JSDG 🦏 | Arrays | Массивы | chapter 7

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

В этой главе обсуждаются массивы - фундаментальный тип данных в JavaScript и большинстве других языков программирования.

Массив представляет собой упорядоченную коллекцию значений.

Каждое значение называется элементом, и каждый элемент имеет числовую позицию в массиве, известную как индекс.

Массивы JavaScript являются нетипизированными: элемент массива может относиться к любому типу, а разные элементы одного массива могут иметь отличающиеся типы.

Элементы массива могут быть даже объектами или другими массивами, что позволяет создавать сложные структуры данных, такие как массивы объектов и массивы массивов.

Индексы в массивах JavaScript начинаются с нуля и представляют собой 32-битные числа: индекс первого элемента равен 0, а наибольший возможный индекс составляет 4294967294 (232- 2) для максимального размера массива в 4294967 295 элементов.

Массивы JavaScript являются динамическими: по мере надобности они увеличиваются или уменьшаются, а при создании массива нет необходимости объявлять для него фиксированный размер или повторно размещать в памяти в случае изменения его размера.

Массивы JavaScript могут быть разреженными: элементы не обязаны иметь смежные индексы, поэтому возможно наличие брешей.

Каждый массив JavaScript имеет свойство length. Для

неразреженных массивов свойство length указывает количество элементов в массиве. Для разреженных массивов значение length больше самого высокого индекса любого элемента.

Массивы JavaScript являются специализированной формой объекта JavaScript, и в действительности индексы массивов - не многим более чем имена свойств, которые просто оказались целыми числами.

Реализации обычно оптимизируют массивы, так что доступ к элементам массива с числовой индексаций, как правило, выполняется гораздо быстрее, нежели доступ к обыкновенным свойствам объектов.

Массивы наследуют свойства от Array.prototype, который определяет богатый набор методов манипулирования массивами, раскрываемый в разделе 7.8.

Большинство этих методов являются обобщенными, а значит, они корректно работают не только с подлинными массивами, но также с любым "объектом, похожим на массив".

___

Создание массивов

• литералов типа массивов; // []

• операции распространения (...) на итерируемом объекте;

```
let a = [1, 2, 3];
let b = [0, ...a, 4]; // b == [0, 1, 2,3, 4]
```

• конструктора Array();

• Вызов без аргументов: let a= new Array (); Такая методика создает пустой массив без элементов и эквивалентен литералу типа массива [].

• Вызов с одиночным числовым аргументом, который указывает длину: let a= new Array (10); Такая методика создает массив с указанной длиной.

• Явное указание двух и более элементов массива или одиночного нечислового элемента:

let a = new Array (5, 4, 3, 2, 1, "testing, testing");

В такой форме аргументы конструктора становятся элементами нового массива. Применять литерал типа массива почти всегда проще, чем конструктор Array() в подобной манере.

• фабричных методов **Array.of() и Array.from().**

• Когда функция конструктора Array() вызывается с одним числовым аргументом, она использует его как длину массива. Но при вызове с большим количеством числовых аргументов функция конструктора Array () трактует их как элементы для создаваемого массива. Это означает, что конструктор Array () не может применяться для создания массива с единственным числовым элементом.

• В версии ES6 функция **Array.of()** решает описанную проблему: она представляет собой фабричный метод, который создает и возвращает новый массив, используя значения своих аргументов (независимо от их количества) в качестве элементов массива:

Array.of() // => []; при вызове без аргументов возвращает пустой массив

Array.of(10) // => [10]; при вызове с единственным числовым аргументом способна создавать массивы

Array.of(1,2,3) // => [1, 2, 3]

Array.from — еще один фабричный метод, введенный в ES6.

Он ожидает в первом аргументе итерируемый или похожий на массив объект и возвращает новый массив, который содержит элементы переданного объекта.

С итерируемым аргументом Array.from(iterable) работает подобно операции распространения [...iterable].

Кроме того, она предлагает простой способ для получения копии массива:

let copy = Array.from(original);

Метод Array.from() также важен оттого, что он определяет способ создания копии в виде подлинного массива из объекта, похожего на массив.

