第 3 章　运算符、表达式和语句

3.1　运算符与表达式

运算符指明对操作数所进行的运算。按操作数的数目来分，可分为一元运算符(如＋＋、－－)、二元运算符(如＋、－)和三元运算符(如?:)，对于一元运算符来说，可以有前缀表达式(如＋＋i)和后缀表达式(如 i＋＋)。按照运算符功能划分，基本的运算符分为下面几类。

- 算术运算符(＋，－，＊，/，%，＋＋，－－)；
- 关系运算符(＞，＜，＞＝，＜＝，＝＝，!＝)；
- 布尔逻辑运算符(!，&&，‖)；
- 位运算符(＞＞，＜＜，＞＞＞，&，|，^，～)；
- 赋值运算符(＝，及其扩展赋值运算符如＋＝)；
- 条件运算符(?:)；
- 其他，包括分量运算符(．)、下标运算符([])、实例运算符(instance of)、内存分配运算符(new)、强制类型转换运算符和方法调用运算符等。

3.1.1　算术运算符

算术运算符完成对整型数据或浮点型数据的算术运算。

1. 二元算术运算符

二元算术运算符的功能描述如表 3-1 所示。

表 3-1　二元算术运算符的功能描述

运算符	用法	描述	运算符	用法	描述
＋	op1＋op2	加	/	op1/op2	除
－	op1－op2	减	%	op1%op2	取模(求余)
＊	op1＊op2	乘			

与 C、C++ 不同，对取模运算符%来说，其操作数可以为浮点数，如 37.2%10＝7.2。

Java 对加运算符进行了扩展，使它能够进行字符串的连接，如"abc"＋"de"，得到串"abcde"。

当算数运算符做运算时，根据左、右两个操作数的类型决定结果值的类型。其规则如表 3-2 所示。

表 3-2　结果值的类型规则

左右操作数的类型	结果值类型
两个都是整数，其中之一是 long 类型	long
两个都是整数，而且都不是 long 类型	int
两个至少一个是 double 类型	double
两个至少一个是 float 类型，但都不是 double 类型	float

结果值的类型规则举例如图 3-1 所示。

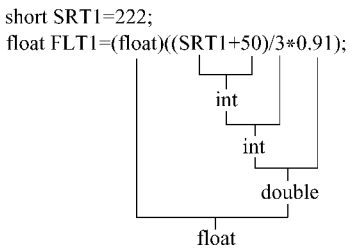

图 3-1　结果值类型规则举例

2. 一元算术运算符

所有的一元算术运算符如表 3-3 所示。

表 3-3　一元算术运算符功能描述

运　算　符	用　　法	描　　述
＋	＋op	正值
－	－op	负值
＋＋	＋＋op	先取得 op 变量的值，然后再将 op 变量储存的值加 1
＋＋	op＋＋	先将 op 变量储存的值加 1，然后再取得 op 变量的值
－－	－－op	先取得 op 变量的值，然后再将 op 变量储存的值减 1
－－	op－－	先将 op 变量储存的值减 1，然后再取得 op 变量的值

＋＋、－－，通常都是用于配合循环语句(for、while 等)使用。处理的对象必须是整数或浮点类型的变量，而不是数值常数(10、3.33 等)。这两个运算符放在“操作数之前”与放在“操作数之后”，两种情况所代表的意义不同。

i＋＋与＋＋i 的区别如下：

(1) i＋＋在使用 i 之后，因此执行完 i＋＋后，i 的值变为 i＋1，表达式值为 i。

(2) ＋＋i 在使用 i 之前，使 i 的值加 1，因此执行完＋＋i 后，整个表达式和 i 的值均为 i＋1。

对 i－－与－－i 和上面的道理相同。

例 3-1 所示为算术运算符的使用。

例 3-1 ArithmaticOp.java

```
1  public class ArithmaticOp
2  {
3      public static void main(String args[])
4      {
5          int a=5+4;//a=9
6          int b=a*2;//b=18
7          int c=b/4;//c=4
8          int d=b-c;//d=14
9          int e=-d;//e=-14
10         int f=e%4;//f=-2
11         double g=18.4;
12         double h=g%4;//h=2.4
13         int i=3;
14         int j=i++;//i=4,j=3
15         int k=++i;//i=5,k=5
16         System.out.println("a="+a);
17         System.out.println("b="+b);
18         System.out.println("c="+c);
19         System.out.println("d="+d);
20         System.out.println("e="+e);
21         System.out.println("f="+f);
22         System.out.println("g="+g);
23         System.out.println("h="+h);
24         System.out.println("i="+i);
25         System.out.println("j="+j);
26         System.out.println("k="+k);
27     }
28 }
```

