第 3 章 运算符、表达式和语句

扫一扫

视频讲解

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。
- for 语句与数组。

扫一扫

视频讲解

3.1 运算符与表达式

Java 提供了丰富的运算符，例如算术运算符、关系运算符、逻辑运算符、位运算符等。Java 语言中的绝大多数运算符和 C 语言相同，基本语句（例如条件分支语句、循环语句等）也和 C 语言类似，因此本章对主要知识点进行简单的介绍。

3.1.1 算术运算符与算术表达式

❶ 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，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。

❷ 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，再将得到的结果除以 8。乘、除和求余运算符的操作元素是整型或浮点型数据，乘、除和求余运算符的优先级是 3 级。

用算术运算符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。例如 x * 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，可以放在操作元素之前，也可以放在操作元素之后。操作元素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，假如 x 是 byte 型，＋＋x 和 x＋＋的作用相当于 x＝(byte)(x＋1)。＋＋x 和 x＋＋的不同之处在于，＋＋x 是先执行 x＝(byte)(x＋1)再使用 x 的值，而 x＋＋是先使用 x

的值再执行 x=(byte)(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”排列的顺序是 byte→short→char→int→long→float→double。

在 Java 中计算算术表达式的值时使用下列精度运算规则。

(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) 如果表达式中的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) 如果表达式中的最高精度是 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。

Java 允许把不超出 byte、short 和 char 的取值范围的常量算术表达式的值赋给 byte、short 和 char 型变量。例如,(byte)30+'a'是结果为 127 的 int 型常量。

```
byte x = (byte)20 + 'a';
```

是正确的,但

```
byte x = (byte)30 + 'b';
```

无法通过编译,编译错误是“可能损失精度,找到 int 需要 byte”,原因是(byte)30+'b'的结果是 int 型常量,其值超出了 byte 变量的取值范围(见上面精度运算规划的第 4 条)。

需要特别注意的是,当赋值号右边的表达式中有变量时,编译只检查变量的类型,不检查变量中的值。例如,“byte x=97+1;”和“byte y=1;”都是正确的,但是“byte z=97+y;”是错误的,其原因是编译器不检查表达式 97+y 中变量 y 的值,只检查 y 的类型,并认为表达式 97+y 的结果是 int 型精度(见上面精度运算规则的第 4 条)。所以,对于“byte z=97+y;”,编译器会提示“不兼容的类型:从 int 型转换为 byte 型可能会有损失”的信息。

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

续表

运 算 符	优 先 级	用 法	含 义	结 合 方 向
>=	6	op1 >= op2	大于或等于	从左到右
<=	6	op1 <= op2	小于或等于	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;!为单目运算符,实现逻辑非。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 给出了用逻辑运算符进行逻辑运算的结果。

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&&8 > 2 相当于(2 > 8)&&(9 > 2)。

逻辑运算符 && 和||也称作短路逻辑运算符,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,直接就得出 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,直接就得出 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,结合方向是从右到左。

赋值表达式的值就是=左面变量的值。例如,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。

需要注意的是，对于+=、*=、/=、--缩略运算符，编译器自动将赋值符号右侧的表达式的值转换为左边变量所要求的类型。例如，b+=120 等同于 b=(byte)(b+120)。

3.1.7　位运算符

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

```
00000000 00000000 00000000 00000111
```

左面最高位是符号位，最高位是 0 表示正数，是 1 表示负数。负数采用补码表示，例如，-8 的补码表示是：

```
111111111 111111111 1111111 11111000
```

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

❶ 按位与运算

按位与运算符 & 是双目运算符，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 两个数据对应的位都是 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
```

❷ 按位或运算

按位或运算符|是二目运算符，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 两个数据对应的位都是 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。

❸ 按位非运算

按位非运算符~是单目运算符，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 对应的位是 0，则 c 的该位是 1，否则是 0。

❹ 按位异或运算

按位异或运算符^是二目运算符，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 两个数据对应的位相同，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^b，那么 a=c^b。也就是说，^的逆运算仍然是^，即 a^b^b 等于 a。

位运算符也可以操作逻辑型数据，法则是：

当 a、b 都是 true 时，a&b 是 true，否则 a&b 是 false。

当 a、b 都是 false 时，a|b 是 false，否则 a|b 是 true。

当 a 是 true 时，~a 是 false；当 a 是 false 时，~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 的值之后再给出运算的结果。例如，x 的初值是 1，那么经过下列逻辑比较运算后

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

x 的值仍然是 1，但是经过下列位运算之后

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

x 的值将是 6。

在下面的例子 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。

图 3.1　异或运算

例子 1

Example3_1.java

