**JS**

**JS严格区分大小写**

**数据类型：number string boolean null undefined object**

**组成部分：**Dom 文档对象模型

　　 Bom 浏览器对象模型

　　 ECMAscript 核心

　　js起源 javascript==>布兰登艾奇 面向对象的语言 脚本语言

　　js作用：可以和页面做交互

　　变量：在程序执行过程中其值可以改变的量。

　　常量：在程序执行过程中其值不可以改变的量。

**定义变量结构：**

　　 关键字：**var**

　　 结构： var 变量名=变量值

变量、数据类型与表达式上

变量:在程序执行过程中其值可以改变的量

多个变量可以在一起定义,中间以**逗号**分隔

var a=1，b='str';

**var 可以省略 加var可以使变量具有变量声明提升的作用**

常量与变量

　　 常量:在程序执行过程中其值不可以改变的量

如何定义变量？==>关键字:var

var 变量名=变量值; //js 弱类型语言

变量默认是没有类型 变量的类型取决于变量的值

**5种基本数据类型**

**number 数值** ==>整型、浮点型统称为number类型

NaN not a Number 不是一个数值 是Number类型的一种

console.log(3-'abc') //NaN

console.log(typeof NaN) //number

isNaN() 判断一个值是否是非数值 返回值为 布尔型

console.log(isNaN(3)) //false

console.log(isNaN('XYZ')) //true

**string 字符串**==>被一对单引号或双引号包含的内容==>引号必须成对出现

字符串拼接

**console.log() //控制台打印**

**document.write() //页面中打印文本或标签**

张三今年18岁,性别男

要求 年龄和性别用变量表示 要求:年龄加粗 性别斜体 在页面中打印出来

var age=18,sex='男';

ES5 字符串拼接: 将非变量部分用引号包起来,如果拼接变量 则用'+'做连接符

document.write('张三今年<b>'+age+"</b>岁,性别<em>"+sex+'</em>');

ES6 字符串拼接 :

document.write(`张三今年<b>${age}</b>岁,性别<em>${sex}</em>`);

document.write("<img src='a.png'>");//ES5

document.write(`<img src=a.png>`);//ES6

**null 空类型**==> 空指针对象 他是一种特殊的对象(对象占位符)

**undefined 未定义类型=**=>=如果定义了变量 没有赋值 则默认保存undefined

var x;

　　　console.log(x) //undefined

**boolean 布尔型** 只有两个值: true（对） false（错）

**复杂数据类型 函数、数组、对象 object**

**typeof(被检测的内容)**  typeof 被检测的内容 操作符 一元(单目)运算符

语法结构：typeof（被检测的内容）

typeof空格 被检测的内容

**typeof 返回值为 string类型**

console.log();//在控制台打印 方便调试代码

console.log(typeof 123); //'number'

console.log(typeof "xyz"); //'string'

console.log(typeof undefined); //'undefined'

　　　 console.log(typeof false); //'boolean'

**console.log(typeof null); //'object'**

console.log(typeof typeof 123); //'string'

console.log(typeof typeof undefined); //'string'

**命名规范**

　　　js中严格区分大小写

var sum,Sum,SUM; //在内存中占着不同的内存

标示符 :变量名、函数名、对象名

标示符的命名规范:

//1.必须由字母、数字、下划线和$组成

//2.必须以字母或下划线或$开头

//3.不能使用关键字和保留字

//4.使用驼峰命名法==>如果变量是复合单词 则除了第一个单词以外的单词首字母大写

//5.语义化==>见名识意

//可选分号 语句的结束

**注释**

//单行注释

/\*多行注释\*/ ctrl+shift+?

注释的作用:1.解释作用 2.阻止代码的执行

**算数运算符**

根据操作数的个数来分类:

一元(单目)运算符 ++ -- ! + -

二元(双目)运算符 + - \* / % > < >= <= != == !== === && ||

三元(三目)运算符==>条件运算符 条件1?条件2:条件3

**加性运算符:** + -

'+':

1) 正常运算

console.log(3+5); //8

2) 字符串拼接

3) 转Number

var str='5';

console.log(typeof +str); //'Number'

console.log(typeof str); //'string'

'-':

1)正常运算

console.log(10-8); //2

2)转Number

console.log(-'5') //-5

**乘性运算符** \* / %

正无穷 任何正数除以0 都等于正无穷 Infinity

负无穷 任何负数初以0 都等于负无穷 -Infinity

console.log(5/0); //Infinity

console.log(-3.14/0); //-Infinity

console.log(0/0); //NaN

自增自减运算符 ++ -- ==>只能被变量使用

分为前置和后置 ++x x++ --x x--

前置:对变量先进行增一(减一)操作,然后再进行运算

后置:先运算,再对变量先进行增一(减一)操作

**alert()==>系统弹出框**

**confirm()==>系统确认框**

Var a=confirm(‘确定吗’);

Console.log(a); 点击确定返回true；点击取消返回false；

**prompt() ==>系统输入框 ，输出方式**

Var a-prompt("提示文字","默认文本");

Console.log(a) 点击确定返回默认文本中输入的内容（内容为string类型）；点击取消返回null（空类型）；

Console.log(typeof a)； 点击确定返回string；

var a;

console.log(a) //undefined

console.log(b) //报错

　　变量在使用之前 必须先定义 如果变量名相同 后定义的覆盖定义的

console.log(typeof a) //'undefined'

console.log(typeof b) //'undefined'

console.log((0.1\*10+0.2\*10)/10==0.3) //true

　　 console.log((0.1+0.2)==0.3) //false

**js两大类**

基本数据类型 number string boolean null undefined

复杂数据类型 obejct 数组 对象 函数

**typeof 的返回值有6种:**

**number string undefined boolean object function**

**1) Number() 强转函数==>可以将任意类型转为数值**

console.log(Number(true));//1

console.log(Number(false))//0

console.log(Number('abc'));//NaN

console.log(Number('5'));//5

console.log(Number('12ab'));//NaN

console.log(Number(''));//0 （无空格）

console.log(Number(' '));//0 （空格）

console.log(Number(null));//0

console.log(Number(undefined));//NaN

**2)parseInt(被转换的字符串) ==>将字符串转为整型**

（1)会忽略前置空格 从第一个非空格字符开始往后查找 直到找到第一个非数值字符 停止 并返回该非数值前的内容

（2)忽略前置空格 如果第一个非空格字符就是非数值 则返回NaN

console.log(parseInt('3.14'));//3

console.log(parseInt(' 45.56'));//45

console.log(parseInt(' w123'));//NaN

console.log(parseInt('100',2));//4 /100是二进制

**3)parseFloat()==>将字符串转为小数**

console.log(parseFloat('12.34'));//12.34

console.log(parseFloat('12.34.56'));//12.34

console.log(parseFloat('12.3a4'));//12.3

console.log(parseFloat('a12.34'));//NaN

console.log(parseFloat('12.3456e2'));//1234.56

**4) +**

console.log(+'5');//5

console.log('5'\*1);//5

**隐式转换==>后台调用Number()方法**

**- \* / % > < >= <= isNaN() != == +**

　　　　　　(当两边都不是字符串时,加号也具有隐式转换功能)