程序运行结果如下：

```
a=9
b=18
c=4
d=14
e=-14
f=-2
g=18.4
h=2.3999999999999986
i=5
j=3
k=5
```

3.1.2　关系运算符

关系运算符用来比较两个值，返回布尔类型的值 true 或 false，通常用在条件语句中，作为判断的依据。关系运算符都是二元运算符，而所处理的对象必须是数值数据类型（整数、浮点数，包括 char），返回的结果则一定是 boolean 类型的值。而“关系运算符”总共有 6 个，如表 3-4 所示。

表 3-4　关系运算符功能描述

运算符	用法	返回 true 的情况
＞	op1＞op2	op1 大于 op2
＞＝	op1＞＝op2	op1 大于或等于 op2
＜	op1＜op2	op1 小于 op2
＜＝	op1＜＝op2	op1 小于或等于 op2
＝＝	op1＝＝op2	op1 与 op2 相等
！＝	op1！＝op2	op1 与 op2 不等

在 Java 中，任何数据类型的数据（包括基本类型和组合类型）都可以通过＝＝或！＝来比较是否相等（这与 C、C++ 不同）。

关系运算的结果返回 true 或 false，而不是 C、C++ 中的 1 或 0。

关系运算符常与布尔逻辑运算符一起使用，作为流控制语句的判断条件。如 if(a＞b&&b＝＝c)。

3.1.3　布尔逻辑运算符

布尔逻辑运算符进行布尔逻辑运算，和关系运算符一样返回的是 boolean 值，通常也用在条件语句中，作为判断的依据，因此“布尔运算符”也称为“条件运算符”，如表 3-5 所示。

表 3-5　布尔逻辑运算符功能描述

操作	op1	op2	op1&&op2	op1 ‖ op2	op1^op2	!op1
结果	false	false	false	false	false	true
结果	false	true	false	true	true	true
结果	true	false	false	true	true	false
结果	true	true	true	true	false	false

“‖”运算符，按 Shift＋\键就能输入一个“|”，而“‖”则是由连续两个“|”组成。

“&&”、“‖”、“^”为二元运算符，实现逻辑与、逻辑或、异或操作。

对于二元运算符“&&”，当左、右两边的操作数的值都是 true 时，所返回的结果才是 true；否则结果为 false。当“&&”作处理时，只要左方操作数的值为 false，就不再检验右方操作数的值，而直接返回结果为 false。

对于二元运算符“‖”，当左、右两边的操作数的值有一个以上是 true 时，所返回的结

果就是 true;否则返回结果为 false。当“‖”作处理时,只要左方操作数的值为 true,就不再检验右方操作数的值,而直接返回结果为 true。

对于二元运算符“^”,若处理对象是布尔类型的操作数时,且左、右两边操作数的值互斥(一个 true,一个 false),所返回的结果才是 true;否则结果则是 false。

“!”为一元运算符,实现逻辑非。其作用是对操作数做反向的运算。若操作数的值为布尔值的 false,返回结果是 true;相反,结果是 false。

对于布尔逻辑运算,先求出运算符左边的表达式的值,对或运算如果为 true,则整个表达式的结果为 true,不必对运算符右边的表达式再进行运算;同样,对与运算,如果左边表达式的值为 false,则不必对右边的表达式求值,整个表达式的结果为 false。例 3-2 说明了关系运算符和布尔逻辑运算符的使用。

例 3-2 RelationAndConditionOp.java

```
public class RelationAndConditionOp
{
    public static void main(String args[])
    {
        int a=25,b=3;
        boolean d=a<b;//d=false
        System.out.println("a<b="+d);
        int e=3;
        if(e!=0&&a/e>5)
            System.out.println("a/e="+a/e);
        int f=0;
        if(f!=0&&a/f>5)
            System.out.println("a/f="+a/f);
        else
            System.out.println("f="+f);
    }
}
```