```
public class Example3_1 {
    public static void main(String args[]) {
        char a1 = '十',a2 = '点',a3 = '进',a4 = '攻';
        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，左面的操作元是一个对象，右面是一个类。当左面的对象是右面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(有关细节详见 5.3.2 节)。

3.1.9　运算符综述

Java 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)。大家没有必要去记忆运算符的优先级别，在编写程序时尽量地使用括号运算符()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；逻辑否运算符!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，这里不再赘述。

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	对象的归类，自增和自减运算，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	?:	从右到左
14	赋值运算	=	从右到左

扫一扫

视频讲解

3.2　语句概述

Java 中的语句可分为以下 6 类。

❶ 方法调用语句

例如：

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

❷ 表达式语句

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;
```

❸ 复合语句

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");
}
```

❹ 空语句

一个分号也是一个语句，称作空语句。

❺ 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，将在 3.3 节～3.5 节介绍。

❻ package 语句和 import 语句

package 语句和 import 语句与类、对象有关，将在第 4 章讲解。

扫一扫

视频讲解

3.3 if 条件分支语句

条件分支语句按语法格式可细分为 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、单分支语句,即根据一个条件来控制程序执行的流程。

if 语句的语法格式为:

```
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。

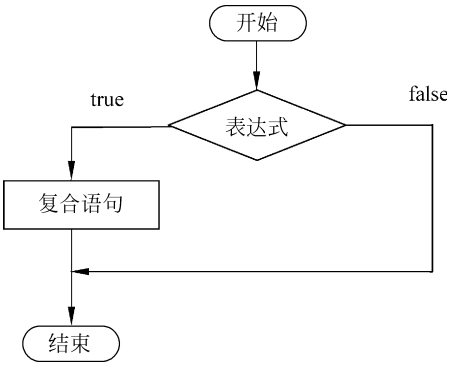

图 3.2 if 单条件、单分支语句流程

需要注意的是,在 if 语句中,其中的复合语句如果只有一个语句,{}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)。

在下面的例子 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)。

例子 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、双分支语句，即根据一个条件来控制程序执行的流程。

if-else 语句的语法格式为：

```
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。

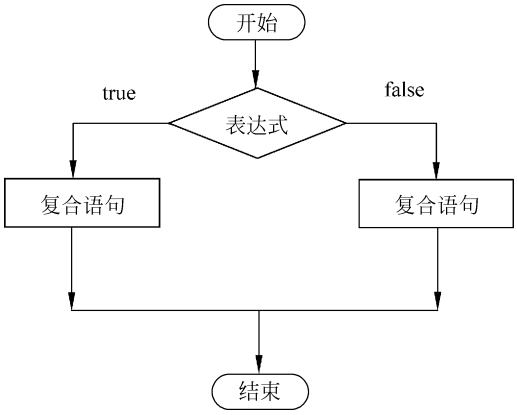

图 3.3　if-else 单条件、双分支语句流程

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;
```

需要注意的是，在 if-else 语句中，复合语句如果只有一个语句，{}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)。

在例子 3 中有两个 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。

```
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

例子 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("英语是优秀");
        }
        else {
            System.out.println("英语不是优秀");
        }
        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、多分支语句,即根据多个条件来控制程序执行的流程。