console.log(5-'2'+true); //4

console.log(8.4+'NaN');//'8.4NaN'

console.log(isNaN('5'));//false

NaN

1)任何涉及到NaN的操作 都返回NaN

2)NaN不等于任何数 包括它自己

js规定 undefined只等于它本身和null null只等于它本身和undefined

console.log(undefined==null);//true /==值相等

console.log(undefined===null);//false /===类型不等，值相等

各种类型转字符串

**1) String()==>强转函数 可以将各类型转为字符串**

console.log(String(true));//'true'

console.log(String(123));//'123'

console.log(String(null));//'null'

console.log(String(undefined));//'undefined'

console.log(typeof String(fn));//'String'

**2)\*.toString() ==>可以将除了null和undefined以外的所有类型转为字符串(参数必须是变量)**

var a=3;

console.log(a.toString());//'3'

var x=null;

console.log(x.toString());//报错

字符串下标==>从0开始 ，向右依次递增

　　　var str='qwert'; //

　　　　　　　　　（字符串中首个字符下标为0 ，第二个字符下标为1...）

console.log(str[3]);//‘r’

**\*.length==>获取字符串和数组的长度**

console.log(str.length);//5

获取最后一个字符：console.log(str[str.length-1]);//‘t’

各种类型转布尔

**Boolean()==>强转函数 可以将任意类型转为boolean**

console.log(Boolean(123));//true

console.log(Boolean(-34.5));//true

console.log(Boolean(NaN));//false

console.log(Boolean(0));//false

console.log(Boolean('abc'));//true

console.log(Boolean(''));//false

console.log(Boolean(' '));//true

console.log(Boolean(undefined));//false

console.log(Boolean(null));//false

var obj={}

console.log(Boolean(obj));

**true false**

**Number 除了NaN和0以外的所有数 NaN和0**

**String 除了空串以外的字符串 空串**

**undefined 无 undefined**

**对象(object) 除了null以外的所有对象 null**

表达式的概念

按照特定的规则 由运算符和操作数组成的式子

运算符的分类:

按照操作数个数分类:

一元运算符: ++ -- + - !

二元运算符: \* / % > < == !=

三元运算符： 条件运算符 条件1?条件2:条件3

（条件1为true执行条件2；条件1为false执行条件3；）

操作数类型和结果类型

操作数可以是:基本数据类型 和复杂数据类型

运算符的优先级

() [] > ++ -- ! +(转数值) - > \* / % > > < >= <= > + - (运算、拼接) > && ||

**隐式转换:后台调用Number()方法**

具有隐式转换功能的运算符

+(操作数都不是字符串时) - \* / % < > <= >= == != isNaN()

console.log(true+null);//1

**加性运算符 + -**

+号的作用 1)运算 2) 拼接(加号遇到字符串拼接) 3)转数值

-号的作用 1)运算 2)转数值

**乘性运算符 \* / %**

console.log('5'\*true);//5

**关系运算符 > < <= >= 返回值:boolean类型**

console.log(true>2);//false

console.log(''>-1);//true

相等和不相等运算符 == != === !==

**相等和不相等 == != 会进行隐式转换 只比较值**

console.log('2'==2);//true

console.log(NaN=='NaN');//false (NaN不等于任何值，包括它本身）

console.log(''!=0);//false

**全等和不全等 === !== 不会进行隐式转换 先比较类型是否一致 再去比较值是否相等**

console.log('20'===20);//false

console.log('abc'==='xyz');//false

console.log(null==undefined);//true

console.log(null===undefined);//false （类型不同）

console.log(null=='');//false （null只等于null和undefined)

　　　　　　　　　　　　　　　　（undefined只等于null和undefined)

console.log(3!==3);//false

console.log('5'!==5);//true

**逻辑运算符 !>&&>||**

**! 返回值为boolean类型** 非真即假,非假即真 逻辑非

console.log(!3);//false

console.log(!0);//true

console.log(!!'abc');//true

**&& ==> 逻辑与and**

1)找假(false)==> 当表达式中,有假时,返回第一个为假的值

2)如果表达式中 全部为真,则返回最后一个为真的值

console.log(3&&'abc'+88&&0);//0

console.log(undefined&&'xyz');//undefined

console.log(3\*2 && null && 'qqq');//null

console.log(3&&4);//4

console.log('aaa'&&3\*4);//12

console.log(''+1 && true && 'asdf'); //'asdf'

**逻辑或 ||**

<1>找真==>当表达式中,有真时,返回第一个为真的值

<2>全部为假时,返回最后一个为假的值

console.log( 3|| undefined);//3

console.log('we' || null || 123);//'we'

console.log(null || undefined || false);//false

console.log(''|| 3\*false || NaN);//NaN

console.log(false|| !undefined); //true

Console.log(false|| undefined);// undefined

**初等运算符 () [] ==>优先级最高的运算符**

**逗号运算符** ==>优先级最低的运算符 默认取的是最后一个值

(有括号取最后一个值，无括号取第一个值）

var a;

a=(1,2,3);

console.log(a);// 3

var b;

b=1,2,3;

console.log(b);// 1

**赋值运算符** = 运算顺序 从右到左,左边必须只能是变量

**复合赋值运算符** += -= \*= /= %= 运算顺序 从右到左

var a=3;

a+=4; //==> a=a+4;

a\*=5; //==> a=a\*5;

**条件运算符 ?:**

条件1?条件2:条件3

var x='和嘎嘎哈哈'?alert('我是条件2'):alert("我是条件3");\*/

（条件1为true取条件2，条件1为false取条件2）

**运算符的优先级**

**() [] 》 ++ -- ! 》 \* / % 》 + - 》 < > <= >= 》 == != === !== 》 && 》 || 》 ?: 》 = += -= 》 ,**

**if**语句 **选择结构**(分支结构)的一种 **switch**语句

if语句有**三种形态**

**单分支** ==>判断小括号里的内容 结果是否成立,成立则执行大括号中的内容,不成立则什么都不做

当大括号中的内容 只有一条时 可以省略大号

if(判断条件){

代码块;

