# 运算符优先级





相关主题
*JavaScript*

**教程:**

▸ 快速入门
JavaScript 基础知识
JavaScript first steps
JavaScript building blocks
Introducing JavaScript objects

▸ JavaScript 指南
Introduction
Grammar and types
Control flow and error handling
Loops and iteration
Functions
Expressions and operators
Numbers and dates
Text formatting
Regular expressions
Indexed collections
Keyed collections
Working with objects
Details of the object model
Using promises
Iterators and generators
Meta programming
JavaScript modules

▸ 中级教程
Introducing JavaScript objects
Client-side web APIs
深入 JavaScript
JavaScript 数据结构
如何正确判断相等性
Closures

▸ 高级
继承和原型链
严格模式
JavaScript 类型化数组
内存管理
Concurrency model and Event Loop

**引用:**

▸ 内置对象
AggregateError
Array
ArrayBuffer
AsyncFunction
AsyncIterator
Atomics
BigInt
BigInt64Array
BigUint64Array
Boolean
DataView
Date
Error
EvalError
Float32Array
Float64Array
Function
Generator
GeneratorFunction
Infinity
Int16Array
Int32Array
Int8Array
InternalError
Intl
Intl.Collator
Intl.DateTimeFormat
Intl.DisplayNames
Intl.ListFormat
Intl.Locale
Intl.NumberFormat
Intl.PluralRules
Intl.RelativeTimeFormat
Iterator [我来译!]
JSON
Map
Math
NaN
Number
Object
Promise
Proxy
RangeError
ReferenceError
Reflect
RegExp
Set
SharedArrayBuffer
String
Symbol
SyntaxError
TypeError
TypedArray
URIError
Uint16Array
Uint32Array
Uint8Array
Uint8ClampedArray
WeakMap
WeakSet
WebAssembly
decodeURI()
decodeURIComponent()
encodeURI()
encodeURIComponent()
escape()
eval()
globalThis
isFinite()
isNaN()
null
parseFloat
parseInt
undefined
unescape()
uneval()

▾ 表达式和运算符
算术运算符
数组推导式
赋值运算符
按位操作符
逗号操作符
比较操作符
条件运算符
解构赋值
Expression closures
函数表达式
Generator推导式
圆括号运算符
旧式生成器函数
逻辑运算符
Nullish coalescing operator
对象初始化
运算符优先级
可选链
管道操作符
属性访问器
展开语法
async function expression
await
类表达式
delete 操作符
function* 表达式

运算符的优先级决定了表达式中运算执行的先后顺序，优先级高的运算符最先被执行。

**JavaScript Demo: Expressions - Operator precedence**

```
console.log(3 + 4 * 5); // 3 + 20
// expected output: 23

console.log(4 * 3 ** 2); // 4 * 9
// expected output: 36

let a;
let b;

console.log(a = b = 5);
// expected output: 5

```

Run ›
Reset

## 关联性

关联性决定了拥有相同优先级的运算符的执行顺序。考虑下面这个表达式:

```
a OP b OP c
```

左关联（左到右）相当于把左边的子表达式加上小括号(`a OP b) OP c`，右关联（右到左）相当于`a OP (b OP c)`。赋值运算符是右关联的，所以你可以这么写:

```
a = b = 5;
```

结果 a 和 b 的值都会成为5。这是因为赋值运算符的返回结果就是赋值运算符右边的那个值，
具体过程是：b被赋值为5，然后a也被赋值为 b=5 的返回值，也就是5。

## 示例

```
3 > 2 && 2 > 1
// return true

3 > 2 > 1
// 返回 false，因为 3 > 2 是 true，并且 true > 1 is false
// 加括号可以更清楚：(3 > 2) > 1
```

## 汇总表

下面的表将所有运算符按照优先级的不同从高（20）到低（1）排列。

| 优先级 | 运算类型 | 关联性 | 运算符 |
|---|---|---|---|
| 20 | 圆括号 | n/a（不相关） | `( … )` |
| 19 | 成员访问 | 从左到右 | `… . …` |
| | 需计算的成员访问 | 从左到右 | `… [ … ]` |
| | new (带参数列表) | n/a | `new … ( … )` |
| | 函数调用 | 从左到右 | `… ( … )` |
| | 可选链（Optional chaining） | 从左到右 | `?.` |
| 18 | new (无参数列表) | 从右到左 | `new …` |
| 17 | 后置递增(运算符在后) | n/a | `… ++` |
| | 后置递减(运算符在后) | | `… --` |
| 16 | 逻辑非 | 从右到左 | `! …` |
| | 按位非 | | `~ …` |
| | 一元加法 | | `+ …` |
| | 一元减法 | | `- …` |
| | 前置递增 | | `++ …` |
| | 前置递减 | | `-- …` |
| | typeof | | `typeof …` |
| | void | | `void …` |
| | delete | | `delete …` |
| | await | | `await …` |
| 15 | 幂 | 从右到左 | `… ** …` |
| 14 | 乘法 | 从左到右 | `… * …` |
| | 除法 | | `… / …` |
| | 取模 | | `… % …` |
| 13 | 加法 | 从左到右 | `… + …` |
| | 减法 | | `… - …` |
| 12 | 按位左移 | 从左到右 | `… << …` |
| | 按位右移 | | `… >> …` |
| | 无符号右移 | | `… >>> …` |
| 11 | 小于 | 从左到右 | `… < …` |
| | 小于等于 | | `… <= …` |
| | 大于 | | `… > …` |
| | 大于等于 | | `… >= …` |
| | in | | `… in …` |
| | instanceof | | `… instanceof …` |
| 10 | 等号 | 从左到右 | `… == …` |
| | 非等号 | | `… != …` |
| | 全等号 | | `… === …` |
| | 非全等号 | | `… !== …` |
| 9 | 按位与 | 从左到右 | `… & …` |
| 8 | 按位异或 | 从左到右 | `… ^ …` |
| 7 | 按位或 | 从左到右 | `… \| …` |
| 6 | 逻辑与 | 从左到右 | `… && …` |
| 5 | 逻辑或 | 从左到右 | `… \|\| …` |
| 4 | 条件运算符 | 从右到左 | `… ? … : …` |
| 3 | 赋值 | 从右到左 | `… = …` |
| | | | `… += …` |
| | | | `… -= …` |
| | | | `… *= …` |
| | | | `… /= …` |
| | | | `… %= …` |
| | | | `… <<= …` |
| | | | `… >>= …` |
| | | | `… >>>= …` |
| | | | `… &= …` |
| | | | `… ^= …` |
| | | | `… \|= …` |
| 2 | yield | 从右到左 | `yield …` |
| | yield* | | `yield* …` |
| 1 | 展开运算符 | n/a | `... …` |
| 0 | 逗号 | 从左到右 | `… , …` |

**最后修改：** 2019年10月7日, 由 Mozilla 贡献者编辑