**DOM(document object model)**  
console.log(document.parentNode);//null

document.getElementsByTagName(‘head’)[0].parentNode.removeChild(document.getElementsByTagName('head')[0]) //用于删除子节点//删除某一个元素，需要找到他的父元素

console.log(document.getElementsByTagName('body')[0].childNodes);//列出所有的子节点（包含空文本）

console.log(document.getElementsByTagName('body')[0].children);//列出所有子节点不包含空文本

console.log(document.getElementsByTagName('div')[0].hasChildNodes());//判断是否有子节点

console.log(document.getElementsByTagName('body')[0].innerText)//列出所有文本

console.log(document.getElementsByTagName('body')[0].innerHTML)//列出html标签

1.document.getElementsByTagName('input')[0].**setAttribute**('type','password')//设置某一个属性//pNode.setAttributes('id','pNode') //pNode1.id='pNode1'

2.console.log(document.getElementsByTagName('input')[0].**getAttribute**('type'))//获取某一个属性

3.console.log(document.getElementsByTagName('input')[0].**removeAttribute**('type'))//移除某一个属性

4.console.log(document.getElementsByTagName('input')[0].attributes)//列出所有的属性

5.console.log(document.getElementsByTagName('input')[0].hasAttribute('type'))//判断是否拥有某个属性

var pNode = document.createElement('p');pNode.innerHTML = '新节点/strHTML'

//1.父节点内部插入节点

document.getElementsByTagName('body')[0].**innerHTML**=document.createTextNode('p').innerText='123'

document.getElementsByTagName('body')[0].**innerHTML**='<p>456<p>'

2.尾部

document.getElementsByTagName('h1')[0].**appendChild**(**pNode**)//父节点内部插入子节点(只能是节点，不能为html)-尾部

3--removeChild()删除子节点   
4.replaceChild(新,老)替换子节点

document.getElementsByTagName('body')[0].**replaceChild**(pNode1,document.getElementsByTagName('h1')[0])//replaceChild(新,老)替换子节点

5.document.getElementsByTagName('body')[0].**insertBefore**(pNode1,document.getElementsByTagName('h1')[0])//在节点外部前边插入

document.getElementsByTagName('body')[0].**nodeName**//只读属性 元素节点的nodeName与标签名相同(大写);属性节点的nodeName与属性名相同;文本节点的nodeName始终是#text文档节点的nodeName始终是#document

document.getElementsByTagName('body')[0].**nodeValue**//元素节点的 nodeValue 是 undefined 或 null;文本节点的 nodeValue 是文本本身;属性节点的 nodeValue 是属性值

document.getElementsByTagName('body')[0].nodeType//元素 1;属性 2;文本3;注释8;文档9

for(var i =0;i<document.getElementsByTagName('h1').length;i++){

setTimeout(function(){//批量删除--使用定时器确保元素被删除

document.getElementsByTagName('body')[0].removeChild(document.getElementsByTagName('h1')[0])

},10)

}

document.**forms**.length//获取表单数目document.forms[0].name--返回第一个表单名字----.action地址

document.**images**.length//返回文档中图片数目document.images[0].id

document.getElementsByClassName('photolst')[0].getElementsByTagName('img').length可获取指定位置图片数目

document.**links**.length//返回文档链接数目--.href 返回链接

document.cookie //返回所有的cookie document.domain--返回加载文档的服务器名称

document.lastModified//返回文档最后一次修改时间 document.title 返回文档标题 document.URl 返回文档完整url document.getElementById('a').search--返回查询部分 即：？id=3

var x=document.getElementById("frm1");//button=submit frm1为表单ID

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

document.write(**x.elements[i].name+x.elements[i].value)**;//name为input的name;

document.write("<br>");

}

//body上绑定事件 event.**keyCode**--（onkeyup事件-event.**shiftKey**==1-shift键是否被按下）哪一个被按下 event.**button**--(onmousedown)哪一个鼠标键被按下，0左键，1中键，2右键；event.clientX--绑定在哪个上边相对于哪一个移动

//document.getElementById("select").**size**--可以设置select列表的显示长度-**selectedIndex**可显示被选中索引 **x.options[x.selectedIndex]**.text = "更换" x.options显示更多选项信息--x.**remove**[x.selectedIndex]删除指定选项

//document.getElementById('myTable').cols[0].innerHTML--获取指定行的信息 var x=document.getElementById('myTable').rows[0].cells;x[0].innerHTML;--获取指定cell的值

//x=document.getElementById('myTable').createCaption();x.innerHTML="表的标题";--为表格指定标题

//i=r.parentNode.parentNode.**rowIndex**;document.getElementById('myTable').**deleteRow**(i);//onclick（this） 传递表格(td)当前dom,

//var x=document.getElementById('myTable').**insertRow**(0);var y=x.**insertCell**(0);var z=x.insertCell(1);y.innerHTML="新单元格1";z.innerHTML="新单元格2";--添加一行

//x=document.getElementById('tr1').insertCell(0);x.innerHTML="The famous";--一行中新加单元格

var x=document.getElementById('myTable').rows[0].cells;x[0].colSpan="2";x[1].colSpan="6";--改变跨行列数  
总结：

