

[🏠](#) → [Язык JavaScript](#) → [Типы данных](#)

 25 января 2023 г.

Map и Set

Сейчас мы знаем о следующих сложных структурах данных:

- Объекты для хранения именованных коллекций.
- Массивы для хранения упорядоченных коллекций.

Но этого не всегда достаточно для решения повседневных задач. Поэтому также существуют `Map` и `Set`.

Map

`Map` – это коллекция ключ/значение, как и `Object`. Но основное отличие в том, что `Map` позволяет использовать ключи любого типа.

Методы и свойства:

- `new Map()` – создаёт коллекцию.
- `map.set(key, value)` – записывает по ключу `key` значение `value`.
- `map.get(key)` – возвращает значение по ключу или `undefined`, если ключ `key` отсутствует.
- `map.has(key)` – возвращает `true`, если ключ `key` присутствует в коллекции, иначе `false`.
- `map.delete(key)` – удаляет элемент (пару «ключ/значение») по ключу `key`.
- `map.clear()` – очищает коллекцию от всех элементов.
- `map.size` – возвращает текущее количество элементов.

Например:



```

1  let map = new Map();
2
3  map.set("1", "str1");    // строка в качестве ключа
4  map.set(1, "num1");     // цифра как ключ
5  map.set(true, "bool1"); // булево значение как ключ
6
7  // помните, обычный объект Object приводит ключи к строкам?
8  // Map сохраняет тип ключей, так что в этом случае сохранится 2 разных значения
9  alert(map.get(1)); // "num1"
10 alert(map.get("1")); // "str1"
11
12 alert(map.size); // 3

```

Как мы видим, в отличие от объектов, ключи не были приведены к строкам. Можно использовать любые типы данных для ключей.

i `map[key]` это не совсем правильный способ использования `Map`

Хотя `map[key]` также работает, например, мы можем установить `map[key] = 2`, в этом случае `map` рассматривался бы как обычный JavaScript объект, таким образом это ведёт ко всем соответствующим ограничениям (только строки/символьные ключи и так далее).

Поэтому нам следует использовать методы `map : set`, `get` и так далее.

Map может использовать объекты в качестве ключей.

Например:

```
1 let john = { name: "John" };
2
3 // давайте сохраним количество посещений для каждого пользователя
4 let visitsCountMap = new Map();
5
6 // объект john - это ключ для значения в объекте Map
7 visitsCountMap.set(john, 123);
8
9 alert(visitsCountMap.get(john)); // 123
```



Использование объектов в качестве ключей – одна из наиболее заметных и важных функций `Map`. Это то что невозможно для `Object`. Строка в качестве ключа в `Object` – это нормально, но мы не можем использовать другой `Object` в качестве ключа в `Object`.

Давайте попробуем заменить `Map` на `Object`:

```
1 let john = { name: "John" };
2 let ben = { name: "Ben" };
3
4 let visitsCountObj = {}; // попробуем использовать объект
5
6 visitsCountObj[ben] = 234; // пробуем использовать объект ben в качестве ключа
7 visitsCountObj[john] = 123; // пробуем использовать объект john в качестве кля
8
9 // Вот что там было записано!
10 alert( visitsCountObj["[object Object]"] ); // 123
```



Так как `visitsCountObj` является объектом, он преобразует все ключи `Object`, такие как `john` и `ben`, в одну и ту же строку `"[object Object]"`. Это определено не то, чего мы хотим.

i Как объект `Map` сравнивает ключи

Чтобы сравнивать ключи, объект `Map` использует алгоритм `SameValueZero`. Это почти такое же сравнение, что и `===`, с той лишь разницей, что `NaN` считается равным `NaN`. Так что `NaN` также может использоваться в качестве ключа.

Этот алгоритм не может быть заменён или модифицирован.

i Цепочка вызовов

Каждый вызов `map.set` возвращает объект `map`, так что мы можем объединить вызовы в цепочку:

```
1 map.set("1", "str1")
2   .set(1, "num1")
3   .set(true, "bool1");
```

Перебор Map

Для перебора коллекции `Map` есть 3 метода:

- `map.keys()` – возвращает итерируемый объект по ключам,
- `map.values()` – возвращает итерируемый объект по значениям,
- `map.entries()` – возвращает итерируемый объект по парам вида `[ключ, значение]`, этот вариант используется по умолчанию в `for..of`.

