JSDG 🦏 | Statements | Операторы | chapter 5

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

операторы являются предложениями или командами JavaScript

___

Выражения *вычисляются* для выдачи значения, но операторы *выполняются,* чтобы что-то произошло.

Выражения с побочными эффектами, такими как присваивание и вызов функции, могут выступать в качестве операторов, и в случае использования подобным образом они называются ***операторами-выражениями***.

Похожая категория операторов называется ***операторами объявлений***, которые объявляют новые объявления и определяют новые функции.

Условные операторы if и switch ...
Операторы циклов while и for ...
Операторы переходов break, return и throw,

___

Операторный блок — это просто последовательность операторов, помещенная внутрь фигурных скобок.

Элементарные операторы в блоке завершаются точками с запятой, но сам блок — нет.

Противоположностью будет *пустой оператор*: он дает возможность не включать операторы там, где ожидается один оператор - **;**

При выполнении пустого оператора интерпретатор JavaScript не предпринимает никаких действий. Пустой оператор иногда полезен, когда желательно создать цикл с пустым телом.

for(let i = 0; i < a.length; a[i++] = 0) ;

Вся работа в данном цикле делается выражением a [i++] = 0, а потому отсутствует необходимость в теле цикла.

___

Циклы

while
do/while
for
for/of (и его разновидность for/await) и
for/in

___

while

while (*выражение) оператор*

```
let count = 0;
while(count < 10) {
	console.log(count);
	count++;
}
```

___

do/while

Цикл do/while похож на цикл while, но только выражение цикла проверяется в конце цикла, а не в начале.

Таким образом, тело цикла всегда выполняется, по крайней мере, один раз.

Вот его синтаксис:
do
оператор
while *(выражение*);

Цикл do/while применяется не так часто, как while —на практике

довольно редко имеется уверенность в том, что цикл желательно выполнить хотя бы раз.

___

for

for (инициализация : проверка : инкрементирование)
    оператор

===

Объяснить, как работает цикл for, проще всего, показав эквивалентный цикл while:

инициализация:
while (проверка) {
    оператор
    инкрементирование;
}

___

Обратите внимание, что в показанном цикле отсутствует выражение инициализация.

В цикле for можно опускать любое из трех выражений, но две точки с запятой обязательны.

Если вы опустите выражение проверка, тогда цикл будет повторяться нескончаемо долгий период времени,

и for (; ;) - еще один способ записи бесконечного цикла, подобный while (true).

___

for / of

В ES6 определен новый оператор цикла: for / of.

В новом цикле применяется ключевое слово for, но он

представляет собой совершенно другой вид цикла в сравнении с обыкновенным циклом for.

Цикл for / of работает с *итерируемыми* объектами.

Итерация по массивам производится “вживую” — изменения, внесенные во время итерации, могут влиять на ее исход.

Если мы модифицируем код, добавив внутрь тела цикла строку data.push(sum);, то создадим бесконечный цикл, т.к. итерация никогда не доберется до последнего элемента массива.

```
let o = { x: 1, y: 2, z: 3 };
for (let element of o) {
      // Генерируется TypeEггог, потому что o - не итерируемый
объект
      console.log(element);
)

let keys = "";
for (let k of Object.keys(o)) keys += k; keys // => "xyz"
let sum = 0; for (let v of Object.values (o)) sum += v; // sum => 6
let pairs = "";

for(let [k, v] of Object.entries(o)) pairs += k + v; pairs // =>
"xly2z3"
```

Метод **Object.entries** () возвращает массив массивов, где каждый внутренний массив представляет пару “ключ/значение” для одного свойства объекта. Мы используем в примере кода деструктурирующее присваивание для распаковки таких внутренних массивов в две индивидуальные переменные.

___

for / of с классами Set и Map

Встроенные классы Set и Map в ES6 являются итерируемыми.

При итерации по Set с помощью for/of тело цикла выполняется однократно для каждого элемента множества.

