**Java 运算符**

计算机的最基本用途之一就是执行数学运算，作为一门计算机语言，Java也提供了一套丰富的运算符来操纵变量。我们可以把运算符分成以下几组：

* 算术运算符
* 关系运算符
* 位运算符
* 逻辑运算符
* 赋值运算符
* 其他运算符

**算术运算符**

算术运算符用在数学表达式中，它们的作用和在数学中的作用一样。下表列出了所有的算术运算符。

表格中的实例假设整数变量A的值为10，变量B的值为20：

|  |  |  |
| --- | --- | --- |
| **操作符** | **描述** | **例子** |
| + | 加法 - 相加运算符两侧的值 | A + B等于30 |
| - | 减法 - 左操作数减去右操作数 | A – B等于-10 |
| \* | 乘法 - 相乘操作符两侧的值 | A \* B等于200 |
| / | 除法 - 左操作数除以右操作数 | B / A等于2 |
| ％ | 取模 - 右操作数除左操作数的余数 | B%A等于0 |
| + + | 自增 - 操作数的值增加1 | B + +等于21 |
| -- | 自减 -- 操作数的值减少1 | B - -等于19 |

**实例**

下面的简单示例程序演示了算术运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

public class Test {

public static void main(String args[]) {

int a = 10;

int b = 20;

int c = 25;

int d = 25;

System.out.println("a + b = " + (a + b) );

System.out.println("a - b = " + (a - b) );

System.out.println("a \* b = " + (a \* b) );

System.out.println("b / a = " + (b / a) );

System.out.println("b % a = " + (b % a) );

System.out.println("c % a = " + (c % a) );

System.out.println("a++ = " + (a++) );

System.out.println("a-- = " + (a--) );

// 查看 d++ 与 ++d 的不同

System.out.println("d++ = " + (d++) );

System.out.println("++d = " + (++d) );

}

}

[尝试一下 »](http://www.runoob.com/try/showjava.php?filename=test_operator)

以上实例编译运行结果如下：

a + b = 30

a - b = -10

a \* b = 200

b / a = 2

b % a = 0

c % a = 5

a++ = 10

b-- = 11

d++ = 25

++d = 27

**关系运算符**

下表为Java支持的关系运算符

表格中的实例整数变量A的值为10，变量B的值为20：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **例子** |
| == | 检查如果两个操作数的值是否相等，如果相等则条件为真。 | （A == B）为假(非真)。 |
| != | 检查如果两个操作数的值是否相等，如果值不相等则条件为真。 | (A != B) 为真。 |
| > | 检查左操作数的值是否大于右操作数的值，如果是那么条件为真。 | （A> B）非真。 |
| < | 检查左操作数的值是否小于右操作数的值，如果是那么条件为真。 | （A <B）为真。 |
| > = | 检查左操作数的值是否大于或等于右操作数的值，如果是那么条件为真。 | （A> = B）为假。 |
| <= | 检查左操作数的值是否小于或等于右操作数的值，如果是那么条件为真。 | （A <= B）为真。 |

**实例**

下面的简单示例程序演示了关系运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

public class Test {

public static void main(String args[]) {

int a = 10;

int b = 20;

System.out.println("a == b = " + (a == b) );

System.out.println("a != b = " + (a != b) );

System.out.println("a > b = " + (a > b) );

System.out.println("a < b = " + (a < b) );

System.out.println("b >= a = " + (b >= a) );

System.out.println("b <= a = " + (b <= a) );

