# 热身1

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';

# 属性操作

img/1.jpg

file:///C:/Users/Administrator/Desktop/11-4-JS1/img/1.jpg

if( oImg.src == 'img/1.jpg' )

不能做为判断条件：

1、所有的相对路径地址

img src

href

2、颜色值：color: red #f00 rgb() rgba()

oDiv.style.float = 'right';

oDiv.style.styleFloat = 'right'; // IE

oDiv.style.cssFloat = 'left'; // 非IE

### cssText: “ float:left ”;

IE（**styleFloat**）

非IE（**cssFloat**）

# for 应用

var oUl = document.getElementById('list'); 静态方法

document.getElementsByTagName('li'); 动态方法

var aLi = oUl.getElementsByTagName('li');

// aLi => [ li, li, li ] 元素的集合

aLi.length 3

Li[0]

// 在用 TagNasme 的时候，必须要加上：[]

document.title = 123;

document.body.innerHTML = 'abc';

oDiv.onclick = function (){

// oDiv.style.width = '200px';

oDiv.style.cssText = ' width:200px; height:200px; ';

# cssText 可以直接像CSS那样写样式

};

oBtn.onclick = function (){

// oDiv.style.width = '100px';

oDiv.style.cssText = '';

};

# 自定义属性

// aBtn[0].abc = 123; // 自定义属性

// alert( aBtn[0].abc );

// aBtn[0].abc = 456;

// JS 可以为任何HTML元素添加任意个自定义属性

# 数据类型、类型转换

ECMAScript：标准、核心

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 返回值是string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) ); 200

var c = '12.34元';

// alert( parseFloat(c) ); 12.34

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

NaN是number类型（数字类型）

isNaN(); // 先用Number() 判断，如果能转成数字就返回false，否则返回true

// 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( [] ) );

str = aInp[0].value;

// HTML 中拿到的内容，类型都是字符串

// alert( typeof str );

# 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！

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

# Arguments

// 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;

}

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

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

var a = 1;

function fn2( a ){

arguments[0] = 3;

alert(a); // 3

var a = 2;

alert( arguments[0] ); // 2

}

fn2(a);

alert(a); // 1

$('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

/\*p

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 );

# 字符串方法

var str = '妙味课堂';

alert( str.length );

alert( str.charAt() );

alert( str.charCodeAt() ); // 22937

alert( str.charCodeAt(1) ); // 21619

alert( str.charCodeAt() ); // 0~9 48~57 a~z 97~122 A~Z 65~90

alert( String.fromCharCode(22937, 21619) );

var str = 'www.miaov.com/2013ww';

alert( str.indexOf('m') );

alert( str.indexOf('m', 5) ); 第二个参数表示从哪个位置开始找

alert( str.indexOf('X') ); // -1 表示没找到

alert( str.indexOf('ww', 2) );

alert( str.lastIndexOf('妙味', 38) ); // 如果第2个值为负数，默认当成0来处理

var str = '妙味课堂是一支独具特色的IT培训团队';

// alert( str.substring(0,2) );

// alert( str.substring(2,0) ); // 可以检测两个数，大的往后扔，小的往前扔

// alert( str.substring(-3, 2) ); // -3 当成0处理

// alert( str.substring(2, -3) );

// alert( str.slice( 2, 0 ) ); // 不交换位置

alert( str.slice( -4, -2 ) ); // 负数从后面倒着往前数~

// alert( str.toUpperCase() ); // 转成大写

// alert( str.toLowerCase() ); // 转成小写

var str = 'www.miaov.com';

// alert( typeof str.split('.') ); // [ 'www', 'miaov', 'com' ]

var arr = str.split( '.' );

// alert( arr[1] );

var str1 = 'leo';

// alert( typeof str1.split() ); // [ 'leo' ]

// alert( str1.split('') ); // [ 'l', 'e', 'o' ]

var str2 = '妙味课堂';

// alert( str2.split('味') );

var str3 = '/www.miaov.com/';

// alert( str3.split('/').length ); // [ , www.miaov.com, ]

var str4 = '2013-11-29-23-07';

alert( str4.split('-', 3) );

var arr = [ 'aa', 'bb', 'cc' ];

// alert( typeof arr.join() ); // 'aa'+','+'bb'+','+'cc'

// alert( arr.join().length );

