#Set

用于存储任何类型的唯一值，无论是基本类型还是对象引用。

- 只能保存值没有键名
- 严格类型检测如字符串数字不等于数值型数字
- 值是唯一的
- 遍历顺序是添加的顺序，方便保存回调函数

#基本使用

对象可以属性最终都会转为字符串

```
let obj = { 1: "hdcms", "1": "houdunren" };
console.table(obj); //{1:"houdunren"}

```

使用对象做为键名时，会将对象转为字符串后使用

```
let obj = { 1: "hdcms", "1": "houdunren" };
console.table(obj);

let hd = { [obj]: "后盾人" };
console.table(hd);

console.log(hd[obj.()]);
console.log(hd["[object Object]"]);

```

使用数组做初始数据

```
let hd = new Set(['后盾人', 'hdcms']);
console.log(hd.values()); //{"后盾人", "hdcms"}

```

Set 中是严格类型约束的，下面的数值`1`与字符串`1`属于两个不同的值

```
let set = new Set();
set.add(1);
```

```
set.add("1");
console.log(set); //Set(2) {1, "1"}

```

使用 `add` 添加元素，不允许重复添加`hdcms`值

```
let hd = new Set();

hd.add('houdunren');
hd.add('hdcms');
hd.add('hdcms')

console.log(hd.values()); //SetIterator {"houdunren", "hdcms"}

```

#获取数量

获取元素数量

```
let hd = new Set(['后盾人', 'hdcms']);
console.log(hd.size); //2

```

#元素检测

检测元素是否存在

```
let hd = new Set();
hd.add('hdcms');
console.log(hd.has('hdcms'));//true

```

#删除元素

使用 `delete` 方法删除单个元素，返回值为`boolean`类型

```
let hd = new Set();
hd.add("hdcms");
```

```
hd.add("houdunren");

console.log(hd.delete("hdcms")); //true

console.log(hd.values());
console.log(hd.has("hdcms")); //false

```

使用 `clear` 删除所有元素

```
let hd = new Set();
hd.add('hdcms');
hd.add('houdunren');
hd.clear();
console.log(hd.values());

```

#数组转换

可以使用点语法 或 `Array.form` 静态方法将Set类型转为数组，这样就可以使用数组处理函数了

```
const set = new Set(["hdcms", "houdunren"]);
console.log([...set]); //["hdcms", "houdunren"]
console.log(Array.from(set)); //["hdcms", "houdunren"]

```

移除Set中大于5的数值

```
let hd = new Set("123456789");
hd = new Set([...hd].filter(item => item < 5));
console.log(hd);

```

#去除重复

去除字符串重复

```
console.log([...new Set("houdunren")].join(""));//houdnre
```

```
2
```

去除数组重复

```
const arr = [1, 2, 3, 5, 2, 3];
console.log(...new Set(arr)); // 1,2,4,5

```

#遍历数据

使用 `keys()/values()/entries()` 都可以返回迭代对象，因为`set`类型只有值所以 `keys与values` 方法结果一致。

```
const hd = new Set(["hdcms", "houdunren"]);
console.log(hd.values()); //SetIterator {"hdcms", "houdunren"}
console.log(hd.keys()); //SetIterator {"hdcms", "houdunren"}
console.log(hd.entries()); //SetIterator {"hdcms" => "hdcms", "houdunren" =>
"houdunren"}

```

可以使用 `forEach` 遍历Set数据，默认使用 `values` 方法创建迭代器。
为了保持和遍历数组参数统一，函数中的value与key是一样的。

```
let arr = [7, 6, 2, 8, 2, 6];
let set = new Set(arr);
//使用forEach遍历
set.forEach((item,key) => console.log(item,key));

```

也可以使用 `forof` 遍历Set数据，默认使用 `values` 方法创建迭代器

```
//使用for/of遍历
let set = new Set([7, 6, 2, 8, 2, 6]);

for (const iterator of set) {
        console.log(iterator);
}

```

#搜索实例

下面通过历史搜索的示例体验`Set` 类型


```
<style>
  body {
      padding: 200px;
  }

  * {
      padding: 0;
      margin: 0;
  }

  input {
      width: 200px;
      border: solid 1px #d63031;
      outline: none;
      padding: 10px;
      box-sizing: border-box;
  }

  ul {
      list-style: none;
      width: 200px;
```

```
        padding-top: 20px;
    }

    ul li {
        border: solid 1px #ddd;
        padding: 10px;
        margin-bottom: -1px;
    }

    ul li:nth-of-type(odd) {
        background: #00b894;
    }
</style>

<body>
    <input type="text">
    <ul></ul>
</body>
<script>
    let obj = {
        words: new Set(),
        set keyword(word) {
            this.words.add(word);
        },
        show() {
            let ul = document.querySelector('ul');
            ul.innerHTML = '';
            this.words.forEach((item) => {
                ul.innerHTML += ('<li>' + item + '</li>');
            })
        }
    }

    document.querySelector('input').addEventListener('blur', function () {
        obj.keyword = this.value;
        obj.show();
    });
</script>

```

#交集

获取两个集合中共同存在的元素

```
let hd = new Set(['hdcms', 'houdunren']);
let cms = new Set(['后盾人', 'hdcms']);
let newSet = new Set(
 [...hd].filter(item => cms.has(item))
);
console.log(newSet); //{"hdcms"}

```

#差集

在集合a中出现但不在集合b中出现元素集合

```
let hd = new Set(['hdcms', 'houdunren']);
let cms = new Set(['后盾人', 'hdcms']);
let newSet = new Set(
 [...hd].filter(item => !cms.has(item))
);
console.log(newSet); //{"houdunren"}

