ul无序 ol有序有标号 dl 缩进

Object

数据属性：

Object.getOwnPropertyDes criptor(abc, 'b')

{value: 0, writable: true, enumerable: true, configurable: true}  
访问属性：

Object. Object.defineProperties(o, ‘year’,{

get: function(){} // 获取属性执行

set: function(){} // 设置属性执行

enumerable

configurable

})

判断是否是自己的属性： hasOwnProperty

判断是否是属于自己活低级属性的： 属性 in obj

Array

排列公式： A(10, 2) = 9\*10 = 90 从10个元素里取出2个有序组合

组合公式： C(10, 2) = 9\*10 / 2\*1 从10个元素取出2个无序组合，

a.sort(function(value1, value2){  
 if(value1>value2){ // return -1时value1大则降序，value1小则升序

return -1;

}else if(value1<value2){ //相反

return1

}else{  
 return 0;

}

});

生成新的数组：concat() slice()（左闭右开）

改变原数组：splice(开始，删除，插入) 插入，删除

查询下标： indexOf(元素，开始); lastIndexOf(元素，开始)

join() //改变字符串

es6迭代:

数据格式判断：

every() 判断数组中所有元素是否满足一个元素返回boolea

some() 判断数组中每一个元素是否满足一个元素返回boolea

filter() map() forEach()

es6归并（折叠 注入）：

arr.reduce(function(pre，current，当前index, array), 10) 累加累乘

reduceRight()

Math：Math.E 自然对数底数，常量E的值

.LN10 10的自然对数

.LN2 2的自然对数

.LOG2E 以2为底的对数

.LOG10E 以10为底的对数

.PI 派的值

.SQRT1\_2 1/2的平方根（即2的平方根的倒数）

.SQRT2 2的平方根

.min() 若干参数取小

.max() 若干参数取大

.ceil() 向上

.floor() 向下

.round() 四舍五入

.random() 返回0-1之间的数

function random(min, max) {

return Math.random()\*(max-min)+min

}

.abs(num) 返回num的绝对值

.exp(num) 返回.E的num次幂

.log(num) 返回num的自然对数

.pow(num, power) 返回num的power次幂

.sqrt(num) 返回num的平方根

acos(x) 返回x的反余弦值

asin(x) 返回x的反正弦值

atan(x) 返回x的反正切值

atan2(y,x) 返回y/x的反正切值

cos(x) 返回x的余弦值

sin(x) 返回x的正弦值

tan(x) 返回x的正切值



浏览器延时器

setTimeout(function(){}, t) 可能会不准可能会长（加入js任务队列）

浏览器定时器

setInterval(function(){}, t) 可能会不准可能长可能短（加入js任务队列）

clearInteval(timer)

Date

new Date() 参数： 数字毫秒

国外参数：‘月’，‘日’，‘年’

标准： 年，月，日，时，分，秒

h5 Date.now(); 参数：字符串-获取毫秒

Date.parse(年，月，日); 返回毫秒 月从0开始

指定的日期和时间据 1970/1/1 午夜（GMT 时间）之间的毫秒数

Date.UTC(year,month,day,hours,minutes,seconds,ms) 月从0开始

返回指定的时间距 GMT 时间 1970 年 1 月 1 日午夜的毫秒数。

.toLocaleTimeString(); "上午10:44:53"

.toLocaleDateString(); “2019/01/01”

toLocaleString(); "2019/7/1 上午10:45:54"

