MDN

> Эта страница была переведена с английского языка силами сообщества. Вы тоже можете внести свой вклад, присоединившись к русскоязычному сообществу MDN Web Docs.

Выражения и операторы

Эта глава описывает выражения и операторы языка JavaScript, такие как операторы присваивания, сравнения, арифметические, битовые, логические, строчные, и различные специальные операторы.

Полный и детальный список операторов и выражений также доступен в этом руководстве.

Операторы

В JavaScript есть следующие типы операторов. Данный подраздел описывает каждый тип и содержит информацию об их приоритетах друг над другом.

- Операторы присваивания
- Операторы сравнения
- Арифметические операторы
- Битовые (поразрядные) операторы
- Логические операторы
- Строковые операторы
- Условный (тернарный) оператор
- Оператор запятая
- Унарные операторы

- Операторы отношения
- Приоритет операторов

JavaScript поддерживает бинарные и унарные операторы, а также ещё один специальный тернарный оператор - условный оператор. Бинарная операция использует два операнда, один перед оператором и другой за ним:

```
operand1 operator operand2
```

Например: `3+4` или `x*y`.

В свою очередь унарная операция использует один операнд, перед или после оператора:

```
operator operand
```

или

```
operand operator
```

Например: `x++` или `++x`.

Операторы присваивания

В результате операции присваивания операнду слева от оператора присваивания (en-US) (знак "=") устанавливается значение , которое берётся из правого операнда. Основным оператором присваивания является =, он присваивает значение правого операнда операнду, находящемуся слева. Таким образом, выражение x = y означает, что x присваивается значение y.

Существуют также составные операторы присваивания, которые используются для сокращённого представления операций, описанных в следующей таблице:

Имя	Сокращённый оператор	Смысл
Присваивание (en-US)	`x = y`	`x = y`
Присваивание со сложением (en-US)	`x += y`	`x = x + y`
Присваивание с вычитанием (en-US)	`x -= y`	`x = x - y`
Присваивание с умножением (en-US)	`x *= y`	`x = x * y`
Присваивание с делением (en-US)	`x /= y`	`x = x / y`
Присваивание по модулю (en-US)	`x %= y`	`x = x % y`
Присваивание с левым сдвигом (en-US)	`x <<= y`	`x = x << y`
Присваивание с правым сдвигом (en-US)	`x >>= y`	`x = x >> y`
Присваивание с беззнаковым сдвигом вправо (en-US)	`x >>>= y`	`x = x >>> y`
Присваивание с побитовым AND (en-US)	`x &= y`	`x = x & y`
Присваивание с побитовым XOR (en-US)	`x ^= y`	`x = x ^ y`
Присваивание с побитовым OR (en-US)	`x |= y`	`x = x | y`

Деструктуризация

Для более сложного присваивания в JavaScript есть синтаксис деструктуризации - это выражение, которое позволяет извлекать данные из массивов или объектов, используя синтаксис, который зеркалирует конструкторы массивов и литералы объектов.

JS

```
var foo = ["one", "two", "three"];

// без деструктуризации
var one = foo[0];
var two = foo[1];
var three = foo[2];
```

```
// с деструктуризацией
var [one, two, three] = foo;
```

Операторы сравнения

Оператор сравнения (en-US) сравнивает свои операнды и возвращает логическое значение, базируясь на истинности сравнения. Операнды могут быть числами, строками, логическими величинами или объектами. Строки сравниваются на основании стандартного лексикографического порядка, используя Unicode-значения. В большинстве случаев, если операнды имеют разный тип, то JavaScript пробует преобразовать их в тип, подходящий для сравнения. Такое поведение обычно происходит при сравнении числовых операндов. Единственным исключением из данного правила является сравнение с использованием операторов `===` и `!==`, которые производят строгое сравнение на равенство или неравенство. Эти операторы не пытаются преобразовать операнды перед их сравнением. Следующая таблица описывает операторы сравнения в контексте следующего примера кода:

JS

```
var var1 = 3,
  var2 = 4;
```

Оператор	Описание	Примеры, возвращающие true
Равно (==)	Возвращает true, если операнды равны.	3 == var1 "3" == var1 3 == '3'
Не равно (!=)	Возвращает true, если операнды не равны.	var1 != 4 var2 != "3"
Строго равно (===)	Возвращает true, если операнды равны и имеют одинаковый тип. Смотрите также Object.is и sameness in JS.	3 === var1
Строго не равно(!==)	Возвращает true, если операнды не равны и/или имеют разный тип.	var1 !== "3" 3 !== '3'
Больше (>)	Возвращает true, если операнд слева больше операнда справа.	var2 > var1 "12" > 2

