console.log()

alert() 警告消息框

confirm() 确定消息框

prompt() 提示消息框······就是专门来给用户提供输入窗口的

函数：把完成特定功能的一段代码抽象出来，使之成为程序中的一个独立实体，起个名字（函数名）。可以在同一个程序或其他程序中多次重复使用(通过函数名调用)。（当调用时执行的可重复使用的代码块。）

无参函数的声明：

function 函数名(){

函数体；

}

注：函数体内，外面怎么写JS代码，这里也怎么写。

功能：函数可以封装任意多条语句，而且可以在任何地方，任何时候调用执行。

注：封装函数的时候，函数名=标识符

必须见名思意，当前函数的函数名必须体现处函数的功能性

函数分类：

内置函数 （系统函数、官方函数）

alert() pareInt() document.write()

自定义函数

用户根据实际的需求，自己封装一个函数。

有参函数的声明：

function 函数名(形参···){

函数体；

}

作用域

全局作用域

全局作用域作声明的变量 全局作用域

局部作用域 函数

局部作用域声明的变量 局部变量，局部变量的生命周期和生效的范围，都是声明该变量的函数区域。当函数调用完成以后，就直接销毁。

如果全局作用域和局部作用域重名，采用就近原则。以前访问这个变量离哪个作用域近，就是哪个作用域。

递归概念：

函数可以自己调用自己，称为递归调用。

注：我们可以写出递归，但我们不知道他是如何得出结果的。

方法：

1. 首先找出临界值，即无需计算，获得的值；

2. 找这一次和上一次的关系；

3. 假设当前函数已经可以使用了，调用自身计算上一次的运行结果，再写出这次的运行结果。

特点：

1. 必须有参数；

2. 必须有return.

注：递归会在短时间内，使内存剧增。

事件

事件驱动函数：在和页面交互的过程中所调用的函数，该函数被称之为事件驱动函数。

事件：和页面交互的行为称之为事件。比如：鼠标点击某个按钮时（onclick）、鼠标浮动或离开到某个区域时（onmouseover、onmouseout）、文本框获取焦点和失去焦点时（onfocus、onblur）等。

window.onload在页面加载完成后。

数组

创建方法：

1.使用new运算符创建数组

参数：是我们要在数组中存储的数据

参数：如果当前的参数，只有一个，并且该参数是数字的话。表示数组长度的数组。

var arr = new Array(1,true "hello");

2.省略new运算符创建数组

var arr = Array(1,true "hello");

3.通过常量创建数组

注：1.在数组中存储的数据，一般情况下叫做数组中的元素。

2.数组中的访问记录，通过下标完成，下表也索引，从0开始。

例：var arr = [1,2,3];

栈：先进后出

push()

格式：数组.push(元素…);

功能：给数组末尾添加元素

参数：我们要添加的元素，参数个数随意。

返回值：添加元素以后，数组的长度。

pop()

格式：数组.pop();

功能：移除数组末尾的最后一个元素

返回值：移除的元素

队列：先进先出

push()

shift()

格式：数组.shift();

功能：从数组的头部取下一个元素

返回值：取下的元素

unshift()

格式：数组.unshift(参数...);

功能：从数组的头部插入元素

参数：我们插入数组的元素，数组的个数随意

返回值：插入元素以后，数组的长度。

数组的方法

concat()

格式：数组1.concat(数组2);

功能：将两个数组，合并成一个新数组，原数组并不会改变

slice()

格式：数组.slice(start,end);

功能：基于当前数组获取指定区域元素并创建一个新数组。原数组不会改变。

参数：start开始获取区域的下标，end结束获取区域的下标，不包括end下标位置的元素。

返回值：指定区域元素生成的新数组。

splice() 可以完成删除，插入，替换操作

参数：参数1 截取的开始下标

参数2 截取的长度

参数3 在截取的开始下标位置，我们要插入的元素，插入的元素个数随意。

返回值：截取掉的元素，组成的数组。

注：会对原数组进行修改

插入功能实现：参数2设为0，此时返回值为空，原数组新增元素。

替换功能：先删除，然后再插入

join()

格式：数组.join(拼接符);

功能：使用拼接符将数组中元素拼接成字符串

参数：拼接符

返回值：拼接好的字符串

数组排序

系统提供的排序方法

reverse() 逆向排序

格式：数组.reverse();

sort()

格式：数组.sort();

功能：将数组中的元素升序排序

注：sort默认按照字符串进行排序。

要求：一般情况下，我们需要自己去编写排序算法，系统提供给我们的排序函数，用的比较少。

冒泡排序

原理：前后两个数两两进行比较，如果符合交换条件，交换位置。直到所有的数据排序完成，结束比较。

每一轮比较后，得出当前轮的最大数

var arr = [9,8,7,6,5,4,3,2,1];

for (var i=0;i<arr.length;i++) { //比较的轮数

//每一轮比较的次数

for (var j = 0; j<arr.length-i-1;j++) {

//判断前后两个数是否符合交换条件

if (arr[j]>arr[j+1]) {

var tmp = arr[j+1];

arr[j+1] = arr[j];

arr[j] = tmp;

