# JavaScript

目录

[JavaScript 1](#_Toc335)

[ JavaScript 基础部分（ES） 2](#_Toc32468)

[1、 函数中断退出 2](#_Toc22038)

[2、 常用事件：（及事件监听） 3](#_Toc7115)

[3、 事件监听addEventListener/removeEventListener 3](#_Toc30574)

[4、 定时器 4](#_Toc28849)

[5、 a 元素阻止默认事件和取消冒泡停止蔓延 4](#_Toc26903)

[6、 冒泡单击事件的时机(加载顺序) 4](#_Toc31042)

[7、 转义字符的另一种简单方法 5](#_Toc10968)

[8、 动态生成页面元素的内容（数组动态拼接） 5](#_Toc17731)

[9、 Math的方法 6](#_Toc10285)

[10、 日期Date(); 7](#_Toc27101)

[11、 生成随机颜色： 8](#_Toc8884)

[12、 关联数组 8](#_Toc1770)

[13、 数组排序sort（双重排序） 9](#_Toc21706)

[14、 数组API，filter，slice、splice，join、concat 9](#_Toc18322)

[15、 数组reduce()方法-计算数组总和 12](#_Toc13871)

[16、 String: indexOf/search/match/替换replace/切割split 14](#_Toc16619)

[17、 正则表达式及验证 17](#_Toc23213)

[18、 正则—千位分隔符（数目用，号分隔千位） 20](#_Toc26232)

[19、 js将手机号中间四位变成\*号 20](#_Toc25364)

[20、 支付金额验证-正数、两位小数，无加号+ 21](#_Toc19384)

[21、 对象和构造函数 21](#_Toc31505)

[22、 EncodeURI/decodeURI地址栏传中文参数乱码解决 22](#_Toc25387)

[23、 Function 和 eval 22](#_Toc29342)

[ js核心/高级 24](#_Toc11890)

[ 本地储存localStorage/sessionStorage 26](#_Toc14372)

[24、 1. 保存数据到本地 26](#_Toc12402)

[25、 2. 从本地存储获取数据 26](#_Toc24714)

[26、 3. 本地存储中删除某个保存的数据 26](#_Toc14764)

[27、 4. 删除所有保存的数据 26](#_Toc5306)

[28、 5. 监听本地存储的变化 27](#_Toc10271)

[ 循环 27](#_Toc14447)

[29、 一维数组转二维 27](#_Toc16242)

[30、 Switch-case-break; 28](#_Toc31811)

[31、 栈、队列 push pop (js把十进制数转二进制) 28](#_Toc9015)

[32、 计算程序运行时间：循环计时 29](#_Toc23890)

[33、 判断是什么平台（Android或iPhone）的浏览器 30](#_Toc3209)

[ BOM和DOM部分 31](#_Toc26362)

[34、 屏幕/浏览器宽度、高度和适配 31](#_Toc23891)

[35、 横屏和竖屏 32](#_Toc28533)

[36、 弹出的窗口 33](#_Toc1785)

[37、 元素js – 页面滚动至顶部动画 33](#_Toc3680)

[38、 模态框（页面弹层）出现阻止底部body滚动 34](#_Toc10054)

[39、 7、窗口大小 34](#_Toc17521)

[40、 鼠标指针位置和页面滚动位置 35](#_Toc5577)

[41、 获得元素距离页面顶部的距离(offset.top) 35](#_Toc17897)

[42、 JS/JQ获取各种屏幕的高度和宽度 35](#_Toc11520)

[43、 eval(),js计算表达式 36](#_Toc9604)

[44、 解决苹果键盘收回body长度没有恢复问题（定位/input焦点错乱） 36](#_Toc13834)

[45、 滚动到溢出元素内指定位置（如菜单联动） 38](#_Toc19445)

[46、 js判断一个元素是否在页面中出现 38](#_Toc21171)

[ JS同步执行逻辑 39](#_Toc18476)

[47、 JS函数同步执行逻辑 39](#_Toc24855)

[48、 判断是什么平台（Android或iPhone）的浏览器 40](#_Toc9985)

[ ES6 41](#_Toc27956)

[49、 剩余参数语法 - function sum(...theArgs) - 省略号。 41](#_Toc13557)

[50、 export和import 41](#_Toc10266)

[ jQuery部分 42](#_Toc25375)

[51、 获得元素名称： 43](#_Toc22947)

[52、 父元素增加子元素/添加子元素-prepend和append 43](#_Toc4239)

[53、 弹出错误/自定义插件时判断是否引入jq文件 44](#_Toc10450)

[54、 jQuery事件处理的另一种写法： 44](#_Toc17315)

[55、 jq 按钮只能点一次（事件只处理一次） 44](#_Toc14414)

[56、 jQuery自带循环获取数组中每一个对象 44](#_Toc31523)

[57、 jq判断checkbox是否被选择的三种方法 44](#_Toc16276)

[58、 将canvas转换成微信可以识别的二维码图片 45](#_Toc15648)

[59、 Promise 46](#_Toc20179)

## JavaScript 基础部分（ES）

### 函数中断退出

btn.onclick=function(){

if(b>3){return;}; //如果b>3则退出函数执行，即style.color=“red”不执行

style.color=”red”

}

### 常用事件：（及事件监听）

1. onclick 单击
2. oncontextmenu 右键
3. ondblclick 双击
4. onfocus 获得焦点
5. onblur 失去焦点；
6. onchange select 当值发生改变时
7. onmouseover/out/enter/leave/hover/move/down/up

onmouseenter/leave ->(jq)进出子元素不会反复出发父元素事件处理函数

onmouseenter/leave -》jq简写🡪hover

1. onkewdown/up 键盘敲击
2. window.Onload 页面所有东西加载完成；注意：一个页面只能执行一次onload
3. window.onresize 事件：窗口或框架大小改变时触发
4. Domcontentloaded 仅DOM内容(html和js)加载完成——提前
   * + 如何: $(document).ready(function(){ ... ... })
     + 可简写为: $().ready(function(){ ... ... })
     + 还可简写: $(function(){ ... ... })
     + ES6: $(()=>{ ... ... })
     + 放在body结尾的script中的代码，默认就是DOM内容加载后执行.
     + 注意：一个页面可以增加多个ready
5. **jQuery常用绑定事件处理的另一种写法：**

**$(“.nav”).on(mouseenter,”a”,function(e){})**  //”.nav”是“a”的父元素；“a”是“.nav”的子元素，函数内可以不用e.target的冒泡判断=>if($(e.target).is(“a”)){…}

### 事件监听addEventListener/removeEventListener

网址：<https://www.cnblogs.com/dirkhe/p/6020779.html>

* addEventListener语法

element.addEventListener(type，handler，false/true)

type:事件类型

handler:事件执行触发的函数

false/true:false为冒泡/ture为捕获，参数是true，表示在捕获阶段调用事件处理程序；如果是false，表示在冒泡阶段调用事件处理程序。

事件捕获：父级元素先触发，子集元素后触发；

事件冒泡：子集元素先触发，父级元素后触发；

一般的绑定事件，都是采用冒泡方式，也就是使用false

* **removeEventListener语法**

element.removeEventListener(type，handler，false/true)

参数值含义和上述一样。

### 定时器

周期性定时器:

* + 启动定时器: var timer=setInterval(task,间隔ms)
    - 强调: task是回调函数，不止执行一次，不加括号！
  + 停止定时器: clearInterval(timer)

1. **一次性定时器:** 
   * 启动定时器: var timer=setTimeout(task,等待ms)
   * 停止定时器: clearTimeout(timer)
     + 指在执行之前，停止等待，取消执行。

### a 元素阻止默认事件和取消冒泡停止蔓延

1、e.prenventDefault();

2、e.stopPropagation();

