

Раздел

[Типы данных](#)

Навигация по уроку

Добавление/удаление элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`Большинство методов поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠](#) → [Язык программирования JavaScript](#)
→ [Типы данных](#)

29-го января 2020

Методы массивов

Массивы предоставляют множество методов. Чтобы было проще, в этой главе они разбиты на группы.

Добавление/удаление элементов

Мы уже знаем методы, которые добавляют и удаляют элементы из начала или конца:

- `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.

Есть и другие.

`splice`

Как удалить элемент из массива?

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

```
1 let arr = ["I", "go", "home"];
2
3 delete arr[1]; // удалить "go"
4
5 alert( arr[1] ); // undefined
6
7 // теперь arr = ["I", , "home"];
8 alert( arr.length ); // 3
```

Вроде бы, элемент и был удалён, но при проверке оказывается, что массив всё ещё имеет 3 элемента `arr.length == 3`.

Это нормально, потому что всё, что делает `delete obj.key` – это удаляет значение с данным ключом `key`. Это нормально для объектов, но для массивов мы обычно хотим, чтобы оставшиеся элементы сдвинулись и заняли освободившееся место. Мы ждём, что массив станет короче.

Поэтому для этого нужно использовать специальные методы.

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

Его синтаксис:

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

Он начинает с позиции `index`, удаляет `deleteCount` элементов и вставляет `elem1, ..., elemN` на их место. Возвращает массив из удалённых элементов.

Этот метод проще всего понять, рассмотрев примеры.

Начнём с удаления:

```
1 let arr = ["Я", "изучаю", "JavaScript"];
2
3 arr.splice(1, 1); // начиная с позиции 1, удалить 1 элемент
4
```

```
5
    alert( arr ); // осталось ["Я", "JavaScript"]
```

Легко, правда? Начиная с позиции 1, он убрал 1 элемент.

В следующем примере мы удалим 3 элемента и заменим их двумя другими.

```
1 let arr = ["Я", "изучаю", "JavaScript", "прямо", "сейчас"];
2
3 // удалить 3 первых элемента и заменить их другими
4 arr.splice(0, 3, "Давай", "танцевать");
5
6 alert( arr ); // теперь ["Давай", "танцевать", "прямо",
```

Здесь видно, что `splice` возвращает массив из удалённых элементов:

```
1 let arr = ["Я", "изучаю", "JavaScript", "прямо", "сейчас"];
2
3 // удалить 2 первых элемента
4 let removed = arr.splice(0, 2);
5
6 alert( removed ); // "Я", "изучаю" <-- массив из удалён
```

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

```
1 let arr = ["Я", "изучаю", "JavaScript"];
2
3 // с позиции 2
4 // удалить 0 элементов
5 // вставить "сложный", "язык"
6 arr.splice(2, 0, "сложный", "язык");
7
8 alert( arr ); // "Я", "изучаю", "сложный", "язык", "JavaScript"
```

Отрицательные индексы разрешены

В этом и в других методах массива допускается использование отрицательного индекса. Он позволяет начать отсчёт элементов с конца, как тут:

```
1 let arr = [1, 2, 5];
2
3 // начиная с индекса -1 (перед последним элементом)
4 // удалить 0 элементов,
5 // затем вставить числа 3 и 4
6 arr.splice(-1, 0, 3, 4);
7
8 alert( arr ); // 1,2,3,4,5
```

slice

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

Его синтаксис:

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

Он возвращает новый массив, в который копирует элементы, начиная с индекса `start` и до `end` (не включая `end`). Оба индекса `start` и `end`

Раздел

Типы данных

Навигация по уроку

Добавление/удаление
элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов
поддерживают «thisArg»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



Раздел

Типы данных

Навигация по уроку

Добавление/удаление элементов

Перебор: forEach

Поиск в массиве

Преобразование массива

Array.isArray

Большинство методов поддерживают «thisArg»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



могут быть отрицательными. В таком случае отсчёт будет осуществляться с конца массива.