}

var a=prompt('输入内容')

if(a>=90){

alert('成才了');

alert('哈哈啊哈');

}

**双分支=**=>判断if后小括号中的内容是否成立 如果成立 则执行if后大括号中的内容 如果不成立 则执行else中的内容

**if else**

if(判断条件){

代码块

}

else{

代码块

}

var age=prompt('请输入你的年龄','在这里输入..');

if(age>=18){

alert('你老啦');

}

else{

alert('你快老了');

}

**多分支 三条及三条以上 if else-if**

90-100 优秀

80-89 良

70-79 中

其他情况不及格

if(){

}

else if(){

}

else if(){

}

.....

else{

}

var score=prompt("请输入你的分数");

if(score>=90 && score<=100){

alert('优秀');}

else if(score>=80 && score<90)

{alert('良');

}

else if(score>=70 && score<80)

{alert('中');

}

else

{alert('不及格');

}

**if语句的嵌套**

var score=prompt("请输入你的分数");

先判断输入的内容是否是数字

再判断输入的数字是否在0--100之间

90-100 优秀

80-89 良

70-79 中

其他情况不及格

if(isNaN(score)){

alert("请输入数字");

}

else if(score>=0 && score<=100){

if(score>=90){

alert('优秀');

}

else if(score>=80){

alert('良');

}

else if(score>=70){

alert('中');

}

else{

alert('不及格');

}

}

else{

alert("请输入正确的区间值");

}

alert('dasd'); //**系统弹出框**

var x=confirm("确定吗");// 系统确认框 点确认返回true 点取消返回false

console.log(x);

**系统输入框**

var x=prompt('提示文字','默认文本'); //点击确认返回输入框中的文本(String类型) 点击取消返回null

console.log(typeof x);//string

Console.log(x);//提示文字

**switch语句 是选择结构的一种**

switch后小括号中和case后可以是任意类型的内容

switch会用 switch后 小括号中的值 和 每个case后的值 做**全等比较** 如果全等 执行 相应case后的代码块

break:立即结束switch的执行 (可选的)

执行语句中,没有遇到break 则会一直往下执行 直到遇到break 如果一支没有遇到 break 则会将switch语句执行完毕

case 和default的位置 不会影响执行效果

多个case 可以执行 同一条语句

default:缺省值 (可选的)

**语法结构:**

switch('a'){

case 'a':代码块;break;

case 'a':代码块;

case 'a':代码块;

case 'a':代码块;

case 'a':代码块;

...

default:代码块;

}

var day=prompt("请输入今天是周几?");

switch(Number(day)){

case 4:alert('四');break;

case 4:alert('六');break;

case 1:alert('一');break;

default:alert('输入有误');break;

case 2:alert('二');break;

case 3:alert('三');break;

case 5:alert('五');break;

case 7:alert('日');

}

输入月份,弹出相应的季节

var month=prompt("请输入月份");

switch(Number(month)){

case 12:

case 1:

case 2:alert('冬天');break;

case 3:

case 4:

case 5:alert('春天');break;

case 6:

case 7:

case 8:alert('夏天');break;

case 9:

case 10:

case 11:alert('秋天');break;

}

**if和switch的区别**

1)可以互相替换

2) 当分支在三条以内时(<=3)使用if语句,当分支在四条及以上(>=4)时 使用switch语句

**选择结构 分支结构 if语句 switch语句**

**循环**

循环结构 for循环 while循环 do-while循环

**当型循环: for while 先判断 后执行**

**直到型循环: do-while 先执行 后判断**

循环:反复执行某一部分操作 意义: 节省代码量,提高运行效率 方便修改维护代码

**do while**

**1.初始值 ;**

**do{**

**4.代码块;**

**3.步长值;}while(2.控制条件 ; )**

var= 0;

do{

console.log(i);

i++;

} while(i < 0)

**For （）里的值为布尔类型**

(1.初始值 ; 2.控制条件 ; 3.步长值 ){

4.代码块

}

**执行顺序: 1 ==> 2 ==> 4 ==> 3 ==> 2 ==> 4 ==>3**

在控制台中打印1--5

<1>for(var i=1; i<=5; i++ ){

console.log(i);

}

var i=1;

<2>for( ; ; ){ //两个分号必须写

if(i>5){

break;

}

console.log(i);

i++;

}

for(var i=1;i<5;i+=1);

console.log(i); //5

循环遍历

在控制台打印1-100的奇数

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

if(i%2==1){

console.log(i);

}

}

在控制台打印1-100的偶数

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

if(i%2!==1){

console.log(i);

}

}

**while**

**1.初值**

**while(2.控制条件){**

**3.步长值**

**4.代码块**

**}**

var i=1;

while(i<10){

console.log(i);

i++;

}

**当循环次数已知时 用for 当循环次数未知时 用while循环**

**Count**  打印次数

珠峰折纸问题

884443m 1m

var zhi=1,mon=884443,count=0;

while(zhi<=mon){

zhi\*=2;

document.write(zhi+" ");

count++;

}

document.write('<br><hr>打印次数'+count)

**打印三位数中的水仙花数**

var a,b,c;

for(var i=100; i<=999; i++){

a=parseInt(i/100); //提取百位数上的数字

b=parseInt((i-a\*100)/10); //提取十位数上的数字

c=i%10; //提取个位数上的数字

if(a\*a\*a+b\*b\*b+c\*c\*c==i){ // 判断是否为水仙花数

console.log(i);}

}

**倒序打印星星**

for(var i=1;i<=5;i++){ //外循环控制行

for(var j=1;j<=6-i;j++){ //内循环控制列

document.write("\*");

}

document.write('<br/>');

}

**倒序打印星**

var i=1;

while(i<=9){

var j=9;

while(j>=i){

document.write('\*');

j--;

}

i++;

document.write('<br />');

}

**正序打印星星**

for(i=1;i<=10;i++){

for(j=1;j<=i;j++){

document.write('☆'+' ')

}

document.write('<br>')

}

**打印九九乘法表**

for(i=1;i<=9;i++){

for(j=1;j<=i;j++){

document.write(j+'\*'+i+'='+i\*j+' ')

}

document.write('<br>')

}

**break和continue 的区别**

break可以在switch和所有循环中 单独使用 立即结束循环

continue 只能在循环中使用 立即结束本次循环,继续执行下一次循环

for(var i=1; i<=10; i++){

if(i%3==0){

continue;

