[一、 内存 3](#_Toc489884056)

[二、 直接量（字面量）； 4](#_Toc489884057)

[三、 变量 会变的量 5](#_Toc489884058)

[1. 简介： 5](#_Toc489884059)

[2. 变量的作用域：变量的作用范围 5](#_Toc489884060)

[3. 变量名的命名规则： 5](#_Toc489884061)

[四、 数据类型；（五种原始数据类型） 5](#_Toc489884062)

[五、 运算符 6](#_Toc489884063)

[六、 分支结构： 6](#_Toc489884064)

[4. if分支结构5 6](#_Toc489884065)

[5. switch结构——switch case 7](#_Toc489884066)

[七、 循环结构 7](#_Toc489884067)

[1. for循环 7](#_Toc489884068)

[2. while循环 8](#_Toc489884069)

[3. do while循环体 8](#_Toc489884070)

[4. 跳出循环 9](#_Toc489884071)

[八、 function函数算法： 9](#_Toc489884072)

[1. 函数调用 9](#_Toc489884073)

[2. 返回数据return 10](#_Toc489884074)

[3. 匿名函数 10](#_Toc489884075)

[4. 内调函数 11](#_Toc489884076)

[5. 闭包 11](#_Toc489884077)

[九、 DOM 12](#_Toc489884078)

[1. 获取元素 12](#_Toc489884079)

[通过ID获取单个元素 12](#_Toc489884080)

[获取指定标签名的所有元素 12](#_Toc489884081)

[2. 查找父子兄弟节点 13](#_Toc489884082)

[3. 创建、删除、复制节点、追加元素对象 13](#_Toc489884083)

[通过DOM操作属性 13](#_Toc489884084)

[设置元素节点的属性值：元素节点对象.属性名称=新值; 13](#_Toc489884085)

[更改目标样式 14](#_Toc489884086)

[十、 事件 14](#_Toc489884087)

[1. 事件三要素： 14](#_Toc489884088)

[2. 事件模型 14](#_Toc489884089)

[1.1 内联模型(传统事件绑定) 基础 14](#_Toc489884090)

[1.1 脚本模型 流行 14](#_Toc489884091)

[1.2 W3C事件 新兴 15](#_Toc489884092)

[十一、 数组 15](#_Toc489884093)

[1. 数字数组 15](#_Toc489884094)

[1.1 数组的原理： 15](#_Toc489884095)

[1.2 数组的定义： 16](#_Toc489884096)

[1.3 使用数组 16](#_Toc489884097)

[2. 二维数组 18](#_Toc489884098)

[3. 对象数组 18](#_Toc489884099)

[十二、 日期 18](#_Toc489884100)

[1. 日期的获取 18](#_Toc489884101)

[十三、 字符串 19](#_Toc489884102)

[十四、 面向对象 19](#_Toc489884103)

[1. 对象：两个基本组成部分：属性 方法 19](#_Toc489884104)

[1.1. Object 对象 20](#_Toc489884105)

[1.2. 工厂模式创造对象 20](#_Toc489884106)

[1.3. 构造函数 20](#_Toc489884107)

[1.4. 混合模式创造对象： 21](#_Toc489884108)

[1.5. json形式创建对象 21](#_Toc489884109)

[2. 继承 22](#_Toc489884110)

[属性的继承：父类函数名.call(this,属性,属性);// 此为对象冒充， 22](#_Toc489884111)

[方法行为的继承： 22](#_Toc489884112)

[十五、 编写JS库 22](#_Toc489884113)

[十六、 正则表达式 23](#_Toc489884114)

[十七、 画布 24](#_Toc489884115)

[1. 图形绘制基本步骤： 24](#_Toc489884116)

[1. 线条样式 25](#_Toc489884117)

[2. 矩形 25](#_Toc489884118)

[3. 路径 25](#_Toc489884119)

[4. 转换 26](#_Toc489884120)

[5. 文本 26](#_Toc489884121)

[6. 图像绘制 26](#_Toc489884122)

[十八、 控制伪类元素 28](#_Toc489884123)

[十九、 移除ID或class 28](#_Toc489884124)

**注意：**

代码中所有的符号字符均为英文半角；

进行页面显示初始化的js必须放在head里面，因为初始化都要求提前进行（如给页面body设置css等）；而如果是通过事件调用执行的function那么对位置没什么要求的。