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

Например:

```
1 let arr = ["t", "e", "s", "t"];
2
3 alert( arr.slice(1, 3) ); // e,s (копирует с 1 до 3)
4
5 alert( arr.slice(-2) ); // s,t (копирует с -2 до конца)
```

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

concat

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

Его синтаксис:

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

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

В результате мы получаем новый массив, включающий в себя элементы из `arr`, а также `arg1`, `arg2` и так далее...

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

Например:

```
1 let arr = [1, 2];
2
3 // создать массив из: arr и [3,4]
4 alert( arr.concat([3, 4]) ); // 1,2,3,4
5
6 // создать массив из: arr и [3,4] и [5,6]
7 alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6
8
9 // создать массив из: arr и [3,4], потом добавить значе
10 alert( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6
```

Обычно он просто копирует элементы из массивов. Другие объекты, даже если они выглядят как массивы, добавляются как есть:

```
1 let arr = [1, 2];
2
3 let arrayLike = {
4   0: "что-то",
5   length: 1
6 };
7
8 alert( arr.concat(arrayLike) ); // 1,2,[object Object]
```

...Но если объект имеет специальное свойство

`Symbol.isConcatSpreadable`, то он обрабатывается `concat` как массив: вместо него добавляются его числовые свойства.

Для корректной обработки в объекте должны быть числовые свойства и `length`:

```
1 let arr = [1, 2];
2
3 let arrayLike = {
4   0: "что-то",
5   1: "ещё",
6   [Symbol.isConcatSpreadable]: true,
7   length: 2
8 };
9
10 alert( arr.concat(arrayLike) ); // 1,2,что-то,ещё
```

Перебор: `forEach`

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

Его синтаксис:

```
1 arr.forEach(function(item, index, array) {
2   // ... делать что-то с item
3 });
```

Например, этот код выведет на экран каждый элемент массива:

```
1 // Вызов alert для каждого элемента
2 ["Bilbo", "Gandalf", "Nazgul"].forEach(alert);
```

А этот вдобавок расскажет и о своей позиции в массиве:

```
1 ["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, ar
2   alert(`${item} имеет позицию ${index} в ${array}`);
3 });
```

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

Поиск в массиве

Далее рассмотрим методы, которые помогут найти что-нибудь в массиве.

`indexOf/lastIndexOf` и `includes`

Методы `arr.indexOf`, `arr.lastIndexOf` и `arr.includes` имеют одинаковый синтаксис и делают по сути то же самое, что и их строковые аналоги, но работают с элементами вместо символов:

- `arr.indexOf(item, from)` ищет `item`, начиная с индекса `from`, и возвращает индекс, на котором был найден искомый элемент, в противном случае `-1`.
- `arr.lastIndexOf(item, from)` – то же самое, но ищет справа налево.
- `arr.includes(item, from)` – ищет `item`, начиная с индекса `from`, и возвращает `true`, если поиск успешен.

Например:

```
1 let arr = [1, 0, false];
2
3 alert( arr.indexOf(0) ); // 1
4 alert( arr.indexOf(false) ); // 2
5 alert( arr.indexOf(null) ); // -1
6
```

Раздел

Типы данных

Навигация по уроку

Добавление/удаление
элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов
поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub

```
alert( arr.includes(1) ); // true
```

Раздел

Типы данных

Навигация по уроку

Добавление/удаление
элементов

Перебор: forEach

Поиск в массиве

Преобразование массива

Array.isArray

Большинство методов
поддерживают «thisArg»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



Обратите внимание, что методы используют строгое сравнение `===`. Таким образом, если мы ищем `false`, он находит именно `false`, а не ноль.

Если мы хотим проверить наличие элемента, и нет необходимости знать его точный индекс, тогда предпочтительным является `arr.includes`.

Кроме того, очень незначительным отличием `includes` является то, что он правильно обрабатывает `NaN` в отличие от `indexOf/lastIndexOf`:

```
1 const arr = [NaN];
2 alert( arr.indexOf(NaN) ); // -1 (должен быть 0, но ===
3 alert( arr.includes(NaN) ); // true (верно)
```

find и findIndex

Представьте, что у нас есть массив объектов. Как нам найти объект с определённым условием?

Здесь пригодится метод `arr.find`.