### 冒泡单击事件的时机(加载顺序)

<**body**>  
 <**h1**>添加/删除节点</**h1**>  
 <**button id="add-block"**>添加区块</**button**>  
 <**div class="container"**></**div**>  
 <**script src="js/jquery-1.11.3.js"**></**script**>  
 <**script**>  
 **var *$container***=**$**(**".container"**);  
 **$**(**"#add-block"**).click(()=>{  
 **var *$block***=**$**(`<div **class**=**"block"**><span **class**=**"close"**>×</span></div>`);  
 *//修改$block的背景色* **var *r***= parseInt(**Math**.random()\*256);  
 **var *g***= parseInt(**Math**.random()\*256);  
 **var *b***= parseInt(**Math**.random()\*256);  
 ***$block***.css(**"background"**, `rgb(${***r***},${***g***},${***b***})`);  
 ***$container***.append(***$block***).prepend(***$block***);

//给子元素.close添加点击事件，如果父元素container内容没有创建则无法添加单机事件

*//$(".container>.block .close").click(e=>{$(e.target).parent().remove()})* });

//最好在父元素.container绑定单机事件 ，冒泡到每个子元素.close

***$container***.delegate(**".close"**,**"click"**,**function**(){**$**(**this**).parent().remove();});  
 </**script**>  
 </**body**>

### 转义字符的另一种简单方法

var pathname=String.raw`c:\windows\temp\new\folder\index.html`

console.log(pathname); // c:\windows\temp\new\folder\index.html

raw:原生

### 动态生成页面元素的内容（数组动态拼接）

**id.innerHTML=**

**"<ANY>"+arr.join("</ANY><ANY>")+"</ANY>"**

//从服务器端收到省份列表数组

var provs= ["北京市","河北省","河南省","山东省","陕西省"];

var html="<option>-请选择-</option><option>"+

provs.join("</option><option>") +"</option>";

//设置selProvs元素的内容为html

selProvs.innerHTML=html;

### Math的方法

1. Math.ceil(num) //上取整: 只要超过，就取下一个整数;
2. Math.floor(num) //下取整: 舍弃小数部分

vs parseInt(str): 1. parseInt可去掉结尾非数字字符floor不行！

2. parseInt要求传入字符串,floor要求数字

1. Math.round(num) // 四舍五入取整

**vs .toFixed(d):** 1. round只能取整, toFixed 可按任意小数位数四舍五入

2. round返回number，可直接用于计算toFixed 返回string，不能直接用于+计算

1. 自定义round(转换任意小数位数四舍五入)

乘方和开平方:

乘方: Math.pow(底数,幂); 比如: Math.pow(2,3)=8

开平方: Math.sqrt(num)

5、 es6:n次方console.log(12\*\*5); //12的5次方=248832

**例子：**

function round(num,d){

//num\* 10的d次方

num\*=Math.pow(10,d);

//对num四舍五入取整

num=Math.round(num);

//num/ 10的d次方

num/=Math.pow(10,d);

return num;//返回num

}

var n=345.678;

console.log(

round(n,2), //345.68

round(n,0), //346

round(n,-1) //350

);

1. Math.max(值1,值2,...)
2. Math.min(值1,值2,...) //max和min不支持查找数组中的最大值/最小值
3. Math.max.apply(null,arr)// apply可打散数组类型的参数为单个值，再传递给调用国的函数
4. **随机数:**

1. Math.random(): 单独使用返回0~1之间的一个小数 // var r=Math.random();

2. 在min~max之间取随机整数:

parseInt(Math.random()\*(max-min+1)+min)

var r=parseInt(Math.random()\*33+1); //1~33之间生成随机整数r

3. 如果从0~max之间取随机整数:

parseInt(Math.random()\*(max +1))

### 日期Date();

* 1. **javascript  时间戳 转换成24 小时格式**

**（默认是12小时格式：2017/9/1 下午6:16:43）**

var timeTrans = new Date(parseInt(1459408942) \* 1000);

console.log(timeTrans.toLocaleString('chinese',{hour12:false}));

*// var d=new Date(1504261003000).toLocaleString();* **var *d***=**new** Date(1504261003000).toLocaleString(**'chines'**,{**hour12**:**false**});*// 2017/9/1 18:16:43* **console**.log(***d***)

* 1. **php  时间戳 24 小时转换**

date("Y/m/d H:i:s",1446634507 ) //2015/11/04 18:55:77

**js获取某月的天数**

**方法一：**

new Date(2019,2,0).getDate(); // 28

js中的月份是0-11，

天数是：1-31.

如果传入的参数中 天  的值是0，js会返回上一个月的最后一天。

这样就能得到某月的天数

**方法二：通过闰年判断获取**

function getDays(year, month) {

let days = [31,28,31,30,31,30,31,30,30,31,30,31]

if ( (year % 4 ===0) && (year % 100 !==0 || year % 400 ===0) ) {

days[1] = 29

}  
　　return days[month]

}

**获取某天凌晨时间戳**

new ***Date***().setHours(0,0,0,0), // 今天凌晨的时间戳

**new Dat()e兼容苹果**

**const** outTime = **new** Date(**'2019-09-09'**.replace(/'-'/, '/')).getTime()

### 生成随机颜色：

**function** *rc*(){  
 **var** r=**Math**.floor(**Math**.random()\*256);  
 **var** g=**Math**.floor(**Math**.random()\*256);  
 **var** b=**Math**.floor(**Math**.random()\*256);  
 **return** `rgb(${r},${g},${b})`; **//关联**  
 }  
 **console**.log(*rc*()) //**rgb(91,103,187)  
$**.ajax({  
 **type**:**"GET"**,  
 **url**:**"/salesdata"**,  
 success:**function**(data){  
 **var** ctx=c2.getContext(**"2d"**);  
 **var** x=50;  
 **for**(**var** obj of data){  
 **var** str=obj.**label**;  
 ctx.**fillStyle**=*rc*();**//关联** ctx.fillRect(x,500,50,-obj.value);  
 ctx.**font**=**"20px SimHei"**;  
 ctx.**fillStyle**=**"black"**;  
 ctx.fillText(str,x,500-obj.value-20);  
 x+=100;  
 }  
 },  
 error:**function**(){  
 alert(**"网络故障，请检查"**);  
 }  
})  
</**script**>

### 关联数组

什么是: 可自定义下标名称的数组

强调: length=0，失效

访问元素: 数组名["自定义下标名称"]

**遍历:** for(var key in hash){//in会依次取出hash中每个key

key //下标名称

hash[key] //当前正在遍历的元素内容

}

强调: hash[key] 不要加""， 因为key是个变量，每循环依次都会变化

var fbb=[];

fbb["name"]="范冰冰";//杨幂

fbb["math"]=89; //81

fbb["chs"]=65; //59

fbb["eng"]=95; //89

console.dir(fbb);

console.log(fbb["chs"]);

fbb["chs"]+=10;

console.log(fbb["chs"]);

for(var key in fbb){

console.log(key+":"+fbb[key]);

}

### 数组排序sort（双重排序）

// 先按数字排序，如果数字相同则按中文排序

arr.sort((a,b) =>{

//sort为数字  
 **let** sort1 = a.sort;  
 **let** sort2 = b.sort;

//如果sort相等则按中文首字母排序  
 **if** (sort1 === sort2) {  
 **return** a.goodName.localeCompare(b.goodsName);  
 }  
 **return** sort1 - sort2  
})

### 数组API，filter，slice、splice，join、concat

* **数据过滤：**

**Var arr = [1,5,3,4,9]**

**Var arr = arr.filter((n) => {**

**Return n > 5**

**})**

**Console.log(arr) // [9]**

* **转字符串：var str=String(arr); 将arr的每个元素都转为字符串，用逗号相连.（拍照）**
* **var str=arr.join("自定义的连接符")**

*//无缝拼接***var *chars***=[**"H"**,**"e"**,**"l"**,**"l"**,**"o"**];  
***console***.log(***chars***.join(**""**)); //Hello

* + - **动态生成页面元素的内容**

**id.innerHTML=**

**"<ANY>"+arr.join("</ANY><ANY>")+"</ANY>"**

//从服务器端收到省份列表数组

var provs= ["北京市","河北省","河南省","山东省","陕西省"];

var html="<option>-请选择-</option><option>"+

provs.join("</option><option>") +"</option>";

//设置selProvs元素的内容为html

selProvs.innerHTML=html;

* **拼接: 将其他数组或元素，拼接到当前数组的末尾，组成新数组**

var arr3=arr1.concat(值1,值2,...,arr2,...)

