Array

---->> ES5数组的方法:

push、 pop、 shift、 unshift、 splice(***)、 slice、 sort、 conact、 reverse、 join、indexOf(采用===比较)、 lastIndexOf

回调函数方法(他们的参数第一个是一个匿名函数function(value, index, self)，第二个是上下文context)-->>forEach、 mpa、 filter、every、 some、 < IE8不兼容，需要自己去扩展 >

---->> ES6数组的方法:: **find(callback)、findIndex(callback)、 includes(sarchValue)、 fill、 copyWithin、**

遍历数组: keys()、 values()、 entries()、 for循环、 for -- in循环、 优化for循环

静态方法: Array.from()、 Array.of()

---->> ES5数组的缺点:

1) 遍历数组的缺陷: for循环、 优化for循环、 for -- in循环 、 ES5中的forEach方法中的缺陷(使用return;不能结束forEach函数的运行，只是结束当前函数，而不是外面的forEach函数, for循环直接使用break指令就退出了)

2) 类数组转为数组问题 (ES5解决办法-->> []/Array.prototype.slice.call(likeAry), 封装函数likeAry() ES6解决办法 Array.from(ary)) 和 采用扩展符解决 [...arguments] 、 [...document.querySeletorAll('div')]

3) 创建数组构造函数的缺陷 new Array(3) -->>[, , ,] new Array('3') -->> ['3'] ES5直接字面量创建, ES6采用Array.of()方法创建行为统一了

4) 在指定的数组查找某个值,没有返回-1，在进行if判断需要在进行条件判断(if (ary.indexOf(val) > -1)),ES6采用了find

5) indexOf(searchValue)在查找元素的时候缺陷:(ES6引入了includes方法去解决了)

A1):一是不够语义化，它的含义是找到参数值的第一个出现位置，所以要去比较是否不等于-1，表达起来不够直观

A2): 它内部使用严格相等运算符（===）进行判断，这会导致对NaN的误判。 [NaN].indexOf(NaN); //-->> -1　[NaN].includes(NaN); //-->> true

indexOf() ---->> find()

ES6新增知识点:

-->> 扩展运算符(...) 将一个数组转为用逗号分隔的参数序列。[...[1,2,3,4]] ==>> [1,2,3,4]　类数组转为数组

-->> Array.from()

-->> Array.of()

-->> 数组实例的 copyWithin()

-->> 数组实例的 find() 和 findIndex()

-->> 数组实例的fill()

-->> 数组实例的 entries()，keys() 和 values()

-->> 数组实例的 includes()

-->> 数组的空位

1) 扩展运算符（spread）是三个点..., 它好比 rest 参数的逆运算，将一个数组转为用逗号分隔的参数序列。eg: ...[1, 2, 3] << 数组变为参数序列 >>

作用:

A): 获取数组里面的最大值和最小值(扩展运算符代替apply方法) return ES5 Math.max.apply(null, ary); return ES6 Math.max(...ary);

B): 将一个数组添加到另一个数组的尾部。 ES5 conact方法,和for循环 ES6 ary = [1,2] ary2 = [null, true]; -->> ary1.push(...ary2)

C): 数组合并 ES5采用conact [1, 2].conact(ary2) ES6 [1, 2, ...ary2] eg: [...ary1, ...ary2, ...ary3] 首先将数组转为参数列表，外面有[] ,所以转化了数组

D): 字符串转为数组 [...'hello'] -->>["h", "e", "l", "l", "o"] 获取字符串的长度 [...'hello'].length; -->>5

E): 任何 Iterator 接口的对象(有length属性),都可以用扩展运算符转为真正的数组。

F): Set和Map数据结构转为数组 [...(new Set([1,2,3]))]

G): 类数组转为数组 格式-->> [...类数组]

2) Array.from(array-like(数组、类数组、Set和Map数据), map方法(匿名函数), context(上下文this)): 用于将两类对象转为真正的数组-->> 类似数组的对象（array-like object）和可遍历（iterable）的对象（包括ES6新增的数据结构Set和Map）。

注意:

1) 只要是部署了Iterator接口的数据结构(有length属性)，Array.from都能将其转为数组。和扩展运算符（spread）一样 Array.from('hello world'); Array.from(new Set(['1', null, 10]))

2) 封装一下 const toArray = (()=> Array.from ? Array.from: (likeArray)=>[].slice.call(likeArray))();

总结类数组转为数组方法:

A1): ES5 借助数组原型上的call方法，this改为类数组 [].slice.call(likeAry);、 Array.prototype.call(likeAry);

A2): ES6 使用数组的静态方法from Array.from(likeAry) 使用数组的扩展运算符 [...likeAry]

3) Array.of();创建数组，解决new Array()创建数组的缺陷 -->> 用于将一组值，转换为数组。

eg: new Array(3)-->> [, , ,] new Array('3') -->> ['3']

Array.of(3) -->> [3] Array.of('3') -->> ['3']

4) 解决了 indexOf() 查找的时候，找不到NaN的情况

Array.prototype.find(callback); 于找出"第一个"符合条件的"数组成员"。它的参数是一个回调函数，所有数组成员依次执行该回调函数，直到找出第一个返回值为true的成员，然后返回该成员。如果没有符合条件的成员，则返回undefined。

Array.prototype.findIndex(callback); "第一个"符合条件的数组成员的"位置"，如果所有成员都不符合条件，则返回-1。

eg: [1, 2, 3, null, NaN].find((n)=> n > 1); // -->> 值2

eg: [1, 2, 3, null, NaN].findIndex((n)=> n > 1); // -->>下标1

比较的三种方法

1) val1 == val2 有隐式转换 类型不一样，首先将类型转为一样，在进行比较 ![] == [] 0 == false

2) val1 === val2 严格的类型和值比较

3) Object.is(val1, val2); 主要处理NaN这个特殊的比较，前俩个都返回false

5) Array.prototype.includes(searValue): 方法返回一个布尔值，表示某个数组是否包含给定的值，与字符串的includes方法类似

indexOf(searchValue)的缺陷

1):一是不够语义化，它的含义是找到参数值的第一个出现位置，所以要去比较是否不等于-1，表达起来不够直观

2): 它内部使用严格相等运算符（===）进行判断，这会导致对NaN的误判。 [NaN].indexOf(NaN); //-->> -1 [NaN].includes(NaN); //-->> true

6) Array.prototype.fill(填充的值[,填充的起始位置,结束位置]); 使用给定值，填充一个数组。 ['a', 'b', 'c'].fill(7); //-->>[7, 7, 7]

fill方法用于空数组的初始化非常方便。数组中已有的元素，会被全部抹去。

7) Array.copyWithin()

8)数组的便利

ES5:

1)for循环, 优化for循环

2) for -- in

3) forEach((item, index, self) => {}),this);

ES6:

下面的方法都用于遍历数组。它们都返回一个遍历器对象,可以用for...of循环进行遍历，唯一的区别是keys()是对键名的遍历、values()是对键值的遍历，entries()是对键值对的遍历。

Array.prototype.keys(); //-->> 键名的遍历(对于数组来说是下标)

Array.prototype.values(); //-->> 键值的遍历(对于数组来说是数组元素)

Array.prototype.entries();//-->> 键值对的遍历(对于数组来说是下标,数组元素)

eg: let ary = [1, 3, 4, null, 5];

```
for (let index of ary.keys() {console.log(index);}
for (let value of ary.values() {console.log(value);}
for (let [index ,value] of ary.entries() {console.log(index, value);}
```

案例:

1);

```
let ary = [1, 2, 3, 4, 5, 6];
ary.forEach(function (value, index, item) {
    if (value===2) {
```

```
            return;
        } else {
            console.log(value);
        }

    });

//    console.clear();
    for (let i=0,len =ary.length;i<len;i++) {
        if (ary[i] ===2) {
            break;
        } else {
            console.log(ary[i]);
        }
    }
```

接受三个参数，依次为当前的值、当前的位置和原数组。

1. find(function(value, index, self){}) 找出第一个符合条件的数组成员。找到返回， 找不到返回undefined
2. findIndex(function(value, index, self){})；第一个符合条件的数组成员的位置，如果所有成员都不符合条件，则返回-1。