1. 多重循环跳出break mutipleAnser([‘a’,’b’,’c’],[‘c’,’a’]) //[1,0,1]
2. //格式化多选题提交数据 选上的为1 不选上的为0
3. mutipleAnser(originList,resultList) {
4. let tempList = originList.slice()
5. one:
6. for(let i=0;i<tempList.length;i++) {
7. two:
8. for(let j=0;j<resultList.length;j++) {
9. if(resultList[j] === tempList[i]) {
10. tempList[i] = 1
11. break two;
12. }
13. }
14. }
15. for(let i=0;i<tempList.length;i++) {
16. if(tempList[i] !== 1) {
17. tempList[i] = 0
18. }
19. }
20. return tempList
21. }

2.async await语法让异步代码看起来更像同步代码，体现在async异步函数内部，从外部看async依旧是一个异步操作



//async函数返回的是promise对象，执行return 3相当于resolve(3)，执行throw ‘err’相当于reject(‘err’)

async function a() {

//await的结果为紧跟其后的promise对象resolve的值,若await后面的promise对象reject,async函数直接结束，async函数返回的promise为rejectded状态,值为await后面的promise对象reject的值,

let i = await new Promise(

(resolve,reject)=>{setTimeout(()=>{resolve(1)},6000)}

)

console.log(i);

console.log(2);

return 3;

}

a().then((res)=>{console.log(res)}) //1 2 3

3. await命令后面的 Promise 对象如果变为reject状态，则reject的参数会被catch方法的回调函数接收到。

    任何一个await语句后面的 Promise 对象变为reject状态，那么整个async函数都会中断执行（相当于抛出错误throw xxx）。



3. 前面已经说过，await命令后面的Promise对象，运行结果可能是rejected，所以最好把await命令放在try...catch代码块中。



4. 多个await命令后面的异步操作，如果不存在继发关系，最好让它们同时触发。



上面代码中，getFoo和getBar是两个独立的异步操作（即互不依赖），被写成继发关系。这样比较耗时，因为只有getFoo完成以后，才会执行getBar，完全可以让它们同时触发。



上面两种写法，getFoo和getBar都是同时触发，这样就会缩短程序的执行时间。

5. await命令只能用在async函数之中，如果用在普通函数，就会报错。



上面代码会报错，因为await用在普通函数之中了。但是，如果将forEach方法的参数改成async函数，也有问题。

正确的写法是采用for循环。



6.arguments对象

需要注意的是，虽然arguments很像数组，但它是一个对象。数组专有的方法（比如slice和forEach），不能在arguments对象上直接使用。

如果要让arguments对象使用数组方法，真正的解决方法是将arguments转为真正的数组。下面是两种常用的转换方法：slice方法和逐一填入新数组。



7.对象的深拷贝与浅拷贝

1）浅拷贝是拷贝地址 一方改变别的也改变 例如:

1.

var a = {c:1}

var b = a;

b.c = 2

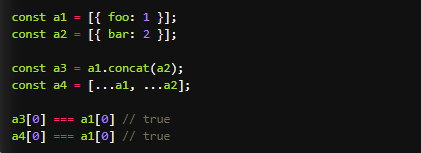
console.log(a.c) //2

2.

数组的concat方法中如果数组成员包括对象,concat方法返回当前数组的一个浅拷贝(slice方法同理) (若数组成员不是对象而是普通数据类型，则是深拷贝)



3.es6的扩展运算符…同理 (Array.from生成数组也是一样，var a = Array.from(b))



上面代码中，a3和a4是用两种不同方法合并而成的新数组，但是它们的成员都是对原数组成员的引用，这就是浅拷贝。如果修改了引用指向的值，会同步反映到新数组。

2)深拷贝 一方改变别的不变 如

1.利用map数组的函数和扩展运算符快速深拷贝

var data = [{a:1},{b:2}]

var cacheData = data.map(item => ({ ...item }))