___

Асинхронная итерация с помощью for/await

В ES2018 вводится новый вид итератора, называемый асинхронным итератором, и разновидность цикла for/ of, известная как цикл for/await, который работает с асинхронными итераторами.

Чтобы понять цикл for/await, вам необходимо ознакомиться с материалом глав 12 и 13, но ниже показано, как он выглядит:

// Читать порции из асинхронно итерируемого потока данных и выводить их
```
async function printStream (stream) {
    for await (let chunk of stream) console.log (chunk);
}
```

___

for / in

Оператор for/in организует цикл по именам свойств указанного объекта.

Синтаксис выглядит следующим образом: for (переменная in объект) оператор

```
for (переменная in объект)
    оператор
```

Цикл for / i n в действительности не перечисляет все свойства объекта.

Он не перечисляет свойства, именами которых являются значения Symbol.

К тому же среди свойств со строковыми именами цикл for / in проходит только по *перечислимым* свойствам

Перечислимые унаследованные свойства (см. подраздел 6.3.2)

также перечисляются циклом for/in.

Это означает, что если вы применяете циклы for / in и также используете код, где определяются объекты, которые наследуются всеми объектами, тогда ваши циклы могут вести себя не так, как ожидалось.

По указанной причине многие программисты вместо цикла for/in предпочитают применять цикл for/of с Object.keys ().

Если в теле цикла for/ in удаляется свойство, которое еще не перечислялось, то оно перечисляться не будет.

Если в теле цикла определяются новые свойства объекта, то они могут как перечисляться, так и не перечисляться. Дополнительные сведения о порядке перечисления свойств циклом for/ in ищите в подразделе 6.6.1.

===

keys(obj) returns only an array with the own properties of the object, while the for...in returns also the keys found in the prototype chain,

link

___

Переходы

операторы переходов.
break
continue

JavaScript разрешает операторам быть именованными, или *помеченными*, a break и continue способны идентифицировать целевую метку цикла или другого оператора.

throw *инициирует* исключение и предназначен для работы с оператором t r y / c a t c h / f i n a l l y

___

Помеченные операторы

идентификатор*: *оператор

mainloop: while (token !== null) {
// Код опущен. ..
continue mainloop; // Перейти к следующей итерации именованного цикла / Код опущен.

___

оператор **break**

- for (let i = 0; i < a.length; i++) { if (a[i] === target) break; }
- JavaScript также разрешает дополнять ключевое слово break меткой операто ра (просто идентификатором без двоеточия):
  - **break** *имя__метки*;
  - Когда оператор break используется с меткой, он вызывает переход в конец, или прекращение, включающего оператора, который имеет указанную метку. Если эта форма break применяется в отсутствие включающего оператора с указанной меткой, то возникает синтаксическая ошибка. При такой форме оператора break именованный оператор не обязательно должен быть циклом или switch: break может "выйти" из любого включающего оператора. Он может быть даже операторным блоком, сгруппированным внутри фигурных скобок с единственной целью - снабжение блока меткой.

```
// Начать с помеченного оператора, из которого можно выйти
// в случае возникновения ошибки
computeSum: if (matrix) {
    for (let x = 0; × < matrix. length; x++) I
        let row = matrix[x];
        if (!row) break computeSum;

        for (let y = 0; y < row. length; y++) {
            let cell = rowly];
```

```
            if (isNaN (cell)) break computeSum;
            sum += cell;
        }
        success = true;

        // Операторы break переходят сюда. Если мы
оказываемся здесь
        // с success == false, тогда с матрицей что-то пошло не
так.
        // В противном случае sum содержит сумму всех ячеек
матрицы.
```

___

Оператор continue

похож на break. Тем не менее, вместо выхода из цикла continue перезапускает цикл со следующей итерации. Синтаксис оператора continue в той же мере прост, как синтаксис break:

```
continue;
```

Оператор continue также может применяться с меткой:
continue имя метки;

В помеченной и непомеченной формах оператор continue может использоваться только внутри тела цикла.

