**Масиви з числовими індексами.** **Методи.**

**1. Масиви з числовими індексами.**

**2. Методи.**

Масив - різновид об'єкта, який призначений для зберігання пронумерованих значень і пропонує додаткові методи для зручного маніпулювання такою колекцією.

Вони зазвичай використовуються для зберігання впорядкованих колекцій даних, наприклад - списку товарів на сторінці, студентів в групі тощо

**Оголошення**

Синтаксис для створення нового масиву - квадратні дужки зі списком елементів всередині.

Порожній масив:

var arr = [];

Масив fruits з трьома елементами:

var fruits = ["Яблуко", "Апельсин", "Слива"];

**Елементи нумеруються, починаючи з нуля.**

Щоб отримати потрібний елемент з масиву - вказується його номер в квадратних дужках:

var fruits = ["Яблуко", "Апельсин", "Слива"];

alert( fruits[0] ); // Яблуко

alert( fruits[1] ); // Апельсин

alert( fruits[2] ); // Слива

Елемент можна завжди замінити:

fruits[2] = 'Груша'; // тепер ["Яблуко", "Апельсин", "Груша"]

…Чи додати:

fruits[3] = 'Лимон'; // тепер ["Яблуко", "Апельсин", "Груша", "Лимон"]

Загальна кількість об'єктів, що зберігаються в масиві, міститься в його властивості length:

var fruits = ["Яблуко", "Апельсин", "Груша"];

alert( fruits.length ); // 3

**Через alert можна вивести і масив цілком.**

При цьому його елементи будуть перераховані через кому:

var fruits = ["Яблуко", "Апельсин", "Груша"];

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

**У масиві може зберігатися будь-яке число елементів будь-якого типу.**

// мікс значень

var arr = [ 1, 'Ім’я', { name: 'Петя' }, true ];

// отримати об’єкт з масиву і тут же – його властивість

alert( arr[2].name ); // Петя

**Методи pop / push, shift / unshift**

Одне із застосувань масиву - це черга. У класичному програмуванні так називають впорядковану колекцію елементів, таку що елементи додаються в кінець, а обробляються - з початку.



У реальному житті ця структура даних зустрічається дуже часто. Наприклад, черга повідомлень, які треба показати на екрані.

Дуже близька до черги ще одна структура даних: стек. Це така колекція елементів, в якій нові елементи додаються в кінець і беруться з кінця.



Наприклад, стеком є колода карт, в яку нові карти кладуться зверху, і беруться - теж зверху.

Для того, щоб реалізовувати ці структури даних, і просто для більш зручної роботи з початком і кінцем масиву існують спеціальні методи.

**Кінець масиву**

**Pop**

Видаляє останній елемент з масиву і повертає його:

var fruits = ["Яблуко", "Апельсин", "Груша"];

alert( fruits.pop() ); // видалили "Груша"

alert( fruits ); // Яблуко, Апельсин

**push**

Додає елемент в *кінець* масиву:

var fruits = ["Яблуко", "Апельсин"];

fruits.push("Груша");

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

Виклик fruits.push(...) рівнозначний fruits[fruits.length] = ....

**Початок масиву**

**shift**

Видаляє з масиву *перший* елемент і повертає його:

var fruits = ["Яблуко", "Апельсин", "Груша"];

alert( fruits.shift() ); // видалили Яблуко

alert( fruits ); // Апельсин, Груша

**unshift**

Додає елемент в *початок* масиву:

var fruits = ["Апельсин", "Груша"];

fruits.unshift('Яблуко');

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

Методи push  і unshift  можуть додавати відразу по кілька елементів:

var fruits = ["Яблуко"];

fruits.push("Апельсин", "Персик");

fruits.unshift("Ананас", "Лимон");

// результат: ["Ананас", "Лимон", "Яблуко", "Апельсин", "Персик"]

alert( fruits );

**Внутрішній устрій масиву**

Масив - це об'єкт, де в якості ключів обрані цифри, з додатковими методами і властивістю length.

Так як це об'єкт, то в функцію він передається по посиланню:

function eat(arr) {

arr.pop();

}

var arr = ["нам", "не", "страшний", "сірий", "вовк"]

alert( arr.length ); // 5

eat(arr);

eat(arr);

alert( arr.length ); // 3, в функцію масив не скопійований, а передані посилання

**Ще один наслідок - можна привласнювати в масив будь-які властивості.**

Наприклад:

var fruits = []; // створити масив

fruits[99999] = 5; // присвоїти властивість з любим номером

fruits.age = 25; // назначити властивість зі строковим ім’ям

... Але масиви для того і придумані в JavaScript, щоб зручно працювати саме з впорядкованими, нумерованими даними. Для цього в них існують спеціальні методи і властивість length.