Например:

```
1 let recipeMap = new Map([
2   ["огурец", 500],
3   ["помидор", 350],
4   ["лук", 50]
5 ]);
6
7 // перебор по ключам (овощи)
8 for (let vegetable of recipeMap.keys()) {
9   alert(vegetable); // огурец, помидор, лук
10 }
11
12 // перебор по значениям (числа)
13 for (let amount of recipeMap.values()) {
14   alert(amount); // 500, 350, 50
15 }
16
17 // перебор по элементам в формате [ключ, значение]
18 for (let entry of recipeMap) { // то же самое, что и recipeMap.entries()
19   alert(entry); // огурец,500 (и так далее)
20 }
```

i Используется порядок вставки

В отличие от обычных объектов `Object`, в `Map` перебор происходит в том же порядке, в каком происходило добавление элементов.

Кроме этого, `Map` имеет встроенный метод `forEach`, схожий со встроенным методом массивов `Array`:

```
1 // выполняем функцию для каждой пары (ключ, значение)
2 recipeMap.forEach((value, key, map) => {
```

```
3   alert(`${key}: ${value}`); // огурец: 500 и так далее
4   });
```

Object.entries: Map из Object

При создании **Map** мы можем указать массив (или другой итерируемый объект) с парами ключ-значение для инициализации, как здесь:

```
1 // массив пар [ключ, значение]
2 let map = new Map([
3   ['1', 'str1'],
4   [1, 'num1'],
5   [true, 'bool1']
6 ]);
7
8 alert( map.get('1') ); // str1
```

Если у нас уже есть обычный объект, и мы хотели бы создать **Map** из него, то поможет встроенный метод **Object.entries(obj)**, который получает объект и возвращает массив пар ключ-значение для него, как раз в этом формате.

Так что мы можем создать **Map** из обычного объекта следующим образом:

```
1 let obj = {
2   name: "John",
3   age: 30
4 };
5
6 let map = new Map(Object.entries(obj));
7
8 alert( map.get('name') ); // John
```

Здесь **Object.entries** возвращает массив пар ключ-значение: [["name", "John"], ["age", 30]]. Это именно то, что нужно для создания **Map**.

Object.fromEntries: Object из Map

Мы только что видели, как создать **Map** из обычного объекта при помощи **Object.entries(obj)**.

Есть метод **Object.fromEntries**, который делает противоположное: получив массив пар вида [ключ, значение], он создаёт из них объект:

```
1 let prices = Object.fromEntries([
2   ['banana', 1],
3   ['orange', 2],
4   ['meat', 4]
5 ]);
6
7 // prices = { banana: 1, orange: 2, meat: 4 }
8
9
```

```
alert(prices.orange); // 2
```

Мы можем использовать `Object.fromEntries`, чтобы получить обычный объект из `Map`.

К примеру, у нас данные в `Map`, но их нужно передать в сторонний код, который ожидает обычный объект.

Вот как это сделать:



```
1 let map = new Map();
2 map.set('banana', 1);
3 map.set('orange', 2);
4 map.set('meat', 4);
5
6 let obj = Object.fromEntries(map.entries()); // создаём обычный объект (*)
7
8 // готово!
9 // obj = { banana: 1, orange: 2, meat: 4 }
10
11 alert(obj.orange); // 2
```

Вызов `map.entries()` возвращает итерируемый объект пар ключ/значение, как раз в нужном формате для `Object.fromEntries`.

Мы могли бы написать строку (*) ещё короче:

```
1 let obj = Object.fromEntries(map); // убрать .entries()
```

Это то же самое, так как `Object.fromEntries` ожидает перебираемый объект в качестве аргумента, не обязательно массив. А перебор `map` как раз возвращает пары ключ/значение, так же, как и `map.entries()`. Так что в итоге у нас будет обычный объект с теми же ключами/значениями, что и в `map`.

Set

Объект `Set` – это особый вид коллекции: «множество» значений (без ключей), где каждое значение может появляться только один раз.

Его основные методы это:

- `new Set(iterable)` – создаёт `Set`, и если в качестве аргумента был предоставлен итерируемый объект (обычно это массив), то копирует его значения в новый `Set`.
- `set.add(value)` – добавляет значение (если оно уже есть, то ничего не делает), возвращает тот же объект `set`.
- `set.delete(value)` – удаляет значение, возвращает `true`, если `value` было в множестве на момент вызова, иначе `false`.
- `set.has(value)` – возвращает `true`, если значение присутствует в множестве, иначе `false`.
- `set.clear()` – удаляет все имеющиеся значения.
- `set.size` – возвращает количество элементов в множестве.