обрабатывать такие значения может быть легче, если сначала преобразовать их в подлинные массивы:

let truearray = Array.from(arraylike);

Метод Array.from() принимает необязательный второй аргумент. Если вы передадите во втором аргументе функцию, тогда в ходе построения нового массива каждый элемент из исходного объекта будет передаваться указанной вами функции, а возвращаемое ею значение будет сохраняться в массиве вместо первоначального значения.

(Метод Array.from() во многом похож на метод тар () массива, который будет представлен позже в главе, но выполнять сопоставление эффективнее во время построения массива, чем создать массив и затем отобразить его на другой новый массив.)

___

Операция распространения (...) работает с любым итерируемым объектом.

Итерируемые обьекты - такие объекты, по которым выполняет проход цикл for/of;

Строки итерируемы, поэтому вы можете применять операцию распространения для превращения любой строки в массив односимвольных строк:

let digits = [... "0123456789ABCDEF"'];
digits / => ["O", "1", "2", "3", "4", "5", "6", "7", "g", "9", "A", "B", "C", "D", "E" ", "F"]

Объекты множеств Set

Итерируемы, так что легкий способ удаления дублированных элементов из массива предусматривает его преобразование в объект множества и затем немедленное преобразование множества в массив с использованием операции распространения:

let letters = [..."hello world"]; [...new Set (letters)] / => ["h", "e","l", "o"," ", "w", "r", "d"]

___

Чтение и запись элементов массива

Вспомните, что массивы - специализированный вид объекта. Квадратные скобки, используемые для доступа к элементам массива, работают в точности как квадратные скобки, применяемые для доступа к свойствам массива.

====

Интерпретатор JavaScript преобразует указанный вами числовой индекс массива в строку - индекс 1 становится строкой "1", - после чего использует результирующую строку в качестве имени свойства.
В преобразовании индекса из числа в строку нет ничего особенного: вы можете поступать так и с обычными объектами:

let o = 11; // Создать простой объект
o[1] = "one"; // Индексировать его с помощью целого числа
o("1") // => "one"; числовые и строковые имена свойств считаются одинаковыми

Полезно четко отличать индекс массива от имени свойства объекта.

Все индексы являются именами свойств, то только имена свойств, которые представляют собой целые числа между 0 и 2 в 32 степени минус (-) 2, будут индексами.

Все массивы являются объектами, и вы можете создавать в них свойства с любыми именами.

Однако если вы применяете свойства, которые представляют собой индексы массива, то массивы поддерживают особое поведение по обновлению их свойства length по мере необходимости.

___

Разреженные массивы

Разреженный массив - это массив, элементы которого не имеют непрерывных индексов, начинающихся с 0.
Обычно свойство length массива указывает количество элементов в массиве.
Когда массив разреженный, значение свойства length будет больше количества элементов.

Разреженные массивы можно создавать с помощью конструктора Array() и просто за счет присваивания по индексу, превышающему текущее значение свойства length массива.

let al = [,]; // Этот массив не содержит элементов и значение length равно 1
let a2 = [undefined]; // Этот массив имеет один элемент undefined
0 in al // => false: a1 не содержит элемента по индексу 0
O in a2 / => true: a2 имеет значение undefined по индексу 0

Длина массива

Каждый массив имеет свойство length, и именно оно делает массивы отличающимися от обыкновенных объектов JavaScript. Для плотных (т.е. неразреженных) массивов свойство length указывает количество элементов. Его значение на единицу больше самого высокого индекса в массиве:

[]. length // => 0: массив не содержит элементов
["a", "b", "c"]. length // => 3: самый высокий индекс равен 2, значение length равно 3

Когда массив разреженный, значение свойства length больше количества элементов, и мы можем сказать лишь то, что значение length гарантированно превышает индекс любого

элемента в массиве.

___

Добавление и удаление элементов массива

let a = []; / Начать с пустого массива
a[0] = "zero" // И добавить в него элементы
a[1] = "one";

let a = []; // Начать с пустого массива
a.push ("zero"); // Добавить значение в конец. a = ["zero"]
a.push ("one", "two"); // Добавить еще два значения. a = ("zero", "one", "two"]

Метод pop () - противоположность push () : он удаляет последний элемент мас-
сива, уменьшая его длину на 1.