Его синтаксис таков:

```
1 let result = arr.find(function(item, index, array) {
2   // если true - возвращается текущий элемент и перебор
3   // если все итерации оказались ложными, возвращается
4 });
```

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`.

Например, у нас есть массив пользователей, каждый из которых имеет поля `id` и `name`. Попробуем найти того, кто с `id == 1`:

```
1 let users = [
2   {id: 1, name: "Вася"},
3   {id: 2, name: "Петя"},
4   {id: 3, name: "Маша"}
5 ];
6
7 let user = users.find(item => item.id == 1);
8
9 alert(user.name); // Вася
```

В реальной жизни массивы объектов – обычное дело, поэтому метод `find` крайне полезен.

Обратите внимание, что в данном примере мы передаём `find` функцию `item => item.id == 1`, с одним аргументом. Это типично, дополнительные аргументы этой функции используются редко.

Метод `arr.findIndex` – по сути, то же самое, но возвращает индекс, на котором был найден элемент, а не сам элемент, и `-1`, если ничего не найдено.

filter

Метод `find` ищет один (первый попавшийся) элемент, на котором функция-колбэк вернёт `true`.

Раздел

Типы данных

Навигация по уроку

Добавление/удаление элементов

Перебор: forEach

Поиск в массиве

Преобразование массива

Array.isArray

Большинство методов поддерживают «thisArg»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



На тот случай, если найденных элементов может быть много, предусмотрен метод `arr.filter(fn)`.

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

```
1 let results = arr.filter(function(item, index, array) {
2   // если true - элемент добавляется к результату, и пе
3   // возвращается пустой массив в случае, если ничего н
4 });
```

Например:

```
1 let users = [
2   {id: 1, name: "Вася"},
3   {id: 2, name: "Петя"},
4   {id: 3, name: "Маша"}
5 ];
6
7 // возвращает массив, состоящий из двух первых пользова
8 let someUsers = users.filter(item => item.id < 3);
9
10 alert(someUsers.length); // 2
```

Преобразование массива

Перейдём к методам преобразования и упорядочения массива.

map

Метод `arr.map` является одним из наиболее полезных и часто используемых.

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

Синтаксис:

```
1 let result = arr.map(function(item, index, array) {
2   // возвращается новое значение вместо элемента
3 });
```

Например, здесь мы преобразуем каждый элемент в его длину:

```
1 let lengths = ["Bilbo", "Gandalf", "Nazgul"].map(item =>
2   alert(lengths); // 5,7,6
```

sort(fn)

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

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

Например:

```
1 let arr = [ 1, 2, 15 ];
2
3 // метод сортирует содержимое arr
4 arr.sort();
5
6 alert( arr ); // 1, 15, 2
```

Не заметили ничего странного в этом примере?

Порядок стал 1, 15, 2. Это неправильно! Но почему?

Раздел

[Типы данных](#)

Навигация по уроку

Добавление/удаление элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



По умолчанию элементы сортируются как строки.

Буквально, элементы преобразуются в строки при сравнении. Для строк применяется лексикографический порядок, и действительно выходит, что `"2" > "15"`.

Чтобы использовать наш собственный порядок сортировки, нам нужно предоставить функцию в качестве аргумента `arr.sort()`.

Функция должна для пары значений возвращать:

```
1 function compare(a, b) {
2   if (a > b) return 1; // если первое значение больше в
3   if (a == b) return 0; // если равны
4   if (a < b) return -1; // если первое значение меньше
5 }
```

Например, для сортировки чисел:

```
1 function compareNumeric(a, b) {
2   if (a > b) return 1;
3   if (a == b) return 0;
4   if (a < b) return -1;
5 }
6
7 let arr = [ 1, 2, 15 ];
8
9 arr.sort(compareNumeric);
10
11 alert(arr); // 1, 2, 15
```



Теперь всё работает как надо.



Давайте возьмём паузу и подумаем, что же происходит. Упомянутый ранее массив `arr` может быть массивом чего угодно, верно? Он может содержать числа, строки, объекты или что-то ещё. У нас есть набор *каких-то элементов*. Чтобы отсортировать его, нам нужна *функция, определяющая порядок*, которая знает, как сравнивать его элементы. По умолчанию элементы сортируются как строки.

