|  |  |
| --- | --- |
| **Map** — набор пар ключ-значение, где ключ может быть любого типа. Map запоминает первоначальный порядок добавления элементов, это означает, что данные могут быть извлечены в том же порядке в каком они были вставлены.  Другими словами, Map имеет характеристики как объекта, так и массива:  -Как и объект, поддерживает структуру пар ключ-значение.  -Как и массив, запоминает порядок вставки.  **Как создать и инициализировать Map в JavaScript**  Новый Map можно создать так:  **const map = new Map();**  Другой способ создания Map — с начальными значениями. Вот как создать Map с тремя парами ключ-значение:  **const freeCodeCampBlog = new Map([**  **['name', 'freeCodeCamp'],**  **['type', 'blog'],**  **['writer', 'Tapas Adhikary'],**  **]);**  **Как добавить значения в Map**  Чтобы добавить значение в Map используйте метод set(ключ, значение).  **map.set('name', 'freeCodeCamp');**  **Как получить значение из Map**  Для получения значения из Map используйте метод get(ключ):  map.get('name'); // вернёт freeCodeCamp  **Свойства и методы Map**  JavaScript Map имеет встроенный свойства и методы, упрощающие его использование. Вот некоторые основные:  **Используйте свойство size**, чтобы узнать сколько элементов в Map:  console.log('Количество элементов в map: ', map.size);  **Поиск элементов с помощью метода has(ключ):**  // вернёт true, если map содержит элемент с ключём, 'John'  console.log(map.has('John'));  // вернёт false, если map не содержит элемент с ключём, 'Tapas'  console.log(map.has('Tapas'));  **Удаление элемента методом delete(ключ):**  map.delete('Sam'); // удалит элемент с ключём, 'Sam'.  Для удаления всех элементов используйте метод clear():  // **Очистить map удалив все элементы**  map.clear();  **Методы keys(), values() и entries()** возвращают MapIterator, что превосходно, потому что вы можете использовать цикл for-of или forEach непосредственно на нём.  Сначала создадим простой Map:  const ageMap = new Map([  ['Jack', 20],  ['Alan', 34],  ['Bill', 10],  ['Sam', 9]  ]);  **Получим все ключи:**  console.log(ageMap.keys());  // Вывод:  // MapIterator {"Jack", "Alan", "Bill", "Sam"}  **Получим все значения:**  console.log(ageMap.values());  // Вывод:  // MapIterator {20, 34, 10, 9}  **Получим все записи (пары ключ-значение):**  console.log(ageMap.entries());  // Вывод:  // MapIterator {"Jack" => 20, "Alan" => 34, "Bill" => 10, "Sam" => 9}  **Как перебрать все элементы Map**  Вы можете использовать цикл forEach или for-of для перебора Map:  **// c forEach**  ageMap.forEach((value, key) => {  console.log(`${key} is ${value} years old!`);  });  **// c for-of**  for(const [key, value] of ageMap) {  console.log(`${key} is ${value} years old!`);  }  В обоих случаях результат будет одинаковым:  Jack is 20 years old!  Alan is 34 years old!  Bill is 10 years old!  Sam is 9 years old! | **Set** — коллекция уникальных элементов, которые могут быть любого типа. Так же Set является упорядоченной коллекцией — элементы будут извлекаться в том же порядке, в каком они были вставлены.  Новый Set можно создать следующим образом:  **const set = new Set();**  console.log(set);  // Вывод:  Set(0) {}  **Вот так можно создать Set с начальными значениями:**  const fruitSet = new Set(['🍉', '🍎', '🍈', '🍏']);  console.log(fruitSet);  // Вывод:  Set(4) {"🍉", "🍎", "🍈", "🍏"}  **Методы и Свойства Set**  В Set есть методы для добавления элемента, удаления одного из элементов, проверка наличия элемента и полная очистка (удаление всех элементов):  **Свойство size** возвращает количество элементов Set:  set.size  **Для добавления элементов в Set используем метод add(элемент):**  // Создаём Set - saladSet  const saladSet = new Set();  // Add some vegetables to it  saladSet.add('🍅'); // помидор  console.log(saladSet);  **Мне нравятся помидоры! Добавим ещё один?**  **О нет, я не могу — Set это набор уникальных элементов.**  **Используем метод has**(элемент) для поиска моркови(🥕) или брокколи(🥦) в Set:  // В saladSet есть 🥕, результат: true  console.log('В салате есть морковь?', saladSet.has('🥕'));  // В saladSet нет 🥦, результат false  console.log('В салате есть брокколи?', saladSet.has(' 🍅 '));  **Используем метод delete**(элемент) для удаления помидора(🍅) из салата:  saladSet.delete(' 🍅 ');  **Используем метод clear()** для удаления всех элементов  saladSet.clear(); |