}

console.log(i); } //1 2 4 5 7 8 10

for(var i=1; i<=10; i++){

if(i%3==0){

Break;

}

console.log(i); } //1 2

**if 返回值是boolean;**

**isNaN 返回值是boolean;**

**typeof 返回值是string;**

**parseInt 返回值是number;**

**函数**

**函数定义**

函数:函数是指由事件驱动或可重复调用执行的代码块

**作用:一次封装 多次调用 节省代码量 提高加载速度**

函数的创建

1.**函数声明**

function 函数名(形参列表){

执行代码块;

}

2.**函数表达式**==>将一个匿名函数赋给变量的方式

var fn=function(){

}

**函数声明和函数表达式的区别:**

**函数声明定义函数之后 可以在任何地方调用**

**函数表达式 必须先定义才能调用 如果先调用了会报错**

var fn=function(形参){

代码块 }

fn(实参);

**形参**:形式参数 ==>用于接受实参的变量 写在创建函数时 小括号里的内容

如果形参没有接收到值 默认保存undefined

**实参**:真实参数 ==>用来传递数据给形参的变量 调用函数时 小括号里的内容

参数传递:是单向的 只能 实参传给形参(一次性的) 不能形参传给实参

**arguments** 函数中 用来保存实参的集合(伪数组)

数组用来保存数据的集合

arguments 和形参 映射关系

**agruments和形参存在一一对应的关系**

如果某个形参的值发生了改变, 则相应的arguments值也发生了改变,同样,

如果agruments的某个值 发生了改变 则相应的形参也会发生改变

function fn(a,b){

arguments[0]=50;//a=50

a=10;

console.log(a+b); //14

}

fn(3,4);

**模拟函数重载==>通过arguments的长度 来判断实参个数 做不同的事**

function fn(){

if(arguments.length==1){

console.log(arguments[0]); //7

}

else if(arguments.length==2){

alert(arguments[0]+arguments[1]); //11

}

else if(arguments.length==3){

alert(arguments[0]\*arguments[1]\*arguments[2]); //24

}

}

fn(2,3,4)

fn(5,6)

fn(7)

函数的返回值 return关键字==>返回相应的值到主调函数中

**return的作用: 1. 返回相应的值(可以返回任何类型的内容) 2.立即结束函数的执行**

function fn(a,b){

if((a+b)%2==1){

return true;

}

else{

return false;

}

}

console.log(fn(2,3)) //true

undefined 调用函数 默认值为 undefined

function fn(){

console.log(1); //1

return 2;

console.log(3);

console.log(4);

}

console.log(fn()\*2); //4

function fn(){

return x;

var x=10; //var有变量提升功能

}

Console.log(fn()) //没有返回值 默认返回undefined

**全局变量:**在函数外部定义的变量叫做全局变量 全局变量的作用范围为script标签中

**局部变量:**在函数内部定义的 带var的变量叫做局部变量 **没var的依然是全局变量**

作用范围只在函数中有效 (函数的形参也是局部变量)

var x=5;

function fn(){

console.log(x); // undefined

var x=10;

console.log(x); // 10

}

fn();

**函数自执行：**

**(function(){})();**

**(function(){}());**

**!function(){}(); ==> (!还可以是+ - ~）**

**字符串的创建:**

var str="hello world";

\*.length 获取长度

console.log(str.length); //11

通过 下标来获取相应的字符

console.log(str[4]); //'o'

console.log(str[str.length-1]); //'d'

var newStr=''; 空串长度为0；

console.log(newStr.length); //0

**字符串截取方法 返回值string**

**字符串的创建有两种方式:**

var s = '123';

var s = new String('123');

**字符串.charAt(下标值)==>通过下标 获取字符串中相应的字符**

var str='hello world';

console.log(str.charAt(7)); //'o'

**字符串.charCodeAt(下标值)==>通过下标 获取字符串中相应的字符的ASCII值**

var str='hello world';

console.log(str.charCodeAt(1)); //下标为1的是e；e的ASSCII值为101；

A的ASSCII值为65，B的ASSCII值为66；a的ASSCII值为97(A的ASSCII值+32);

**字符串.slice(起始下标,结束下标);** 左包右不包 遇见负数加字符串长度(length)

var str='hello world';

console.log(str.slice(1));//'ello world'

console.log(str.slice(3,8));//'lo wo'

console.log(str.slice(-4));//'orld'

console.log(str.slice(-1,5));//''

Console.log(str.slice(5,5));//''

console.log(str.slice(5,6));//''

console.log(str.slice(-8,-5));//'lo '

**字符串.substr(起始下标,截取个数);**

var str='hello world';

console.log(str.substr(1));//'ello world'

console.log(str.substr(3,7));//'lo worl'

console.log(str.substr(-4));//'orld'

console.log(str.substr(-1,5));//'d'

console.log(str.substr(-5,-5));//'' （两个都为负值且下标个数一样都为‘’）

**substring(起始下标,结束下标)** 左包右不包

var str='hello world';

console.log(str.slice(2,5));//'llo'

console.log(str.substring(2,5));//'llo'

**console.log(str.slice(5,2));//""**

**console.log(str.substring(5,2));//'llo'**

console.log(str.substring(-1,5));//'hello' **（-1默认为0）遇到负数为0**

console.log(str.substring(-5,-3));//'' **（两个为负数默认为空）**

**查找方法**

**.indexOf(查找字符(串),起始下标)**==> 查找字符(串)在字符串中**首次**出现的位置

(下标)

**通过字符找下标**

**左==>右**

**查找不着 默认返回-1**

var str='hello world';

console.log(str.slice(8,10));//'rl'

console.log(str); //'hello world'

console.log(str.indexOf('e'));//1

console.log(str.indexOf('o'));//4

**console.log(str.indexOf('o',5));//7** (从第五个字符开始第一个o的下标）

console.log(str.indexOf('x'));//-1 **查找不着 默认返回-1**

**console.log(str.indexOf('wor'));//6**

**.lastIndexOf(查找字符(串),起始下标)**==> 查找字符(串)在字符串中**最后**一次出现的位置(下标)

**右==>左**

var str='hello world';

console.log(str.lastIndexOf('o'));//7

**console.log(str.lastIndexOf('lo'));//3**

**console.log(str.lastIndexOf("l",6));//3**

**.toUpperCase() 转大写**

var str='hello world';

console.log(str.toUpperCase());//HELLO WORLD

**.toLowerCase() 转小写**

var x='AbCd';

console.log(x.toLowerCase()); //abcd

**Math.ceil 向上取整 返回值number**

console.log(Math.ceil(3.01));//4

console.log(Math.ceil(3.99));//4

**Math.floor 向下取整 返回值number**

console.log(Math.floor(3.01));//3

console.log(Math.floor(3.99));//3

**Math.round 四舍五入 返回值number**

console.log(Math.round(3.49));//3

console.log(Math.round(3.51));//4

**Math.min 最小值 返回值number**

**console.log(Math.min(5,true,8,null,'abc')); //NaN**

**Math.max 最大值 返回值number**

**Math.random() 随机数** 随即生成0--1之间的数 **[0,1)**

假设 你想获取20--30之间的随机数

**Math.floor(Math.random()\*(最大值-最小值+1)+最小值);**

Math.floor(Math.random()\*11+20);

console.log(Math.random());