}

}

}

alert(arr);

选择排序法（打擂台法）

原理：通过比较首先选出最小的数放在第一个位置上，然后在其余的数中选择次小数放在第二个位置，以此类推，直到所有的数成为有序序列。

var arr = [9,8,7,6,5,4];

for(var i = 0;i<arr.length-1;i++) {

for(var j = i+1;j<arr.length;j++) {

if(arr[i]>arr[j]) {

var tmp = arr[i];

arr[i] = arr[j];

arr[j] = tmp;

}

}

}

alert(arr);

作用域问题

申明在当前函数中的变量和当前函数的形参，作用域都限制在该函数内。当函数调用结束后，这些变量和形参会被释放。

数组 复合数据类型/引用数据类型

注：所有的复合数据类型都不是直接存储在我们函数中的，存储在堆段。

函数运行的内存空间是预先分配好的。一旦被分配好内存空间，就不能进行改变了。

所有的复合数据类型，存储的都是门牌号，也就是地址。所以这类型的数据，我们叫做引用数据类型。

字符串：将JS中所有的单引号或者双引号括起来得都叫字符串

字符串的创建方式：

1.使用new运算符创建

例：var str = new String("hello");

alert(typeof str); //object对象有属性和函数。

2.new可以省略

例：var str = String("hello");

alert(typeof str);

3.通过常量创建数组

例：var str = "hello";

alert(typeof str);

字符串的属性

length 返回的是当前字符串中字符的个数

访问字符串的字符

charAt()

格式：字符串.chartAt(下标);

返回值：对应下标的字符

例：alert(str.charAt(1));

注：我们可以直接通过字符下标去访问该字符

例：alert(str[1]);

注：ECMAScript中的字符串是不可改变的，也就是说，字符串一旦创建，它们的值就不能改变。要改变某个变量保存的字符串，首先要销毁原来的字符串，然后再用另一个包含新值的字符串填充该变量。

例：var str = "hello";

// str[1] = "x"

str = "hxllo";

alert(str);

注：+号可以作为字符串拼接符存在。

例：var str = "hello"+"world";

alert(str);

字符串函数

big() 用大号字体显示字符串

blink() 显示闪动字符串（IE下无效）

bold() 使用粗体显示字符串

fixed() 以打印机文本显示字符串

strike() 使用删除线来显示字符串

fontcolor() 使用指定颜色来显示字符串

fontsize() 使用指定尺寸来显示字符串

link() 将字符串显示为链接

sub() 把字符串显示为下标

sup() 把字符串显示为上标

以上均在document.write()中使用

charCodeAt()

格式：字符串.charCodeAt(下标);

返回值：返回字符串中对应下标字符的ASCII码值

String.fromCharCode()

格式：字符串.String.fromCharCode(ASCII码值);

参数：ASCII码值，个数任意

返回值：ASCII码值对应字符组成的字符串

concat()

格式：字符串1.concat(字符串2);

返回值：拼接成的字符串，生成新字符串。

注：一般情况下很少使用，使用字符串拼接符 +。

字符串查找方法

indexOf()

格式：字符串.indexOf(字符串，开始查找的位置);

返回值：如果在字符串中查找到了字串第一次出现的位置，返回字串出现的位置，否则没有查找到，返回-1；

lastIndexOf()

格式：字符串.lastIndexOf(字串);

返回值：如果在字符串中查找到了字串最后一次出现的位置，返回字串出现的位置，否则没有查找到，返回-1；

search() 正则匹配

参数可以是正则表达式

注：正则表达式可以添加修饰符，i代表忽略大小写，g代表全局匹配

replace() 替换字符串

格式：字符串.replace(匹配的字符串/正则表达式，替换成的新字符串);

返回值：替换完成以后生成的新字符串。

例：var str = "how are are you";

alert(str.replace("are","old are")); //将原数组中的are替换成old are ，但是若原数组有中连续相同的字符，此时是无法全部替换的

要想替换所有符合条件字符串，就必须使用正则表达式完成。

例：上述例子中要想实现可以写成

alert(str.replace(/are/g,"old are"));

substring()

格式：字符串.substring(start,end);

作用：字符串提取，在指定范围内，提取字符串，生成新字符串。

返回值：生成新字符串。注：不包含结束位置的。

split() 字符串分割

格式：字符串.split(分隔符，生成的数组的长度);

返回值：通过分割符，分割成的装有子串的数组。

例： var str = "This is a box";

var arr = str.split(" ",2); //输出This is

注：1.分割符，整体

2.会分割出空字符串

3.如果分割符是空字符串""，那么我们字符串会分割成单个字符

4.字符串=>数组 split 数组=>字符串 join

例： var str = "This is a box";

1.var arr = str.split(" ");//输出This is,a box (两个空格)

2.var arr = str.split(" ");//输出This,,is,a box (会多出一个空字符串)

