希望把某个元素移除你的视线：

1、display:none;                显示为无

2、visibility:hidden;    隐藏

3、width \ height

4、透明度

5、left \ top

6、拿一个白色DIV盖住它

7、margin负值

 ...

JS中如何获取元素：

1、通过ID名称来获取元素：

         document get element by id 'link'

          docuemnt.getElementById('link');

2、事件：鼠标事件、键盘事件、系统事件、表单事件、自定义事件……

        onclick

        onmouseover

        onmouseout

        onmousedown

        onmouseup

        onmousemove        就像是鼠标抚摸一样的事件

        ……

        onload                加载完以后执行……

        window.onload = 事情

        img.onload

        body.onload

        ……

如何添加事件：

        元素.onmouseover

 函数：可以理解为-命令，做一些事~~

        function abc(){        // 肯定不会主动执行的！

            ……

        }

        1、直接调用：abc();

        2、事件调用：元素.事件 = 函数名 oDiv.onclick = abc;

        function (){}        匿名函数

        元素.事件 = function (){};

 测试：

        alert(1);        带一个确定按钮的警告框

        alert('ok');    'ok' 字符串

        alert("ok");

  变量：

        var li = document.getElementById('lis');

        var num = 123;

        var name = 'leo';

// JS 不允许出现"-"

**数据类型、类型转换**

HTML 标签类型：block、inline、inline-block、table……

JS中的数据类型：数字、字符串、布尔、函数、对象(obj、[]、{}、null)、未定义

typeof 判断数据类型

var a = '+100';

// alert( a+100 );        // '100100'

// alert( Number(a) );            // 100

var a1 = ' ';

// alert( Number(a1) );        // 0

var a2 = true;

// alert( Number(a2) );            // true-1 false-0

var a3 = [ 1 ];

// alert( Number(a3) );            // 1 0

var a4 = null;

// alert( Number(a4) );                // 0

// var b = ' 200px';

// alert( parseInt(b) );

var c = '12.34元';

// alert( parseFloat(c) );

var num = '200.45';

if( parseInt(num) == parseFloat(num) ){

    alert( num + '是整数' );

}else{

    alert( num + '是小数' );

}

////////////////////////////////////////////////////////////////////////////

var json = {};

// alert( Number(json) );

var u;

// alert( Number(u) );

var a3 = function (){ alert(1); };

// alert( Number(a3) );            // NaN

显式类型转换（强制类型转换）：

Number()

parseInt()

parseFloat()

隐式类型转换：

    +                                200 + '3'        变成字符串

    - \* / %                '200' - 3 变成数字

    ++ --                        变成数字

    > <                         数字的比较 、字符串的比较

    !    取反                    把右边的数据类型转成布尔值

    ==

\*/

// alert( Number('……') );            NaN

// alert( '……'-9 );                                    NaN

// alert( '2' == 2 );

// alert( '10000000' > '9' );

// 数字的比较与字符串的比较

// '1000000' '9'

// alert( '2' === 2 );

// JS中的数据类型：数字（NaN）、字符串、布尔、函数、对象(obj、[]、{}、null)、未定义

var a = Number('abc');

// alert( a );            // NaN

// NaN：not a number 不是个 数字 的 数字类型

// alert( typeof (a) );            // number

// 一旦写程序中出现：NaN 肯定进行了非法的运算操作

// alert( '200px' - 100 );

// NaN 是 false

if( a ){

    // alert( '真' );

}else{

    // alert( '假' );

}

var abc = [];

// alert( abc === abc );                true

// NaN 与自己都不相等！！

alert( a === a );                // false

// NaN：not a number 不是个 数字 的 数字类型

// isNaN：Is not a number 是不是不是一个数字（不是数字）

isNaN();                    // true            false

// 判断某些值是不是数字

// 不喜欢数字、讨厌数字

// alert( isNaN( function(){ alert(1) } ) );

// alert( isNaN('250') );

// Number() '250' => 250 => false

// alert( isNaN( [] ) );

**函数传参**

函数传递参数

    参数=JS的数据类型：

        数字、字符串、布尔、函数、对象、未定义

/\*

    重用代码：

        1、尽量保证 HTML 代码结构一致，可以通过父级选取子元素

        2、把核心主程序实现，用函数包起来

        3、把每组里不同的值找出来，通过传参实现

\*/

**作用域**

// 作用域：

// 域：空间、范围、区域……

// 作用：读、写

    script        全局变量、全局函数

        自上而下

    函数

        由里到外

    {}

    浏览器：

        “JS解析器”

            1）“找一些东西”    ：var function 参数

                         a = ...

