Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Массив – разновидность объекта, которая предназначена для хранения пронумерованных значений и предлагает дополнительные методы для удобного манипулирования такой коллекцией. Они обычно используются для хранения упорядоченных коллекций данных, например – списка товаров на странице, студентов в группе и т.п.

Объявление

Синтаксис для создания нового массива – квадратные скобки со списком элементов внутри.

```
Пустой массив:
const arr = [];

Массив fruits с тремя элементами:
constfruits = ["Яблоко", "Апельсин", "Слива"];

Элементы нумеруются, начиная с нуля.
Чтобы получить нужный элемент из массива – указывается его номер в квадратных скобках:

const fruits = ["Яблоко", "Апельсин", "Слива"];
alert( fruits[0] ); // Яблоко
alert( fruits[1] ); // Апельсин
alert( fruits[2] ); // Слива

Элемент можно всегда заменить:

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

…Или добавить:

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

Общее число элементов, хранимых в массиве, содержится в его свойстве length:

constfruits = ["Яблоко", "Апельсин", "Груша"];
alert( fruits.length ); // 3
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Через alert можно вывести и массив целиком.

```
При этом его элементы будут перечислены через запятую:

const fruits = ["Яблоко", "Апельсин", "Груша"];

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

В массиве может храниться любое число элементов любого типа.
В том числе, строки, числа, объекты, вот например:

// микс значений
const arr = [ 1, 'Имя', { name: 'Петя' }, true ];

// получить объект из массива и тут же -- его свойство
alert( arr[2].name ); // Петя
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Методы pop/push, shift/unshift

Одно из применений массива – это очередь. В классическом программировании так называют упорядоченную коллекцию элементов, такую что элементы добавляются в конец, а обрабатываются – с начала. В реальной жизни эта структура данных встречается очень часто. Например, очередь сообщений, которые надо показать на экране. Очень близка к очереди еще одна структура данных: стек. Это такая коллекция элементов, в которой новые элементы добавляются в конец и берутся с конца. Например, стеком является колода карт, в которую новые карты кладутся сверху, и берутся – тоже сверху. Для того, чтобы реализовывать эти структуры данных, и просто для более удобной работы с началом и концом массива существуют специальные методы.

Конец массива

pop

Удаляет последний элемент из массива и возвращает его:

```
const fruits = ["Яблоко", "Апельсин", "Груша"];
alert( fruits.pop() ); // удалили "Груша"
alert( fruits ); // Яблоко, Апельсин
```

push

Добавляет элемент в конец массива:

```
const fruits = ["Яблоко", "Апельсин"];
fruits.push("Груша");
alert( fruits ); // Яблоко, Апельсин, Груша
```

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

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Начало массива

shift

Удаляет из массива первый элемент и возвращает его:

```
const fruits = ["Яблоко", "Апельсин", "Груша"];
alert( fruits.shift() ); // удалили Яблоко
alert( fruits ); // Апельсин, Груша
```

unshift

Добавляет элемент в начало массива:

```
const fruits = ["Апельсин", "Груша"];
fruits.unshift('Яблоко');
alert( fruits ); // Яблоко, Апельсин, Груша
```

Методы push и unshift могут добавлять сразу по несколько элементов:

```
const fruits = ["Яблоко"];
fruits.push("Апельсин", "Персик");
fruits.unshift("Ананас", "Лимон");
// результат: ["Ананас", "Лимон", "Яблоко", "Апельсин", "Персик"]
alert( fruits );
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутреннее устройство массива

Массив – это объект, где в качестве ключей выбраны цифры, с дополнительными методами и свойством length.

Так как это объект, то в функцию он передаётся по ссылке:

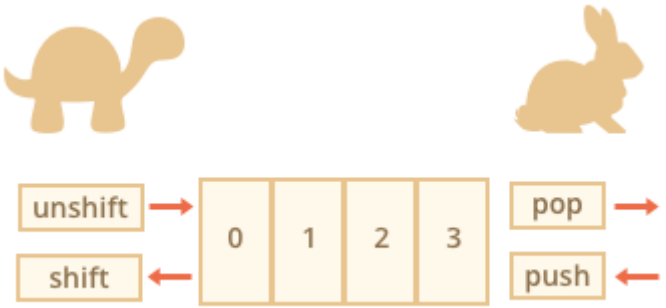
```
function eat(arr) {
  arr.pop();
}

const arr = ["нам", "не", "страшен", "серый", "волк"]
alert( arr.length ); // 5
eat(arr);
eat(arr);
alert( arr.length ); // 3, в функцию массив не скопирован, а передана ссылка

Ещё одно следствие – можно присваивать в массив любые свойства.