3.var arr = str.split("");//输出T,h,i,s, , ,i,s,a, ,b,o,x

alert(arr);

4.var arr = str.split("");

alert(arr);

alert(arr.join("="));//输出T=h=i=s= = =i=s=a= =b=o=x

字符串大小写方法

toLowerCase() 全小写

toUpperCase() 全大写

例：var str = "HELLO world";

alert(str.toLowerCase()) //全小写

alert(str.toUpperCase()) //全大写

ECMA5 第五版

严格模式

注：从字面意思理解，即这种严格模式下，浏览器会对JS的要求更加的苛刻。

"use strict";写在哪个作用域下，这个作用域下所有的代码都遵从严格模式

注：不要轻易在全局范围开头增加"use strict";建议在作用域内使用。

例：function m1() {

max = 10; //如果在给变量赋值的时候，没有使用var声明该变量，那么当成全局变量进行处理。

}

m1();

alert(max); //输出结果为10

当声明严格模式时

例：function m1() {

"use strict";

max = 10;

}

m1();

alert(max); //控制台会报错，只有当将max = 10;改为var max = 10不会再报错。;

ECMA5中新增的数组方法

indexOf() 数组

格式：数组.indexOf(数组，index);

使用方法与字符串的方式一致。

forEach()

格式：数组.forEach(function(item,index,array) {

item 当前遍历到的元素

index 当前遍历到的下标

array 当前数组

});

功能：遍历数组

var arr = [10,20,30];

/\*arr.forEach(function(item,index,array) {

alert((item+", "+index+", "+array))

})\*/

或者：

arr.forEach(alert);

map 映射 遍历—>操作—>返回

例：var arr = [20,30,40,50];

var newArr = arr.map(function(item,index,array) {

return item+2;

})

alert(newArr); //22,32,42,52

reduce 归并

例：var arr = [10,20,30,40,50];

var res = arr.reduce(function(pre,next,index,array) {

/\* pre 上一次遍历return后面的值

next 当前遍历到的元素 \*/

alert(pre + ", " + next);

return pre + next;

})

alert(res);//输出结果依次为10 20—>30 30—>60 40—>100 50—>150 最终结果为数组所有元素的和

filter 过滤

var res = [10,20,30,40,50].filter(function(item,index,array) {

return item>30; //找出所有大于30的值

})

alert(res);//输出结果为40，50

some 某些

判断return后面的表达式，在当前数组中是否成立。如果成立true，否则false

注：如果匹配成功，则返回true，后面不再进行遍历

例：var arr = [10,20,30,40,50];

var res = arr.some(function(item,index,array) {

alert(item); //输出结果只有10 20 30，后面不再输出，原因见“注”

return item == 30;//判断数组中是否存在30

//判断return后面的表达式，在当前数组中是否成立。如果成立true，否则false

})

alert(res);//输出结果为true

every 跟some一样，但是要求每一项都符合，才返回true，有一项不符合就返回false

注：如果判断元素有不符合条件的，返回false，直接终止循环

例：var arr = [10,20,30,40,50];

var res = arr.every(function(item,index,array) {

alert(item); //输出结果只有10，后面不再输出，原因见“注”

return item >20;//判断数组中是否每一项都大于20

})

alert(res);//输出结果为false

Math对象 用于执行数学任务

Math常用的属性

Math.PI 约等于3.1415

Math对象常用的函数

Math.round() 四舍五入

例：alert(Math.round(3.5));

Math.random() 0~1之间的随机数 //每次输出结果都是0~1之间的随机数

Math.max() 返回较大数

Math.min() 返回较小数

例：alert(Math.max(10,20,30)); //输出30，Math.min()同理

Math.abs() // 返回绝对值

例：alert(Math.abs(-10)); //数据结果为10

Math.ceil() 向上取整

例：alert(Math.ceil(3.1)); //输出结果为4

Math.floor() 向下取整

例：alert(Math.floor(3.9)); //输出结果为3

Math.pow(x,y) 求x的y次方 x是底数 y是指数

例：alert(Math.pow(2,5)); //输出结果为32

Math.aqrt() 开平方

例：alert(Math.aqrt(4)); //输出结果为2

Math对象勾股函数

参数：都应该是弧度。 Math.PI = 180弧度

1弧度 = Math.PI / 180;

Math.sin()/cos()/tan()； //正弦/余弦/正切

例：求30度的正弦值 alert(Math(30\*Math.PI / 180));

对象

对象也是数据类型

分析：数据存储的历程

变量（单个数据）=> 数组（批量数组）=> 对象（数据，函数）

注：什么是对象，其实就是一种类型，即【引用】类型。而对象就是【引用类型】的实例。在ECMAScript中引用类型是一种【数据结构】，用于将【数据和功能】组织在一起。

对象的创建

1.使用new运算符创建对象

var person = new Object();

//给该对象添加数据/属性

person.name = "xxx";

//给该对象添加函数、方法

person.showName = function() {

alert(person.name);

