JavaScript - массивы

массив и методы массива

Массив

Массив в JavaScript — это структура данных, которая используется для хранения упорядоченных коллекций элементов. Каждый элемент в массиве имеет индекс, который начинается с нуля. Массивы могут содержать элементы разных типов, включая числа, строки, объекты и другие массивы.

Массив – это особый подвид объектов. Квадратные скобки, используемые для того, чтобы получить доступ к свойству arr[0] – это по сути обычный синтаксис доступа по ключу, как obj[key], где в роли obj у нас arr, а в качестве ключа – числовой индекс.

Массивы расширяют объекты, так как предусматривают специальные методы для работы с упорядоченными коллекциями данных, а также свойство length. Но в основе всё равно лежит объект.

Следует помнить, что в JavaScript существует 8 основных типов данных. Массив является объектом и, следовательно, ведет себя как объект.

Объявление массива

В JavaScript массивы можно создавать несколькими способами. Наиболее распространенные из них:

Использование квадратных скобок:

```
let fruits = ["Apple", "Banana", "Cherry"];
```

Использование конструктора Array:

```
let numbers = new Array(1, 2, 3, 4, 5);
```

В массиве могут храниться элементы любого типа.

Доступ к элементам массива

Мы можем получить элемент, указав его номер в квадратных скобках, также мы можем заменить элемент или добавить новый к существующему массиву. (По аналогии с манипуляциями над объектом)

```
let fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits[0] ); // Яблоко
alert( fruits[1] ); // Апельсин
alert( fruits[2] ); // Слива
```

```
fruits[2] = 'Груша'; // теперь ["Яблоко", "Апельсин", "Груша"]
```

```
fruits[3] = 'Лимон'; // теперь ["Яблоко", "Апельсин", "Груша", "Лимон"]
```

Длина массива (размер массива)

Общее число элементов массива содержится в его свойстве length:

Свойство length автоматически обновляется при изменении массива. Если быть точными, это не количество элементов массива, а наибольший цифровой индекс плюс один.

```
let fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits.length ); // 3
```

```
let fruits = [];
fruits[123] = "Яблоко";

alert( fruits.length ); // 124
```

Фишка length

С помощью length можно обнулять массив, как это работает?:

Ещё один интересный факт о свойстве length – его можно перезаписать.

Если мы вручную увеличим его, ничего интересного не произойдёт. Зато, если мы уменьшим его, массив станет короче.

Таким образом, самый простой способ очистить массив – это arr.length = 0.

```
let arr = [1, 2, 3, 4, 5];

arr.length = 2; // укорачиваем до двух элементов
alert( arr ); // [1, 2]

arr.length = 5; // возвращаем length как было
alert( arr[3] ); // undefined: значения не восстановились
```

Получение последних элементов при помощи «at»

Допустим, нам нужен последний элемент массива.

Некоторые языки программирования позволяют использовать отрицательные индексы для той же цели, как-то так: fruits[-1].

Однако, в JavaScript такая запись не сработает. Её результатом будет undefined, поскольку индекс в квадратных скобках понимается буквально.

Мы можем явно вычислить индекс последнего элемента, а затем получить к нему доступ вот так: fruits[fruits.length - 1].

К счастью, есть более короткий синтаксис: fruits.at(-1):

Другими словами, arr.at(i):

- это ровно то же самое, что и arr[i], если i >= 0.
- для отрицательных значений i, он отступает от конца массива.

```
let fruits = ["Apple", "Orange", "Plum"];
alert( fruits[fruits.length-1] ); // Plum
```

```
let fruits = ["Apple", "Orange", "Plum"];
// то же самое, что и fruits[fruits.length-1]
alert( fruits.at(-1) ); // Plum
```

Как добавить в массив что то и убрать?

Методы pop/push, shift/unshift

Методы, работающие с концом массива:

- push
- pop

Методы, работающие с началом массива:

- shift
- unshift

push (добавить)

Добавляет элемент в конец массива:

Вызов fruits.push(...) равнозначен fruits[fruits.length] = ...

```
let fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша
```

pop (убрать)

Удаляет последний элемент из массива и возвращает его:

И fruits.pop() и fruits.at(-1) возвращают последний элемент массива, но fruits.pop() также изменяет массив, удаляя его.

