Set 和 Map 数据结构

1. Set

1.1. 基本用法

Set 本身是一个构造函数，用来生成 Set 数据结构。它类似于数组，但是成员的值都是唯一的，没有重复的值。

```
const s = new Set();
[2, 3, 5, 4, 5, 2, 2].forEach(x => s.add(x));
for (let i of s) {
  console.log(i);
}
// 2 3 5 4
```

通过 forEach 遍历数组 [2, 3, 5, 4, 5, 2, 2] 将其成员加入到 s，最终 s 只有 2 3 5 4 四个没有重复的值，后面的 5 2 2 因为与前面的成员重复，没有被加入到 s。证明 **Set 数据结构的值都是唯一的，没有重复。**

Set 函数可以接受一个数组（或者具有 iterable 接口的其他数据结构，如字符串、NodeList）作为参数，用来初始化。

```
// 例一，数组作为 Set 的参数
const set1 = new Set([1, 2, 3, 3]);
[...set1]; // [1, 2, 3]
set1.size; // 3

// 例二，字符串作为 Set 的参数
const set2 = new Set("abcbcdcde");
set2; // Set(5) {'a', 'b', 'c', 'd', 'e'}
[...set2]; // ['a', 'b', 'c', 'd', 'e']

// 例三
const set3 = new Set(document.querySelectorAll('div'));
set3.size // 31，本页面 div 标签的个数
```

根据 Set 数据结构的成员是唯一的，没有重复这一特点，可以用于数组去除重复成员：

```
[...new Set(array)];
Array.from(new Set(array));
```

也可以用于去除字符串里面的重复字符：

```
// Set 数据结构可以接受字符串作为参数，将其转换为数组，再转换为字符串，达到去除字符串重复成员的目的
[...new Set('abcbcdcde')].join(''); // 'abcde'
```

在 Set 内部，两个 NaN 是相等的。

```
let set = new Set();
let a = NaN;
let b = NaN;
set.add(a);
set.add(b);
set // Set {NaN}
// 添加了两个 NaN，但是只保留了一个，由于 Set 数据结构中值是唯一的没有重复，证明 Set 内部将两个 NaN 认为是重复的
```

在 Set 内部，两个对象总是不相等的。

```
let set = new Set();
set.add({});
set.add({});
set.size // 2
// 添加了两个空对象，成员是两个，证明将两个空对象被视为两个值。
```

1.2. Set 实例的属性

1.2.1. Set.prototype.constructor

`Set.prototype.constructor` 构造函数，默认就是 Set 函数。

```
Set.prototype.constructor === Set; // true
```

1.2.2. Set.prototype.size

`Set.prototype.size` 返回 Set 实例的成员总数。

```
let s = new Set([1, 2, 3]);
s.size; // 3
```

1.3. Set 的操作方法

> - Set.prototype.add(value)：添加某个值，返回 Set 结构本身。
> - Set.prototype.delete(value)：删除某个值，返回一个布尔值，表示删除是否成功。

> - Set.prototype.has(value)：返回一个布尔值，表示该值是否为Set的成员。
> - Set.prototype.clear()：清除所有成员，没有返回值。

```
let s = new Set();
s.add(1).add(2).add(2);

s.size // 2

s.has(1) // true
s.has(2) // true
s.has(3) // false

s.delete(2) // true
s.has(2) // false

s.clear(); // undefined（没有返回值）
s.size; // 0
```

1.4. Set 的遍历方法

> - Set.prototype.keys()：返回键名的遍历器
> - Set.prototype.values()：返回键值的遍历器
> - Set.prototype.entries()：返回键值对的遍历器
> - Set.prototype.forEach()：使用回调函数遍历每个成员

Set的遍历顺序就是插入顺序，使用 Set 保存一个回调函数列表，调用时就能保证按照添加顺序调用。

1.4.1. keys()，values()，entries()

`keys()`、`values()`、`entries()` 返回的都是遍历器对象。由于 Set 结构没有键名，只有键值（或者说键名和键值是同一个值），所以 `keys()` 方法和 `values()` 方法的行为完全一致。

```
let set = new Set(['red', 'green', 'blue']);

for (let item of set.keys()) {
  console.log(item);
}
// red
// green
// blue

for (let item of set.values()) {
  console.log(item);
}
// red
// green
// blue

for (let item of set.entries()) {
```

```
  console.log(item);
}
// ["red", "red"]
// ["green", "green"]
// ["blue", "blue"]
```

上例中，`entries()` 返回的遍历器，同时包括键名和键值，每次输出一个数组，它的两个成员完全相等。

Set 结构的实例默认可遍历，它的默认遍历器生成函数就是它的 `values()` 方法。

```
Set.prototype[Symbol.iterator] === Set.prototype.values; // true
```

这意味着，**可以省略 `values()` 方法，直接用 `for...of` 循环遍历 Set。**

```
let set = new Set(['red', 'green', 'blue']);
for (let x of set) {
  console.log(x);
}
// red
// green
// blue
```

1.4.2. forEach()

Set 结构的实例与数组一样，也拥有 `forEach` 方法，用于对每个成员执行某种操作，没有返回值。

```
let set = new Set([1, 4, 9]);
set.forEach((value, key) => console.log(key + ' : ' + value))
// 1 : 1
// 4 : 4
// 9 : 9
```

