

Java 教程

Java 教程

Java 简介

Java 开发环境配置

Java 基础语法

Java 对象和类

Java 基本数据类型

Java 变量类型

Java 修饰符

Java 运算符

Java 循环结构

Java 分支结构

Java Number & Math 类

Java Character 类

Java String 类

Java StringBuffer

Java 数组

Java 日期时间

Java 正则表达式

Java 方法

Java Stream、File、IO

Java Scanner 类

Java 异常处理

Java 面向对象

Java 继承

Java Override/Overload

Java 多态

Java 抽象类

Java 封装

Java 接口

Java 包(package)

← Java 修饰符

Java 循环结构 – for, while 及 do...while →

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 - 左操作数除右操作数的余数	B%A等于0
++	自增: 操作数的值增加1	B++ 或 ++B 等于 21 (区别详见下文)
--	自减: 操作数的值减少1	B-- 或 --B 等于 19 (区别详见下

HTML / CSS

JavaScript

服务端

数据库

移动端

XML 教程

ASP.NET

Web Service

开发工具

网站建设

Advertisement



我的建议

Java 高级教程

Java 数据结构

Java 集合框架

Java 泛型

Java 序列化

Java 网络编程

Java 发送邮件

Java 多线程编程

Java Applet 基础

Java 文档注释

Java 实例

Java 8 新特性

Java MySQL 连接

文)

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：

实例

```
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) );

    }

}
```

运行实例 »

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
a-- = 11
d++ = 25
++d = 27
```

自增自减运算符

1、自增 (++) 自减 (--) 运算符是一种特殊的算术运算符，在算术运算符中需要两个操作数来进行运算，而自增自减运算符是一个操作数。

实例

```
public class selfAddMinus{

    public static void main(String[] args){

        int a = 3; //定义一个变量;
        int b = ++a; //自增运算
        int c = 3;
        int d = --c; //自减运算
        System.out.println("进行自增运算后的值等于"+b);

    }

}
```



我的建议

```
        System.out.println("进行自减运算后的值等于"+d);
    }
}
```

运行结果为：

进行自增运算后的值等于4
进行自减运算后的值等于2

解析：

int b = ++a; 拆分运算过程为: a=a+1=4; b=a=4, 最后结果为b=4,a=4

int d = --c; 拆分运算过程为: c=c-1=2; d=c=2, 最后结果为d=2,c=2

- 2、前缀自增自减法(++a,--a): 先进行自增或者自减运算，再进行表达式运算。
- 3、后缀自增自减法(a++,a--): 先进行表达式运算，再进行自增或者自减运算 实例：

实例

```
public class selfAddMinus{
    public static void main(String[] args){
        int a = 5;//定义一个变量;
        int b = 5;
        int x = 2*++a;
        int y = 2*b++;
        System.out.println("自增运算符前缀运算后
a="+a+", x="+x);
        System.out.println("自增运算符后缀运算后
b="+b+", y="+y);
    }
}
```

运行结果为：

自增运算符前缀运算后a=6 , x=12
自增运算符后缀运算后b=6 , y=10

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：

Test.java 文件代码：

```
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，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
A ^ 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
&	如果相对应位都是1，则结果为1，否则为0	(A & B)，得到12，即0000 1100
	如果相对应位都是0，则结果为0，否则为1	(A B) 得到61，即 0011 1101
^	如果相对应位值相同，则结果为0，否则为1	(A ^ B) 得到49，即 0011 0001
~	按位补运算符翻转操作数的每一位，即0变成1，1变成0。	(~A) 得到-61，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：

Test.java 文件代码：

```
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：

实例

```
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
```

短路逻辑运算符

当使用与逻辑运算符时，在两个操作数都为true时，结果才为true，但是当得到第一个操作为false时，其结果就必定是false，这时候就不会再判断第二个操作了。



实例

```
public class LuoJi{
    public static void main(String[] args){
        int a = 5; //定义一个变量;
        boolean b = (a<4) && (a++<10);
        System.out.println("使用短路逻辑运算符的结果为"+b);
        System.out.println("a的结果为"+a);
    }
}
```

运行结果为：

使用短路逻辑运算符的结果为false
a的结果为5

*解析：*该程序使用到了短路逻辑运算符(&&)，首先判断 a<4 的结果为 false，则 b 的结果必定是 false，所以不再执行第二个操作 a++<10 的判断，所以 a 的值为 5。

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



我的建议

		&2
<code>^ =</code>	按位异或赋值操作符	<code>C ^ = 2</code> 等价于 <code>C = C ^ 2</code>
<code> =</code>	按位或赋值操作符	<code>C = 2</code> 等价于 <code>C = C 2</code>

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：

Test.java 文件代码：

```
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

Test.java 文件代码：

```
public class Test {
    public static void main(String args[]){
        int a , b;
        a = 10;
        // 如果 a 等于 1 成立，则设置 b 为 20，否则为 30
        b = (a == 1) ? 20 : 30;
        System.out.println( "Value of b is : " + b );

        // 如果 a 等于 10 成立，则设置 b 为 20，否则为 30
        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 String
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

笔记列表

 在判断一个实例引用的类型时，使用的是实际类型，而不是声明的类型。在下面的代码中，

```
Vehicle v2 = new Car(); // v2 是 Car 类型
```

v2 是 Car 类型，而不是 Vehicle 类型。

```
class Vehicle {}

public class Car extends Vehicle {
    public static void main(String args[]){
        Car c1 = new Car();

        Vehicle v2 = new Car(); // v2 是 Car 类型

        Vehicle v3 = new Vehicle();

        //Car 是 Vehicle类型, Vehicle 不是 Car 类型

        boolean result1 = c1 instanceof Vehicle;
        // true
        boolean result2 = v2 instanceof Car;
        // true
        boolean result3 = v2 instanceof Vehicle;
        // true
        boolean result4 = v3 instanceof Car;
        // false

        System.out.println(result1);
        System.out.println(result2);
        System.out.println(result3);
        System.out.println(result4);
    }
}
```

stinkaroo 4周前 (05-09)



 点我分享笔记

笔记需要是本篇文章的内容扩展！
[注册邀请码获取方式](#)

在线实例

- [HTML 实例](#)
- [CSS 实例](#)
- [JavaScript 实例](#)
- [Ajax 实例](#)
- [jQuery 实例](#)
- [XML 实例](#)
- [Java 实例](#)

字符集&工具

- [HTML 字符集设置](#)
- [HTML ASCII 字符集](#)
- [HTML ISO-8859-1](#)
- [HTML 实体符号](#)
- [HTML 拾色器](#)
- [JSON 格式化工具](#)

最新更新

- [kotlin 委托](#)
- [委托模式](#)
- [Kotlin 对象表达...](#)
- [Kotlin 枚举类](#)
- [Kotlin 泛型](#)
- [Kotlin 数据类与...](#)
- [Kotlin 扩展](#)

站点信息

- [意见反馈](#)
- [免责声明](#)
- [关于我们](#)
- [文章归档](#)

关注微信



Copyright © 2013-2017 菜鸟教程 **runoob.com** All Rights Reserved. 备案号：闽ICP备15012807号-1