**var *arr1***=[1,2,3];  
**var *arr2***=[12,23,34];  
**var *arr3***=***arr1***.concat(4,5,***arr2***,6);  
***console***.log(String(***arr1***));  
***console***.log(***arr3***); // [1, 2, 3, 4, 5, 12, 23, 34, 6]  
***console***.dir(***arr3***);

* **选取: 复制出原数组中指定位置的几个元素值，组成新数组.**

**var sub=arr.slice(starti,endi+1);**

强调: 凡是两个参数都是下标位置的函数，都含头不含尾.

1. 可用负数参数直接代表倒数第n个位置:

arr.slice(starti,-n+1); 截取到倒数第n个位置

原理: arr.slice(starti,arr.length-n+1);

2. 省略第二个参数，表示到结尾！

3. 两个参数都省: 表示从头到尾完整复制整个数组

**var *arr***=[0,1,2,3,4,5,6];  
 *//0 1 2 3 4 5 6  
 // -3 2 1***var *sub***=*//arr.slice(2,4+1);  
 //倒数第3个  
 //arr.slice(2,arr.length-3+1);* ***arr***.slice(2,-3+1);  
 **var *c***=***arr***.slice(2,5)  
***console***.log(String(***arr***)); //0,1,2,3,4,5,6  
***console***.log(String(***sub***));// 2,3,4  
***sub***=***arr***.slice(3); ***console***.log(String(***sub***)); //3,4,5,6

* + **修改数组: 插入，删除，替换**

强调: 直接修改原数组

1. 删除: arr.splice(starti,n); 删除arr中starti开始的n个元素

强调: 不考虑含头不含尾,因为第二个参数不是下标

简写: 1. 省略第二个参数: 一直删到结尾

2. 支持负数参数: 表示倒数第n个位置

其实splice有返回值: 返回被删除元素组成的临时新数组

var deletes=arr.splice(starti,n);

强调: 即使仅删除一个元素，也会放在数组中返回。

2. 插入: arr.splice(starti,0,值1,值2,...)

原插入位置及其之后的元素，依次向后顺移

vs concat: 1. splice可在任意位置插入新元素,concat只能在末尾拼接

2. splice直接修改原数组，concat不修改原数组，返回新数组

3. splice不支持打散数组参数，concat 支持大散数组参数为单个元素

**var *arr***=[0,1,2,3];  
**var *arr2***=[12,23,34];  
***arr***.splice(2,0,4,5,***arr2***,6);  
***console***.log(String(***arr***)); // 0,1,4,5,12,23,34,6,2,3  
***console***.dir(***arr***);

3. 替换: 先删除元素，再在原位置插入新元素

arr.splice(starti,n,值1,值2,...); 先删除n个，再插入新值

强调: 删除的元素个数和插入的新元素个数不必相同！

**var *arr***=[0,1,2,3,4,5];  
*//先删除2个元素，再插入3个****arr***.splice(2,2,12,23,34);  
***console***.log(String(***arr***));// 0,1,12,23,34,4,5

* + **翻转: arr.reverse();**
  + **排序(原生使用冒泡排序)**

**arr.sort();**

arr.sort(function(a,b){return a-b;})

//升序改降序: 只要颠倒比较器函数返回值的正负号

### 数组reduce()方法-计算数组总和

**let** numbers = [1, 2, 3, 4]  
  
**let** he = numbers.reduce((total, num) => {  
 **return** total + num  
})  
console.log(he) // 10

arr.reduce(callback,[initialValue])

callback （执行数组中每个值的函数，包含四个参数）

1、previousValue （上一次调用回调返回的值，或者是提供的初始值（initialValue））

2、currentValue （数组中当前被处理的元素）

3、index （当前元素在数组中的索引）

4、array （调用 reduce 的数组）

initialValue （作为第一次调用 callback 的第一个参数。）

var arr = [1, 2, 3, 4];var sum = arr.reduce(function(prev, cur, index, arr) {

console.log(prev, cur, index);

return prev + cur;}，0) //注意这里设置了初始值

console.log(arr, sum);

1. 计算数组中每个元素出现的次数

let names = ['Alice', 'Bob', 'Tiff', 'Bruce', 'Alice'];

let nameNum = names.reduce((pre,cur)=>{

if(cur in pre){

pre[cur]++

}else{

pre[cur] = 1

}

return pre},{})

console.log(nameNum); //{Alice: 2, Bob: 1, Tiff: 1, Bruce: 1}

1. 数组去重

let arr = [1,2,3,4,4,1]let newArr = arr.reduce((pre,cur)=>{

if(!pre.includes(cur)){

return pre.concat(cur)

}else{

return pre

}},[])

console.log(newArr);// [1, 2, 3, 4]

1. 将二维数组转化为一维

let arr = [[0, 1], [2, 3], [4, 5]]let newArr = arr.reduce((pre,cur)=>{

return pre.concat(cur)},[])

console.log(newArr); // [0, 1, 2, 3, 4, 5]

1. 对象里的属性求和

var result = [

{

subject: 'math',

score: 10

},

{

subject: 'chinese',

score: 20

},

{

subject: 'english',

score: 30

}];

var sum = result.reduce(function(prev, cur) {

return cur.score + prev;}, 0);

console.log(sum) //60

### String: indexOf/search/match/替换replace/切割split

* + **大小写转换：**

str.toUpperCase() 转大写

str.toLowerCase() 转小写

* + **获得指定位置的字符:**

str.charAt(i) -> str[i]

获得指定位置字符的unicode号:

str.charCodeAt(i) 获得i位置字符的unicode号

将一个unicode号，反向转回字:

String.fromCharCode(code)

选取指定位置的子字符串:

str.substring() => str.slice()

唯一差别: substring不支持负数参数

解决: 可用str.length-n，表示倒数第n的位置

从当前位置开始，取n个

str.substr(starti,n)

* + **查找关键词:**

1. 查找一个固定的关键词的位置: （每次只能找一个）

var i=str.indexOf("关键词",fromi)

简写: 省略fromi，默认从头开始找第一个关键词

**返回值:** 返回找到的下一个关键词的下标位置i

如果没找到，返回-1

**在str中找最后一个关键词的位置**

var i=str.lastIndexOf("关键词")

2. 判断是否包含某类关键词:

var i=str.search(/正则表达式/i);

在str中找第一个符合正则表达式要求的关键词的位置

如果没找到，返回-1

问题: 默认，所有正则区分大小写！

解决: 如果希望不区分大小写时，在第二个/后加i

3. 获得所有关键词的内容:

var kwords=str.match(/正则表达式/ig);

在str中找所有符合正则表达式要求的敏感词，保存在数组中返回。

如果没找到，返回null!