}

//如果我们想要访问上述对象的属性和函数

alert(person.name);

//调用对象的方法

person.showName();

2.new运算符可以省略

var person = Object();

3.使用常量/字面量去创建对象

var person = {};

person.name = "xxx"; 或者person【name】 = "xxx";

person.showName = function() {

alert(person.name);

}

person.showName();

日期对象 Data

日期的创建方式

参数：如果我们不传参数，默认获取的就是当前系统的时间。

注：我们所用的时间都是根据格林尼治时间，中国北京处于东八区

var d = new Date();

alert(d); //可输出当前日期时间

生成日期对象的参数可以是：

Date("2015/08/22");

Date("2015-08-22");

Date(2016,04,13,14,34);

参数可以是毫秒数，计算从1970年开始+毫秒数，计算出这个时间。

1秒 = 1000毫秒

以1970年1月1日0时0分0秒 为参照物。1970年是Unix的诞生日。

日起对象格式化方法

格式：日期对象

box.toDateString(); //以特定的格式显示星期几、月、日和年

box.toTimeString(); //以特定的格式显示时、分、秒和时区

box.toLocaleDateString(); //以特定地区格式显示星期几、月、日和年

box.toLocaleTimeString(); //以特定地区格式显示时、分、秒和时区

box.toUTCString(); //以特定的格式显示完整的UTC日期





凡是不带UTC的返回系统当前时间，凡是带UTC的返回格林尼治时间

获取一周中的某一天时，从0开始，星期0代表周日。只能获取，不能设置

getDay();

获取月份的时候，从0开始。

set/getMonth()

d.set/getTime(); 设置/获取当前日期距离1970年的毫秒数

例：var d = new Date();

alert(d.gerTime()); //可以获取当前时间的毫秒数

var d2 = new Date(毫秒数);

alert(d2); //可以得到当前的时间

Date.parse("2015-08-24");

参数：是一个日期格式的字符串

功能：返回这个日期，距离1970年的毫秒数

例：var time = Date.parse("2015-08-24");

var d = new Date(time);

alert(d);

定时器 setInterval();

格式：setInterval(函数/匿名函数,毫秒数);

serInterval(function() {

//我们想要去做的代码

}，毫秒数);

功能：每隔所传参数的毫秒数，就调用一次所传参数的函数。

返回值：当前页面上对于这个定时器的唯一标识，定时器的ID。

例：<script type="text/javascript">

var i = 0;

// function show() {

// document.write(i++ +"<br/>");

// }或者写成

var show = function() {

document.write(i++ +"<br/>");

}

window.onload = function() {

var oBtn= document.getElementById("btn");

oBtn.onclick = function() {

// setInterval(show,1000);//函数传参 或者写成

setInterval(function() {

document.write(i++ +"<br/>");

},1000);

}

}

</script>

<body>

<input type="button" id="btn" value="按钮">

</body>

clearInterval() 取消定时器

参数：定时器的ID

例：上述例子当i为5时，取消定时器，只需将上述中的

setInterval(function() {

document.write(i++ +"<br/>");

},1000);

改为

var timer = setInterval(function() {

document.write(i++ +"<br/>");

if(i == 5) {

clearInterval(timer);

}

},1000);

innerHTML 标签间的所有内容

如果在innerHTML包含标签，标签会被识别，并且会被解析，呈现对应的效果。

例：

<script type="text/javascript">

window.onload = function() {

var oDiv = document.getElementById("div1");

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

btn.onclick = function() {

//获取div标签间的内容

// alert(oDiv.innerHTML);

//弹窗会显示<em>斜体</em>

// 设置div标签间的内容

oDiv.innerHTML = "<h1>我是替换文本</h1>"

}

}

</script>

</head>

<body>

<div id="div1"><em>斜体</em></div>

<input type="button" id="btn" value="按钮">

</body>

秒表实现

<script>

// 可以将查找标签节点的操作进行简化

function $(id){

return document.getElementById(id);

}

window.onload=function(){

// 点击开始时，开始计数

var count = 0; //开始技术以后，累加的总秒数

var timer = null;

$("start").onclick = function(){

timer = setInterval(function(){

count++;

// 需要改变页面上时分秒的值

$("id\_S").innerHTML = showNum(count % 60);

$("id\_M").innerHTML = showNum(parseInt(count/60) % 60);

$("id\_H").innerHTML = showNum(parseInt(count/3600));

},1000);

}

// 暂停

$("pause").onclick = function() {

clearInterval(timer);

}

// 停止 停止计数，数据清零<1>数据清零<2>页面展示数据清零

$("stop").onclick = function() {

clearInterval(timer);

// <1>数据清零

count = 0;

// <2>页面展示数据清零

$("id\_S").innerHTML = "00";

$("id\_M").innerHTML = "00";

$("id\_H").innerHTML = "00";

}

}

// 处理单个数字

function showNum(num) {

if(num<10) {

return "0"+num;

}else{

return num;

