a2 ^secret);
        a3 = (char)(a3 ^secret);
        a4 = (char)(a4 ^secret);
        System.out.println("密文:" + a1 + a2 + a3 + a4);
        a1 = (char)(a1 ^secret);
        a2 = (char)(a2 ^secret);
        a3 = (char)(a3 ^secret);
        a4 = (char)(a4 ^secret);
        System.out.println("原文:" + a1 + a2 + a3 + a4);
    }
}
```

3.1.8 instanceof 运算符

instanceof 运算符是二目运算符,左边的操作元是一个对象,右边是一个类。当左边的对象是右边的类或子类创建的对象时,该运算符运算的结果是 true,否则是 false。

3.1.9 运算符综述

Java 的表达式就是用运算符连接起来的符合 Java 规则的式子,运算符的优先级决定了表达式中运算执行的先后顺序。例如,$x<y\&\&!z$ 相当于$(x<y)\&\&(!z)$。用户没有必要去记忆运算符的优先级别,在编写程序时尽量使用()运算符来实现想要的运算次序,以免产生难以阅读或含糊不清的计算顺序。运算符的结合性决定了并列的相同级别运算符的先后顺序,例如,加、减的结合性是从左到右,8-5+3 相当于(8-5)+3;逻辑运算符!的结合性是从右到左,!!x 相当于!(!x)。表 3.4 是 Java 中所有运算符的优先级和结合性,有些运算符和 C 语言中相同,在此不再赘述。

表 3.4　运算符的优先级和结合性

优　先　级	描　　述	运　算　符	结　合　性
1	分隔符	[ ]、( )、.、,、;	
2	对象归类,自增、自减运算,逻辑非	instanceof、++、--、!	从右到左
3	算术乘、除运算	*、/、%	从左到右
4	算术加、减运算	+、-	从左到右
5	移位运算	>>、<<、>>>	从左到右
6	大小关系运算	<、<=、>、>=	从左到右
7	相等关系运算	==、!=	从左到右
8	按位与运算	&	从左到右
9	按位异或运算	^	从左到右
10	按位或运算	|	从左到右
11	逻辑与运算	&&	从左到右
12	逻辑或运算	||	从左到右
13	三目条件运算	?、:	从左到右
14	赋值运算	=	从右到左

3.2　语 句 概 述

Java 中的语句可分为以下 6 类。

(1) 方法调用语句。例如:

```
System.out.println(" Hello");
```

(2) 表达式语句。表达式语句指由一个表达式构成一个语句,即在表达式尾加上分号。例如赋值语句:

```
x = 23;
```

(3) 复合语句。在 Java 中,可以用{ }把一些语句括起来构成复合语句,例如:

```
{   z = 123 + x;
    System.out.println("How are you");
}
```

(4) 空语句。一个分号也是一条语句,称为空语句。

(5) 控制语句。控制语句分为条件分支语句、开关语句和循环语句 3 种类型,将在后面的 3.3 节、3.4 节和 3.5 节进行介绍。

(6) package 语句和 import 语句。package 语句和 import 语句和类、对象有关,将在第 4 章讲解。

3.3　条件分支语句

条件分支语句按照语法格式可细分为 3 种形式,以下是这 3 种形式的详细讲解。

3.3.1　if 语句

if 语句是单条件分支语句,即根据一个条件来控制程序执行的流程。

if 语句的语法格式如下：

```
if(表达式){
    若干语句
}
```

if 语句的流程图如图 3.2 所示。在 if 语句中，关键字 if 后面的一对小括号()内的表达式的值必须是 boolean 类型，当值为 true 时，执行紧跟着的复合语句，结束当前 if 语句的执行；如果表达式的值为 false，结束当前 if 语句的执行。

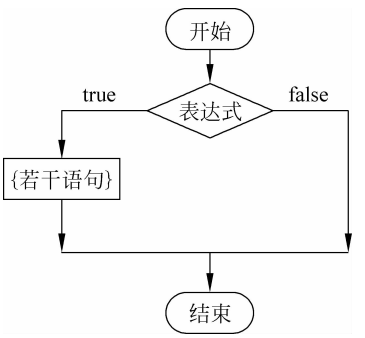

图 3.2　if 条件语句

需要注意的是，在 if 语句中，如果复合语句中只有一条语句，{}可以省略不写，但为了增强程序的可读性，最好不要省略(这是一个很好的编程习惯)。

在下面的例 3.2 中，将变量 a、b、c 中的数值按大小顺序进行互换(从小到大排列)。

例 3.2

Example3_2.java