// alert( arr.join('') );

alert( arr.join('-') );

# json和数组方法

var json = { name : 'leo', age : 32 };

var imgData = {

url : [ 'img/1.png', 'img/2.png', 'img/3.png', 'img/4.png' ],

text : [ '小宠物', '图片二', '图片三', '面具' ]

};

var json2 = { 'name' : 'miaov' }; name 使用字符串模式可以保障安全

// alert( json2.name );

// alert( json2['name'] );

var json5 = {

'url' : [ 'img/1.png', 'img/2.png', 'img/3.png', 'img/4.png' ],

'text' : [ '小宠物', '图片二', '图片三', '面具' ]

};

// var arr = [ {}, {}, {} ];

For in 循环

for ( var attr in json5 ) {

for ( var i=0; i < json5[attr].length; i++ ) {

alert( json5[attr][i] );

}

}

数组

var arr = [ 1,2,3 ];

// alert( arr.push( 'abc' ) ); 在最后添加元素，并返回添加后的长度

// alert( arr );

alert( arr.unshift( 0 ) ); // 在最前添加元素，并返回添加后的长度IE 6 7 不支持 unshift 返回值

// alert( arr );

var arr = [ 'TM', '钟毅', '张森', '杜鹏', 'Leo' ];

// alert( arr.pop() ); 删除最后一个，并返回删除掉的元素

// alert( arr );

alert( arr.shift() ); 删除第一个，并返回删除掉的元素

// arr.shift();

alert( arr );

var arr = [ 'TM', '钟毅', '张森', '杜鹏', 'Leo' ];

// 删除、替换、添加

alert( arr.splice( 0 , 0, 'aaaaaa' ) );

splice（在哪里，删除个数，替换成什么（可以有多个） ） 只返回删除掉的东西，不删除则无返回

// arr.splice( 0, 2, '莫涛 or 钟毅' );

// alert( arr.splice( 1, 0, '钟毅媳妇儿~', '钟毅媳妇们~' ) );

alert( arr );

var arr = [ 'c', 'd', 'a', 'e' ];

// arr.sort();

// alert( arr );

var arr2 = [ 4,3,5,5,76,2,0,8 ];

// arr2.sort();

// alert( arr2 );

arr2.sort(function ( a, b ) {

return a - b;

});

var arr1 = [ 1,2,3 ];

var arr2 = [ 4,5,6 ];

var arr3 = [ 7,8,9 ];

alert( arr1.concat( arr2, arr3 ) ); 字符串连接

var arr1 = [ 1,2,3,4,5,6 ];

arr1.reverse(); 字符串反转

alert( arr1 );

var str = 'abcdef';

// alert(str.split('').reverse().join(''));

# slice()方法与substr和substring的区别

slice() 定义和用法

slice() 方法可从已有的数组中返回选定的元素。

string.slice(start, end)提取一个字符串

string.substring(start, end)提取一个字符串,end不支持负数

string.substr(start, len)提取一个长度为len的字符串

1、slice和substring接收的是起始位置和结束位置(不包括结束位置)，而substr接收的则是起始位置和所要返回的字符串长度。直接看下面例子：

var test = 'hello world';

alert(test.slice(4,7)); //o w

alert(test.substring(4,7)); //o w

alert(test.substr(4,7)); //o world

2、substring是以两个参数中较小一个作为起始位置，较大的参数作为结束位置。如：

alert(test.substring(7,4)); //o w

3、当接收的参数是负数时，slice会将它字符串的长度与对应的负数相加，结果作为参数；substr则仅仅是将第一个参数与字符串长度相加后的结果作为第一个参数；substring则干脆将负参数都直接转换为0。测试代码如下：

var test = 'hello world';

alert(test.slice(-3)); //rld

alert(test.substring(-3)); //hello world

alert(test.substr(-3)); //rld

alert(test.slice(3,-4)); //lo w

alert(test.substring(3,-4)); //hel

alert(test.substr(3,-4)); //空字符串

slice() 方法可从已有的数组中返回选定的元素。

语法

arrayObject.slice(start,end)

参数 描述

start 必需。规定从何处开始选取。如果是负数，那么它规定从数组尾部开始算起的位置。也就是说，-1 指最后一个元素，-2 指倒数第二个元素，以此类推。

end 可选。规定从何处结束选取。该参数是数组片断结束处的数组下标。如果没有指定该参数，那么切分的数组包含从 start 到数组结束的所有元素。如果这个参数是负数，那么它规定的是从数组尾部开始算起的元素。