Основная «изюминка» – это то, что при повторных вызовах `set.add()` с одним и тем же значением ничего не происходит, за счёт этого как раз и получается, что каждое значение появляется один раз.

Например, мы ожидаем посетителей, и нам необходимо составить их список. Но повторные визиты не должны приводить к дубликатам. Каждый посетитель должен появиться в списке только один раз.

Множество **Set** – как раз то, что нужно для этого:



```
1 let set = new Set();
2
3 let john = { name: "John" };
4 let pete = { name: "Pete" };
5 let mary = { name: "Mary" };
6
7 // считаем гостей, некоторые приходят несколько раз
8 set.add(john);
9 set.add(pete);
10 set.add(mary);
11 set.add(john);
12 set.add(mary);
13
14 // set хранит только 3 уникальных значения
15 alert(set.size); // 3
16
17 for (let user of set) {
18   alert(user.name); // John (потом Pete и Mary)
19 }
```

Альтернативой множеству **Set** может выступать массив для хранения гостей и дополнительный код для проверки уже имеющегося элемента с помощью **arr.find**. Но в этом случае будет хуже производительность, потому что **arr.find** проходит весь массив для проверки наличия элемента. Множество **Set** лучше оптимизировано для добавлений, оно автоматически проверяет на уникальность.

Перебор объекта Set

Мы можем перебрать содержимое объекта **set** как с помощью метода **for...of**, так и используя **forEach**:



```
1 let set = new Set(["апельсин", "яблоко", "банан"]);
2
3 for (let value of set) alert(value);
4
5 // то же самое с forEach:
6 set.forEach((value, valueAgain, set) => {
7   alert(value);
8 });
```

Заметим забавную вещь. Функция в **forEach** у **Set** имеет 3 аргумента: значение **value**, потом *снова то же самое значение* **valueAgain**, и только потом целевой объект. Это действительно так, значение появляется в списке аргументов дважды.

Это сделано для совместимости с объектом **Map**, в котором колбэк **forEach** имеет 3 аргумента. Выглядит немного странно, но в некоторых случаях может помочь легко заменить **Map** на **Set** и наоборот.

Set имеет те же встроенные методы, что и **Map**:

- **set.keys()** – возвращает перебираемый объект для значений,

- `set.values()` – то же самое, что и `set.keys()`, присутствует для обратной совместимости с `Map`,
- `set.entries()` – возвращает перебираемый объект для пар вида `[значение, значение]`, присутствует для обратной совместимости с `Map`.

Итого

`Map` – коллекция пар ключ-значение.

Методы и свойства:

- `new Map([iterable])` – создаёт коллекцию, можно указать перебираемый объект (обычно массив) из пар `[ключ, значение]` для инициализации.
- `map.set(key, value)` – записывает по ключу `key` значение `value`.
- `map.get(key)` – возвращает значение по ключу или `undefined`, если ключ `key` отсутствует.
- `map.has(key)` – возвращает `true`, если ключ `key` присутствует в коллекции, иначе `false`.
- `map.delete(key)` – удаляет элемент по ключу `key`.
- `map.clear()` – очищает коллекцию от всех элементов.
- `map.size` – возвращает текущее количество элементов.

Отличия от обычного объекта `Object`:

- Что угодно может быть ключом, в том числе и объекты.
- Есть дополнительные методы, свойство `size`.

`Set` – коллекция уникальных значений, так называемое «множество».

Методы и свойства:

- `new Set(iterable)` – создаёт `Set`, можно указать перебираемый объект со значениями для инициализации.
- `set.add(value)` – добавляет значение (если оно уже есть, то ничего не делает), возвращает тот же объект `set`.
- `set.delete(value)` – удаляет значение, возвращает `true` если `value` было в множестве на момент вызова, иначе `false`.
- `set.has(value)` – возвращает `true`, если значение присутствует в множестве, иначе `false`.
- `set.clear()` – удаляет все имеющиеся значения.
- `set.size` – возвращает количество элементов в множестве.

