**Set、WeakSet、Map、WeakMap、WeakRef介绍及使用场景**

**一、Set**

**1、介绍**

①set类似于数组，但成员的数值都是唯一的，无重复的值，可以存储任何类型的数据。  
②set只有一个size属性。

const set = new Set([1,2,3]);

console.log(set.size) // 3

③方法：

set.has(1); // true

set.add(4); // {1,2,3,4}

set.delete(4); // true

set.clear(); // 无返回值

set.keys(); // SetIterator {1, 2, 3}

set.values(); // SetIterator {1, 2, 3}

set.entries(); // SetIterator {1 => 1, 2 => 2, 3 => 3}

**2、使用场景**

①[数组去重](https://so.csdn.net/so/search?q=%E6%95%B0%E7%BB%84%E5%8E%BB%E9%87%8D&spm=1001.2101.3001.7020)

let arr = [1, 2, 3, 3]

console.log([...new Set(arr)]); // [1, 2, 3]

②数组转换

const set = new Set([1,2,3]);

console.log(Array.from(set)); // [1, 2, 3]

③取数组的并集/交集

const arr1 = [1,2,3,4,5,6]

const arr2 = [3,4,5,6,7,8]

// 并集

const mergeArr = [...new Set([...arr1,...arr2])] // [1, 2, 3, 4, 5, 6, 7, 8]

// 交集

const overlapArr = [...new Set(arr1)].filter(item => {

return new Set(arr2).has(item)

}) // [3, 4, 5, 6]

**二、WeakSet**

**1、介绍**

①：weakset是弱集合,成员的数值不能重复，数据类型只能是对象。  
②：如果其他对象都不再引用该对象，那么[垃圾回收机制](https://so.csdn.net/so/search?q=%E5%9E%83%E5%9C%BE%E5%9B%9E%E6%94%B6%E6%9C%BA%E5%88%B6&spm=1001.2101.3001.7020)会自动回收该对象所占用的内存。  
③：适合跟踪对象（例如检查对象是否已处理过）  
④：无size属性，不可遍历。  
⑤：方法：

const obj1 = { name: "garlic" }

const obj2 = { name: "lily" }

const weakSet= new WeakSet([obj1])

weakSet.add(obj2)

weakSet.delete(obj2) // true

weakSet.has(obj1) // true

**2、使用场景**

①存储DOM节点，防止移除节点时引起内存泄漏

const disabledElements = new WeakSet();

const loginButton = document.querySelector('.login'); // 通过加入对应集合，给这个节点打上“禁用”标签

disabledElements.add(loginButton);

// 这样，只要 WeakSet 中任何元素从 DOM 树中被删除，垃圾回收程序就可以忽略其存在，而立即释放其内存（假设没有其他地方引用这个对象）。

②weakset对实例的引用，不会被计入内存回收机制，所以删除实例的时候，不用考虑weakset，也不会出现内存泄漏。

// 保证了Foo的实例方法，只能在Foo的实例上调用

const foos = new WeakSet()

class Foo {

constructor() {

foos.add(this)

}

method () {

if (!foos.has(this)) {

throw new TypeError('Foo.prototype.method 只能在Foo的实例上调用！');

}

}

}

**三、Map**

**1、介绍**

①键值对集合，值—值对应（键可以是任何类型），有size属性。

const map = new Map([

['name', 'garlic'],

['age', 1],

])

map.size //2

②方法：

map.set("sex","woman");

map.get("name"); // "garlic"

map.has("name"); // true

map.delete("name"); // true

map.clear(); // 无返回值 undefined

map.keys(); // MapIterator {'name', 'age'}

map.values(); // MapIterator {'garlic', 1}

map.entries(); // MapIterator {'name' => 'garlic', 'age' => 1} 同for(let i of map)

map.forEach((item, index));

**2、Map与Object**

①：Map的键可以是任何类型，Object只能是number、string、symbol。  
②：固定大小内存，Map比Object多存储50%的键值对。  
③：Map适用于大量插入\删除操作，Object适用于大量查找操作。

**3、使用场景**

①：进行条件判断（如路由跳转）

const map = new Map([

["login", "/login"],

["403", "/403"],

["402", "/402"],

]);

this.$router.push({ path: map.get("login") });

[...map] // map转数组

②：map转数组

const map = new Map([

['name', "lily"],

['age', 22],

]);

console.log([...map.values()]); // ['lily', 22]

③：map转对象

const map = new Map([

['name', "lily"],

['age', 12],

]);

function mapToObj(strMap) {

let obj = Object.create(null);

for (let [k,v] of strMap) {

obj[k] = v;

}

return obj;

}

mapToObj(map); // {name: "lily", age: 12}

**四、WeakMap**

**1、介绍**

①：WeakMap只接受对象作为键名（null除外）。  
②：WeakMap的键名所指向的对象，不计入垃圾回收机制。  
③：方法：

const wm = new WeakMap([]);

const obj = { name: "lily" };

wm.set(obj, 18); // {{…} => 18}

wm.get(obj); // 18

wm.has(obj); // true

wm.delete(obj); //true

**2、使用场景**

①：DOM 节点作为键名，在Dom对象上存储数据

let wm = new WeakMap();

let element = document.querySelector(".element");

wm.set(element, "data");

wm.get(elemet); // data

②：实现私有属性

const privateData = new WeakMap()

export default class Student {

constructor(name, age) {

privateData.set(this, {name, age})

}

getName() {

return privateData.get(this).name

}

getAge() {

return privateData.get(this).age

}

}

**五、WeakRef**

**1、介绍**

①：直接创建对象的弱引用。

let target = {};

let wr = new WeakRef(target);

②：主要用处就是作为缓存，未被清除时可以从缓存取值，一旦清除缓存就自动失效。  
③：方法：  
deref()：原始对象存在，该方法返回原始对象；原始对象已经被垃圾回收机制清除，该方法返回undefined。

let target = {};

let wr = new WeakRef(target);

let obj = wr.deref();

if (obj) { // 判断原始对象是否已被清除

// ...

}