Аналогично метод shift () удаляет и возвращает первый элемент массива, уменьшая его длину на 1 и сдвигая все элементы по индексам, которые на единицу меньше их текущих индексов. Дополнительные сведения об указанных методах приведены в разделе 7.8.

___

Итерация по массивам

Начиная с версии ES6, самый легкий способ циклического прохода по массиву (или любому итерируемому объекту) связан с применением цикла for/of,

let letters = [... "Hello world"]; // Массив букв
let string = "";
for (let letter of letters) {
    string += letter;
}
string // => "Hello world"; мы повторно собрали первоначальный текст

Если вы хотите применять с массивом цикл for/of и знать индекс

каждого элемента массива, тогда используйте метод entries () массива вместе с деструктурирующим присваиванием, например:

```
let everyother = ""';
for (let (index, letter] of letters.entries () )
      if (index & 2 === 0) everyother += letter; // буквы по четным индексам
}
everyother // => "Hlowrd"
```

Еще один хороший способ итерации по массивам предполагает применение forEach (). Это не новая форма цикла for, а метод массива, который предлагает функциональный подход к итерации. Вы передаете методу forEach () массива функцию и forEach ( ) будет вызывать указанную функцию по одному разу для каждого элемента в массиве:

```
let uppercase = "";
letters.forEach ( letter => { / / Обратите внимание на синтаксис стрелочной функции
      uppercase += letter.toUpperCase () ;
})
uppercase // => "HELLO WORLD"

let vowels = "";
for (let i = 0; i < letters. length; it+) ( // Для каждого индекса в массиве
      let letter = letters[i]; // Получить элемент по этому индексу
      if (/(aeiou]/.test (letter)) { // Использовать проверку с регулярным выражением
            vowels += letter; // Если гласная буква, то запомнить ее
      }
}
vowels // => "eoo"

// Сохранить длину массива в локальной переменной
for (let i = 0, len = letters.length; i < len; i++) {
      // тело цикла остается прежним
}

// Итерация в обратном направлении с конца до начала массива
```

```
for (let i = letters.length-l; i >= 0; i--) {
      // тело цикла остается прежним
}

for (let i = 0; i < a.length; it+ {
      if (a[i] === undefined) continue; // Пропускать неопределенные
и несуществующие элементы тело цикла
}
```

___

Методы массивов

Методы итераторов для массивов
forEach()
map()
filter()
find() и findIndex()
every() и some()
reduce () И reduceRight ()

___

Выравнивание массивов с помощью

flat() И flatMap()

```
[1, [2, 3]].flat () // => [1, 2,3]
[1, [2, [3]]].flat () // => [1, 2, [3]]


let a = [1, (2, [3, [4]1]]:
a.flat (1) => [1, 2, [3, (4]]]
a.flat (2) => [1, 2, 3, [4]]
a.flat (3) => [1, 2, 3, 4]
a.flat (4) => [1, 2, 3, 4]
```

Метод flatMap () работает точно как метод map() за исключением того, что возвращаемый массив автоматически выравнивается, как если бы он передавался flat(). То есть вызов a.flatMap (f) - то же самое, что и a.map(f).flat (), но эффективнее:

```
let phrases = ["hello world", "the definitive guide"];
let words = phrases. flatMap (phrase => phrase.split (" "));
words // => ["hello", "world", "the", "definitive", "guide");
```

Вы можете считать flatMap () обобщением тар (), которое позволяет отображать каждый элемент входного массива на любое количество элементов выходного массива. В частности, flatMap () дает возможность отобразить входные элементы на пустой массив, который ничего не выравнивает в выходном массиве:

```
// Отобразить неотрицательные числа на их квадратные корни

[-2, -1, 1, 21. flatMap (x => x < 0 ? [] : Math.sqrt (%)) // => (1, 2**0.51
```