返回值

返回一个新的数组，包含从 start 到 end （不包括该元素）的 arrayObject 中的元素。

说明

请注意，该方法并不会修改数组，而是返回一个子数组。如果想删除数组中的一段元素，应该使用方法 Array.splice()。

提示和注释

注释：您可使用负值从数组的尾部选取元素。

注释：如果 end 未被规定，那么 slice() 方法会选取从 start 到数组结尾的所有元素。

# DOM

DOM : Document Object Model 文档对象模型

文档：html页面

文档对象：页面中元素

文档对象模型：定义 为了能够让程序(js)去操作页面中的元素

DOM会把文档看作是一棵树，同时定义了很多方法来操作这棵数中的每一个元素（节点）

DOM节点

getElementById

getElementsByTagName

document

document.body

# childNodes

/\*

元素.childNodes : 只读 属性 子节点列表集合

标准下：包含了文本和元素类型的节点，也会包含非法嵌套的子节点

非标准下：只包含元素类型的节点，ie7以下不会包含非法嵌套子节点

childNodes只包含一级子节点，不包含后辈孙级以下的节点

DOM节点的类型有很多种 12种

元素.nodeType : 只读 属性 当前元素的节点类型

元素节点 : 1

属性节点 : 2

文本节点 : 3

\*/

alert( oUl.childNodes.length );

alert( oUl.nodeType );

alert(oUl.childNodes[0].nodeType);

// 属性

元素.attributes : 只读 属性 属性列表集合

alert( oUl.attributes.length );

alert( oUl.attributes[0].nodeType );

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

if ( oUl.childNodes[i].nodeType == 1 ) {

oUl.childNodes[i].style.background = 'red';

}

}

# Children

var oUl = document.getElementById('ul1');

/\*

元素.children : 只读 属性 子节点列表集合

标准下：只包含元素类型的节点

非标准下：只包含元素类型的节点

\*/

//alert( oUl.children.length );

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

oUl.children[i].style.background = 'red';

}

# 兄弟元素

var oUl = document.getElementById('ul1');

/\*

元素.firstChild : 只读 属性 第一个子节点

标准下：firstChild会包含文本类型的节点

非标准下：只包含元素节点

元素.firstElementChild : 只读 属性 标准下获取第一个元素类型的子节点

\*/

//alert( oUl.firstChild );

//alert( oUl.firstElementChild );

/\*if ( oUl.firstElementChild ) {

oUl.firstElementChild.style.background = 'red';

} else {

oUl.firstChild.style.background = 'red';

}\*/

var oFirst = oUl.firstElementChild || oUl.firstChild;

oFirst.style.background = 'red';

/\*

元素.lastChild || 元素.lastElementChild 最后一个子节点

\*/

var oLast = oUl.lastElementChild || oUl.lastChild;

oLast.style.background = 'yellow';

/\*

元素.nextSibling || 元素.nextElementSibling 下一个兄弟节点

\*/

var oNext = oFirst.nextElementSibling || oFirst.nextSibling;

oNext.style.background = 'blue';

/\*

元素.previousSibling || 元素.previousElementSibling 上一个兄弟节点

\*/

var oPrev = oLast.previousElementSibling || oLast.previousSibling;

oPrev.style.background = 'orange';

# 父节点

var aA = document.getElementsByTagName('a');

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

aA[i].onclick = function() {

/\*

元素.parentNode : 只读 属性 当前节点的父级节点

\*/

this.parentNode.style.display = 'none';

}

}

# OffsetParent

var oDiv3 = document.getElementById('div3');

//parentNode : 父节点

//alert( oDiv3.parentNode.id );

/\*

元素.offsetParent : 只读 属性 离当前元素最近的一个有定位属性的父节点

如果没有定位父级，默认是body

ie7以下，如果当前元素没有定位默认是body，如果有定位则是html

ie7以下，如果当前元素的某个父级触发了layout，那么offsetParent就会被指向到这个触发了layout特性的父节点上

\*/

//layout

/\*alert( document.getElementById('div2').currentStyle.hasLayout );\*/

alert( oDiv3.offsetParent.id );

# offsetleft

var oDiv3 = document.getElementById('div3');