2. JSON内置的方法(值为function的键不适用)(推荐)



3. Object的内置方法assign(拷贝源对象本身的属性、可枚举属性、Symbol属性到目标对象) (不包含继承的属性和不可枚举的属性) (属性的getter和setter也不会加上去)

注：对象嵌套层次过深，超过2层，就会出现浅拷贝的状况，如var a = {x:{y:1}};比如echarts组件的option对象；



4．递归实现



8.判断一个数据的类型

Object.prototype.toString可以看出一个值到底是什么类型。



利用这个特性，可以写出一个比typeof运算符更准确的类型判断函数。



9. 如果一个属性的enumerable为false，下面三个操作不会取到该属性。



另外，JSON.stringify方法会排除enumerable为false的属性，有时可以利用这一点。如果对象的 JSON 格式输出要排除某些属性，就可以把这些属性的enumerable设为false。

10.数组的reduce方法 求和或者取最大值都很方便

求和：



取最大值:

C:\Users\Administrator\AppData\Local\Temp\1595493340(1).png

C:\Users\Administrator\AppData\Local\Temp\1595832340(1).png

11.JSON.stringify的各种参数及作用

1）第一个参数





如果数组的成员是undefined、函数或 XML 对象，则这些值被转成null。



正则对象会被转成空对象。



JSON.stringify方法会忽略对象的不可遍历的属性。



2）第二个参数

这个类似白名单的数组，只对对象的属性有效，对数组无效。



\*第二个参数还可以是一个函数，用来更改JSON.stringify的返回值。



注意，这个处理函数是递归处理所有的键。



递归处理中，每一次处理的对象，都是前一次返回的值。

如果处理函数返回undefined或没有返回值，则该属性会被忽略。



3）第三个参数(作用不大)



12．获得对象的所有属性（不管是自身的还是继承的，也不管是否可枚举），可以使用下面的函数。(不包括symbol类型)



13.setTimetout



上面代码耗时2s



上面代码耗时3s

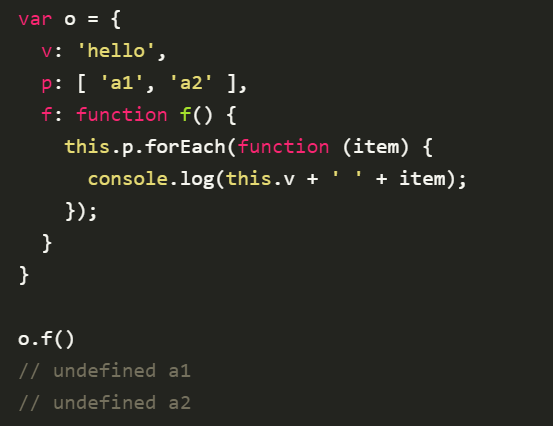
setTimeout的Debounce限流(开关也能做限流)



上面代码中，只要在2000毫秒之内，用户再次击键，就会取消上一次的定时器，然后再新建一个定时器。这样就保证了回调函数之间的调用间隔，至少是2000毫秒。

14.函数的this指向(总是指向某个对象，严格模式有可能是undefined)

1)普通函数的this在定义的时候没有固定一个值，看怎么使用，this会随之变化，若想固定则使用call、apply或bind手动绑定this



上面回调的this指向window，

First:使用第二个参数thisValue;

second:想解决可以使用另一个变量that存this,然后再回调使用；

third:使用bind手动绑定this的值

var a = function (item) {

        console.log(this.v + " " + item);

      };

      undefined;

      var o = {

        v: "hello",

        p: ["a1", "a2"],

        f: function f() {

          this.p.forEach(a.bind(o));

        },

      };

      o.f();

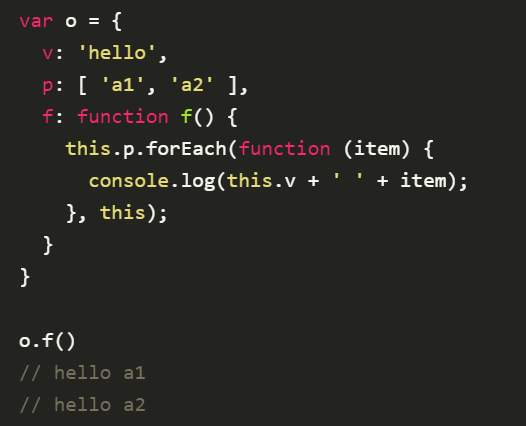
//hello a1

//hello a2

2)

this指向的固定化，并不是因为箭头函数内部有绑定this的机制，实际原因是箭头函数根本没有自己的this，导致内部的this就是外层代码块的this。正是因为它没有this，所以也就不能用作构造函数。