Вызов fruits.push(...) равнозначен fruits.length = fruits.length - 1;

```
let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.pop() ); // удаляем "Груша" и выводим его

alert( fruits ); // Яблоко, Апельсин
```

unshift (анти-сдвиг, добавить в начало)

Добавляет элемент в начало массива:

```
let fruits = ["Апельсин", "Груша"];

fruits.unshift('Яблоко');

alert( fruits ); // Яблоко, Апельсин, Груша
```

shift (сдвиг, убрать из начала и сдвинуть весь массив)

Удаляет из массива первый элемент и возвращает его:

```
let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.shift() ); // удаляем Яблоко и выводим его

alert( fruits ); // Апельсин, Груша
```

Методы push и unshift могут добавлять сразу несколько элементов:

```
let fruits = ["Яблоко"];

fruits.push("Апельсин", "Груша");
fruits.unshift("Ананас", "Лимон");

// ["Ананас", "Лимон", "Яблоко", "Апельсин", "Груша"]
alert( fruits );
```

Эффективность

Методы push/pop выполняются быстро, а методы shift/unshift – медленно.

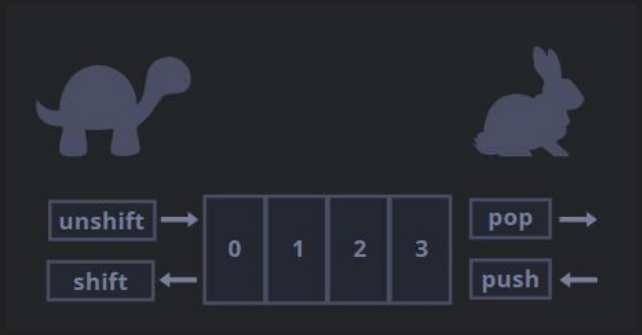
Просто взять и удалить элемент с номером 0 недостаточно. Нужно также заново пронумеровать остальные элементы.

Операция shift должна выполнить 3 действия:

1. Удалить элемент с индексом 0.
2. Сдвинуть все элементы влево, заново пронумеровать их, заменив 1 на 0, 2 на 1 и т.д.
3. Обновить свойство length .

Чем больше элементов содержит массив, тем больше времени потребуется для того, чтобы их переместить, больше операций с памятью.

Метод pop не требует перемещения, потому что остальные элементы остаются с теми же индексами. Именно поэтому он выполняется очень быстро.



Многомерные массивы

Массивы могут содержать элементы, которые тоже являются массивами. Это можно использовать для создания многомерных массивов, например, для хранения матриц:

```
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

alert( matrix[1][1] ); // 5, центральный элемент
```

toString

Массивы по-своему реализуют метод toString, который возвращает список элементов, разделенных запятыми.

```
let arr = [1, 2, 3];

alert( arr ); // 1,2,3
alert( String(arr) === '1,2,3' ); // true
```

```
alert( [] + 1 ); // "1"
alert( [1] + 1 ); // "11"
alert( [1,2] + 1 ); // "1,21"
```

```
alert( "" + 1 ); // "1"
alert( "1" + 1 ); // "11"
alert( "1,2" + 1 ); // "1,21"
```

new Array()

Если new Array вызывается с одним аргументом, который представляет собой число, он создает массив без элементов, но с заданной длиной.

Как мы видим, в коде, представленном выше, в new Array(number) все элементы равны undefined.

Чтобы избежать появления таких неожиданных ситуаций, мы обычно используем квадратные скобки, если, конечно, не знаем точно, что по какой-то причине нужен именно Array.

```
let arr = new Array("Яблоко", "Груша", "и тд");
```

```
let arr = new Array(2); // создастся ли массив [2]?

alert( arr[0] ); // undefined! нет элементов.

alert( arr.length ); // length 2
```

Перебор элементов

```
let arr = ["Яблоко", "Апельсин", "Груша"];

for (let i = 0; i < arr.length; i++) {
  alert( arr[i] );
}
```

```
let arr = ["Яблоко", "Апельсин", "Груша"];

for (let key in arr) {
  alert( arr[key] ); // Яблоко, Апельсин, Груша
}
```

```
let fruits = ["Яблоко", "Апельсин", "Слива"];

// проходит по значениям
for (let fruit of fruits) {
  alert( fruit );
}
```

```
arr.forEach(function(item, index, array) {
  // ... делать что-то с item
});
```

Методы массива

Добавление/удаление элементов:

- arr.push(...items) – добавляет элементы в конец,
- arr.pop() – извлекает элемент из конца,
- arr.shift() – извлекает элемент из начала,
- arr.unshift(...items) – добавляет элементы в начало.