Например:
const fruits = []; // создать массив
fruits[99999] = 5; // присвоить свойство с любым номером
fruits.age = 25; // назначить свойство со строковым именем

… Но массивы для того и придуманы в JavaScript, чтобы удобно работать именно с упорядоченными, нумерованными данными.
Для этого в них существуют специальные методы и свойство length.
Как правило, нет причин использовать массив как обычный объект, хотя технически это и возможно.

Вывод массива с «дырами»

Если в массиве есть пропущенные индексы, то при выводе в большинстве браузеров появляются «лишние» запятые, например:
const a = [];
a[0] = 0;
a[5] = 5;
alert( a ); // 0,,,,,5

Эти запятые появляются потому, что алгоритм вывода массива идёт от 0 до arr.length и выводит всё через запятую.
Отсутствие значений даёт несколько запятых подряд.
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

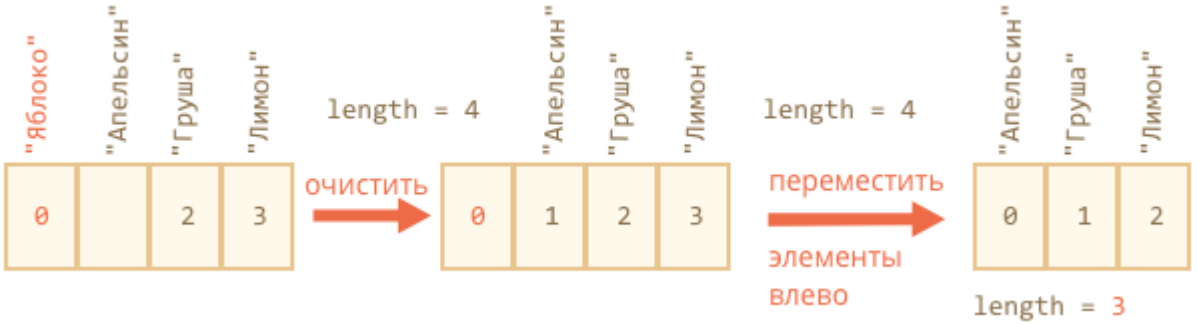
Влияние на быстродействие

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



Чтобы понять, почему работать с концом массива – быстрее, чем с его началом, разберём подробнее происходящее при операции:

```
fruits.shift(); // убрать 1 элемент с начала

При этом, так как все элементы находятся в своих ячейках, просто удалить элемент с номером 0 недостаточно.
Нужно еще и переместить остальные элементы на их новые индексы.

Операция shift должна выполнить целых три действия:
Удалить нулевой элемент.
Переместить все свойства влево, с индекса 1 на 0, с 2 на 1 и так далее.
Обновить свойство length.
```



Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Перебор элементов

Для перебора элементов обычно используется цикл:

```
const arr = ["Яблоко", "Апельсин", "Груша"];
for (let i = 0; i < arr.length; i++) {
  alert( arr[i] );
}
```

Не используйте for..in для массивов
Так как массив является объектом, то возможен и вариант for..in:

```
const arr = ["Яблоко", "Апельсин", "Груша"];
For (let key in arr) {
  alert( arr[key] ); // Яблоко, Апельсин, Груша
}
```

Недостатки этого способа:
Цикл for..in выведет все свойства объекта, а не только цифровые.

В браузере, при работе с объектами страницы, встречаются коллекции элементов, которые по виду как массивы, но имеют дополнительные нецифровые свойства. При переборе таких «похожих на массив» коллекций через for..in эти свойства будут выведены, а они как раз не нужны.