问题: 默认，正则表达式都只匹配第一个关键词

解决: 只要希望匹配所有，都必须加g

问题: 只能返回关键词内容，无法返回每个关键词位置

**即找所有关键词内容，又找所有关键词位置: (用正则：var arr=reg.exec(str);**

**方法)**

**var *str***=**"明明喜欢我，却不告诉我。别理我，我想静静。静静是谁?你先告诉我明明是谁？"**;  
**var *kwords***=***str***.match(/明明|静静/g);  
***console***.log(String(***kwords***)); //明明,静静,静静,明明

* + **替换：**

1. 简单替换: 将所有敏感词，都替换为统一的新内容

str=str.replace(/正则/ig,"新值");

强调: 1. 必须加g, 替换所有

2. 必须用=保存回去，否则无法获得替换结果

例子：手机号码中间加\*\*\*

**let** str = '12345678910';  
**let** s = str.substr(3, 4);  
**let** e = str.replace(s, '\*\*\*\*')  
console.log(e); //123\*\*\*\*8910

2. 高级替换: 根据本次找到的敏感词不同，动态选择不同的新内容替换。

str=str.replace(/正则/ig,function(kw){

kw //自动获得本次找到的关键词内容

return //根据kw的不同选择替换不同的值

})

<**script**>  
 **var *str***=**"那天，我去了她家。我说: 我草，你家真大！她说想和我去草地野餐。我草草的收拾了草地。她说：我去去就来"**;  
 *//我去，我草，我去草，我草草，我去去* ***str***=***str***.replace(  
 /我[去草]{1,2}/g,  
 **function**(kw){  
 **return** kw.**length**==2?**"\*\*"**:**"\*\*\*"**;  
 }  
 );  
 ***console***.log(***str***);  
</**script**>

* + **切割split：**

1. 简单切割: 按一个固定的字符/字符串切割

var subs=str.split("切割符")

固定套路: 将字符串打散为字符数组: var chars=str.split("");

将来如果希望对字符串，使用数组API时，都要先打散。

2. 复杂切割: 切割符不是固定的，但有一定规律

var subs=str.split(/正则表达式/i);

**var *url***=**"http://item.taobao.com/item.html?a=1&c=2&d=xxx&e"**;  
**function** *getData*(url){  
 **var** result={};  
 **var** temp=url.split(**"?"**)[1].split(**"&"**);  
 **console**.log(temp);  
 **for** (**var** i **in** temp){  
 **var** s=temp[i].split(**"="**);  
 **console**.log(s[1]);  
 result[s[0]]=s[1];*//xx[]为对象属性添加值* }  
 **return** result;  
}  
**console**.dir(*getData*(***url***))

### 正则表达式及验证

* 1. /关键词/ 关键词本身
  2. 字符集

1位数字 [0-9]

1位大写字母[A-Z]

1位小写字母[a-z]

1位字母[A-Za-z]

1位字母或数字[A-Za-z0-9]

1位汉字[\u4e00-\u9fa5]

* 1. 预定于字符集：

\d 一位数字

\w 一位字母/数字或\_

\s 一位空字符: 包括空格，Tab...

. 一位任意字符——通配符

强调: 规则必须和预定义字符集的规则完全匹配时，才能使用预定义字符集。否则，使用普通字符集。

\b 匹配单词

\babc\b匹配的是"abc"  
\sabc\s匹配的不是"abc", 前后还带空格" abc "

**正则表达式中/i,/g,/ig,/gi,/m的区别和含义**

**/i (忽略大小写)  
/g (全文查找出现的所有匹配字符)  
/m (多行查找)  
/gi(全文查找、忽略大小写)  
/ig(全文查找、忽略大小写)**

**let** str="ran is not a good man";  
**let** n = 5;  
str = str.replace(/\b[a-z]+\b/ig, **function** (kw) {  
 **return** kw + n++  
})  
console.log(str); // ran5 is6 not7 a8 good9 man10

* 1. 量词

**1. 有明确数量边界: 3组**

字符集{m,n} 至少m次，最多n次

字符集{m,} 至少m次，多了不限！

字符集{m} 必须m次

**2. 没有明确边界: 3个**

字符集\* 可有可无，多了不限

字符集+ 至少一个，多个不限

字符集? 可有可无，最多一个

问题: 默认，一个量词只修改相邻的前一个字符集

* 1. 选择和分组:

选择: 或 | 规则1|规则2 只要匹配任意一个规则即可

分组: (多个规则) 将多个规则分为一组

为什么: 一个量词默认值修改一个字符集

何时: 只要希望一个量词修改多个字符集时

* 1. 指定匹配位置:

开头: ^ 比如: ^\s+ 匹配开头位置的空字符

结尾: $ 比如: \s+$ 匹配结束位置的空字符

单词边界: \b 包括: ^ $ 空格 标点符号

何时: 只要要求匹配一个单词时

**例子：**

**身份证号:** 15位数字 2位数字 1位数字或Xx 可有可无,最多一次

\d{15}(\d\d[0-9Xx])?

手机号: +86或0086 可有可无，最多一次空字符 可有可无，多了不限

1在3,4,5,7,8中选一个9位数字

(\+86|0086)?\s\*1[34578]\d{9}

微 信 wei xin w x

(微|w(ei)?)\s\*(信|x(in)?)

**邮箱：**var unReg=/^[a-zA-Z0-9]{3,18}[@]{1}[a-z0-9.]{5,18}$/ig;

**金钱验证**：<script type="text/javascript">

     var reg = /(^[1-9]([0-9]+)?(\.[0-9]{1,2})?$)|(^(0){1}$)|(^[0-9]\.[0-9]([0-9])?$)/;

     var money = "520.100";

     //000 错

     //0 对

     //0. 错

     //0.0 对

     //050 错

     //00050.12错

     //70.1 对

     //70.11 对

     //70.111错

     //500 正确

     if (reg.test(money)) {

          alert("正确~");

     }else{

          alert("有误~");

     };

 </script>

**只含有汉字、数字、字母、下划线，不能以下划线开头和结尾:**

/^(?!\_)(?!.\*?\_$)[a-zA-Z0-9\_\u4e00-\u9fa5]+$/

**护照通行证验证**

*/\*\*  
 \* 护照验证  
 \* 规则： G + 8位数字, P + 7位数字, S/D + 7或8位数字,等  
 \* 例： G12345678, P1234567  
 \*/*// let reg1 = /^([a-zA-z]|[0-9]){5,17}$/  
*/\*\*  
 \* 台湾居民来往大陆通行证  
 \* 规则： 新版8位或18位数字 或 旧版9位数字 + 英文字母 或 8位数字 + 英文字母  
 \* 样本： 12345678  
 \*/*// let reg2 = /^\\d{8}|^[a-zA-Z0-9]{10}|^[a-zA-Z0-9]{9}|^\\d{18}$/  
*/\*\*  
 \* 港澳居民来往内地通行证  
 \* 规则： H/M + 10位或6位数字  
 \* 例：H1234567890  
 \*/*// let reg3 = /^([A-Z]\\d{6,10}(\\(\\w{1}\\))?)$/

* **验证: 正则表达式对象RegExp:**

创建: 2种:

1. 创建一个固定不变的正则表达式:

var reg=/正则表达式/ig;

字符冲突: 如果正则中出现/，要转为\/

2. 如果需要动态生成正则表达式:

var reg=new RegExp("正则",["ig"]);

字符冲突: 如果正则中出现引号，转为\"正则中正常的\ 都要转为\\

例子：

<script>

var names=["明月","白鹭","青天","紫烟"];

var str="窗前明月光,一行白鹭上青天,日照香炉生紫烟";

var reg=new RegExp(names.join("|"),"g");

var kwds=str.match(reg);

console.log(kwds);

var reg=new RegExp("\\d{6}");

</script>

* + API:

**1. 验证: var bool=reg.test(str);**

用reg验证str是否符合规则的要求;返回bool类型

问题: 正则默认只要找到部分匹配就返回true

解决: 要求从头到尾必须完整匹配

今后，凡是验证，必须同时前加^后加$

**2. 查找: 即找位置，又找内容**

**var arr=reg.exec(str);**

在str中找下一个满足reg要求的关键词的内容和位置

返回值: 本次找到的关键词的内容和位置

数组: 0号房间保存本次找到的关键词内容

index的房间保存本次找到的关键词位置

如果找不到，返回null

强调: 反复调用，会自动向后查找下一个！

<script>

var str="那天，我去了她家。我说: 我草，你家真大！她说想和我去草地野餐。我草草的收拾了草地。她说：我去去就来";

var reg=/我[去草]{1,2}/g;

do{

var arr=reg.exec(str);

if(arr!==null){

console.log(

"在位置"+arr["index"]