1. var a = document.getElementById("test").getElementsByTagName("div");

--**链性操作的**  
2. var b =document.getElementById("test").**childNodes;**--**包含空文本**；

正则过滤  
 for(var i=0; i<b.length;i++){

 if(b[i].nodeName == "#text" && /\s/.test(b.nodeValue)){

 document.getElementById("test").**removeChild**(b[i]);

 }

 }

3. document.getElementById("test").**children去除空文本**  
4. document.getElementById('waterfall\_wraper').**firstChild**//可能空文本-- lastChild

5. document.getElementById('waterfall\_wraper').**firstElementChild**//除去文本和空文本的第一个元素—lastElementChild;

6. document.getElementById("test").parentNode= document.getElementByI

d("test").**parentElement;（ie标准**）;

7. document.getElementById("test").**offsetParent;获取body元素;**8. document.getElementsByTagName('h1')[0].**previousElementSibling**//不包含空文本//--nextElementSibling;

9. document.getElementsByTagName('h1')[0].**previousSibling**//包含空文本—nextSibling;

**继承**

//**每一个*函数*都有一个*prototype/\_\_proto\_\_/constructor*属性;每一个*对象*都有*\_\_proto\_\_/constructor属性*；new:*创建{},fn.prototype=obj.\_\_proto\_\_,fn.call(obj);*访问函数属性时，先从内部，再从构造函数指向的对象上找(\_\_proto)-原型链；每一个*构造函数*都有一个prototype从一个函数指向一个对象，这个对象的所有属性和方法，都会被这个函数的实例所继承-->函数的原型对象；1.构造函数.prototype.isPrototypeOf(实例)-判断prototype和实例的关系2. 实例.hasOwnproperty(‘name’)-判断是本地的还是prototype属性3.属性 in 实例-判断某个实例是否有某个属性（不管是本地还是继承）；constructor属性指向对象的构造函数，所有函数的构造函数最终指向Function  
call的参数是直接放进去的，第二第三第n个参数全都用逗号分隔，直接放到后面  obj.myFun.call(db,'成都', ... ,'string' )；**

**apply的所有参数都必须放在一个数组里面传进去  obj.myFun.apply(db,['成都', ..., 'string' ]);**

**bind除了返回是函数以外，它 的参数和call 一样。**

this就是函数运行时所在的环境对象  
1.函数还可以作为某个对象的方法调用，这时this就指这个上级对象。  
**function app(fn){**

**this.name = 'david'**

**fn()**

**console.log(111)**

**// setTimeout(function(){**

**// },1000)**

**}**

**function fn(){**

**this.name ='fn'**

**console.log(this.name)//如果有this.name ='fn' 则输出fn,无的话输出david**

**setTimeout(function(){**

**console.log(222)**

**},1000)**

**}**

**app(fn);//111 222 函数作为参数传递，不会阻断程序；setTimeout同样不会阻断程序**

2.所谓构造函数，就是通过这个函数，可以生成一个新对象。这时，this就指这个新对象。

3. apply()是函数的一个方法（也是对象），作用是改变函数的调用对象。它的第一个参数就表示改变后的调用这个函数的对象。因此，这时this指的就是这第一个参数。apply()的参数为空时，默认调用全局对象，也是先从函数内部找，再从调用对象上找。因此，这时的运行结果为0，证明this指的是全局对象。

var o ={name:'sss'}

function aa(){

// this.name = 'aaa'

console.log(name)

console.log(this.name)

}

aa.call(o)//有this.name 输出aaa 无输出sss  
继承实例：  
1. function Animal(){

this.species = "动物";

}

function Cat(name,color){

Animal.apply(this, arguments);

this.name = name;

this.color = color;

}

var cat1 = new Cat("大毛","黄色");

cat1.species

2. Cat.prototype = new Animal();-prototype被重写，以前编辑的被删除

　　Cat.prototype.**constructor** = Cat;- 每一个实例也有一个**constructor**属性，默认调用prototype对象的constructor属性。需要重新定向构造函数为Cat

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // **动物tip:定义在Cat上的prototype的方法继承后将被删除，需要定义在内部才可使用；**

3. function extend(Child, Parent) {  
  
　　　　var F = function(){};

　　　　F.prototype = Parent.prototype;

　　　　Child.prototype = new F();

　　　　Child.prototype.constructor = Child;

　　　　Child.uber = Parent.prototype;

}

extend(Cat,Animal);

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // 动物

4.拷贝继承

　function extend2(Child, Parent) {

　　　　var p = Parent.prototype;

　　　　var c = Child.prototype;

　　　　for (var i in p) {

　　　　　　c[i] = p[i];

　　　　　　}

　　　　c.uber = p;

}

extend2(Cat, Animal);

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // 动物

例如：function Animal(){

this.pro='pro-Animal';

this.species = "动物";

this.do=function(){

console.log('do-Animal');

}

}

Animal.prototype.eat=function(){

console.log('Animal');