箭头函数导致this总是指向函数定义生效时所在的对象(this为箭头函数定义生效时函数的上一级作用域所在对象)，箭头函数的this在定义的时候就已经固定一个值，依据上下文，所以在forEach方法的回调中用箭头函数，即使使用第二个参数thisValue改变this,也无法改变已经固定的this值



var o = {

v: 'hello',

p: [ 'a1', 'a2' ],

f: function f() {

this.p.forEach( (item)=> {

console.log(this.v + ' ' + item);

});

}

}

o.f()

// hello a1

// hello a2

箭头函数没有自己的this，它的this会捕获其定义时所在的父作用域所处的环境作为自己的this; 因为箭头函数本身没有this，所以用bind/call/apply硬绑定没有任何作用







除了this，以下三个变量在箭头函数之中也是不存在的，指向外层函数的对应变量：arguments、super、new.target。

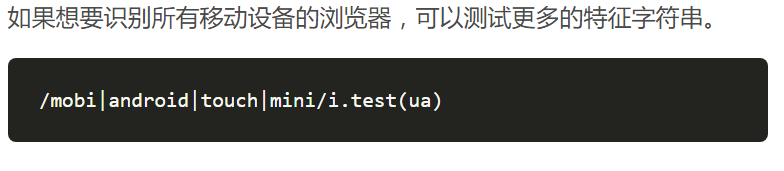


上面代码中，箭头函数内部的变量arguments，其实是函数foo的arguments变量。在箭头函数中推荐使用rest参数…arr来实现arguments的作用。

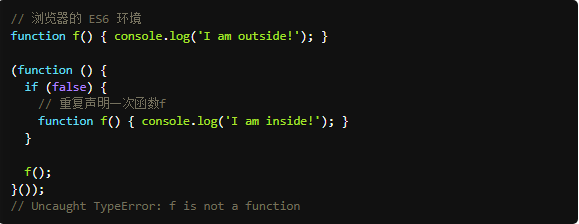
另外，由于箭头函数没有自己的this，所以当然也就不能用call()、apply()、bind()这些方法去改变this的指向。

15.识别移动设备浏览器

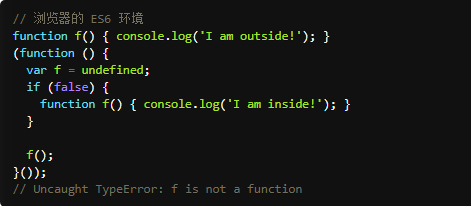




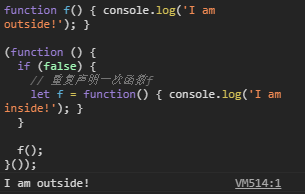
16. 考虑到环境导致的行为差异太大，应该避免在块级作用域内声明函数。如果确实需要，也应该写成函数表达式，而不是函数声明语句。（参考es6教程https://es6.ruanyifeng.com）



实际上：



改成：



17.获取url中的参数函数

//获取url的参数

// ()表示组匹配;^表示开头;$表示结束;|表示或;(^|&)表示以空字符串''或者&开头;[^&]\*表示任意字符有零个或者多个。

    function GetQueryString(name) {

        var reg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)");

        console.log(reg);

        var r = window.location.search.substr(1).match(reg);

        console.log(r);

        if (r != null) return unescape(r[2]);

        return null;

    };

18.函数传参默认值问题

1)参数为简单类型数据

function a(b=1) {

    console.log(b)

}

a() //1

2)实参为对象时(重点看es6版的数组结构赋值)

**ES5 版本**

function drawES5Chart(options) {

options = options === undefined ? {} : options;

var size = options.size === undefined ? 'big' : options.size;

var cords = options.cords === undefined ? { x: 0, y: 0 } : options.cords;

var radius = options.radius === undefined ? 25 : options.radius;

console.log(size, cords, radius);