if-else if-else 语句的语法格式为:

```
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 型。程序在执行 if-else if-else 时,按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,以此类推,假设计算出第 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。

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

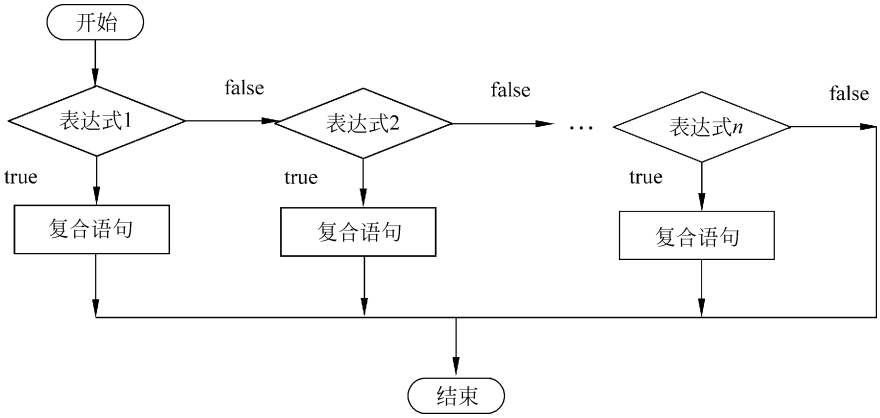

图 3.5　if-else if-else 多条件、多分支语句流程

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，复合语句如果只有一个语句，{}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　switch 开关语句

扫一扫

视频讲解

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”的值可以为 byte、short、int、char 和 String 型(见 8.1 节)；“常量值 1”到“常量值 n”也是 byte、short、int、char 和 String 型，而且要互不相同。

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 中的若干语句，而且继续执行后面的 case 中的若干语句，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。

前面学习的分支语句(if 语句、if-else 语句和 if-else if-else 语句)的共同特点是根据一个条件选择执行一个分支操作,而不是选择执行多个分支操作。在 switch 语句中,通过合理地使用 break 语句,可以达到根据一个条件选择执行一个分支操作(一个 case)或多个分支操作(多个 case)的结果。

下面的例子 4 使用了 switch 语句判断用户从键盘输入的正整数是否为中奖号码。

例子 4

Example3_4.java

```
import java.util.Scanner;
public class Example3_4{
   public static void main(String args[]) {
      int number = 0;
      System.out.println("输入正整数(回车确定)");
      Scanner reader = new Scanner(System.in);
      number = reader.nextInt();
      switch(number) {
        case 9:
        case 131:
        case 12:    System.out.println(number + "是三等奖");
                    break;
        case 209:
        case 596:
        case 27:   System.out.println(number + "是二等奖");
                   break;
        case 875:
        case 316:
        case 59:   System.out.println(number + "是一等奖");
                   break;
        default:   System.out.println(number + "未中奖");
      }
   }
}
```

需要强调的是,switch 语句中表达式的值可以是 byte、short、int、char 型,但不可以是 long 型数据。如果将例子 4 中的

```
int number = 0;
```

更改为

```
long number = 0;
```

将导致编译错误。

扫一扫

视频讲解

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 语句的语法格式为：

```
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 必须是一个值为 boolean 型数据的表达式，而复合语句称作循环体。当循环体中只有一个语句时，大括号{}可以省略，但最好不要省略，以增加程序的可读性。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(3)，否则进行(4)。

(3) 执行循环体，然后计算表达式 3，以便改变循环条件，进行(2)。

(4) 结束 for 语句的执行。

for 语句的执行流程如图 3.6 所示。

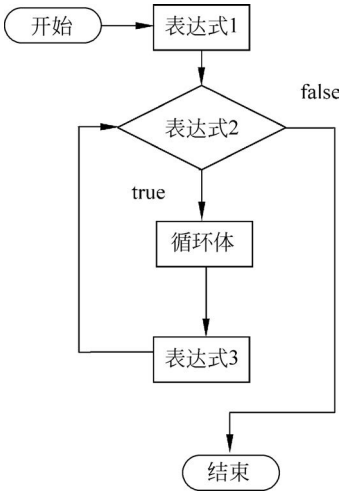

图 3.6　for 循环语句流程

下面的例子 5 计算 8＋88＋888＋8888＋…的前 12 项之和。

例子 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 语句的语法格式为：

```
while(表达式) {
    若干语句
}
```

while 语句由关键字 while、一对小括号()中的一个值为 boolean 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。当循环体中只有一个语句时，大括号{}可以省

略,但最好不要省略,以增加程序的可读性。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,就进行(2),否则进行(3)。

(2) 执行循环体,再进行(1)。

(3) 结束 while 语句的执行。

while 语句的执行流程如图 3.7 所示。

3.5.3 do-while 循环语句

do-while 循环语句的语法格式为:

```
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 3.8 所示。

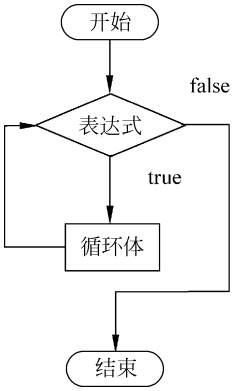

图 3.7 while 循环语句流程

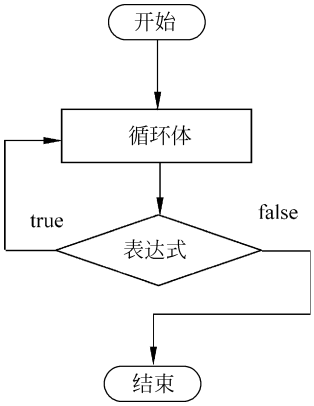

图 3.8 do-while 循环语句流程

下面的例子 6 用 while 语句计算 $1+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\cdots$的前 20 项之和。

例子 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);
   }
}
```