}

function type(){

this.pro='type-Animal'

this.type='ttype';

this.do=function(){

console.log('do-type');

}

}

type.prototype.eat=function(){

console.log('type');

}

function Cat(name,color){

this.pro='Cat-Animal';

// Animal.call(this)

this.name = name;

this.color = color;

this.do=function(){

console.log('do-Cat');

}

}

Cat.prototype.eat=function(){

console.log('Cat');

}

Animal.prototype = new type();

Cat.prototype = new Animal();

var cat1 = new Cat("大毛","黄色");

cat1.do();  
5.非构造函数继承（创造构造函数）

var Chinese = {  
　　　　nation:'中国'  
　　};  
 var Doctor ={  
　　　　career:'医生'  
　　}

　function object(o) {

　　　　function F() {}

　　　　F.**prototype** = o;

　　　　return new F();

　　}

var Doctor = object(Chinese);

Doctor.career = '医生';  
6.深拷贝（拷贝基本类型）  
　　function deepCopy(p, c) {

　　　　var c = c || {};

　　　　for (var i in p) {

　　　　　　if (typeof p[i] === 'object') {

　　　　　　　　c[i] = (p[i].constructor === Array) ? [] : {};

　　　　　　　　deepCopy(p[i], c[i]);

　　　　　　} else {

　　　　　　　　　c[i] = p[i];

　　　　　　}  
　　　　}

　　　　return c;  
　　}

**Array(数组)**

var arr1 = [1,2,3],arr2=[4,5,6],arr3 = [11,22,33,44,55,66,'asd'],str='abcde',sum = 0;

//Object.**assign**('obj1','obj2')//合并两个对象，返回新对象，以第一个为基准返回覆盖对象，浅度覆盖—改变obj2的属性，返回的也改变  
可以用$.extend(true,object1, object2);实现深度复制

Array.**isArray**(arr3)//判断对象是否为数组，如果对象是数组返回 true，否则返回 false >=ie8

Array.from(str,function(val){console.log(val)})//from() 方法用于通过拥有 length 属性的对象或可迭代的对象来返回一个数组,如果对象是数组返回 true，否则返回 false。 >ie12

**Array.prototype.fn=function(a,b){return a.concat(b)};arr1.fn(arr1,arr2);//所有的数组可用；**

arr1.**concat**(arr2);//**array1.concat(array2,array3,...,arrayX)链接两个或多个数组**

arr3.**copyWithin**(2,0,2)//array.copyWithin(target, start, end(负数倒叙));指定位置拷贝元素到数组的另一个指定位置中；注意位数不足，会保留最后-添加数组内的元素

1.arr3.**some**(check);functioncheck(val,index){**return** val>44}//array.some(function(currentValue,index,arr),thisValue) >=ie9返回true或者false

2.arr3.**every**(check);function check(val,index){**return** val>44}//为每一个数组元素执行函数，如果有一个不满足返回false，否则返回true；空数组不会执行every，every不会改变原始数组

3.arr3.**find**(function(n){**return** n>33})//array.find(function(currentValue,index,arr), thisValue)-返回第一检测通过的元素

4.arr3.**findIndex**(function(n){**return** n>33})//array.findIndex(function(currentValue,index,arr), thisValue)-filter() 为每个数组元素都执行一次，检查到时返回该值索引，不在继续执行。不检查空数组，不改变原数组 >ie12

5.**filter**() arr3.filter(function(n){return n>33})//array.filter(function(currentValue,index,arr), thisValue)-filter() 方法创建一个新的数组并返回，新数组中的元素是通过检查指定数组中符合条件的所有元素。不检查空数组，不改变原数组 >ie9

1.arr3.forEach(function(n){sum+=n})//array.forEach(function(currentValue,index,arr), thisValue)-forEach() 为每个数组元素都执行一次回调函数，不检查空数组**(不会对数组的空元素检测['sf',11,22,33,,66])**，不改变原数组ie>9.0

arr3.map(function(val){return val+1})//array.map(function(currentValue,index,arr), thisValue) 为每个数组元素都执行一次回调函数，不检查空数组，不改变原数组ie>9.0

//数组的迭代方法：every、some、filter、forEach、map、**均不能使用break或者continue进行中断循环**。以上几个函数的参数都是：一个回调函数 和 一个this的指向 array.map(function(currentValue,index,arr), thisValue)

arr3.**fill**('123',3,8);//使用固定元素填充数组，array.fill(value, start, end)；//若果end不够长，会被截取掉-可以增加数组长度

//var arrTst =[];arrTst.length=24,arrTst.fill(0)

arr3.**includes**(33)//数组是否包含指定内容-字符串和数组不同，返回true/flase >ie14

arr3.**indexOf**('asd')//array.indexOf(item,start)-找到返回第一次出现的**位置**，找不到返回**-1**；>ie8

arr3.**lastIndexOf**('asd')//array.lastIndexOf(item,start)-从后向前找到返回第一次出现的位置，找不到返回-1；>ie8

arr3.**join**('-')//把数组按照指定的字符串分割成字符串;默认为','==array.toString()

arr3.**splice**(0,2,'12','34')//array.splice(index,howmany,item1,.....,itemX)删除的起始位置，删除数量，添加的元素

