JSDG 🦏 | Expressions and Operators | Выражения и операции | chapter 4

`JavaScript: The Definitive Guide 7th EDITION • Master the World's Most-Used Programming Language • David Flanagan • 2021`

Выражение — это синтаксическая конструкция JavaScript (или фраза), которая может быть *вычислена* для выдачи значения. Константа, литерально встроенная в программу, является очень простым видом выражения. Имя переменной — тоже простое выражение, которое вычисляется в любое значение, присвоенное данной переменной.

Самый распространенный способ построения сложного выражения из более простых выражений предусматривает применение ***операции***. Операция каким-то образом объединяет свои операнды (обычно два) и вычисляет новое значение. Простой пример — операция умножения *. Выражение x * y вычисляется в виде произведения значений выражений x и y. Для простоты иногда мы говорим, что операция *возвращает*, а не “вычисляет” значение.

___

Первичные выражения

Простейшими выражениями, известными как ***первичные***, считаются автономные выражения — они не включают какие-то более простые выражения.

Первичными выражениями в JavaScript **являются константы или *литеральные* значения**, определенные ключевые слова языка и ссылки на переменные.

Литералы — это постоянные значения, встроенные напрямую в программу. Вот на что они похожи:

1.23 // Числовой литерал
"hello" // Строковый литерал
/шаблон/ // Литерал в виде регулярного выражения

Некоторые зарезервированные слова JavaScript являются

первичными выражениями:

true // Вычисляется как булевское истинное значение
false // Вычисляется как булевское ложное значение
null // Вычисляется как нулевое значение
this //Вычисляется как "текущий" объект

В отличие от остальных ключевое слово this — не константа; в разных местах программы оно вычисляется в виде разных значений.

третий тип первичных выражений представляет собой ссылку на переменную, константу или свойство глобального объекта:

i // Вычисляется как значение переменной i
sum // Вычисляется как значение переменной sum
undefined // Вычисляется как "неопределенное" свойство глобального объекта

Если переменная с таким именем отсутствует, тогда попытка оценки несуществующей переменной приводит к гене рации ошибки ссылки ReferenceError.

——

Инициализаторы объектов и массивов — это выражения, значениями которых будут вновь созданные объекты или массивы.

В литерал массива можно включать неопределенные элементы, просто опуская значение между запятыми. Скажем, следующий массив содержит пять элементов, в том числе три неопределенных:

let sparseArray = [1,,,,5];

Выражение определения функции определяет функцию JavaScript, а его значением будет вновь определенная функция. В некотором смысле выражение определения функции является “литералом функции” — в том же духе, как инициализатор объекта считается “объектным литералом”.

Выражение доступа к свойству вычисляется как значение свойства объекта или элемент массива.

выражение . идентификатор
***выражение [выражение*]**

___

Условный доступ к свойствам

В ES2020 появились два новых вида выражений доступа к свойствам:

выражение* ?. *идентификатор выражение* ?. [*выражение]

В JavaScript не располагают свойствами только два значения, null и undefined.

В обыкновенном выражении доступа к свойствам, использующем . или [], вы получите ошибку TypeEггог, если результатом вычисления выражения слева оказывается null или undefined.

Чтобы защититься от ошибок такого типа, вы можете применять синтаксис ?. и ?. [].

let a = { b: null }; a.b?.c.d // => undefined

Разумеется, если a.b — объект, который не имеет свойства по имени c, то a.b?.c.d снова сгенерирует ошибку TypeEггог, и мы захотим применить еще один условный доступ к свойству:

let a = { b: {} };
a.b?.c?.d // => undefined

___

Выражение вызова — это синтаксис JavaScript для вызова (либо выполнения) функции или метода.

f (0) // f - выражение функции; 0 - выражение аргумента

Math.max(x,y,z) // Math.шах - функция; x, y и z - аргументы
a.sort() // a.sort - функция; аргументы отсутствуют

Если в функции применяется оператор return ДЛЯ возврата значения, то оно становится значением выражения вызова, иначе значением выражения вызова будет **undefined**

Если оно представляет собой выражения доступа к свойству, тогда мы получаем *вызов метода*. В вызовах методов объект или массив, к свойству которого происходит доступ, становится значением ключевого слова this , пока выполняется тело функции. В итоге обеспечивается парадигма объектно-ориентированного программирования, когда функции (называемые “методами” при таком способе использования) работают на объекте, которому они принадлежат.