/\*

元素.offsetLeft[Top] : 只读 属性 当前元素到定位父级的距离（偏移值）

到当前元素的offsetParent的距离

如果没有定位父级

offsetParent -> body

offsetLeft -> html

如果有定位父级

ie7以下：如果自己没有定位，那么offsetLeft[Top]是到body的距离

如果自己有定位，那么就是到定位父级的距离

其他：到定位父级的距离

\*/

alert( oDiv3.offsetTop );

# 元素宽高

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

/\*

width height

style.width : 样式宽

clientWidth : 可视区宽

offsetWidth : 占位宽

\*/

alert( oDiv.style.width ); //100

alert( oDiv.clientWidth ); //样式宽 + padding 120

alert( oDiv.offsetWidth ); //样式宽 + padding + border 可视区宽 + 边框 122

# 获取元素到页面的距离

function getPos(obj) {

var pos = {left:0, top:0};

while (obj) {

pos.left += obj.offsetLeft;

pos.top += obj.offsetTop;

obj = obj.offsetParent;

}

return pos;

}

# 操作元素属性

var oText = document.getElementById('text1');

/\*

.

[]

\*/

//oText.value = '222';

//alert( oText.value );

//oText['value'] = '222';

//alert( oText['value'] );

//var name = 'value';

//oText.name; 有问题

//oText[name]; OK

/\*

元素.getAttribute(属性名称); 方法 获取指定元素的指定属性的值

\*/

//alert( oText.getAttribute('value') );

/\*

元素.setAttribute(属性名称，属性值); 方法 给指定元素指定的属性设置值

\*/

//oText.setAttribute( 'value', 'hello' );

/\*

元素.removeAttribute(属性名称); 方法 移除指定的元素的指定的属性

\*/

//oText.removeAttribute( 'value' );

/\*

1.用.和[]的形式无法操作元素的自定义属性

getAttribute可以操作元素的自定义属性

\*/

//alert( oText.\_name\_ );

//alert( oText['\_name\_'] );

//alert( oText.getAttribute('\_name\_') );

var oImg = document.getElementById('img1');

/\*

2.可以获取元素属性实际的值

ie7下还是会返回资源的绝对路径

\*/

//alert( oImg.src );

//alert( oImg['src'] );

//alert( oImg.getAttribute('src') );

//alert( oImg.style.getAttribute('width') );

# 元素的创建

var oText = document.getElementById('text1');

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

var oUl = document.getElementById('ul1');

oBtn.onclick = function() {

/\*

document.createElement(标签名称); 创建元素

\*/

var oLi = document.createElement('li');

//oLi.innerHTML = oText.value + '<a href="javascript:;">删除</a>'; 错

oLi.innerHTML = oText.value;

var oA = document.createElement('a');

oA.innerHTML = '删除';

oA.href = 'javascript:;';

oA.onclick = function() {

/\*

父级.removeChild(要删除的元素); 删除元素

\*/

oUl.removeChild( this.parentNode );

}

oLi.appendChild( oA );

//添加到页面中

/\*

父级.appendChild(要添加的元素) 方法 追加子元素

\*/

//oUl.appendChild( oLi );

/\*

父级.insertBefore(新的元素，被插入的元素) 方法 在指定元素前面插入一个新元素

在ie下如果第二个参数的节点不存在，会报错

在其他标准浏览器下如果第二个参数的节点不存在，则会以appendChild的形式进行添加

\*/

//oUl.insertBefore( oLi, oUl.children[0] );

if ( oUl.children[0] ) {

oUl.insertBefore( oLi, oUl.children[0] );

} else {

oUl.appendChild( oLi );

}

}

# replacechild

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

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

var oP = document.getElementById('p1');

oBtn.onclick = function() {

/\*

父级.replaceChild(新节点，被替换节点) 替换子节点

\*/

//document.body.replaceChild( oDiv, oP );

/\*

appendChild,insertBefore,replaceChild都可以操作动态创建出来的节点，也可以操作已有节点

\*/

//oP.appendChild( oDiv );

}

# Bom

### Open() close()

var aInput = document.getElementsByTagName('input');

var opener = null;

aInput[0].onclick = function() {

//open(地址默认是空白页面，打开方式默认新窗口) 打开一个新窗口

//window.open('http://www.baidu.com', '\_self');

opener = window.open();//返回值 返回的新开页面的window对象

//alert(opener == window) false

//opener.document.body.style.background = 'red';

