Яндекс

Яндекс

Массивы

Объявление массива

Литерал массива

```
var numbers = [1, 2, 3];
```

Функция-конструктор

```
var numbers = new Array(1, 2, 3);
```

Что такое функция-конструктор и как она работает узнаем на следующем занятии, пока просто запоминаем

Доступ к элементу массива

Независимо от того, как объявлен массив

```
var numbers = [1, 2, 3];
```

```
console.log(numbers[1]); // ??
```

Доступ к элементу массива

В качестве индекса можно передать переменную. В таком случае будет взят индекс с номером, равным значению переменной. Аналогично, можно использовать вообще любые выражения

```
var numbers = [1, 2, 3];
```

```
var n = 1;
```

```
console.log(numbers[n]); // 2
```

Доступ к несуществующему элементу массива

Если запросить индекс, которого нет в массиве, получим undefined

```
var numbers = [1, 2, 3];

console.log(numbers[10]); // undefined
```

Многомерные массивы

Элементами массива может быть буквально что угодно, в том числе - массивы. Таким образом, чтобы добраться до значения потребуется два индекса. Такие массивы называют дву- трех- и т.п. мерными (многомерными).

```
var numbers = [[1, 2, 3], [4, 5, 6]];
```

```
console.log(numbers[1][1]); // ??
```

Во фронтэнде вы едва ли встретитесь более, чем с двумерными массивами (но это не точно)

Длина массива

В массиве есть свойство .length, в котором всегда будет лежать длина массива.

```
var numbers = [1, 2, 3];

console.log(numbers.length); // 3
```

Стоп, что за свойство? Точка же только для объектов! Верно подмечено, разберем как это работает тоже на следующем занятии, а пока будем просто мириться с тем, что оно неожиданно так работает.

Вставьте
изображение

Перебор элементов в массиве

С учетом свойства .length и доступа к элементам с помощью переменных, простой и классический проход по всем элементам массива с помощью цикла for выглядит вот так

```
var numbers = [1, 2, 3];


for (var i = 0; i < numbers.length; i++) {
    console.log(numbers[i]);
}
```

Редактивать элемент в массиве

Элементы массива можно, ничего не стесняясь, перезаписывать.

```
var numbers = [1, 2, 3];
```

```
numbers[1] = 4; // numbers теперь [1, 4, 3]
```

Добавить новый элемент в массив

С учетом наличия свойства length, добавить новый элемент в массив можно вот такой формулой

```
var numbers = [1, 2, 3];

numbers[numbers.length] = 4; // numbers теперь [1, 2, 3, 4]
```

Можно добавлять элементы в массив и не по порядку, но тогда встроенные оптимизации перестанут работать, и станет значительно медленнее

Знакомьтесь - jsperf

Методы для работы с массивом

Все методы для работы с массивом работают через точку к массиву, что должно вызывать удивление на данном этапе - ведь массив - эт о не объект, откуда они взялись?

Как и со свойством .length, сорвем покровы на следующем занятии, а пока просто пользуемся

Добавить новый элемент в массив

Добавить элемент в массив можно простым вызовом .push

```
var numbers = [1, 2, 3];
```

```
numbers.push(4); // numbers теперь [1, 2, 3, 4]
```

Метод возвращает новую длинну массива, то есть 4 в данном случае.
Это, правда, редко используется.

Достать элемент из массива

Достать - значит получить последний элемент массива и удалить его из массива

```
var numbers = [1, 2, 3];
```

```
var num = numbers.pop(); // numbers теперь [1, 2], а num === 3
```

Если массив пустой - метод возвращает null

Вставьте
изображение

Альтернативный перебор элементов в массиве

Метод .pop позволяет перебрать все элементы массива еще одним способом

```
var numbers = [1, 2, 3],
    num;


while (num = numbers.pop()) {
   console.log(num);
}
```

В конце numbers станет пустым массивом []

Работа с началом массива

Достаем/удаляем элементы не с хвоста, а с начала массива

```
var numbers = [1, 2, 3];
```

```
var num = numbers.shift(); // numbers теперь [2, 3], а num === 1
```