1.**arr3.pop()**//删除数组最后一个元素，并返回最后一个元素，改变数组长度；

2.**arr3.shift()-**删除数组的第一个元素，并返回第一个元素，改变长度。

3.**arr3.unshift(00)**//数组开头添加元素，返回长度ie>=8

4**.arr3.push(00)**//向数组末尾添加元素，并返回长度-length

arr1.**reduce**(function(total,val,index){return total+=val})//返回累加结果,空数组不计算 ie>=9.0

arr1.reduceRight(function(total,val,index){return total+=val})//从最后一位开始，返回累加结果,空数组不计算 ie>=9.0

arr3.reverse();//颠倒数组元素的顺序。

arr3.**slice**(0,2)//只截取两个元素；arr3.slice(start,end)-负数的话，倒叙，最后一个-1，以此类推

arr3.**sort**(function(a,b){return a-b})//升序排列数组，**必须指定函数**，否则40排在5前边

**Boolean()布尔**

false == NaN //false

false ==null //false

false ==new Boolean(NaN);//true

false == new Boolean(null);//true

false == ''//ture

false ==new boolean('');//true

false ==0//true

false =='fasd'//false

**Date()时间**

//4种方式获得时间--**new Date();new Date('2019-04-15 8:00:15');new Date(155533562929);new Date(2018,3,18,8,18,18,15);**

var d=new Date();

d.getFullYear();//从 Date 对象以四位数字返回年份。

d.getMonth();//从 Date 对象返回月份 (0 ~ 11)。

d.getDate();//从 Date 对象返回一个月中的某一天 (1 ~ 31)。

d.getDay();//从 Date 对象返回一周中的某一天 (**0 ~ 6**)。

d.getHours();//返回 Date 对象的小时 (0 ~ 23)。

d.getMinutes();//返回 Date 对象的分钟 (0 ~ 59)。

d.getSeconds();//返回 Date 对象的秒数 (0 ~ 59)。

d.getMilliseconds();//返回 Date 对象的毫秒(0 ~ 999)。

d.getTime();//返回 1970 年 1 月 1 日至今的毫秒数。

Date.parse('时间字符串');//返回1970年1月1日午夜到指定日期（必须是字符串）的毫秒数。==d.getTime()

d.setFullYear();//Date.setFullYear(year,month,day);

d.setMonth();//Date.setMonth(month,day)

d.**setDate**();//Date.setDate(day) ---**day:0 为上一个月的最后一天,-1 为上一个月最后一天之前的一天,如果当月有 31 天,32 为下个月的第一天-,-如果当月有 30天32 为下一个月的第二天**

d.setHours();//Date.setHours(hour,min,sec,millisec) ;

d.setMinutes();//Date.setMinutes(min,sec,millisec)

d.setSeconds();//Date.setSeconds(sec,millisec)

d.setMilliseconds();//setMilliseconds() 方法用于设置指定时间的毫秒字段。

d.setTime('毫秒数')//可正可负，1970至今时间

d1.toJSON()

//"2019-04-14T16:00:00.000Z"--不准

d1.toLocaleDateString()

//"2019/4/15"

d1.toLocaleString()

//"2019/4/15 上午12:00:00"

d1.toLocaleTimeString()

//"上午12:00:00"

d1.valueOf()//返回 Date 对象的原始值--毫秒数

**Math对象**  
var m = 18,n=-18,p=1.35,q=-1.35;

//Math 对象并不像 Date 和 String 那样是对象的类，因此没有构造函数 Math()。

Math.abs(n);//返回绝对值，n必须为一个数字，否则返回NAN,如果为null或者空字符串则返回0；

Math.ceil(p);//向上取整，取大的，如-1.35=-1；1.35=2;

Math.floor(p);//向下取整 取小的，如-1.35=-2；1.35=1;

Math.max(m,n,p,q);//返回最大值 如果没有参数，则返回 -Infinity。如果有某个参数为 NaN，或是不能转换成数字的非数字值(true/false/''可以转化为数字-字符串不可以)，则返回 NaN。

Math.min(m,n,p,q);//参数中最小的值。如果没有参数，则返回 Infinity。如果有某个参数为 NaN，或是不能转换成数字的非数字值(true/false/''可以转化为数字-字符串不可以)，则返回 NaN

Math.pow(4,3); //4\*4\*4

Math.random();//返回介于 0（包含） ~ 1（不包含） 之间的一个随机数。

Math.round();//四舍五入-方法可把一个数字舍入为最接近的整数。

**Number对象**

var n1 =1,n2=1.32;

