[Map и Set](https://learn.javascript.ru/map-set)

Сейчас мы знаем о следующих сложных структурах данных:

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

**[Map](https://learn.javascript.ru/map-set" \l "map)**

[Map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/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()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/Map) – создаёт коллекцию.
* [map.set(key, value)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/set) – записывает по ключу key значение value.
* [map.get(key)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/get) – возвращает значение по ключу или undefined, если ключ key отсутствует.
* [map.has(key)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/has) – возвращает true, если ключ key присутствует в коллекции, иначе false.
* [map.delete(key)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/delete) – удаляет элемент (пару «ключ/значение») по ключу key.
* [map.clear()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/clear) – очищает коллекцию от всех элементов.
* [map.size](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/size) – возвращает текущее количество элементов.

let map = new Map();

map.set("1", "str1"); // строка в качестве ключа

map.set(1, "num1"); // цифра как ключ

map.set(true, "bool1"); // булево значение как ключ

// помните, обычный объект Object приводит ключи к строкам?

// Map сохраняет тип ключей, так что в этом случае сохранится 2 разных значения:

alert(map.get(1)); // "num1"

alert(map.get("1")); // "str1"

alert(map.size); // 3

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

**map[key] это не совсем правильный способ использования Map**

Хотя map[key] также работает(не работает), например, мы можем установить map[key] = 2, в этом случаеmap 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.**

Использование объектов в качестве ключей – одна из наиболее заметных и важных функций Map.

let john = { name: "John" };

// давайте сохраним количество посещений для каждого пользователя

let visitsCountMap = new Map();

// объект john - это ключ для значения в объекте Map

visitsCountMap.set(john, 123);

alert(visitsCountMap.get(john)); // 123

Это то что невозможно для Object.

let john = { name: "John" };

let ben = { name: "Ben" };

let visitsCountObj = {}; // попробуем использовать объект

visitsCountObj[ben] = 234; // пробуем использовать объект ben в качестве ключа

visitsCountObj[john] = 123; // пробуем использовать объект john в качестве ключа, при этом объект ben будет замещён

// Вот что там было записано!

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]".

**Как объект Map сравнивает ключи**

Чтобы сравнивать ключи, объект Map использует алгоритм [SameValueZero](https://tc39.github.io/ecma262/" \l "sec-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.

**Цепочка вызовов**

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

map.set("1", "str1")

.set(1, "num1")

.set(true, "bool1");

## [Перебор Map](https://learn.javascript.ru/map-set" \l "perebor-map)

Для перебора коллекции Map есть 3 метода:

* [map.keys()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/keys) – возвращает итерируемый объект по ключам,
* [map.values()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/values) – возвращает итерируемый объект по значениям,
* [map.entries()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/entries) – возвращает итерируемый объект по парам вида [ключ, значение], этот вариант используется по умолчанию в for..of.

let recipeMap = new Map([

["огурец", 500],

["помидор", 350],

["лук", 50]

]);

// перебор по ключам (овощи)

for (let vegetable of recipeMap.keys()) {

alert(vegetable); // огурец, помидор, лук

}

// перебор по значениям (числа)

for (let amount of recipeMap.values()) {

alert(amount); // 500, 350, 50

}

// перебор по элементам в формате [ключ, значение]

for (let entry of recipeMap) { // то же самое, что и recipeMap.entries()

alert(entry); // огурец,500 (и так далее)

}

**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:

// выполняем функцию для каждой пары (ключ, значение)

recipeMap.forEach((value, key, map) => {

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

});

## [Object.entries: Map из Object](https://learn.javascript.ru/map-set" \l "object-entries-map-iz-object)

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

// массив пар [ключ, значение]

let map = new Map([

['1', 'str1'],

[1, 'num1'],

[true, 'bool1']

]);

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)](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/entries), 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:

let obj = {

name: "John",

age: 30

};

let map = new Map(Object.entries(obj));

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](https://learn.javascript.ru/map-set" \l "object-fromentries-object-iz-map)

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

let prices = Object.fromEntries([

['banana', 1],

['orange', 2],

['meat', 4]

]);

// prices = { banana: 1, orange: 2, meat: 4 }

alert(prices.orange); // 2

Можно 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:

let map = new Map();

map.set('banana', 1);

map.set('orange', 2);

map.set('meat', 4);

let obj = Object.fromEntries(map.entries()); // создаём обычный объект (\*)

// готово!

// obj = { banana: 1, orange: 2, meat: 4 }

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:

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](https://learn.javascript.ru/map-set" \l "set)

Объект [Set](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set) – это особый вид коллекции: «множество» значений (без ключей), где каждое значение может появляться только один раз.

Его основные методы это:

* [new Set(iterable)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/Set) – создаёт Set, и если в качестве аргумента был предоставлен итерируемый объект (обычно это массив), то копирует его значения в новый Set.
* [set.add(value)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/add) – добавляет значение (если оно уже есть, то ничего не делает), возвращает тот же объект set.
* [set.delete(value)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/delete) – удаляет значение, возвращает true, если value было в множестве на момент вызова, иначе false.
* [set.has(value)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/has) – возвращает true, если значение присутствует в множестве, иначе false.
* [set.clear()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/clear) – удаляет все имеющиеся значения.
* [set.size](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/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

let set = new Set();

let john = { name: "John" };

let pete = { name: "Pete" };

let mary = { name: "Mary" };

// считаем гостей, некоторые приходят несколько раз

set.add(john);

set.add(pete);

set.add(mary);

set.add(john);

set.add(mary);

// set хранит только 3 уникальных значения

alert(set.size); // 3

for (let user of set) {

alert(user.name); // John (потом Pete и Mary)

}

Альтернативой множеству Set может выступать массив для хранения гостей и дополнительный код для проверки уже имеющегося элемента с помощью [arr.find](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/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](https://learn.javascript.ru/map-set" \l "perebor-obekta-set)

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

let set = new Set(["апельсин", "яблоко", "банан"]);

for (let value of set) alert(value);

// то же самое с forEach:

set.forEach((value, valueAgain, set) => {

alert(value);

});

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()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/keys) – возвращает перебираемый объект для значений,
* [set.values()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/values) – то же самое, что и set.keys(), присутствует для обратной совместимости с Map,
* [set.entries()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/entries) – возвращает перебираемый объект для пар вида [значение, значение], присутствует для обратной совместимости с Map.

Отличия map от обычного объекта Object:

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

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.