Бывают и библиотеки, которые предоставляют такие коллекции. Классический for надёжно выведет только цифровые свойства, что обычно и требуется.

Цикл for (let i=0; i<arr.length; i++) в современных браузерах выполняется в 10-100 раз быстрее. Казалось бы, по виду он сложнее, но браузер особым образом оптимизирует такие циклы.

Если коротко: цикл for(let i=0; i<arr.length...) надёжнее и быстрее.

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Особенности работы length

Встроенные методы для работы с массивом автоматически обновляют его длину **length**.

Длина **length** – не количество элементов массива, а последний индекс + 1.

Так уж оно устроено.

Это легко увидеть на следующем примере:

```
const arr = [];
arr[1000] = true;
alert(arr.length); // 1001
```

```
Кстати, если у вас элементы массива нумеруются случайно или с большими пропусками, то стоит подумать о том, чтобы
использовать обычный объект. Массивы предназначены именно для работы с непрерывной упорядоченной коллекцией элементов.
```

Используем length для укорачивания массива

Обычно нам не нужно самостоятельно менять **length**… Но есть один фокус, который можно провернуть.

При уменьшении **length** массив укорачивается.

Причем этот процесс необратимый, т.е. даже если потом вернуть **length** обратно – значения не восстановятся:

```
const arr = [1, 2, 3, 4, 5];
arr.length = 2; // укоротить до 2 элементов
alert( arr ); // [1, 2]
arr.length = 5; // вернуть length обратно, как было
alert( arr[3] ); // undefined: значения не вернулись
```

```
Самый простой способ очистить массив – это arr.length=0.
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Создание вызовом new Array

new Array()

Существует еще один синтаксис для создания массива:

```
const arr = new Array("Яблоко", "Груша", "и т.п.");

Он редко используется, т.к. квадратные скобки [] короче.
Кроме того, у него есть одна особенность. Обычно new Array(элементы, ...) создаёт массив из данных элементов, но если у
него один аргумент-число new Array(число), то он создает массив без элементов, но с заданной длиной.
Проверим это:

const arr = new Array(2, 3);
alert( arr[0] ); // 2, создан массив [2, 3], всё ок
arr = new Array(2); // создаст массив [2] ?
alert( arr[0] ); // undefined! у нас массив без элементов, длины 2

Что же такое этот «массив без элементов, но с длиной»? Как такое возможно?
Оказывается, очень даже возможно и соответствует объекту {length: 2}. Получившийся массив ведёт себя так, как будто его
элементы равны undefined.

Это может быть неожиданным сюрпризом, поэтому обычно используют квадратные скобки.
```

Многомерные массивы

Массивы в JavaScript могут содержать в качестве элементов другие массивы. Это можно использовать для создания многомерных массивов, например матриц:

```
const matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
alert( matrix[1][1] ); // центральный элемент
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутреннее представление массивов

Числовые массивы, согласно спецификации, являются объектами, в которые добавили ряд свойств, методов и автоматическую длину length.

Но внутри они, как правило, устроены по-другому.

Современные интерпретаторы стараются оптимизировать их и хранить в памяти не в виде хэш-таблицы, а в виде непрерывной области памяти, по которой легко пробежаться от начала до конца.

Операции с массивами также оптимизируются, особенно если массив хранит только один тип данных, например только числа.

Порождаемый набор инструкций для процессора получается очень эффективным.

Чтобы у интерпретатора получались эти оптимизации, программист не должен мешать.

В частности:

Не ставить массиву произвольные свойства, такие как arr.test = 5. То есть, работать именно как с массивом, а не как с объектом.

Заполнять массив непрерывно и по возрастающей. Как только браузер встречает необычное поведение массива, например устанавливается значение arr[0], а потом сразу arr[1000], то он начинает работать с ним, как с обычным объектом.

Как правило, это влечёт преобразование его в хэш-таблицу.

Если следовать этим принципам, то массивы будут занимать меньше памяти и быстрее работать

По материалам https://learn.javascript.ru/array.

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутренне представление массивов в V8.