上例 `forEach` 方法的参数就是一个处理函数。该函数的参数与数组的 `forEach` 一致，依次为键值、键名、集合本身（上例省略了该参数）。**Set 结构的键名就是键值（两者是同一个值），因此第一个参数与第二个参数的值永远都是一样的。**

1.4.3. 遍历的应用

扩展运算符（...）内部使用for...of循环，所以也可以用于 Set 结构。

```
let set = new Set(['red', 'green', 'blue']);
let arr = [...set];
// ['red', 'green', 'blue']
```

数组的 `map` 和 `filter` 方法也可以间接用于 Set 了。

```
let set = new Set([1, 2, 3]);
set = new Set([...set].map(x => x * 2));
// 返回Set结构：{2, 4, 6}

let set = new Set([1, 2, 3, 4, 5]);
set = new Set([...set].filter(x => (x % 2) == 0));
// 返回Set结构：{2, 4}
```

因此使用 Set 可以很容易地实现并集（Union）、交集（Intersect）和差集（Difference）。

```
// 先构造需要求并集/交集/差集的 Set 数据结构 a 和 b
let a = new Set([1, 2, 3]);
let b = new Set([4, 3, 2]);

// 并集。将 a 和 b 转换成数组并合并起来
let union = new Set([...a, ...b]);
// Set {1, 2, 3, 4}

// 交集。将 a 转换成数组，并求 b 中“不包含 a 的成员”的成员；或将 b 转换成数组，并求 a 中“不包含 b 的成员”的成员
let intersect1 = new Set([...a].filter(x => b.has(x)));
// set {2, 3}
let intersect2 = new Set([...b].filter(x => a.has(x)));
// set {3, 2}

//  (a 相对于 b 的) 差集
let difference1 = new Set([...a].filter(x => !b.has(x)));
// Set {1}

//  (b 相对于 a 的) 差集
let difference2 = new Set([...b].filter(x => !a.has(x)));
// Set {4}
```

2. Map

2.1. 含义和基本用法

JavaScript 的对象（Object），只能用字符串当作键。ES6 提供了 Map 数据结构，它类似于对象，也是键值对的集合，但是各种类型的值（包括对象）都可以当作键。Object 结构提供了“字符串—值”的对应，Map 结构提供了“值—值”的对应，是一种更完善的 Hash 结构实现。如果需要“键值对”的数据结构，Map 比 Object 更合适。

```
const m = new Map();
const o = {p: 'Hello World'};
```

```
m.set(o, 'content'); // 给 m 加成员
m.get(o) // "content" , 读取成员

m.has(o) // true, 判断是否有该成员
m.delete(o) // true, 删除成员
m.has(o) // false
```

Map 作为构造函数，可以接受一个数组作为参数。该数组的成员是一个个表示键值对的数组。

```
const map = new Map([['name', '张三'], ['title', 'Author']]);
map.size // 2
map.has('name') // true
map.get('name') // "张三"
map.has('title') // true
map.get('title') // "Author"
```

`Map` 构造函数接受数组作为参数，实际上执行的是下面的算法：

```
const items = [['name', '张三'], ['title', 'Author']];
const map = new Map();
items.forEach(([key, value]) => map.set(key, value));
```

事实上，不仅仅是数组，任何具有 Iterator 接口、且每个成员都是一个双元素的数组的数据结构都可以当作 Map 构造函数的参数。这就是说，Set 和 Map 都可以用来生成新的 Map。

```
const set = new Set([['foo', 1], ['bar', 2]]);
const m1 = new Map(set);
m1.get('foo') // 1

const m2 = new Map([['baz', 3]]);
const m3 = new Map(m2);
m3.get('baz') // 3
```

如果对同一个键多次赋值，后面的值将覆盖前面的值。

```
const map = new Map();
map.set(1, 'aaa').set(1, 'bbb'); // 'bbb' 将覆盖前面的
map.get(1) // "bbb"
```

如果读取一个未知的键，则返回 `undefined`。

```
new Map().get('asfddfsasadf'); // undefined
```

只有对同一个对象的引用，Map 结构才将其视为同一个键。

```
const map = new Map();
map.set(['a'], 555);
map.get(['a']) // undefined
```

上例的 set 和 get 方法，表面是针对同一个键，但实际上这是两个不同的数组实例，内存地址是不一样的，因此 get 方法无法读取该键，返回 undefined。

同样的值的两个实例，在 Map 结构中被视为两个键。

```
const map = new Map();
const k1 = ['a'];
const k2 = ['a'];
map.set(k1, 111).set(k2, 222);
map.get(k1) // 111
map.get(k2) // 222
```

Map 的键实际上是跟内存地址绑定的，只要内存地址不一样，就视为两个键。这就解决了同名属性碰撞（clash）的问题，我们扩展别人的库的时候，如果使用对象作为键名，就不用担心自己的属性与原作者的属性同名。

如果 Map 的键是一个简单类型的值（数字、字符串、布尔值），则只要两个值严格相等，Map 将其视为一个键：

```
let map = new Map();

// 0 和 -0 就是一个键
map.set(-0, 123);
map.get(+0) // 123

// 布尔值 true 和字符串 'true' 则是两个不同的键
map.set(true, 1);
map.set('true', 2);
map.get(true) // 1

// undefined 和 null 是两个不同的键
map.set(undefined, 3);
map.set(null, 4);
map.get(undefined) // 3

// NaN 是一个键
map.set(NaN, 123);
map.get(NaN) // 123
```