}

aInput[1].onclick = function() {

window.close();

/\*

ff : 无法关闭

chrome : 直接关闭

ie : 询问用户

\*/

}

aInput[2].onclick = function() {

opener.close(); //可以关闭用window.open方法打开的窗口

}

//window.navigator.userAgent : 浏览器信息

//alert( window.navigator.userAgent )

if ( window.navigator.userAgent.indexOf('MSIE') != -1 ) {

alert('我是ie');

} else {

alert('我不是ie');

}

//window.location : 地址

/\*

window.location.href = window.location内容

window.location.search = url?后面的内容

window.location.hash = url#后面的内容

\*/

//alert( window.location );

//alert( window.location.href );//地址

//alert( window.location.search );

//alert( window.location.hash );

# 窗口尺寸

//可视区尺寸

//alert( document.documentElement.clientHeight );

document.onclick = function() {

//滚动距离

//alert( document.documentElement.scrollTop ); //普通浏览器

//alert( document.body.scrollTop ); // chrome

//document.documentElement.scrollTop = 100

//var scrollTop = document.documentElement.scrollTop || document.body.scrollTop;

//alert(scrollTop)

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

//scrollHeight : 内容实际宽高

alert(oDiv.scrollHeight);

}

//offsetHeight

//alert( document.body.offsetHeight );

//ie : 如果内容没有可视区高，那么文档高就是可视区

//alert( document.documentElement.offsetHeight );

alert( document.body.offsetHeight );

//onscroll : 当滚动条滚动的时候触发（每隔一段时间检测）

var i = 0;

window.onscroll = function() {

document.title = i++;

}

//onresize : 当窗口大小发生变化的时候触发（每隔一段时间检测）

window.onresize = function() {

document.title = i++;

}

# Event

焦点 : 使浏览器能够区分用户输入的对象，当一个元素有焦点的时候，那么他就可以接收用户的输入。

我们可以通过一些方式给元素设置焦点

1.点击

2.tab

3.js

不是所有元素都能够接收焦点的.能够响应用户操作的元素才有焦点

//onfocus : 当元素获取到焦点的时候触发

oText.onfocus = function() {

if ( this.value == '请输入内容' ) {

this.value = '';

}

}

//onblur : 当元素失去焦点的时候触发

oText.onblur = function() {

if ( this.value == '' ) {

this.value = '请输入内容';

}

}

/\*

obj.focus() 给指定的元素设置焦点

obj.blur() 取消指定元素的焦点

obj.select() 选择指定元素里面的文本内容

\*/

oText.focus();

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

oBtn.onclick = function() {

oText.select();

}

event : 事件对象 , 当一个事件发生的时候，和当前这个对象发生的这个事件有关的一些详细的信息都会被临时保存到一个指定地方-event对象，供我们在需要的调用。飞机-黑匣子

事件对象必须在一个事件调用的函数里面使用才有内容

事件函数：事件调用的函数，一个函数是不是事件函数，不在定义的决定，而是取决于这个调用的时候

兼容

ie/chrome : event是一个内置全局对象

标准下 : 事件对象是通过事件函数的第一个参数传入

如果一个函数是被事件调用的那么，这个函数定义的第一个参数就是事件对象

# clientX[Y]

当一个事件发生的时候，鼠标到页面可视区的距离

\*/

//alert( event ); //这里没有事件

/\*document.onclick = function() {

alert(event);

};\*/

function fn1(ev) {

//alert( event );

//alert( ev );

var ev = ev || event;

//alert(ev);

/\*for ( var attr in ev ) {

console.log( attr + ' = ' + ev[attr] );

}\*/

alert(ev.clientX);

}

//fn1(); //不是事件调用的函数

document.onclick = fn1; //是事件调用的函数，所以event有内容

# 事件冒泡

/\*

事件冒泡 : 当一个元素接收到事件的时候，会把他接收到的所有传播给他的父级，一直到顶层window.事件冒泡机制

\*/

var oDiv1 = document.getElementById('div1');

var oDiv2 = document.getElementById('div2');

var oDiv3 = document.getElementById('div3');

function fn1() {

alert( this.id );