+"发现敏感词"+arr[0]

+",下次从"+reg.lastIndex+"开始"

);

}else{ break; }

}while(true);

</script>

### 正则—千位分隔符（数目用，号分隔千位）

string.replace(/\B(?=(\d{3})+$)/g, ',')

### [js将手机号中间四位变成\*号](https://www.cnblogs.com/chengkun101/p/7879522.html)

**方法一、利用数组splice，split，join方法**

var tel = 18810399133;

tel = "" + tel;var ary = tel.split("");

ary.splice(3,4,"\*\*\*\*");var tel1=ary.join("");

console.log(tel1);

**方法二、利用字符串的substr方法**

var tel = 18810399133;

tel = "" + tel;var tel1 = tel.substr(0,3) + "\*\*\*\*" + tel.substr(7)

console.log(tel1);

**方法三、利用字符串substring方法**

var tel = 18810399133;

tel = "" + tel;var tel1 =tel.replace(tel.substring(3,7), "\*\*\*\*")

console.log(tel1);

方法四、利用正则

var tel = 18810399133;

tel = "" + tel;var reg=/(\d{3})\d{4}(\d{4})/;var tel1 = tel.replace(reg, "$1\*\*\*\*$2")

console.log(tel1);

### 支付金额验证-正数、两位小数，无加号+

<input type="number" class="input-price"  
 v-model="payMoney" placeholder="请输入金额"  
 @keyup="regFloat"  
/>

// JS

// 验证输入框是否为两位小数  
regFloat(event) {  
 **if** (event.key === '-' || event.key === '+') event.target.value = ''  
 event.target.value = event.target.value.match(/^\d\*(\.?\d{0,2})/g)[0] || ''  
},

### 对象和构造函数

Object.assign()解释整理——复制、合并对象

复制：

var obj = { a: 1 };

var copy = Object.assign({}, obj);

console.log(copy); // { a: 1 }

合并对象：

var o1 = { a: 1 };

var o2 = { b: 2 };

var o3 = { c: 3 };

var obj = Object.assign(o1, o2, o3);

console.log(obj); // { a: 1, b: 2, c: 3 }

console.log(o1); // { a: 1, b: 2, c: 3 }, 注意目标对象自身也会改变。

合并具有相同属性的对象：

var o1 = { a: 1, b: 1, c: 1 };

var o2 = { b: 2, c: 2 };

var o3 = { c: 3 };

var obj = Object.assign({}, o1, o2, o3);

console.log(obj); // { a: 1, b: 2, c: 3 }

更多内容：<http://blog.csdn.net/wang252949/article/details/79106160>

### EncodeURI/decodeURI地址栏传中文参数乱码解决

* Html：
  + a.html:

// 地址参数编码  
**let** word = '微信';  
location.href = 'history.html?word=' + word

* + b.html

**let** word = location.href.split('=')[1];  
console.log(decodeURI(word)); //微信

* java - jsp:
  + c.jsp:

// 地址参数编码  
**let** titleName = encodeURI(encodeURI('拒绝发货'));

* + d.jsp

// 地址参数解码  
**let** title = decodeURI(titleName); //拒绝发货

### Function 和 eval

eval(String) // 可以执行字符串内容

<script type="text/javascript">

eval("x=10;y=20;document.write(x\*y)");

document.write(eval("2+2"));

var x=10;

document.write(eval(x+17));

</script>

输出为：200 4 27输出为：200 4 27

var jsstring="var un=1;console.log(un);";

eval(jsstring);

jsstring="var deux=2;console.log(deux);";

new Function(jsstring)();

jsstring="var trois=3;console.log(trois);";

(function () {

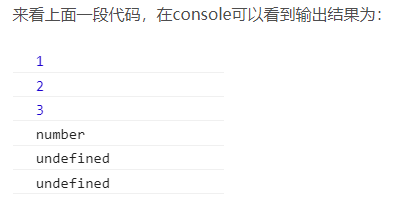
eval(jsstring);

}());

console.log(typeof un);

console.log(typeof deux);

console.log(typeof trois);



首先定义了全局变量 jsstring为字符串，eval执行了内容的计算，在控制台显示1，运行完成后un变为全局变量；Function中的代码将在局部函数运行，因此代码中deux不会成为全局变量，因此后面输出deux，trois的类型都是未定义。

无论哪里执行Function都仅仅能看到全局作用域，不改变局部变量，eval（）则会访问和修改它外部作用域的变量。

<script type="text/javascript">

(function(){

var local=1;

eval("local=3;console.log(local)"); //3

console.log(local);//3

}());

(function(){

var local=1;

Function("console.log(typeof local);")();//undefined

console.log(local); //1

}());

</script>

有上述例子可以知道，Function不会把局部变量修改为全局变量

## js核心/高级

// 阿里面试题  
var ***length*** = 100  
function f1() {  
 ***console***.log(this.length)  
}  
var ***obj*** = {  
 length: 10,  
 f2: function (f1) {  
 f1() // 100  
 arguments[0]() // arguments调用f1() this指向arguments，arguments的长度为2。arguments 为参数集合， 类似于数组 里面也拥有length属性  
 }  
}  
***obj***.f2(f1, 1); // 100 2  
  
// this的指向：谁调用它就指向谁，即调用者  
var ***length*** = 100  
function f1() {  
 ***console***.log(this.length)  
}  
f1(); // 普通函数 f1() 无调用者 默认指向window  
  
// 有调用者  
var ***obj*** = {  
 x: 100,  
 y: 100,  
 length: 10,  
 f2: f1  
};  
  
***obj***.f2(); // 10  
  
function f2() {  
 ***console***.log(this)  
}  
var ***arr*** = [f2, 100, 10]  
***arr***[0]() // this指向数组  
  
// 解析// 两种方式调用者都是obj  
var ***obj2*** = {  
 id: 2  
}  
***console***.log(***obj2***.id)  
***console***.log(***obj2***['id'])  
  
function f3() {  
 ***console***.log(arguments)  
}  
f3(1,2,3,4,5,6,7)

## 本地储存localStorage/sessionStorage

### 1. 保存数据到本地

const info = {

name: 'Lee',

age: 20,

id: '001'

};

sessionStorage.setItem('key', JSON.stringify(info));

localStorage.setItem('key', JSON.stringify(info));

### 2. 从本地存储获取数据

var data1 = JSON.parse(sessionStorage.getItem('key'));

var data2 = JSON.parse(localStorage.getItem('key'));

### 3. 本地存储中删除某个保存的数据

sessionStorage.removeItem('key');

localStorage.removeItem('key');

### 4. 删除所有保存的数据

sessionStorage.clear();

localStorage.clear();

### 5. 监听本地存储的变化

Storage 发生变化（增加、更新、删除）时的 触发，同一个页面发生的改变不会触发，只会监听同一域名下其他页面改变 Storage

window.addEventListener('storage', function (e) {

console.log('key', e.key);

console.log('oldValue', e.oldValue);

console.log('newValue', e.newValue);

console.log('url', e.url);

})

## 循环

### 一维数组转二维

var arr = [

{"id":"1001","name":"值1","value":"111"},

{"id":"1001","name":"值1","value":"11111"},

{"id":"1002","name":"值2","value":"25462"},

{"id":"1002","name":"值2","value":"23131"},

{"id":"1002","name":"值2","value":"2315432"},

{"id":"1003","name":"值3","value":"333333"}

];

var map = {},

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

var ai = arr[i];

if(!map[ai.id]){

dest.push({

id: ai.id,

name: ai.name,

data: [ai]