Изменение elements kind необратимо при добавлении новых типов элементов в массив:

```
const array = [1, 2, 3];
// elements kind: PACKED_SMI_ELEMENTS
array.push(4.56);
// elements kind: PACKED_DOUBLE_ELEMENTS
array.push('x');
// elements kind: PACKED_ELEMENTS

const array = [1, 2, 3, 4.56, 'x'];
// elements kind: PACKED_ELEMENTS
array.length; // 5
array[9] = 1; // array[5] until array[8] are now holes
// elements kind: HOLEY_ELEMENTS
```

Итого:

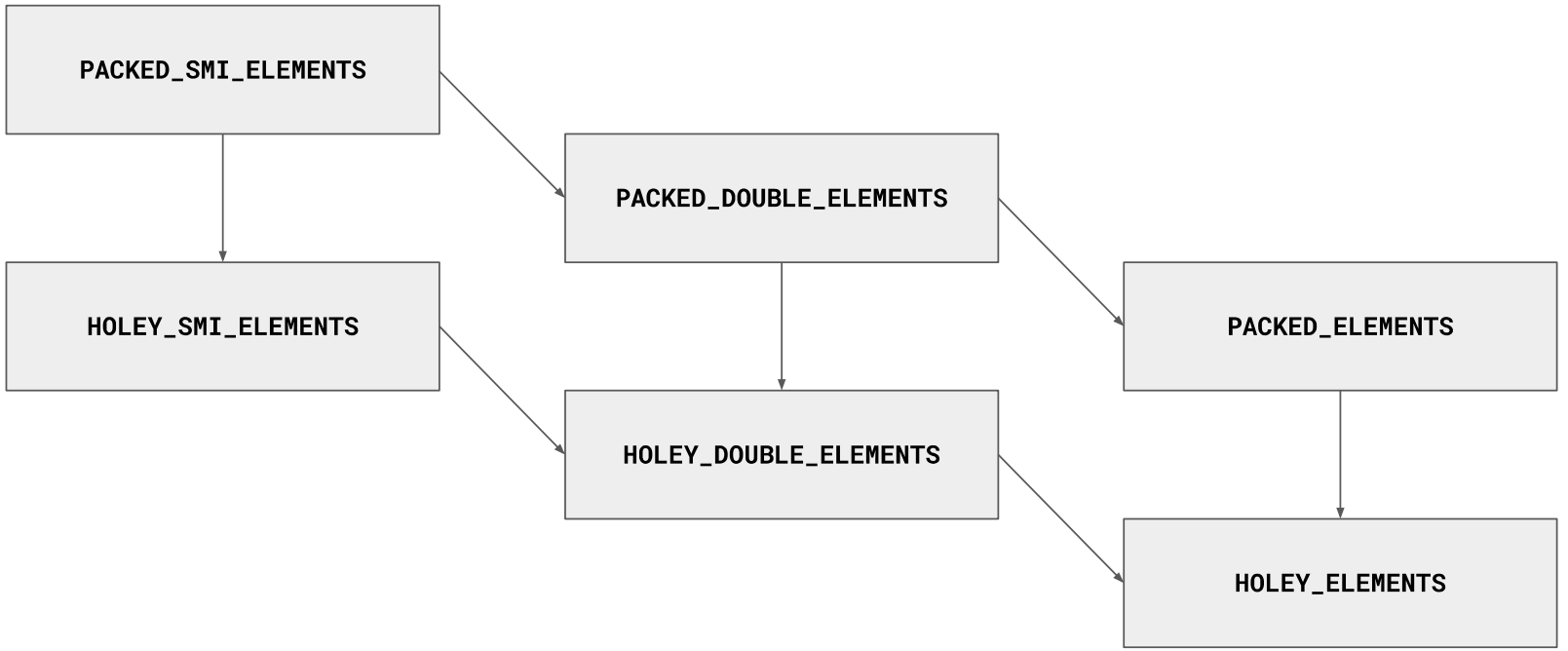
- Самые употребляемые характеристики элементов в массивах имеют разновидности PACKED и HOLEY.
- Операции над упакованными (PACKED) массивами более эффективны чем на «дырявыми» (HOLEY).
- Характеристики элементов в массивах могут переходить из PACKED в HOLEY.