___

Условный вызов

В ES2020 функцию можно вызывать также с применением ?. () вместо (). Обычно при вызове функции в случае, если выражение слева от круглых скобок оказывается null или undefined либо чем-то, отличающимся от функции, то генерируется ошибка TypeEггог. В контексте нового синтаксиса вызовов ?. (), если выражение слева от ?. вычисляется как null и undefined, тогда результатом вычисления целого выражения вызова будет undefined без генерирования каких-либо исключений.

```
function square (x, log) {
   if (log) { log(x); }
   return x * x;
}

function square(x, log) { // Второй аргумент - необязательная функция
      log?.(x); // Вызвать функцию, если она есть
      return x * x; // Возвратить квадрат аргумента
}

o.f() o?.f() o.f?. ()
// Обыкновенный доступ к свойству, обыкновенный вызов //
```

Условный доступ к свойству, обыкновенный вызов
// Обыкновенный доступ к свойству, условный вызов

o.m () // Обыкновенный доступ к свойству, обыкновенный вызов
o?.m() // Условный доступ к свойству, обыкновенный вызов
o.m?. () // Обыкновенный доступ к свойству, условный вызов

Условный вызов с помощью ?. () — одна из новейших возможностей JavaScript. По состоянию на начало 2020 года такой новый синтаксис поддержи вался в текущих или бета-версиях большинства ведущих браузеров.

——

Выражения создания объектов

Выражение создания объекта создает новый объект и вызывает функцию (называемую конструктором) для инициализации свойств этого объекта.

new Object()
new Point (2, 3)

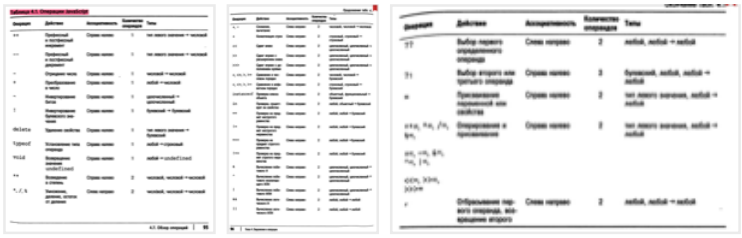
Если функции конструктора в выражении создания объекта аргументы не передаются, тогда пустую пару круглых скобок можно опустить:

new Object
new Date

——

Обзор операций

Таблица 4.1. Операции JavaScript



Однако некоторые операции представлены ключевыми словами наподобие delete и instanceof. Операции, представленные ключевыми словами, являются обыкновенными операциями, как те, что выражены с помощью символов пунктуации; у них просто менее лаконичный синтаксис.

___

Количество операндов

Операции можно разбить на категории, основываясь на количестве ожидаемых ими операндов (их **арности**).

Большинство операций JavaScript, подобных операции умножения *, являются бинарными операциями, которые объединяют два выражения в одно более сложное выражение. То есть они ожидают два операнда.

В JavaScript также поддерживается несколько унарных операций, которые превращают одиночное выражение в одиночное более сложное выражение.

Операция - в выражении -x представляет собой унарную операцию, выполняющую отрицание операнда x.

Наконец, в JavaScript имеется одна тернарная операция - условная операция **?:** , которая объединяет три выражения в одно.

___

Приоритеты операций

typeof my.functions [x] (y)

Хотя typeof - одна из операций с самым высоким приоритетом,

она будет выполняться на результате доступа к свойству, индексации массива и вызове функции, которые обладают более высоким приоритетом, чем typeof.

—

Арифметические выражения

Базовыми арифметическими операциями являются ** (возведение в степень), * (умножение), / (деление), %(модуль: остаток от деления), + (сложение) и - (вычитание).

Как упоминалось, мы обсудим операцию + в отдельном разделе. Остальные пять базовых операций просто оценивают свои операнды, при необходимости преобразуют значения в числа и затем вычисляют степень, произведение, частное, остаток или разность.

Нечисловые операнды, которые не удается преобразовать в числа, получают значение **NaN.**

—

Операция +

1 + 2
"1" + "2" "1" + 2 1+U
true + true
2 + null
2 + undefined

1 + 2 // => 3: сложение
'1' + '2' // => "12": конкатенация =>"12" : конкатенация после преобразования числа в строку
1+ {} // "1 [object Object]": конкатенация после преобразования объекта в строку
true + true // => 2: сложение после преобразования булевского значения в число
2 + null // => 2: сложение после преобразования null в 0
2 + undefined // => NaN: сложение после преобразования undefined в NaN

—

Побитовые операции

Побитовые операции выполняют низкоуровневые манипуляции битами в двоичном представлении чисел.