//如果一个参数值不能转换为一个数字将返回 NaN (非数字值),false/true/空字符串可以转化，字符串不可以。

isNaN()//全局函数，传入1-false,null-false,''-false,undefinded-true;

n2.toFixed(3)//把数字转换为字符串，结果的小数点后有指定位数的数字,toFixed() 方法可把 Number 四舍五入为指定小数位数的数字。默认0，即四舍五入；

n2.valueOf();//返回一个 Number 对象的基本数字值

//Number.isInteger(): 用来判断给定的参数是否为整数。

//Number.isSafeInteger(): 判断传入的参数值是否是一个"安全整数"。

**String 对象**

var str ='string test';

str.charAt(0)//返回指定位置的字符，第一个字符位置为 0, 第二个字符位置为 1,以此类推.-s

str.charCodeAt(0);//返回字符串第一个字符的 Unicode 编码-115

str.concat('123');//string.concat(string1, string2, ..., stringX) 返回新的字符串--用'+'更直接

String.fromCharCode(115);//String.fromCharCode(n1, n2, ..., nX) 一个或多个 Unicode 值，即要创建的字符串中的字符的 Unicode 编码。逗号隔开

str.indexOf('est')//string.indexOf(searchvalue,start) 返回某个字符串首次出现的位置，找不到返回-1

str.lastIndexOf('est'); //倒叙检查，位置数不变,z最后出现的位置

str.includes(' ');//string.includes(searchvalue, start) -includes() 方法用于判断字符串是否包含指定的子字符串。一个空格和两个空格是不同的；true/false >=ie12

str.match(/t/g);//string.match(regexp) 无g返回一个--返回值数组，包含下标父str，有g返回所有（数组）--

str.repeat(100);//string.repeat(count) 复制字符串并把他们链接在一起-无空格

str.search(/t/);//string.search(searchvalue)返回Index位置，search() 方法用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串。

str.replace('tes','t')//string.replace(searchvalue,newvalue)-第一个参数可为正则表达式；

str.slice(1,2);//string.slice(start,end),slice(start, end) 方法可提取字符串的某个部分，并以新的字符串返回被提取的部分。

str.split(' ');//string.split(separator,limit),字符串转为数组

str.startsWith('t')//string.startsWith(searchvalue, start) -startsWith() 方法用于检测字符串是否以指定的子字符串开始。返回true/false

str.substr(0,3);//string.substr(start,length)-返回开始位置指定长度的字符串

str.substring(0,5)//string.substring(from, to);substring() 方法返回的子串包括 开始 处的字符，但不包括 结束 处的字符。

str.trim();//去除字符串两边的空格

**JavaScript 全局属性，方法**null == null;//true

//全局属性

NaN ==NaN;//false 请使用 isNaN() 来判断一个值是否是数字。原因是 NaN 与所有值都不相等，包括它自己。

Infinity;//代表正的无穷大的数值。

undefined;//指示未定义的值

//全局方法

escape('sdfas asdf');//对字符串编码

unescape('sdfas%20asdf')//对由 escape() 编码的字符串进行解码。

eval('console.log(2+3)')//eval(string)-eval() 函数计算 JavaScript 字符串，并把它作为脚本代码来执行。

isNaN('');//如果参数值为 NaN 或字符串(isNaN('')=false)、对象、undefined等非数字值则返回 true, 否则返回 false。

// 需要注意的是，isNaN() 函数其实并不能像它的描述中所写的那样，数字值返回 false，其他返回 true。

// 实际上，它是判断一个值能否被 Number() 合法地转化成数字。

// 这中间有什么区别呢，主要提现在一些特别的情况如下：

// 1、数字形式的字符串。例如 "123"、"-3.14"，虽然是字符串型，但被 isNaN() 判为数，返回 false。（"12,345,678"，"1.2.3" 这些返回 true）

// 2、空值。null、空字符串""、空数组[]，都可被Number()合法的转为0，于是被isNaN认为是数，返回false。（undefined、空对象{}、空函数等无法转数字，返回true）