|  |  |
| --- | --- |
| 方法 | 描述 |
| [Date()](http://www.w3school.com.cn/jsref/jsref_Date.asp) | 返回当日的日期和时间。 |
| [getDate()](http://www.w3school.com.cn/jsref/jsref_getDate.asp) | 从 Date 对象返回一个月中的某一天 (1 ~ 31)。 |
| [getDay()](http://www.w3school.com.cn/jsref/jsref_getDay.asp) | 从 Date 对象返回一周中的某一天 (0 ~ 6)。 |
| [getMonth()](http://www.w3school.com.cn/jsref/jsref_getMonth.asp) | 从 Date 对象返回月份 (0 ~ 11)。 |
| [getFullYear()](http://www.w3school.com.cn/jsref/jsref_getFullYear.asp) | 从 Date 对象以四位数字返回年份。 |
| [getYear()](http://www.w3school.com.cn/jsref/jsref_getYear.asp) | 请使用 getFullYear() 方法代替。 |
| [getHours()](http://www.w3school.com.cn/jsref/jsref_getHours.asp) | 返回 Date 对象的小时 (0 ~ 23)。 |
| [getMinutes()](http://www.w3school.com.cn/jsref/jsref_getMinutes.asp) | 返回 Date 对象的分钟 (0 ~ 59)。 |
| [getSeconds()](http://www.w3school.com.cn/jsref/jsref_getSeconds.asp) | 返回 Date 对象的秒数 (0 ~ 59)。 |
| [getMilliseconds()](http://www.w3school.com.cn/jsref/jsref_getMilliseconds.asp) | 返回 Date 对象的毫秒(0 ~ 999)。 |
| [getTime()](http://www.w3school.com.cn/jsref/jsref_getTime.asp) | 返回 1970 年 1 月 1 日至今的毫秒数。 |

String

chartCodeAt() //判断字符code

indexOf() //传字符串返回index

charAt() //传入index 返回字符

slice() //传入结束，开始（左闭右开）返回新字符串

split()

trim() //去空格

Dom

父节点是否有子节点： (父).hasChildNodes.(子节点)

NodeList：类数组querySelectorAll，本身没有文本节点

.childNodes(会包含文本节点换行符) firstChild lastChild firstElementChild

parentNode

nextSibling(会包含文本节点换行符)

previousSibling(会包含文本节点换行符)

（父）.appendChild(子) 放最后，从其他位置挪过去

.append(新text节点)

（父）.insertBefor (新，子) 插入 如果

（父）.replaceChild (新，子) 替换

包含HTMLCollection text

HTMLCollection: 内没有文本节点，类数组getElementsByTagName()

childElementCoount: 子元素个数（不包含文本节点和注释）

firstElementChild: 指向第一个元素：firstChild 的元素版

lastElementChild: 指向最后一个元素：lastChild 的元素版

nextElementChild: 指向第一个元素：nextSibling的元素版

previousElementChild: 指向最后一个元素：previousSibling的元素版

child：返回所有的子元素

（父）.removeChild (子节点) 可以删除文本

（子节点）.remove () 可以删除文本

classList: 获取class[] 添加样式可

以.add(‘demo’) .remove(‘demo’) .toggle(‘msg’)(删除，添加)

str.toLowerCase(变小写)

createElement(‘li’)

事件冒泡，事件捕获，  
 如：onclick先捕获后冒泡

事件绑定，事件监听，事件委托

onclick addEventListener 子委托父，回调里判断子

UI事件

load

error

select input选中

resize 窗口缩放

scroll 滚动打滚动

焦点事件

blur

focus

文本事件：

textinpupt D3级事件

鼠标事件和滚动事件

mousedown

mouseup

click

dbclick

可以按着点击组合

shiftKey altKey crtKey metaKey(win||cmd)

mouseWheel鼠标滚轮的滚动事件

键盘事件：

keydown

keyCode ASCII编码

code 按键英文名称（详细）

key 按键英文名称

keypress 按下字符键，可以判断是否输入汉字

keyup

移动端事件

单位 vw vh相当于比例

touchstart:

触摸

touchmove

滑动 preventDefault()可以组织滚动

touchend

移开

touchcancel

changedTouches

数组表示自上次触摸以来发生了什么改变的Touch对像的数组

clientX Y 相对于在视口的坐标

pageX Y 目标在页面的坐标

screenX Y 目标在屏幕中的坐标

identifier： 标识触摸的唯一ID

target: 触摸的DOM节点目标。

targetTouches

特定于事件目标的Touch对像的数组（特定）

touches

数组表示当前跟踪的触摸操作的Touch对像的数组（所有）

其他事件对像：

（只读属性）

bubbles: 表明事件是否冒泡

cancelable： 表明可以取消事件的默认改为

currentTarget: 其事件处理程序当前正在处理事件的那个元素

defaultPrevented ： 为true时表示已经调用了preventDefault() (DOM3级事件新增)

