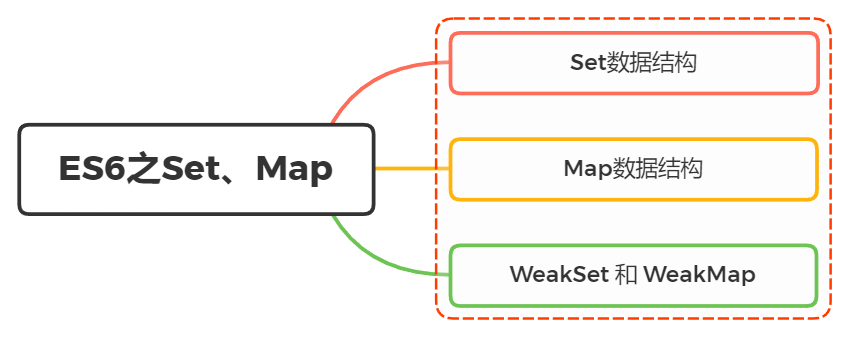
# 面试官：你是怎么理解ES6新增Set、Map两种数据结构的？



如果要用一句来描述，我们可以说

Set是一种叫做集合的数据结构，Map是一种叫做字典的数据结构

什么是集合？什么又是字典？

* 集合  
  是由一堆无序的、相关联的，且不重复的内存结构【数学中称为元素】组成的组合
* 字典  
  是一些元素的集合。每个元素有一个称作key 的域，不同元素的key 各不相同

区别？

* 共同点：集合、字典都可以存储不重复的值
* 不同点：集合是以[值，值]的形式存储元素，字典是以[键，值]的形式存储

## 一、Set

Set是es6新增的数据结构，类似于数组，但是成员的值都是唯一的，没有重复的值，我们一般称为集合

Set本身是一个构造函数，用来生成 Set 数据结构

const s = new Set();

### 增删改查

Set的实例关于增删改查的方法：

* add()
* delete()
* has()
* clear()

### add()

添加某个值，返回 Set 结构本身

当添加实例中已经存在的元素，set不会进行处理添加

s.add(1).add(2).add(2); // 2只被添加了一次

### delete()

删除某个值，返回一个布尔值，表示删除是否成功

s.delete(1)

### has()

返回一个布尔值，判断该值是否为Set的成员

s.has(2)

### clear()

清除所有成员，没有返回值

s.clear()

### 遍历

Set实例遍历的方法有如下：

关于遍历的方法，有如下：

* keys()：返回键名的遍历器
* values()：返回键值的遍历器
* entries()：返回键值对的遍历器
* 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

keys方法、values方法、entries方法返回的都是遍历器对象

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"]

forEach()用于对每个成员执行某种操作，没有返回值，键值、键名都相等，同样的forEach方法有第二个参数，用于绑定处理函数的this

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

扩展运算符和Set 结构相结合实现数组或字符串去重

// 数组  
let arr = [3, 5, 2, 2, 5, 5];  
let unique = [...new Set(arr)]; // [3, 5, 2]  
  
// 字符串  
let str = "352255";  
let unique = [...new Set(str)].join(""); // ""

实现并集、交集、和差集

let a = new Set([1, 2, 3]);  
let b = new Set([4, 3, 2]);  
  
// 并集  
let union = new Set([...a, ...b]);  
// Set {1, 2, 3, 4}  
  
// 交集  
let intersect = new Set([...a].filter(x => b.has(x)));  
// set {2, 3}  
  
// （a 相对于 b 的）差集  
let difference = new Set([...a].filter(x => !b.has(x)));  
// Set {1}

## 二、Map

Map类型是键值对的有序列表，而键和值都可以是任意类型

Map本身是一个构造函数，用来生成 Map 数据结构

const m = new Map()

### 增删改查

Map 结构的实例针对增删改查有以下属性和操作方法：

* size 属性
* set()
* get()
* has()
* delete()
* clear()

### size

size属性返回 Map 结构的成员总数。

const map = new Map();  
map.set('foo', true);  
map.set('bar', false);  
  
map.size // 2

### set()

设置键名key对应的键值为value，然后返回整个 Map 结构

如果key已经有值，则键值会被更新，否则就新生成该键

同时返回的是当前Map对象，可采用链式写法

const m = new Map();  
  
m.set('edition', 6) // 键是字符串  
m.set(262, 'standard') // 键是数值  
m.set(undefined, 'nah') // 键是 undefined  
m.set(1, 'a').set(2, 'b').set(3, 'c') // 链式操作

### get()

get方法读取key对应的键值，如果找不到key，返回undefined

const m = new Map();  
  
const hello = function() {console.log('hello');};  
m.set(hello, 'Hello ES6!') // 键是函数  
  
m.get(hello) // Hello ES6!

