作者：爱前端不爱恋爱  
链接：https://zhuanlan.zhihu.com/p/143862373  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

**数组 Array**

**数组去重**

**方案一：Set + ...**

function noRepeat(arr) {

return [...new Set(arr)];

}

noRepeat([1,2,3,1,2,3])

**方案二：Set + Array.from**

function noRepeat(arr) {

return Array.from(new Set(arr));

}

noRepeat([1,2,3,1,2,3])

**方案三：双重遍历比对下标**

function noRepeat(arr) {

return arr.filter((v, idx)=>idx == arr.lastIndexOf(v))

}

noRepeat([1,2,3,1,2,3])

**方案四：单遍历 + Object 特性**

Object 的特性是 Key 不会重复。  
这里使用 values 是因为可以保留类型，keys 会变成字符串。

function noRepeat(arr) {

return Object.values(arr.reduce((s,n)=>{

s[n] = n;

return s

},{}))

}

noRepeat([1,2,3,1,2,3])

**后记**

针对于上述的方案，还有其他变种实现。

**查找数组最大**

**方案一：Math.max + ...**

function arrayMax(arr) {

return Math.max(...arr);

}

arrayMax([-1,-4,5,2,0])

**方案二：Math.max + apply**

function arrayMax(arr) {

return Math.max.apply(Math, arr)

}

arrayMax([-1,-4,5,2,0])

**方案三：Math.max + 遍历**

function arrayMax(arr) {

return arr.reduce((s,n)=>Math.max(s, n))

}

arrayMax([-1,-4,5,2,0])

**方案四：比较、条件运算法 + 遍历**

function arrayMax(arr) {

return arr.reduce((s,n)=>s>n?s:n)

}

arrayMax([-1,-4,5,2,0])

**方案五：排序**

function arrayMax(arr) {

return arr.sort((n,m)=>m-n)[0]

}

arrayMax([-1,-4,5,2,0])

**查找数组最小**

同上，*不明白为什么要分成两个题目*。

1. Math.max 换成 Math.min
2. s>n?s:n 换成 s<n?s:n
3. (n,m)=>m-n 换成 (n,m)=>n-m，或者直接取最后一个元素

**返回已 size 为长度的数组分割的原数组**

**方案一：Array.from + slice**

function chunk(arr, size = 1) {

return Array.from(

{

length: Math.ceil(arr.length / size),

},

(v, i) => arr.slice(i \* size, i \* size + size)

);

}

chunk([1,2,3,4,5,6,7,8],3)

**方案二：Array.from + splice**

function chunk(arr, size = 1) {

return Array.from(

{

length: Math.ceil(arr.length / size),

},

(v, i) => arr.splice(0, size)

);

}

chunk([1,2,3,4,5,6,7,8],3)

**方案三：遍历 + splice**

function chunk(arr, size = 1) {

var \_returnArr = [];

while(arr.length){

\_returnArr.push(arr.splice(0, size))

}

return \_returnArr

}

chunk([1,2,3,4,5,6,7,8],3)

**检查数组中某元素出现的次数**

**方案一：reduce**

function countOccurrences(arr, value) {

return arr.reduce((a, v) => (v === value ? a + 1 : a + 0), 0);

}

countOccurrences([1,2,3,4,5,1,2,1,2,3], 1)

**方案二：filter**

function countOccurrences(arr, value) {

return arr.filter(v=>v===value).length

}

countOccurrences([1,2,3,4,5,1,2,1,2,3], 1)

**扁平化数组**

**方案一：递归 + ...**

function flatten(arr, depth = -1) {

if (depth === -1) {

return [].concat(

...arr.map((v) => (Array.isArray(v) ? this.flatten(v) : v))

);

}

if (depth === 1) {

return arr.reduce((a, v) => a.concat(v), []);

}

return arr.reduce(

(a, v) => a.concat(Array.isArray(v) ? this.flatten(v, depth - 1) : v),

[]

);

}

flatten([1,[2,[3]]])

**方案二：es6 原生 flat**

function flatten(arr, depth = Infinity) {

return arr.flat(depth)