扫一扫

视频讲解

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 continue 语句后面的语句，而转入进行下一次循环。

下面的例子 7 使用了 break 和 continue 语句。

例子 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 <= 100;j++) {               //求 100 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");
            }
        }
    }
}
```

扫一扫

视频讲解

3.7　for 语句与数组

JDK 5 对 for 语句的功能给予了扩充、增强，以便用户更好地遍历数组。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，为了便于读者理解上述 for 语句，可以将这种形式的 for 语句翻译成“对于循

环变量依次取数组中的每一个元素的值”。

下面的例子 8 分别使用 for 语句的传统方式和改进方式遍历数组。

例子 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 中的每一个元素的值(改进方式)
         System.out.println(i);
      }
      for(char ch:b) {            //循环变量 ch 依次取数组 b 中的每一个元素的值(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,上述例子 8 中的第 3 个 for 语句不可以如下分开写成一个变量声明和一个 for 语句:

```
int i = 0;                    //变量声明
for(i:a) {                    //for 语句
  System.out.println(i);
}
```

扫一扫

视频讲解

3.8 应用举例

在 2.4 节中介绍了 Scanner 类,可以使用该类创建一个对象:

```
Scanner reader = new Scanner(System.in);
```

然后 reader 对象调用下列方法,读取用户在命令行中输入的各种基本类型的数据:nextBoolean()、nextByte()、nextShort()、nextInt()、nextLong()、nextFloat()、nextDouble()。

上述方法在执行时都会阻塞,等待用户在命令行中输入数据并回车确认(知识点见 2.4 节)。例如,如果用户在键盘中输入一个 byte 取值范围内的整数 89,那么 reader 对象调用 hasNextByte()、hasNextInt()、hasNextLong()以及 hasNextDouble()返回的值都是 true。需要注意的是,如果用户从键盘输入带小数点的数字,例如 12.34,那么 reader 对象调用 hasNextDouble()返回的值是 true,而调用 hasNextByte()、hasNextInt()以及 hasNextLong()

返回的值都是 false。

在从键盘输入数据时，经常让 reader 对象先调用 hasNextXXX()方法等待用户在键盘输入数据，然后再调用 nextXXX()方法获取用户输入的数据。

在下面的例子 9 中，用户在键盘依次输入若干数字，最后输入一个非数字字符序列结束整个输入操作，这些数字之间以及数字和最后的非数字序列之间需要用空白(例如空格或回行)分隔，然后回车确认，程序将计算出这些数的和以及平均值，效果如图 3.9 所示。

```
98
129.77
865.88
end
3个数的和为1093.650000
3个数的平均值是364.550000
```

图 3.9　计算平均值

例子 9

Example3_9.java

```
import java.util.*;
public class Example3_9 {
    public static void main(String args[]){
        Scanner reader = new Scanner(System.in);
        double sum = 0;
        int m = 0;
        while(reader.hasNextDouble()){
            double x = reader.nextDouble();
            m = m+1;
            sum = sum+x;
        }
        System.out.printf("%d个数的和为%f\n",m,sum);
        System.out.printf("%d个数的平均值是%f\n",m,sum/m);
    }
}
```

3.9　小结

(1) Java 提供了丰富的运算符，例如算术运算符、关系运算符、逻辑运算符、位运算符等。

(2) Java 语言中常用的控制语句与 C 语言中的很相似。

(3) Java 提供了遍历数组的循环语句。

3.10　课外读物

课外读物均来自作者的教学辅助微信公众号 java-violin，扫描二维码即可观看、学习。

(1) 爱情故事。

(2) 生命的千年时移势迁。

(3) Kaprekar 数字黑洞。

(4) switch 表达式。

(1)　(2)　(3)　(4)

习 题 3

扫一扫

习题

扫一扫

自测题