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 по ключу `key`.
- `map.clear()` – очищает коллекцию от всех элементов.
- `map.size` – возвращает текущее количество элементов.

Например:

```
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 стро
// Map сохраняет тип ключей, так что в этом случае сохр
alert(map.get(1)); // "num1"
alert(map.get("1")); // "str1"

alert(map.size); // 3
```

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

Map может использовать объекты в качестве ключей.

Например:

```
let john = { name: "John" };

// давайте сохраним количество посещений для каждого по
let visitsCountMap = new Map();

// объект john - это ключ для значения в объекте Map
visitsCountMap.set(john, 123);

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

Использование объектов в качестве ключей – это одна из известных и часто применяемых возможностей объекта `Map`. При строковых ключах

обычный объект `Object` может подойти, но для ключей-объектов – уже нет.

Попробуем заменить `Map` на `Object` в примере выше:

```
let john = { name: "John" };

let visitsCountObj = {}; // попробуем использовать объе

visitsCountObj[john] = 123; // возьмём объект john как

// Вот как это было записано!
alert( visitsCountObj["[object Object]"] ); // 123
```

Так как `visitsCountObj` – это объект, то все ключи он автоматически преобразует к строке, в итоге получился строковой ключ `"[object Object]"`. Это не то, чего мы хотим.

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.

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

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:

```
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 re
  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

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), 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));
```

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:

```
let prices = Object.fromEntries([
  ['banana', 1],
  ['orange', 2],
  ['meat', 4]
]);

// now 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()); // make a
7
8  // готово!
9  // obj = { banana: 1, orange: 2, meat: 4 }
10
11 alert(obj.orange); // 2
```

Вызов `map.entries()` возвращает массив 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" };
```

```
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. 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`:

```
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.values()` – возвращает перебираемый объект для значений,
- `set.keys()` – то же самое, что и `set.values()`, 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:

```
function unique(arr) {
  /* ваш код */
}

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

alert( unique(values) ); // Hare,Krishna,:-O
```

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

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",
2
3  alert( aclean(arr) ); // "nap,teachers,ear" or "PAN,che
```

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

Открыть песочницу с тестами для задачи.

решение

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

Комментарии

перед тем как писать...