Оператор	Описание	Примеры, возвращающие true
Больше или равно (>=)	Возвращает true, если операнд слева больше или равен операнду справа.	var2 >= var1 var1 >= 3
Меньше (<)	Возвращает true, если операнд слева меньше операнда справа.	var1 < var2 "2" < 12
Меньше или равно (<=)	Возвращает true, если операнд слева меньше или равен операнду справа.	var1 <= var2 var2 <= 5

> Примечание: (=>) не оператор, а нотация Стрелочных функций.

Арифметические операторы

Арифметические операторы (en-US) используют в качестве своих операндов числа (также литералы или переменные) и в качестве результата возвращают одно числовое значение. Стандартными арифметическими операторами являются сложение (+), вычитание (-), умножение (*), и деление (/). При работе с числами с плавающей точкой эти операторы работают аналогично их работе в большинстве других языках программирования (обратите внимание, что деление на ноль возвращает бесконечность `Infinity`). Например:

JS

```
console.log(1 / 2); /* возвращает 0.5 */
console.log(1 / 2 == 1.0 / 2.0); /* возвращает true */
```

Кроме того, JavaScript позволяет использовать следующие арифметические операторы, представленные в таблице:

Оператор	Описание	Пример
Остаток от деления (en-US) (%)	Бинарный оператор. Возвращает целочисленный остаток от деления двух операндов.	12 % 5 вернёт 2.

Оператор	Описание	Пример
Инкремент (en-US) (++)	Унарный оператор. Добавляет единицу к своему операнду. Если используется в качестве префикса (++x), то возвращает значение операнда с добавленной к нему единицей; а в случае применения в качестве окончания (x++) возвращает значение операнда перед добавлением к нему единицы.	Если x равно 3, тогда ++x установит значение x равным 4 и вернёт 4, напротив x++ вернёт 3 и потом установит значение x равным 4.
Декремент (en-US) (--)	Унарный оператор. Вычитает единицу из значения своего операнда. Логика данного оператора аналогична оператору инкремента.	Если x равно 3, тогда --x установит значение x равным 2 и вернёт 2, напротив x-- вернёт 3 и потом установит значение x равным 2.
Унарный минус (en-US) -	Унарный оператор. Возвращает отрицательное значение своего операнда.	Если x равно 3, тогда -x вернёт -3.
Унарный плюс (en-US) (+)	Унарный оператор. Пытается конвертировать операнд в число, если он ещё не оно.	+"3" вернёт 3 . +true вернёт 1.
Возведение в степень (en-US) (**)	Возводит основание в показатель степени , как, основаниестепень	2 ** 3 вернёт 8 . 10 ** -1 вернёт 0.1 .

Битовые (поразрядные) операторы

Битовые операторы (en-US) обрабатывают свои операнды как последовательности из 32 бит (нулей и единиц), а не как десятичные, шестнадцатеричные или восьмеричные числа. Например, десятичное число 9 имеет двоичное представление 1001. Битовые операторы выполняют операции над таким двоичным представлением, но результат возвращают как обычное числовое значение JavaScript.

Следующая таблица обобщает битовые операторы JavaScript.

Оператор	Использование	Описание
Побитовое И (en-US)	`a & b`	Возвращает единицу в каждой битовой позиции, для которой соответствующие биты обеих операндов являются единицами.
Побитовое ИЛИ (en-US)	`a | b`	Возвращает единицу в каждой битовой позиции, для которой один из соответствующих битов или оба бита операндов являются единицами.
Исключающее ИЛИ (en-US)	`a ^ b`	Возвращает единицу в каждой битовой позиции, для которой только один из соответствующих битов операндов является единицей.
Побитовое НЕ (en-US)	`~ a`	Заменяет биты операнда на противоположные.
Сдвиг влево (en-US)	`a << b`	Сдвигает `a` в двоичном представлении на `b` бит влево, добавляя справа нули.
Сдвиг вправо с переносом знака (en-US)	`a >> b`	Сдвигает `a` в двоичном представлении на `b` бит вправо, отбрасывая сдвигаемые биты.
Сдвиг вправо с заполнением нулями (en-US)	`a >>> b`	Сдвигает `a` в двоичном представлении на `b` бит вправо, отбрасывая сдвигаемые биты и добавляя слева нули.

