Основные операторы Javascript

Оглавление

[Операторы и операнды 0](#_Toc447573385)

[Приоритет операций 1](#_Toc447573386)

[Побитовые операторы 6](#_Toc447573387)

[Используемая литература 7](#_Toc447573388)

# Операторы и операнды

У операторов есть своя терминология, которая используется во всех языках программирования.

Прежде, чем мы двинемся дальше – несколько терминов, чтобы понимать, о чём речь.

* Операнд – то, к чему применяется оператор. Например: 5 \* 2 – оператор умножения с левым и правым операндами. Другое название: «аргумент оператора».
* Унарным называется оператор, который применяется к одному выражению. Например, оператор унарный минус "-" меняет знак числа на противоположный:

|  |
| --- |
| var x = 1;  x = -x;  alert( x ); // -1, применили унарный минус |

* Бинарным называется оператор, который применяется к двум операндам. Тот же минус существует и в бинарной форме:

|  |
| --- |
| var x = 1, y = 3;  alert( y - x ); // 2, бинарный минус |

Обычно при помощи плюса '+' складывают числа.

Но если бинарный оператор '+' применить к строкам, то он их объединяет в одну:

|  |
| --- |
| var a = "моя" + "строка";  alert( a ); // моястрока |

Иначе говорят, что «плюс производит конкатенацию (сложение) строк».

Если хотя бы один аргумент является строкой, то второй будет также преобразован к строке!

Причем не важно, справа или слева находится операнд-строка, в любом случае нестроковый аргумент будет преобразован. Например:

|  |
| --- |
| alert( '1' + 2 ); // "12"  alert( 2 + '1' ); // "21" |

Это приведение к строке – особенность исключительно бинарного оператора "+".

Остальные арифметические операторы работают только с числами и всегда приводят аргументы к числу.

Например:

|  |
| --- |
| alert( 2 - '1' ); // 1  alert( 6 / '2' ); // 3 |

# Приоритет операций

В том случае, если в выражении есть несколько операторов – порядок их выполнения определяется приоритетом.

Из школы мы знаем, что умножение в выражении 2 \* 2 + 1 выполнится раньше сложения, т.к. его приоритет выше, а скобки явно задают порядок выполнения. Но в JavaScript – гораздо больше операторов, поэтому существует целая таблица приоритетов.

Она содержит как уже пройденные операторы, так и те, которые мы еще не проходили. В ней каждому оператору задан числовой приоритет. Тот, у кого число больше – выполнится раньше. Если приоритет одинаковый, то порядок выполнения – слева направо.