}

}

</script>

</head>

<body>

<div id="div1">

<div id="count1">

<span id="id\_H">00</span>

<span id="id\_M">00</span>

<span id="id\_S">00</span>

</div>

<input value="开始" type="button" id="start">

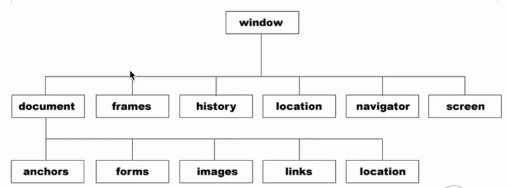
<input value="暂停" type="button" id="pause">

<input value="停止" type="button" id="stop">

</div>

</body>

BOM(Browser Object Model) 浏览器对象模型



window对象及下属属性方法

系统对话框：

浏览器可以通过调用系统对话框，向用户显示信息。

系统提供了三个函数，可以完成系统对话框的操作。

注：window下的函数，都可以省略window直接去调用

alert() 直接弹出警告框

参数，警告框上显示的内容

confirm() 弹出警告框

功能：弹出一个带有确定和取消按钮的警告框

返回值：如果点击确定，返回true；点击取消，返回false

prompt()

功能：弹出一个带输入框的提示框

参数 第一个参数：要在提示框上显示的内容

第二个参数：输入框内默认的值

返回值 点击确定：返回值是输入的内容

点击取消：返回值是null

window.open()

可以导航到一个特定的URL，也可以打开一个新的浏览器窗口

参数可以是：

1.要加载URL

例：open("http://www.baidu.com"); //新建页面并打开百度

2.窗口的名称或者窗口的目标

例：open("http://www.baidu.com","baidu"); //新建页面并命名窗口并打开百度

open("http://www.baidu.com",\_parent); //在本页窗开打开，\_blank是新建

3.一串具有特殊意义的字符串



例：open("http://www.baidu.com",'width=400,height=400,top=200,left=200');

即：设置当前打开窗口的新参数

opener

例：新建两个页面

父窗口sup.html 黄色

子窗口sub.html 蓝色

Location对象

location是BOM对象之一，提供了与当前窗口加载的文档有关的信息，还提供了一些导航功能。location对象是window对象的属性，也是document对象的属性，古window.location和document.locatin等效。



![0VB~6CMPPWK](V@27ZKGUII.png](data:image/png;base64,)

location对象方法

assign() 跳转到指定的URL

reload() 重新加载当前的URL

如果传参，参数为true的时候，强制加载，从服务器源头重新加载

replace() 用新的URL替换当前页面，可以避免产生跳转前的历史记录

history对象

history对象时window对象的属性，它保护这用户上网的记录，从窗口被打开的那一刻算起。

属性

history.length; //history对象中的记录数

方法

history.back(); //前往浏览器历史条目前一个URL，类似后退

history.forward(); //前往浏览器历史条目下一个URL，类似前进

history.go(num); //浏览器在history对象中向前或向后

参数： 0 重载当前页面

正数 前进对应数量的记录

负数 后退对应数量的记录

综合实例：

<script type="text/javascript">

/\*

?id=5&search=ok

获取url中的search，通过传入对应的key，返回key对应的value.

例子：传入id=5，返回5

\*/

function getValue(search,key) {

// <1>找出key第一次出现的位置

var start = search.indexOf(key);

if(start == -1){

return;

} else {

// <2>找出键值对结束的位置

var end = search.indexOf("&",start);

if(end == -1){

// 这是最后一个键值对

end = search.length;

}

}

// <3>将这个键值对提取出来

var str = search.substring(start,end);

alert(str);

}

var search = "?id=5&search=ok";

getValue(search,"search");

</script>

DOM对象 (document object model) 文档对象模型

注：W3C文档对象模型（DOM）是中立于平台和语言的接口，它允许程序和脚本动态的访问和更新文档的的内容、结构和样式。

html css页面内容

js 页面行为操作

注：DOM是打通html、css和JS壁垒的一个工具。

DOM中节点种类一共有三种

注：在JS中所有的节点都是对象。

<div title = "属性节点">测试Div</div>

元素节点 <div></div>

属性节点 title = "属性节点"

文本节点 测试Div

查找元素节点的方法

document.getElementById(id);

功能：通过当前元素节点的id，获取对应元素的节点。

元素节点的属性

通过这个节点对象，访问它的一系列属性

tagName 获取元素节点的标签名

innerHTML 获取元素节点标签间的内容 解析标签



访问这些属性：元素节点.属性名

document.getElementsByTagName();

node.getElementsByTagName();

注：从node节点开始，找出所有符合条件的元素节点

参数：标签名

功能：获取当前页面上所有符合该标签名标准的元素节点

返回值：一个装有符合条件的元素节点的数组

document.getElementsByName();

参数：name的值

返回值：装有符合条件的元素节点的数组

注：name属性一般情况下，只有文本输入框的元素节点才有。

node.getElementsByClassName();