Як правило, немає причин використовувати масив як звичайний об'єкт, хоча технічно і це можливо.

**Виведення масиву з «дірками»**

Якщо в масиві є пропущені індекси, то при виводі в більшості браузерів з'являються «зайві» коми, наприклад:

var a = [];

a[0] = 0;

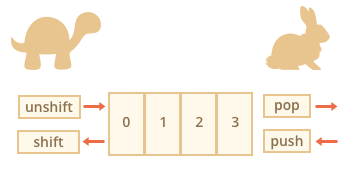
a[5] = 5;

alert( a ); // 0,,,,,5

Ці коми з'являються тому, що алгоритм виведення масиву йде від 0 до arr.length і виводить все через кому. Відсутність значень дає кілька ком поспіль.

**Вплив на швидкодію**

Методи push/pop виконуються швидко, а shift/unshift - повільно.



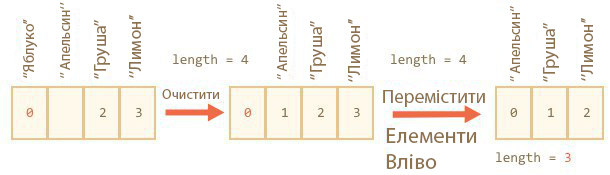
Щоб зрозуміти, чому працювати з кінцем масиву - швидше, ніж з його початком, розберемо докладніше, що відбувається під час операції:

fruits.shift(); // прибрати 1 елемент з початку

При цьому, так як всі елементи знаходяться в своїх осередках, просто видалити елемент з номером 0 недостатньо. Потрібно ще й перемістити інші елементи на їх нові індекси.

Операція shift повинна виконати три дії:

1. Видалити нульовий елемент.
2. Перемістити усі елементи вліво, з індексу 1 на 0, з 2 на 1 і так далі.
3. Оновити властивість length.



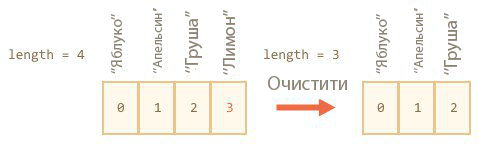
**Чим більше елементів в масиві, тим довше їх переміщати, це багато операцій з пам'яттю.**

Аналогічно працює unshift: щоб додати елемент в початок масиву, потрібно спочатку перенести всі існуючі елементи вправо, в збільшені індекси.

А що ж з push/pop? Їм якраз переміщати нічого не треба. Для того, щоб видалити елемент, метод pop очищає осередок і зменшує значення length.

Дії при операції:

fruits.pop(); // прибрати 1 елемент з кінця



**Переміщати при pop  не потрібно, так як інші елементи після цієї операції залишаються на тих же індексах.**

Аналогічно працює push.

**Перебір елементів**

Для перебору елементів зазвичай використовується цикл:

var arr = ["Яблуко", "Апельсин", "Груша"];

for (var i = 0; i < arr.length; i++) {

alert( arr[i] );

}

**Не використовуйте for..in для масивів**

Так як масив є об'єктом, то можливий і варіант for..in:

var arr = ["Яблуко", "Апельсин", "Груша"];

for (var key in arr) {

alert( arr[key] ); // Яблуко, Апельсин, Груша

}

Недоліки цього способу:

Цикл for..in виведе всі властивості об'єкта, а не тільки цифрові.

У браузері, при роботі з об'єктами сторінки, зустрічаються колекції елементів, які на вигляд масиви, але мають додаткові нецифрові властивості. При переборі таких «схожих на масив» колекцій через for..in ці властивості будуть виведені, а вони як раз не потрібні.

Бувають і бібліотеки, які надають такі колекції. Класичний for надійно виведе тільки цифрові властивості, що зазвичай і потрібно.

Цикл for (var i=0; i<arr.length; i++) в сучасних браузерах виконується в 10-100 разів швидше. Здавалося б, з вигляду він складніше, але браузер особливим чином оптимізує такі цикли.

Якщо коротко: цикл for(var i=0; i<arr.length...) надійніше і швидше.

**Особливості роботи length**

Вбудовані методи для роботи з масивом автоматично оновлюють його довжину length.

Довжина length - не кількість елементів масиву, а останній індекс + 1.