detail： 与事件相关的细节信息

eventPhase： 调用事件处理程序的阶段：1表示捕获阶段，2表示“处于目标”，3冒泡

preventDefalut: 取消事件的默认行为。如果cancelable为true

stopInmediatePropagation: 取消事件的进一步捕获或冒泡，同时阻止任何事件处理程序被调用（D3级）

stopPropagation: 取消事件的进一步捕获或冒泡，如果bubbles为true则可以使用这个方法

target: 事件的目标

trusted: 为true表示是浏览器生成的，为false表示事件是由开发人员通过js创建（D3）

type: 被触发的事件类型

view: 与事件关联的抽象视图，等同于发生事件的window对像。

组合拖拽事件：

元素的新位置 = 元素的旧位置+元素的偏移量

clientX Y 相当于目标元素到浏览器边缘的距离

获取元素初始位置：initX = demo.offsetTop； initX = demo.offsetLeft (相当于坐标系的)

<div id='demo' data-initx='0' data-inity='0'></div>

var demo = document.querySelector('#demo');

var x1, y1, x2, y2;

var data = demo.dataset;

var tag = false;

demo.addEventListener('mousedown', drag);

document.addEventListener('mousemove', drag);

document.addEventListener('mouseup', drag);

function drag(ev){

switch(ev.type){

case 'mousedown':

x1 = ev.clientX;

y1 = ev.clientY;

tag = true;

console.log('mousedown', x1, x2);

break;

case 'mousemove':

if(tag){

x2 = ev.clientX;

y2 = ev.clientY;

demo.style.transform = 'translate('+ (parseInt(data.initx)+(x2-x1)) +'px, '+ (parseInt(data.inity) + (y2-y1)) +'px)';

console.log('mousemove', y1, y2);

}

break;

case 'mouseup':

tag = false;

data.initx = parseInt(data.initx)+x2-x1;

data.inity = parseInt(data.inity)+y2-y1;

console.log('mouseup', y1, y2);

break;

}

}

对像：

构造函数：

\_proto\_ 继承父的

prototype 自己的

[].\_proto\_ = Array.prototype

new一个对像所经历的四个步骤：

创建一个新对像

将构造函数的作用域赋给新对像（因此this就指向了这个新对象）

执行构造函数的方法（为这个新对像添加属性）

返回新对像

定义一个构造函数（普通函数），会挂在到window上，当执行构造函数(普通函数)时，函数内的this.属性会挂载到window上。

工厂模式：

function createPerson(name, age, company){

let o = new obj

o.name = name; o.age = age; o.company=company

o.sayName=function(){console.log(name)}

return o;

}

原型链：

Object.definePropery(Person.prototype.girlFriend), {  
value:[‘石’, ‘大’]

enumerable: false,

config:false,

writable:false  
}

Person.prototype.constructor = Person

Person.prototype 原型

Person.prototype = {} 会覆盖person里的constructor。

原型的检测, 检测实例的构造函数： 构造函数.prototype.isPrototypeof( new 构造函数)

instanceof 检测所有先辈构造函数  
  
构造函数.prototype.isPrototypeof( new 构造函数) 与 instanceof 等价

构造函数. constructor. prototype (返回原型，这样取值可以兼容) 等不同于（对像.\_proto）

Object.getPrototypeOf(wanzhang) (返回原型，这样取值可以兼容) 等不同于（对像.\_proto）

原型动态继承：

原型重写时的动态问题

实例对象后，再修改他的构造函数（整个原型替换），实例对像的构造函数就不是实例对像的构造函数。

原生对像的原型：

Object:

Array:

Date：

组合模式：

属性写在构造函数里，方法写在原型里

动态原型模式：

把原型添加方法属性，写在构造函数里，并添加条件，使之动态赋值原型属性

寄生构造函数：

构造函数里写个new对象，并retrun，相当于工厂模式

稳妥构造函数：

构造函数内无自己的私有属性，而构造函数里所需要用的变量值都通过参数传递过去，在new对象时只用一次。

原型继承：

子构造函数原型继承父构造函数实体。然后new一个新的子实体对像

canvas:

只设置width，默认宽高比例默认为2：1

只有当body hmtl都设置100%时

1. 可画区域
2. 笔加颜色
3. 构思待纪绘画基本参数
4. 下笔

<canvas class='bubble' width='1000' height='1000'></canvas>

let canvas = document.querySelector('.bubble');

canvas.width = window.innerWidth;

canvas.height = window.innerHeight;

let canCon = canvas.getContext('2d'); //可画区域

canCon.fillStyle = 'red'; // 加颜料

canCon.arc(233, 233, 66, 0, Math.PI\*2); //构思确定参数

canCon.fill(); //下笔作画

call apply bind

函数只不过是在特点环境中执行代码的对象！

经典继承（伪造对像）call

all apply 立即执行，执行完后函数this恢复指向

bind 返回一个新的函数，该函数里的this永久指向bind后的对像

组合继承

call和原型继承一起，容易重复继承。

寄生式继承

function object(o){  
 function F(){}

F.prototype = o;

return new F();

}

function createAnother(original){  
 var clone = object(original);

clone.sayHi = function(){  
 console.log(‘hi’);

}return clone;

}

原型式继承

function object(o){  
 function F(){}

F.prototype = o;

return new F();

}

寄生组合继承

扫除了寄生的bug

function superType(name){

this.name = name;

this.color = [‘red’, ‘blue’, ‘green’];

}

SuperType.prototype.sayName = function(){console.log(this.name)}

function SubType(){  
 SuperType.call(this, name);

this.age = age;

}

function object(o){  
 function F(){}

F.prototype = o;

return new F();

}

function inheritPrototype(subtype, superType){  
 var prototype = object(superType.prototype) // 创建对像

prototype.constructor = subtype; // 增强对像

subtype.prototype = prototype; //指定对像

}

递归函数

自带缓存功能的递归函数，可用计算阶乘

function factorial(n){

if(!(isFinite(n) && n>0 && n==Math.round(n))){ //有限正整数

return NaN;

}

if(!(n in factorial)){

if(n==1){

return 1;

}else {

factorial[n] = n\*factorial(n-1); //计算结果缓存

}

}

return factorial[n]; //返回缓存结果

}

闭包和变量

闭包是指有权访问另一个函数作用域中的变量的函数

缺点：由于闭包会携带包含它的函数的作用域，因此会比其他函数占用更多内在。虽然像V8等优化后的js引擎会尝试回收被闭包占用的内存，但请大家还是要慎重使用闭包。

高阶函数

操作函数的函数，函数为参数，返回也是函数

function not(f){

return function(){

alert(this.name);

console.log(this.name);

let result = f.apply(this, arguments);

return !result;

}

}

let even = function(x){

return x%2 ===0;

}

let odd = not(even);

[1, 1, 3, 5, 5].every(odd);

// 带记忆力的函数

function memorize(f){

var cache = {}; //闭包

return function(){

var key = arguments.length + Array.prototype.join.call(arguments, ',');

if(key in caches){

return cache[key]

}else{

return cache[key] = f.apply(this, arguments);

}

}

}

function fn(n){

if(n<=1){

return 1

}else {

return n\*factorial(n-1);

}

}

let factorial=memorize(fn);

函数的柯理化

函数编程的重要思想，也是高阶函数中的下重要的应用

其含义：是给函数分步传递参数，每次传递部分参数，并返回一个具体的函数接收剩下的参数，这中间可嵌套多层这样的接收部分参数的函数，直至返回最后的结果。

function addCurrying(a){  
 return function(b){

return function(c){

return a+b+c

}

}

}

addCurrying(1)(2)(3)

百变柯理化参数

function local(sheng, shi, name){

alert(name+"是"+sheng+"省"+shi+"市的人")

}

function currying(func, args){

var arity = func.length;

var args = args||[];

return function(){

let \_args = [].slice.call(arguments)

Array.prototype.unshift.apply(\_args, args);

if(\_args.length<arity){

return currying.call(null, func, \_args);

}

return func.apply(null, \_args);

}

}

currying(local,)('黑龙江')('佳木司')('zzd');