Перебор `Map` и `Set` всегда осуществляется в порядке добавления элементов, так что нельзя сказать, что это – неупорядоченные коллекции, но поменять порядок элементов или получить элемент напрямую по его номеру нельзя.

✓ Задачи

Фильтрация уникальных элементов массива

важность: 5

Допустим, у нас есть массив `arr`.

Создайте функцию `unique(arr)`, которая вернёт массив уникальных, не повторяющихся значений массива `arr`.

Например:

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

P.S. Здесь мы используем строки, но значения могут быть любого типа.

P.P.S. Используйте `Set` для хранения уникальных значений.

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

решение

```
1 function unique(arr) {  
2   return Array.from(new Set(arr));  
3 }
```

[Открыть решение с тестами в песочнице.](#)

Отфильтруйте анаграммы

важность: 4

Анаграммы – это слова, у которых те же буквы в том же количестве, но они располагаются в другом порядке.

Например:

```
1 nap - pan  
2 ear - are - era  
3 cheaters - hectares - teachers
```

Напишите функцию `aclean(arr)`, которая возвращает массив слов, очищенный от анаграмм.

Например:

```
1 let arr = ["nap", "teachers", "cheaters", "PAN", "ear", "era", "hectares"];  
2  
3 alert( aclean(arr) ); // "nap,teachers,ear" или "PAN,cheaters,era"
```

Из каждой группы анаграмм должно остаться только одно слово, не важно какое.

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

Чтобы найти все анаграммы, давайте разобьём каждое слово на буквы и отсортируем их, а потом объединим получившийся массив снова в строку. После этого все анаграммы будут одинаковы.

Например:

```
1 nap, pan -> anp
2 ear, era, are -> aer
3 cheaters, hectares, teachers -> aceehrst
4 ...
```

Мы будем использовать отсортированные строки как ключи в коллекции Map, для того чтобы сопоставить каждому ключу только одно значение:

```
1 function aclean(arr) {
2   let map = new Map();
3
4   for (let word of arr) {
5     // разбиваем слово на буквы, сортируем и объединяем снова в строку
6     let sorted = word.toLowerCase().split("").sort().join(""); // (*)
7     map.set(sorted, word);
8   }
9
10  return Array.from(map.values());
11 }
12
13 let arr = ["nap", "teachers", "cheaters", "PAN", "ear", "era", "hectare"];
14
15 alert( aclean(arr) );
```

Строка с отсортированными буквами получается в результате цепочки вызовов в строке (*).

Для удобства, давайте разделим это на несколько строк:

```
1 let sorted = arr[i] // PAN
2   .toLowerCase() // pan
3   .split("") // ["p","a","n"]
4   .sort() // ["a","n","p"]
5   .join(""); // anp
```

Два разных слова 'PAN' и 'nap' принимают ту же самую форму после сортировки букв – 'anp'.

Следующая строчка кода помещает слово в объект Map:

```
1 map.set(sorted, word);
```

Если мы когда-либо ещё встретим слово в той же отсортированной форме, тогда это слово перезапишет значение с тем же ключом в объекте. Таким образом, нескольким словам у нас будет

всегда соответствовать одна отсортированная форма.

В конце `Array.from(map.values())` принимает итерируемый объект значений объекта Map (в данном случае нам не нужны ключи) и возвращает их в виде массива.

Также в этом случае вместо `Map` мы можем использовать простой объект, потому что ключи являются строками.

В этом случае решение может выглядеть так:

```
1 function aclean(arr) {
2   let obj = {};
3
4   for (let i = 0; i < arr.length; i++) {
5     let sorted = arr[i].toLowerCase().split("").sort().join("");
6     obj[sorted] = arr[i];
7   }
8
9   return Object.values(obj);
10 }
11
12 let arr = ["nap", "teachers", "cheaters", "PAN", "ear", "era", "hectare"];
13
14 alert( aclean(arr) );
```

[Открыть решение с тестами в песочнице.](#)

Перебираемые ключи

важность: 5

Мы хотели бы получить массив ключей `map.keys()` в переменную и далее работать с ними, например, применить метод `.push`.

Но это не выходит:

```
1 let map = new Map();
2
3 map.set("name", "John");
4
5 let keys = map.keys();
6
7 // Error: keys.push is not a function
8 // Ошибка: keys.push -- это не функция
9 keys.push("more");
```

Почему? Что нужно поправить в коде, чтобы вызов `keys.push` сработал?