}

}

以上实例编译运行结果如下：

a == b = false

a != b = true

a > b = false

a < b = true

b >= a = true

b <= a = false

**位运算符**

Java定义了位运算符，应用于整数类型(int)，长整型(long)，短整型(short)，字符型(char)，和字节型(byte)等类型。

位运算符作用在所有的位上，并且按位运算。假设a = 60，和b = 13;它们的二进制格式表示将如下：

A = 0011 1100

B = 0000 1101

-----------------

A&b = 0000 1100

A | B = 0011 1101

^ B = 0011 0001

~A= 1100 0011

下表列出了位运算符的基本运算,假设整数变量A的值为60和变量B的值为13：

|  |  |  |
| --- | --- | --- |
| **操作符** | **描述** | **例子** |
| ＆ | 按位与操作符，当且仅当两个操作数的某一位都非0时候结果的该位才为1。 | （A＆B），得到12，即0000 1100 |
| | | 按位或操作符，只要两个操作数的某一位有一个非0时候结果的该位就为1。 | （A | B）得到61，即 0011 1101 |
| ^ | 按位异或操作符，两个操作数的某一位不相同时候结果的该位就为1。 | （A ^ B）得到49，即 0011 0001 |
| 〜 | 按位补运算符翻转操作数的每一位。 | （〜A）得到-60，即1100 0011 |
| << | 按位左移运算符。左操作数按位左移右操作数指定的位数。 | A << 2得到240，即 1111 0000 |
| >> | 按位右移运算符。左操作数按位右移右操作数指定的位数。 | A >> 2得到15即 1111 |
| >>> | 按位右移补零操作符。左操作数的值按右操作数指定的位数右移，移动得到的空位以零填充。 | A>>>2得到15即0000 1111 |

**实例**

下面的简单示例程序演示了位运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

public class Test {

  public static void main(String args[]) {

    int a = 60; /\* 60 = 0011 1100 \*/

     int b = 13; /\* 13 = 0000 1101 \*/

     int c = 0;

     c = a & b;       /\* 12 = 0000 1100 \*/

     System.out.println("a & b = " + c );

     c = a | b;       /\* 61 = 0011 1101 \*/

     System.out.println("a | b = " + c );

     c = a ^ b;       /\* 49 = 0011 0001 \*/

     System.out.println("a ^ b = " + c );

     c = ~a;          /\*-61 = 1100 0011 \*/

     System.out.println("~a = " + c );

     c = a << 2;     /\* 240 = 1111 0000 \*/

     System.out.println("a << 2 = " + c );

     c = a >> 2;     /\* 15 = 1111 \*/

     System.out.println("a >> 2  = " + c );

   c = a >>> 2;     /\* 15 = 0000 1111 \*/

     System.out.println("a >>> 2 = " + c );

  }

}

以上实例编译运行结果如下：

a & b = 12

a | b = 61

a ^ b = 49

~a = -61

a << 2 = 240

a >> 15

a >>> 15

**逻辑运算符**

下表列出了逻辑运算符的基本运算，假设布尔变量A为真，变量B为假

|  |  |  |
| --- | --- | --- |
| **操作符** | **描述** | **例子** |
| && | 称为逻辑与运算符。当且仅当两个操作数都为真，条件才为真。 | （A && B）为假。 |
| | | | 称为逻辑或操作符。如果任何两个操作数任何一个为真，条件为真。 | （A | | B）为真。 |
| ！ | 称为逻辑非运算符。用来反转操作数的逻辑状态。如果条件为true，则逻辑非运算符将得到false。 | ！（A && B）为真。 |

**实例**

下面的简单示例程序演示了逻辑运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

public class Test {

  public static void main(String args[]) {

     boolean a = true;

     boolean b = false;

     System.out.println("a && b = " + (a&&b));

     System.out.println("a || b = " + (a||b) );

     System.out.println("!(a && b) = " + !(a && b));

  }

}

以上实例编译运行结果如下：

a && b = false

a || b = true

!(a && b) = true

**赋值运算符**

下面是Java语言支持的赋值运算符：

|  |  |  |
| --- | --- | --- |
| **操作符** | **描述** | **例子** |
| = | 简单的赋值运算符，将右操作数的值赋给左侧操作数 | C = A + B将把A + B得到的值赋给C |
| + = | 加和赋值操作符，它把左操作数和右操作数相加赋值给左操作数 | C + = A等价于C = C + A |
| - = | 减和赋值操作符，它把左操作数和右操作数相减赋值给左操作数 | C - = A等价于C = C -  A |
| \* = | 乘和赋值操作符，它把左操作数和右操作数相乘赋值给左操作数 | C \* = A等价于C = C \* A |
| / = | 除和赋值操作符，它把左操作数和右操作数相除赋值给左操作数 | C / = A等价于C = C / A |
| （％）= | 取模和赋值操作符，它把左操作数和右操作数取模后赋值给左操作数 | C％= A等价于C = C％A |
| << = | 左移位赋值运算符 | C << = 2等价于C = C << 2 |
| >> = | 右移位赋值运算符 | C >> = 2等价于C = C >> 2 |
| ＆= | 按位与赋值运算符 | C＆= 2等价于C = C＆2 |
| ^ = | 按位异或赋值操作符 | C ^ = 2等价于C = C ^ 2 |
| | = | 按位或赋值操作符 | C | = 2等价于C = C | 2 |