安装sublime后首先按shift+Ctrl+p键安装install package control用Ctrl+~确认。后安装Emmet。

两个相同的变量赋值不同则上面的会被覆盖。

顺序结构：代码自上向下执行

垃圾回收机制不是一直运行，是在固定的时间扫描所有内存，发现没有被使用的数据后会进行垃圾回收释放内存。

使用完毕后应该立即释放内存方法为：arr1=null； 注：删除之后不可找回。

快速注释

Ctrl+/ **行注释反注释**

Ctrl+shift+/ **段注释反注释。**

window.onload function（）{} **可在窗口加载完成后在运行该匿名函数，仅可出现一次。**

1.原始数据类型(基本数据类型)：

boolean string number undefined null （error为错误，不属于）

2.引用类型

Function 函数

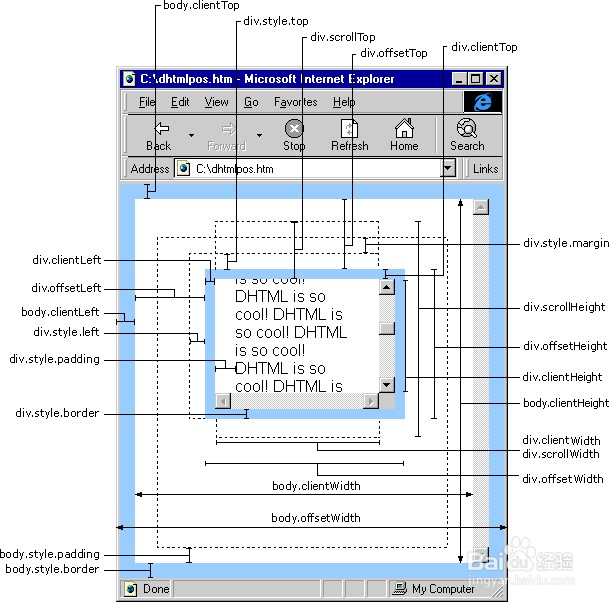
Date 时间

Array 数组

Object对象

RegExp 正则表达式

引用类型有多少个？无数个

****

# **内存**

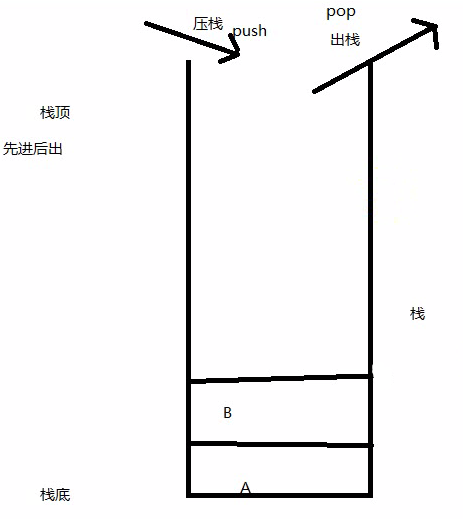
栈结构（stack）

特点：比较小；灵活，快速释放，快速读取，存放数据少，

最上边为栈顶，最下边为栈底。

放入数据为压栈，只能放在最后，出栈为删除数据，只能从删除最后一个开始，一个个删除。

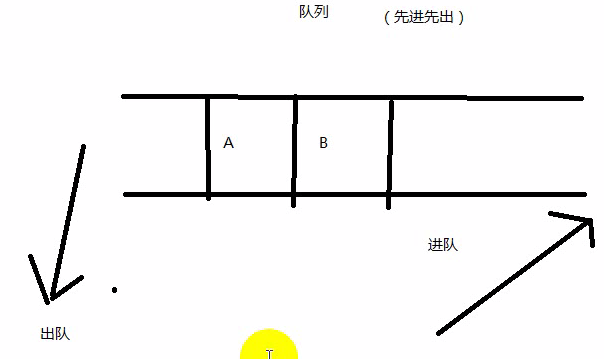
先进后出，后进先出的调取顺序。



队列结构

进队在最后，出队在前面，先进先出，后进后出的调取顺序。

删除呢从第一个开始，添加数据从最后一个添加。



堆结构

特点：无限大，内存有多大就能多大，读取慢，存放数据多，

# **直接量（字面量）**；

* 1. 数值1,2,3,4,5直接输入
  2. 字符串类汉字或字母用“”和‘’把内容包裹起来

# **变量 会变的量**

## 简介：

定义一个变量：var变量名；访问变量中存储的内容，通过变量名

例var dou （定义变量名称为dou） dou=“你好”；（变量赋值，在内存中开辟一块空间，将数据存储到该空间） alert（dou）；（变量应用）

简写为：var address=“瑞达路”；（变量的初始化）

简写二：var aa=123,bb=“哈哈”； 这样也可。

## 变量的作用域：变量的作用范围

(1)全局变量：

声明在函数外部和声明在函数内部但是没有用var关键字声明的，作用范围是整个脚本代码区域

(2)局部变量

声明在函数内部(包括形参)变量，都是局部变量，作用范围只有函数内部

(3)原始数据类型：进行赋值的时候，传递是值

引用类型数据：进行赋值的时候，传递的是地址

注：函数括号内的值为局部变量，而函数内未定义var的变量默认为全局变量。

尽量使用局部变量，少使用全局变量

## 变量名的命名规则：

1. 首字母必须以$ 或\_ 或字母开头
2. 变量名由$和\_ 或字母，数字组成
3. 区分大小写
4. 变量名不能用关键字和保留字（保留字是开发者为以后的添加预留的字）
5. 变量可以重复赋值
6. 变量的声明的时候，无数据类型。

# **数据类型；（五种原始数据类型）**

1. 数值类型：number 1,2,3,100,100.1,0.001,3.14;
2. 未定义 undefined
3. 字符串类型：string，以‘’或“”包裹起来的内容。
4. 布尔类型：boolean true（真）false（假）
5. 空 ： null类型.

注：

boolean类型进行数学运算时，true认为是1，false是0。

进行判断时除了0，其他数字都认为是true。例：if(-1){document.write("明天周末");}

typeof（）语句可以检测变量类型。检测写在括号内的

**类型转换**

* 1. 其他类型转换到string类型，仅需要加上“”即可。例： var a1=a+"";
  2. string转换到number类型，当是数字内容的字符串可以使用取整或者使用取小数；例：var a21=parseInt(‘10’);//转换为整数；var a31=parseFloat(‘10.1’);//转换为小数；
  3. 如果本身不是数4则无法转化为number类型，若前面是数字后面是其他字符则前面的转换，后面的忽略；因程序自左向右运行所以字符串不被转化后面数字的也不直接略过了。例：var b11=parseInt(true);//NaN not a number；var b12=parseInt(‘12sdff’);//12 ；var b12=parseInt(‘dsfs12’);//NaN

# **运算符**

1. 赋值运算符 =
2. 算术运算符 + - \* /（除） %（取余）+号的两边有一个为字符串则执行字符的拼接，而不是相加。

简化为：a=a+1为a+=1 ; a=a-1为a-=1；

1. 取余的意思是除后取整数，用法相同。

Num- -等价于num=num-1；num++等价于num=num+1；

++ ，- -前置先运算后赋值，后置先赋值后运算。+ + - -只能作用于变量不可作用于数值。从左向右运算。后面括号内的变量为前面运算后的。

注：+号运算符：

当变量两边都为number类型时为数学运算

当有一边为string类型时为为字符串拼接，但是加号仅作用于加号两端的内容，之外的内容不作用。

1. 关系运算符：作用于两个变量或表达式的，关系运算符的运算结果为布尔类型。

>< >= >= <= ==(等于只判断值是否相等，不判断数据类型) !=（不等于判断是否不等于） ===（真等于，要求数值和数据类型均相等。） ！==（真不等于）

1. 逻辑运算符：作用于两个变量或表达式，运算结果为布尔类型。&&逻辑与两边都为真，结果才为true，||逻辑或一个为true则都为true。，！非；

逻辑运算符不仅可以作用于boolean类型，可以作用于任何类型

例如：||

（1），若其中一个操作数为对象，则运算结果为此对象；例：var a=arr||true;

（2），若第一个操作数为为false，则返回第二个操作数；例：var b=false||arr;

（3），若两边操作数都为对象，则返回第一个对象操作数。例：var c=arr||arr1;

注意：undefined和null和0均可以被认为是false。

1. 三元操作符： 表达式?结果1:结果2

//若表达式结果为true，则执行结果1，否则执行结果2

a=1

a!=1?alert("真"):alert("假");

返回的是一个值，不可用于return；

# **分支结构：**

## if分支结构5

语法：

if（判断条件）{

若判断条件成立（true），则执行此处的代码 }

else{若条件不成立，则执行此处代码且，不能设置条件 }

if（条件1）{

条件1成立，则执行

}else if（条件2）{条件2成立，则执行

}else if （条件n）{条件n成立，则执行

}else{以上条件均不成立，则执行}

说明：判断条件的结果必须为布尔类型，判断条件可以使变量，也可以是循环结构。

<script type="text/javascript">

var myage = 18;

if(myage>=18) //myage>=18是判断条件

{ document.write("你是成年人。");}

else //否则年龄小于18

{ document.write("未满18岁，你不是成年人。");}

</script>

## switch结构——switch case

switch(参与比较的条件){

case 值1:

若参与比较的条件与值1相等，则执行此处代码

break;

case 值n:

若参与比较的条件与值n相等，则执行此处代码

break;

default:

若上面均不成立，则执行此处代码

break;

}

break用于跳出该case，若是没有写上会继续往下进行，会执行下面case的代码。该作用的某些方面可以使用，该功能为case 穿透。**结构用于单值比较**

# 循环结构

循环体的嵌套无可厚非，大同小异的。正常嵌套即可。不过要注意循环顺序。

## for循环

语法与原理for(循环条件的初始化;循环条件;循环迭代) {循环体}

(1)循环条件的初始化 var i=1; （此步骤只在初次执行）

(2)进行循环条件的判断 i<7;若条件成立，则继续第(3)步;若循环条件不成立，则结束循环

(3)执行循环体

(4)进行循环条件的迭代 i++; (i=i+1),继续回到第(2)步

例子：//输出1--100之间的偶数

for(var i=1;i<=100;i++){

if(i%2==0){

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

}

循环中的每个条件都可以被替代：但是必须有这些条件。

var i=1;

for(;;){

if(i>=10){

break;

}else{

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

}

i++;}

提取整数中的位数：

例：153 res 或者：res=a\*100+b\*10+c

1. 百位：var bai=parseInt(res/100); 或者a=100\*a
2. 十位：var shi=parseInt(res/10)%10; 或者b=10\*b
3. 个位：var ge=res%10; 或者c=c

## while循环

语法：

while(循环条件){

循环体

}

当循环条件成立时(表达式或者变量，布尔类型)，执行循环体

var i=1;

while(i<6){

document.write("hello<br>");

i++;