Методы посложнее:

- splice
- slice
- concat
- forEach
- find
- filter
- findIndex и findLastIndex

delete (кажется решение?)

Так как массивы – это объекты, то можно попробовать delete:

Элемент был удален, но в массиве всё ещё три элемента, мы можем увидеть, что arr.length == 3.

Это естественно, потому что delete obj.key удаляет значение по ключу key. Это всё, что он делает. Хорошо для объектов. Но для массивов мы обычно хотим, чтобы оставшиеся элементы сдвинулись и заняли освободившееся место. Мы ждём, что массив станет короче.

```
let arr = ["I", "go", "home"];

delete arr[1]; // удалить "go"

alert( arr[1] ); // undefined

// теперь arr = ["I",  , "home"];
alert( arr.length ); // 3
```

splice

Метод arr.splice – это универсальный «швейцарский нож» для работы с массивами. Умеет всё: добавлять, удалять и заменять элементы.

Он изменяет arr начиная с индекса start: удаляет deleteCount элементов и затем вставляет elem1, ..., elemN на их место. Возвращает массив из удаленных элементов.

```
let arr = ["Я", "изучаю", "JavaScript"];

arr.splice(1, 1); // начиная с индекса 1, удалить 1 элемент

alert( arr ); // осталось ["Я", "JavaScript"]
```

```
let arr = ["Я", "изучаю", "JavaScript", "прямо", "сейчас"];

// удалить 3 первых элемента и заменить их другими
arr.splice(0, 3, "Давай", "танцевать");

alert( arr ) // теперь ["Давай", "танцевать", "прямо", "сейчас"]
```

```
let arr = ["Я", "изучаю", "JavaScript", "прямо", "сейчас"];

// удалить 2 первых элемента
let removed = arr.splice(0, 2);

alert( removed ); // "Я", "изучаю" <-- массив из удалённых элементов
```

```
arr.splice(start[, deleteCount, elem1, ..., elemN])
```

slice

Метод arr.slice намного проще, чем похожий на него arr.splice.

Он возвращает новый массив, в который копирует все элементы с индекса start до end (не включая end). start и end могут быть отрицательными, в этом случае отсчет позиции будет вестись с конца массива.

Это похоже на строковый метод str.slice, но вместо подстрок возвращает подмассивы.

Можно вызвать slice без аргументов: arr.slice() создает копию arr. Это часто используют, чтобы создать копию массива для дальнейших преобразований, которые не должны менять исходный массив.

```
arr.slice([start], [end])
```

```
let arr = ["t", "e", "s", "t"];

alert( arr.slice(1, 3) ); // e,s (копирует с 1 до 3)

alert( arr.slice(-2) ); // s,t (копирует с -2 до конца)
```

concat

Метод arr.concat создаёт новый массив, в который копирует данные из других массивов и дополнительные значения.

Он принимает любое количество аргументов, которые могут быть как массивами, так и простыми значениями.

В результате – новый массив, включающий в себя элементы из arr, затем arg1, arg2 и так далее.

Если аргумент argN – массив, то копируются все его элементы. Иначе копируется сам аргумент.

```
arr.concat(arg1, arg2...)
```

```
let arr = [1, 2];

// создать массив из: arr и [3,4]
alert( arr.concat([3, 4]) ); // 1,2,3,4

// создать массив из: arr и [3,4] и [5,6]
alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6

// создать массив из: arr и [3,4], потом добавить значения 5 и 6
alert( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6
```

Перебор: forEach

```
arr.forEach(function(item, index, array) {
  // ... делать что-то с item
});
```

Метод arr.forEach позволяет запускать функцию для каждого элемента массива.

Результат функции (если она что-то возвращает) отбрасывается и игнорируется.

```
["Бильбо", "Гэндальф", "Назгул"].forEach((item, index, array) => {
  alert(`У ${item} индекс ${index} в ${array}`);
});
```

Поиск элемента (find)

Функция вызывается по очереди для каждого элемента массива:

item – очередной элемент.

index – его индекс.

array – сам массив.