**Math.abs() 取绝对值**

console.log(Math.abs(1),Math.abs(-1)); //1 1

**Math.pow(m,n) 求m的n次幂**

console.log(Math.pow(2,3)); //8

var str='qwertyuiosdfghjkfgdsja';

连缀写法

console.log(str.slice(0,str.length-1)+str[str.length-1].toUpperCase());//

'qwertyuiosdfghjkfgdsjA'

**数组**是什么? 在内存在占用连续内存 保存数据的集合

**返回值object**

数组的创建

//1.

**var arr=[]; //数组字面量(直接量)**

//2.**用构造器(构造函数)创建数组**

var arr=new Array(5);// 构造器创建函数时 如果小括号中的值为 单个数值时 则被视为数组长度

**数组元素的读(获取) 写(设置)**

var arr=['a',null,undefined];

console.log(arr[0]);//'a'

console.log(arr[arr.length-1]);//undefined

arr[10]='xyz'; console.log(arr.length); //11

arr[1]=123; console.log(arr.length); //3

arr[100]='asdas'; console.log(arr.length);//101

**清空数组**

arr.length=0;

console.log(arr); //[]

**删除数组中的最后一个元素**

var arr=['a',null,undefined];

arr.length-=1; （arr.length=arr.length-1;）

console.log(arr); //["a", null]

**.length 属性**

var arr=[1,2,3,4,5,6,'a',null];

console.log(arr.length);//8

**数组的遍历**

var arr=['abc',345,456,'x',undefined];

**正序遍历数组**

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

console.log(arr[i]);

}

倒叙遍历数组

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

console.log(arr[i]);

}

var arr=[1,2,3,4,5,6,'a',null];

console.log(typeof arr); //' object'

**instanceof 检测类型**

function fn(){}

console.log(fn instanceof Array) false

**转换方法**

**字符串转数组 ==> 字符串.split(分隔符);**

1.如果 小括号中 没有参数,则默认该字符串为一个数组项

2.如果 小括号中 参数为空字符串 则字符串中的每个字符 都是一个数组项

3.如果 小括号中 参数为 字符串中 不存在的字符 则默认该字符串为一个数组项

4.如果 小括号中 参数为字符串中的某个字符 则该字符变为逗号,逗号与逗号之间如果没有

内容 则默认为 空字符串,如果该字符为前置或后缀字符,则前置字符前和后缀字符后需添加空字符串

Var str='abbaceattyar';

console.log(str.split());//["abbaceattyar"]

console.log(str.split(""));//["a", "b", "b", "a", "c", "e", "a", "t", "t", "y", "a", "r"]

console.log(str.split(' '));//["abbaceattyar"]

console.log(str.split('a'));// ["", "bb", "ce", "tty", "r"]

var str='qqqwqrqqqq';

console.log(str.split('q'));//[ '','' ,'' , 'w' , 'r' ,'' ,'' ,'' ,'']

var str='wertwwtwtw';

console.log(str.split('w'));

//[ '' , 'ert' , '' , 't' , 't' , '' ]

**数组转字符串 join(连接符);**

var arr=['xy',123,'bcd'];

console.log(arr.join());//xy,123,bcd

console.log(arr.join('-'));//xy-123-bcd

console.log(arr.join(""));//xy123bcd

var arr=['xy',123,'bc'];

**栈方法 ==>会改变原数组**

**push(添加项1,添加项2....)** 在队尾添加一个或多个项,返回值为新数组长度

console.log(arr.push(5,6)); //5

console.log(arr); //['xy',123,'bc',5,6]

**pop()**==> 在队尾删除一个项,返回值为被删除的项

console.log(arr.pop());//'bc'

console.log(arr);//['xy',123]

**队方法 ==>会改变原数组**

**unshift(添加项1,添加项2....)** 在队头添加一个或多个项 返回值为 新数组长度

console.log(arr.unshift(1,'we'));//5

console.log(arr);//[1,'we','xy',123,'bc']

**shift()**==>在队头删除一个项,返回值为被删除的项

console.log(arr.shift());//'xy'

console.log(arr);//[123,'bc']

**重排序方法 ==>会改变原数组**

**reverse() 数组翻转**

var arr=[4,5,2,1,3];

console.log(arr.reverse());//[3, 1, 2, 5, 4]

console.log(arr);//[3, 1, 2, 5, 4]

**sort() 数组排序**==>**默认按照 数组项的ASCII值排序 从小到大**

**会改变原数组**

var arr=[12,4,222,34];

console.log(arr.sort()); //[12, 222, 34, 4]

**console.log(arr.sort(function(x,y){**

**return x-y; //第一个参数减第二个参数 升序排序 //[4, 12, 34, 222]**

**return y-x; //第二个参数减第一个参数 降序排序 //[222, 34, 12, 4]**

**}));**

Console.log(arr); //[222, 34, 12, 4]

**操作方法**

**数组.concat(拼接项1,拼接项2...) 数组拼接(拼接常量)**

返回值为:拼接后的数组， **原数组不变**

var arr=[1,2],arr1=['ab','c'],arr2=['ty',null];

console.log(arr.concat(undefined,456)); //[1, 2, undefined, 456]

console.log(arr) //[1, 2]

**数组.slice() 数组截取**

var arr=[4,5,1,'a',undefined];

console.log(arr.slice(1,3)); //[5,1]

**数组.splice(起始下标,删除个数,添加项1,添加项2...) 会改变原数组**

返回值为被删除项组成的数组

数组的增删改方法

var arr=[1,3,'aaa',null];

console.log(arr.splice(2,3,4,5));//["aaa",null]

console.log(arr);//[1,3,4,5]

var arr=[5,4,'ab',undefined,250];

console.log(arr.splice(1,2,3,4)); //[4,'ab']

console.log(arr);//[5,3,4,undefined,250]

var arr=[23,45,11,78,'ab'];

console.log(arr.splice(1,0,34,44));//[]