// 3、布尔值。Number(true)=1,Number(false)=0，所以isNaN对布尔值也返回false。

// 4、长度为 1 的数组。结果取决于其中元素，即：isNaN([a])=isNaN(a)，可递归。例如isNaN([["1.5"]])=false。

// 5、数字特殊形式。例如"0xabc"、"2.5e+7"，这样的十六进制和科学计数法，即使是字符串也能转数字，所以也返回false。

Number(null);//-0;Number() 函数把对象的值转换为数字。

parseFloat('12.0asdf')//12

// 注意： 字符串中只返回第一个数字。

// 注意： 开头和结尾的空格是允许的。

// 注意： 如果字符串的第一个字符不能被转换为数字，那么 parseFloat() 会返回 NaN。

parseInt('080');//80

//如果 string 以 "0x" 开头，parseInt() 会把 string 的其余部分解析为十六进制的整数。

//如果 string 以 0 开头，那么 ECMAScript v3 允许 parseInt() 的一个实现把其后的字符解析为八进制或十六进制的数字。

//如果 string 以 1 ~ 9 的数字开头，parseInt() 将把它解析为十进制的整数。

// 注意： 只有字符串中的第一个数字会被返回。

// 注意： 开头和结尾的空格是允许的。

// 注意：如果字符串的第一个字符不能被转换为数字，那么 parseInt() 会返回 NaN。

|  |  |
| --- | --- |
| 运算符--!== | 值与类型均不等（不恒等于）例：x !== "5"—true |
|
| JavaScript 位运算符  位运算符工作于32位的数字上。任何数字操作都将转换为32位。结果会转换为 JavaScript 数字。   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 运算符 | 描述 | 例子 | 类似于 | 结果 | 十进制 | | & | AND | x = 5 & 1 | 0101 & 0001 | 0001 | 1 | | | | OR | x = 5 | 1 | 0101 | 0001 | 0101 | 5 | | ~ | 取反 | x = ~ 5 | ~0101 | 1010 | -6 | | ^ | 异或 | x = 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 | | << | 左移 | x = 5 << 1 | 0101 << 1 | 1010 | 10 | | >> | 右移 | x = 5 >> 1 | 0101 >> 1 | 0010 | 2 |   **RegExp 正则**  修饰符用于执行区分大小写和全局匹配:  修饰符 描述  i 执行对大小写不敏感的匹配。  g 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。  m 执行多行匹配。  方括号:  方括号用于查找某个范围内的字符：  表达式 描述  [abc] 查找方括号之间的任何字符。  [^abc] 查找任何不在方括号之间的字符。  [0-9] 查找任何从 0 至 9 的数字。  [a-z] 查找任何从小写 a 到小写 z 的字符。  [A-Z] 查找任何从大写 A 到大写 Z 的字符。  [A-z] 查找任何从大写 A 到小写 z 的字符。  [adgk] 查找给定集合内的任何字符。  [^adgk] 查找给定集合外的任何字符。  (red|blue|green) 查找任何指定的选项。  元字符  元字符（Metacharacter）是拥有特殊含义的字符：  元字符 描述  . 查找单个字符，除了换行和行结束符(只查找一次)可以匹配指定书目.....-匹配5个。  \w 查找单词字符。单词字符包括：a-z、A-Z、0-9，以及下划线, 包含 \_ (下划线) 字符。  \W 查找非单词字符。  \d 查找数字。  \D 查找非数字字符。  \s 查找空白字符。  空格符 (space character)  制表符 (tab character)  回车符 (carriage return character)  换行符 (new line character)  垂直换行符 (vertical tab character)  换页符 (form feed character)  \S 查找非空白字符。  \b 匹配单词边界。  \B 匹配非单词边界。  \b 元字符匹配单词边界。  在单词边界匹配的位置，单词字符后面或前面不与另一个单词字符直接相邻。请注意，匹配的单词边界并不包含在匹配中。换句话说，匹配的单词边界的长度为零。（不要与 [\b] 混淆。）  如果未找到匹配，则返回 null。  提示：\b 元字符通常用于查找位于单词的开头或结尾的匹配。  例子：  /\bm/ 匹配 "moon" 中的 'm'；  /oo\b/ 不匹配 "moon" 中的 'oo'，因为 'oo' 后面的 'n' 是一个单词字符；  /oon\b/ 匹配 "moon" 中的 'oon'，因为 'oon' 位于字符串的末端，后面没有单词字符；  /\w\b\w/ 不匹配任何字符，因为单词字符之后绝不会同时紧跟着非单词字符和单词字符。  \0 查找 NULL 字符。  \n 查找换行符。  \f 查找换页符。  \r 查找回车符。  \t 查找制表符。  \v 查找垂直制表符。  \xxx 查找以八进制数 xxx 规定的字符。  \xdd 查找以十六进制数 dd 规定的字符。  \uxxxx 查找以十六进制数 xxxx 规定的 Unicode 字符。  量词  量词 描述  n+  匹配任何包含至少一个 n 的字符串。  例如，/a+/ 匹配 "candy" 中的 "a"，"caaaaaaandy" 中所有的 "a"。  n\*  匹配任何包含零个或多个 n 的字符串。--b后边可以有一个或者多个o  例如，/bo\*/ 匹配 "A ghost booooed" 中的 "boooo"，"A bird warbled" 中的 "b"，但是不匹配 "A goat grunted"。  n?  匹配任何包含零个或一个 n 的字符串。--单词结尾一个或者0个  例如，/e?le?/ 匹配 "angel" 中的 "el"，"angle" 中的 "le"。  n{X}  匹配包含 X 个 n 的序列的字符串。--/\d{4}/--四位数  例如，/a{2}/ 不匹配 "candy," 中的 "a"，但是匹配 "caandy," 中的两个 "a"，且匹配 "caaandy." 中的前两个 "a"。  n{X,}  X 是一个正整数。前面的模式 n 连续出现至少 X 次时匹配。  例如，/a{2,}/ 不匹配 "candy" 中的 "a"，但是匹配 "caandy" 和 "caaaaaaandy." 中所有的 "a"。  n{X,Y}  X 和 Y 为正整数。前面的模式 n 连续出现至少 X 次，至多 Y 次时匹配。  例如，/a{1,3}/ 不匹配 "cndy"，匹配 "candy," 中的 "a"，"caandy," 中的两个 "a"，匹配 "caaaaaaandy" 中的前面三个 "a"。注意，当匹配 "caaaaaaandy" 时，即使原始字符串拥有更多的 "a"，匹配项也是 "aaa"。  n$ 匹配任何结尾为 n 的字符串。/is$/g;  ^n 匹配任何开头为 n 的字符串。/^Is/g;  ?=n 匹配任何其后紧接指定字符串 n 的字符串。/is(?= all)/g;  ?!n 匹配任何其后没有紧接指定字符串 n 的字符串。/is(?! all)/gi;  str.match(reg)-- match() 方法可在字符串内检索指定的值，或找到一个或多个正则表达式的匹配。该方法类似 indexOf() 和 lastIndexOf()，但是它返回指定的值，而不是字符串的位置。  该数组的第 0 个元素存放的是匹配文本，而其余的元素存放的是与正则表达式的子表达式匹配的文本。除了这些常规的数组元素之外，返回的数组还含有两个对象属性。index 属性声明的是匹配文本的起始字符在 stringObject 中的位置，input 属性声明的是对 stringObject 的引用。  //获取网址参数 var str = document.URL;  var patt1 = /(\w+):\/\/([^/:]+)(:\d\*)?([^# ]\*)/;  arr = str.match(patt1);  for (var i = 0; i < arr.length ; i++) {  //document.write(arr[i]);  //document.write("<br>");  }  var proStr=str.match(/\??\&?\w+\=\w+/g),obj={};  proStr.forEach(function(val){  var strB = val.substring(1);  obj[strB.substr(0,strB.indexOf('='))]=strB.substr(strB.indexOf('=')+1);  })  obj  {rnd: "310", name: "home"}  var str = document.URL;  var patt1 = /(\w+):\/\/([^/:]+)(:\d\*)?([^# ]\*)/;  arr = str.match(patt1);  var proStr,obj={};  //正则？？？中文不显示  // var proStr=str.match(/\??\&?\w+\=(?:\w+|[\u4E00-\u9FA5]{0,})/gi),obj={};  // console.log(proStr)  // proStr.forEach(function(val){  // //console.log(val)  // var strB = val.substring(1);  // obj[strB.substr(0,strB.indexOf('='))]=strB.substr(strB.indexOf('=')+1);  // })  proStr = str.substring(str.indexOf('?')+1);  var proArr = proStr.split('&');  proArr.forEach(function(val){  var valArr = val.split('=');  obj[valArr[0]]=valArr[1]  })  obj  {rnd: "310", name: "home", wd: "%E5%8F%B0%E6%B9%BE%E5%9C%B0%E9%9C%87"} |  |
|  |  |