Если функция возвращает true, поиск прерывается и возвращается item. Если ничего не найдено, возвращается undefined.

```
let result = arr.find(function(item, index, array) {
  // если true - возвращается текущий элемент и перебор прерывается
  // если все итерации оказались ложными, возвращается undefined
});
```

Поиск индекса (findIndex & findLastIndex)

У метода arr.findIndex такой же синтаксис, но он возвращает индекс, на котором был найден элемент, а не сам элемент. Значение -1 возвращается, если ничего не найдено.

Метод arr.findLastIndex похож на findIndex, но ищет справа налево, наподобие lastIndexOf.

```
let users = [
  {id: 1, name: "Вася"},
  {id: 2, name: "Петя"},
  {id: 3, name: "Маша"},
  {id: 4, name: "Вася"}
];

// Найти индекс первого Васи
alert(users.findIndex(user => user.name == 'Вася')); // 0

// Найти индекс последнего Васи
alert(users.findLastIndex(user => user.name == 'Вася')); // 3
```

Фильтрация (тоже своего рода поиск) - filter

Метод find ищет один (первый) элемент, который заставит функцию вернуть true.

Если найденных элементов может быть много, можно использовать arr.filter(fn).

Синтаксис схож с find, но filter возвращает массив из всех подходящих элементов:

```
let users = [
  {id: 1, name: "Вася"},
  {id: 2, name: "Петя"},
  {id: 3, name: "Маша"}
];

// возвращает массив, состоящий из двух первых пользователей
let someUsers = users.filter(item => item.id < 3);

alert(someUsers.length); // 2
```

```
let results = arr.filter(function(item, index, array) {
  // если `true` -- элемент добавляется к results и перебор продолжается
  // возвращается пустой массив в случае, если ничего не найдено
});
```

Преобразование массива (методы экземпляра)

- map
  “маппинг” массива - трансформация массива в другой массив
- sort
  сортировка (мутирует используемый массив)
- reverse
  переделывает массив “задом-наперед”
- split и join
  разбивать строку в массив и собирать массив в строку
- reduce/reduceRight
  сложный комплексный метод (универсальный)
- some/every
  для проверки массива на наличие требуемого элемента

MAP (не меняет исходный массив)

Он вызывает функцию для каждого элемента массива и возвращает массив результатов выполнения этой функции.

На основе одного массива “мапит” другой

```
let result = arr.map(function(item, index, array) {
  // возвращается новое значение вместо элемента
});
```

```
let lengths = ["Бильбо", "Гэндальф", "Назгул"].map(item => item.length);
alert(lengths); // 6,8,6
```

SORT (меняет исходный массив)

Вызов arr.sort() сортирует массив на месте, меняя в нём порядок элементов.

Он также возвращает отсортированный массив, но обычно возвращаемое значение игнорируется, так как изменяется сам arr.

По умолчанию элементы сортируются как строки.

Тут функционал “колбека” данного метода очень тесно связан с алгоритмами сортировки

```
function compareNumeric(a, b) {
  if (a > b) return 1;
  if (a == b) return 0;
  if (a < b) return -1;
}

let arr = [ 1, 2, 15 ];

arr.sort(compareNumeric);

alert(arr);  // 1, 2, 15
```

REVERSE (меняет исходный массив)

Метод arr.reverse меняет порядок элементов в arr на обратный.

Он также возвращает массив arr с измененным порядком элементов.

```
let arr = [1, 2, 3, 4, 5];
arr.reverse();

alert( arr ); // 5,4,3,2,1
```

SPLIT & JOIN

Метод split - (метод экземпляра строки “string”) - преобразует строку в массив за разделитель принимает заданную методу строку также получаемый массив может быть ограничен с помощью второго аргумента

Вызов arr.join(glue) делает в точности противоположное split. Он создаёт строку из элементов arr, вставляя glue между ними.

```
let arr = 'Вася, Петя, Маша, Саша'.split(', ', 2);

alert(arr); // Вася, Петя
```

```
let str = "тест";

alert( str.split('') ); // т,е,с,т
```

```
let arr = ['Вася', 'Петя', 'Маша'];

let str = arr.join(';'); // объединить массив в строку через ;

alert( str ); // Вася;Петя;Маша
```

REDUCE & REDUCERIGHT

```
let value = arr.reduce(function(accumulator, item, index, array) {
  // ...
}, [initial]);
```

Функция применяется по очереди ко всем элементам массива и «переносит» свой результат на следующий вызов.