решение





Это потому что `map.keys()` возвращает итерируемый объект, а не массив.

Мы можем конвертировать его в массив с помощью `Array.from`:

```
1 let map = new Map();
2
3 map.set("name", "John");
4
5 let keys = Array.from(map.keys());
6
7 keys.push("more");
8
9 alert(keys); // name, more
```



Предыдущий урок

Следующий урок



Поделиться



Карта учебника

Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите [на GitHub](#).
- Для одной строки кода используйте тег `<code>`, для нескольких строк кода — тег `<pre>`, если больше 10 строк — ссылку на песочницу ([plnkr](#), [JSBin](#), [codepen...](#))
- Если что-то непонятно в статье — пишите, что именно и с какого места.

Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS ?

Имя

♡ 21

Поделиться

Лучшие Новые Старые

A

Artemoon

14 дней назад

2 задание

```
function aclean(arr) {
  let s = new Map();
  arr.forEach(arElement => {
    const sortElement = [...arElement.toLowerCase()].sort().join("");
    if (!s.has(sortElement)){
      s.set(sortElement, arElement);
    }
  })
  return Array.from(s.values());
}
```

0 0 Ответить • Поделиться ›



Nazariy Klymenko

месяц назад

Решил второе задание с помощью сета

```
function aclean(arr) {
  let standartize = arr.map(str => str.toLowerCase().split("").sort().join(""));
  let uniques = new Set(standartize);

  let uniqueArr = [];
  uniques.forEach(str => uniqueArr.push(arr[standartize.indexOf(str)]));

  return uniqueArr;
}
```

0 0 Ответить • Поделиться ›



Комментарий был удален.

Ава1

H

Ниска Де

→ Guest

месяц назад

Ответы кроются в предыдущих темах: строки, массивы

0 0 Ответить • Поделиться ›

```
let key = [1,2]
let m = new Map()
m.set(key, "12")
console.log(m.get(key)) // 12
console.log(m.get([1,2])) // undefined
```

Почему когда я хочу получить значение не по имени переменной key а по значению [1,2] то такого нет - undefined

И если добавлять так

```
m.set([1,2], "12")
m.set([1,2], "123")
m.set([1,2], "1234")
```

то в мап будет

```
Map(4) { (2) [...] → "12", (2) [...] → "12", (2) [...] → "123", (2) [...] → "1234" }
size: 4
<entries>
0: Array [ 1, 2 ] → "12"
1: Array [ 1, 2 ] → "12"
2: Array [ 1, 2 ] → "123"
3: Array [ 1, 2 ] → "1234"
```

0 0 Ответить • Поделиться ›

Н

Ниска Де

→ darknefrit

месяц назад

Интересное замечание. Ответ кроется в том, что это разные массивы хоть и выглядят так же. Более подробно про это в теме про объекты и ссылочные типы данных. Например [1,2] == [1,2] // false

0 0 Ответить • Поделиться ›

T

TheDark King

2 месяца назад

1 задача

```
let arr = ["Hare", "Krishna", "Hare", "Krishna", "Krishna", "Krishna", "Hare", "Hare", ":-O"];
function unique(arr) {
  return arr = new Set(arr);
}
console.log( unique(arr) ); // Set(3) { 'Hare', 'Krishna', ':-O' }
```

0 0 Ответить • Поделиться ›



Джун на фронте

2 месяца назад

Я хочу рассказать вам о своем пути в IT-сфере. Давным-давно я мечтал стать разработчиком, но не знал, с чего начать. И тогда я обнаружил этот сайт, который помог мне начать этот путь. 😊

Я был так вдохновлен этим обучением, что создал телеграм-канал "Лжун на фронте" где записываю

Если так удовлетворен этим сообщением, то создайте Telegram канал, дружите на фронте, где сможете свою статистику. Если вы тоже мечтаете о карьере в разработке и дизайне, то присоединяйтесь. 🚀

0 0 Ответить • Поделиться ›

П

Павел Поздняков

2 месяца назад

— 🚩

Хорошо, Set возвращает коллекцию уникальных значений. А есть ли какая-нибудь структура или метод, который позволил бы, к примеру, из двух массивов получить структуру данных, содержащую исключительно повторяющиеся в этих массивах значения?