}

也可以直接选择循环不停止，当事件触发后才停止。

var i=1;

while(true){

i++;

if(i>6){

break;

}

document.write("hello<br>");

}

## do while循环体

do{

循环体

}while(循环条件);

dowhile 循环体至少执行一次

例子：var i=1;

do{

document.write("hello<br>");

i++;

}while(i<5);

## 跳出循环

continue的作用是仅仅跳过本次循环，而整个循环体继续执行。

语法：for(初始条件;判断条件;循环后条件值更新)

{

if(特殊情况)

{ continue; }

循环代码

}

例：



# function函数算法：

函数==方法 驼峰命名法 :从第二个单词开始之后的单词，首字母大写

函数定义：

function 函数名(){

函数体

}//一般情况下，变量名是名词；函数名是动词

函数自己不会执行， 函数只有被调用的时候，才可以执行

函数的调用：函数名();

函数的调用位置与函数定义位置无关；但是需要在一个js标签内。

调用函数名意为调用这个函数后面有括号代表调用后的结果。例

function add2(){

var sum = 3 + 2;

alert(sum);

}

括号内为所要输入的变量名，仅需要在引用函数内直接输入变量名即可，可以有多个。

可以将函数赋值给变量，调用变量即为调用函数，相当于名字的作用。

## 函数调用

函数定义时的参数：形参

函数调用时的参数：实参

理论上函数可以有任意多个参数,参数之间用“,”隔开

//形参 l

function sayHello(na,isMan,age){

if(isMan){

document.write("hello boy"+na+"年龄为："+age);

}else{

document.write("hello girl"+na+"年龄为："+age);

}

}

实参

// sayHello("Jack",true); 4

sayHello("Lucy",false,22);

## 返回数据return

return 有两种作用：

（1）返回数据（只能返回一个数据）

return 表达式;

(2)结束当前函数

例如： function calReg(w,h){

// var area=w\*h;

return w\*h;//返回长方形面积，，结束当前函数

//垃圾代码

document.write("计算完毕");//未执行

var a=10;

var b=20;

var c=a>b&&a<b;

console.log(c,a,b);

}

var str=calReg(10,20);//接收函数的返回值

document.write(str);

## 匿名函数

匿名函数有两种执行方式：

(1)自我执行

(2)事件绑定的时候

//匿名函数自我执行

(function(){

document.write("hello");

})();

//带参数的匿名函数

(function(n){

document.write("hello "+n);

})("Jack");

//带返回值的匿名函数

var num=(function(){

return 5;

})();

document.write(num);

应用到点击事件中

<button id="btn">按钮</button>

<script type="text/javascript">

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

o.onclick=function(){

alert("click");

};

</script>

## 内调函数

当需要对函数进行递归算法时可以采用内调函数。例如当做杨辉三角就需要使用内调函数，对值进行递归计算。

function cal(m,n){

//每一行第一列

if(n==1){

return 1;

}else if(m==n){

//每一行最后一列

return 1;

}else{

return cal(m-1,n)+cal(m-1,n-1);//递归

}

}

## 闭包

在一个函数内部有权访问另一个函数内部的变量

外部无法调取函数内函数，且外部调取函数后内部函数已经执行完毕，会造成内部函数的值无法获取到顺序值，只能获取到最后的值。

内部函数是在全部赋值数组后才赋值变量。

解决：1，先在内部自我调取；

2，赋一个值后再读取各自的值。

例1： function show(){

var arr=[];

for(var i=0;i<5;i++){

arr[i]=(function(a){用小括号包住自己代表自己为一个个体，后面再接小括号，代表调用自己。

return a;

console.log(i);

})(i);

}

return arr;

}

var a=show();

console.log(a);

例2： for(var i=0;i<os.length;i++){

os[i].\_index=i; 可以对标签设置一个没有的属性，该属性可以作为标签的单独变量使用。

os[i].onclick=function(){

// alert(i);

alert(this.\_index);

};

}

# DOM

文档对象模型DOM（Document Object Model）定义访问和处理HTML文档的标准方法。DOM 将HTML文档呈现为带有元素、属性和文本的树结构（节点树）。



元素节点：上图中<html>、<body>、<p>等都是元素节点，即标签。

属性节点:元素属性，如<a>标签的链接属性href="http://www.imooc.com"。

document是Document类型的一个对象,指HTML文档节点 、元素（指HTML中的标签）

文本节点:向用户展示的内容，如<li>...</li>中的JavaScript、DOM、CSS等文本。

元素节点（ELEMENTNODE）、属性节点（ATTRIBUTENODE）、文本节点（TEXTNODE）

父子关系(子元素 父元素)兄弟关系.

DOM是一套标准的交互体系,设计来允许开发人员对HTML文档进行修改、增加、删除等操作

## 获取元素

### 通过ID获取单个元素

document.getElementById(id); 跟据ID来查找元素，若查找到了，则返回此元素对象，若查找不到，则返回null

document.getElementById(“id”) （文档中的获取元素为ID）innerHTM指的是div的尖括号内内容

<p id="con">JavaScript</p>

<script type="text/javascript">

var mychar=document.getElementById('con'); ;

document.write("结果:"+mychar); //输出获取的P标签。

</script>

### 获取指定标签名的所有元素

getElementsByTagName() 方法可返回带有指定标签名的对象的集合。

例：节点. getElementsByTagName("li");

返回的值为例length:2与对象的对应关系，可根据此关系设置程序。

## 查找父子兄弟节点

console.log(o.childNodes);// 获得所有的孩子节点

console.log(o.firstChild.parentNode);// 获得第一个孩子节点

console.log(o.lastChild);// 获得最后一个孩子节点。

console.log(o.childNodes[1]);// 某个孩子节点从0开始算起

console.log(o.firstChild.nextSibling);// 下一个兄弟节点

console.log(o.lastChild.previousSibling);// 上一个兄弟节点

console.log(o.firstChild.parentNode);// 由孩子节点查找父节点

## 创建、删除、复制节点、追加元素对象

1. 创建节点createElement();

例：var o\_p=document.createElement("p");

o\_p.innerHTML="我是一个段落";

o\_div.appendChild(o\_p);

1. 删除节点removeChild();

例：o\_div.removeChild(o\_p);

console.log(o\_div);

1. 复制节点cloneNode(参数); 参数为boolean类型，若参数为true,则复制孩子节点，若参数为false，则不复制孩子节点,默认为false

例：var o2=o1.cloneNode();

var o3=o.cloneNode(true);

1. 追加元素对象appendChild();

例：document.body.appendChild(o\_a);

### 通过DOM操作属性

获取元素节点属性值：方法为：元素节点对象.属性名称

例：<img src="images/01.jpg" id="pic" title="图片提示">

var ig=document.getElementById("pic");

var str=ig.src;

console.log(str);

console.log(ig.title);

### 设置元素节点的属性值：元素节点对象.属性名称=新值;

1. d.className=’onemore’; 通过加。直接更改，个别标签需要变形。
2. pic[“src”]=”images/01.jpg”； 通过中括号更改属性，个别标签需要变形。
3. 为目标添加属性，并设置值。变量.setAttribute（‘属性名’，“值”）个别标签不需要变形直接更改；例如：pic.setAttribute（”title”，”图片”）
4. 获取属性的值：getAttribute例如：var sr=pic.getAttribute(“src”);

注：特殊的标签不可直接写上，因为与系统关键字相同所以名为htmlFor

例如：var o=document.getElementById("lab");

o.htmlFor="txt";