**Как конвертировать Объект в Map**

Вы можете столкнуться с ситуацией, когда нужно преобразовать object в Map структуру. Можно использовать метод entries Object и сделать так:

const address = {

'Tapas': 'Bangalore',

'James': 'Huston',

'Selva': 'Srilanka'

};

const addressMap = new Map(Object.entries(address));

**Как конвертировать Map в Объект**

Конвертировать Map в объект можно с помощью метода fromEntries:

Object.fromEntries(map)

Как конвертировать Map в Массив

Есть несколько способов преобразования Map в массив:

Использовать Array.from(map):

const map = new Map();

map.set('milk', 200);

map.set("tea", 300);

map.set('coffee', 500);

console.log(Array.from(map));

Использовать оператор Spread (...):

console.log([...map]);

**Map или Объект: Когда использовать?**

У Map есть характеристики, как object, так и array. Однако Map больше похож на object, чем на array из-за природы хранения данных в формате ключ-значение.

На этом сходство с объектами заканчивается. Как вы видели, Map во многом отличается. Итак, какой из этих типов и когда следует использовать? Как вы считаете?

**Когда использовать Map:**

-Ваши потребности не так просты. Вы можете захотеть создать ключи, которые не являются строками. Хранение объекта в качестве ключа — очень мощный подход. Map даёт такую возможность по умолчанию.

-Вам нужна структура данных с упорядоченными элементами. Обычные объекты не сохраняют порядок своих записей.

-Вы ищете гибкость не полагаясь на внешнюю библиотеку, такую, как lodash. В конечном итоге вы можете использовать библиотеку подобную lodash, так как у обычных объектов нет методов has(), values(), delete() или свойств size. Map упрощает задачу предоставляя такие методы по умолчанию.

**Когда использовать object:**

-У вас нет ни одной из вышеперечисленных потребностей.

-Вы полагаетесь на JSON.parse(), поскольку Map не может быть проанализирована им.

**Как перебрать все элементы Set**

В Set есть метод values(), возвращающий SetIterator для получения всех значений:

// Создаём Set

const houseNos = new Set([360, 567, 101]);

// Получаем SetIterator используя метод `values()`

console.log(houseNos.values());

// Вывод:

SetIterator {360, 567, 101}

**Можно использовать цикл forEach или for-of для получения значений.**

Поскольку в Set нет ключей, метод keys() возвращает SetIterator для получения значений:

console.log(houseNos.keys());

// Вывод:

// SetIterator {360, 567, 101}

**Метод entry()** возвращает итератор для извлечения пар ключ-значение. Опять же в Set нет ключей, поэтому entry() возвращает SetIterator для получения пар значение-значение:

console.log(houseNos.entries());

// Вывод:

// SetIterator {360 => 360, 567 => 567, 101 => 101}

**Мы можем вывести все значения Set используя циклы forEach и for-of:**

**// forEach**

houseNos.forEach((value) => {

console.log(value);

});

**// for-of**

for(const value of houseNos) {

console.log(value);

}

Вывод в обоих случаях одинаковый:

360

567

101

**Set и Массивы**

Массив, как и Set, позволяет добавлять и удалять элементы. Но Set совершенно иной тип и не предназначен для замены массивов.

Основное отличие в том, что массивы могут иметь повторяющиеся элементы. Кроме того, некоторые операции Set, такие, как delete() выполняются быстрее, чем операции с массивами, такие как shift() или splice().

Думайте о Set, как о расширении обычного массива, только с большим количеством мышц. Структура данных Set не является заменой массива, оба могут решать интересные задачи.

**Как конвертировать Set в Массив**

Set очень просто конвертируется в массив:

const arr = [...houseNos];

console.log(arr);

// Вывод:

// (3) [360, 567, 101]

**Получение уникальных значений из массива с помощью Set**

Создание Set — простой способ удаления повторяющихся элементов из массива:

// Создадим массив mixedFruit с несколькими дубликатами фруктов

const mixedFruit = ['🍉', '🍎', '🍉', '🍈', '🍏', '🍎', '🍈'];

// Создаём из массива уникальный набор

const mixedFruitSet = **new Set(mixedFruit);**

**Set** — набор уникальных элементов. Изменив свойство объекта, мы не изменили сам объект. Следовательно, Set не допустит дублирования элементов.

**Set** — отличная структура данных, которую можно использовать в дополнение к массивам. Однако у него нет большого преимущества пред обычными массивами.

**Используйте Set**, когда нужно поддерживать отдельный набор данных для выполнения операций над наборами, как объединение, пересечение и т.д.