}

//oDiv1.onclick = fn1; 给\*\*加事件，给元素加事件处理函数

//事件函数绑定

oDiv1.onclick = fn1;//告诉div1，如果他接收到了一个点击事件，那么他就去执行fn1

//oDiv2.onclick = fn1;

oDiv3.onclick = fn1;

//我在马路边捡到一分钱，把他交个警察叔叔

/\*我.on马路边捡到一分钱 = function() {

把他交个警察叔叔

}\*/

# 阻止冒泡

阻止冒泡 : 当前要阻止冒泡的事件函数中调用 event.cancelBubble = true; IE下

event.stopPropagation() 标准下

\*/

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

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

oBtn.onclick = function(ev) {

var ev = ev || event;

ev.cancelBubble = true;//阻止当前对象的当前事件的冒泡

oDiv.style.display = 'block';

}

/\*oBtn.onmouseover = function(ev) {

var ev = ev || event;

ev.cancelBubble = true;

}\*/

document.onclick = function() {

/\*setTimeout(function() {

oDiv.style.display = 'none';

}, 1000);\*/

oDiv.style.display = 'none';

}

# 事件绑定

//给一个对象绑定一个事件处理函数的第一种形式

//obj.onclick = fn;

function fn1() {

alert(this);

}

function fn2() {

alert(2);

}

//document.onclick = fn1;

//document.onclick = fn2; //会覆盖前面绑定fn1

//给一个对象的同一个事件绑定多个不同的函数

//给一个元素绑定事件函数的第二种形式

/\*

ie：obj.attachEvent(事件名称，事件函数);

1.没有捕获

2.事件名称有on

3.事件函数执行的顺序：标准ie-》正序 非标准ie-》倒序

4.this指向window

标准：obj.addEventListener(事件名称，事件函数，是否捕获);

1.有捕获

2.事件名称没有on

3.事件执行的顺序是正序

4.this指向触发该事件的对象

\*/

/\*document.attachEvent('onclick', function() {

fn1.call(document);

});

document.attachEvent('onclick', fn2);\*/

//是否捕获 : 默认是false false:冒泡 true：捕获

/\*document.addEventListener('click', fn1, false);

document.addEventListener('click', fn2, false);\*/

function bind(obj, evname, fn) {

if (obj.addEventListener) {

obj.addEventListener(evname, fn, false);

} else {

obj.attachEvent('on' + evname, function() {

fn.call(obj);

});

}

}

bind(document, 'click', fn1);

bind(document, 'click', fn2);

# call()

//call 函数下的一个方法，call方法第一个参数可以改变函数执行过程中的内部this的指向，call方法第二个参数开始就是原来函数的参数列表

function fn1(a, b) {

alert(this);

alert(a + b);

}

//fn1(); //window

fn1.call(null, 10, 20); //调用函数 fn1() == fn1.call()

# 事件捕获

function fn1() {

alert( this.id );

}

/\*oDiv1.onclick = fn1;

oDiv2.onclick = fn1;

oDiv3.onclick = fn1;\*/

//false = 冒泡

//告诉div1，如果有一个出去的事件触发了你，你就去执行fn1这个函数

（由内到外）

/\*oDiv1.addEventListener('click', fn1, false);

oDiv2.addEventListener('click', fn1, false);

oDiv3.addEventListener('click', fn1, false);\*/

//告诉div1，如果有一个进去的事件触发了你，你就去执行fn1这个函数

（由外到内）

/\*oDiv1.addEventListener('click', fn1, true);

oDiv2.addEventListener('click', fn1, true);

oDiv3.addEventListener('click', fn1, true);\*/

oDiv1.addEventListener('click', function() {

alert(1);

}, false);

oDiv1.addEventListener('click', function() {

alert(3);

}, true);

oDiv3.addEventListener('click', function() {

alert(2);

}, false);

# 事件的取消

/\*

第一种事件绑定形式的取消

\*/

function fn1() {

alert(1);

}

function fn2() {

alert(2);

}

//document.onclick = fn1;

//document.onclick = null; //取消

/\*

ie : obj.detachEvent(事件名称，事件函数);

标准 : obj.removeEventListener(事件名称，事件函数，是否捕获);

\*/

/\*document.attachEvent('onclick', fn1);

document.attachEvent('onclick', fn2);

document.detachEvent('onclick', fn1);\*/

document.addEventListener('click', fn1, false);

document.addEventListener('click', fn1, true);

document.addEventListener('click', fn2, false);

document.removeEventListener('click', fn1, false);