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence** | **Operator type** | **Associativity** | **Individual operators** |
| 19 | [Grouping](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Grouping) | n/a | ( … ) |
| 18 | [Member Access](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_Accessors#Dot_notation) | left-to-right | … . … |
| [Computed Member Access](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_Accessors#Bracket_notation) | left-to-right | … [ … ] |
| [new](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new) (with argument list) | n/a | new … ( … ) |
| 17 | [Function Call](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions) | left-to-right | … ( *…*) |
| [new](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new) (without argument list) | right-to-left | new … |
| 16 | [Postfix Increment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Increment) | n/a | … ++ |
| [Postfix Decrement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Decrement) | n/a | … -- |
| 15 | [Logical NOT](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Operators#Logical_NOT) | right-to-left | ! … |
| [Bitwise NOT](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_NOT) | right-to-left | ~ … |
| [Unary Plus](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Unary_plus) | right-to-left | + … |
| [Unary Negation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Unary_negation) | right-to-left | - … |
| [Prefix Increment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Increment) | right-to-left | ++ … |
| [Prefix Decrement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Decrement) | right-to-left | -- … |
| [typeof](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof) | right-to-left | typeof … |
| [void](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/void) | right-to-left | void … |
| [delete](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete) | right-to-left | delete … |
| 14 | [Exponentiation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Exponentiation) | right-to-left | … \*\* … |
| [Multiplication](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Multiplication) | left-to-right | … \* … |
| [Division](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Division) | left-to-right | … / … |
| [Remainder](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Remainder) | left-to-right | … % … |
| 13 | [Addition](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Addition) | left-to-right | … + … |
| [Subtraction](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Subtraction) | left-to-right | … - … |
| 12 | [Bitwise Left Shift](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators) | left-to-right | … << … |
| [Bitwise Right Shift](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators) | left-to-right | … >> … |
| [Bitwise Unsigned Right Shift](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators) | left-to-right | … >>> … |
| 11 | [Less Than](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Less_than_operator) | left-to-right | … < … |
| [Less Than Or Equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Less_than__or_equal_operator) | left-to-right | … <= … |
| [Greater Than](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Greater_than_operator) | left-to-right | … > … |
| [Greater Than Or Equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Greater_than_or_equal_operator) | left-to-right | … >= … |
| [in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/in) | left-to-right | … in … |
| [instanceof](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/instanceof) | left-to-right | … instanceof … |
| 10 | [Equality](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Equality) | left-to-right | … == … |
| [Inequality](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Inequality) | left-to-right | … != … |
| [Strict Equality](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Identity) | left-to-right | … === … |
| [Strict Inequality](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Nonidentity) | left-to-right | … !== … |
| 9 | [Bitwise AND](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_AND) | left-to-right | … & … |
| 8 | [Bitwise XOR](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_XOR) | left-to-right | … ^ … |
| 7 | [Bitwise OR](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_OR) | left-to-right | … | … |
| 6 | [Logical AND](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Operators#Logical_AND) | left-to-right | … && … |
| 5 | [Logical OR](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Operators#Logical_OR) | left-to-right | … || … |
| 4 | [Conditional](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator) | right-to-left | … ? … : … |
| 3 | [Assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators) | right-to-left | … = … |
| … += … |
| … -= … |
| … \*\*= … |
| … \*= … |
| … /= … |
| … %= … |
| … <<= … |
| … >>= … |
| … >>>= … |
| … &= … |
| … ^= … |
| … |= … |
| 2 | [yield](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield) | right-to-left | yield … |
| [yield\*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield*) | right-to-left | yield\* … |
| 1 | [Spread](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator) | n/a | ... … |
| 0 | [Comma / Sequence](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comma_Operator) | left-to-right | … , … |

Обратим внимание, в таблице приоритетов также есть оператор присваивания =.

У него – один из самых низких приоритетов: 3.

Именно поэтому, когда переменную чему-либо присваивают, например, x = 2 \* 2 + 1 сначала выполнится арифметика, а уже затем – произойдёт присвоение =.

|  |
| --- |
| var x = 2 \* 2 + 1;  alert( x ); // 5  // Возможно так же выравнивание по цепочке  var a, b, c;  a = b = c = 2 + 2;  alert( a ); // 4  alert( b ); // 4  alert( c ); // 4 |

Такое присваивание работает справа-налево, то есть сначала вычислятся самое правое выражение2+2, присвоится в c, затем выполнится b = c и, наконец, a = b.

Одной из наиболее частых операций в JavaScript, как и во многих других языках программирования, является увеличение или уменьшение переменной на единицу.

Для этого существуют даже специальные операторы:

Инкремент ++ увеличивает на 1:

|  |
| --- |
| var i = 2;  i++; // более короткая запись для i = i + 1.  alert(i); // 3 |

Декремент -- уменьшает на 1:

|  |
| --- |
| var i = 2;  i--; // более короткая запись для i = i - 1.  alert(i); // 1 |

Вызывать эти операторы можно не только после, но и перед переменной: i++ (называется «постфиксная форма») или ++i («префиксная форма»).

Обе эти формы записи делают одно и то же: увеличивают на 1.

Тем не менее, между ними существует разница. Она видна только в том случае, когда мы хотим не только увеличить/уменьшить переменную, но и использовать результат в том же выражении.

**Постфиксная форма i++ отличается от префиксной ++i тем, что возвращает старое значение, бывшее до увеличения.**

# Побитовые операторы

Побитовые операторы рассматривают аргументы как 32-разрядные целые числа и работают на уровне их внутреннего двоичного представления.