功能：获取node节点下，所有符合条件的元素节点

参数：class

返回值：所有符合条件的元素节点组成的数组

注：getElementsByClassName在低版本的IE浏览器下，不支持

node.style.···只能获取行间的css样式

获取当前有效样式 getComputedStyle(元素节点)[获取样式类型]； 火狐/谷歌/safari支持

元素节点.currentStyle[获取样式类型]; IE

浏览器兼容问题，需要进行判断浏览器

function getStyle(elem,attr) {

return elem.currentStyle ? elem.currentStyle[attr];

getComputedStyle(elem)[attr];

}

获取元素节点

几种方法：

id document.getElementById();

tagName node.getElementsByTagName();

name document.getElmentsByName();

className document.getElementsByClassName();

通过封装函数，简化上述的操作

封装一个函数，可以拥有上述几种获取元素节点的功能。

注：css选择器

#id 通过id获取元素节点

.class 通过className获取元素节点

tagName 通过tagName获取元素节点

name=xxx 通过name获取元素节点

封装函数：

<script>

function $(vArg) {

// <1>对参数进行区分

switch(vArg[0]){

case "#": //id

return document.getElementById(vArg.substring(1));

break;

case ".": //className

return elementsByClassName(document,vArg.substring(1));

break;

default:

//对参数的前五个字符进行判断

var str = vArg.substring(0,5);

if(str == "name="){

return document.getElementsByName(vArg.substring(5));

}else{ //tagName

return document.getElementsByTagName(vArg);

}

} break;

}

function elementsByClassName(parent,classStr) {

// <1>找到parent下所有的元素节点

var nodes = parent.getElementsByClassName("\*");

var result = []; //用来记录符合条件的元素节点

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

// <2>如果符合条件，添加到数组中

if(nods[i].className == classStr){

result.push(nodes[i]);

}

}

return result;

}

</script>

测试：

window.onload = function(){

//id

//alert($("#div1").innerHTML);

//className

// alert($(".box").length);

// name

// alert($("name=hello")[0].innerHTML);

alert($("name=hello").length);

}

</script>

</head>

<body>

<div id="div1">111111</div>

<div class="box">222222</div>

<div class="box">333333</div>

<div name="hello">444444</div>

</body>

</html>

Attribute系列方法

get/setAttribute();

removeAttribute();

注：都是操作当前元素节点中某个属性的。

getAttribute()方法

将获取元素中某个属性的值。它和直接使用.属性获取属性值的方法有一定的区别

区别：1.class属性范围区别，点操作是通过className，而set/getAttribute是通过class

2. set/getAttribute用户自定义属性是支持的

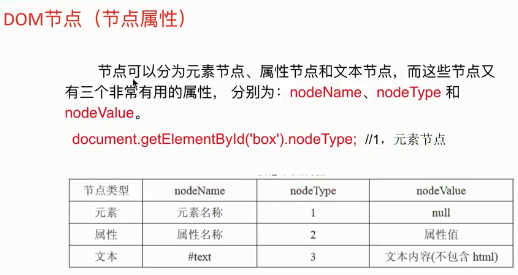
3.removeAttribute可移除属性

元素节点属性

childNodes 获取当前元素节点的所有子节点，包括两种节点类型：

1.元素节点

2.文本节点



节点包括：回车 换行 Tab键 空格 字符

如何将空白节点取出

注：识别出空白节点

/^\s+$/.test()

使用左侧的正则去进行验证，如果是空白节点，返回true，否则返回false

删除空白节点：

<script type="text/javascript">

window.onload = function(){

var oDiv = document.getElementById("div1");

// alert(oDiv.childNodes.length); //5

// 如何将空白节点取出

// 注：识别出空白节点

// /^\s+$/.test()

// 使用左侧的正则去进行验证，如果是空白节点，返回true，否则返回false

// alert(/^\s+$/.test(" "));//true

// alert(oDiv.childNodes.length);

// alert(oDiv.childNodes[0].nodeType);

// var nodes = removeSpaceNode(oDiv.childNodes);

// alert(nodes[0].nodeName);

// alert(oDiv.childNodes.length);

/\*

使用firstChild lastChild

必须从父节点上删除空白节点

\*/

removeSpaceNode2(oDiv);

alert(oDiv.childNodes.length);

alert(oDiv.firstChild.nodeName);

}

function removeSpaceNode2(parent){

var nodes = parent.childNodes;

for(var i =nodes.length-1;i>=0;i--){

if(nodes[i].nodeType == 3 && /^\s+$/.test(nodes[i].nodeValue)){

// 删除该空白节点

parent.removeChild(nodes[i]);

}

}

}

// 删除空白节点

function rem oveSpaceNode(nodes){

var result = []; //用来存放不是空白节点的节点

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

// 判断是否有空白节点

if(nodes[i].nodeType == 3 &&/^\s+$/.test(nodes[i].nodeValue)){

continue;

}

result.push(nodes[i]);

}

return result;