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутренне представление массивов в V8.

Диаграмма переходов характеристик элементов в массивах:



Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутренне представление массивов в V8.

Советы по оптимизации производительности операций с массивами:

Избегайте создания «дыр».

```
const array = new Array(3);
// Массив разреженный в этой точке, поэтому он маркирован как
// `HOLEY_SMI_ELEMENTS`, т. е. Наиболее специфичная возможность на текущий момент
array[0] = 'a';
// Постойте, это строка вместо простого целого числа … Значить характеристика переходит
// в `HOLEY_ELEMENTS`.
array[1] = 'b';
array[2] = 'c';
// В этой точке, все три позиции в массиве заполнены, значит массив упакован (т. е. уже не разреженный)
// Однако, мы не можем перейти к более спецефичной характеристике такой как `PACKED_ELEMENTS`.
// Характеристика элементовостается `HOLEY_ELEMENTS`.
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутренне представление массивов в V8.

Советы по оптимизации производительности операций с массивами:

Избегайте чтения элементов за пределами длины массива.

```
// Не делайте так!
for (let i = 0, item; (item = items[i]) != null; i++) {
  doSomething(item);
}
// Правильно делать так
for (let index = 0; index < items.length; index++) {
  const item = items[index];
  doSomething(item);
}
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутренне представление массивов в V8.

Советы по оптимизации производительности операций с массивами:

Для коллекций (например NodeLists):

```
for (const item of items) {
  doSomething(item);
}
```

Для массивов особенно, можно использовать встроенный **forEach**:

```
items.forEach((item) => {
  doSomething(item);
});
```

В настоящее время производительность **for - of** и **forEach** находятся на одном уровне с устаревшим циклом **for**.

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутренне представление массивов в V8.

Советы по оптимизации производительности операций с массивами:

Избегайте смены характеристик элементов массива.

```
const array = [3, 2, 1, +0];
// PACKED_SMI_ELEMENTS
array.push(-0);
// PACKED_DOUBLE_ELEMENTS

const array = [3, 2, 1];
// PACKED_SMI_ELEMENTS
array.push(NaN, Infinity);
// PACKED_DOUBLE_ELEMENTS
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутренне представление массивов в V8.

Советы по оптимизации производительности операций с массивами:

Предпочитайте использовать массивы нежели массиво-подобные объекты.

```
const arrayLike = {};
arrayLike[0] = 'a';
arrayLike[1] = 'b';
arrayLike[2] = 'c';
arrayLike.length = 3;
Array.prototype.forEach.call(arrayLike, (value, index) => {
  console.log(`${ index }: ${ value }`);
});
// Это выведет '0: a', затем '1: b', и наконец '2: c'.

const actualArray = Array.prototype.slice.call(arrayLike, 0);
actualArray.forEach((value, index) => {
  console.log(`${ index }: ${ value }`);
});
// Это выведет '0: a', затем '1: b', и наконец '2: c'.
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутренне представление массивов в V8.

Советы по оптимизации производительности операций с массивами:

```
const logArgs = function() {
  Array.prototype.forEach.call(arguments, (value, index) => {
    console.log(`${ index }: ${ value }`);
  });
};
logArgs('a', 'b', 'c');
// This logs '0: a', then '1: b', and finally '2: c'.

const logArgs = (...args) => {
  args.forEach((value, index) => {
    console.log(`${ index }: ${ value }`);
  });
};
logArgs('a', 'b', 'c');
// This logs '0: a', then '1: b', and finally '2: c'.
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутренне представление массивов в V8.

Советы по оптимизации производительности операций с массивами:

Избегайте полиморфизма.