Битовые логические операторы

Основной смысл работы битовых логических операторов состоит в следующем:

- Операнды преобразуются в 32-битные целые числа и представляются в виде последовательности бит (нулей и единиц). Числа, имеющие более 32 битов будут сокращены. Например, следующее число имеет больше 32 битов и сконвертируется в 32-х битное:

```
До     : 11100110111110100000000000000110000000000001
После  :             10100000000000000110000000000001
```

- Каждый бит первого операнда связывается с соответствующим битом второго операнда: первый бит с первым битом, второй бит - со вторым, и так далее.
- К каждой паре бит применяется заданный оператор, и побитово формируется итоговый результат.

Например, двоичным представлением числа 9 является 1001, а двоичным представлением пятнадцати - 1111. Результаты применения к этим числам битовых логических операторов выглядят следующим образом:

Выражение	Результат	Двоичное описание
15 & 9	9	1111 & 1001 = 1001
15 | 9	15	1111 | 1001 = 1111
15 ^ 9	6	1111 ^ 1001 = 0110
~15	-16	~``00000000...``00001111 = ``1111``1111``...``11110000
~9	-10	~``00000000``...``0000``1001 = ``1111``1111``...``1111``0110

Обратите внимание, что все 32 бита преобразуются с использованием битового оператора НЕ, и что величины с наиболее значимым (самым левым) битом равным 1 представляют собой отрицательные числа (в представлении дополнения до двух).

Битовые операторы сдвига

Битовые операторы сдвига используют два операнда: первый представляет величину, подлежащую сдвигу, а второй операнд указывает число битовых позиций на которое должен быть сдвинут первый операнд. Направление операции сдвига определяется используемым оператором.

Операторы сдвига преобразуют свои операнды в 32-битные целые числа и возвращают результат того же типа, каким является левый операнд.

Операторы сдвига перечислены в следующей таблице.

Оператор	Описание	Пример
Сдвиг влево (en-US) (<<)	Данный оператор сдвигает первый операнд на указанное количество бит влево. Излишние биты, сдвинутые влево, отбрасываются. Справа число дополняется нулевыми битами.	`9<<2` равно 36, так как 1001 после сдвига на 2 бита влево превращается в 100100, что соответствует числу 36.
Сдвиг вправо с переносом знака (en-US) (>>)	Данный оператор сдвигает первый операнд на указанное количество бит вправо. Излишние биты, сдвинутые вправо, отбрасываются. Слева число дополняется копиями крайнего слева бита.	`9>>2` равно 2, так как 1001 после сдвига на 2 бита вправо превращается в 10, что соответствует числу 2. Подобным же образом `-9>>2` равно -3, так как знак сохраняется.
Сдвиг вправо с заполнением нулями (en-US) (>>>)	Данный оператор сдвигает первый операнд на указанное количество бит вправо. Излишние биты, сдвинутые вправо, отбрасываются. Слева число дополняется нулевыми битами.	`19>>>2` равно 4, так как 10011 после сдвига на 2 бита вправо превращается в 100, что соответствует числу 4. Для неотрицательных чисел сдвиг вправо с заполнением нулями и сдвиг вправо с переносом знака дают одинаковый результат.

Логические операторы

Логические операторы (en-US) обычно используются с булевыми (логическими) значениями; при этом возвращаемое ими значение также является булевым. Однако операторы && и || фактически возвращают значение одного из операндов, поэтому, если эти операторы используются с небулевыми величинами, то возвращаемая ими величина также может быть не булевой. Логические операторы описаны в следующей таблице.

Оператор	Использование	Описание
Логическое И (en-US) (&&)	`expr1 && expr2`	(Логическое И) Возвращает операнд `expr1`, если он может быть преобразован в `false`; в противном случае возвращает операнд `expr2`. Таким образом, при использовании булевых величин в качестве операндов, оператор `&&` возвращает `true`, если оба операнда `true`; в противном случае возвращает `false`.
Логическое ИЛИ (en-US) (||)	`expr1 || expr2`	(Логическое ИЛИ) Возвращает операнд `expr1`, если он может быть преобразован в `true`; в противном случае возвращает операнд `expr2`. Таким образом, при использовании булевых величин в качестве операндов, оператор `||` возвращает `true`, если один из операндов `true`; если же оба `false`, то возвращает `false`.
Логическое НЕ (en-US) (!)	`!expr`	(Логическое НЕ) Возвращает `false`, если операнд может быть преобразован в `true`; в противном случае возвращает `true`.