**（删除个数为0；无删除返回值为[]）**

console.log(arr);//[23, 34, 44, 45, 11, 78, "ab"]

**使用prompt弹出输入框，要求用户输入一个英文的语句，将输入语句的第二个单词截取出来**

var str=prompt("请输入一句话");

var arr=str.**split**(" "); **（split字符串转数组）**

alert(arr[1]);

**数组.replace(被替换项,替换项);** 将被替换项首次出现的位置 替换成 替换项

**不会改变原数组**

var str='hello world';

console.log(str.replace('o',1)); //hell1 world

console.log(str) //hello world

**日期对象创建**

**var 变量名=new Date(你要设置的时间);**

**var time=new Date();**

console.log(time);//Thu Sep 06 2018 13:39:46 GMT+0800 (中国标准时间)

console.log(typeof time);// **'object' 数组**

unction fn(){} console.log(typeof fn);//**'Function '**

**toLocaleString() ==>按照当地的日期格式显示时间和日期**

console.log(typeof time.toLocaleString()); //**’String’**

**toLocaleDateString() 按照当地格式 显示 日期**

console.log(time.toLocaleDateString());//2018/9/6

console.log(time.toLocaleTimeString());//下午1:47:59(当前时间）

**toString() 可以将除了null和undefined以外的所有类型转为字符串**

console.log(time.toString());//'Thu Sep 06 2018 13:46:17 GMT+0800 (中国标准时间)'

**时间戳**==> 1970年1月1日00:00:00 到现在的时间的**毫秒**形式

**日期对象.valueOf(); ==>将日期对象转时间戳**

console.log(time.valueOf()); //\*\*\*\*\*\*\*\*\*\*\*\*

console.log(time\*1); //\*\*\*\*\*\*\*\*\*\*\*\*

console.log(+time); //\*\*\*\*\*\*\*\*\*\*\*\*

console.log(time.getTime()); //\*\*\*\*\*\*\*\*\*\*\*\*

**日期对象.getFullYear() 获取年份**

console.log(time.getFullYear());//2018

**日期对象.getMonth() 获取月份**

console.log(time.getMonth());//[0,11] **0代表1月 11代表12月**

**日期对象.getDate() 获取日期**

console.log(time.getDate());//[1,31]

**日期对象.getDay() 获取星期**

console.log(time.getDay());// **1-6 依次代表 星期一到六 0代表星期天**

**日期对象.getHours() 获取小时**

console.log(time.getHours()); //[0,23] 0代表24时

**日期对象.getMinutes() 获取分钟**

console.log(time.getMinutes());//[0,59]

**日期对象.getSeconds() 获取秒数**

console.log(time.getSeconds());//[0,59]

**日期对象.getMilliSeconds() 获取毫秒**

console.log(time.getMilliseconds());//[0,999]

var timer=new Date();

**日期对象.setTime(设置时间(毫秒形式));** ==>

console.log(timer.setTime(1000)); //1000

console.log(timer); //Thu Jan 01 1970 08:00:01 GMT+0800 (中国标准时间)

**setFullYear(设置的年份)**

console.log(timer.setFullYear(2096)); //\*\*\*\*\*\*\*\*\*\*\*\*

**setMonth(设置的月份)**

console.log(timer.setMonth(7)); //\*\*\*\*\*\*\*\*\*\*\*\*

**setDate(设置的日期);**

timer.setDate(40);

**setHours(设置小时)**

timer.setHours(18);

**setMinutes(设置分钟)**

timer.setMinutes(55);

**setSeconds(设置秒)**

timer.setSeconds(70);

console.log(timer) //Sun Sep 09 2096 18:56:10 GMT+0800 (中国标准时间)

**日期对象的容错能力**

var time=new Date("2008-8-8 20:00:00");

//日期之间可以 以 , / - 连接 时间直接以:连接

var time=new Date(2008,17,45);

Console.log(time); //2009 7 15

var time=new Date(2020,2,0);

console.log(time);//2020-2-29

console.log(new Date(2010,5,0).getDate());//31

console.log(new Date(2022,33,44)); //2024 10 13

**如果要操作日期对象 则必须先new Date()**

**（forEach\map\filter\some\every）数组的五大迭代方法**

只有forEach没有返回值 其它都有

**var arr=[2,3,1,4,5];**

**forEach 数组遍历**

**var val=arr.forEach(function(item){**

**console.log(item);**

**})**

**console.log(val)**

**every检测数组项中是否 满足条件 全部满足返回true 否则返回false**

var val=arr.every(function(item){

return item>0;

})

console.log(val); //true

**some检测数组项中是否 满足条件 只有有一个满足条件则返回true 全部不满足则返回false**

var val=arr .some(function(item){

return item>5;

})

console.log(val); //false

**map 返回 条件和数组中的每一项比较的结果 组成的数组**

var val=arr.map(function(item){

return item>2;

})

console.log(val); //[false, true, false, true, true]

**filter 返回数组项中 满足条件的数组项组成的数组**

var arr=['王峰','王我','赵丹丹','王五','陈帅'];

var val=arr.filter(function(item){

return item.charAt(0)=='王';

})

console.log(val); //["王峰", "王我", "王五"]

**DOM**

**JS的组成部分:** DOM文档对象模型 BOM浏览器对象模型 ECMAscript JS核心

DOM文档对象模型 document object model

html结构 css样式 js 行为

**节点是什么? 浏览器将html源文件中的 所有内容都视为节点**

<div></div> 标签(元素)节点

//哈哈哈 注释节点

<p style="width:200px;"></p> 属性节点

整个document 被称为 文档节点

<div>我是div</div> //文本节点

**元素中文本默认的类型为string类型**

**<body><!-- 哈哈 -->**

**<li class="qq">77777</li>**

**<div id="box" class='www' name='zxc'>**

**<li>11</li>**

**<li class="li">**

**<p>vvvv</p>**

**<p>qwe</p>**

**<p>saa</p>**

**</li>**

**<li class="qq">33</li>**

**<li>44</li>**

**<p name="ww"></p>**

**<div id="sonBox" class="qq" name="ww"></div>**

**<p name="ww"></p>**

**</div>**

**<li>55555</li>**

**<li name='ww'>66666</li>**

**</body>**

**document.getElementById('id名')** 效率最高 性能最优

通过id名获取元素只能获取一个

var box=document.getElementById('box');

var sonBox=document.getElementById('sonBox');