### 更改目标样式

样式修改：能在CSS中完成的，不需要用代码实现

1. 样式修改：

对象.style=“新的样式值”；会覆盖之前的样式，不推荐。

对象.style.样式名称=新的样式值; 单独进行修改某个样式。

注意：若样式名称为多个单词组成例如：font-size

则去掉“-”，从第二个单词开始，首字母大写 例如：fontSize

1. 样式值的获取：

元素对象.style.样式名。例：o.style.color; 注意：此种方式只能获取行内样式的值

若：样式的声明在样式表中，则通过如下方式获取

var sty=window.getComputedStyle(o,null);//o指，要获取样式的对象; sty.样式名。

var h1=sty.height;再从中调取所选样式。

offsetLeft属性只能读取，不能赋值。获取到的直接是数值。

o．offsetLeft；

# 事件

## 事件三要素：

(1)触发事件的对象

(2)触发了什么事件

(3)事件触发后要执行的事情

注意：事件自己不会执行，只有被触发后才会执行

## 事件模型

### 内联模型(传统事件绑定) 基础

将事件作为属性声明在元素标签中

例：<button onclick="show()">按钮</button>

function show(){

alert("hello");

}

优点：简单

缺陷：不适应分工合作。易于出错。

### 脚本模型 流行

为了实现分工合作，前端写入标签与ID中后，通过JS对该目标设置事件。

例：<div id="box" ></div>

var o=document.getElementById("box");

o.onclick=function(){

o.style.background="green";

};

### W3C事件 新兴

1)添加事件

addEventListener(事件的名称,事件处理函数,冒泡或捕获);

第三个参数是一个boolean类型，默认是false；

false意为冒泡传递（事件句柄）：事件由内部向外部元素层层传递。绝大数都是冒泡机制。

true意为捕获传递

阻止冒泡传递stopPropagation() 不兼容低版本IE

例：e.stopPropagation();

兼容性写法 处理事件冒泡

window.event?e.cancelBubble=true:e.stopPropagation();

2)移除事件

removeEventListener(事件的名称,事件处理函数,冒泡或捕获);

注意:添加事件时的函数应和移除时的函数为同一个

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

//添加事件

document.getElementById("btnAdd").onclick=function(){

btn.addEventListener("click",show,true);

};

//移除事件

document.getElementById("btnRemove").onclick=function(){

btn.removeEventListener("click",show,true);

};

### 获取坐标的方法

onmousemove鼠标移动事件。

1.PageX/PageX:鼠标在页面上的位置,从页面左上角开始,即是以页面为参考点,不随滑动条移动而变化

2.clientX/clientY:鼠标在页面上可视区域的位置,从浏览器可视区域左上角开始,即是以浏览器滑动条此刻的滑动到的位置为参考点,随滑动条移动 而变化.

可是悲剧的是,PageX只有FF特有,IE这个悲剧没有啊T\_T,所以大牛们想出了一个办法

PageY=clientY+scrollTop-clientTop;(只讨论Y轴,X轴同理,下同)

3.screenX/screenY：鼠标在屏幕上的位置,从屏幕左上角开始（w3c标准）

4.offsetX/offsetY:IE特有,鼠标相比较于触发事件的元素的位置,以元素盒子模型的内容区域的左上角为参考点,如果有boder,可能出现负值

5.

layerX/layerY：FF特有,鼠标相比较于当前坐标系的位置,即如果触发元素没有设置绝对定位或相对定位,以页面为参考点,如果有,将改变参考坐标系,从触发元素盒子模型的border区域的左上角为参考点也就是当触发元素设置了相对或者绝对定位后,layerX和offsetX就幸福地生活在一起^-^,几乎相等,唯一不同就是一个从border为参考点,一个以内容为参考点

chrome和safari一条龙通杀!完全支持所有属性.其中(offsetX和layerX都是以border为参考点)

下面这个是获取相对于屏幕的坐标

document.onmousemove=function(e){e=e? e:window.event;document.writeln("X:"+e.screenX+"Y:"+e.screenY);}

### 滚轮事件

$(document).on("mousewheel DOMMouseScroll", function (e) {

var delta = (e.originalEvent.wheelDelta && (e.originalEvent.wheelDelta > 0 ? 1 : -1)) || // chrome & ie

(e.originalEvent.detail && (e.originalEvent.detail > 0 ? -1 : 1)); // firefox

if (delta > 0) {

// 向上滚

console.log("wheelup");

} else if (delta < 0) {

// 向下滚

console.log("wheeldown");

}

});

# 数组

数组是用来处理(管理)具有相同特点的数据，数组存储的数据可以是任何类型（数字、字符、布尔值等）。

注意：数组有如下特性：（两个不限制）

JavaScript数组中理论可以存放无限多个数据(不限制数据个数)

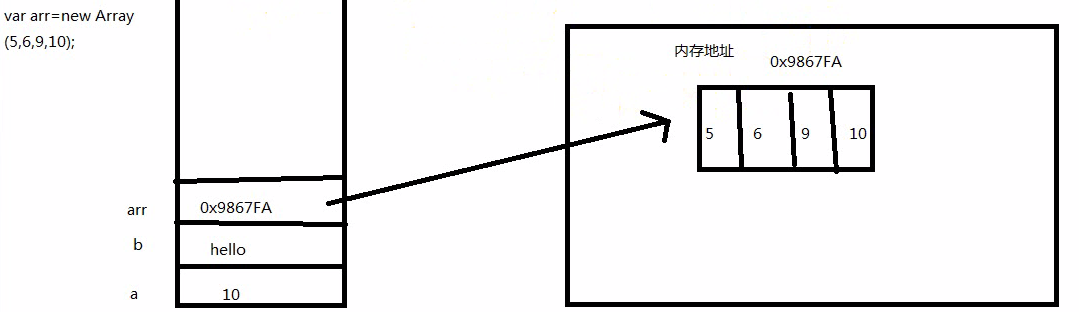
JavaScript数组中可以存放任意类型的数据(不推荐)(不限制数据类型)，最好分开放置。

## 数字数组

### 数组的原理：

var arr=new Array();//将数据存储在堆空间中，并且在栈内存中开辟空间，存储堆区的内存地址

例：var arr=[5,6,9,10];



console.log(arr);//arr代表堆区的内存地址引用，但浏览器为了页面友好会将内存地址翻译为内容而不显示内存地址，并不是存的是内容。

### 数组的定义：

new Array (); 创造一个数组的实例。

var myarr=new Array(8); 后面括号内为储存多少数据，创造一个数组的实例

var arr=new Array(5,7,6); 直接赋值数据

var arr=[]; 通过字面量的形式来构建数组；[]为下标，需通过下标访问数组中的某个对象。

例： var arr=[5,90,10,20];

### 使用数组

#### 访问数组中的数据

数组中，每个数据均有一个下标索引，此下标索引从0开始

数组名[索引];

#### 赋值数组

myarr[0]=”你好”;

简写：例：var myarray = new Array(66,80,90,77,59);//创建数组同时赋值

简写：例：var myarray = [66,80,90,77,59];//直接输入一个数组（称 “字面量数组”）

alert=myarr[0];

#### 数组的个数 length

数组名.length;

myarray.length; //获得数组myarray的长度

document.write(arr.length); //显示数组长度

数组长度可变

arr.length=10; //增大数组的长度

#### 数组的修改

从头部加入一个或者多个数据：unshift

例：unshift(item1,...,itemn);

从头部删除数据：shift

例：shift(); 出队：删除队首元素

从尾部加入数据：push

例：push(item); 进队: 将item添加到队尾

从尾部删除数据：pop

例：arr.pop(20);

按照ascall码排序sort(); 不是按照大小排列而是按照排码排列。

例：var arr=[5,100,2,234820,1,23423,34534,34536432,2];

var arr0=arr.sort();