程序运行结果如下:

```
a<b=false
a/e=8
f=0
```

在上例中第二个 if 语句在运行时不会发生除 0 溢出的错误,因为 f!=0 为 false,所以就不需要对 a/f 进行运算。

^、&、| 这三个布尔运算符,同时也是“位操作逻辑运算符”。

3.1.4 位运算符

用来对二进制位进行操作,Java 中提供了如表 3-6 所示的位运算符。

表 3-6　位运算符功能描述

运　算　符	说　　明	运　算　符	说　　明
&	按位与运算	^	按位异或运算
|	按位或运算	～	按位取反运算

位运算符中，除～以外，其余均为二元运算符。操作数只能为整型和字符型数据。

1. 补码

Java 使用补码来表示二进制数，在补码表示中，最高位为符号位，正数的符号位为 0，负数为 1。补码的规定如下：

(1) 正数。最高位为 0，其余各位代表数值本身(以二进制表示)，如＋42 的补码为 00101010。

(2) 负数。把该数绝对值的补码按位取反，然后对整个数加 1，即得该数的补码。如－42 的补码为 11010110(00101010 按位取反 11010101＋111010110)。

用补码来表示数，0 的补码是唯一的，都为 00000000。而在原码、反码表示中，＋0 和－0 的表示是不唯一的。可以用 111111 表示－1 的补码(这也是补码与原码和反码的区别)。

2. 按位取反运算符～

～是一元运算符，对数据按位取反，即把 1 变为 0，把 0 变为 1。

例如：

```
0010101=~1101010
```

～运算符与－运算符不同，～ 21≠ －21。

3. 按位与运算符 &

& 是二元运算符，对两个整型数据 a，b 按位进行与运算，运算结果是一个整型数据 c。运算法则是，参与运算的两个值，如果两个相应位都为 1，则该位的结果为 1，否则为 0。

```
0 & 0=0,0 & 1=0, 1 & 0=0, 1 & 1=1
```

如果 b 的精度高于 a，那么结果 c 的精度和 b 相同。

4. 按位或运算符|

|是二元运算符，对两个整型数据 a，b 按位进行或运算，运算结果是一个整型数据 c。运算法则是，参与运算的两个值，如果两个相应位都为 0，则该位的结果为 0，否则为 1。

```
0 | 0=0,0 | 1=1, 1 | 0=1, 1| 1=1
```

如果 b 的精度高于 a，那么结果 c 的精度和 b 相同。

5. 按位异或运算符^

^是二元运算符，对两个整型数据 a，b 按位进行异或运算，运算结果是一个整型数据

c。运算法则是,参与运算的两个值,如果两个对应位相同,则该位的结果为0,否则为1。

```
0 ^ 0=0,0 ^ 1=1, 1 ^ 0=1, 1 ^ 1=0
```

如果b的精度高于a,那么结果c的精度和b相同。

由异或的运算法则可知:

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

因此,如果c=a^b,那么a=c^b,即用同一个数b对数a进行两次“异或”运算的结果仍是数a。

3.1.5 赋值运算符

赋值是用等号运算符(=)进行的。它的意思是计算右边表达式的值,把计算结果复制到左边变量。右边的值可以是常数、变量或者表达式,只要能产生一个值就行。但左边的值必须是一个明确的、已命名的变量。也就是说,它必须有一个物理性的空间来保存右边的值。例如,可将一个常数赋给一个变量(如A=4),但不可将任何东西赋给一个常数(如不能4=A)。表3-7是赋值运算符的功能描述。

表3-7 赋值运算符的功能描述

运算符	功能	举例	等价于
=	右边数赋给左部变量	X=8.8赋给x	
+=	左右两边数相加,结果赋给左部变量	X=8,x+=3将11赋给x	X=x+3
-=	左右两边数相减,结果赋给左部变量	X=8,x-=3将5赋给x	X=x-3
*=	左右两边数相乘,结果赋给左部变量	X=8,x*=3将24赋给x	X=x*3
/=	右边数除以左边数,结果赋给左部变量	X=8,x/=4将2赋给x	X=x/4
%=	右边数除以左边数,余数赋给左部变量	X=16,x%=7将2赋给x	X=x%7

3.1.6 条件运算符

“?:”为三元运算符(不包括引号),“?”前面的语句为判断条件,“?”后面的语句用“:”隔开,为两个执行语句,如果“?”前面的条件为真,则执行“:”前面的语句;如果“?”前面的条件为假,则执行“:”后面的语句。例如:

```
(x<0)?1:3
```