Побитовое ***И*** (&)
Операция & выполняет булевскую операцию И с каждым битом своих целочисленных аргументов. Бит устанавливается в результате, толь ко если соответствующий бит установлен в обоих операндах. Например, 0x1234 &OxOOFF вычисляется как 0x0034.

Побитовое ИЛИ (|)
Операция | выполняет булевскую операцию ИЛИ с каждым битом своих целочисленных аргументов. Бит устанавливается в результате, если соответствующий бит установлен в одном или обоих операндах. Например, 0x1234 | OxOOFF вычисляется как 0xl2FF.

Побитовое исключающее ИЛИ (A)
Операция Авыполняет булевскую операцию исключающего ИЛИ с каж дым битом своих целочисленных аргументов. Исключающее ИЛИ означает, что либо первый операнд равен true, либо второй операнд равен true, но не оба. Бит устанавливается в результате этой операции, если соответствующий бит установлен в одном из двух операндов (но не в обоих). Например, OxFFOO л OxFOFO вычисляется как OxOFFO.

Побитовое НЕ (~)
Операция - является унарной операцией, указываемой перед своим единственным целочисленным операндом. Она работает путем переключения всех битов в операнде. Из-за способа представления целых чисел со знаком в JavaScript применение операции ~ к значению эквивалентно изменению его знака и вычитанию 1. Например, -OxOF вычисляется как OxFFFFFFFO, или -16.

Сдвиг влево («)
Операция « перемещает все биты в первом операнде влево на

количество позиций, указанное во втором операнде, которое должно быть целым числом между 0 и 31. Скажем, в действии **a** « 1 первый бит **a** становится вторым, второй бит **a** — третьим и т.д. Для нового первого бита используется ноль, а значение 32-го бита утрачивается. Сдвиг значения влево на одну позицию эквивалентен умножению на 2, сдвиг на две позиции — умножению на 4 и т.д. Например, 7 « 2 вычисляется как 28.

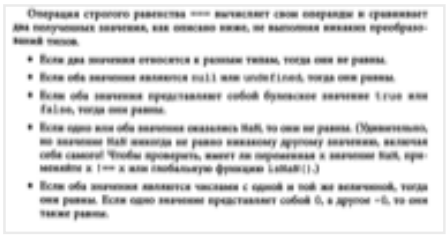
Сдвиг вправо с расширением знака **(»)**
Операция » перемещает все биты в первом операнде вправо на количест во позиций, указанное во втором операнде (целое число между 0 и 31). Биты, сдвинутые за правую границу числа, утрачиваются. Биты, заполня емые слева, зависят от знакового бита исходного операнда, чтобы предох ранять знак результата. Если первый операнд положительный, то старшие биты результата заполняются нулями; если первый операнд отрицатель ный, тогда в старшие биты результата помещаются единицы. Сдвиг по ложительного значения вправо на одну позицию эквивалентен делению на 2 (с отбрасыванием остатка), сдвиг вправо на две позиции равноценен целочисленному делению на 4 и т.д. Например, 7 » 1 вычисляется как 3, но имейте в виду, что -7 » 1 вычисляется как -4.

Сдвиг вправо с дополнением нулями (» >)
Операция » > похожа на операцию » за исключением того, что старшие биты при сдвиге будут всегда нулевыми независимо от знака первого опе ранда. Операция полезна, когда 32-битные значения желательно тракто вать как целые числа без знака. Например, -1 » 4 вычисляется как -1, но -1 > » 4 — как OxOFFFFFFF. Это единственная побитовая операция JavaScript, которую нельзя применять со значениями BigInt. Тип BigInt не представляет отрицательные числа, устанавливая старший бит так, как поступают 32-битные целые числа, а потому данная операция является просто дополнением предшествующих двух.

——

Операции равенства и неравенства

Строгое равенство и Равенство с преобразованием типов




Операции сравнения

Меньше (<)
Операция < вычисляется как true, если первый операнд меньше второго операнда; иначе она вычисляется как false.

Больше (>)
Операция > вычисляется как true, если первый операнд больше второго операнда; иначе она вычисляется как false.

Меньше или равно (<=)
Операция <= вычисляется как true, если первый операнд меньше второго операнда или равен ему; иначе она вычисляется как false.

Больше или равно (>«)
Операция >= вычисляется как true, если первый операнд больше второго операнда или равен ему; иначе она вычисляется как false.