颠倒数组中的元素reverse(); 意为将倒数一个与第一个调换，倒数第二个与正数第二个依次全部调换。

arr.reverse();

数组合并将一个或多个数组进行合并 ,返回一个新数组concat(arr1,...,arrn);

例：var arr=[1,34,534,346,345];

var arr1=["hello","dsd","|dsf","dsaf"];

var arr0=arr.concat(arr1); 若没有合并对象则会对arr进行复制。

数组的分割slice(start,end); //start 分割起始位置，end 分割的结束位置不分割

返回新数组值为 [start,end) 原数组不会改变。

例：var arr=["hello","Nihao","good","bad","bus"];

var arr0=arr.slice(1,3);

console.log(arr0,arr);

向数组中添加元素或者删除数组元素

splice(index,howmany);

从index位置开始，从数组中删除howmany个数据，并返回的数据的数组

例： var arr=[2,342,345,345,346,457,4764,213,747478467];

var arr0=arr.splice(1,3);

从index位置开始，删除howmany个元素，并且从index开始，添加元素

范例：splice(index,howmany,item1...itemx);

例：var arr0=arr.splice(1,0,"hello");

将数组拼接为一个字符串join(spe（符号，字符，空格）) 将数组以spe拼接为一个字符串

var arr=[2,234,24,22532,23532,2345,2];

var str=arr.join("+");

#### 数组的遍历

for...in 语句用于对数组或者对象的属性进行循环操作。

例： for(var a in arr){//a为数组的下标

ss+=arr[a];

}

注：若使用未定义索引跨度大会增加数组长度，但中间的数组没赋值。

var arr=[98,76,54,56,76]; // 包含5个数值的数组

document.write(arr.length); //显示数组的长度5

arr[15]=34; //增加元素，使用索引为15,赋值为34

alert(arr.length); //显示数组的长度16

## 二维数组

一维数组，我们看成一组盒子，每个盒子只能放一个内容。

一维数组的表示: myarray[ ]

二维数组，我们看成一组盒子，不过每个盒子里还可以放多个盒子。

二维数组的表示: myarray[ ][ ]

注意: 二维数组的两个维度的索引值也是从0开始，两个维度的最后一个索引值为长度-1。

第二个数组创建时不需要var，因为已经存在一维数组。

例：var myarr=[]; //先声明一维

for(var i=0;i<2;i++){ //一维长度为2

myarr[i]=[]; //再声明二维

for(var j=0;j<3;j++){ //二维长度为3

myarr[i][j]=i+j; // 赋值，每个数组元素的值为i+j

}

}

## 对象数组

下标为字符串类型，无长度，不可通过数字下标访问。

可以通过for。。。in。。。语句查看所有数组对象。

例： var arr=[];

arr["姓名"]="Jack";

arr["年龄"]=18;

arr["工作"]="程序猿";

for(var i in arr){

console.log(arr[i]); }

# 日期

## 日期的获取

获取当前时区的时间，并使用变量存返回值：

例：var time=new Date();

若获取制定时间在括号内传参的方式为 2014-11-13-20-30-00 该方法通用，其他的ios不认识。

提取其中的日期时间：

例： var hour=time.getHours();

var minute=time.getMinutes();

var second=time.getSeconds();

var week=time.getDay();周的返回值是从0到6

var month=time.getMonth();月的返回值是从从0到11

还有更多的属性在手册。

**setInterval() 定时器调用函数或计算表达式**

setInterval(code（）,1000) 前面为函数，后面为调用的时间间隔。

范例：t=setInterval(function(){o.innerHTML=o.innerHTML+"HELLO WORLD <br>"; },50);

可以通过给定时器赋值变量并将其在控制台输出查看系统的编码为多少。

console.log（t）;

停止调用

停止btnStop.onclick=function(){clearInterval(t);//清除定时器}

**setTimeout 指定时间后执行代码一次**

例：setTimeout(function(){o.style .opacity=1;},2000);

# 字符串

字符串的使用修改方法与数组相同。

length 字符串的长度:字符串中字符的个数。每个字符为一个数组。

通过下标可以访问每个字符。字符串中每一个字符均有一个索引号，而且索引从0开始。

**(1)charAt(index) 返回指定位置的字符**

例：var c=str.charAt(2); 从0开始

**(2)concat(str1,...strn) 拼接一个或多个字符串,,返回拼接后的字符串**

str必须也是个字符串。

例：s1=str.concat("hello","world","342");

**(3)indexOf(str,fromIndex); 返回某个指定的字符串值在字符串中首次出现的位置。若找不到返回-1**

**fromIndex 指定开始搜索的位置**

例：var str="jalksdfjq9weruoqwi5u293ij4elk";

var i=str.indexOf("k",5);

**(4)lastIndexOf(str,fromIndex) 指定字符最后一次出现的位置，从后向前搜索**

例：str.lastIndexOf("a"),str.length); 后面的数字也是从前往后数的。

**(5)replace(oldStr,newStr) 替换字符串 会返回一个更改后的值。**

例：var str="hello world hello hhhhhh";

str.replace("hello","你好")); 替换仅发生一次，替换第一次找到的，若全部替换需要遍历。

**(6)slice(start,end) 获取指定区间的字符串 [start,end)**

例：var str="hello";=

console.log(str.slice(1,3)) 左开右闭。

**(7)split(c) 用指定字符来将字符串分割为数组，返回值为分割的数组**

例：var str="hello world 你好 hhahah aldfjalsdk hahdfah";

var arr=str.split(" ");

**(8)substring (star,stop) 获取从指定位置开始，到number结束，并不截取最后一个。 [start,stop)**

例：var str="今天天气真不错";

var s=str.substring(1,4);

**(9)substr(start,length) 从指定位置获取固定长度的字符串**

例：console.log(str.substr(1,3));

**(10)toUpperCase()**

将所有字符大写。

**(11)toLowerCase()**

将所有字符小写。

# 面向对象

## 对象：两个基本组成部分：属性 方法

属性的内容是不相同的，不过行为方法可以通过绑定原型实现公共使用，减少内存消耗。对对象新建属性与值：defineProperty或者下标的方法。

### Object 对象

例： //自定义对象：

var student=new Object();//创造一个对象

// console.log(typeof student);

//属性

student.name="李四";

student.age=19;

student.no="10001";

//方法

student.login=function(){

console.log("登录成功");

};

//属性的访问

console.log(student.name,student.age,student.no);

student.login();

### 工厂模式创造对象

function createStudent(name,age,no){

var stu=new Object();

stu.name=name;

stu.age=age;

stu.no=no;

stu.login=function(){

console.log(this.name+"登录成功");

};

return stu; 必须返回值，否则无法获取到

}

var stu1=createStudent("jack",19,"1023");

console.log(stu1.age,stu1.name,stu1.no);

stu1.login();

var stu2=createStudent("Tom",20,"2342");

console.log(stu2.age,stu2.name,stu2.no);

stu2.login();

var stu3=createStudent("Merry",90,"2342");

stu3.login();

console.log(stu1.login==stu2.login);//false 两个相同的行为却占用不容的内存

对象原型:构造函数所拥有的一块空间，这块空间是共享的(所有由此构造函数创造的对象，均可以访问)

//将方法绑定在构造函数的原型上

Student.prototype.login=function(){

console.log(this.name+"登录成功");

};

### 构造函数

构造函数被调用时，做了两件事情：

1，创建了一个对象，并给这个对象追加行为和属性(用代码完成),

2.把创建的对象返回给调用者

function Person(name,age){

this.name=name;

this.age=age;

this.walk=function(){

console.log(this.name+"walking");

};

