



# Java 的运算系统

- 所有碰到优先级问题就加括号, ( ) 的优先级最高

## ▼ 算术运算符 Arithmetic Operators

运算符	含义	<code>a = 10, b = 3</code>	结果
<code>+</code>	加	<code>a + b</code>	13
<code>-</code>	减	<code>a - b</code>	7
<code>*</code>	乘	<code>a * b</code>	30
<code>/</code>	除	<code>a / b</code>	3 (整数除法)
<code>%</code>	取余数	<code>a % b</code>	1

## ▼ 整数除法是地板除

- ✓ 当两个整数相除时, **小数部分会被截断**, 保留整数部分 📌 **向下取整 地板除**

```
System.out.println(7 / 2); // 输出 3
```

- `7 / 2` 都是 `int`, 所以执行的是整数除法
- 小数部分 `.5` 被截断了

- ✓ 如果想得到小数, 至少要有一个是 `float` 或 `double`

```
System.out.println(7.0 / 2); // 输出 3.5
```

## ▼ `a+=` `a++` `++a`

- `a += 1` 📌 `a = a + 1`

▼ `a++` `++a`

表达式	名称	含义	使用时机
<code>a++</code>	后缀自增	先使用 <code>a</code> ，再执行 <code>a = a + 1</code>	先用后加
<code>++a</code>	前缀自增	先执行 <code>a = a + 1</code> ，再使用 <code>a</code>	先加后用

```
int a = 3;
System.out.println(a++); // 输出 3, 但 a 变成 4

int b = 3;
System.out.println(++b); // 输出 4, 因为先加了再用

int a = 5;
int b = a++; // b = 5, a = 6 (先把a赋值给b, 然后a再加1)

int x = 5;
int y = ++x; // x = 6, y = 6 (先加1, 再赋值)
```

- `a++`：先用原值，再加一（“先用后增”）
- `++a`：先加一，再用新值（“先增后用”）

## ▼ 比较运算符 Comparison Operators

用于比较两个值，返回 `true` 或 `false`：

运算符	含义	示例
<code>==</code>	等于	<code>a == b</code>
<code>!=</code>	不等于	<code>a != b</code>
<code>&gt;</code>	大于	<code>a &gt; b</code>
<code>&lt;</code>	小于	<code>a &lt; b</code>
<code>&gt;=</code>	大于等于	<code>a &gt;= b</code>
<code>&lt;=</code>	小于等于	<code>a &lt;= b</code>

## ▼ 逻辑运算符 Logical Operators

运算符	含义	示例	结果	
&&	与	true && false	false	一假全假
	或	true    false	true	一真全真
!	非	!true	false	

#### ▼ 短路特性 Short-Circuit Evaluation

##### && 逻辑与

- 如果左边是 `false`，右边**不再执行**
- 因为无论右边真假，整个表达式都是 `false`

```
if (false && someMethod()) {
    // someMethod 不会被执行
}
```

##### || 逻辑或

- 如果左边是 `true`，右边**不再执行**
- 因为无论右边真假，整个表达式都是 `true`

```
if (true || someMethod()) {
    // someMethod 不会被执行
}
```

### 防止空指针异常

```
String str = null;

// 正确：先判断是否为 null
if (str != null && str.length() > 5) {
    System.out.println("字符串长度大于 5");
}
```

```
// 错误（没有短路）：先执行 str.length() 会报 NullPointerException
if (str.length() > 5 && str != null) {
    // ❌ 报错：空指针异常
}
```

## ✅ 总结口诀：

- `&&` 遇到 `false` 就停（左边是 `false`，右边不执行）
- `||` 遇到 `true` 就停（左边是 `true`，右边不执行）

### ▼ 三元运算符 Ternary Operator

用于根据条件快速返回一个值

## ✅ 语法格式：

条件表达式 ? 表达式1 : 表达式2;

## 含义：

- 如果 `条件表达式` 为 `true`，返回 `表达式1`
- 如果为 `false`，返回 `表达式2`

```
int age = 18;
System.out.println(age >= 18 ? "成年人" : "未成年");
// 输出：成年人
```

## ✅ 嵌套使用

```
int score = 75;
String result = (score >= 90) ? "优秀" : (score >= 60) ? "及格" : "不及格";
// 输出：及格
```

### ▼ 赋值运算符 Assignment Operators

运算符	含义	示例	等价于
<code>=</code>	赋值	<code>a = b</code>	
<code>+=</code>	加后赋值	<code>a += 1</code>	<code>a = a + 1</code>
<code>--</code>	减后赋值	<code>a -= 1</code>	<code>a = a - 1</code>
<code>*=</code>	乘后赋值	<code>a *= 2</code>	<code>a = a * 2</code>
<code>/=</code>	除后赋值	<code>a /= 2</code>	<code>a = a / 2</code>
<code>%=</code>	取余后赋值	<code>a %= 2</code>	<code>a = a % 2</code>


### ▼ 自增/自减运算符 Increment / Decrement

运算符	含义	示例	说明
<code>++</code>	自增1	<code>a++</code>	先使用a，再加1（后缀）
<code>++</code>	自增1	<code>++a</code>	先加1，再使用a（前缀）
<code>--</code>	自减1	<code>a--</code>	同上

### ▼ 位运算符 Bitwise Operators

位运算符是**对整数的二进制位进行直接操作**，常用于性能优化、底层开发、权限控制等场景

运算符	名称	示例（假设 a = 5, b = 3）	解释
<code>&amp;</code>	按位与	<code>a &amp; b</code> → 1	两位都为 1 才为 1
<code> </code>	按位或	<code>a   b</code> → 7	任一为 1 即为 1
<code>^</code>	按位异或	<code>a ^ b</code> → 6	不同为 1，相同为 0
<code>~</code>	按位取反	<code>~a</code> → -6	所有位取反（包括符号位）
<code>&lt;&lt;</code>	左移	<code>a &lt;&lt; 1</code> → 10	向左移动 n 位，相当于乘以 2 <sup>n</sup>
<code>&gt;&gt;</code>	右移	<code>a &gt;&gt; 1</code> → 2	向右移动 n 位，相当于除以 2 <sup>n</sup> （保符号）
<code>&gt;&gt;&gt;</code>	无符号右移	<code>a &gt;&gt;&gt; 1</code>	右移 n 位，高位补 0（无符号）

 **a = 5, b = 3**

```
int a = 5; // 二进制 0101
int b = 3; // 二进制 0011
```

```
System.out.println(a & b); // 1 ⇒ 0001
System.out.println(a | b); // 7 ⇒ 0111
System.out.println(a ^ b); // 6 ⇒ 0110
System.out.println(~a);    // -6 ⇒ 取反结果是补码
System.out.println(a << 1); // 10 ⇒ 左移一位，相当于乘2
System.out.println(a >> 1); // 2  ⇒ 右移一位，相当于除2
```

## ✅ 左移、右移图解（以 8-bit 为例）

```
a = 5    → 00000101
a << 1   → 00001010 (5 × 2 = 10)
a >> 1   → 00000010 (5 / 2 = 2)
```

## 📌 应用场景：

1. 权限系统（每一位表示一个开关）
2. 性能优化（乘除 2 替代算术运算）
3. 掩码判断（与运算判断某位是否为1）
4. 颜色分离（图像处理中拆分 RGB）