Примерами выражений, которые могут быть преобразованы в false являются: null, 0, NaN, пустая строка ("") или undefined.

Следующий код демонстрирует примеры использования оператора && (логическое И).

JS

```
var a1 = true && true; // t && t возвращает true
var a2 = true && false; // t && f возвращает false
var a3 = false && true; // f && t возвращает false
var a4 = false && 3 == 4; // f && f возвращает false
var a5 = "Cat" && "Dog"; // t && t возвращает Dog
var a6 = false && "Cat"; // f && t возвращает false
var a7 = "Cat" && false; // t && f возвращает false
```

Следующий код демонстрирует примеры использования оператора || (логическое ИЛИ).

JS

```
var o1 = true || true; // t || t возвращает true
var o2 = false || true; // f || t возвращает true
var o3 = true || false; // t || f возвращает true
var o4 = false || 3 == 4; // f || f возвращает false
var o5 = "Cat" || "Dog"; // t || t возвращает Cat
var o6 = false || "Cat"; // f || t возвращает Cat
var o7 = "Cat" || false; // t || f возвращает Cat
```

Следующий код демонстрирует примеры использования оператора ! (логическое НЕ).

JS

```
var n1 = !true; // !t возвращает false
var n2 = !false; // !f возвращает true
var n3 = !"Cat"; // !t возвращает false
```

Сокращённая оценка

Так как логические выражения вычисляются слева направо, они проверяются на возможность выполнения сокращённой оценки с использованием следующих правил:

- `false && anything` - сокращение с результатом false.
- `true || anything` - сокращение с результатом true.

Правила логики гарантируют, что данные вычисления всегда корректны. Обратите внимание, что часть "*anything*" представленных выше выражений не вычисляется, таким образом удаётся избежать любых побочных эффектов вычисления данной части.

Строковые операторы

В дополнение к операторам сравнения, которые могут использоваться со строковыми значениями, оператор (+) позволяет объединить две строки, возвращая при этом третью строку, которая представляет собой объединение двух строк-операндов:

```
JS
```

```
console.log("my " + "string"); // в консоли выведется строка  "my string".
```

Сокращённый оператор присваивания += также может быть использован для объединения (конкатенации) строк:

```
JS
```

```
var mystring = "alpha";
mystring += "bet"; // получается значение "alphabet" и присваивается mystring.
```

Условный (тернарный) оператор

Условный оператор является единственным оператором JavaScript, который использует три операнда. Оператор принимает одно из двух значений в зависимости от заданного условия. Синтаксис оператора:

```
condition ? val1 : val2
```

> Предупреждение: *val1 и val2 обязательно должны что-то возвращать, поэтому в этой конструкции нельзя использовать continue или break*

Если `condition (условие)` - истина, то оператор принимает значение `val1`. В противном случае оператор принимает значение `val2`. Вы можете использовать условный оператор во всех случаях, где может быть использован стандартный оператор.

```
JS
```

```
var status = age >= 18 ? "adult" : "minor";
```

Данное выражение присваивает значение "adult" переменной `status`, если `age` имеет значение 18 или более. В противном случае переменной `status` присваивается значение "minor".

Оператор запятая

Оператор запятая (,) просто вычисляет оба операнда и возвращает значение последнего операнда. Данный оператор в основном используется внутри цикла `for`, что позволяет при каждом прохождении цикла одновременно обновлять значения нескольких переменных.

Например, если `a` является двумерным массивом, каждая строка которого содержит 10 элементов, то следующий код с использованием оператора запятая позволяет выполнять одновременное приращение двух переменных. Данный код выводит на экран значения диагональных элементов массива:

```
JS
for (var i = 0, j = 9; i <= 9; i++, j--)
  document.writeln("a[" + i + "][" + j + "]= " + a[i][j]);
```

Унарные операторы

Унарная операция - операция только с одним операндом.

delete

Оператор `delete` выполняет удаление объекта, свойства объекта, или элемента массива с заданным индексом. Синтаксис оператора:

```
JS
delete objectName;
delete objectName.property;
delete objectName[index];
delete property; // допустимо только внутри with
```

где `objectName` представляет собой имя объекта, `property` - свойство объекта, а `index` - целое число, указывающее на положение (номер позиции) элемента в массиве.