Так вже вона влаштована.

Це легко побачити на наступному прикладі:

var arr = [];

arr[1000] = true;

alert(arr.length); // 1001

До речі, якщо у вас елементи масиву нумеруються випадково або з великими пропусками, то варто подумати про те, щоб використовувати звичайний об'єкт. Масиви призначені саме для роботи з безперервною впорядкованої колекцією елементів.

**Використовуємо length для укорочення масиву**

Зазвичай нам не потрібно самостійно змінювати length... Але є один фокус, який можна провернути.

**При зменшенні length масив коротшає.**

Причому цей процес незворотний, тобто навіть якщо потім повернути length назад - значення не відновляться:

var arr = [1, 2, 3, 4, 5];

arr.length = 2; // вкоротити до 2 елементів

alert( arr ); // [1, 2]

arr.length = 5; // повернути length назад, як було

alert( arr[3] ); // undefined: значення не повернулись

Найпростіший спосіб очистити масив - це arr.length=0.

**Створення викликом new Array**

**new Array ()**

Існує ще один синтаксис для створення масиву:

var arr = new Array("Яблуко", "Груша", "и т.п.");

Він рідко використовується, тому що квадратні дужки [] коротше.

Крім того, у нього є одна особливість. Зазвичай new Array(елементи, ...) створює масив з даних елементів, але якщо у нього один аргумент-число new Array(число), то він створює масив *без елементів, але із заданою довжиною*.

Перевіримо це:

var arr = new Array(2, 3);

alert( arr[0] ); // 2, створений масив [2, 3], все ок

arr = new Array(2); // створить масив [2] ?

alert( arr[0] ); // undefined! у нас масив без елементів, довжини 2

Що ж таке цей «масив без елементів, але з довжиною»? Як таке можливо?

Виявляється, дуже навіть можливо і відповідає об'єкту {length: 2}. Одержаний масив поводиться так, як ніби його елементи рівні undefined.

Це може бути несподіваним сюрпризом, тому зазвичай використовують квадратні дужки.

**Багатовимірні масиви**

Масиви в JavaScript можуть містити як елементи інші масиви. Це можна використовувати для створення багатовимірних масивів, наприклад матриць:

var matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

alert( matrix[1][1] ); // центральний елемент

**Висновки:**

Масиви існують для роботи з впорядкованим набором елементів.

**Оголошення:**

var arr = [елемент1, елемент2...];

// new Array

var arr = new Array(елемент1, елемент2...);

При цьому new Array(число) створює масив заданої довжини, без елементів. Щоб уникнути помилок, кращий перший синтаксис.

**Властивість length** - довжина масиву. Якщо точніше, то останній індекс масиву плюс 1. Якщо її зменшити вручну, то масив коротшатиме. Якщо length більше реальної кількості елементів, то відсутні елементи рівні undefined.

Масив можна використовувати як чергу або стек.

**Операції з кінцем масиву:**

* arr.push(елемент1, елемент2...) додає елементи в кінець.
* var elem = arr.pop() видаляє і повертає останній елемент.

**Операції з початком масиву:**

* arr.unshift(елемент1, елемент2...) додає елементи в початок.
* var elem = arr.shift() видаляє і повертає перший елемент.

Ці операції перенумеровують всі елементи, тому працюють повільно.

**2.Масиви: Методи**

У цьому розділі ми розглянемо вбудовані методи масивів JavaScript.

**Метод split**

Ситуація з реального життя. Ми пишемо сервіс відсилання повідомлень і відвідувач вводить імена тих, кому його відправити: Маша, Петя, Марина, Василь.... Але нам-то набагато зручніше працювати з масивом імен, ніж з одним рядком.

На щастя, є метод split(s), який дозволяє перетворити рядок в масив, розбивши її по роздільнику s. У прикладі нижче таким роздільником є рядок з коми і пропуску.

var names = 'Маша, Петя, Марина, Василь';

var arr = names.split(', ');

for (var i = 0; i < arr.length; i++) {

alert( 'Вам повідомлення ' + arr[i] );

}

**Другий аргумент split**

У методу split є необов'язковий другий аргумент - обмеження на кількість елементів в масиві. Якщо їх більше, ніж вказано - залишок масиву буде відкинутий:

alert( "a,b,c,d".split(',', 2) ); // a,b

**Розбивка по буквах**

Виклик split з нового рядка розіб'є по буквах:

var str = "тест";

alert( str.split('') ); // т,е,с,т

**Метод join**

Виклик arr.join(str) робить в точності протилежне split. Він бере масив і склеює його в рядок, використовуючи str як роздільник.