___

return

return *выражение;*

```
function displayObject(o) {
    // Немедленный возврат, если аргумент равен null или
undefined
    if (o) return;
    // остальной код функции..
}
```

yield

Оператор yield во многом похож на return, но используется только в генераторных функциях ES6.

```
// Генераторная функция, которая выдает диапазон целых чисел
function* range (from, to) {
    for (let i = from; i <= to; i++)        yield i;
}
```

throw

Исключение представляет собой сигнал, который указывает, что возникло какое-то необычное условие или ошибка.

Генерация исключения означает предупреждение о таком ошибочном или необычном условии.

Перехват исключения означает его обработку - выполнение любых действий, необходимых или
подходящих для восстановления после исключения.

Исключения перехватываются с помощью оператора try / catch / finally
Оператор throw имеет следующий синтаксис:
throw выражение;

```
if (x < 0) throw new Error ("Значение x не должно быть отрицательным”);
```

try/catch/finally:

```
try {
    // В нормальной ситуации этот код выполняется от начала до конца блока
    // безо всяких проблем. Но иногда он может генерировать исключение,
    //  либо напрямую с помощью оператора throw, либо косвенно за счет вызова
    //  метода, который генерирует исключение.
}
catch (e) {
```

```
    // Операторы в данном блоке выполняются, если и только если в блоке try
    //  было сгенерировано исключение. Эти операторы могут использовать
    // локальную переменную e для ссылки на обьект Error или другое значение,
    // которое было указано в throw. В блоке можно каким-то образом
    // обработать исключение, проигнорировать его, ничего не делая,
    // или повторно сгенерировать исключение с помощью throw.
}
finally {
    // Данный блок содержит операторы, которые всегда выполняются
    // независимо от того, что произошло в блоке try.
    // Они выполняются при завершении блока try:
    // 1) нормальным образом после того, как достигнут конец блока;
    // 2) из-за оператора break, continue или return;
    / 3) из-за исключения, которое было обработано конструкцией catch выше;
    // 4) из-за необработанного исключения, которое продолжилосвое распространение.
}
```

Смешанные операторы - with, debugger и "use strict"

with (объект)
оператор

Этот оператор создает временную область видимости со свойствами *объекта* в качестве переменных и затем выполняет *оператор* внутри такой области видимости.
Код JavaScript, в котором используется with, трудно поддается оптимизации и, вероятно, будет выполняться значительно медленнее, чем эквивалентный код, написанный без оператора with.

debugger

Оператор debugger обычно ничего не делает. Тем не менее, если программа отладчика доступна и работает, то реализация может (но не обязана) предпринимать определенные действия отладки.

```
function flo) 1
      if (o === undefined) debugger; // Временная строка для
отладочных целее,  далее идет остальной код функции
}
```

"use strict"

"use strict" — это директива, появившаяся в ES5.

—

Объявления

Ключевые слова **const, let, var, function, class, import и export формально не являются операторами, но они во многом похожи на операторы, и в книге они неформально относятся к операторам**

Такие ключевые слова более точно называть объявлениями, а не операторами,

В ES6 и последующих версиях объявление class создает новый класс и назначает ему имя, посредством которого на класс можно ссылаться.

```
class Circle {
      constructor (radius) { this.r = radius; ]
      area () { return Math.PI * this.r * this.r; )
      circumference () { return 2 * Math.PI * this.r;
}
```

import и export

import Circle from './geometry/circle.js';
import { PI, TAU | from ' /geometry/constants.is';
import { magnitude as hypotenuse } from '. /vectors/utils.js';

Директива export имеет больше вариантов, чем директива import. Во один из них:

// geometry/constants.js
const PI = Math.PI;
const TAU = 2 * PI;
export (PI, TAU };

И когда модуль экспортирует только одно значение, то это делается с помощью специальной формы export default:

export const TAU = 2 * Math. PI;
export function magnitude (×, y) I return Math.sqrt (x*x + y*y) ; J
export default class Circle { /* определение класса опущено * /)

___

Резюме по операторам JavaScript

Таблица 5.1.

___