Четвёртый вариант использования позволяет удалить свойство объекта, но допускается только внутри `with`.

Вы можете использовать оператор `delete` для удаления переменных, объявленных неявно, но вы не можете с его помощью удалять переменные, объявленные с помощью `var`.

После применения оператора `delete` свойство элемента меняется на `undefined`. Оператор `delete` возвращает `true` если выполнение операции возможно; оператор возвращает `false`, если выполнение операции невозможно.

JS

```
x = 42;
var y = 43;
myobj = new Number();
myobj.h = 4; // создаём свойство h
delete x; // возвращает true (можно удалить переменную объявленную неявно)
delete y; // возвращает false (нельзя удалить переменную объявленную с помощью var)
delete Math.PI; // возвращает false (нельзя удалить встроенные свойства)
delete myobj.h; // возвращает true (можно удалить пользовательские свойства)
delete myobj; // возвращает true (можно удалить объект объявленный неявно)
```

УДАЛЕНИЕ ЭЛЕМЕНТОВ МАССИВА

Удаление элемента массива не влияет на длину массива. Например, если вы удалите `a[3]`, элемент `a[4]` останется `a[4]`, `a[3]` станет undefined.

Когда элемент массива удаляется с помощью оператора `delete`, то из массива удаляется значение данного элемента. В следующем примере элемент `trees[3]` удалён с помощью оператора `delete`. Однако, элемент `trees[3]` остаётся адресуемым и возвращает значение `undefined`.

JS

```
var trees = new Array("redwood", "bay", "cedar", "oak", "maple");
delete trees[3];
if (3 in trees) {
  // условие не выполняется
}
```

Если вы хотите, чтобы элемент оставался в массиве, но имел значение undefined, то используйте ключевое слово `undefined` вместо оператора `delete`. В следующем

примере элементу `trees[3]` присвоено значение `undefined`, но элемент при этом остаётся в массиве:

```
JS
var trees = new Array("redwood", "bay", "cedar", "oak", "maple");
trees[3] = undefined;
if (3 in trees) {
  // данный блок кода выполняется
}
```

Оператор typeof

Оператор `typeof` используется одним из следующих способов:

```
typeof operand
typeof (operand)
```

Оператор `typeof` возвращает строку обозначающую тип невычисленного операнда. Значение `operand` может быть строкой, переменной, дескриптором, или объектом, тип которого следует определить. Скобки вокруг операнда необязательны.

Предположим, вы определяете следующие переменные:

```
JS
var myFun = new Function("5 + 2");
var shape = "round";
var size = 1;
var today = new Date();
```

Оператор `typeof` возвращает следующие результаты для этих переменных:

```
JS
typeof myFun; // возвращает "function"
typeof shape; // возвращает "string"
typeof size; // возвращает "number"
typeof today; // возвращает "object"
typeof dontExist; // возвращает "undefined"
```

Для дескрипторов `true` и `null` оператор `typeof` возвращает следующие результаты:

```
JS
```

```
typeof true; // возвращает "boolean"
typeof null; // возвращает "object"
```

Для чисел и строк оператор `typeof` возвращает следующие результаты:

```
JS
```

```
typeof 62; // возвращает "number"
typeof "Hello world"; // возвращает "string"
```

Для свойств оператор `typeof` возвращает тип значения данного свойства:

```
JS
```

```
typeof document.lastModified; // возвращает "string"
typeof window.length; // возвращает "number"
typeof Math.LN2; // возвращает "number"
```

Для методов и функций оператор `typeof` возвращает следующие результаты:

```
JS
```

```
typeof blur; // возвращает "function"
typeof eval; // возвращает "function"
typeof parseInt; // возвращает "function"
typeof shape.split; // возвращает "function"
```

Для встроенных объектов оператор `typeof` возвращает следующие результаты:

```
JS
```

```
typeof Date; // возвращает "function"
typeof Function; // возвращает "function"
typeof Math; // возвращает "object"
typeof Option; // возвращает "function"
typeof String; // возвращает "function"
```

Оператор void

Оператор `void` используется любым из следующих способов:

```
void (expression)
void expression
```

Оператор `void` определяет выражение, которое должно быть вычислено без возвращения результата. `expression` - это выражение JavaScript, требующее вычисления. Скобки вокруг выражения необязательны, но их использование является правилом хорошего тона.