### has()

has方法返回一个布尔值，表示某个键是否在当前 Map 对象之中

const m = new Map();  
  
m.set('edition', 6);  
m.set(262, 'standard');  
m.set(undefined, 'nah');  
  
m.has('edition') // true  
m.has('years') // false  
m.has(262) // true  
m.has(undefined) // true

### delete()

delete方法删除某个键，返回true。如果删除失败，返回false

const m = new Map();  
m.set(undefined, 'nah');  
m.has(undefined) // true  
  
m.delete(undefined)  
m.has(undefined) // false

### clear()

clear方法清除所有成员，没有返回值

let map = new Map();  
map.set('foo', true);  
map.set('bar', false);  
  
map.size // 2  
map.clear()  
map.size // 0

### 遍历

Map结构原生提供三个遍历器生成函数和一个遍历方法：

* keys()：返回键名的遍历器
* values()：返回键值的遍历器
* entries()：返回所有成员的遍历器
* forEach()：遍历 Map 的所有成员

遍历顺序就是插入顺序

const map = new Map([  
 ['F', 'no'],  
 ['T', 'yes'],  
]);  
  
for (let key of map.keys()) {  
 console.log(key);  
}  
// "F"  
// "T"  
  
for (let value of map.values()) {  
 console.log(value);  
}  
// "no"  
// "yes"  
  
for (let item of map.entries()) {  
 console.log(item[0], item[1]);  
}  
// "F" "no"  
// "T" "yes"  
  
// 或者  
for (let [key, value] of map.entries()) {  
 console.log(key, value);  
}  
// "F" "no"  
// "T" "yes"  
  
// 等同于使用map.entries()  
for (let [key, value] of map) {  
 console.log(key, value);  
}  
// "F" "no"  
// "T" "yes"  
  
map.forEach(function(value, key, map) {  
 console.log("Key: %s, Value: %s", key, value);  
});

## 三、WeakSet 和 WeakMap

### WeakSet

创建WeakSet实例

const ws = new WeakSet();

WeakSet可以接受一个具有 Iterable接口的对象作为参数

const a = [[1, 2], [3, 4]];  
const ws = new WeakSet(a);  
// WeakSet {[1, 2], [3, 4]}

在API中WeakSet与Set有两个区别：

* 没有遍历操作的API
* 没有size属性

WeackSet只能成员只能是引用类型，而不能是其他类型的值

let ws=new WeakSet();  
  
// 成员不是引用类型  
let weakSet=new WeakSet([2,3]);  
console.log(weakSet) // 报错  
  
// 成员为引用类型  
let obj1={name:1}  
let obj2={name:1}  
let ws=new WeakSet([obj1,obj2]);   
console.log(ws) //WeakSet {{…}, {…}}

WeakSet里面的引用只要在外部消失，它在 WeakSet里面的引用就会自动消失

### WeakMap

WeakMap结构与Map结构类似，也是用于生成键值对的集合

在API中WeakMap与Map有两个区别：

* 没有遍历操作的API
* 没有clear清空方法

// WeakMap 可以使用 set 方法添加成员  
const wm1 = new WeakMap();  
const key = {foo: 1};  
wm1.set(key, 2);  
wm1.get(key) // 2  
  
// WeakMap 也可以接受一个数组，  
// 作为构造函数的参数  
const k1 = [1, 2, 3];  
const k2 = [4, 5, 6];  
const wm2 = new WeakMap([[k1, 'foo'], [k2, 'bar']]);  
wm2.get(k2) // "bar"

WeakMap只接受对象作为键名（null除外），不接受其他类型的值作为键名

const map = new WeakMap();  
map.set(1, 2)  
// TypeError: 1 is not an object!  
map.set(Symbol(), 2)  
// TypeError: Invalid value used as weak map key  
map.set(null, 2)  
// TypeError: Invalid value used as weak map key

WeakMap的键名所指向的对象，一旦不再需要，里面的键名对象和所对应的键值对会自动消失，不用手动删除引用

举个场景例子：

在网页的 DOM 元素上添加数据，就可以使用WeakMap结构，当该 DOM 元素被清除，其所对应的WeakMap记录就会自动被移除

const wm = new WeakMap();  
  
const element = document.getElementById('example');  
  
wm.set(element, 'some information');  
wm.get(element) // "some information"

注意：WeakMap 弱引用的只是键名，而不是键值。键值依然是正常引用

下面代码中，键值obj会在WeakMap产生新的引用，当你修改obj不会影响到内部

const wm = new WeakMap();  
let key = {};  
let obj = {foo: 1};  
  
wm.set(key, obj);  
obj = null;  
wm.get(key)  
// Object {foo: 1}

## 参考文献

* https://es6.ruanyifeng.com/#docs/set-map