}

flatten([1,[2,[3]]])

**对比两个数组并且返回其中不同的元素**

**方案一：filter + includes**

他原文有问题，以下方法的 4,5 没有返回

function diffrence(arrA, arrB) {

return arrA.filter((v) => !arrB.includes(v));

}

diffrence([1,2,3], [3,4,5,2])

需要再操作一遍

function diffrence(arrA, arrB) {

return arrA.filter((v) => !arrB.includes(v))

.concat(arrB.filter((v) => !arrA.includes(v)));

}

diffrence([1,2,3], [3,4,5,2])

**方案二：hash + 遍历**

算是方案1的变种吧，优化了 includes 的性能。

**返回两个数组中相同的元素**

**方案一：filter + includes**

function intersection(arr1, arr2) {

return arr2.filter((v) => arr1.includes(v));

}

intersection([1,2,3], [3,4,5,2])

**方案二：同理变种用 hash**

function intersection(arr1, arr2) {

var set = new Set(arr2)

return arr1.filter((v) => set.has(v));

}

intersection([1,2,3], [3,4,5,2])

**从右删除 n 个元素**

**方案一：slice**

function dropRight(arr, n = 0) {

return n < arr.length ? arr.slice(0, arr.length - n) : [];

}

dropRight([1,2,3,4,5], 2)

**方案二: splice**

function dropRight(arr, n = 0) {

return arr.splice(0, arr.length - n)

}

dropRight([1,2,3,4,5], 2)

**方案三: slice 另一种**

function dropRight(arr, n = 0) {

return arr.slice(0, -n)

}

dropRight([1,2,3,4,5], 2)

**方案四: 修改 length**

function dropRight(arr, n = 0) {

arr.length = Math.max(arr.length - n, 0)

return arr

}

dropRight([1,2,3,4,5], 2)

**截取第一个符合条件的元素及其以后的元素**

**方案一：slice + 循环**

function dropElements(arr, fn) {

while (arr.length && !fn(arr[0])) arr = arr.slice(1);

return arr;

}

dropElements([1,2,3,4,5,1,2,3], (v) => v == 2)

**方案二：findIndex + slice**

function dropElements(arr, fn) {

return arr.slice(Math.max(arr.findIndex(fn), 0));

}

dropElements([1,2,3,4,5,1,2,3], (v) => v === 3)

**方案三：splice + 循环**

function dropElements(arr, fn) {

while (arr.length && !fn(arr[0])) arr.splice(0,1);

return arr;

}

dropElements([1,2,3,4,5,1,2,3], (v) => v == 2)

**返回数组中下标间隔 nth 的元素**

**方案一：filter**

function everyNth(arr, nth) {

return arr.filter((v, i) => i % nth === nth - 1);

}

everyNth([1,2,3,4,5,6,7,8], 2)

**方案二：方案一修改判断条件**

function everyNth(arr, nth) {

return arr.filter((v, i) => (i+1) % nth === 0);

}

everyNth([1,2,3,4,5,6,7,8], 2)

**返回数组中第 n 个元素（支持负数）**

**方案一：slice**

function nthElement(arr, n = 0) {

return (n >= 0 ? arr.slice(n, n + 1) : arr.slice(n))[0];

}

nthElement([1,2,3,4,5], 0)

nthElement([1,2,3,4,5], -1)

**方案二：三目运算符**

function nthElement(arr, n = 0) {

return (n >= 0 ? arr[0] : arr[arr.length + n])

}

nthElement([1,2,3,4,5], 0)

nthElement([1,2,3,4,5], -1)

**返回数组头元素**

**方案一：**

function head(arr) {

return arr[0];

}

head([1,2,3,4])

**方案二：**

function head(arr) {

return arr.slice(0,1)[0];

}

head([1,2,3,4])

**返回数组末尾元素**

**方案一：**

function last(arr) {

return arr[arr.length - 1];

}

**方案二：**

function last(arr) {

return arr.slice(-1)[0];

}

last([1,2,3,4,5])