}

</script>

</head>

<body>

<div id="div1">

<em>斜体</em>

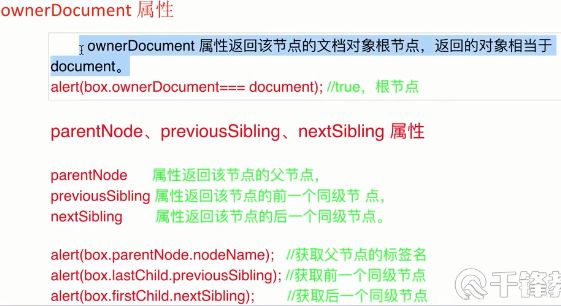
文本内容

<strong>粗体</strong>

</div>

</body>

元素节点属性



属性节点

attribute属性返回该节点的属性节点【集合】（object NamedNodeMap）

集合的特点：1、不重复 2、无序

属性

attributes.length 返回属性节点个数

访问其中一个属性节点 getNamedItem();

例：alert(oDiv.attributes.getNamedItem("id")); //object Atrr

alert(oDiv.attributes["id"]);

属性节点

nodeName nodeType nodeValue

属性名 类型2 值

alert(oDiv.attributes["id"].nodeName);

alert(oDiv.attributes["id"].nodeType);

alert(oDiv.attributes["id"].nodeValue);

节点操作



document.write();添加内容时，会将原有页面的内容覆盖掉

createElement()

格式：document.createElement(标签名);

注：创建节点并不会显示出来

appendChild()

格式：parent.appendChild(newNode);

功能：将newNode插入到parent子节点的末尾

createTextNode()

格式：document.createTextNode(文本);

功能：创建文本节点

// 创建一个带文本的元素节点

function createElementWithTxt(tagName,txt){

var node = document.createElement(tagName);

var oTxt = document.createTextNode(txt);

node.appendChild(oTxt);

return node;

}

inserBefore()

格式：父节点.inserBefore(插入的节点,旧节点);

功能：将插入的节点插入到旧节点之前

// 插入节点到某个节点之后

function insertAfter(newNode,oldNode){

//判断oldNode是否是最后一个节点

var parent = oldNode.parentNode;

if(oldNode == parent.lastChild){

// 最后一个节点，直接插入到子节点的末尾

parent.appendChild(newNode);

}else{

// 插入到oldNode的下一个节点之前

parent.insertBefore(newNode,oldNode.nextSibling);

}

}

replaceChild()

格式：parent.replaceChild(newNode,oldNode);

功能：用newNode将oldNode给替换掉

例：oDiv.parent.replaceChild(newNode,oldNode);

cloneNode()

格式：node.cloneNode();

功能：克隆节点

返回值：新克隆出来的节点

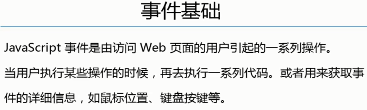
参数：true 默认是false 如果传true，就会复制元素节点中的innerHTML

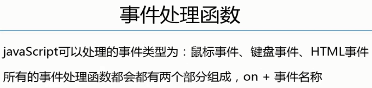
removeChild()

格式：node.parentNode.removeChild(node);

功能：删除节点

事件

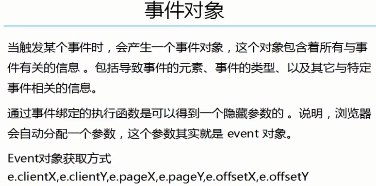




鼠标事件：onclick、onmouseover、onmousedown、onmouseup

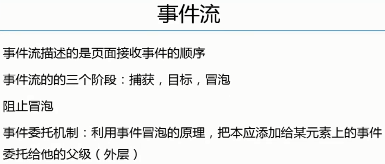
其中onclick包括onmousedown和onmouseup

文本事件：onsubmit、onfocus（获得焦点）、onblur（失去焦点）、onchange（内容改变，并失去焦点）、oninput （输入）

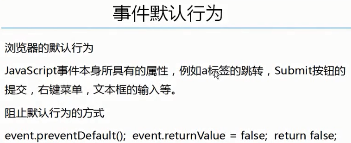


e.client针对可视区、e.page针对整个页面（有滚动条时e.page>e.client）、e.offset针对事件源

键盘事件：onkeypress(按下任意字符键)、onkeydown（按下任意键）、(属性：altKey、ctrKey、shiftKey)



阻止事件冒泡：cancelBubble、stopPropagation()



例：点击a标签，打印aaa，实际情况，由于点击a标签会跳转刷新，故无法打印，这时需要阻止跳转刷新行为

window.onload = function(){

var oLink = document.getElementsByTagName("a")[0];

oLink.onclick = function(e){

var evt = e ||event;

console.log("aaaa");

// return false;

// evt.preventDefault();

evt.returnValue = false;

}

}



HTTP协议：超文本传输协议，用于从web服务器传输超文本到本地浏览器的传输协议，它是一个无状态的协议。

cookie：是指缓存在本地客户端的数据

cookie基本操作包括增、删、改、查四个部分

<script type="text/javascript">

// 查询cookie

console.log(document.cookie);