}

var p=new Person("Tom",20);

console.log(p.name,p.age);

p.walk();\*/

### 混合模式创造对象：

意为通过工厂模式对对象设置完属性后通过公用的行为来对不同对象的相同属性进行设置。

(1)利用构造函数来完成对象私有属性的创造

(2)将对象所公有的行为(方法),绑定在构造函数的原型上

function Person(height,weigth,age,name){

this.hei=height;//创造一个属性height,把变量height中的值赋值给新属性

this.weigth=weigth;

this.age=age;

this.name=name;

}

Person.prototype.walk=function(){

console.log(this.name+"跑的真快");

};

Person.prototype.say=function(){

console.log(this.name+"说的真好");

};

var p1=new Person(190,200,20,"赵六");

var p2=new Person(190,200,20,"田七");

console.log(p1.age,p1.name,p1.hei,p1.weigth);

p1.walk();

p1.say();

console.log(p2.age,p2.name,p2.hei,p2.weigth);

p2.walk();

p2.say();

### json形式创建对象

使用var person={}大括号可以直接创建对象。里面的行结束符都是逗号不是分号。

var person={

name:"Jack",

age:18,

walk:function(){

console.log(this.name+" is walking");

}

};

person.walk();

console.log(person.name,person.age);

</script>

## 继承

父类 是同有的属性或方法，子类是独有的属性或方法。

### 属性的继承：父类函数名.call(this,属性,属性);// 此为对象冒充，

查找呼叫父类函数，对于this（它）的 属性 进行继承，中间用逗号隔开。

例： var Student=function(name,gen,no){

Person.call(this,name,gen);//对象冒充

}; 不影响添加独有属性。

call方法意为更改其this指向，指向传入的对象，若没有就是默认的对象。

apply方法与call相同不同之处为，传入的参数必须是数组形式，例：add.apply(s,[5,6])

### 方法行为的继承：

1. 赋值，相当与Student与Person公用一块内存空间

Student.prototype=Person.prototype; 这样就是在公用一块空间他们的所有行为会相同，不建议

1. 完成方法的继承

使用对父类原型的遍历将父类方法全部赋给子类，子类不影响添加独有的行为，这样两者的所有行为是不一样的。

例： for(var i in Person.prototype){

Student.prototype[i]=Person.prototype[i];

}

### 实例：

function Persion(name,age){

this.name = name;

this.age = age;

}

// 父类的原型对象属性

Persion.prototype.id = 10;

// 子类

function Boy(name,age,sex){

//call apply 实现继承

Persion.call(this,name,age);

this.sex = sex;

}

// 原型继承实现 参数为空 代表 父类的实例和父类的原型对象的关系了

Boy.prototype = new Persion();

var b = new Boy('c5',27,'男');

alert(b.name)// c5

alert(b.id)//10

通过混合模式实现继承。

# 编写JS库

用原生JS将开发中常用到的功能，按照一定的逻辑封装起来，生成一个js文件，JS库。

常用的JS库有：jQuery、Exjo

库封装完成后，可以加快开发节奏

编写js

1. 创建函数，创造Base对象，并且将创造的对象返回。

function $(){

return new Base();

}

1. 定义变量属性：

var Base=function(){类库构造函数

this.index=[]; 定义一个属性，用来保存查找到节点对象

};

获取目标（ID,className，Name）： 获取后存入对象的属性，并返回对象，这样才能实现连缀写入样式；

例：Base.prototype.getId=function(id){ 根据ID查找元素，找到元素后，返回当前对象

var o=document.getElementById(id);

this.index.push(o);

return this;

};

例：Base.prototype.getClaaName=function(className){

var o=document.getElementsByClassName(className);

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

this.index.push(o[i]);

};return this; //返回当前对象，目的是为了连缀

}

1. 对目标属性进行更改赋值；判断考虑为空时要返回指定属性的值；如果不为空则更改目标属性。

例：Base.prototype.html=function(innerValue){

if(arguments.length==0){

return this.index[0].innerHTML;

}

else{

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

this.index[i].innerHTML=innerValue;

}

}

return this;

}

1. html方法为两用：

（1）若传递有参数，则为设置元素对象的innerHTML值

（2）若不传递参数，则为获取元素对象的innerHTML值

例：Base.prototype.html=function(innerValue){

// console.log(arguments.length);//参数的长度

if(arguments.length==0){

//未传递参数,功能未获取元素对象的innerHMTL值

return this.elements[0].innerHTML;

}else{

//有参数，设置

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

this.elements[i].innerHTML=innerValue;

}

}

return this;

};

# 正则表达式

正则表达式:是一种规则或者由特殊字符组成的式子

主要用作在用户表单提交时，进行数据的校验

正则表达式比较难懂，又称火星文

创建正则表达式：new RegExp() 此方法不推荐使用

正则表达式的书写：/pattern/flag

正则表达式的书写：

/pattern/flag

flag的值可以为 i(ignore 忽略大小写)

g(globle 全局) m(mulit) 多行匹配

例：var str="aaasdjfa29358 kmfaa^$%(4 6sd.C,m.";

字符类：

(1)[] 任意字符: [abd890]

(2)范围：[a-z] [0-9] [A-Z]

(3)排除：^

[^a-z]

注意：若携写在[]内部表示排除

若写在[]外部表示：以xx开头

(4)组合用法：

[0-9a-zA-Z\_]

例：var s=str.replace(/[0-9]/g,"\*");

s=str.replace(/[a-zA-Z]/g,"0");

s=str.replace(/[^0-9]/g,"\*");

系统定义的字符类：

\d 匹配所有的数字 [0-9]

\w 匹配所有的单词 [a-zA-Z0-9]

\s 匹配空格 换行 制表符

\D 匹配所有的非数字 [^0-9]

\W 匹配所有的非单词 [^a-zA-Z0-9]

\S 对\s 取反

量词： 作用：用来限定指定字符出现的次数

+ 一次或多次 [1,正无穷]

\* 零次或多次 [0,正无穷]

? 零次或一次 [0,1]

{n,} 至少n次

{n, m} 至少n次, 最多m次

{n} n次

$：表示以xx结束

例： var s=str.replace(/\d/g,"\*");

var s=str.replace(/\w/g,"\*");

var s=str.replace(/\s/g,"你好");

var s=str.replace(/[a]+/g,"你好");

var s=str.replace(/[a]\*/g,"你好");

var s=str.replace(/[a]{4,}/g,"你好");

# 画布

## 图形绘制基本步骤：

(1)设置画布的宽高(在属性中设置)

(2)获取2d对象getContext("2d")

例：var ctx=cav.getContext("2d");

(3)起点：ctx.moveTo(250,250)