**浏览器对象模型 (BOM)**window.innerHeight - 浏览器窗口的内部高度(包括滚动条)

window.innerWidth - 浏览器窗口的内部宽度(包括滚动条)

8=>ie>=5

document.documentElement.clientHeight

document.documentElement.clientWidth

window.open() - 打开新窗口

window.close() - 关闭当前窗口

window.moveTo() - 移动当前窗口

window.resizeTo() - 调整当前窗口的尺寸  
screen.availWidth - 可用的屏幕宽度

screen.availHeight - 可用的屏幕高度

**ajax**

var xmlhttp;

xmlhttp=window.XMLHttpRequest?new XMLHttpRequest():new ActiveXObject('Microsoft.XMLHTTP');

xmlhttp.open('GET','http://wxweb.socian.com.cn/order/getproductlist?mobile=1312068167',true);

// 如果需要像 HTML 表单那样 POST 数据，请使用 setRequestHeader() 来添加 HTTP 头。然后在 send() 方法中规定您希望发送的数据

xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");

xmlhttp.send();//xmlhttp.send('mobile=1312068167&');-POST

xmlhttp.onreadystatechange=function(){

console.log(xmlhttp.readyState+' '+xmlhttp.status)

if(xmlhttp.readyState==4&&xmlhttp.status==200){//readyState:0-初始化，1-建立连接，2-请求接受，3-请求处理，4-请求完成返回；status:200-ok,400-未找到页面；

// console.log(xmlhttp.responseText)//获得字符串形式的响应数据-返回数据

// console.log(xmlhttp.responseXML)//获得 XML 形式的响应数据。返回NUll

console.log(xmlhttp.response)//返回数据

}

}

**instanceof**  
instanceof运算符用来判断一个构造函数的prototype属性所指向的对象是否存在另外一个要检测对象的原型链上

基本类型的数据是没有属性和方法的（Number,String,Boolean、undefined、null存储在栈中），引用类型(Object,function,Array—存储在堆中)才会有，但是为什么String类型却又那么多方法呢？  
值类型**：1、空间固定；2，复制本身；3，typeof 检测；4，基本数据类型是值类型**引用类型：**1，空间不固定2，保存复制的是指针3，instanceof检测4，new构造出的是引用类型；**

 // 利用typeof来区分

 typeof undefined // undefined

 typeof 'abc' // string

 typeof 123 // number

 typeof true // Boolean

typeof console.log //function（引用类型）

 // typeof 区分不出来引用类型（除了函数）

 typeof {} // object

 typeof [] // object

 typeof null // object

Typeof RegExp//object

function setName(obj) {

obj.name = "Nicholas";

obj = new Object();

obj.name = "Greg";