Метод `arr.sort(fn)` реализует общий алгоритм сортировки. Нам не нужно заботиться о том, как он работает внутри (в большинстве случаев это оптимизированная [быстрая сортировка](#)). Она проходит по массиву, сравнивает его элементы с помощью предоставленной функции и переупорядочивает их. Всё, что остаётся нам, это предоставить `fn`, которая делает это сравнение.

Кстати, если мы когда-нибудь захотим узнать, какие элементы сравниваются – ничто не мешает нам вывести их на экран:

```
1 [1, -2, 15, 2, 0, 8].sort(function(a, b) {
2   alert( a + " <> " + b );
3 });
```



В процессе работы алгоритм может сравнивать элемент с другими по несколько раз, но он старается сделать как можно меньше сравнений.



Раздел

Типы данных

Навигация по уроку

Добавление/удаление элементов

Перебор: forEach

Поиск в массиве

Преобразование массива

Array.isArray

Большинство методов поддерживают «thisArg»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



i Функция сравнения может вернуть любое число

На самом деле от функции сравнения требуется любое положительное число, чтобы сказать «больше», и отрицательное число, чтобы сказать «меньше».

Это позволяет писать более короткие функции:

```
1 let arr = [ 1, 2, 15 ];
2
3 arr.sort(function(a, b) { return a - b; });
4
5 alert(arr); // 1, 2, 15
```



i Лучше использовать стрелочные функции

Помните [стрелочные функции](#)? Можно использовать их здесь для того, чтобы сортировка выглядела более аккуратной:

```
1 arr.sort( (a, b) => a - b );
```

Будет работать точно так же, как и более длинная версия выше.

reverse

Метод [arr.reverse](#) меняет порядок элементов в `arr` на обратный.

Например:

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



Он также возвращает массив `arr` с изменённым порядком элементов.

split и join

Ситуация из реальной жизни. Мы пишем приложение для обмена сообщениями, и посетитель вводит имена тех, кому его отправить, через запятую: Вася, Петя, Маша. Но нам-то гораздо удобнее работать с массивом имён, чем с одной строкой. Как его получить?

Метод [str.split\(delim\)](#) именно это и делает. Он разбивает строку на массив по заданному разделителю `delim`.

В примере ниже таким разделителем является строка из запятой и пробела.

```
1 let names = 'Вася, Петя, Маша';
2
3 let arr = names.split(', ');
4
5 for (let name of arr) {
6   alert( `Сообщение получат: ${name}.` ); // Сообщение
7 }
```



У метода `split` есть необязательный второй числовой аргумент – ограничение на количество элементов в массиве. Если их больше, чем указано, то остаток массива будет отброшен. На практике это редко используется:

Раздел

Типы данных

Навигация по уроку

Добавление/удаление
элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов
поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



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

Разбивка по буквам

Вызов `split(s)` с пустым аргументом `s` разбил бы строку на массив букв:

```
1 let str = "тест";
2
3 alert( str.split('') ); // т,е,с,т
```

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

Например:

```
1 let arr = ['Вася', 'Петя', 'Маша'];
2
3 let str = arr.join(';'); // объединить массив в строку
4
5 alert( str ); // Вася;Петя;Маша
```

reduce/reduceRight

Если нам нужно перебрать массив – мы можем использовать `forEach`, `for` или `for..of`.

Если нам нужно перебрать массив и вернуть данные для каждого элемента – мы используем `map`.

Методы `arr.reduce` и `arr.reduceRight` похожи на методы выше, но они немного сложнее. Они используются для вычисления какого-нибудь единого значения на основе всего массива.

Синтаксис:

```
1 let value = arr.reduce(function(previousValue, item, index) {
2   // ...
3 }, [initial]);
```

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

Аргументы:

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

При вызове функции результат её вызова на предыдущем элементе массива передаётся как первый аргумент.

Звучит сложно, но всё становится проще, если думать о первом аргументе как «аккумулирующем» результат предыдущих вызовов функции. По окончании он становится результатом `reduce`.

Этот метод проще всего понять на примере.