Вы можете использовать оператор `void` для указания на то, что операнд-выражение является гипертекстовой ссылкой. При этом выражение обрабатывается, но не загружается в текущий документ.

Следующий код служит примером создания гипертекстовой ссылки, которая бездействует при нажатии на неё пользователем. Когда пользователь нажимает на ссылку, `void(0)` вычисляется равным `undefined`, что не приводит ни к каким действиям в JavaScript.

HTML

```
<a href="javascript:void(0)">Нажмите здесь, чтобы ничего не произошло</a>
```

Приведённый ниже код создаёт гипертекстовую ссылку, которая подтверждает отправку формы при клике на ней пользователем:

HTML

```
<a href="javascript:void(document.form.submit())">
  Нажмите здесь, чтобы подтвердить отправку формы</a
>
```

Операторы отношения

Оператор отношения сравнивает свои операнды и возвращает результат сравнения в виде булева значения.

Оператор in

Оператор `in` возвращает true, если указанный объект имеет указанное свойство. Синтаксис оператора:

```
propNameOrNumber in objectName
```

где `propNameOrNumber` - строка или числовое выражение, представляющее имя свойства или индекс массива, а `objectName` - имя объекта.

Некоторые примеры способов использования оператора `in`:

```JS
// Массивы
var trees = new Array("redwood", "bay", "cedar", "oak", "maple");
0 in trees; // возвращает true
3 in trees; // возвращает true
6 in trees; // возвращает false
"bay" in trees; // возвращает false (следует указать индекс элемента массива,
// а не значение элемента)
"length" in trees; // возвращает true (length является свойством объекта Array)

// Встроенные объекты
"PI" in Math; // возвращает true
var myString = new String("coral");
"length" in myString; // возвращает true

// Пользовательские объекты
var mycar = { make: "Honda", model: "Accord", year: 1998 };
"make" in mycar; // возвращает true
"model" in mycar; // возвращает true
```

Оператор instanceof

Оператор `instanceof` возвращает true, если заданный объект является объектом указанного типа. Его синтаксис:

```
objectName instanceof objectType
```

где `objectName` - имя объекта, тип которого необходимо сравнить с `objectType`, а `objectType` - тип объекта, например, `Date` или `Array`.

Используйте оператор instanceof, когда вам необходимо подтвердить тип объекта во время выполнения программы. Например, при перехвате исключений вы можете создать различные программные переходы для обработки исключений в зависимости от типа обрабатываемого исключения.

Например, следующий код использует оператор instanceof для проверки того, является ли объект theDay объектом типа Date. Так как theDay действительно является объектом типа Date, то программа выполняет код, содержащийся в утверждении if.

JS

```
var theDay = new Date(1995, 12, 17);
if (theDay instanceof Date) {
  // выполняемый код
}
```

Приоритет операторов

Приоритет операторов определяет порядок их выполнения при вычислении выражения. Вы можете влиять на приоритет операторов с помощью скобок.

Приведённая ниже таблица описывает приоритет операторов от наивысшего до низшего.

Тип оператора	Операторы
свойство объекта	. []
вызов, создание экземпляра объекта	() new
отрицание, инкремент	! ~ - + ++ -- typeof void delete
умножение, деление	* / %
сложение, вычитание	+ -
побитовый сдвиг	<< >> >>>
сравнение, вхождение	< <= > >= in instanceof

Тип оператора	Операторы
равенство	`== != === !==`
битовое-и	`&`
битовое-исключающее-или	`^`
битовое-или	`|`
логическое-и	`&&`
логическое-или	`||`
условный (тернарный) оператор	`?:`
присваивание	`= += -= *= /= %= <<= >>= >>>= &= ^= |=`
запятая	`,`

Более подробная версия данной таблицы, содержащая ссылки и дополнительную информацию по каждому оператору, находится в справочнике JavaScript.

Выражения

Выражением является любой корректный блок кода, который возвращает значение.

Концептуально, существуют два типа выражений: те которые присваивают переменной значение, и те, которые вычисляют значение без его присваивания.