}

var person = new Object();

person.name = 'aa'

setName(person);

person.name; //"Nicholas"

如果 person 是按引用传递的，那么 person 就会自动被修改为指向其 name 属性值为"Greg"的新对象。但是，当接下来再访问 person.name 时，显示的值仍然是"Nicholas"。

这说明即使在函数内部修改了参数的值，但原始的引用仍然保持未变。实际上，当在函数内部重写 obj 时，这个变量引用的就是一个局部对象了。而这个局部**对象会在函数执行完毕后立即被销毁。**

强制类型转换  
 // 字符串拼接

var a = 100 + 10 //100

var b = 100 + '10' //10010

// == 运算符

100 == '100' //true

0 == '' //true

null == undefined //true

// 语句

var a = true

if(a){}

var b = 100

if(b){} // 把数字转换为true

var c = ''

if(c){} // 把空字符串转换为false，有值的字符串转化为true

// 逻辑运算

console.log(10&&0); // 0 把10转换成true

console.log('' || 'abc'); // 'abc' 把空字符串转换为false

console.log(!window.abc); // window.abc是undefined 把非undefined转换成true

false == undefined//false  
 false == null//true

//判断一个变量会被当做true还是false

var a = 100

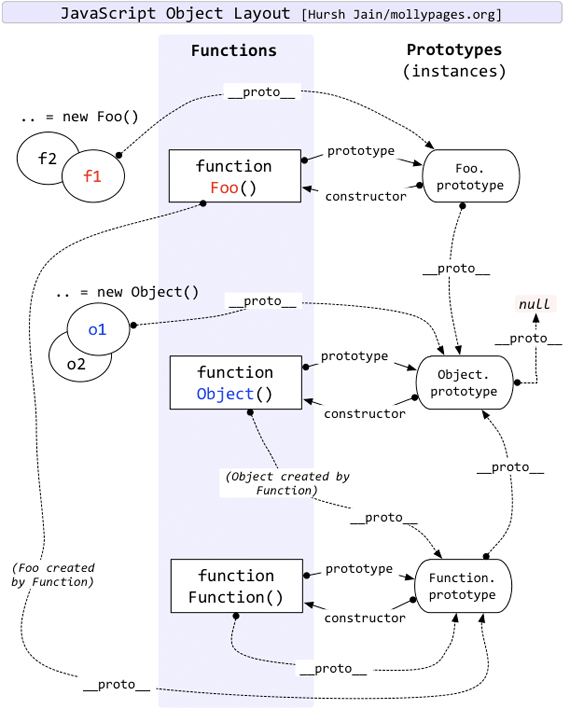
console.log(!!a); // true  
在 JavaScript 中, constructor 属性返回对象的**构造函数**。 返回值是**函数的引用，不是函数名**;   
所有 **JavaScript 对象都有 \_\_proto\_\_ 属性**，**只有 Object.prototype.\_\_proto\_\_ === nul**l ；

**构造函数的 prototype 属性指向它的原型对象，而构造函数实例的 \_\_proto\_\_ 属性也指向该原型对象；真正的语义是检查 b.prototype 是否在 a 的原型链上，仅此而已  
str 的原型链：**

**str ---> String.prototype ---> Object.prototype**

**String 的原型链：**

**String ---> Function.prototype ---> Object.prototype**



instanceof的高级用法我把它概括如下: 1.所有对象和函数 instanceof Object  //true     （因为JS万物皆对象，函数也是对象） 2.所有函数 instanceof Function  //true      （这个很好理解，包括普通函数和构造函数） 3.除Object和Function之外的构造函数 instanceof 自身  //false     （因为构造函数的原型链上只有Function.prototype和Object.prototype而没有它们自身的prototype，这一点很不容易理解！

string(原始资料类型) , string(原始资料类型) , object(包装对象)

**因为: 这是JS中的设计。** 原始资料类型的方法与属性是"借"来的 一个原始的资料类型值，并没有如对象会有属性或方法， 原始的资料类型在运算时用的属性与方法，是向包装对象"借来"的用的， 所以原始资料类型是可以向 new String() 或者 new Number() 借来所有的方法。但是自己本身却没有属性和方法。

var string = "aaa";

var string2 = string.substring(0)); //aaa

基本类型的确没有方法和属性，因为一旦创建了属性或者方法就会被立刻销毁，但是为了方便程序员对这类的基本类型数据比较方便的操作，在底层做了一些工作，其实这段代码相当于：(其实应该是中间变量 string-- string1操作 string2)

var string = new String("aaa");

var string2 = string.substring(0);

string = null;

(1)创建一个String类型的实例(使用 new 操作符创建的引用类型的实例，在执行流离开当前作用域之前都一直保存在内存中。而自动创建的基本包装类型的对象，则只存在于一  
行代码的执行瞬间，然后立即被销毁 )

(2)在实例上调用指定方法

(3)销毁该实例

经过这三步，字符串值就变得和对象一样了，在Boolean  Number中也是这样

String  Boolean  Number 这三种类型也叫做  基本包装类型