});

map[ai.id] = ai;

}else{

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

var dj = dest[j];

if(dj.id == ai.id){

dj.data.push(ai);

break;

}

}

}

}

console.log(dest);

### Switch-case-break;

**switch**(s) {  
 **case** -1:  
 **return** '已拒绝';  
 **break**;  
 **case** 5:  
 **return** '待处理';  
 **break**;  
 **case** 7:  
 **return** '已发货';  
 **break**;  
 **case** -9:  
 **return** '已退款';  
 **break**;  
 **default**:  
 **return** '';  
}

### 栈、队列 push pop (js把十进制数转二进制)

* + **栈(stack):**

什么是: 一端封闭，只能从另一端进出的数组

1. 开头封闭, 结尾出入栈:

入栈: arr.push(值) 简化了arr[arr.length]=值

出栈: var last=arr.pop();

2. 结尾封闭，开头出入栈:

入栈: arr.unshift(值)

强调: 开头入栈的顺序和结尾入栈的结果相反！

何时: 只要希望获得与结尾入栈相反的元素顺序时

出栈: var first=arr.shift();

* + 队列(queue):

什么是: 只能从结尾进入，从开头出的数组

特点: FIFO

何时: 只要希望按先来后到的顺序使用数组元素时

如何:

结尾入队列: arr.push(值)

开头出队列: var first=arr.shift();

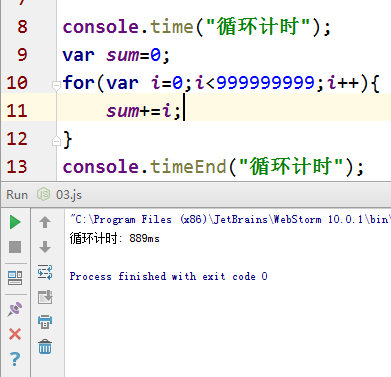
**var *n***=90;  
*//定义函数dec2bin,接收一个十进制数dec***function** *dec2bin*(dec){  
 *//定义空数组bin* **var** bin=[];  
 *//反复* **do**{  
 *//将dec%2的余数压入bin中* bin.unshift(dec%2);  
 *//将dec/2再取整，再保存回dec中* dec=*parseInt*(dec/2);  
 *//只要(dec>0),就反复执行* }**while**(dec>0)  
 *//将bin翻转(怎样就不需要翻转？)  
 //返回bin无缝拼接的结果* **return** bin  
}  
*dec2bin*(***n***)  
***console***.log(*dec2bin*(***n***), ***n***.toString(2));

### 计算程序运行时间：循环计时

console.time(“循环计时”)； //开始计时

function; //…..运行的程序；

console.timeEnd(“循环计时”)； //结束计时，循环计时的内容要一致



### 判断是什么平台（Android或iPhone）的浏览器

<**script *type***=**"text/javascript"**>  
 *//手机端判断各个平台浏览器及操作系统平台* **function** checkPlatform(){  
 **if**(/android/i.test(***navigator***.userAgent)){  
 ***document***.write(**"This is Android'browser."**);*//这是Android平台下浏览器* }  
 **if**(/(iPhone|iPad|iPod|iOS)/i.test(***navigator***.userAgent)){  
 ***document***.write(**"This is iOS'browser."**);*//这是iOS平台下浏览器* }  
 **if**(/Linux/i.test(***navigator***.userAgent)){  
 ***document***.write(**"This is Linux'browser."**);*//这是Linux平台下浏览器* }  
 **if**(/Linux/i.test(***navigator***.platform)){  
 ***document***.write(**"This is Linux operating system."**);*//这是Linux操作系统平台* }  
 **if**(/MicroMessenger/i.test(***navigator***.userAgent)){  
 ***document***.write(**"This is MicroMessenger'browser."**);*//这是微信平台下浏览器* }  
 }  
$(***document***).ready(**function**(){  
 *alert*(***navigator***.platform);  
 checkPlatform();  
});  
</script>

## BOM和DOM部分

### 屏幕/浏览器宽度、高度和适配

* **宽高：**

document.body.clientWidth; //网页可见区域宽(body)

document.body.clientHeight; //网页可见区域高(body)

document.body.offsetWidth; //网页可见区域宽(body)，包括border、margin等

document.body.offsetHeight; //网页可见区域宽(body)，包括border、margin等

document.body.scrollWidth; //网页正文全文宽，包括有滚动条时的未见区域

document.body.scrollHeight; //网页正文全文高，包括有滚动条时的未见区域

document.body.scrollTop; //网页被卷去的Top(滚动条)

document.body.scrollLeft; //网页被卷去的Left(滚动条)

window.screenTop; //浏览器距离Top window.screenLeft; //浏览器距离Left

window.screen.height; //屏幕分辨率的高

window.screen.width; //屏幕分辨率的宽

window.screen.availHeight; //屏幕可用工作区的高

window.screen.availWidth; //屏幕可用工作区的宽

window.innerWidth; //窗口的文档显示区域宽度

window.innerHeight; //窗口的文档显示区域高度

* **屏幕比例 | 尺寸**

1024\*500 （8.9寸）

1024\*768 （比例4：3 | 10.4寸、12.1寸、14.1寸、15寸; ）

1280\*800（16：10 |15.4寸）

1280\*1024(比例：5：4 | 14.1寸、15.0寸)

1280\*854(比例：15：10 | 15.2） 1366\*768 (比例：16：9 | 不常见）

1440\*900 （16：10 17寸 仅苹果用）

1440\*1050（比例：5：4 | 14.1寸、15.0寸）

1600\*1024（14：9 不常见） 1600\*1200 （4：3 | 15、16.1）

1680\*1050（16：10 | 15.4寸、20.0寸）

1920\*1200 (23寸）

* **屏幕常用宽度：**

1024 1280 1366 1440 1680 1920

* **PC端响应式媒体断点**

@media (min-width: 1024px){ body{font-size: 18px} /\*>=1024的设备\*/ }

@media (min-width: 1100px) { body{font-size: 20px} /\*>=1100的设备\*/ }

@media (min-width: 1280px) { body{font-size: 22px;} /\*>=1280的设备\*/ }

@media (min-width: 1366px) { body{font-size: 24px;} }

@media (min-width: 1440px) { body{font-size: 25px;} }

@media (min-width: 1680px) { body{font-size: 28px;} }

@media (min-width: 1920px) { body{font-size: 33px;} }

### 横屏和竖屏

1. **通过在html中分别引用横屏和竖屏的样式：**

<link rel="stylesheet" media="all and (orientation:portrait)" href="portrait.css" rel="external nofollow" > //引用竖屏的CSS  
<link rel="stylesheet" media="all and (orientation:landscape)" href="landscape.css" rel="external nofollow" > //引用横屏的CSS

1. **CSS中通过媒体查询的方法来判断：**

@media (orientation: portrait ){  
//竖屏CSS  
}  
@media ( orientation: landscape ){  
//横屏CSS  
}

1. **js判断是否为横屏竖屏:**

***window***.addEventListener("onorientationchange" in ***window*** ? orientationchange" : "resize", function() {  
if (***window***.orientation === 180 || ***window***.orientation === 0) {  
 alert('竖屏状态！');  
}  
if (***window***.orientation === 90 || ***window***.orientation === -90 ){  
 alert('横屏状态！');  
}  
}, false);

只要用户改变了设备的查看模式，就会触发onorientationchange事件。

orientation有4个值：0,90，-90,180

值为0和180的时候为竖屏（180为倒过来的竖屏）；

90和-90时为横屏（-90为倒过来的竖屏模式）；

1. 可以通过访问对象的 matches 属性来查看查询结果：

var mql = ***window***.matchMedia('(orientation: portrait)')  
***console***.log(mql)  
  
function handleOrientationChange(mql) {  
 if ( mql.matches ) {  
 ***console***.log('portrait') // 竖屏  
 }  
 else {  
 ***console***.log('landscape') // 横屏  
 }  
}  
  