Эти операторы не являются чем-то специфичным для JavaScript, они поддерживаются в большинстве языков программирования.

Побитовые операторы в JavaScript работают с 32-битными целыми числами в их двоичном представлении.

Это представление называется «32-битное целое со знаком, старшим битом слева и дополнением до двойки».

Разберём, как устроены числа внутри подробнее, это необходимо знать для битовых операций с ними.

Что такое двоичная система счисления, вам, надеюсь, уже известно. При разборе побитовых операций мы будем обсуждать именно двоичное представление чисел, из 32 бит.

Старший бит слева – это научное название для самого обычного порядка записи цифр (от большего разряда к меньшему). При этом, если больший разряд отсутствует, то соответствующий бит равен нулю.

Примеры представления чисел в двоичной системе:

|  |
| --- |
| a = 0; // 00000000000000000000000000000000  a = 1; // 00000000000000000000000000000001  a = 2; // 00000000000000000000000000000010  a = 3; // 00000000000000000000000000000011  a = 255;// 00000000000000000000000011111111 |

Обратите внимание, каждое число состоит ровно из 32-битов.

Дополнение до двойки – это название способа поддержки отрицательных чисел.

Двоичный вид числа, обратного данному (например, 5 и -5) получается путём обращения всех битов с прибавлением 1.

То есть, нули заменяются на единицы, единицы – на нули и к числу прибавляется 1. Получается внутреннее представление того же числа, но со знаком минус.

В следующей таблице перечислены все побитовые операторы. Далее операторы разобраны более подробно

|  |  |  |
| --- | --- | --- |
| **Оператор** | **Использование** | **Описание** |
| Побитовое И (AND) | a & b | Ставит 1 на бит результата, для которого соответствующие биты операндов равны 1. |
| Побитовое ИЛИ (OR) | a | b | Ставит 1 на бит результата, для которого хотя бы один из соответствующих битов операндов равен 1. |
| Побитовое исключающее ИЛИ (XOR) | a ^ b | Ставит 1 на бит результата, для которого только один из соответствующих битов операндов равен 1 (но не оба). |
| Побитовое НЕ (NOT) | ~a | Заменяет каждый бит операнда на противоположный. |
| Левый сдвиг | `a << b` | Сдвигает двоичное представление a на b битов влево, добавляя справа нули. |
| Правый сдвиг, переносящий знак | `a >> b` | Сдвигает двоичное представление a на b битов вправо, отбрасывая сдвигаемые биты. |
| Правый сдвиг с заполнением нулями | `a >>> b` | Сдвигает двоичное представление a на b битов вправо, отбрасывая сдвигаемые биты и добавляя нули слева. |

Побитовые операторы работают следующим образом:

Операнды преобразуются в 32-битные целые числа, представленные последовательностью битов. Дробная часть, если она есть, отбрасывается.

Для бинарных операторов – каждый бит в первом операнде рассматривается вместе с соответствующим битом второго операнда: первый бит с первым, второй со вторым и т.п. Оператор применяется к каждой паре бит, давая соответствующий бит результата.

Получившаяся в результате последовательность бит интерпретируется как обычное число.

# Операторы сравнения

Как и другие операторы, сравнение возвращает значение. Это значение имеет логический тип.

Существует всего два логических значения:

true – имеет смысл «да», «верно», «истина».

false – означает «нет», «неверно», «ложь».

Например:

|  |
| --- |
| alert( 2 > 1 ); // true, верно  alert( 2 == 1 ); // false, неверно  alert( 2 != 1 ); // true |

Строки сравниваются побуквенно:

|  |
| --- |
| alert( 'Б' > 'А' ); // true |

Если строка состоит из нескольких букв, то сравнение осуществляется как в телефонной книжке или в словаре. Сначала сравниваются первые буквы, потом вторые, и так далее, пока одна не будет больше другой.

Иными словами, больше – та строка, которая в телефонной книге была бы на большей странице.

При сравнении значений разных типов, используется числовое преобразование. Оно применяется к обоим значениям.