—

Операция in

Операция in ожидает с левой стороны операнд, который является строкой, символом или значением, допускающим преобразование в строку.
С правой стороны она ожидает операнд, который должен быть объектом.

Операция in вычисляется как true, если значением с левой стороны будет имя свойства объекта с правой стороны.

—

Операция **instanceof**

let d = new Date () ; / / Создание нового объекта с помощью конструктора Date ()
d instanceof Date => true: объект d был создан с помощью Date ( )
d instanceof Object / / => true: все объекты являются экземплярами Object
d instanceof Number / / => false: d - не объект Number
let a = [1, 2, 31; // Создание нового массива посредством синтаксиса литерала типа массива
a instanceof Array => true: объект a - массив
a instanceof Object // => true: все массивы являются объектами
a instanceof RegExp // => false: массивы - не регулярные выражения

Обратите внимание, что все объекты являются экземплярами Object. Принимая решение о том, будет ли объект экземпляром класса, опера-
чия instanceof учитывает "суперклассы". Если операнд с левой стороны instanceof - не объект, тогда instanceof возвращает false. Если операнд с
правой стороны - не класс объектов, то генерируется ошибка TypeError.

Для понимания работы операции instanceof вы должны знать о "цепочке прототипов». Она представляет собой механизм наследования JavaScript и описана в подразделе 6.3.2. Чтобы вычислить выражение o instanceof f , интерпретатор JavaScript вычисляет f. prototype и затем ищет это значение в цепочке прототипов O. Если оно находится, тогда o -- экземпляр f (или какого-топодкласса f) и операшия возвращает true.

Если f.prototype не входит в состав значений цепочки прототипов O, то O - не экземпляр f и instanceof возвращает false.

—

Логические выражения

Погические операции &&, !! и ! выполняют действия булевой алгебры и часто применяются в сочетании с операциями отношений для объединения двух выражений отношений в одно более сложное выражение.

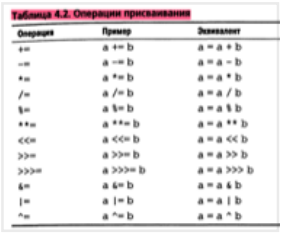
Логическое НЕ(!)

Таким образом, операция ! всегда возвращает true или false и вы можете преобразовать любое значение x в эквивалентное булевское значение, применив эту операцию дважды: ! !x

// Законы де Моргана
!(p && q) === (!p || !q) // => true : для всех значений p и q
!(p II q) === (!p && !q) //=>true : для всех значений p и q

Присваивание с действием

Таблица 4.2. Операции присваивания



Операция	Пример	Эквивалент
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
**=	a **= b	a = a ** b
<<=	a <<= b	a = a << b
>>=	a >>= b	a = a >> b
>>>=	a >>>= b	a = a >>> b
&=	a &= b	a = a & b
|=	a |= b	a = a | b
^=	a ^= b	a = a ^ b

Вычисление выражений

С помощью такого сложного языка, как JavaScript, не существует способа сделать пользовательский ввод безопасным для применения с функцией eval ().

HTTP-заголовок Content-Security-Policy, чтобы запретить применение eval () на всем веб-сайте.

eval() является функцией, но рассматривается в этой главе, посвящен ной операциям, т.к. в действительности она должна была быть операцией, функция eval () была определена в самых ранних версиях языка и с тех пор проектировщики языка и разработчики интерпретатора накладывали на нее ограничения, делая ее все больше и больше похожей на операцию.

Проблема с определением eval () как функции в том, что ей можно давать другие имена:

let f = eval;
let g = f;

Если это разрешено, то интерпретатор не может точно знать, какие функции вызывают eval(), и энергичная оптимизация невозможна. Проблемы мож но было бы избежать, если бы eval () являлась операцией (и зарезервированным словом).

В подразделах 4.12.2 и 4.12.3 мы рассмотрим ограничения, налагаемые на функцию eval(), которые делают ее похожей на операцию.

“Прямой вызов” — это вызов функции eval() с помощью выражения, при меняющего точное, безусловное имя “eval” (которое начинает воспринимать ся как зарезервированное слово). Прямые вызовы eval() используют таб лицу переменных вызывающего контекста. Любой другой вызов — непрямой вызов — применяет в качестве таблицы переменных глобальный объект и не может читать, устанавливать или определять локальные переменные или функции. (И прямой, и непрямой вызовы могут определять новые переменные толь ко посредством var. Использование let и const внутри вычисляемой строки приводит к созданию переменных и констант, которые являются локальными по отношению к вызову eval() и не изменяют вызывающую или глобальную среду.)