Наприклад:

var arr = ['Маша', 'Петя', 'Марина', 'Василь'];

var str = arr.join(';');

alert( str ); // Маша;Петя;Марина;Василь

**new Array + join = Повторення рядка**

Код для повторення рядка 3 рази:

alert( new Array(4).join("ля") ); // ляляля

Як видно, new Array(4) робить масив без елементів довжини 4, який join об'єднує в рядок, вставляючи *між його елементами рядок* "ля".

В результаті, так як елементи порожні, виходить повторення рядка.

**Видалення з масиву**

Так як масиви є об'єктами, то для видалення ключа можна скористатися звичайним delete:

var arr = ["Я", "йду", "додому"];

delete arr[1]; // значення з індексом 1 видалено

// тепер arr = ["Я", undefined, "додому"];

alert( arr[1] ); // undefined

Так, елемент видалений з масиву, але не так, як нам цього хочеться. Утворилася «дірка».

Це тому, що оператор delete видаляє пару «ключ-значення». Це - все, що він робить. Звичайно ж при видаленні з масиву ми хочемо, щоб залишилися елементи зрушили і заповнили проміжок,.

Тому для видалення використовуються спеціальні методи: з початку - shift, з кінця - pop, а з середини - splice, з яким ми зараз познайомимося.

**Метод splice**

Метод splice - це універсальний розкладний ніж для роботи з масивами. Вміє все: видаляти елементи, вставляти елементи, замінювати елементи - по черзі і одночасно.

Його синтаксис:

**arr.splice(index[, deleteCount, elem1, ..., elemN])**

Видалити deleteCount елементів, починаючи з номера index, а потім вставити elem1, ..., elemN на їх місце. Повертає масив з видалених елементів.

Цей метод найпростіше зрозуміти, розглянувши приклади.

Почнемо з видалення:

var arr = ["Я", "вивчаю", "JavaScript"];

arr.splice(1, 1); // починаючи з позиції 1, видалити 1 елемент

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

У наступному прикладі ми видалимо 3 елементи і вставимо інші на їх місце:

var arr = ["Я", "зараз", "вивчаю", "JavaScript"];

// видалити 3 перших елемента і додати інші замість них

arr.splice(0, 3, "Ми", "вивчаємо")

alert( arr ) // тепер ["Ми", "вивчаємо", "JavaScript"]

Тут видно, що splice повертає масив з видалених елементів:

var arr = ["Я", "зараз", "вивчаю", "JavaScript"];

// видалити 2 перших елемента

var removed = arr.splice(0, 2);

alert( removed ); // "Я", "зараз" <-- array of removed elements

Метод splice також може вставляти елементи без видалення, для цього достатньо встановити deleteCount в 0:

var arr = ["Я", "вивчаю", "JavaScript"];

// з позиції 2

// видалити 0

// вставити "складну", "мову"

arr.splice(2, 0, "складну", "мову");

alert( arr ); // "Я", "вивчаю", "складну", "мову", "JavaScript"

Допускається використання від’ємного номера позиції, яка в цьому випадку відраховується з кінця:

var arr = [1, 2, 5]

// починаючи з позиції індексу -1 (перед останнім елементом)

// видалити 0 елементів,

// після вставити числа 3 и 4

arr.splice(-1, 0, 3, 4);

alert( arr ); // результат: 1,2,3,4,5

**Метод slice**

Метод slice(begin, end) копіює ділянку масиву від begin  до end, не включаючи end. Вихідний масив при цьому не змінюється.

Наприклад:

var arr = ["Чому", "потрібно", "вчити", "JavaScript"];

var arr2 = arr.slice(1, 3); // елементи 1, 2 (не включаючи 3)

alert( arr2 ); // потрібно, вчити

Аргументи поводяться так само, як і в строковому slice:

Якщо не вказати end - копіювання буде до кінця масиву:

var arr = ["Чому", "потрібно", "вчити", "JavaScript"];

alert( arr.slice(1) ); // взяти всі елементи, починаючи з номеру 1

Можна використовувати від’ємні індекси, вони відраховуються з кінця:

var arr2 = arr.slice(-2); // копіювати від 2-го елементу з кінця і далі

Якщо взагалі не вказати аргументів - скопіюється весь масив:

var fullCopy = arr.slice();

**Сортування, метод sort (fn)**

Метод sort() сортує масив на місці. Наприклад:

var arr = [ 1, 2, 15 ];

arr.sort();

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

Чи не помітили нічого дивного в цьому прикладі?