// now finally do some chart drawing

}

drawES5Chart({

cords: { x: 18, y: 30 },

radius: 30

});

//big {x: 18, y: 30} 30

#### ES2015 版本（原理是es6的数组结构赋值）

function drawES2015Chart({size = 'big', cords = { x: 0, y: 0 }, radius = 25} = {})

{

  console.log(size, cords, radius);

  // do some chart drawing

}

drawES2015Chart({

  cords: { x: 18, y: 30 },

  radius: 30

});

// big {x: 18,y:30 } 30

在上面的 **drawES2015Chart** 的函数签名中，解构的左手边被分配给右手边的空对象字面值：{size = 'big', cords = {x: 0, y: 0}, radius = 25} = {}。你也可以在没有右侧分配的情况下编写函数。但是，如果你忽略了右边的赋值，那么函数会在被调用的时候查找至少一个被提供的参数，而在当前的形式下，你可以直接调用 **drawES2015Chart()** 而不提供任何参数。如果你希望能够在不提供任何参数的情况下调用该函数，则当前的设计非常有用，而另一种方法在您确保将对象传递给函数时非常有用。

19.数组去重

1)推荐数组filter方法，不改变原数组

var a = [1,2,3,1,2,4,5,6];

1. filter((ele,index,arr)=>{return arr.indexOf(ele)===index}) //[1,2,3,4,5,6]

2)利用Set数据结构和扩展运算符…

var a = [1,2,3,1,2,4,5,6];

var b = new Set(a)

[…b] //[1,2,3,4,5,6]

20.异步编程

1）setTimeout是在下一轮事件循环执行，Promise对象的then是在本轮事件末执行，所以then会比全局的setTimeout早执行(then里没有别的异步操作)

如：

setTimeout(()=>{console.log(1)},0);

new Promise((resolve)=>{resolve()}).then(()=>{console.log(2)});

console.log(3)

//3 2 1

2) 异步任务串行执行和并行执行

串行就例如setTimeout的嵌套; 并行例如setTimeout不嵌套

21.js垃圾回收机制(WeakMap)

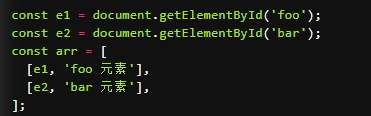
垃圾回收机制依赖引用计数，如果一个值的引用次数不为0，垃圾回收机制就不会释放这块内存。

但是这个过程不是时时的，因为其开销比较大，所以垃圾回收机器会按照固定的时间间隔周期性的执行。

1. WeakMap的键名所指向的对象是弱引用，不需要手动释放内存，键名引用的对象的外部引用计数为0的时候，当垃圾回收机制运行，WeakMap里的这对键值对就会消失，防止了内存泄漏

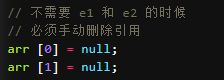
例子：

1）普通引用需要手动释放内存

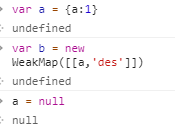


上面代码中，e1和e2是两个对象，我们通过arr数组对这两个对象添加一些文字说明。这就形成了arr对e1和e2的引用。

一旦不再需要这两个对象e1=null;e2=null，我们就必须手动删除这个引用，否则垃圾回收机制就不会释放e1和e2占用的内存。



2)WeakMap中的弱引用则不需要手动释放内存



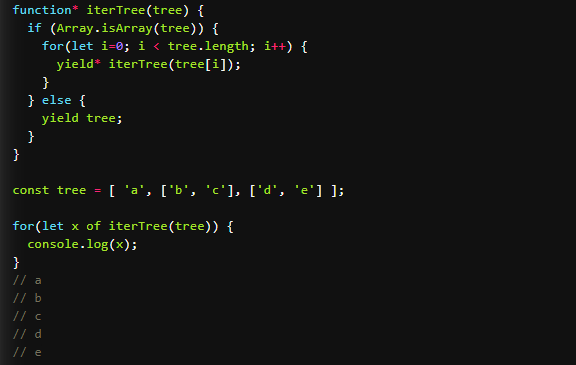
当对象{a:1}的引用计数为0，垃圾回收机制运行的时候b中对应的键值对就会消失。（Chrome 浏览器的 Dev Tools 的 Memory 面板，有一个垃圾桶的按钮，可以强制垃圾回收（garbage collect）。这个按钮也能用来观察 WeakMap 里面的引用是否消失。）