```
public class Example3_2 {
    public static void main(String args[]) {
        int a = 9,b = 5,c = 7,t = 0;
        if(b<a) {
           t = a;
           a = b;
           b = t;
        }
        if(c<a) {
           t = a;
           a = c;
           c = t;
        }
        if(c<b) {
           t = b;
           b = c;
           c = t;
        }
        System.out.println("a = " + a + ",b = " + b + ",c = " + c);
    }
}
```

3.3.2　if-else 语句

if-else 语句是单条件分支语句，即根据一个条件来控制程序执行的流程。

if-else 语句的语法格式如下：

```
if(表达式) {
    若干语句
  }
  else {
    若干语句
  }
```

if-else 语句的流程图如图 3.3 所示。在 if-else 语句中，关键字 if 后面的一对小括号() 内的表达式的值必须是 boolean 类型，当值为 true 时，执行紧跟着的复合语句，结束当前 if-else 语句的执行；如果表达式的值为 false，则执行关键字 else 后面的复合语句，结束当前 if-else 语句的执行。

下面是有语法错误的 if-else 语句：

```
if(x>0)
   y=10;
   z=20;
else
   y=-100;
```

正确的写法是：

```
if(x>0){
   y=10;
   z=20;
}
else
   y=100;
```

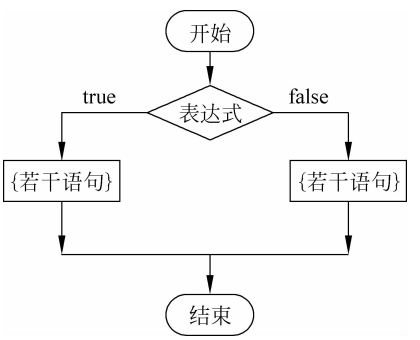

图 3.3　if-else 条件语句

需要注意的是，在 if-else 语句中，如果复合语句中只有一条语句，{}可以省略不写，但为了增强程序的可读性，最好不要省略(这是一个很好的编程习惯)。

下面的例 3.3 中有两条 if-else 语句，其作用是根据成绩输出相应的信息，运行效果如图 3.4 所示。

例 3.3

Example3_3.java

```
public class Example3_3 {
    public static void main(String args[]) {
        int math=65 ,english=85;
        if(math>60) {
            System.out.println("数学及格了");
        }
        else {
            System.out.println("数学不及格");
        }
        if(english>90) {
            System.out.println("英语是优");
        }
```

```
C:\chapter3>java Example3_3
数学及格了
英语不是优
我在学习if-else语句
```

图 3.4　使用 if-else 语句

```
            else {
                System.out.println("英语不是优");
            }
            System.out.println("我在学习 if-else 语句");
        }
    }
```

3.3.3 if-else if-else 语句

if-else if-else 语句是多条件分支语句,即根据多个条件来控制程序执行的流程。

if-else if-else 语句的语法格式如下:

```
if(表达式) {
    若干语句
}
else if(表达式) {
    若干语句
  }
  ⋮
  else {
    若干语句
  }
```

if-else if-else 语句的流程图如图 3.5 所示。在 if-else if-else 语句中,if 以及多个 else if 后面的一对小括号()内的表达式的值必须是 boolean 类型。程序在执行 if-else if-else 语句时,按照该语句中表达式的顺序,首先计算第 1 个表达式的值,如果计算结果为 true,则执行紧跟着的复合语句,结束当前 if-else if-else 语句的执行;如果计算结果为 false,则继续计算第 2 个表达式的值;依此类推,假设计算第 m 个表达式的值为 true,则执行紧跟着的复合语句,结束当前 if-else if-else 语句的执行,否则继续计算第 $m+1$ 个表达式的值。如果所有表达式的值都为 false,则执行关键字 else 后面的复合语句,结束当前 if-else if-else 语句的执行。

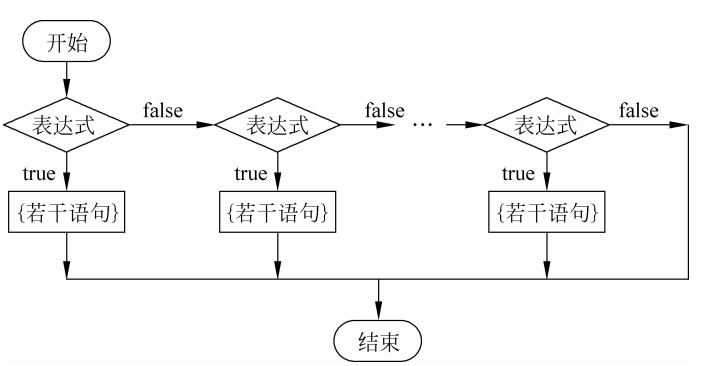

图 3.5 if-else if-else 多条件语句

if-else if-else 语句中的 else 部分是可选项,如果没有 else 部分,当所有表达式的值都为 false 时,结束当前 if-else if-else 语句的执行(该语句什么都没有做)。

需要注意的是，在 if-else if-else 语句中，如果复合语句中只有一条语句，{}可以省略不写，但为了增强程序的可读性，最好不要省略。

3.4 开关语句

switch 语句是单条件多分支的开关语句，它的一般格式如下（其中，break 语句是可选的）：

```
switch(表达式)
{
    case 常量值 1:
              若干语句
              break;
    case 常量值 2:
              若干语句
              break;
    ⋮
    case 常量值 n:
              若干语句
              break;
    default:
          若干语句
}
```

在 switch 语句中，“表达式”的值必须为 byte、short、int、char 型或枚举类型（枚举类型见第 2 章的 2.5 节）；“常量值 1”到“常量值 n”也必须是 byte、short、int、char 型或枚举类型常量，而且互不相同。

switch 语句首先计算表达式的值，如果表达式的值和某个 case 后面的常量值相等，就执行该 case 中的若干个语句，直到碰到 break 语句为止。如果某个 case 中没有使用 break 语句，一旦表达式的值和该 case 后面的常量值相等，程序不仅执行该 case 中的若干个语句，而且继续执行后继的 case 中的若干个语句，直到碰到 break 语句为止。若 switch 语句中的表达式的值不与任何 case 的常量值相等，则执行 default 后面的若干个语句。switch 语句中的 default 是可选的，如果它不存在，并且 switch 语句中表达式的值不与任何 case 的常量值相等，那么 switch 语句就不会进行任何处理。

```
C:\chapter3>java Example3_4
97
97
春天种下种子
秋天收获果实
```

图 3.6 使用 switch 语句

下面的例 3.4 使用了 switch 语句，运行效果如图 3.6 所示。

例 3.4

Example3_4.java

```
enum Season {
    春季,夏季,秋季,冬季
}
public class Example3_4 {
    public static void main(String args[]) {
```

```
        int x = 96,y = 1;
        Season season = Season.春季;
        switch(x + y) {
            case 1 :
              System.out.println(x + y);
              break;
            case 'a':
              System.out.println(x + y);
            case 10:
              System.out.println(x + y);
              break;
            default: System.out.println("没有般配的" + (x + y));
        }
        switch(season) {
            case 冬季:
                System.out.println("隆冬腊月");
            case 春季:
                System.out.println("春天种下种子");
            case 秋季:
                System.out.println("秋天收获果实");
                break;
            case 夏季:
                System.out.println("暑假真好");
        }
    }
}
```

需要强调的是,switch 语句中表达式的值必须是 byte、short、int、char 型或枚举类型。如果将例 3.4 中的

```
int x = 96,y = 1;
```

更改为

```
long x = 96,y = 1;
```

编译时将出现错误。

3.5 循 环 语 句

循环语句是根据条件,要求程序反复执行某些操作,直到程序“满意”为止。

3.5.1 for 循环语句

for 语句的语法格式如下:

```
for (表达式 1; 表达式 2; 表达式 3) {
      若干语句
}
```

for 语句由关键字 for、一对小括号()中用分号分隔的 3 个表达式以及一个复合语句组成，其中的“表达式 2”必须是一个求值为 boolean 型数据的表达式，复合语句称为循环体。当循环体只有一条语句时，大括号“{}”可以省略，但最好不要省略，以便增加程序的可读性。“表达式 1”负责完成变量的初始化；“表达式 2”是值为 boolean 型的表达式，称为循环条件；“表达式 3”用来修整变量，改变循环条件。for 语句的执行规则如下：

(1) 计算“表达式 1”，完成必要的初始化工作。

(2) 判断“表达式 2”的值，若“表达式 2”的值为 true，则进行第(3)步，否则进行第(4)步。

(3) 执行循环体，然后计算“表达式 3”，以便改变循环条件，进行第(2)步。

(4) 结束 for 语句的执行。

for 语句的执行流程如图 3.7 所示。

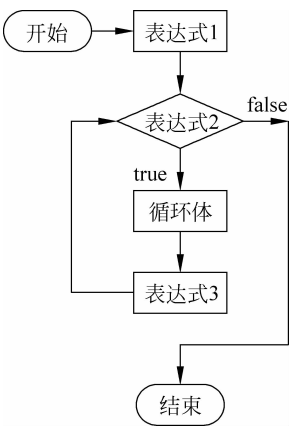

图 3.7　for 循环语句

下面的例 3.5 计算 8 + 88 + 888 + 8888 + …的前 12 项的和。

例 3.5

Example3_5.java

```
public class Example3_5 {
    public static void main(String args[]) {
        long sum = 0,a = 8,item = a,n = 12,i = 1;
        for(i = 1;i <= n;i++ ) {
              sum = sum + item;
              item = item * 10 + a;
        }
        System.out.println(sum);
    }
}
```

3.5.2　while 循环语句

while 语句的语法格式如下：

```
while (表达式) {
      若干语句
}
```

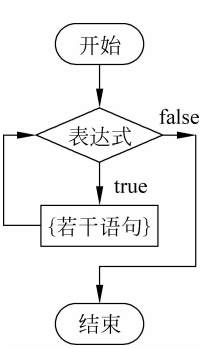

图 3.8　while 循环语句

while 语句由关键字 while、一对括号()中的一个求值为 boolean 类型数据的表达式和一个复合语句组成，其中的复合语句称为循环体。当循环体只有一条语句时，大括号{}可以省略，但最好不要省略，以便增加程序的可读性。另外，在该句中，表达式称为循环条件。while 语句的执行规则如下：

(1) 计算表达式的值，如果该值是 true，进行第(2)步，否则执行第(3)步。

(2) 执行循环体，再进行第(1)步。

(3) 结束 while 语句的执行。

while 语句的执行流程如图 3.8 所示。

3.5.3 do-while 循环语句

do-while 循环语句的语法格式如下：

```
do {
    若干语句
} while(表达式);
```

do-while 循环和 while 循环的区别是，do-while 的循环体至少被执行一次，执行流程如图 3.9 所示。

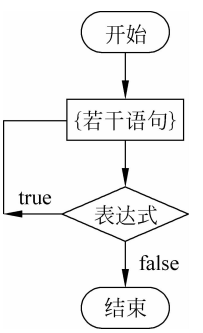

图 3.9 do-while 循环语句

下面的例 3.6 用 while 语句计算 1 + 1/2! + 1/3! + 1/4! + …的前 20 项，并使用数组输出 Fibinacci 数列的前 12 项。Fibinacci 数列的前 2 项为 1，从第 3 项开始，每项是它的前两项之和。

例 3.6

Example3_6.java

```
public class Example3_6 {
    public static void main(String args[]) {
        double sum = 0,item = 1;
        int i = 1,n = 20;
        while(i <= n) {
            sum = sum + item;
            i = i + 1;
            item = item * (1.0/i);
        }
        System.out.println("sum = " + sum);
    }
}
```

3.6 break 和 continue 语句

break 和 continue 语句是用关键字 break 或 continue 加上分号构成的语句，例如：

```
break;
```

在循环体中可以使用 break 语句和 continue 语句。在一个循环中，例如循环 50 次的循环语句中，如果在某次循环中执行了 break 语句，那么整个循环语句就结束。如果在某次循环中执行了 continue 语句，那么本次循环就结束，即不再执行本次循环中循环体中的 continue 语句后面的语句，而转入进行下一次循环。

下面的例 3.7 使用了 break 和 continue 语句。

例 3.7

Example3_7.java