```

#并集

将两个集合合并成一个新的集合，由于Set特性当然也不会产生重复元素。

```
let hd = new Set(['hdcms', 'houdunren']);
let cms = new Set(['后盾人', 'hdcms']);
let newSet = [...hd, ...cms];
console.log(newSet);

```

#WeakSet

WeakSet结构同样不会存储重复的值，它的成员必须只能是对象类型的值。

- 垃圾回收不考虑WeakSet，即被WeakSet引用时引用计数器不加一，所以对象不被引用时不管

WeakSet是否在使用都将删除

- 因为WeakSet 是弱引用，由于其他地方操作成员可能会不存在，所以不可以进行forEach( )遍历等操作
- 也是因为弱引用，WeakSet 结构没有keys( )，values( )，entries( )等方法和size属性
- 因为是弱引用所以当外部引用删除时，希望自动删除数据时使用

WeakMap

#声明定义

以下操作由于数据不是对象类型将产生错误

```
new WeakSet(["hdcms", "houdunren"]); //Invalid value used in weak set

new WeakSet("hdcms"); //Invalid value used in weak set

```

WeakSet的值必须为对象类型

```
new WeakSet([["hdcms"], ["houdunren"]]);

```

将DOM节点保存到`WeakSet`

```
document.querySelectorAll("button").forEach(item => Wset.add(item));

```

#基本操作

下面是WeakSet的常用指令

```
const hd = new WeakSet();
const arr = ["hdcms"];
//添加操作
hd.add(arr);
console.log(hd.has(arr));

//删除操作
hd.delete(arr);
```

```
9
10 //检索判断
11 console.log(hd.has(arr));
12
```

#垃圾回收

WeaSet保存的对象不会增加引用计数器，如果一个对象不被引用了会自动删除。

- 下例中的数组被

arr 引用了，引用计数器+1

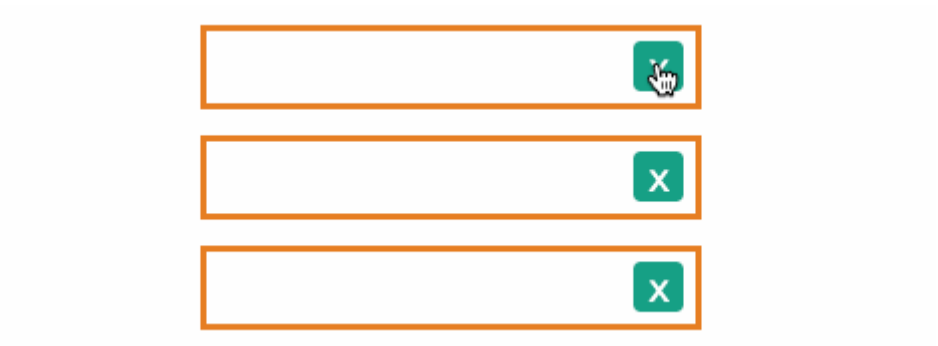
- 数据又添加到了 hd 的WeaSet中，引用计数还是1
- 当

arr 设置为null时，引用计数–1 此时对象引用为0

- 当垃圾回收时对象被删除，这时WakeSet也就没有记录了

```
1 const hd = new WeakSet();
2 let arr = ["hdcms"];
3 hd.add(arr);
4 console.log(hd.has(arr));
5
6 arr = null;
7 console.log(hd); //WeakSet {Array(1)}
8
9 setTimeout(() => {
10   console.log(hd); //WeakSet {}
11 }, 1000);
12
```

#案例操作



```
<style>
  * {
    padding: 0;
    margin: 0;
  }
  body {
    padding: 200px;
  }
  ul {
    list-style: none;
    display: flex;
    width: 200px;
    flex-direction: column;
  }
  li {
    height: 30px;
    border: solid 2px #e67e22;
    margin-bottom: 10px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding-left: 10px;
    color: #333;
    transition: 1s;
  }
  a {
    border-radius: 3px;
    width: 20px;
    height: 20px;
    text-decoration: none;
    text-align: center;
    background: #16a085;
    color: white;
    cursor: pointer;
    display: flex;
    justify-content: center;
    align-items: center;
    margin-right: 5px;
```

```
  }
  .remove {
    border: solid 2px #eee;
    opacity: 0.8;
    color: #eee;
  }
  .remove a {
    background: #eee;
  }
</style>

<body>
  <ul>
    <li>houdunren.com <a href="javascript:;">x</a></li>
    <li>hdcms.com <a href="javascript:;">x</a></li>
    <li>houdunwang.com <a href="javascript:;">x</a></li>
  </ul>
</body>

<script>
  class Todos {
    constructor() {}
    run() {
      this.items = document.querySelectorAll("ul>li");
      this.lists = new WeakSet();
      this.record();
      this.addEvent();
    }
    addEvent() {
      this.items.forEach(item => {
        item.querySelector("a").addEventListener("click", event => {
          //检测WakeSet中是否存在Li元素
          const parentElement = event.target.parentElement;
          if (!this.lists.has(parentElement)) {
            alert("已经删除此TODO");
          } else {
            //删除后从记录的WakeSet中移除
            parentElement.classList.add("remove");
            this.lists.delete(parentElement);
          }
```

```
        });
      });
    }
    record() {
      this.items.forEach(item => this.lists.add(item));
    }
  }
  new Todos().run();
</script>

```