Например:

|  |
| --- |
| alert( '2' > 1 ); // true, сравнивается как 2 > 1  alert( '01' == 1 ); // true, сравнивается как 1 == 1  alert( false == 0 ); // true, false становится числом 0  alert( true == 1 ); // true, так как true становится числом 1. |

В обычном операторе == есть «проблема» – он не может отличить 0 от false:

|  |
| --- |
| alert( 0 == false ); // true  // Та же ситуация с пустой строкой  alert( '' == false ); // true |

Это естественное следствие того, что операнды разных типов преобразовались к числу. Пустая строка, как и false, при преобразовании к числу дают 0.

Что же делать, если всё же нужно отличить 0 от false?

Для проверки равенства без преобразования типов используются операторы строгого равенства === (тройное равно) и !==.

Если тип разный, то они всегда возвращают false:

|  |
| --- |
| alert( 0 === false ); // false, т.к. типы различны |

Проблемы со специальными значениями возможны, когда к переменной применяется операция сравнения > < <= >=, а у неё может быть как численное значение, так и null/undefined.

Интуитивно кажется, что null/undefined эквивалентны нулю, но это не так.

Они ведут себя по-другому.

Значения null и undefined равны == друг другу и не равны чему бы то ни было ещё. Это жёсткое правило буквально прописано в спецификации языка.

При преобразовании в число null становится 0, а undefined становится NaN.

Посмотрим забавные следствия.

Некорректный результат сравнения null с 0

Сравним null с нулём:

|  |
| --- |
| alert( null > 0 ); // false  alert( null == 0 ); // false |

Итак, мы получили, что null не больше и не равен нулю. А теперь…

|  |
| --- |
| alert(null >= 0); // true |

Как такое возможно? Если нечто «больше или равно нулю», то резонно полагать, что оно либо больше, либо равно. Но здесь это не так.

Дело в том, что алгоритмы проверки равенства == и сравнения >= > < <= работают по-разному.

Сравнение честно приводит к числу, получается ноль. А при проверке равенства значения null и undefined обрабатываются особым образом: они равны друг другу, но не равны чему-то ещё.

В результате получается странная с точки зрения здравого смысла ситуация, которую мы видели в примере выше.

Несравнимый undefined

Значение undefined вообще нельзя сравнивать:

|  |
| --- |
| alert( undefined > 0 ); // false (1)  alert( undefined < 0 ); // false (2)  alert( undefined == 0 ); // false (3) |

Сравнения (1) и (2) дают false потому, что undefined при преобразовании к числу даёт NaN. А значение NaN по стандарту устроено так, что сравнения ==, <, >, <=, >= и даже === с ним возвращают false.

Проверка равенства (3) даёт false, потому что в стандарте явно прописано, что undefined равно лишь null и ничему другому.

Вывод: любые сравнения с undefined/null, кроме точного ===, следует делать с осторожностью.

Желательно не использовать сравнения >= > < <= с ними, во избежание ошибок в коде.

# Домашнее задание

* Есть такой код:

|  |
| --- |
| var a = 1, b = 1, c, d;  c = ++a; alert(c); // 2  d = b++; alert(d); // 1  c = (2+ ++a); alert(c); // 5  d = (2+ b++); alert(d); // 4  alert(a); // 3  alert(b); // 3 |

Необходимо разобраться почему работает именно так. Объяснения написать в виде комментария к каждой строчке

* Чему будет равен x в примере ниже?

|  |
| --- |
| var a = 2;  var x = 1 + (a \*= 2) |

* \* Забегая немного вперед. Напишите программу, которая выводит через console.log все числа от 1 до 100, с двумя исключениями. Для чисел, нацело делящихся на 3, она должна выводить ‘Fizz’, а для чисел, делящихся на 5 (но не на 3) – ‘Buzz’.Когда сумеете – исправьте её так, чтобы она выводила «FizzBuzz» для всех чисел, которые делятся и на 3 и на 5.

# Используемая литература

* [Mozilla Developer Network](https://developer.mozilla.org/)
* [MSDN](http://msdn.microsoft.com/)
* [Safari Developer Library](https://developer.apple.com/library/safari/navigation/index.html)
* [Современный учебник JavaScript](http://learn.javascript.ru)