```
const each = (array, callback) => {
  for (let index = 0; index < array.length; ++index) {
    const item = array[index];
    callback(item);
  }
};
const doSomething = (item) => console.log(item);

each([], () => {});

each(['a', 'b', 'c'], doSomething);
// `each` вызывается `PACKED_ELEMENTS`. V8 использует inline cache
// (или “IC”) для запоминания что `each` вызывается с определенной характеристикой элементов
// V8 оптимистичен и предполагает что `array.length` и `array[index]` доступы внутри функции `each`
// являются мономорфичными (т. е. получают только один тип элементов) дотех пор пока не будет
// доказано обратное. Для каждого последующего вызова `each`, V8 проверяет является ли характеристика
// элементов `PACKED_ELEMENTS`. Если так, V8 может переиспользовать предыдущий сгенерированный код
// Если нет, то требуется больше работы.
```

Incode Group Workshop #1
Массивы в Javascript

Массивы с числовыми индексами

Внутренне представление массивов в V8.

Советы по оптимизации производительности операций с массивами:

```
each([1.1, 2.2, 3.3], doSomething);
// `each` вызывается с `PACKED_DOUBLE_ELEMENTS`. Поскольку V8 теперь видит различные характеристики
// элементов переданные в `each` в своем IC, `array.length` и `array[index]` доступы внутри функции `each`
// помечаются как полиморфные . V8 теперь требует дополнительной проверки каждый раз,
// когда `each` вызывается: один для `PACKED_ELEMENTS` (как перед этим),
// новую для `PACKED_DOUBLE_ELEMENTS`, и одну для любой другой характеристики элемента
// (как перед этим). Это влечет к проблемам производительности.

each([1, 2, 3], doSomething);
// `each` вызывается с `PACKED_SMI_ELEMENTS`. Это переключает иную степень полиморфизма.
// Теперь есть три различных характеристики элементов в IC для `each`. Для каждого вызова `each`
// с настоящего момента для друхих характеристик элементов требуется переиспользование сгенерированного
// кода для  `PACKED_SMI_ELEMENTS`. Это происходит за счет потерь производительности.
```

По материалам https://v8.dev/blog/elements-kinds

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

Метод split

Ситуация из реальной жизни. Мы пишем сервис отсылки сообщений и посетитель вводит имена тех, кому его отправить: Маша, Петя, Марина, Василий.... Но нам-то гораздо удобнее работать с массивом имен, чем с одной строкой.

К счастью, есть метод split(s), который позволяет превратить строку в массив, разбив ее по разделителю s. В примере ниже таким разделителем является строка из запятой и пробела.

```
const names = 'Маша, Петя, Марина, Василий';
const arr = names.split(', ');
for (let i = 0; i < arr.length; i++) {
  alert( 'Вам сообщение ' + arr[i] );
}
```

```
Второй аргумент split

У метода split есть необязательный второй аргумент – ограничение на количество элементов в массиве. Если их больше, чем указано – остаток массива будет отброшен:

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


Разбивка по буквам

Вызов split с пустой строкой разобьет по буквам:

const str = "тест";
alert( str.split('') ); // т,е,с,т
```

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

Метод join

Вызов arr.join(str) делает в точности противоположное split. Он берет массив и склеивает его в строку, используя str как разделитель.

Например:

```
const arr = ['Маша', 'Петя', 'Марина', 'Василий'];
const str = arr.join(';');
alert( str ); // Маша;Петя;Марина;Василий

new Array + join = Повторение строки

Код для повторения строки 3 раза:
alert( new Array(4).join("ля") ); // ляляля

Как видно, new Array(4) делает массив без элементов длины 4, который join объединяет в строку,
вставляя между его элементами строку "ля".

В результате, так как элементы пусты, получается повторение строки. Такой вот небольшой трюк.
```

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

Удаление из массива

Так как массивы являются объектами, то для удаления ключа можно воспользоваться обычным delete:

```
const arr = ["Я", "иду", "домой"];
delete arr[1]; // значение с индексом 1 удалено
// теперь arr = ["Я", undefined, "домой"];
alert( arr[1] ); // undefined
```

Да, элемент удален из массива, но не так, как нам этого хочется. Образовалась «дырка».

Это потому, что оператор delete удаляет пару «ключ-значение». Это – все, что он делает.

Обычно же при удалении из массива мы хотим, чтобы оставшиеся элементы сдвинулись и заполнили образовавшийся промежуток.

Поэтому для удаления используются специальные методы: из начала – shift, с конца – pop, а из середины – splice, с которым мы сейчас познакомимся.

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

Метод splice

Метод splice – это универсальный раскладной нож для работы с массивами. Умеет все: удалять элементы, вставлять элементы, заменять элементы – по очереди и одновременно.

Его синтаксис:

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

Удалить deleteCount элементов, начиная с номера index, а затем вставить elem1, ..., elemN на их место. Возвращает массив из удалённых элементов.

Этот метод проще всего понять, рассмотрев примеры.

Начнём с удаления:

```
const arr = ["Я", "изучаю", "JavaScript"];
arr.splice(1, 1); // начиная с позиции 1, удалить 1 элемент
alert( arr ); //  осталось ["Я", "JavaScript"]
```

В следующем примере мы удалим 3 элемента и вставим другие на их место:

```
const arr = ["Я", "сейчас", "изучаю", "JavaScript"];
// удалить 3 первых элемента и добавить другие вместо них
arr.splice(0, 3, "Мы", "изучаем")
alert( arr ) // теперь ["Мы", "изучаем", "JavaScript"]
```

Здесь видно, что splice возвращает массив из удаленных элементов:

```
const arr = ["Я", "сейчас", "изучаю", "JavaScript"];
// удалить 2 первых элемента
const removed = arr.splice(0, 2);
alert( removed ); // "Я", "сейчас" <-- array of removed elements
```

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

Метод splice

Метод splice также может вставлять элементы без удаления, для этого достаточно установить deleteCount в 0:

```
const arr = ["Я", "изучаю", "JavaScript"];
// с позиции 2
// удалить 0
// вставить "сложный", "язык"
arr.splice(2, 0, "сложный", "язык");
alert( arr ); // "Я", "изучаю", "сложный", "язык", "JavaScript"

Допускается использование отрицательного номера позиции, которая в этом случае отсчитывается с конца:
const arr = [1, 2, 5]
// начиная с позиции индексом -1 (перед последним элементом)
// удалить 0 элементов,
// затем вставить числа 3 и 4
arr.splice(-1, 0, 3, 4);
alert( arr ); // результат: 1,2,3,4,5
```

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

Метод slice

Метод slice(begin, end) копирует участок массива от begin до end, не включая end. Исходный массив при этом не меняется.

Например:

```
const arr = ["Почему", "надо", "учить", "JavaScript"];
cinst arr2 = arr.slice(1, 3); // элементы 1, 2 (не включая 3)
alert( arr2 ); // надо, учить

Аргументы ведут себя так же, как и в строковом slice:

Если не указать end – копирование будет до конца массива:
const arr = ["Почему", "надо", "учить", "JavaScript"];
alert( arr.slice(1) ); // взять все элементы, начиная с номера 1

Можно использовать отрицательные индексы, они отсчитываются с конца:
const arr2 = arr.slice(-2); // копировать от 2-го элемента с конца и дальше

Если вообще не указать аргументов – скопируется весь массив:
const fullCopy = arr.slice();

Совсем как в строках
Синтаксис метода slice одинаков для строк и для массивов. Тем проще его запомнить.
```

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

Сортировка, метод sort(fn)

Метод sort() сортирует массив на месте. Например:

```
const arr = [ 1, 2, 15 ];
arr.sort();
alert( arr );  // 1, 15, 2
```

```
Не заметили ничего странного в этом примере?
Порядок стал 1, 15, 2, это точно не сортировка чисел. Почему?
Это произошло потому, что по умолчанию sort сортирует, преобразуя элементы к строке.
Поэтому и порядок у них строковый, ведь "2" > "15".
```

Свой порядок сортировки

Для указания своего порядка сортировки в метод arr.sort(fn) нужно передать функцию fn от двух элементов, которая умеет сравнивать их.

Внутренний алгоритм функции сортировки умеет сортировать любые массивы – апельсинов, яблок, пользователей, и тех и других и третьих – чего угодно. Но для этого ему нужно знать, как их сравнивать. Эту роль и выполняет fn.

Если эту функцию не указать, то элементы сортируются как строки.

Например, укажем эту функцию явно, отсортируем элементы массива как числа:

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

const arr = [ 1, 2, 15 ];
arr.sort(compareNumeric);
alert(arr);  // 1, 2, 15
```