```
public class Example3_7 {
    public static void main(String args[]) {
```

```
        int sum = 0,i,j;
        for( i = 1;i <= 10;i++ ) {
            if(i % 2 == 0) {                    //计算 1 + 3 + 5 + 7 + 9
                continue;
            }
            sum = sum + i;
        }
        System.out.println("sum = " + sum);
        for(j = 2;j <= 50;j++ ) {               //求 50 以内的素数
            for( i = 2;i <= j/2;i++ ) {
                if(j % i == 0)
                    break;
            }
            if(i > j/2) {
                System.out.println("" + j + "是素数");
            }
        }
    }
}
```

3.7 数组与 for 语句

JDK 1.5 版本对 for 语句的功能给予扩充、增强，以便更好地遍历数组。其语法格式如下：

```
for(声明循环变量: 数组的名字) {
    ⋮
}
```

其中，声明的循环变量的类型必须和数组的类型相同。

这种形式的 for 语句类似自然语言中的 for each 语句，为了便于理解上述 for 语句，可以将这种形式的 for 语句翻译成“对于循环变量依次取数组的每一个元素的值”。

下面的例 3.8 分别使用 for 语句的传统方式和改进方式遍历数组。

例 3.8

Example3_8.java

```
public class Example3_8 {
    public static void main(String args[]) {
      int a[] = {1,2,3,4};
      char b[] = {'a','b','c','d'};
      for(int n = 0;n < a.length;n++ ) {     //传统方式
          System.out.println(a[n]);
      }
      for(int n = 0;n < b.length;n++ ) {     //传统方式
          System.out.println(b[n]);
      }
```

```
        for(int i:a) {                  //循环变量 i 依次取数组 a 的每一个元素的值(改进方式)
            System.out.println(i);
        }
        for(char ch:b) {                //循环变量 ch 依次取数组 b 的每一个元素的值(改进方式)
            System.out.println(ch);
        }
    }
}
```

需要特别注意的是,

```
for(声明循环变量: 数组的名字)
```

中的“声明循环变量”必须是变量声明,不可以使用已经声明过的变量。例如,上述例 3.8 中的第一个改进方式 for 语句不可以分开写成如下两条语句:

```
int i = 0;
for(i:a) {
    System.out.println(i);
}
```

3.8 枚举类型与 for、switch 语句

在第 2 章学习了枚举类型,例如:

```
enum WeekDay {
    sun,mon,tue,wed,thu,fri,sat
}
```

声明了名字为 WeekDay 的枚举类型,该枚举类型有 7 个常量。

在声明了一个枚举类型后,就可以用该枚举类型声明一个枚举变量了。该枚举变量只能取枚举类型中的常量,通过使用枚举名和“.”运算符获得枚举类型中的常量。例如:

```
WeekDay day = WeekDay.mon;
```

枚举类型可以用如下形式:

```
枚举类型的名字.values();
```

返回一个数组,该数组元素的值和该枚举类型中的常量依次对应。例如:

```
WeekDay a[ ] = WeekDay.values();
```

那么,a[0]~a[6]的值依次为 sun、mon、tue、wed、thu、fri 和 sat。

JDK 1.5 之后的版本可以使用 for 语句遍历枚举类型中的常量。在下面的例 3.9 中,输出从红、蓝、绿、黄、黑颜色中取出 3 种不同颜色的排列(不是组合),运行效果如图 3.10 所示。