(4)绘制

fillStyle 设置或返回用于填充绘画的颜色、渐变或模式

|  |  |
| --- | --- |
| **属性** | **描述** |
| [fillStyle](http://www.w3school.com.cn/tags/canvas_fillstyle.asp" \o "HTML5 canvas fillStyle 属性) | 设置或返回用于填充绘画的颜色、渐变或模式 |
| [strokeStyle](http://www.w3school.com.cn/tags/canvas_strokestyle.asp" \o "HTML5 canvas strokeStyle 属性) | 设置或返回用于笔触的颜色、渐变或模式 |
| [shadowColor](http://www.w3school.com.cn/tags/canvas_shadowcolor.asp" \o "HTML5 canvas shadowColor 属性) | 设置或返回用于阴影的颜色 |
| [shadowBlur](http://www.w3school.com.cn/tags/canvas_shadowblur.asp" \o "HTML5 canvas shadowBlur 属性) | 设置或返回用于阴影的模糊级别 |
| [shadowOffsetX](http://www.w3school.com.cn/tags/canvas_shadowoffsetx.asp" \o "HTML5 canvas shadowOffsetX 属性) | 设置或返回阴影距形状的水平距离 |
| [shadowOffsetY](http://www.w3school.com.cn/tags/canvas_shadowoffsety.asp" \o "HTML5 canvas shadowOffsetY 属性) | 设置或返回阴影距形状的垂直距离 |

|  |  |
| --- | --- |
| **方法** | **描述** |
| [createLinearGradient()](http://www.w3school.com.cn/tags/canvas_createlineargradient.asp" \o "HTML5 canvas createLinearGradient() 方法) | 创建线性渐变（用在画布内容上） |
| [createPattern()](http://www.w3school.com.cn/tags/canvas_createpattern.asp" \o "HTML5 canvas createPattern() 方法) | 在指定的方向上重复指定的元素 |
| [createRadialGradient()](http://www.w3school.com.cn/tags/canvas_createradialgradient.asp" \o "HTML5 canvas createRadialGradient() 方法) | 创建放射状/环形的渐变（用在画布内容上） |
| [addColorStop()](http://www.w3school.com.cn/tags/canvas_addcolorstop.asp" \o "HTML5 canvas addColorStop() 方法) | 规定渐变对象中的颜色和停止位置 |

## 线条样式

|  |  |
| --- | --- |
| **属性** | **描述** |
| [lineCap](http://www.w3school.com.cn/tags/canvas_linecap.asp" \o "HTML5 canvas lineCap 属性) | 设置或返回线条的结束端点样式 |
| [lineJoin](http://www.w3school.com.cn/tags/canvas_linejoin.asp" \o "HTML5 canvas lineJoin 属性) | 设置或返回两条线相交时，所创建的拐角类型 |
| [lineWidth](http://www.w3school.com.cn/tags/canvas_linewidth.asp" \o "HTML5 canvas lineWidth 属性) | 设置或返回当前的线条宽度 |
| [miterLimit](http://www.w3school.com.cn/tags/canvas_miterlimit.asp" \o "HTML5 canvas miterLimit 属性) | 设置或返回最大斜接长度 |

## 矩形

|  |  |
| --- | --- |
| **方法** | **描述** |
| [rect()](http://www.w3school.com.cn/tags/canvas_rect.asp" \o "HTML5 canvas rect() 方法) | 创建矩形 |
| [fillRect()](http://www.w3school.com.cn/tags/canvas_fillrect.asp" \o "HTML5 canvas fillRect() 方法) | 绘制“被填充”的矩形 |
| [strokeRect()](http://www.w3school.com.cn/tags/canvas_strokerect.asp" \o "HTML5 canvas strokeRect() 方法) | 绘制矩形（无填充） |
| [clearRect()](http://www.w3school.com.cn/tags/canvas_clearrect.asp" \o "HTML5 canvas clearRect() 方法) | 在给定的矩形内清除指定的像素 |

## 路径

|  |  |
| --- | --- |
| **方法** | **描述** |
| [fill()](http://www.w3school.com.cn/tags/canvas_fill.asp" \o "HTML5 canvas fill() 方法) | 填充当前绘图（路径） |
| [stroke()](http://www.w3school.com.cn/tags/canvas_stroke.asp" \o "HTML5 canvas stroke() 方法) | 绘制已定义的路径 |
| [beginPath()](http://www.w3school.com.cn/tags/canvas_beginpath.asp" \o "HTML5 canvas beginPath() 方法) | 起始一条路径，或重置当前路径 |
| [moveTo()](http://www.w3school.com.cn/tags/canvas_moveto.asp" \o "HTML5 canvas moveTo() 方法) | 把路径移动到画布中的指定点，不创建线条 |
| [closePath()](http://www.w3school.com.cn/tags/canvas_closepath.asp" \o "HTML5 canvas closePath() 方法) | 创建从当前点回到起始点的路径 |
| [lineTo()](http://www.w3school.com.cn/tags/canvas_lineto.asp" \o "HTML5 canvas lineTo() 方法) | 添加一个新点，然后在画布中创建从该点到最后指定点的线条 |
| [clip()](http://www.w3school.com.cn/tags/canvas_clip.asp" \o "HTML5 canvas clip() 方法) | 从原始画布剪切任意形状和尺寸的区域 |
| [arc()](http://www.w3school.com.cn/tags/canvas_arc.asp" \o "HTML5 canvas arc() 方法) | 创建弧/曲线（用于创建圆形或部分圆） |
| [arcTo()](http://www.w3school.com.cn/tags/canvas_arcto.asp" \o "HTML5 canvas arcTo() 方法) | 创建两切线之间的弧/曲线 |
| [isPointInPath()](http://www.w3school.com.cn/tags/canvas_ispointinpath.asp" \o "HTML5 canvas isPointInPath() 方法) | 如果指定的点位于当前路径中，则返回 true，否则返回 false |

## 转换

|  |  |
| --- | --- |
| **方法** | **描述** |
| [scale()](http://www.w3school.com.cn/tags/canvas_scale.asp" \o "HTML5 canvas scale() 方法) | 缩放当前绘图至更大或更小 |
| [rotate()](http://www.w3school.com.cn/tags/canvas_rotate.asp" \o "HTML5 canvas rotate() 方法) | 旋转当前绘图 |
| [translate()](http://www.w3school.com.cn/tags/canvas_translate.asp" \o "HTML5 canvas translate() 方法) | 重新映射画布上的 (0,0) 位置 |
| [transform()](http://www.w3school.com.cn/tags/canvas_transform.asp" \o "HTML5 canvas transform() 方法) | 替换绘图的当前转换矩阵 |
| [setTransform()](http://www.w3school.com.cn/tags/canvas_settransform.asp" \o "HTML5 canvas setTransform() 方法) | 将当前转换重置为单位矩阵。然后运行 transform() |

## 文本

|  |  |
| --- | --- |
| **属性** | **描述** |
| [font](http://www.w3school.com.cn/tags/canvas_font.asp" \o "HTML5 canvas font 属性) | 设置或返回文本内容的当前字体属性 |
| [textAlign](http://www.w3school.com.cn/tags/canvas_textalign.asp" \o "HTML5 canvas textAlign 属性) | 设置或返回文本内容的当前对齐方式 |
| [textBaseline](http://www.w3school.com.cn/tags/canvas_textbaseline.asp" \o "HTML5 canvas textBaseline 属性) | 设置或返回在绘制文本时使用的当前文本基线 |

|  |  |
| --- | --- |
| **方法** | **描述** |
| [fillText()](http://www.w3school.com.cn/tags/canvas_filltext.asp" \o "HTML5 canvas fillText() 方法) | 在画布上绘制“被填充的”文本 |
| [strokeText()](http://www.w3school.com.cn/tags/canvas_stroketext.asp" \o "HTML5 canvas strokeText() 方法) | 在画布上绘制文本（无填充） |
| [measureText()](http://www.w3school.com.cn/tags/canvas_measuretext.asp" \o "HTML5 canvas measureText() 方法) | 返回包含指定文本宽度的对象 |

## 图像绘制

|  |  |
| --- | --- |
| **方法** | **描述** |
| [drawImage()](http://www.w3school.com.cn/tags/canvas_drawimage.asp" \o "HTML5 canvas drawImage() 方法) | 向画布上绘制图像、画布或视频 |

**confirm 消息对话框通常用于允许用户做选择的动作，如：“你对吗？”等。**

**弹出对话框(包括一个确定按钮和一个取消按钮)。**

var mymessage=confirm("你喜欢JavaScript吗?");

if(mymessage==true)

{ document.write("很好,加油!"); }

else

{ document.write("JS功能强大，要学习噢!"); }

**prompt弹出消息对话框,通常用于询问一些需要与用户交互的信息。**

prompt(str1, str2); str1:

要显示在消息对话框中的文本，不可修改。str2：文本框中的内容，可以修改

例如：var myname=prompt("请输入你的姓名:");

if(myname!=null)

{ alert("你好"+myname); }

else

{ alert("你好 my friend."); }

**this**

this代表上一个元素如果没有则为body

**下标**

os=aa．getElementsByTagName("li"); //可以取出其中的所有li标签

os[i].onclick=function(){ls.href="css/"+this.innerHTML+".css";};对某个li标签点击事件，

**innerHTML 属性用于获取或替换 HTML 元素的内容。**

例子：Object.innerHTML=“”；

this.innerHTML意为获取当前对象内容。

**显示和隐藏（display属性）**

Object.style.display = none;

**清空所设置的所有style样式**

Object.style=’none’

**alert 弹出消息对话框**

alert("hello!");

alert(mynum);

**window.open([URL], [窗口名称], [参数字符串])打开新的链接窗口**



例如：

window.open('http://www.imooc.com','\_blank','width=300,height=200,menubar=no,toolbar=no,status=no,scrollbars=yes')

**关闭窗口（window.close）**

<窗口对象>.close(); //关闭指定的窗口window.close();//关闭本窗口

var mywin=window.open('ht tp://www.imooc.com'); //将新打的窗口对象，存储在变量mywin中

mywin.close();

注意:上面代码在打开新窗口的同时，关闭该窗口，看不到被打开的窗口。

**窗口尺寸发生改变事件window.onresize=函数**

window.onresize=function(){调用函数};

**获取页面可视区域的宽高**

宽：var w=document.documentElement.clientWidth;

高：var h=document.documentElement.clientHeight;

**indexOf（‘字符串’）查找字符串位置**

indexOf("01.jpg")=-1；查找字符串，若查找到则返回位置，若没有查到则返回-1；

返回str在父串中的位置，若父串中不包含str，则返回-1

**获取该对象的所有样式，并取得其中的某个元素**

例: var w1=window.getComputedStyle(o,null).width;

var h1=window.getComputedStyle(o,null).height;

# 控制伪类元素

通过添加class，在其后面添加：after或其他，在js中通过增删class来操作伪类元素。

# 移除ID或class

移除id或者class需要在document.body.removeChild

无缝动画

requestAnimationFrame属性是专为动画所做的，类似于settimeout那俩，制作的动画会无缝动。当我们写window.requestAnimationFrame(回调函数)时，浏览器会在下次重绘前执行回调函数。

用法function render(){

// 一些更新界面的操作

requestAnimationFrame(render);

}

render();

网址<http://www.jianshu.com/p/98e32000b0c9>

# JSON

parse用于从一个字符串中解析出json对象,如

var str = '{"name":"huangxiaojian","age":"23"}'

结果：

JSON.parse(str)

stringify()用于从一个对象解析出字符串，如

var a = {a:1,b:2}

结果：

JSON.stringify(a)

# 页面跳转，数据

self.location="./home.html";在本页面打开。

window.location.href="news-msg.html"在本页面打开。

window.open("./home.html");新建页面打开。

window.navigate(url):加载指定的url所指定的位置上的文档，替换当前文档，即从新加载加载页面为url。

top.location='news-msg.html';

如果不是iframe或者frameset的话，top=window。top.location.href=window.location.href。

如果网页是内嵌网页的话，则不相同。

这句话的作用是如果当前网页不是直接打开的顶层网页而是内嵌网页的话，会重新加载使当前网页为顶层网页。这种办法一般为了防止别人恶意盗链，或者在登录页上用的比较多。

window.history.back(-1);浏览记录返回上一级。

局部跳转

ifram可以实现局部跳转。

存储：window.sessionStorage.desition=a;

获取：var a=sessionStorage.getItem("desition");若是对象或者数组形式可以转换为json格式的。

### 各种嵌套页面传参

条件：以下支持火狐浏览器，若在谷歌浏览器中使用，子窗口需要配置父窗口的跨域。

1、js在子窗口获取父窗口的window

window.parent

//此时可以调用父窗口的window方法，比如获取父窗口的高度window.parent.innerHeight，宽度window.parent.innerWidth，移动父窗口的位置等，具体的可以看本人博客的《获取浏览器窗口的尺寸及对浏览器窗口的一些操作》篇。

2、js在父窗口获取子窗口的window

$("#testiframe")[0].contentWindow==document.getElementById("testiframe").contentWindow==window.frames[0]

以上三种都可以获取子窗口的window,上面三种对应的<body>如下所示：

<div style="border:1px solid black;">

<iframe id='testiframe' height='400px' width='600px' src='xxxxx'></iframe>

</div>

3、jquery在iframe子页面获取父页面元素代码如下:

$("#objid",parent.document)

4、jquery在父页面获取iframe子页面的元素  
代码如下:

$("#objid",document.frames('iframename').document)

5、js在iframe子页面获取父页面元素代码如下:

window.parent.document.getElementByIdx\_x("元素id");

6、js在父页面获取iframe子页面元素代码如下:

window.frames["iframe\_ID"].document.getElementByIdx\_x("元素id");

7、子窗口iframe内调用父窗口函数：

parent.func();

//前提是父窗口的js中必须有func()这个方法。

8、父窗口中调用子窗口函数

document.getElementById(子窗口 ifram 元素的ID值).contentWindow.子窗口中的方法()

子窗口 ifram 元素的name值.window.子窗口中的方法()

下面是我写的一个demo，大家在测试示例时，注意要在火狐浏览器下测试，原因请看本文的开头

父窗口页面  outter.html

[html] [view plain](http://blog.csdn.net/mooncom/article/details/53130756" \o "view plain) [copy](http://blog.csdn.net/mooncom/article/details/53130756" \o "copy)

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge,chrome=1">

    <title>Document</title>

    <script>

        function outter(){

            alert("这是父窗口中的方法");

        }

        function parent(){

            //Name.window.inner();          //第一种方法

            document.getElementById("ID").contentWindow.inner();

        }

    </script>

</head>

<body>

    <iframe src="inner.html" frameborder="1" id="ID" name="Name"></iframe>

    <hr>

    这是父窗口页面

    <br>

    <input type="button" value="点击调用子窗口的方法" id="but1" onclick="parent()"><br>

</body>

</html>

子窗口页面   inner.html

[html] [view plain](http://blog.csdn.net/mooncom/article/details/53130756" \o "view plain) [copy](http://blog.csdn.net/mooncom/article/details/53130756" \o "copy)

<!DOCTYPE html>

<html lang="zh-cn">

<head>

    <meta charset="UTF-8">

    <title>Document</title>

    <script>

    function inner(){

        alert("这是子窗口中的方法");

    }

    function child(){

        //window.parent.outter();                 //第一种方法

        parent.outter();

    }

    </script>

</head>

<body>

    这是子窗口页面

    <br>

    <input type="button" value="点击调用父窗口的方法" class="but2" onclick="child()">

</body>

</html>

# 异步循环数据

循环的情况下异步数据无法正常获取，需要通过闭包的形式让异步数据发送请求，让循环继续进行。每一个异步应该为单独的形式。

例： for(var i=0;i<50;){

(function(num){

setTimeout(function aa(){

console.log(num)

},1000)

})(i)

i++;

}

# 获取页面传参

function GetQueryString(name) {  
 var reg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)", "i");  
 var r = window.location.search.substr(1).match(reg); //获取url中"?"符后的字符串并正则匹配  
 var context = "";  
 if (r != null)  
 context = r[2];  
 reg = null;  
 r = null;  
 return context == null || context == "" || context == "undefined" ? "" : context;  
}