                           所有的变量，在正式运行代码之前，都提前赋了一个值：未定义

                            fn1 = function fn1(){ alert(2); }

                                    所有的函数，在正式运行代码之前，都是整个函数块

                            JS 的预解析

                        遇到重名的：只留一个

                            变量和函数重名了，就只留下函数

            2）逐行解读代码：

                        表达式：= + - \* / % ++ -- ! 参数……

                        表达式可以修改预解析的值！

**运算符**

/\*

    真假的问题：数据类型-数字（NaN）、字符串、布尔、函数、对象（elem、[]、{}、null）、未定义

    真：非0的数字、非空字符串、true、函数、能找到的元素、[]、{}

    假：0、NaN、空字符串''、false、不能找到的元素、null、未定义

\*/

**函数详解**

// return 返回值

// 数字、字符串、布尔、函数、对象（元素\[]\{}\null）、未定义

return：返回值

        1) 函数名+括号：fn1()    ==> return 后面的值；

        2) 所有函数默认返回值：未定义；

        3) return 后面任何代码都不执行了；

**$符的简化版**

$( function(){

    $('#btn1').click(function(){

        $(this).css('background', 'red');

    });

});

**arguments**

fn1( 1,2,3 );                                        // 实参——实际传递的参数

// function fn1( a,b,c ){        // 形参——形式上，abc这些名代表123

function fn1(){

    // arguments => [ 1,2,3 ] —— 实参的集合

    // alert( arguments );

    // alert( arguments.length );

    // alert( arguments[arguments.length-1] );

}

// 当函数的参数个数无法确定的时候：用 arguments

// alert( sum( 1,2,3 ) );                // 6

// alert( sum( 1,2,3,4 ) );            // 10

function sum (){

    var n = 0;

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

        n += arguments[i];

    }

    return n;

}

**获取元素样式**

$('btn1').onclick = function (){

        // alert( $('div1').style.width );

        // $('div1').style.cssText = 'width:350px;';

        // alert( getComputedStyle( $('div1') ).width );            // IE6 7 8 不兼容

        // alert( $('div1').currentStyle.width );                          // 标准浏览器不兼容

        /\*

        if( $('div1').currentStyle ){

            alert( $('div1').currentStyle.width );

        } else {

            alert( getComputedStyle( $('div1'), 250 ).width );

            // FF 4.0 之前

        }

        \*/

        // alert( getStyle( $('div1'), 'width' ) );

        // alert( getStyle( $('div1'), 'height' ) );

        alert( getStyle( $('div1'), 'marginRight' ) );

        /\*

 获取到的是计算机（浏览器）计算后的样式

                background: url() red ……        复合样式（不要获取）

                backgroundColor                           单一样式（不要用来做判断）

                不要有空格

                不要获取未设置后的样式：不兼容

        \*/

**定时器：**

定时器：时间概念

        var timer = setInterval( 函数, 毫秒 );                   重复执行（发动机）

        clearInterval( timer );                                                                 清除

        var timer = setTimeout( 函数, 毫秒 );                执行一次（炸弹）

        clearTimeout( timer );

**日期对象：**

/\*

    系统时间对象 alert( new Date() );        // 当前系统的时间对象  object

\*/

        var myTime = new Date();   //当前系统时间

        // number

        var iYear = myTime.getFullYear();         //年

        var iMonth = myTime.getMonth()+1;    //月   月份要+1

        var iDate = myTime.getDate();             //日

        var iWeek = myTime.getDay();             //星期

        var iHours = myTime.getHours();          //时

        var iMin = myTime.getMinutes();          //分

        var iSec = myTime.getSeconds();         //秒

        var str = '';

        if( iWeek === 0 ) iWeek = '星期日';

        if( iWeek === 1 ) iWeek = '星期一';

        if( iWeek === 2 ) iWeek = '星期二';

        if( iWeek === 3 ) iWeek = '星期三';

        if( iWeek === 4 ) iWeek = '星期四';

        if( iWeek === 5 ) iWeek = '星期五';

        if( iWeek === 6 ) iWeek = '星期六';

str = iYear+ '年' +iMonth+'月'+iDate+'日 '+iWeek+' '+ toTwo(iHours)+' : '+ toTwo(iMin)+' : '+ toTwo(iSec);

        oBody.innerHTML = str;

**倒计时**

<script>

// 现在的时间点（在变）

// 未来的时间点（不变）

var iNow = new Date();

// var iNew = new Date( 2013, 10, 27, 21,56,0 );

var iNew = new Date( 'November 27,2013 22:3:0' );

var t = Math.floor((iNew - iNow)/1000);

// 毫秒 - 秒

var str = Math.floor(t/86400)+'天'+Math.floor(t%86400/3600)+'时'+Math.floor(t%86400%3600/60)+'分'+t%60+'秒';

alert( str );

// 天：Math.floor(t/86400)

// 时：Math.floor(t%86400/3600)

// 分：Math.floor(t%86400%3600/60)

// 秒：t%60

// 数字形式：new Date( 2013,4,1,9,48,12 );

// 字符串形式：new Date('June 10,2013 12:12:12');