Аргументы:

- accumulator – результат предыдущего вызова этой функции, равен initial при первом вызове (если передан initial),
- item – очередной элемент массива,
- index – его позиция,
- array – сам массив.

При вызове функции результат ее предыдущего вызова передается на следующий вызов в качестве первого аргумента.

Так, первый аргумент является по сути аккумулятором, который хранит объединенный результат всех предыдущих вызовов функции. По окончании он становится результатом reduce.

Метод arr.reduceRight работает аналогично, но проходит по массиву справа налево.

Пример алгоритма REDUCE

Функция, переданная в reduce, использует только два аргумента, этого обычно достаточно.

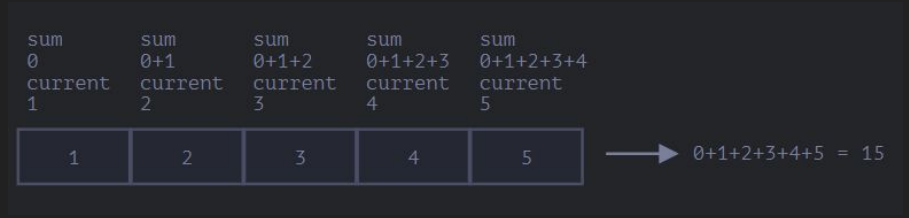
Разберём детально как это работает.

- При первом запуске sum равен initial (последний аргумент reduce), то есть 0, а current – первый элемент массива, равный 1. Таким образом, результат функции равен 1.
- При втором запуске sum = 1, к нему мы добавляем второй элемент массива (2) и возвращаем.
- При третьем запуске sum = 3, к которому мы добавляем следующий элемент, и так далее…

```
let arr = [1, 2, 3, 4, 5];

let result = arr.reduce((sum, current) => sum + current, 0);

alert(result); // 15
```



	sum	current	result
первый вызов	0	1	1
второй вызов	1	2	3
третий вызов	3	3	6
четвёртый вызов	6	4	10
пятый вызов	10	5	15

SOME & EVERY

В JavaScript методы some и every используются для проверки элементов массива на соответствие определенному условию. Оба метода принимают в качестве аргумента функцию обратного вызова (callback), которая выполняется для каждого элемента массива.

Метод some проверяет, удовлетворяет ли хотя бы один элемент массива условию, заданному в функции обратного вызова. Если хотя бы один элемент массива соответствует условию, метод возвращает true. В противном случае возвращает false.

Метод every проверяет, удовлетворяют ли все элементы массива условию, заданному в функции обратного вызова. Если все элементы массива соответствуют условию, метод возвращает true. В противном случае возвращает false.

```
array.some(callback(element, index, array), thisArg);
```

```
let numbers = [1, 2, 3, 4, 5];

let hasEvenNumber = numbers.some(function(element) {
    return element % 2 === 0;
});

console.log(hasEvenNumber); // true, так как есть хотя бы одно четное число
```

```
array.every(callback(element, index, array), thisArg);
```

```
let numbers = [1, 2, 3, 4, 5];

let allPositive = numbers.every(function(element) {
    return element > 0;
});

console.log(allPositive); // true, так как все числа положительные
```

Array.isArray (статический метод)

Массивы не образуют отдельный тип данных. Они основаны на объектах.

Поэтому typeof не может отличить простой объект от массива:

```
alert(typeof {}); // object
alert(typeof []); // тоже object
```

…Но массивы используются настолько часто, что для этого придумали специальный метод: Array.isArray(value). Он возвращает true, если value массив, и false, если нет.

```
alert(Array.isArray({})); // false

alert(Array.isArray([])); // true
```

Ресурсы

Массивы - статья на русском языке (учебник) - ТЫК

Методы массивов - статья на русском (учебник) - ТЫК

MDN документация массивов (не все переведено) - ТЫК