—

Смешанные операции

Условная операция (?:)

Условная операция — единственная операция с тремя операндами в JavaScript, которая иногда называется *тернарной операцией*.

x > 0 ? x : -x // Абсолютное значение x

Операция выбора первого определенного операнда (??)

Подобно && и | | операция ?? работает по принципу короткого замыкания: она вычисляет второй операнд, только если первый операнд оказался null или undefined.

a ?? b эквивалентно:

(a !== null && a !== undefined) ? a : b

Операция ?? является удобной альтернативой операции | | (см. подраздел 4 10.2), когда вы хотите выбрать первый *определенный,* а не первый истинный операнд. Хотя | | — формально операция логического ИЛИ,

// Если значение maxWidth истинное, то использовать его.
// Иначе искать значение в объекте preferences. Если оно не истинное,
// тогда использовать жестко закодированную константу,

let шах = maxWidth || preferences.maxWidth || 500;

Проблема с таким идиоматическим применением состоит в том, что ноль, пустая строка и false — ложные значения, которые в определенных обстоятельствах могут быть совершенно допустимыми.

Но если мы изменим операцию | | на ??, тогда получим выражение, где ноль оказывается допустимым значением:

let max = maxWidth ?? preferences.maxWidth ?? 500;

let options = { timeout: 0, title: verbose: false, n: null };
options.timeout ?? 1000
options.title ?? "Untitled"
options.verbose ?? true
options.quiet ?? false
options.n ?? 10

// => 0: как определено в объекте
// => "" как определено в объекте
// => false: как определено в объекте

// => false: свойство не определено
// => 10: свойство равно null

Операция ?? похожа на операции && и | |, но ее приоритет не выше и не ниже, чем у них.

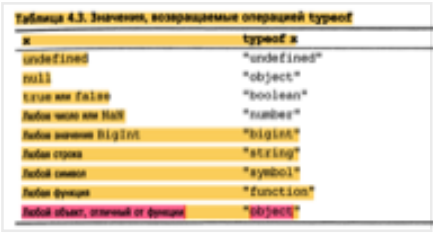
(a ?? b) || c //Сначала ??, затем ||
a ?? (b | | c) // Сначала ||, затем ??
a ?? b || c // SyntaxError: круглые скобки обязательны

—

Операция typeof

typeof — унарная операция, помещаемая перед своим единственным операндом, который может быть любого типа.

Таблица 4.3. Значения, возвращаемые операцией typeof



x	typeof x
undefined	"undefined"
null	"object"
true или false	"boolean"
Любое число или NaN	"number"
Любое значение bigint	"bigint"
Любая строка	"string"
Любой символ	"symbol"
Любая функция	"function"
Любой объект, отличный от функции	"object"

Обратите внимание, что typeof возвращает "object", когда операнд имеет значение null.

Чтобы различать один класс объектов от другого, вам придется использовать другие методики, такие как операция instanceof (см. подраздел 4.9.4), атрибут c l a s s (см. подраздел 14.4.3) или свойство constructor (см. подраздел 9.2.2 и раздел 14.3).

—

Операция delete

delete — унарная операция, которая пытается удалить свойство объекта или элемент массива, указанный в качестве ее операнда.

let a = [1,2,3];
delete a [2];

2 in a // => false: элемент 2 массива больше не существует a.length

let o = {x: 1, y: 2};

// Удалять переменную нельзя; возвращается false
// или генерируется SyntaxError в строгом режиме

delete o;

// Неудаляемое свойство: возвращается false
// или генерируется TypeEггог в строгом режиме

delete Object.prototype;
——

Операция await

Операция await появилась в ES2017 как способ сделать асинхронное программирование в JavaScript более естественным.

——

Операция void

void — унарная операция, располагающаяся перед своим операндом, который может быть любого типа.
Операция void необычна и используется редко; она вычисляет свой операнд, после чего отбрасывает значение и возвращает undefined.
Поскольку значение операнда отбрасывается, применение void имеет смысл, только когда у операнда есть побочные эффекты.

——

Операция "запятая"

Операция “запятая” (,) является бинарной операцией, чьи операнды могут быть любого типа. Она вычисляет свой левый операнд, затем правый операнд и далее возвращает значение правого операнда.

```
for (let i=0,j=10; i < j; i++,j— ) {
      console.log(i+j);
}
```

___