Если массив пустой - метод возвращает null

```
var numbers = [1, 2, 3];
```

```
numbers.unshift(0); // numbers теперь [0, 1, 2, 3]
```

Работа с началом массива ощутимо медленнее

Перебирающие методы

forEach

Для каждого элемента массива выполняет функцию-аргумент, передавая ей в качестве аргумента этот самый элемент массива

```
var numbers = [1, 2, 3];


numbers.forEach(function(item) {
    console.log(item); // выведет в консоль 1, 2, потом 3
});
```

Но как?

Откуда появляется item?

Он нигде не объявлен, мы нигде не вызываем функцию, откуда берутся три вызова?

Давайте примерно опишем механизм, скрытый от наших глаз....

Функция, передаваемая как аргумент

Если представить, что мы передаем и массив и функцию как аргументы какой-то еще функции, то все проясняется:

```
var numbers = [1, 2, 3];

function forEach(array, fn) {
    for (var i = 0; i < array.length; i++) {
        fn(array[i]);
    }
}


forEach(numbers, function(item) {
    console.log(item); // выведет в консоль 1, 2, потом 3
});
```

Подробнее и точнее разберем на лекции про объекты (все ведь заметили точку перед forEach?), но в целом именно так и работает тут и в других методах

map

Вовращает массив, элементами которого будут результаты выполнения функции-аргумента для каждого элемента исходного массива (как в forEach)

```
var numbers = [1, 2, 3];


numbers = numbers.map(function(item) {
    console.log(item); // что будет в результате?
});
```

map

Возвращает массив, элементами которого будут результаты выполнения функции-аргумента для каждого элемента исходного массива (как в forEach)

```
var numbers = [1, 2, 3];


numbers = numbers.map(function(item) {
    console.log(item); // что будет в результате?
});


// [undefined, undefined, undefined]
```

Используйте map, если нужно получить такой же по длине массив, но как-то обработать каждый элемент

map

Возвращает массив, элементами которого будут результаты выполнения функции-аргумента для каждого элемента исходного массива (как в forEach)

```
var numbers = [1, 2, 3];


numbers = numbers.map(function(item) {
    return item;
});


// такой же массив, как исходный
```

map

Возвращает массив, элементами которого будут результаты выполнения функции-аргумента для каждого элемента исходного массива (как в forEach)

```
var numbers = [1, 2, 3];


numbers = numbers.map(function(item) {
    return item > 2 ? item : 3;
});


// [3, 3, 3]
```

Не используйте map вместо forEach

reduce

Позволяет сварить из массива что угодно - массив поменьше, объект, строку и так далее

```
var numbers = [1, 2, 3];


numbers.reduce(function(result, item) {
    if (item > 2) result.push(item);


    return result;
}, []);


// [3]
```

some/every

.some вернет true, если хотя бы для одного из вызовов функция-аргумент вернет true.

.every вернет true, только если каждый вызов возвращает true

```
var numbers = [1, 2, 3];


numbers.some(function(item) {
    return item > 2;
}); // true


numbers.every(function(item) {
    return item > 2;
}); // false
```

indexOf/includes

Поисковые методы. indexOf вернет позицию элемента, равного переданному аргументу. Includes возвращает boolean - есть ли такой элемент в массиве, или нет

```
var numbers = [1, 2, 3];


numbers.indexOf(2); 1

numbers.indexOf(4); -1


numbers.includes(2); // true

numbers.includes(4); // false
```

find

Ищет элемент массива, для которого функция-аргумент вернет true и возвращает его значение

```
var numbers = [1, 2, 3];


numbers.find(function(item) {
    return item > 2;
}); // 3
```

Задачи

Сортировка массива

Автор

Поиск максимума

Автор

Поиск максимума

Автор

Яндекс

Логотип партнёра или NDA

Спасибо

Иван Иванов

Менеджер

ivan@yandex-team.ru

@ivan