// 输出当前屏幕模式  
handleOrientationChange(mql)  
// 监听屏幕模式变化  
mql.addListener(handleOrientationChange)

### 弹出的窗口

* 1. aleart(); 警告
  2. prompt(); 输入
  3. confirm(); 确认

### 元素js – 页面滚动至顶部动画

var getTop = document.getElementById("get-top");

var head = document.getElementById("head");

getTop.onclick = function () {

var time = setInterval(function () {

document.body.scrollTop = document.body.scrollTop - 50;

if (document.body.scrollTop === 0) {

clearInterval(time);

}

}, 1);

};

### 模态框（页面弹层）出现阻止底部body滚动

参考网址：<https://blog.csdn.net/yuhk231/article/details/74171734>

1、只需要把body的overflow改为hidden ~ this.fade为布尔型

**if** (**this**.fade) {  
 document.body.style.overflow = 'hidden'  
}**else**{  
 document.body.style.overflow = 'auto'  
}

2、*/\*\* 阻止弹窗底层body内容滚动, 需传入参数true/false(1 或 0 ) \*\*/***var** toggleBody = **function**(isModal) {  
 **if**(isModal){  
  
 document.body.style.height = '100vh';  
  
 document.body.style['overflow-y'] = 'hidden'ss  
 }  
 **else** {  
  
 document.body.style.height = 'unset'  
  
 document.body.style['overflow-y'] = 'auto'  
  
 }  
}

toggleBody(1);

toggleBody(0)

### 7、窗口大小

窗口的完整大小：window.outerWidth/outerHeight;

仅文档显示区大小: window.innerWidth/innerHeight——用得多

### 鼠标指针位置和页面滚动位置

**1. 鼠标位置:**

相对于屏幕左上角的坐标: e.screenX, e.screenY

相对于文档显示区左上角的坐标: e.clientX,e.clientY

相对于当前元素左上角的坐标: e.offsetX, e.offsetY

**2. 页面滚动:**

事件: window.onscroll=function(){document.body.scrollTop //滚动距离 }

<script>

//为div绑定鼠标按下和鼠标抬起事件

//为body绑定mousemove事件

var pop=document.getElementById("pop");

var canMove=false;//标识是否可移动

var offsetX,offsetY;

pop.onmousedown=function(e){

//准备拖拽

canMove=true;

offsetX= e.offsetX;

offsetY= e.offsetY;

}

document.body.onmousemove=function(e){

if(canMove){

//让pop跟随鼠标移动

var top= e.clientY-offsetY;

var left= e.clientX-offsetX;

pop.style.cssText=

"top:"+top+"px;left:"+left+"px";

} }

pop.onmouseup=function(){ /\*结束拖拽\*/ canMove=false; }

</script>

### 获得元素距离页面顶部的距离(offset.top)

offsetTop=$(“#f1”).offset().top;

### [JS/JQ获取各种屏幕的高度和宽度](http://www.cnblogs.com/zhishiqu/p/6121844.html)

Javascript:

网页可见区域宽： document.body.clientWidth  
网页可见区域高： document.body.clientHeight  
网页可见区域宽： document.body.offsetWidth (包括边线的宽)  
网页可见区域高： document.body.offsetHeight (包括边线的高)  
网页正文全文宽： document.body.scrollWidth  
网页正文全文高： document.body.scrollHeight  
网页被卷去的高： document.body.scrollTop  
网页被卷去的左： document.body.scrollLeft  
网页正文部分上： window.screenTop  
网页正文部分左： window.screenLeft  
屏幕分辨率的高： window.screen.height  
屏幕分辨率的宽： window.screen.width  
屏幕可用工作区高度： window.screen.availHeight  
屏幕可用工作区宽度： window.screen.availWidth

JQuery:

$(document).ready(function(){  
alert($(window).height()); //浏览器当前窗口可视区域高度  
alert($(document).height()); //浏览器当前窗口文档的高度  
alert($(document.body).height());//浏览器当前窗口文档body的高度  
alert($(document.body).outerHeight(true));//浏览器当前窗口文档body的总高度 包括border padding margin  
  
alert($(window).width()); //浏览器当前窗口可视区域宽度  
alert($(document).width());//浏览器当前窗口文档对象宽度  
alert($(document.body).width());//浏览器当前窗口文档body的宽度  
alert($(document.body).outerWidth(true));//浏览器当前窗口文档body的总宽度 包括border padding margin  
  
})

### eval(),js计算表达式

var t=eval(10+20-5-5\*2); // t=1

### 解决苹果键盘收回body长度没有恢复问题（定位/input焦点错乱）

**function** getScrollTop() {  
 **let** scrollTop = 0  
 **if** (document.documentElement && document.documentElement.scrollTop) {  
 scrollTop = document.documentElement.scrollTop  
 } **else if** (document.body) {  
 scrollTop = document.body.scrollTop  
 }  
 **return** scrollTop  
 }  
 // 记录当前滚动位置  
 **let** oldScrollTop = getScrollTop() || 0  
 // 软键盘弹起事件  
 document.body.addEventListener('focusin', () => {  
 // console.log('键盘弹起')  
 // document.title = "键盘弹起" + $(".weui-dialog").css("position") + CBJS.getScrollTop() + $(".weui-dialog").css("top");  
 })  
 // 软键盘关闭事件  
 document.body.addEventListener('focusout', () => {  
 // console.log('键盘收起')  
 // document.title = "键盘收起" + $(".weui-dialog").css("position") + CBJS.getScrollTop() + $(".weui-dialog").css("top");  
  
 **let** ua = window.navigator.userAgent  
 // 键盘收起页面空白问题  
 **if** (ua.indexOf('iPhone') > 0 || ua.indexOf('iPad') > 0) {  
 document.body.scrollTop = oldScrollTop  
 document.documentElement.scrollTop = oldScrollTop  
 }  
 })  
}

### 滚动到溢出元素内指定位置（如菜单联动）

1. **使用a标签+#号**

<a href="#box1">我跳到box1</a>

<div id="box2"></div>

上面这一种的缺点是，url会发生改变，会在url最后面加上片段识别符#box2，这样就改变了URL的路径，这是我们不想看到的。因为再次刷新的时候回出现问题。当使用 vue-router回退时，发现需要回退很多次。

1. **scrollTop滚动到某位置**

document.documentElement.scrollTop=380;//不需要加单位

直接这句就可以实现滚动到某位置

1. **scrollIntoView()实现滚动到某元素**

document.getElementById("ID").scrollIntoView();

### js判断一个元素是否在页面中出现

checkInPage = el => {

const pageHeight = document.documentElement.clientHeight

const contentTop = el.getBoundingClientRect().top

const contentHeight = el.offsetHeight

return (contentTop<pageHeight && contentTop>=0) || (contentTop<0 && (contentTop+contentHeight>0));

## JS同步执行逻辑

### JS函数同步执行逻辑

1、使用ajax同步

**function** function1(){  
 $.ajax({  
 url: "some.action", async: **false**,   
 success:**function**(data){**return** data;}   
 })  
} //需要依赖函数1的结果   
  
**function** function2(data){  
 //基于data进行一些操作 //   
}  
  
/函数3中执行函数1，然后把函数2作为回调函数放入函数1的入参中   
**function** function3(){   
 **var** data = function1();  
 function2(data);  
}

2、使用回调函数

**function** function1(callBack){  
 $.ajax({  
 url: "some.action",  
 async: **true**,  
 success : **function**(data){  
 //判断有没有设置回调函数，如果设置回调函数，则执行回调函数  
 **if**(callBack)  
 callBack(data);  
 }  
 })  
}  
//需要执行的回调函数  
**function** function2(data){  
 //基于data进行一些操作  
 //  
}  
  