**实例**

面的简单示例程序演示了赋值运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

public class Test {

  public static void main(String args[]) {

     int a = 10;

     int b = 20;

     int c = 0;

     c = a + b;

     System.out.println("c = a + b = " + c );

     c += a ;

     System.out.println("c += a  = " + c );

     c -= a ;

     System.out.println("c -= a = " + c );

     c \*= a ;

     System.out.println("c \*= a = " + c );

     a = 10;

     c = 15;

     c /= a ;

     System.out.println("c /= a = " + c );

     a = 10;

     c = 15;

     c %= a ;

     System.out.println("c %= a  = " + c );

     c <<= 2 ;

     System.out.println("c <<= 2 = " + c );

     c >>= 2 ;

     System.out.println("c >>= 2 = " + c );

     c >>= 2 ;

     System.out.println("c >>= a = " + c );

     c &= a ;

     System.out.println("c &= 2  = " + c );

     c ^= a ;

     System.out.println("c ^= a   = " + c );

     c |= a ;

     System.out.println("c |= a   = " + c );

  }

}

以上实例编译运行结果如下：

c = a + b = 30

c += a = 40

c -= a = 30

c \*= a = 300

c /= a = 1

c %= a = 5

c <<= 2 = 20

c >>= 2 = 5

c >>= 2 = 1

c &= a = 0

c ^= a = 10

c |= a = 10

**条件运算符（?:）**

条件运算符也被称为三元运算符。该运算符有3个操作数，并且需要判断布尔表达式的值。该运算符的主要是决定哪个值应该赋值给变量。

variable x = (expression) ? value if true : value if false

**实例**

public class Test {

public static void main(String args[]){

int a , b;

a = 10;

b = (a == 1) ? 20: 30;

System.out.println( "Value of b is : " + b );

b = (a == 10) ? 20: 30;

System.out.println( "Value of b is : " + b );

}

}

以上实例编译运行结果如下：

Value of b is : 30

Value of b is : 20

**instanceOf 运算符**

该运算符用于操作对象实例，检查该对象是否是一个特定类型（类类型或接口类型）。

instanceof运算符使用格式如下：

( Object reference variable ) instanceOf (class/interface type)

如果运算符左侧变量所指的对象，是操作符右侧类或接口(class/interface)的一个对象，那么结果为真。

下面是一个例子：

String name = 'James';

boolean result = name instanceOf String; // 由于name是Strine类型，所以返回真

如果被比较的对象兼容于右侧类型,该运算符仍然返回true。

看下面的例子：

class Vehicle {}

public class Car extends Vehicle {

public static void main(String args[]){

Vehicle a = new Car();

boolean result = a instanceof Car;

System.out.println( result);

}

}

以上实例编译运行结果如下：

true

**Java运算符优先级**

当多个运算符出现在一个表达式中，谁先谁后呢？这就涉及到运算符的优先级别的问题。在一个多运算符的表达式中，运算符优先级不同会导致最后得出的结果差别甚大。

例如，（1+3）＋（3+2）\*2，这个表达式如果按加号最优先计算，答案就是 18，如果按照乘号最优先，答案则是 14。

再如，x = 7 + 3 \* 2;这里x得到13，而不是20，因为乘法运算符比加法运算符有较高的优先级，所以先计算3 \* 2得到6，然后再加7。

下表中具有最高优先级的运算符在的表的最上面，最低优先级的在表的底部。

|  |  |  |
| --- | --- | --- |
| **类别** | **操作符** | **关联性** |
| 后缀 | () [] . (点操作符) | 左到右 |
| 一元 | + + - ！〜 | 从右到左 |
| 乘性 | \* /％ | 左到右 |
| 加性 | + - | 左到右 |
| 移位 | >> >>>  << | 左到右 |
| 关系 | >> = << = | 左到右 |
| 相等 | ==  != | 左到右 |
| 按位与 | ＆ | 左到右 |
| 按位异或 | ^ | 左到右 |
| 按位或 | | | 左到右 |
| 逻辑与 | && | 左到右 |
| 逻辑或 | | | | 左到右 |
| 条件 | ？： | 从右到左 |
| 赋值 | = + = - = \* = / =％= >> = << =＆= ^ = | = | 从右到左 |
| 逗号 | ， | 左到右 |