```
C:\chapter3>java Example3_9
红,蓝,绿 |红,蓝,黄 |红,蓝,黑 |红,绿,蓝 |红,绿,黄 |红,绿,黑 |红,黄,蓝 |红,黄,绿 |
红,黄,黑 |红,黑,蓝 |红,黑,绿 |红,黑,黄 |蓝,红,绿 |蓝,红,黄 |蓝,红,黑 |蓝,绿,红 |
蓝,绿,黄 |蓝,绿,黑 |蓝,黄,红 |蓝,黄,绿 |蓝,黄,黑 |蓝,黑,红 |蓝,黑,绿 |蓝,黑,黄 |
绿,红,蓝 |绿,红,黄 |绿,红,黑 |绿,蓝,红 |绿,蓝,黄 |绿,蓝,黑 |绿,黄,红 |绿,黄,蓝 |
绿,黄,黑 |绿,黑,红 |绿,黑,蓝 |绿,黑,黄 |黄,红,蓝 |黄,红,绿 |黄,红,黑 |黄,蓝,红 |
黄,蓝,绿 |黄,蓝,黑 |黄,绿,红 |黄,绿,蓝 |黄,绿,黑 |黄,黑,红 |黄,黑,蓝 |黄,黑,绿 |
黑,红,蓝 |黑,红,绿 |黑,红,黄 |黑,蓝,红 |黑,蓝,绿 |黑,蓝,黄 |黑,绿,红 |黑,绿,蓝 |
黑,绿,黄 |黑,黄,红 |黑,黄,蓝 |黑,黄,绿 |
```

图 3.10　排列 3 种颜色

例 3.9

Example3_9.java

```
enum Color {
   红,蓝,绿,黄,黑
}
public class Example3_9 {
   public static void main(String args[]) {
      for(Color a:Color.values()) {
         for(Color b:Color.values()) {
            for(Color c:Color.values()) {
               if(a!= b&&a!= c&&b!= c) {
                  System.out.print(a + "," + b + "," + c + " |");
               }
            }
         }
      }
   }
}
```

JDK 1.5 以后的版本允许 switch 语句中表达式的值是枚举类型(见 3.4 节)。下面的例 3.10 结合 for 语句和 switch 语句显示了 5 种水果中部分水果的价格,其中,for 语句和 switch 语句都使用了枚举类型。运行效果如图 3.11 所示。

```
C:\chapter3>java Example3_10
苹果500克的价格：1.5元
香蕉500克的价格：2.8元
芒果500克的价格：6.8元
```

图 3.11　显示部分水果的价格

例 3.10

Example3_10.java

```
enum Fruit {
  苹果,梨,香蕉,西瓜,芒果
}
public class Example3_10 {
  public static void main(String args[]) {
      double price = 0;
      boolean show = false;
      for(Fruit fruit:Fruit.values()) {
          switch(fruit) {
              case 苹果: price = 1.5;
                         show = true;
                         break;
```

```
                case 芒果: price = 6.8;
                           show = true;
                           break;
                case 香蕉: price = 2.8;
                           show = true;
                           break;
                default: show = false;
            }
            if(show) {
                System.out.println(fruit + "500 克的价格: " + price + "元");
            }
        }
    }
}
```

3.9 小　　结

1. Java 提供了丰富的运算符,如算术运算符、关系运算符、逻辑运算符和位运算符等。
2. Java 语言常用的控制语句和 C 语言的非常类似。
3. Java 改进了对数组的循环,这是 Java 独有的特色。

习　题　3

1. 下列程序的输出结果是什么? if-else 语句的书写是否规范?

```
public class E {
    public static void main (String args[]) {
        int x = 10, y = 5, z = 100, result = 0;
        if(x > y)
            x = z;
        else
            y = x;
        z = y;
        result = x + y + z;
        System.out.println(result);
    }
}
```

2. 下列程序的输出结果是什么?

```
public class E {
   public static void main (String args[]) {
      char c = '\0';
      for(int i = 1;i <= 4;i++ ) {
         switch(i) {
            case 1:  c = '你';
                     System.out.print(c);
            case 2:  c = '好';
```

```
                    System.out.print(c);
                    break;
                case 3:  c = '酷';
                    System.out.print(c);
                default: System.out.print("!");
            }
        }
    }
}
```

3. 编写一个应用程序，用 for 循环输出俄文的“字母表”。

4. 编写一个应用程序，求 1! + 2! + … + 20! 。

5. 编写一个应用程序，求 100 以内的全部素数。

6. 分别用 while 和 for 循环计算 1 + 1/2! + 1/3! + 1/4! + …的前 20 项之和。

7. 一个数如果恰好等于它的因子之和，这个数就称为“完数”。编写一个应用程序，求 1000 之内的所有完数。

8. 编写应用程序，计算两个非零正整数的最大公约数和最小公倍数，要求两个非零正整数从键盘输入。

9. 求满足 1 + 2! + 3! + … + n! ≤9999 的最大整数 n。