// 设置cookie

// document.cookie = "username=honey";

// var oDate = new Date();

// oDate.setDate(oDate.getDate()+3); //三天后过期

// document.cookie = "username=honey;expires="+oDate;

// 修改cookie

// document.cookie = "username=honey";

// document.cookie = "username=honey1";

// 删除cookie

var oDate = new Date();

oDate.setDate(oDate.getDate(),1);

document.cookie = "username=honey;expires="+oDate;

</script>

cookie的封装：setCookie()、getCookie()、removeCookie()

<script type="text/javascript">

function setCookie(name,value,day){

var oDate = new Date();

oDate.setDate(oDate.getDate()+day);

document.cookie = name + "="+value+ ";expires="+oDate;

}

setCookie("name1","honey1",1);

setCookie("name2","honey2",1);

console.log(document.cookie);

function getCookie(name){

var str = document.cookie;

var arr = str.split(":");

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

var arr1 = arr[i].split("=");

if(arr1[0]==name){

return arr1[1];

}

}

}

console.log(document.cookie);

function removeCookie(name){

setCookie(name,1,1);

}

removeCookie("name1");

console.log(getCookie("name1"));

</script>

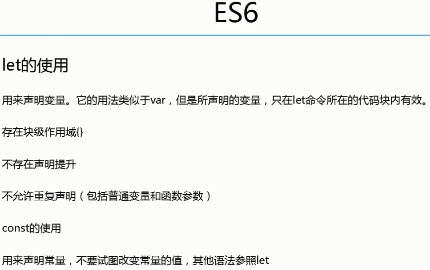
cookie应用，七天免登录：



正则表达式

正则表达式是由普通字符与特殊字符组成的对字符串进行过滤的逻辑公式。

ES6



解构赋值：按照一定的模式，从数组和对象中，对变量进行赋值。

例：数组操作：

let [a,b,c] = [1,2,3] ==> a=1,b=2,c=3

let [a=3,b] = [2] ==> a=2,b=undefined

let c; let [a=2] = [c] ==> a=2

let [a=3,b] = [2] ==> a=2,b=undefined

对象操作：

let (a,b) = {a:"111",b:"222"} ==> a=111,b=222

let (a,b) = {a:"111",b:"222"} ==> a=111,b=222

let (a:b) = {a:111} ==> b=111

let (a,b=5) = {a:1} ==> a=1,b=5

模板字符串

将变量或表达式直接嵌入字符串

使用反引号（``）拼接多行字符串

例：拼接字符串

<script type="text/javascript">

// ES5

var obj = {"name":"john","age":20};

var name = obj.name;

var age = obj.age;

console.log(name+"的年龄是"+age);

// ES6

var obj = {"name":"john","age":20};

let{name,age}=obj;

console.log(`$(name)的年龄是$(age)`);

</script>

箭头函数

1.只含有2个表达式

2.含有多条语句

3.this的指向问题

例： <script type="text/javascript">

// ES5

var foo = function(){

return 1;

}

console.log(foo());

// 相当于ES6

let foo = ()=>1;

console.log(foo());

let foo = (a)=>a;

console.log(foo(10));

let foo = (a)=>{

let b = 10;

return a+b;

}

console.log(foo(10));

</script>

普通函数中this指向window：

例1：

let foo = ()=>{

console.log(this); //指向window

}

例2：

var name = "window";

var obj = {

"name":"john",

"sayHello":function(){

console.log(this.name); //输出john，说明this指向对象obj

}

}

对象中： this指向定义时所在的作用域，而不是执行时所在的工作域

var name = "window";

var obj = {

"name":"john",

"sayHello":()=>{

console.log(this.name); //输出window，说明this指向window

}

}

obj.sayHello();

// 相当于：

var obj = {};

obj.name = "john";

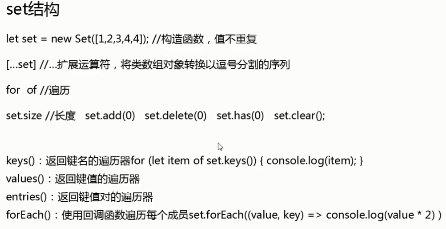
console.log(this);

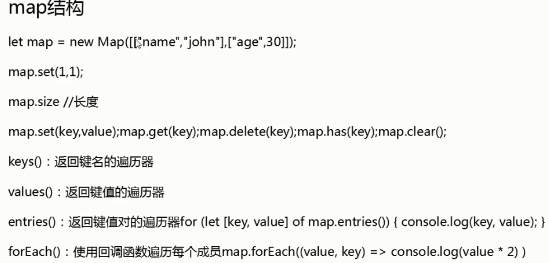
obj.sayHello=()=>{

console.log(this.name);

}

obj.sayHello();





生成器函数（generator）

比普通函数在函数名前多了\*，以及声明变量时前面多了关键字yield，表示暂停函数的运行，并返回一个值。