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

Сортировка, метод sort(fn)

Обратите внимание, мы передаём в sort() именно саму функцию compareNumeric, без вызова через скобки. Был бы ошибкой следующий код:

```
arr.sort( compareNumeric() );  // не сработает
```

Как видно из примера выше, функция, передаваемая sort, должна иметь два аргумента.
Алгоритм сортировки, встроенный в JavaScript, будет передавать ей для сравнения элементы массива. Она должна возвращать:
Положительное значение, если a > b,
Отрицательное значение, если a < b,
Если равны – можно 0, но вообще – не важно, что возвращать, если их взаимный порядок не имеет значения.

Алгоритм сортировки
В методе sort, внутри самого интерпретатора JavaScript, реализован универсальный алгоритм сортировки. Как правило, это ««быстрая сортировка»», дополнительно оптимизированная для небольших массивов. Он решает, какие пары элементов и когда сравнивать, чтобы отсортировать побыстрее. Мы даём ему функцию – способ сравнения, дальше он вызывает её сам.

Кстати, те значения, с которыми sort вызывает функцию сравнения, можно увидеть, если вставить в неё alert:

```
[1, -2, 15, 2, 0, 8].sort(function(a, b) {
  alert( a + " <> " + b );
});
```

Сравнение compareNumeric в одну строку
Функцию compareNumeric для сравнения элементов-чисел можно упростить до одной строчки.

```
function compareNumeric(a, b) {
  return a - b;
}
```

Эта функция вполне подходит для sort, так как возвращает положительное число, если a > b, отрицательное, если наоборот, и 0, если числа равны.

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

reverse

Метод arr.reverse() меняет порядок элементов в массиве на обратный.

```
var arr = [1, 2, 3];
arr.reverse();
alert( arr ); // 3,2,1
```

concat

Метод arr.concat(value1, value2, … valueN) создаёт новый массив, в который копируются элементы из arr, а также value1, value2, ... valueN.

Например:

```
const arr = [1, 2];
const newArr = arr.concat(3, 4);
alert( newArr ); // 1,2,3,4

У concat есть одна забавная особенность.
Если аргумент concat – массив, то concat добавляет элементы из него.
Например:

const arr = [1, 2];
const newArr = arr.concat([3, 4], 5); // то же самое, что arr.concat(3,4,5)
alert( newArr ); // 1,2,3,4,5
```

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

indexOf/lastIndexOf

Метод «arr.indexOf(searchElement[, fromIndex])» возвращает номер элемента searchElement в массиве arr или -1, если его нет.

Поиск начинается с номера fromIndex, если он указан. Если нет – с начала массива.

Для поиска используется строгое сравнение ===.

Например:

```
const arr = [1, 0, false];
alert( arr.indexOf(0) ); // 1
alert( arr.indexOf(false) ); // 2
alert( arr.indexOf(null) ); // -1
```

Как вы могли заметить, по синтаксису он полностью аналогичен методу `indexOf для строк`.

Метод `«arr.lastIndexOf(searchElement[, fromIndex])»` ищет справа-налево: с конца массива или с номера fromIndex, если он указан.

Методы indexOf/lastIndexOf осуществляют поиск перебором.
Если нужно проверить, существует ли значение в массиве – его нужно перебрать. Только так. Внутренняя реализация indexOf/lastIndexOf осуществляет полный перебор, аналогичный циклу for по массиву. Чем длиннее массив, тем дольше он будет работать.

Incode Group Workshop #1
Массивы в Javascript

Массивы: методы

Object.keys(obj)

Ранее мы говорили о том, что свойства объекта можно перебрать в цикле for..in.

Если мы хотим работать с ними в виде массива, то к нашим услугам – замечательный метод Object.keys(obj)::

```
const user = {
  name: "Петя",
  age: 30
}
var keys = Object.keys(user);
alert( keys ); // name, age
```

Кстати один из способов создать массив элементов со значениями от 0 до N с использование метода **keys**, но только для массивов:

```
const arr=Array.from(new Array(N).keys())
```

По материалам https://learn.javascript.ru/array-methods.