Присоединение массивов с помощью concat ()

```
let a = [1,2,3];
a.concat (4, 5) / => [1,2,3,4,5]
a.concat ([4,5], (6,7]) // => [1,2,3,4,5,6,7]; массивы выравниваются
a.concat (4, [5, (6,7]]) // => [1,2,3,4,5, [6,7]]; но не вложенные массивы
```

Организация стеков и очередей с помощью push () , pop (), shift () и unshift ()

```
let stack = [] : // stack == [] *
stack.push (1,2); // stack == [1,21;
stack.pop (); // stack == [1]; возвращается 2
stack.push (3); / stack == [1, 3]
stack.pop (); // stack == [1]: возвращается 3
stack.push ( [4,5]); // stack == [1, [4,5]]
stack.pop () // stack == [1]; возвращается [4,5]
stack.pop () ; // stack == []: возвращается 1

let q = []; // q == []
q.push (1, 2); // g == [1,2]
```

```
q.shift (); // q == (2]: возвращается 1
q.push (3)
q.shift ()
q.shift ()
```

Существует одна особенность метода unshift (), о которой стоит упомянуть, т.к. она может вызвать у вас удивление. В случае передачи unshift () множества аргументов они вставляются все сразу, а это значит, что они окажутся в массиве не в том порядке, в каком они были бы при вставке их по одному за раз:

```
let a = []; // a == []
a.unshift (1) // a == [1]
a.unshift (2) / a == (2, 1]

a = []; // a == []
a.unshift (1,2) // a == [1, 2]
```

___

Работа с подмассивами с помощью slice (), splice (), fill () и copyWithin ()

slice()

Метод slice () возвращает срез, или подмассив, заданного массива. В двух
его аргументах указываются начало и конец среза, подлежащего возвращению.

```
let a = [1,2,3,4,51;
a.slice (0,3); // Возвращается [1,2,3]
a.slice (3); // Возвращается [4,5]
a.slice (1,-1); // Возвращается [2,3,4]
a.slice (-3,-2); // Возвращается [3]
```

splice()

splice () - это универсальный метод для вставки или удаления элементов из массива. В отличие от slice () и concat () метод splice () модифицирует массив, на котором вызывается. Важно отметить, что splice () и slice () имеот очень похожие имена, но

реализуют существенно отличающиеся действия.

let a = [1,2,3,4,5, 6,7,8];
a.splice (4) // => [5,6,7,8]: а теперь [1,2,3,4]
a.splice(1,2) // => (2,3]; а теперь (1,4]
a.splice (1,1) // => [4]: а

Первые два аргумента splice () указывают, какие элементы массива подлежат удалению. За этими аргументами может следовать любое количество дополнительных аргументов, задающих элементы, которые должны быть вставлены в массив, начиная с позиции, указанной в первом аргументе. Вот пример:

let a = [1,2,3,4,5]:
a.splice (2,0, "a", "b") // => []; а теперь [1,2, "a", "b", 3,4,5]
a.splice (2,2, [1,2],3) // => ["a", "b"]; а теперь [1,2, [1,2],3,3,4,5]

Следует отметить, что в отличие от concat () метод splice () вставляет сами массивы, а не их элементы.

fill()

Метод **fill()** устанавливает элементы массива или среза массива в указанное значение.
Он видоизменяет массив, на котором вызывается, и также воз вращает модифицированный массив:

let a = new Array (5); // Начать с массива без элементов длиной 5
a.fill (0) // => [0,0,0,0, 0]: заполнить массив нулями
a.fill (9, 1) => [0,9,9,9, 9]; заполнить значениями 9, начиная с индекса 1
a.fill (8, 2, -1) // => (0,9,8,8, 9]: заполнить значениями 8 по индексам 2, 3

copyWithin()

Метод **copyWithin**() копирует срез массива в новую позицию внутри массива. Он модифицирует массив на месте и возвращает модифицированный массив, но не изменяет длину массива. В первом аргументе задается целевой индекс, по которому будет копироваться первый элемент. Во втором аргументе указывается индекс первого копируемого элемента.