**数组乱排**

**方案一：洗牌算法**

function shuffle(arr) {

let array = arr;

let index = array.length;

while (index) {

index -= 1;

let randomInedx = Math.floor(Math.random() \* index);

let middleware = array[index];

array[index] = array[randomInedx];

array[randomInedx] = middleware;

}

return array;

}

shuffle([1,2,3,4,5])

**方案二：sort + random**

function shuffle(arr) {

return arr.sort((n,m)=>Math.random() - .5)

}

shuffle([1,2,3,4,5])

**伪数组转换为数组**

**方案一：Array.from**

Array.from({length: 2})

**方案二：prototype.slice**

Array.prototype.slice.call({length: 2,1:1})

**方案三：prototype.splice**

Array.prototype.splice.call({length: 2,1:1},0)

**浏览器对象 BOM**

**判读浏览器是否支持 CSS 属性**

/\*\*

\* 告知浏览器支持的指定css属性情况

\* @param {String} key - css属性，是属性的名字，不需要加前缀

\* @returns {String} - 支持的属性情况

\*/

function validateCssKey(key) {

const jsKey = toCamelCase(key); // 有些css属性是连字符号形成

if (jsKey in document.documentElement.style) {

return key;

}

let validKey = "";

// 属性名为前缀在js中的形式，属性值是前缀在css中的形式

// 经尝试，Webkit 也可是首字母小写 webkit

const prefixMap = {

Webkit: "-webkit-",

Moz: "-moz-",

ms: "-ms-",

O: "-o-",

};

for (const jsPrefix in prefixMap) {

const styleKey = toCamelCase(`${jsPrefix}-${jsKey}`);

if (styleKey in document.documentElement.style) {

validKey = prefixMap[jsPrefix] + key;

break;

}

}

return validKey;

}

/\*\*

\* 把有连字符号的字符串转化为驼峰命名法的字符串

\*/

function toCamelCase(value) {

return value.replace(/-(\w)/g, (matched, letter) => {

return letter.toUpperCase();

});

}

/\*\*

\* 检查浏览器是否支持某个css属性值（es6版）

\* @param {String} key - 检查的属性值所属的css属性名

\* @param {String} value - 要检查的css属性值（不要带前缀）

\* @returns {String} - 返回浏览器支持的属性值

\*/

function valiateCssValue(key, value) {

const prefix = ["-o-", "-ms-", "-moz-", "-webkit-", ""];

const prefixValue = prefix.map((item) => {

return item + value;

});

const element = document.createElement("div");

const eleStyle = element.style;

// 应用每个前缀的情况，且最后也要应用上没有前缀的情况，看最后浏览器起效的何种情况

// 这就是最好在prefix里的最后一个元素是''

prefixValue.forEach((item) => {

eleStyle[key] = item;

});

return eleStyle[key];

}

/\*\*

\* 检查浏览器是否支持某个css属性值

\* @param {String} key - 检查的属性值所属的css属性名

\* @param {String} value - 要检查的css属性值（不要带前缀）

\* @returns {String} - 返回浏览器支持的属性值

\*/

function valiateCssValue(key, value) {

var prefix = ["-o-", "-ms-", "-moz-", "-webkit-", ""];

var prefixValue = [];

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

prefixValue.push(prefix[i] + value);

}

var element = document.createElement("div");

var eleStyle = element.style;

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

eleStyle[key] = prefixValue[j];

}

return eleStyle[key];

}

function validCss(key, value) {

const validCss = validateCssKey(key);

if (validCss) {

return validCss;

}

return valiateCssValue(key, value);

}

[https://segmentfault.com/a/11...](https://link.zhihu.com/?target=https%3A//segmentfault.com/a/1190000022623676%23item-2-16)  
*它里面有 forEach。*

**返回当前网页地址**

**方案一：location**

function currentURL() {

return window.location.href;

}

currentURL()

**方案二：a 标签**

function currentURL() {

var el = document.createElement('a')

el.href = ''

return el.href

}

currentURL()

**获取滚动条位置**

function getScrollPosition(el = window) {

return {

x: el.pageXOffset !== undefined ? el.pageXOffset : el.scrollLeft,

y: el.pageYOffset !== undefined ? el.pageYOffset : el.scrollTop,

};