Выражение `x = 7` является примером выражения первого типа. Данное выражение использует *оператор* = для присваивания переменной `x` значения 7. Само выражение также равняется 7.

Код `3 + 4` является примером выражения второго типа. Данное выражение использует *оператор* "+" для сложения чисел 3 и 4 без присваивания переменной полученного результата 7.

Все выражения в JavaScript делятся на следующие категории:

- Арифметические: вычисляются в число, например: 3.14159 (Используют арифметические операторы).
- Строковые: вычисляются в текстовую строку, например: "Fred" или "234" (Используют строковые операторы).
- Логические: вычисляются в true или false (Используют логические операторы).
- Основные выражения: Базовые ключевые слова и основные выражения в JavaScript.
- Левосторонние выражения: Значениям слева назначаются значения справа.

Основные выражения

Базовые ключевые слова и основные выражения в JavaScript.

Оператор this

Используйте ключевое слово `this` для указания на текущий объект. В общем случае `this` указывает на вызываемый объект, которому принадлежит данный метод. Используйте `this` следующим образом:

```
this["propertyName"]
this.propertyName
```

Предположим, функция `validate` выполняет проверку свойства `value` некоторого объекта; задан объект, а также верхняя и нижняя граница величины данного свойства:

JS

```
function validate(obj, lowval, hival) {
  if (obj.value < lowval || obj.value > hival) alert("Неверное значение!");
}
```

Вы можете вызвать функцию `validate` для обработчика события `onChange` для каждого элемента формы, используя `this` для указания на элемент формы, как это показано в следующем примере:

HTML

```
<b>Введите число от 18 до 99:</b>
<input type="text" name="age" size="3" onChange="validate(this, 18, 99);" />
```

Оператор группировки

Оператор группировки "скобки" ( ) контролирует приоритет вычислений в выражениях. Например, вы можете переопределить порядок так, чтобы сложение выполнялось до умножения:

JS

```
var a = 1;
var b = 2;
var c = 3;

// обычный порядок
a + b * c; // 7
// по умолчанию выполняется так
a + (b * c); // 7

// теперь поменяем приоритет с помощью скобок,
// чтобы сложение выполнялось до умножения
(a + b) * c; // 9

// что эквивалентно следующему
a * c + b * c; // 9
```

Упрощённый синтаксис создания массивов и генераторов

Упрощённый синтаксис - экспериментальная возможность JavaScript, которая возможно будет добавлена в будущие версии ECMAScript. Есть 2 версии синтаксиса:

[for (x of y) x] (en-US)

Упрощённый синтаксис для массивов.

(for (x of y) y) (en-US)

Упрощённый синтаксис для генераторов.

Упрощённые синтаксисы существуют во многих языках программирования и позволяют вам быстро собирать новый массив, основанный на существующем.

Например:

```
JS
[for (i of [ 1, 2, 3 ]) i*i ];
// [ 1, 4, 9 ]

var abc = [ "A", "B", "C" ];
[for (letters of abc) letters.toLowerCase()];
// [ "a", "b", "c" ]
```

Левосторонние выражения

Значениям слева назначаются значения справа.

new

Вы можете использовать оператор `new` для создания экземпляра объекта пользовательского типа или одного из встроенных объектов. Используйте оператор `new` следующим образом:

```
JS
var objectName = new objectType([param1, param2, ..., paramN]);
```

super

Ключевое слово используется, чтобы вызывать функции родительского объекта. Это полезно и с классами для вызова конструктора родителя, например.

```
JS
super([arguments]); // вызывает конструктор родителя. super.functionOnParent([arguments]);
```

Оператор расширения

Оператор расширения позволяет выражению расширяться в местах с множеством аргументов (для вызовов функций) или множестве элементов (для массивов).

Пример: Сегодня, если у вас есть массив и вы хотите создать новый с существующей частью первого, то литерального синтаксиса массива уже не достаточно, и вы

должны писать императивный (без вариантов) код, используя комбинацию `push`, `splice`, `concat` и т.д. Но с этим оператором код становится более коротким:

JS

```
var parts = ["shoulder", "knees"];
var lyrics = ["head", ...parts, "and", "toes"];
```

Похожим образом оператор работает с вызовами функций:

JS

```
function f(x, y, z) {}
var args = [0, 1, 2];
f(...args);
```

This page was last modified on 7 авг. 2023 г. by MDN contributors.