Если второй аргумент опущен, тогда применяется 0. В третьем аргументе задается конец среза элементов, подлежащих копированию. Если третий аргумент опущен, то используется длина массива. Копироваться будут элементы от начального индекса и вплоть до конечного индекса, не включая его. Как и в slice (), вы можете указывать индексы относительно конца массива, передавая отрицательные числа:

let a = [1,2,3,4,5];
a.copyWithin (1) // => [1,1,2,3,4]: копировать элементы массива в позиции, начиная с первого
a.copyWithin (2, 3, 5) // => [1,1,3,4,4]: копировать последние 2 элемента по индексу 2
a.copyWithin (0, -2) => [4,4,3,4,4]: отрицательные смещения тоже работают

___

Методы поиска и сортировки массивов

indexOf() и lastIndexOf()

Методы indexOf () и lastIndexOf () ищут в массиве элемент с указанным значением и возвращают индекс первого найденного такого элемента или -1, если ничего не было найдено. Метод indexOf () производит поиск в массиве с начала до конца, а метод lastIndexOf () - с конца до начала:

let a = [0,1,2,1,0];
a.indexOf (1) // => 1: a [1] равно 1
a.lastIndexOf (1) // => 3: a [3] равно 1
a.indexof (3) // => -1: нет элементов со значением 3

includes ()

Метод includes () ИЗ ES2016 принимает единственный аргумент и возвращает true, если массив содержит значение аргумента, или false в противном случае. Он не сообщает индекс, где находится значение, а только то, что оно существует. Метод includes () фактически является проверкой членства во множестве для массивов. Тем не менее, имейте в виду, что массивы не считаются эффективным представлением для множеств, и если

вы работаете с немалым количеством элементов, то должны использовать настоящий объект Set (см. подраздел 11.1.1).

Метод includes () отличается от метода indexOf () одной важной особенностью. Метод indexOf () проверяет равенство с применением такого же алгоритма, как у операции ===, и этот алгоритм проверки равенства считает значение "не число" отличным от любого другого значения, включая самого себя.

Метод includes () использует немного отличающуюся версию алгоритма проверки равенства, которая рассматривает значение NaN как равное самому себе. Таким образом, indexof () не будет обнаруживать значение NaN в массиве, но includes () будет:

```
let a = [1, true, 3, NaN];
a.includes (true) // => true
a.includes (2) // => false
a.includes (NaN) => true
a.indexOf (NaN) // => -1; indexOf не может отыскивать NaN
```

sort()

Метод sort() сортирует элементы массива на месте и возвращает отсортированный массив. При вызове без аргументов sort ( ) сортирует элементы массива в алфавитном порядке (при необходимости временно преобразуя их в строки для выполнения сравнений):

```
let a = ["banana", "cherry", , "apple"];
a.sort (); // a == ["apple", "banana", "cherry")

let a = [33, 4, 1111, 222]:
a.sort () ; // a == [1111, 222, 33, 4]; алфавитный порядок
a.sort (function (a,b) { // Передать функцию сравнения
return a-b; // Возвращает число < 0, 0 или > 0 в зависимости от порядка
}); //a == (4, 33, 222, 1111]: числовой порядок
a.sort ((a,b) => b-a): //a == [1111, 222, 33, 4]: обратный числовой порядок
```

В качестве еще одного примера сортировки элементов массива

вы можете выполнить нечувствительную к регистру сортировку в алфавитном порядке массива строк, передав sort () функцию сравнения, которая приводит оба своих аргумента к нижнему регистру (с помощью метода toLowerCase ()), прежде чем сравнивать их:

```
let a = ["ant", "Bug", "cat", "Dog"];
a.sort() ; // a == ["Bug", "Dog", "ant", "cat"]; сортировка, чувствительная к регистру

a.sort(function (st) {
	let a = s.toLowerCase () ;
	let b = t.toLowerCase () ;

	if (a < b) return -1;
	if (a > b) return 1;
	return 0;
}): // a == ["ant", "Bug", "cat", "Dog"]; сортировка, нечувствительная к регистру
```

reverse()

Метод reverse () изменяет на противоположный порядок следования эле-
ментов в массиве и возвращает обращенный массив. Он делает это на месте;
другими словами, reverse () не создает новый массив с переупорядоченными
элементами, а взамен переставляет их в существующем массиве:

```
let a = [1,2,3];
a.reverse () ; // a == [3,2,1]
```

___

Преобразования массивов в строки

```
let a = [1, 2, 3];
a.join () 11 => "1,2,3"
a.join (" ") ¡ => "1 2 3"
a.join ("") # => "1234
```

let b = new Array (10); // Массив с длиной 10 без элементов
b.join ("-") => 1. -": строка из 9 дефисов

Метод join () является противоположностью метода String.split(), который создает массив, разбивая строку на части.