// January、February、March、April、May、June、

// July、August、September、October、November、December

</script>

**字符串方法**

var str = '[妙味课堂-WWW.miaov.com](http://xn---www-fz1go2qm5c3w8m.miaov.com/)';

str.charAt(1);            // '味'    获取字符串字符

str.charCodeAt(1);            // 21619

String.fromCharCode(22937, 21619);    // '妙味'

str.indexOf('m', 4);            // 9

str.lastIndexOf('o');            // 16

'1000' < '2'            // true

'1000' > 2            // true

str.substring(0, 4);            // '妙味课堂'

str.slice(-3);            // 'com'

str.toUpperCase();            // '[妙味课堂-WWW.MIAOV.COM](http://xn---www-fz1go2qm5c3w8m.miaov.com/)'

str.toLowerCase();            // '[妙味课堂-www.miaov.com](http://xn---www-fz1go2qm5c3w8m.miaov.com/)'

str.split('.', 2);            // [ '妙味课堂-WWW', 'miaov' ]

var arr = [ 'www', 'miaov', 'com' ];

arr.join('aaa');            // '[www.miaov.com](http://www.miaov.com/)'

数组方法—>JSON

arr.unshift( )   数组添加方法

arr.shift( )       数组删除方法

arr.splice()     删除、替换、添加

Dom

DOM:

   Document  Object  Model(文档对象模型)

DOM结构关系：

   父子关系

   祖先关系

   兄弟关系

父子节点 ： 上下两层节点之间的关系。

祖先节点 ：当前节点上面的所有节点的统称。

子孙节点 ：当前节点下面的所有节点的统称。

一般分为三大关系：

   父级关系：只有一层上下级关系（从当前往上找）

   子级关系：只有以下一层的关系

   兄弟关系: 同一个父级（同一级）

   父子节点 ： 上下两层节点之间的关系。

   祖先节点 ：当前节点上面的所有节点的统称。

   子孙节点 ：当前节点下面的所有节点的统称。

按照节点的类型划分：

   nodeType  返回一个数字

   var str = '';

   元素节点：

      数字1

children:不是标准的属性，但是所有浏览器都支持。

找到某个元素下个所有元素子节点。

parentNode:

   查找某个元素的父节点。

找到某个元素的下个兄弟节点：

   nextElementSibling

找到某个元素的上个兄弟节点：

   previousElementSibling

某个元素下的第一个子节点：

   firstElementChild

某个元素下的最后一个子节点：

   lastElementChild

offsetParent:

   最近的有定位属性的祖先节点

   如果祖先节点都没有定位，那么默认为body

offsetLeft:

   外边框到有定位父级的内边框的距离。

offsetTop

offsetLeft:

   左外边框到有定位父级的左内边框的距离。

offsetTop

   上外边框到有定位父级的上内边框的距离。

getBoundingClientRect(返回值为一个对象)：

   获取某个元素的信息（高版本：left,top,bottom,rigit,width,height）

注意：

   获取的值是会根据滚动条变化的。

//获取元素行间的属性

//elem.getAttribute('key')

//设置元素的行间属性

//elem.setAttribute('key','value')

//删除元素的行间属性

//elem.removeAttribute('key’)

elem.clientWidth/ elem.clientHeight (获取某个元素的宽高)

注意：

   不计算边框

   加上padding

elem.offsetWidth / elem.offsetHeight  (获取某个元素的宽高)

注意：

   计算边框

   加上padding

可视区的宽高：

   document.documentElement.clientWidth/document.documentElement.clientHeight

document.createElement('标签的名字');  //创建元素

插入元素：（向父级末尾添加一个元素）

   parentNode.appendChild(childNode)

插入元素：（向父级中的某个元素前插入元素）

parentNode.insertBefore(新添加的元素,已有的元素)

   特性：

      如果第二个参数为假的，那么会将某个元素添加到父元素的末位。

删除节点:

   父级.removeChild(指定的子节点)

注意：

   如果指定的子节点没有，那么会报错。（进行判断解决）

需求：

   当点击按钮的时候，将红色方块替换成蓝色的方块

    父级.replaceChild(要替换成什么,谁被替换)

需求：

   当点击按钮的时候，将蓝色方块克隆一份出来放到红色方块下

   元素.cloneNode()  //克隆某个元素

注意

   在克隆的时候，默认（false）只克隆指定元素本身，不会克隆该元素下的所有子节点。（cloneNode(有参数，默认为false)），如果里面传入true，那么就可以克隆该元素下的所有子节点。

   事件是不会被克隆的。

注意：

   appendChild/insertBefore/replaceChild在操作一个已有的元素时，  是将已有的元素移动，而不是复制一份进行操作。

table.tHead -->  获取tHead

table.rows --> 获取tr 获取的是行数（一组）

table.tFoot---获取表格底部

table.tBodies---获取表格主体

   注意：

      获取到的是一个集合

rows[n].cells

   单元格  就是td