}

**获取 url 中的参数**

**方案一：正则 + reduce**

function getURLParameters(url) {

return url

.match(/([^?=&]+)(=([^&]\*))/g)

.reduce(

(a, v) => (

(a[v.slice(0, v.indexOf("="))] = v.slice(v.indexOf("=") + 1)), a

),

{}

);

}

getURLParameters(location.href)

**方案二：split + reduce**

function getURLParameters(url) {

return url

.split('?') //取？分割

.slice(1) //不要第一部分

.join() //拼接

.split('&')//&分割

.map(v=>v.split('=')) //=分割

.reduce((s,n)=>{s[n[0]] = n[1];return s},{})

}

getURLParameters(location.href)

// getURLParameters('')

**方案三: URLSearchParams**

**页面跳转，是否记录在 history 中**

**方案一：**

function redirect(url, asLink = true) {

asLink ? (window.location.href = url) : window.location.replace(url);

}

**方案二：**

function redirect(url, asLink = true) {

asLink ? window.location.assign(url) : window.location.replace(url);

}

**滚动条回到顶部动画**

**方案一： c - c / 8**

*c 没有定义*

function scrollToTop() {

const scrollTop =

document.documentElement.scrollTop || document.body.scrollTop;

if (scrollTop > 0) {

window.requestAnimationFrame(scrollToTop);

window.scrollTo(0, c - c / 8);

} else {

window.cancelAnimationFrame(scrollToTop);

}

}

scrollToTop()

修正之后

function scrollToTop() {

const scrollTop =

document.documentElement.scrollTop || document.body.scrollTop;

if (scrollTop > 0) {

window.requestAnimationFrame(scrollToTop);

window.scrollTo(0, scrollTop - scrollTop / 8);

} else {

window.cancelAnimationFrame(scrollToTop);

}

}

scrollToTop()

**复制文本**

**方案一：**

function copy(str) {

const el = document.createElement("textarea");

el.value = str;

el.setAttribute("readonly", "");

el.style.position = "absolute";

el.style.left = "-9999px";

el.style.top = "-9999px";

document.body.appendChild(el);

const selected =

document.getSelection().rangeCount > 0

? document.getSelection().getRangeAt(0)

: false;

el.select();

document.execCommand("copy");

document.body.removeChild(el);

if (selected) {

document.getSelection().removeAllRanges();

document.getSelection().addRange(selected);

}

}

**方案二：cliboard.js**

**检测设备类型**

**方案一： ua**

function detectDeviceType() {

return /Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.test(

navigator.userAgent

)

? "Mobile"

: "Desktop";

}

detectDeviceType()

**方案二：事件属性**

function detectDeviceType() {

return ("ontouchstart" in window || navigator.msMaxTouchPoints)

? "Mobile"

: "Desktop";

}

detectDeviceType()

**Cookie**

**增**

function setCookie(key, value, expiredays) {

var exdate = new Date();

exdate.setDate(exdate.getDate() + expiredays);

document.cookie =

key +

"=" +

escape(value) +

(expiredays == null ? "" : ";expires=" + exdate.toGMTString());

}

**删**

function delCookie(name) {

var exp = new Date();

exp.setTime(exp.getTime() - 1);

var cval = getCookie(name);

if (cval != null) {

document.cookie = name + "=" + cval + ";expires=" + exp.toGMTString();

}

}

**查**

function getCookie(name) {

var arr,

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

if ((arr = document.cookie.match(reg))) {

return arr[2];

} else {

return null;

}

}

**清空**

有时候我们想清空，但是又无法获取到所有的cookie。  
这个时候我们可以了利用写满，然后再清空的办法。

**日期 Date**

**时间戳转换为时间**

* 默认为当前时间转换结果
* isMs 为时间戳是否为毫秒

function timestampToTime(timestamp = Date.parse(new Date()), isMs = true) {

const date = new Date(timestamp \* (isMs ? 1 : 1000));