WeakMap 与 Map 在 API 上的区别主要是两个，一是没有遍历操作（即没有keys()、values()和entries()方法），也没有size属性。因为没有办法列出所有键名，某个键名是否存在完全不可预测，跟垃圾回收机制是否运行相关。这一刻可以取到键名，下一刻垃圾回收机制突然运行了，这个键名就没了，为了防止出现不确定性，就统一规定不能取到键名。二是无法清空，即不支持clear方法。因此，WeakMap只有四个方法可用：get()、set()、has()、delete()。

22.for of循环

但凡有iterator接口（Symbol.iterator遍历器方法）的数据接口都能被for of循环遍历；或者Symbol.iterator方法返回的遍历器对象也能被for of循环直接遍历，还有数据的keys()，values()，entries()方法返回的遍历器对象也能直接遍历。

23.利用yield\*命令递归取出嵌套数组的所用成员



由于扩展运算符...默认调用 Iterator 接口，所以上面这个函数也可以用于嵌套数组的平铺。



24.获取url的参数

//获取url的参数

    // ()表示组匹配;^表示开头;$表示结束;|表示或;(^|&)表示以空字符串''或者&开头;[^&]\*表示任意字符有零个或者多个。

    function GetQueryString(name) {

        var reg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)");

        var r = window.location.search.substr(1).match(reg);

        if (r != null) return unescape(r[2]);

        return null;

    };

若是vue项目，可以用this.$route.query.xxx来获取

25.事件冒泡和事件捕获和阻止默认事件

1）定义一个事件默认是冒泡模式，除非特别设置才是捕获模式



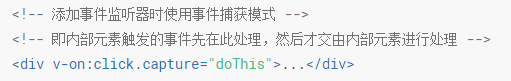


2）JS中默认先执行捕获的事件再执行冒泡的事件

3）阻止事件冒泡：event.stopPropagation()

4) 阻止默认事件：event.preventDefault()

5)vue中：







6）小程序中：bind和capture-bind的含义分别代表事件的冒泡阶段和捕获阶段；bind事件绑定不会阻止冒泡事件向上冒泡，catch事件绑定可以阻止冒泡事件向上冒泡。(capture-catch将中断捕获阶段和取消冒泡阶段)



注：将只触发handleTap2

26.js冒泡排序

//js最佳化冒泡排序算法  
var a = [34,1,4,67,3,2];  
for(var i=0;i<a.length-1;i++){  
 var bool = false;  
 for(var j=0;j<a.length-1-i;j++){  
 if(a[j]>a[j+1]){  
 var temp = a[j];  
 a[j] = a[j+1];  
 a[j+1] = temp;  
 bool = true;  
 }  
 }  
 if(!bool) break;  
}

27.安卓和苹果对js日期的支持比较(其中苹果不能用new Date('2015-2-1')，最好用new Date('2015/2/1')和new Date(2015,2,1)代替；当Date参数为数字时Month不会减一)

var d = new Date('2015-2-1')

var a = new Date('2015/2/1')

var b = new Date(2015,2,1)

安卓：

d.getFullYear(); //2015

d.getMonth(); //1

d.getDate(); //1

a.getFullYear(); //2015

a.getMonth(); //1

a.getDate(); //1

b.getFullYear(); //2015

b.getMonth(); //2

b.getDate(); //1

苹果：

d.getFullYear(); //NaN

d.getMonth(); //NaN

d.getDate(); // NaN

a.getFullYear(); //2015

a.getMonth(); //1

a.getDate(); //1

b.getFullYear(); //2015

b.getMonth(); //2

b.getDate(); //1

兼容性写法：

let fullDate = "1991-03-29 00:00:00";

let date = new Date(fullDate);

// In case its IOS, parse the fulldate parts and re-create the date object.

if(Number.isNaN(date.getMonth())) {

let arr = fullDate.split(/[- :]/);

date = new Date(arr[0], arr[1]-1, arr[2], arr[3], arr[4], arr[5]);