//函数3中执行函数1，然后把函数2作为回调函数放入函数1的入参中  
**function** function3(){  
 function1(**function**(data){  
 function2(data);  
 })  
}

### 判断是什么平台（Android或iPhone）的浏览器

<**script *type***=**"text/javascript"**>  
 *//手机端判断各个平台浏览器及操作系统平台* **function** checkPlatform(){  
 **if**(/android/i.test(***navigator***.userAgent)){  
 ***document***.write(**"This is Android'browser."**);*//这是Android平台下浏览器* }  
 **if**(/(iPhone|iPad|iPod|iOS)/i.test(***navigator***.userAgent)){  
 ***document***.write(**"This is iOS'browser."**);*//这是iOS平台下浏览器* }  
 **if**(/Linux/i.test(***navigator***.userAgent)){  
 ***document***.write(**"This is Linux'browser."**);*//这是Linux平台下浏览器* }  
 **if**(/Linux/i.test(***navigator***.platform)){  
 ***document***.write(**"This is Linux operating system."**);*//这是Linux操作系统平台* }  
 **if**(/MicroMessenger/i.test(***navigator***.userAgent)){  
 ***document***.write(**"This is MicroMessenger'browser."**);*//这是微信平台下浏览器* }  
 }  
$(***document***).ready(**function**(){  
 *alert*(***navigator***.platform);  
 checkPlatform();  
});  
</script>

核心部分

构造函数

## ES6

### 剩余参数语法 - function sum(...theArgs) - 省略号。

function sum(...theArgs) {

return theArgs.reduce((previous, current) => {

return previous + current;

});

}

console.log(sum(1, 2, 3));

// expected output: 6

console.log(sum(1, 2, 3, 4));

// expected output: 10

### export和import

export {myFunction};//命名导出函数

export const value=5;//命名导出原始值

export {object1, object2, ....}; //命名导出对象

命名导出可以导出多个值，但是导入时命名需要与导出命名相同

import $ from 'jquery'; // export default jquery

import \* as xxx from 'xxx';  //通过\*来批量接收，as 来指定接收的名字

    // index.js

export function fn1(data){

  console.log(1)

}

export function fn2(data){

  console.log(2)

}

    import \* as Fn from './index.js'

    Fn.fn1()  // 1

    Fn.fn2()  // 2

export let myName = "我来自test1.js";

import myName as name1 from 'xxx' // 重命名

/\*\*\*\*\*\*\*\*\*\*\*\*\*test2.js\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

export let myName = "我来自test2.js";

/\*\*\*\*\*\*\*\*\*\*\*\*\*index.js\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import {myName as name1} from "./test1.js";

import {myName as name2} from "./test2.js";

console.log(name1); //我来自test1.js

console.log(name2); //我来自test2.js

import {a,b,c} from './a'; // export {a, b, c} || exoprt const a = 1; export const b = 2

import {default as alias, a as a\_a, b, c} from './a';

import React,{Component} from 'react'

## jQuery部分

**jQuery CDN :**

官网：

<script src="http://code.jquery.com/jquery-migrate-1.2.1.min.js"></script>

<script   src="http://code.jquery.com/jquery-3.3.1.slim.min.js"   integrity="sha256-3edrmyuQ0w65f8gfBsqowzjJe2iM6n0nKciPUp8y+7E="   crossorigin="anonymous"></script>

百度：

<script src="http://libs.baidu.com/jquery/2.0.0/jquery.min.js"></script>

### 获得元素名称：

\*原生js：var name = element.tagName;

jq：

1、$( this ).get(0).tagName  
2、$( this )[0].tagName  
3、$( this ).prop("tagName")  
4、$( this ).prop("nodeName")

5、有时可以用is取代：

$(this).is(“b”) //b为元素名称

### 父元素增加子元素/添加子元素-prepend和append

1. **prepend=>向每个匹配的元素内部前置内容**

**例子1：**

<p>I would like to say: </p>

$("p").prepend("<b>Hello</b>");

结果：

[ <p><b>Hello</b>I would like to say: </p> ]

例子2： //将数量，单价和小计拼接为一行，追加到tbody下

$("tbody").prepend(

`<tr>

<td>¥${p.toFixed(2)}</td>

<td>${c}</td>

<td>¥${sub.toFixed(2)}</td>

<td><span class="close">×</span></td>

</tr>`

);

1. **append 向每个匹配的元素内部追加内容。**

<p>I would like to say: </p>

$("p").append("<b>Hello</b>");

结果：

[ <p>I would like to say: <b>Hello</b></p> ]

### 弹出错误/自定义插件时判断是否引入jq文件

if(typeof jQuery!="function"){

throw new Error("myAccordion插件依赖于jQuery，必须先引入jquery.js");

}else{

jQuery.fn.accordion=function(){

this.on("click",".title",e=>

$(e.target)

.next(".content").toggleClass("in")

.siblings(".content").removeClass("in")

);

};

}

### jQuery事件处理的另一种写法：

**$(“.nav”).on(“mouseenter”,”a”,function(){})**  //”.nav”是“a”的父元素；“a”是“.nav”的子元素，函数内可以不用e.target的冒泡判断=>if($(e.target).is(“a”)){…}

### jq 按钮只能点一次（事件只处理一次）

**$**(**"#bt-lottery"**).one(**"click"**,**function**(){  
 **console**.log(1)  
 })

### jQuery自带循环获取数组中每一个对象

**$.each(data,function(idx,obj){});**

//data为数组，idx为数组下标，obj当前遍历的对象（数组内容）

### jq判断checkbox是否被选择的三种方法

**方法一：**if ($("#checkbox-id")get(0).checked) {  
    // do something  
}

**方法二：**if($('#checkbox-id').is(':checked')) {  
    // do something  
}

**方法三：**if ($('#checkbox-id').attr('checked')) {  
    // do something  
}

### 将canvas转换成微信可以识别的二维码图片

<!DOCTYPE html>  
<html>  
<head>  
<meta charset=**"utf-8"**>  
 <meta name=**"viewport"** content=**"width=device-width, initial-scale=1, maximum-scale=1"**>  
 <**title**>微信长按二维码识别</title>  
 </head>  
 <**body**>  
 <div id=**"qrDiv"**></div>  
 </body>  
 </html>  
 <script src=**"http://cdn.bootcss.com/jquery/1.8.3/jquery.min.js"** type=**"text/javascript"** charset=**"utf-8"**></script>  
 <script src=**"http://cdn.bootcss.com/jquery.qrcode/1.0/jquery.qrcode.min.js"** type=**"text/javascript"** charset=**"utf-8"**></script>  
 <script type=**"text/javascript"**>  
 **$**(**"#qrDiv"**).qrcode({  
 **width**: 120, *//宽度* **height**:120, *//高度* **text**: **"需要生成的二维码内容"** *//任意内容* });  
  
*//从 canvas 提取图片 image***function** *convertCanvasToImage*(canvas) {  
 *//新Image对象，可以理解为DOM* **var** image = **new *Image***();  
 *// canvas.toDataURL 返回的是一串Base64编码的URL，当然,浏览器自己肯定支持   
 // 指定格式 PNG* image.src = canvas.toDataURL(**"image/png"**);  
 **return** image;  
}  
  
*//获取网页中的canvas对象***var *mycanvas1***=***document***.getElementsByTagName(**'canvas'**)[0];  
*//将转换后的img标签插入到html中***var *img***=*convertCanvasToImage*(***mycanvas1***);  
**$**(**'#imagQrDiv'**).append(***img***);*//imagQrDiv表示你要插入的容器id*</script>

### Promise

**Promise all(arr)**

1. **for循环内多次调用异步函数并且需要等待执行结果后再出来后面逻辑**