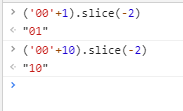
return `${date.getFullYear()}-${

date.getMonth() + 1 < 10 ? "0" + (date.getMonth() + 1) : date.getMonth() + 1

}-${date.getDate()} ${date.getHours()}:${date.getMinutes()}:${date.getSeconds()}`;

}

1. 补位可以改成 padStart
2. 补位还可以改成 slice



如果做海外的话，还会有时区问题，一般我用moment解决。如果想看[原生的](https://link.zhihu.com/?target=https%3A//www.lilnong.top/static/html/date-zone-test.html)

**获取当前时间戳**

基于上一个想到的问题

**方案一：Date.parse(new Date())**

**方案二：Date.now()**

**方案三：+new Date()**

**文档对象 DOM**

**固定滚动条**

/\*\*

\* 功能描述：一些业务场景，如弹框出现时，需要禁止页面滚动，这是兼容安卓和 iOS 禁止页面滚动的解决方案

\*/

let scrollTop = 0;

function preventScroll() {

// 存储当前滚动位置

scrollTop = window.scrollY;

// 将可滚动区域固定定位，可滚动区域高度为 0 后就不能滚动了

document.body.style["overflow-y"] = "hidden";

document.body.style.position = "fixed";

document.body.style.width = "100%";

document.body.style.top = -scrollTop + "px";

// document.body.style['overscroll-behavior'] = 'none'

}

function recoverScroll() {

document.body.style["overflow-y"] = "auto";

document.body.style.position = "static";

// document.querySelector('body').style['overscroll-behavior'] = 'none'

window.scrollTo(0, scrollTop);

}

**判断当前位置是否为页面底部**

* 返回值为 true/false

function bottomVisible() {

return (

document.documentElement.clientHeight + window.scrollY >=

(document.documentElement.scrollHeight ||

document.documentElement.clientHeight)

);

}

**判断元素是否在可视范围内**

* partiallyVisible 为是否为完全可见

function elementIsVisibleInViewport(el, partiallyVisible = false) {

const { top, left, bottom, right } = el.getBoundingClientRect();

return partiallyVisible

? ((top > 0 && top < innerHeight) ||

(bottom > 0 && bottom < innerHeight)) &&

((left > 0 && left < innerWidth) || (right > 0 && right < innerWidth))

: top >= 0 && left >= 0 && bottom <= innerHeight && right <= innerWidth;

}

**获取元素 css 样式**

function getStyle(el, ruleName) {

return getComputedStyle(el, null).getPropertyValue(ruleName);

}

**进入全屏**

function launchFullscreen(element) {

if (element.requestFullscreen) {

element.requestFullscreen();

} else if (element.mozRequestFullScreen) {

element.mozRequestFullScreen();

} else if (element.msRequestFullscreen) {

element.msRequestFullscreen();

} else if (element.webkitRequestFullscreen) {

element.webkitRequestFullScreen();

}

}

launchFullscreen(document.documentElement);

launchFullscreen(document.getElementById("id")); //某个元素进入全屏

**退出全屏**

function exitFullscreen() {

if (document.exitFullscreen) {

document.exitFullscreen();

} else if (document.msExitFullscreen) {

document.msExitFullscreen();

} else if (document.mozCancelFullScreen) {

document.mozCancelFullScreen();

} else if (document.webkitExitFullscreen) {

document.webkitExitFullscreen();

}

}

exitFullscreen();

**全屏事件**

document.addEventListener("fullscreenchange", function (e) {

if (document.fullscreenElement) {

console.log("进入全屏");

} else {

console.log("退出全屏");

}

});

**数字 Number**

**数字千分位分割**

function commafy(num) {

return num.toString().indexOf(".") !== -1

? num.toLocaleString()

: num.toString().replace(/(\d)(?=(?:\d{3})+$)/g, "$1,");

}

commafy(1000)

**生成随机数**

function randomNum(min, max) {

switch (arguments.length) {

case 1:

return parseInt(Math.random() \* min + 1, 10);

case 2:

return parseInt(Math.random() \* (max - min + 1) + min, 10);

default:

return 0;

}

}

randomNum(1,10)