0 0 Ответить • Поделиться ›



adriian

→ Павел Поздняков

месяц назад edited

— 🚩

Думаю есть множество решений, например через фильтрацию элементов итерируемого объекта методом **filter()**

```
function proofArr(arr1, arr2) {
    let arr3 = arr1.filter(value => arr2.includes(value));
    return arr3;
}
console.log(proofArr([1,2,3,4,5],[1,2,4,4]));
```

0 0 Ответить • Поделиться ›

П

Павел Поздняков

2 месяца назад

— 🚩

```
function unique(arr) {
    return [...new Set(arr)];
}

let values = ["Hare", "Krishna", "Hare", "Krishna",
    "Krishna", "Krishna", "Hare", "Hare", ":-0"];

unique(values);
```

0 0 Ответить • Поделиться ›

Е

Евгений Новиков

2 месяца назад

— 🚩

2-я задача:

```
function aclean(array) {
    let map = new Map();

    array.forEach(el => map.set(el.toLowerCase().split("").sort().join(""), el))

    return Array.from(map.values())
}

let arr = ["nap", "teachers", "cheaters", "PAN", "ear", "era", "hectares", "dovod", "vodod", "ooDdv"];

alert( aclean(arr) );
```

0 0 Ответить • Поделиться ›

G

Gora

2 месеца назад

Пример использования Map из практики. Типичный todo лист. Структура задач такая:

```
{
  folder1: [task1, task2]
  newFolder: [task1, task2, ...]
}
```

Папки отображались в правом меню. И тут я захотел добавить DragNDrop папок.

С объектами не получилось из-за того, что пользователь мог назвать папку "123" или "111", и в момент отрисовки папок всегда сначала шло 111 а потом 123, да и гарантировать, что остальные ключи будут выводиться в порядке добавления нельзя. Решение было использовать Map, так как он сохраняет порядок добавления и не преобразовывает строки, похожие на числа.

3 0 Ответить • Поделиться ›



Джун на фронте

3 месеца назад

Приветствую всех, кто стремится улучшить свои знания и навыки в технологиях! Я уже целый год учусь на этом сайте и решил поделиться своим опытом в **телеграм-канале "Джун на фронте!"**.

Если вы интересуетесь изучением верстки, дизайна или программирования на JavaScript, то присоединяйтесь! Вместе мы начнем увлекательный маршрут в мире технологий и найдем новые подходы для улучшения своих знаний и навыков. 🚀🌍

С удовольствием помогу, Юрий

0 0 Ответить • Поделиться ›

K

KingNut Academy

3 месеца назад

Разбор задач, тестов и всего что поможет при прохождении собеседования на должность frontend developer (html, css, js, ts, react). Заходи к нам в ТГ @interview_masters

0 0 Ответить • Поделиться ›

L

Liliana Grabde

3 месеца назад

мне кажется задачу с перебираемыми ключами надо поставить первой

1 0 Ответить • Поделиться ›



Джун на fronte

3 месеца назад

#статистика: День 467 == 611 час в ит

Уже больше года я изучаю жабаскрипт! Всю боль и удачи в обучении я записываю в **тг-дневничек Джун на фронте!** Вбивай в поиске и присоединяйся к увлекательному поиску работы:)

0 1 Ответить • Поделиться ›

J

JabaDushnila

4 месяца назад edited



Занятно, что вызов в консоли операторов перебора вроде `map.keys()` не отобразит то, что требуется. В результате будет только объект итератор.

Зато `Array.from(map.keys())` отображает.

Причина этого не ясна

В последней задаче решение не решает проблему в случае с Firefox. Там это все еще не функция :) наверное из-за 85 версии.

0 0 Ответить • Поделиться ›



Maxim

→ JabaDushnila

4 месяца назад edited



Все верно `Map.prototype.keys`, `Map.prototype.values` и `Map.prototype.entries` возвращают объекты-итераторы. Это не баг, не ошибка так задокументировано и спроектировано.

`Array.from(map.keys())` может принимать объект-итератор и конвертировать его в массив.

Причина ясна, все алгоритмы описаны в спецификации.

2 0 Ответить • Поделиться ›

B

Bohdan

4 месяца назад edited



№1

```
function unique(arr) {  
  return [...new Set(arr)] // operator spread(...) || Array.from();  
}
```

0 0 Ответить • Поделиться ›

Загрузить ещё комментарии

Подписаться

Privacy

Не продавайте мои данные