Как и все объекты JavaScript, массивы метод toString (). Для массива он работает подобно методу join(), вызываемому без аргументов:

[1,2,3].toString() => "1,2,3"
["a", "b", "c"].toString() // => "a,b, c"
[1, [2, "c"].toString () // => "1,2, c"

Обратите внимание, что вывод не включает квадратные скобки или ограничитель любого другого вида вокруг значения типа массива.

Метод toLocaleString () представляет собой локализованную версию tostring (). Он преобразует каждый элемент массива в строку, вызывая метод tolocaleString () элемента, после чего выполняет конкатенацию результирующих строк с использованием специфической к локали (определенной реализацией) строки разделителя.

___

Статические функции массивов

Конструктор Array, а не на массивах. Array.of() и Array.from() - это фабричные методы для создания новых массивов. Они рассматривались в подразделах

Еще одна статическая функция массива, Array. isArray (), полезна для определения, является ли неизвестное значение массивом:

Array.isArray ([]) // => true
Array.isArray ({}) / => false

Объекты, похожие на массивы

• Свойство length автоматически обновляется при добавлении

новых элементов в массив.
• Установка length в меньшее значение усекает массив.
• Массивы наследуют полезные методы от Array. prototype.
• Array. isArray() возвращает true для массивов.

```
// Определяет, является ли о объектом, похожим на массив.
// Строки и функции имеют числовые свойства length, но
исключаются
// проверкой typeof. В коде JavaScript на стороне клиента
текстовые
// узлы DOM имеют числовое свойство length, и может
понадобиться их
// исключить с помощью дополнительной проверки o.nodeType !
== 3.
function isArrayLike (o) {
      if (o && // o - не null, undefined и т.д.
            typeof o === "object" && // o - объект
                  Number.isFinite (o.length) && // o.length - конечное
число
                        o. length >= 0 && // o. length - неотрицательное
                              Number. isInteger (o. length) && / / o. length -
целое число
                                    o. length < 4294967295) 1 // o. length <
2^32 - 1
                              return true; // Тогда объект о похож на массив
                  } else {
            return false; // Иначе объект о не похож на массив



let a = ("0": "a", "1": "b", "2": "c", length: 3); // Объект, похожий на
массив

Array.prototype.join.call (a, "+") ¡ => "a+b+c"

Array .prototype.map.call (a, x => x.toUpperCase ()) // => ("A", "B",
"c")

Array.prototype.slice.call (a, 0) // => ["a", "b", "c"): копирование в
подлинный массив
Array.from (a) // => ["a", "b", "c"]: более легкое копирование
```

Строки как массивы

Array.prototype.join.call ("JavaScript", " ") // => "JavaScript"

Имейте в виду, что строки являются неизменяемыми значениями, поэтому при обхождении с ними как с массивами они будут массивами, доступными только для чтения. Методы массивов вроде push (), sort (), reverse () и splice () модифицируют массив на месте и со строками не работают. Однако попытка модифицировать строку с использованием метода массива не приводит к ошибке: она просто молча заканчивается неудачей.

___

Резюме

• Литералы типа массивов записываются как списки разделенных запятыми значений внутри квадратных скобок.
• Доступ к индивидуальным элементам массива производится путем указания желаемого индекса в массиве внутри квадратных скобок.
• Цикл for/of и операция распространения . .., введенные в ES6, являются чрезвычайно удобными способами итерации по массивам.
• В классе Array определен богатый набор методов для манипулирования массивами, и вы обязательно должны ознакомиться с API-интерфейсом Array.