Тут мы получим сумму всех элементов массива всего одной строкой:

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

Раздел

Типы данных

Навигация по уроку

Добавление/удаление элементов

Перебор: forEach

Поиск в массиве

Преобразование массива

Array.isArray

Большинство методов поддерживают «thisArg»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



```
3 let result = arr.reduce((sum, current) => sum + current
4
5 alert(result); // 15
```

Здесь мы использовали наиболее распространённый вариант `reduce`, который использует только 2 аргумента.

Давайте детальнее разберём, как он работает.

1. 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.
2. При втором запуске `sum` = 1, и к нему мы добавляем второй элемент массива (2).
3. При третьем запуске `sum` = 3, к которому мы добавляем следующий элемент, и так далее...

Поток вычислений получается такой:

sum 0	sum 0+1	sum 0+1+2	sum 0+1+2+3	sum 0+1+2+3+4
current 1	current 2	current 3	current 4	current 5
1	2	3	4	5

→ 0+1+2+3+4+5 = 15

В виде таблицы, где каждая строка – вызов функции на очередном элементе массива:

	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

Здесь отчётливо видно, как результат предыдущего вызова передаётся в первый аргумент следующего.

Мы также можем опустить начальное значение:

```
1 let arr = [1, 2, 3, 4, 5];
2
3 // убрано начальное значение (нет 0 в конце)
4 let result = arr.reduce((sum, current) => sum + current
5
6 alert( result ); // 15
```

Результат – точно такой же! Это потому, что при отсутствии `initial` в качестве первого значения берётся первый элемент массива, а перебор стартует со второго.

Таблица вычислений будет такая же за вычетом первой строки.

Но такое использование требует крайней осторожности. Если массив пуст, то вызов `reduce` без начального значения выдаст ошибку.

Вот пример:

```
1 let arr = [];
2
3 // Error: Reduce of empty array with no initial value
4 // если бы существовало начальное значение, reduce верн
5 arr.reduce((sum, current) => sum + current);
```

Поэтому рекомендуется всегда указывать начальное значение.

Раздел

Типы данных

Навигация по уроку

Добавление/удаление
элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов
поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



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

Array.isArray

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

Поэтому `typeof` не может отличить простой объект от массива:

```
1 alert(typeof {}); // object
2 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.

```
1 alert(Array.isArray({})); // false
2
3 alert(Array.isArray([])); // true
```

Большинство методов поддерживают «thisArg»

Почти все методы массива, которые вызывают функции – такие как `find`, `filter`, `map`, за исключением метода `sort`, принимают необязательный параметр `thisArg`.

Этот параметр не объяснялся выше, так как очень редко используется, но для наиболее полного понимания темы мы обязаны его рассмотреть.

Вот полный синтаксис этих методов:

```
1 arr.find(func, thisArg);
2 arr.filter(func, thisArg);
3 arr.map(func, thisArg);
4 // ...
5 // thisArg - это необязательный последний аргумент
```

Значение параметра `thisArg` становится `this` для `func`.

Например, вот тут мы используем метод объекта `army` как фильтр, и `thisArg` передаёт ему контекст:

```
1 let army = {
2   minAge: 18,
3   maxAge: 27,
4   canJoin(user) {
5     return user.age >= this.minAge && user.age < this.m
6   }
7 };
8
9 let users = [
10  {age: 16},
11  {age: 20},
12  {age: 23},
13  {age: 30}
14 ];
15
16 // найти пользователей, для которых army.canJoin возвра
17 let soldiers = users.filter(army.canJoin, army);
18
19 alert(soldiers.length); // 2
20 alert(soldiers[0].age); // 20
21 alert(soldiers[1].age); // 23
```

Если бы мы в примере выше использовали просто `users.filter(army.canJoin)`, то вызов `army.canJoin` был бы в

Раздел

[Типы данных](#)

Навигация по уроку

Добавление/удаление
элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов
поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



режиме отдельной функции, с `this=undefined`. Это тут же привело бы к ошибке.

Вызов `users.filter(array.canJoin, array)` можно заменить на `users.filter(user => array.canJoin(user))`, который делает то же самое. Последняя запись используется даже чаще, так как функция-стрелка более наглядна.

Итого

Шпаргалка по методам массива:

- Для добавления/удаления элементов:
 - `push(...items)` – добавляет элементы в конец,
 - `pop()` – извлекает элемент с конца,
 - `shift()` – извлекает элемент с начала,
 - `unshift(...items)` – добавляет элементы в начало.
 - `splice(pos, deleteCount, ...items)` – начиная с индекса `pos`, удаляет `deleteCount` элементов и вставляет `items`.
 - `slice(start, end)` – создаёт новый массив, копируя в него элементы с позиции `start` до `end` (не включая `end`).
 - `concat(...items)` – возвращает новый массив: копирует все члены текущего массива и добавляет к нему `items`. Если какой-то из `items` является массивом, тогда берутся его элементы.
- Для поиска среди элементов:
 - `indexOf/lastIndexOf(item, pos)` – ищет `item`, начиная с позиции `pos`, и возвращает его индекс или `-1`, если ничего не найдено.
 - `includes(value)` – возвращает `true`, если в массиве имеется элемент `value`, в противном случае `false`.
 - `find/filter(func)` – фильтрует элементы через функцию и отдаёт первое/все значения, при прохождении которых через функцию возвращается `true`.
 - `findIndex` похож на `find`, но возвращает индекс вместо значения.
- Для перебора элементов:
 - `forEach(func)` – вызывает `func` для каждого элемента. Ничего не возвращает.
- Для преобразования массива:
 - `map(func)` – создаёт новый массив из результатов вызова `func` для каждого элемента.
 - `sort(func)` – сортирует массив «на месте», а потом возвращает его.
 - `reverse()` – «на месте» меняет порядок следования элементов на противоположный и возвращает изменённый массив.
 - `split/join` – преобразует строку в массив и обратно.
 - `reduce(func, initial)` – вычисляет одно значение на основе всего массива, вызывая `func` для каждого элемента и передавая промежуточный результат между вызовами.
- Дополнительно:
 - `Array.isArray(arr)` проверяет, является ли `arr` массивом.

Обратите внимание, что методы `sort`, `reverse` и `splice` изменяют исходный массив.

Изученных нами методов достаточно в 99% случаев, но существуют и другие.

- `arr.some(fn)/arr.every(fn)` проверяет массив.

Функция `fn` вызывается для каждого элемента массива аналогично `map`. Если какие-либо/все результаты вызовов являются `true`, то метод возвращает `true`, иначе `false`.

- `arr.fill(value, start, end)` – заполняет массив повторяющимися `value`, начиная с индекса `start` до `end`.
- `arr.copyWithin(target, start, end)` – копирует свои элементы, начиная со `start` и заканчивая `end`, в собственную позицию `target`

Раздел

[Типы данных](#)

Навигация по уроку

Добавление/удаление элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub

(перезаписывает существующие).

Полный список есть в [справочнике MDN](#).

На первый взгляд может показаться, что существует очень много разных методов, которые довольно сложно запомнить. Но это гораздо проще, чем кажется.

Внимательно изучите шпаргалку, представленную выше, а затем, чтобы попрактиковаться, решите задачи, предложенные в данной главе. Так вы получите необходимый опыт в правильном использовании методов массива.

Всякий раз, когда вам будет необходимо что-то сделать с массивом, а вы не знаете, как это сделать – приходите сюда, смотрите на таблицу и ищите правильный метод. Примеры помогут вам всё сделать правильно, и вскоре вы быстро запомните методы без особых усилий.

✓ Задачи

Переведите текст вида `border-left-width` в `borderLeftWidth`

важность: 5

Напишите функцию `camelize(str)`, которая преобразует строки вида «`my-short-string`» в «`myShortString`».

То есть дефисы удаляются, а все слова после них получают заглавную букву.

Примеры:

```
1 camelize("background-color") == 'backgroundColor';
2 camelize("list-style-image") == 'listStyleImage';
3 camelize("-webkit-transition") == 'WebkitTransition';
```

P.S. Подсказка: используйте `split`, чтобы разбить строку на массив символов, потом переделайте всё как нужно и методом `join` соедините обратно.

[Открыть песочницу с тестами для задачи.](#)

решение

Фильтрация по диапазону

важность: 4

Напишите функцию `filterRange(arr, a, b)`, которая принимает массив `arr`, ищет в нём элементы между `a` и `b` и отдаёт массив этих элементов.

Функция должна возвращать новый массив и не изменять исходный.

Например:

```
1 let arr = [5, 3, 8, 1];
2
3 let filtered = filterRange(arr, 1, 4);
4
5 alert( filtered ); // 3,1 (совпадающие значения)
6
7 alert( arr ); // 5,3,8,1 (без изменений)
```

[Открыть песочницу с тестами для задачи.](#)

решение

Фильтрация по диапазону "на месте"

Раздел

[Типы данных](#)

Навигация по уроку

Добавление/удаление
элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов
поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub

важность: 4

Напишите функцию `filterRangeInPlace(arr, a, b)`, которая принимает массив `arr` и удаляет из него все значения кроме тех, которые находятся между `a` и `b`. То есть, проверка имеет вид $a \leq arr[i] \leq b$.

Функция должна изменять принимаемый массив и ничего не возвращать.

Например:

```
1 let arr = [5, 3, 8, 1];
2
3 filterRangeInPlace(arr, 1, 4); // удалены числа вне диа
4
5 alert( arr ); // [3, 1]
```

[Открыть песочницу с тестами для задачи.](#)

решение

Сортировать в порядке по убыванию

важность: 4

```
1 let arr = [5, 2, 1, -10, 8];
2
3 // ... ваш код для сортировки по убыванию
4
5 alert( arr ); // 8, 5, 2, 1, -10
```

решение

Скопировать и отсортировать массив

важность: 5

У нас есть массив строк `arr`. Нужно получить отсортированную копию, но оставить `arr` неизменённым.

Создайте функцию `copySorted(arr)`, которая будет возвращать такую копию.

```
1 let arr = ["HTML", "JavaScript", "CSS"];
2
3 let sorted = copySorted(arr);
4
5 alert( sorted ); // CSS, HTML, JavaScript
6 alert( arr ); // HTML, JavaScript, CSS (без изменений)
```

решение

Создать расширяемый калькулятор

важность: 5

Создайте функцию конструктор `Calculator`, которая создаёт «расширяемые» объекты калькулятора.

Задание состоит из двух частей.

1.

Во-первых, реализуйте метод `calculate(str)`, который принимает строку типа `"1 + 2"` в формате «ЧИСЛО оператор ЧИСЛО» (разделено пробелами) и возвращает результат. Метод должен понимать плюс `+` и минус `-`.

Раздел

Типы данных

Навигация по уроку

Добавление/удаление
элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов
поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



Пример использования:

```
1 let calc = new Calculator;
2
3 alert( calc.calculate("3 + 7") ); // 10
```

2.

Затем добавьте метод `addMethod(name, func)`, который добавляет в калькулятор новые операции. Он принимает оператор `name` и функцию с двумя аргументами `func(a, b)`, которая описывает его.

Например, давайте добавим умножение `*`, деление `/` и возведение в степень `**`:

```
1 let powerCalc = new Calculator;
2 powerCalc.addMethod("*", (a, b) => a * b);
3 powerCalc.addMethod("/", (a, b) => a / b);
4 powerCalc.addMethod("**", (a, b) => a ** b);
5
6 let result = powerCalc.calculate("2 ** 3");
7 alert( result ); // 8
```

- Для этой задачи не нужны скобки или сложные выражения.
- Числа и оператор разделены ровно одним пробелом.
- Не лишним будет добавить обработку ошибок.

[Открыть песочницу с тестами для задачи.](#)

решение



Трансформировать в массив имён

важность: 5

У вас есть массив объектов `user`, и в каждом из них есть `user.name`. Напишите код, который преобразует их в массив имён.

Например:

```
1 let vasya = { name: "Вася", age: 25 };
2 let petya = { name: "Петя", age: 30 };
3 let masha = { name: "Маша", age: 28 };
4
5 let users = [ vasya, petya, masha ];
6
7 let names = /* ... ваш код */
8
9 alert( names ); // Вася, Петя, Маша
```

решение



Трансформировать в объекты

важность: 5

У вас есть массив объектов `user`, и у каждого из объектов есть `name`, `surname` и `id`.

Напишите код, который создаст ещё один массив объектов с параметрами `id` и `fullName`, где `fullName` – состоит из `name` и `surname`.

Например:

```
1 let vasya = { name: "Вася", surname: "Пупкин", id: 1 };
2 let petya = { name: "Петя", surname: "Иванов", id: 2 };
3 let masha = { name: "Маша", surname: "Петрова", id: 3 }
```

Раздел

[Типы данных](#)

Навигация по уроку

Добавление/удаление
элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов
поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



```
4
5 let users = [ vasya, petya, masha ];
6
7 let usersMapped = /* ... ваш код ... */
8
9 /*
10 usersMapped = [
11   { fullName: "Вася Пупкин", id: 1 },
12   { fullName: "Петя Иванов", id: 2 },
13   { fullName: "Маша Петрова", id: 3 }
14 ]
15 */
16
17 alert( usersMapped[0].id ) // 1
18 alert( usersMapped[0].fullName ) // Вася Пупкин
```

Итак, на самом деле вам нужно трансформировать один массив объектов в другой. Попробуйте использовать `=>`. Это небольшая уловка.

решение

Отсортировать пользователей по возрасту

важность: 5

Напишите функцию `sortByAge(users)`, которая принимает массив объектов со свойством `age` и сортирует их по нему.

Например:

```
1 let vasya = { name: "Вася", age: 25 };
2 let petya = { name: "Петя", age: 30 };
3 let masha = { name: "Маша", age: 28 };
4
5 let arr = [ vasya, petya, masha ];
6
7 sortByAge(arr);
8
9 // теперь: [vasya, masha, petya]
10 alert(arr[0].name); // Вася
11 alert(arr[1].name); // Маша
12 alert(arr[2].name); // Петя
```

решение

Перемешайте массив

важность: 3

Напишите функцию `shuffle(array)`, которая перемешивает (перепорядочивает случайным образом) элементы массива.

Многочисленные прогоны через `shuffle` могут привести к разным последовательностям элементов. Например:

```
1 let arr = [1, 2, 3];
2
3 shuffle(arr);
4 // arr = [3, 2, 1]
5
6 shuffle(arr);
7 // arr = [2, 1, 3]
8
9 shuffle(arr);
10 // arr = [3, 1, 2]
11 // ...
```


Раздел

Типы данных

Навигация по уроку

Добавление/удаление
элементов

Перебор: `forEach`

Поиск в массиве

Преобразование массива

`Array.isArray`

Большинство методов
поддерживают «`thisArg`»

Итого

Задачи (12)

Комментарии

Поделиться



Редактировать на GitHub



Все последовательности элементов должны иметь одинаковую вероятность. Например, `[1, 2, 3]` может быть перепорядочено как `[1, 2, 3]` или `[1, 3, 2]`, или `[3, 1, 2]` и т.д., с равной вероятностью каждого случая.

решение

Получить средний возраст

важность: 4

Напишите функцию `getAverageAge(users)`, которая принимает массив объектов со свойством `age` и возвращает средний возраст.

Формула вычисления среднего арифметического значения: $(age1 + age2 + \dots + ageN) / N$.

Например:

```
1 let vasya = { name: "Вася", age: 25 };
2 let petya = { name: "Петя", age: 30 };
3 let masha = { name: "Маша", age: 29 };
4
5 let arr = [ vasya, petya, masha ];
6
7 alert( getAverageAge(arr) ); // (25 + 30 + 29) / 3 = 28
```

решение

Оставить уникальные элементы массива

важность: 4

Пусть `arr` – массив строк.

Напишите функцию `unique(arr)`, которая возвращает массив, содержащий только уникальные элементы `arr`.

Например:

```
1 function unique(arr) {
2   /* ваш код */
3 }
4
5 let strings = ["кришна", "кришна", "хапе", "хапе",
6   "хапе", "хапе", "кришна", "кришна", "-0"
7 ];
8
9 alert( unique(strings) ); // кришна, хапе, -0
```

[Открыть песочницу с тестами для задачи.](#)

решение

Проводим [курсы по JavaScript и фреймворкам](#).

Комментарии

перед тем как писать...