**document.getElementsByTagName('标签名');**

通过标签名 获取元素的集合==>伪数组的形式

var lis=box.getElementsByTagName('li');

console.log(lis[lis.length-1]); 下标 //**<li>44</li>**

**(id名为box下最后一个li）**

**document.getElmentsByClassName('class名');**

通过class名 获取元素的集合==>伪数组的形式

var qqs=box.getElementsByClassName('qq');

console.log(qqs) //（**id名为box下所有类名为qq）**

**document.getElementsByName('name名');**

通过name名 获取元素的集合==>伪数组的形式

var ps=document.getElementsByName('ww');

**document.querySelector();**

通过标签名、类名、id名 属性获取首个节点

var p=document.querySelector('#box .li p'); //<p>vvvv</p>

**document.querySelectorAll();**

通过标签名、类名、id名 属性获取全部节点

**返回值为 伪数组 NodeList(节点集合)**

var ps=document.querySelectorAll('#box .li p');

console.log(ps); //[p, p, p]

**节点的属性 nodeName nodeValue nodeType**

var x=document;

console.log(x.nodeName,x.nodeValue,x.nodeType);

var x=document.getElementsByTagName('body')[0];

console.log(x.nodeName,x.nodeValue,x.nodeType)

var x=document.getElementById('box');

attributes 用来保存节点的属性的集合(伪数组)

console.log(x.attributes[1].nodeName,x.attributes[1].nodeValue,x.attributes[1].nodeType);

var x=document.querySelector("#box .li p").firstChild;

console.log(x.nodeName,x.nodeValue,x.nodeType)

var node=document.querySelector('body').firstChild;

console.log(node.nodeName,node.nodeValue,node.nodeType);

**nodeName nodeValue nodeType**

**元素节点 大写的标签名 null 1**

**属性节点 属性名 属性值 2**

**文本节点 #text 文本内容 3**

**注释节点 #comment 注释内容 8**

**文档节点 #document null 9**

**四个节点属性**

**firstChild 获取第一个子节点**

**lastChild 获取最后一个子节点**

**nextSibling 获取下一个兄弟节点**

**PreviousSibling 获取上一个兄弟节点**

var box=document.getElementById('box');

**firstChild** == > dom元素.firstChild element 在该方法中添加element 解决兼容问题

console.log(box.**firstElementChild**); 获取box下的第一个子元素节点

**lastChild** ==> dom元素.lastChild element 在该方法中添加element 解决兼容问题

console.log(box.**lastElementChild**); 获取box下的最后一个子元素节点

**nextSbiling** ==> 获取dom元素下一个兄弟节点 dom元素.nextsbiling

var li2=document.getElementById('li2');

console.log(li2.**nextElementSibling**);

**previousSibling** ==>获取dom元素上一个兄弟节点 dom元素.previousSibling

console.log(li2.**previousElementSibling**);

**childNodes children 节点的默认属性** 默认保存着节点的子节点(伪数组)

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

**childNodes 保存着节点的所有子节点**

console.log(box.childNodes);//[text, p, text, p, text, p, text]

**children 保存着节点的所有子元素节点**

console.log(box.children);

**parentNode dom元素.parentNode 查找dom元素的父节点**

console.log(box.parentNode);

var arr=[1,23,23,21]; //**一维数组**

var newArr=[[1,2],[2,3],[3,4]];//当一个一维数组的数组项都是一维数组,则该数组为**二维数组**

假设 每行 都有7列

问[5][6]之前有多少元素 //41个

**css属性与HTML属性操作**

**标签为文本内容默认字符串**

**style属性** ==> 给节点设置或获取样式 只能获取或设置行内样式 一次设置一个样式

**语法格式: 节点.style.样式名=样式值;**  样式名(驼峰写法)

box.style.backgroundColor='skyblue';

box.style.fontSize='20px';

**cssText** 一次设置多个样式 只能获取或设置行内样式

**语法格式: 节点.style.cssText='样式名1:样式值1;样式名2:样式值2;...';**

box.style.cssText='width:500px;height:500px;background:red;font-size:30px';

var x=box.style.cssText;

console.log(x)

**className属性** 给节点设置或获取类名 返回结果为字符串

**节点.className='类名';**

var x=box.className;

**classList属性** 增加 删除 查找类名 返回结果为类数组

**节点.classList.add(增加的类名1,增加的类名2...);**

box.classList.add('qwer','df','dasdas');

**节点.classList.remove(删除项1,删除项2...);**

var x=box.classList.remove('xxxx');

**节点.classList.contains(要查找项);**

如果查找到该类名返回true 找不到该类名返回false

var x=box.classList.contains('xxx');

**节点.classList.item(下标值); 返回相应下标的类名**

var x=box.classList.item(4);

**操作属性**

**getAttribute() 获取属性**

语法结构

**节点.getAttribute(要获取的属性名);** 返回值为相应的属性值 查找不到返回null

var x=box.getAttribute('name');

console.log(x)

**设置属性**

**setAttribute 设置属性**

语法结构

**节点.setAttribute(要设置的属性名,要设置的属性值);** 返回undefined

**删除属性**

**removeAttribute() 删除属性**

语句结构

**节点.removeAttribute(要删除的属性);**

box.removeAttribute('class');

**attributes==> 默认保存着节点的属性集合==>伪数组形式**

console.log(box.attributes[1])

**HTML级事件模型 (内联模型)**

<button id="box"onclick="alert(1)">点我</button>

<div id="btn">点我</div>

<input type="text" id="ipt">

<script>

**DOM 0级事件** 让你的html具有某些效果 触发效果后 做一些操作

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

**节点.on+'事件名'=function(){**

**}**

**节点.on+"事件名"=function(){**

**//执行函数;**

**fn(1,2);**

**}**

function fn(a,b){}

节点.on+"事件名" = fn;

function fn(){}

**鼠标事件**: click单击 dblclick双击 mousedown按下 mouseup抬起 mouseout移出 mouseover移入 mousemove移动

**焦点事件:** blur失去焦点 focus获得焦点

**键盘事件** keydown按下 keyup抬起 keypress

**UI事件:**change失去焦点并改变内容

var ipt=document.getElementById('ipt');

var i=1;

ipt.onchange=function(){

console.log(++i)