}

28.生成可被数组api循环的空数组用Array.apply(全是undefiend)

var arr1 = new Array(10) //长度为10，但元素全是空，即没有键，无法被forEach、map等循环

var arr2 = Array.apply(null,{length:10})//长度为10，元素全是undefiend，有键，能被forEach、map等循环（推荐）

29.常用正则用法

1）Object.prototype.toString.call({})的结果是”[object Object]”筛选出后面的构造函数Object,可以精准判断一个数据的类型

”[object Object]”.match(/\[object (.\*)\]/)[1] //’Object’

说明: ”[object Object]”.match(/\[object (.\*)\]/)结果是一个数组:[ ”[object Object]”,”Object”];\[和\]是转义成[];(.\*)是组匹配,表示有任意字符零个或多个(不包括换行符等),若改成(.\*?)非贪婪模式也可以,因为后面有个],取值会取完’Object’

2)贪婪模式和非贪婪模式和全局修饰符和组匹配和脱字符的区别

var a = "aabbcc"

贪婪：

a.match(/aab\*/) //["aabb"] 不能写成/ab\*/，会变成[“a”],可以写成/ab\*c/,结果为[“abbc”],原因不明，开头对不上最好加上结尾

非贪婪：

1. match(/aab\*?/) //[ "aa"]

全局修饰符：

1. match(/b/g) // ["b", "b"]

组匹配：

1. match(/a(.\*)c/) // ["aabbcc", "abbc"]

脱字符：

1. match(/[^ab]/) //[“c”] 匹配除了a和b之外的字符
2. match(/a[^a]\*c/) // ["abbcc"]

3）利用字符串生成正则，正则特殊字符转义处理

//正则有些符号需要转义如/2+/这种，需改成/2\+/

      let tempArr = this.searchResult.cname.split("");

      for (let i = 0; i < tempArr.length; i++) {

        if (

          tempArr[i] === "^" ||

          tempArr[i] === "." ||

          tempArr[i] === "[" ||

          tempArr[i] === "$" ||

          tempArr[i] === "(" ||

          tempArr[i] === ")" ||

          tempArr[i] === "|" ||

          tempArr[i] === "\*" ||

          tempArr[i] === "+" ||

          tempArr[i] === "?" ||

          tempArr[i] === "{" ||

          tempArr[i] === "\\"

        ) {

          //字符串拼接反斜杠需要转义

          tempArr[i] = "\\" + tempArr[i];

        }

      }

      let tempStr = tempArr.join("");

      //RegExp方法生成正则对象，转义需要使用两个斜杠，因为字符串内部会先转义一次。如(new RegExp('1\\+1')).test('1+1') //true

      //这里传的是变量，字符串已经转义了一次，所以转义成正则对象不再需要额外加\

      let reg = new RegExp(tempStr);

4）先行断言和后行言

1. x(?=y)称为先行断言（Positive look-ahead），x只有在y前面才匹配，y不会被计入返回结果。比如，要匹配后面跟着百分号的数字，可以写成/\d+(?=%)/。

“先行断言”中，括号里的部分是不会返回的。

var m = 'abc'.match(/b(?=c)/);

m // ["b"]

上面的代码使用了先行断言，b在c前面所以被匹配，但是括号对应的c不会被返回。

2.利用split把字符串按正则切割成数组，切割方式分成三部分，前中后

var test = 'aabbcc'

test.split(new RegExp(`(?<=bb)|(?=bb)`, 'i'))" //['aa','bb','cc']

30. 数组对象依据某个字段整合分组

//分组函数

export function groupByType(arr, param) {

    var map = {},

        dest = [];

    for (var i = 0; i < arr.length; i++) {

        var ai = arr[i];

        if (ai[param] && !map[ai[param]]) {

            dest.push({

                [param]: ai[param],

                data: [ai]

            });

            map[ai[param]] = ai;

        } else {

            for (var j = 0; j < dest.length; j++) {

                var dj = dest[j];

                if (dj[param] == ai[param]) {

                    dj.data.push(ai);

                    break;

                }

            }

        }

    }

    return dest;

}