Порядок став 1, 15, 2, це точно не сортування чисел. Чому?

**Це сталося через те, що за замовчуванням sort сортує, перетворюючи елементи до рядка.**

Тому і порядок у них строковий, адже "2" > "15".

**Свій порядок сортування**

Для вказівки свого порядку сортування в метод arr.sort(fn) потрібно передати функцію fn від двох елементів, яка вміє порівнювати їх.

Внутрішній алгоритм функції сортування вміє сортувати будь-які масиви - апельсинів, яблук, користувачів, і тих і інших і третє - чого завгодно. Але для цього йому потрібно знати, як їх порівнювати. Цю роль і виконує fn.

Якщо цю функцію не вказати, то елементи сортуються як рядки.

Наприклад, вкажемо цю функцію явно, відсортуємо елементи масиву як числа:

function compareNumeric(a, b) {

if (a > b) return 1;

if (a < b) return -1;

}

var arr = [ 1, 2, 15 ];

arr.sort(compareNumeric);

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

Зверніть увагу, ми передаємо в sort() саме саму функцію compareNumeric, без виклику через дужки. Був би помилкою наступний код:

arr.sort( compareNumeric() ); // не спрацює

Як видно з прикладу вище, функція, передана sort, повинна мати два аргументи.

Алгоритм сортування, вбудований в JavaScript, буде передавати їй для порівняння елементи масиву. Вона повинна повертати:

Позитивне значення, якщо a > b,

Негативне значення, якщо a < b,

Якщо рівні - можна 0, але взагалі - не важливо, що повертати, якщо їх взаємний порядок не має значення.

**Reverse**

Метод arr.reverse () змінює порядок елементів в масиві на зворотний.

var arr = [1, 2, 3];

arr.reverse();

alert( arr ); // 3,2,1

**Concat**

Метод arr.concat (value1, value2, ... valueN) створює новий масив, в який копіюються елементи з arr, а також value1, value2, ... valueN.

Наприклад:

var arr = [1, 2];

var newArr = arr.concat(3, 4);

alert( newArr ); // 1,2,3,4

У concat є одна кумедна особливість.

Якщо аргумент concat - масив, то concat додає елементи з нього.

Наприклад:

var arr = [1, 2];

var newArr = arr.concat([3, 4], 5); // те саме, що arr.concat(3,4,5)

alert( newArr ); // 1,2,3,4,5

**indexOf / lastIndexOf**

Ці методи не підтримуються в IE8-. Для їх підтримки підключіть бібліотеку ES5-shim.

Метод «arr.indexOf (searchElement [, fromIndex])» повертає номер елемента searchElement в масиві arr або -1, якщо його немає.

Пошук починається з номера fromIndex, якщо він вказаний. Якщо немає - з початку масиву.

**Для пошуку використовується суворе порівняння ===.**

Наприклад:

var arr = [1, 0, false];

alert( arr.indexOf(0) ); // 1

alert( arr.indexOf(false) ); // 2

alert( arr.indexOf(null) ); // -1

Як ви могли помітити, по синтаксису він повністю аналогічний методу indexOf для рядків.

Метод «arr.lastIndexOf (searchElement [, fromIndex])» шукає справа-наліво: з кінця масиву або з номера fromIndex, якщо він вказаний.

**Object.keys (obj)**

Раніше ми говорили про те, що властивості об'єкта можна перебрати в циклі for..in.

Якщо ми хочемо працювати з ними у вигляді масиву, то до наших послуг - чудовий метод Object.keys (obj). Він підтримується всюди, крім IE8-:

var user = {

name: "Петя",

age: 30

}

var keys = Object.keys(user);

alert( keys ); // name, age

**Висновки:**

Методи масивів:

* push/pop, shift/unshift, splice - для додавання і видалення елементів.
* join/split - для перетворення рядка в масив і назад.
* slice - копіює ділянку масиву.
* sort - для сортування масиву. Якщо не передати функцію порівняння - сортує елементи як рядки.
* reverse - змінює порядок елементів на зворотний.
* concat - об'єднує масиви.
* indexOf/lastIndexOf - повертають позицію елемента в масиві (не підтримуються в IE8-).

Додатково:

* Object.keys(obj) повертає масив властивостей об'єкта.

Вивчених нами методів досить в 95% випадках, але існують і інші. Для знайомства з ними рекомендується заглянути в довідник Array і Array в Mozilla Developer Network.