}

getElementById()==>通过id获取节点 (效率最高 性能最优)

getElementsByTagName() 通过标签名获取节点的集合

伪数组

getElementsByClassName() 通过类名获取节点的集合伪数组

getElementsByName() 通过name名获取节点的集合

querySelector() 通过id名 类名 标签获取首个节点

querySelectorAll()通过id名 类名 标签名获取全部节点

**节点的三大属性**

nodeName nodeValue nodeType

**nodeName nodeValue nodeType**

**元素节点 大写的标签名 null 1**

**属性节点 属性名 属性值 2**

**文本节点 #text 文本内容 3**

**注释节点 #comment 注释内容 8**

**文档节点 #document null 9**

**TagName==NodeName 两种属性获取标签名**

**节点属性**

firstChild lastChild nextSibling previousSibling 该四个方法 存在兼容问题 需要在属性中添加Element 解决兼容问题

parentNode 获取父节点

children 节点中默认保存所有子元素节点的集合(伪数组)

childNodes 节点中默认保存所有子节点的集合(伪数组)

**写入方法**

innerHTML 给标签设置或获取 文本内容或标签

innerText 给标签设置或获取 文本内容

**属性方法**

getAttribute()获取属性

setAttribute()设置属性

removeAttribute()删除属性

attributes 默认保存着节点属性的集合

**设置行内样式的属性**

style dom元素.style.样式名(驼峰)='样式值';

style.cssText dom元素.style.cssText='样式名1:样式值1;样式名2:样式值2;...';

**操作类名的属性**

className 默认保存这节点的类名(字符串)

设置(清空)或获取元素的类名

**classList和它的四个方法**

add() 增加类名

remove() 删除类名

contains() 查找类名 如果有返回true 没有返回false

item() 通过下标返回相应的类名

**创建元素和创建文本内容**

**document.createElement('要创建的元素名');**

返回值为创建的元素

/\*var node=document.createElement('div');

console.log(node);\*/

**document.createTextNode('文本内容');**

返回值为创建的文本内容

/\*var text=document.createTextNode("大街上的考拉上来讲",);

console.log(text);\*/

**追加节点**

**父元素.appendChild(要追加的节点)**

/\*var node=document.createTextNode('dajdlksaj');

box.appendChild(node);\*/

**插入节点**

**父元素.insertBefore(要插入的节点,要插入的位置);**

/\*var node=document.createElement('em');

box.insertBefore(node,box.children[1]);

\*/

**删除节点**

**父元素.removeChild(被删除的节点);**

//box.removeChild(box.children[1]);

**替换节点**

**父元素.replaceChild(新节点,被替换节点);**

/\*var node=document.createElement('strong');

box.replaceChild(node,box.children[2]);\*/

**对象 类数组 方便操作数据**

对象的创建

**var obj={} //对象字面量**

**var obj=new Object();//构造函数创建对象**

在对象中 基本类型被称为对象的属性

复杂类型被称为对象的方法

/\* var obj={

name:'哈哈', //键值对

age:18,

sex:'',

fn:function(){

console.log(123);

return 1;

}

}

console.log(obj.fn())\*/

动态的去创建属性

/\*var obj={};

obj.kg='75kg';

obj.cm='260cm';

obj.mm='26000mm';

obj.fn=function(){

alert(123);

}

console.log(obj.fn());\*/

**通过id获取元素方法的封装(byId)**

**function $(id){**

**return document.getElementById(id);**

**}**

**判断DOM元素上是否有某个类(hasClass)**

/\*var box=$('box');

function hasClass(node,val){

//将 节点的类名通过空格转数组

var arr=node.className.split(' ');

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

if(val==arr[i]){

return true;

}

}

return false;

}

console.log(hasClass(box,'six'));\*/

**添加类方法的封装(addClass)**

/\*var box=$('box');

function addClass(node,val){

//将 节点的类名通过空格转数组

var arr=node.className.split(' '),

flag=false;

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

if(val==arr[i]){

flag=true;

}

}

if(!flag){node.className+=' '+val}

}

addClass(box,'seven');\*/

**定时器** 属于bom对象==> window对象

**间歇调用**

**setInterval(要执行的内容(函数),间隔时间ms)**

**var i=0;**

**setInterval(function(){**

**console.log(++i)**

**},1000);**

**var timer=setInterval(fn,1000);**

**function fn(){**

**console.log(++i)**

**}**

**setInterval("fn('xyz')",1000);**

**function fn(a){**

**console.log(a);**

**}**

**clearInterval(定时器的返回值) 清除定时器**

var timer=null,i=0; //定义变量 用来保存定时器返回值

btn1.onclick=function(){

timer=setInterval(function(){

console.log(++i);

},1000);

}

btn2.onclick=function(){

clearInterval(timer);

}

**超时调用 setTimeout(执行代码,间隔时间) 只执行一次**

**清除超时调用 clearTimeout**

**o.name ==> o["name"]**

function getStyle( obj , styleName ){

if( obj.currentStyle ){//IE浏览器下获取DOM元素的样式

return obj.currentStyle[styleName];

}else{

return getComputedStyle(obj,null)[styleName];

// 非IE浏览器下获取DOM元素的样式

}

}

getStyle( box , "width" );

// **IE浏览器下获取DOM元素的样式**

dom元素.currentStyle.styleName

// **非IE浏览器下获取DOM元素的样式**

var box=document.getElementById('box');

console.log(getComputedStyle(box,'sdsa').width);

getComputedStyle( dom对象 , null).styleName;

和style的区别:

style可以设置或获取样式 但是getComputedStyle只能获取

style只能获取或者设置行内样式 getComputedStyle可以获取行内、内嵌、外联

**offset... 返回值都是数值 (不带单位)**

div: width:100px boder-left:2px padding:3px margin:4px;

**offsetWidth** ==> width+ border + padding 108

**offsetHeight**==> height + border + padding 6

**offsetLeft** ==> left + margin-left 30

**offsetTop** ==> top + margin-top 0

**scrollTop ==> 返回滚动条距离顶部位置**

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

console.log(document.documentElement.scrollTop);

**BOM**是什么？浏览器对象模型

**broswer object model**

**window对象**

在全局作用域下定义的变量被当做window下的属性

在全局作用域下定义的函数被当做window下的方法

/\* var a=1;

console.log(a);

console.log(window.a);

function fn(){

console.log(123456);

}

window.fn();\*/

**location()** ==>

**search 路径中?及以后的内容**

**hash #及以后的内容**

**href 完整路径**

**iframe()**

iframe的优点是重载页面时不需要重载整个页面，只需要重载页面中的一个框架页（减少数据的传输，减少网页的加载时间)

**open() 打开**

**close()**

**navigator() 保存浏览器信息**

**navigator.userAgent 返回浏览器头部信息**

**history() 保存浏览器历史记录**

//history(0) 刷新页面

//history(1) 往下一页

//history(-1) 往上一页