如果x<0,那么这个结果为1;否则就是3。

3.2 Java运算符的优先级

当多个运算符出现在同一个表达式时,将根据各个运算符“执行优先权”的高低,来决定它们的执行顺序,而非直接由左向右执行。就像做数学的四则运算时,必须“先乘除,后

加减”的道理一样。而 Java 运算符执行优先权的高低，如表 3-8 所示。

表 3-8　Java 运算符优先级

执行的优先权	说　明	运算符(优先等级相同的分作一组)
高	以括号指定优先	()
	后置处理	[]、(参数)、op++、op--
	一元运算符	++op、--op、+op、-op、~、!
	建对象或转型	new、(类型)op
	乘除	*、/、%
	加减	+、-
	位运算符的位移	<<、>>、>>>
	比较关系	<、>、<=、>=、instantceof
	是否等于	==、!=
	位运算符的且	&
	位运算符的异或	^
	位运算符的或	|
	布尔运算符的且	&&
	布尔运算符的或	||
	条件运算符	? :
低	设定运算符	=、+=、-=、*=、/=、%=、&=、^=、|=、<<=、>>=、>>>=

表 3-8 所列运算符执行的优先权，由上而下代表由高至低的优先执行顺序，如一个表达式中具有“*”和“+”这两种不同层级的运算符时，无论哪一个在左，都会先做“*”运算符的处理。另外，若是同一层级的运算符之间，就依照它们在语句内的位置，从左至右依序执行它们的处理工作。

常用转意符如表 3-9 所示。

表 3-9　常用转意符

转义字符	描　述	转义字符	描　述
\'	单引号字符	\f	走纸换页
\\	反斜杠字符	\t	水平制表
\r	回车	\b	退格
\n	换行		

3.3　语　　句

通过语句可以完成对程序执行流程的控制，在 Java 语言中语句以分号结尾，但块语句或以块语句结尾的语句不需要分号结尾。

3.3.1 语句概述

Java 的语句可分为以下几类，即空语句、复合语句、标号语句、表达式语句、选择语句、循环语句、跳转语句和同步语句等。

1. 空语句

空语句仅包括一个分号，不执行任何操作。

2. 表达式语句

表达式语句是由一个表达式后加分号构成一个语句，分号是语句不可缺少的部分。其中最典型的是赋值语句。

```
x=10;
```

3. 复合语句

可以用{ }把一些语句括起来构成复合语句，例如：

```
{
    z=12+x;
    System.out.println("HelloWorld");
}
```

4. 标号语句

在一个语句之前加一标号(语句与标号以冒号隔开)就构成了标号语句，格式如下：

```
标号:语句
```

标号的命名规则符合标识符的命名规则，其作用域是在它所在的复合语句范围。标号只用来标识语句，并不影响语句的执行效果，其作用可在 break 和 continue 两种跳转语句中体现出来。

3.3.2 分支语句

分支语句提供了一种控制机制，使得程序的执行可以跳过这些语句，而转去执行特定的语句。

1. 条件语句 if-else

if-else 语句根据判定条件的真假来执行两种操作中的一种，即二分支程序结构，格式如下：

```
if(boolean-expression)
    statement1;
else
    statement2;
```

以下对 if-else 二分支程序结构进行具体解释说明。

(1) 布尔表达式 boolean-expression 是任意一个返回布尔型数据的表达式(这比 C、C++ 的限制要严格)。

(2) 每个单一的语句后都必须有分号。

(3) 语句 statement1,statement2 可以为复合语句,这时要用大括号{}括起。建议对单一的语句也用大括号括起,这样可增强程序的可读性,而且有利于程序的扩充(可以在其中添加新的语句)。{}外面不加分号。

(4) else 子句是任选的。

(5) 若布尔表达式的值为 true,则程序执行 statement1,否则执行 statement2。

(6) if-else 语句的一种特殊形式如下:

```
if(expression1)
{
    statement1
}
else if(expression2)
{
    statement2
}
⋮
else if(expressionM)
{
    statementM
}
else
{
    statementN
}
```

else 子句不能单独作为语句使用,它必须和 if 配对使用。else 总是与离它最近的 if 配对。可以通过使用大括号{}来改变配对关系。上述是一种多分支程序结构。

例 3-3 比较两个数的大小,并按从小到大的次序输出。

例 3-3　CompareTwo.java

```
public class CompareTwo
{
    public static void main(String args[])
    {
        double d1=23.4;
        double d2=35.1;
```