# 键盘事件

/\*

onkeydown : 当键盘按键按下的时候触发

onkeyup : 当键盘按键抬起的时候触发

event.keyCode : 数字类型 键盘按键的值 键值

ctrlKey,shiftKey,altKey 布尔值

当一个事件发生的时候，如果ctrl || shift || alt 是按下的状态，返回true，否则返回false

\*/

document.onkeydown = function(ev) {

//alert(1);

var ev = ev || event;

//alert(ev.keyCode);

}

document.onclick = function(ev) {

var ev = ev || event;

alert(ev.ctrlKey);

}

# 事件默认行为

/\*

事件默认行为：当一个事件发生的时候浏览器自己会默认做的事情

怎么阻止？

当前这个行为是什么事件触发的，然后在这个事件的处理函数中使用return false;

\*/

document.onkeydown = function() {

return false;

}

//oncontextmenu : 右键菜单事件，当右键菜单（环境菜单）显示出来的时候触发

document.oncontextmenu = function() {

//alert(1)

return false;

}

# 自定义右键菜单

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

document.oncontextmenu = function(ev) {

var ev = ev || event;

oDiv.style.display = 'block';

oDiv.style.left = ev.clientX + 'px';

oDiv.style.top = ev.clientY + 'px';

return false;

}

document.onclick = function() {

oDiv.style.display = 'none';

}

# 拖拽

原理：

/\*

onmousedown : 选择元素

onmousemove : 移动元素

onmouseup : 释放元素

\*/

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

oDiv.onmousedown = function(ev) {

var ev = ev || event;

var disX = ev.clientX - this.offsetLeft;

var disY = ev.clientY - this.offsetTop;

document.onmousemove = function(ev) {

var ev = ev || event;

oDiv.style.left = ev.clientX - disX + 'px';

oDiv.style.top = ev.clientY - disY + 'px';

}

document.onmouseup = function() {

document.onmousemove = document.onmouseup = null;

}

}

/\*

1.拖拽的时候，如果有文字被选中，会产生问题

原因：当鼠标按下的时候，如果页面中有文字被选中，那么会触发浏览器默认拖拽文字的效果

解决：

标准：阻止默认行为

非标准ie：全局捕获

\*/

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

oDiv.onmousedown = function(ev) {

var ev = ev || event;

var disX = ev.clientX - this.offsetLeft;

var disY = ev.clientY - this.offsetTop;

if ( oDiv.setCapture ) {

oDiv.setCapture();

}

document.onmousemove = function(ev) {

var ev = ev || event;

oDiv.style.left = ev.clientX - disX + 'px';

oDiv.style.top = ev.clientY - disY + 'px';

}

document.onmouseup = function() {

document.onmousemove = document.onmouseup = null;

//释放全局捕获 releaseCapture();

if ( oDiv.releaseCapture ) {

oDiv.releaseCapture();

}

}

//阻止非IE默认行为

return false;

}

# 全局捕获

var aInput = document.getElementsByTagName('input');

aInput[0].setCapture(); //设置全局捕获 ，当我们给一个元素设置全局捕获以后，那么这个元素就会监听后续发生的所有事件，当有事件发生的时候，就会被当前设置了全局捕获的元素所触发

/\*

ie : 有，并且有效果

ff : 有，但是没效果

chrome : 没有

\*/

aInput[0].onclick = function() {

alert(1);

}

aInput[1].onclick = function() {

alert(2);

}

# setCapture（）

setCapture可以将鼠标事件锁定在指定的元素上，当元素捕获了鼠标事件后，该事件只能作用在当前元素上。

以下情况会导致事件锁定失败：

* 当窗口失去焦点时，锁定的事件，自动就会取消。
* alert也会导致事件的锁定取消。解决办法是在alert之后再次锁定。
* 鼠标右键也会导致事件解锁。

setCapture只可以作用于以下事件：

* onclick
* ondblclick
* onmousedown
* onmouseup
* onmouseover
* onmouseout

 setCapture不可作用于键盘等其它事件，只能作用于鼠标事件。主要用于: onmouseover  与  onmouseout 事件。

\* setCapture该法是IE浏览器专有。

2. setCapture 使用格式

setCapture 有一个布尔值参数，用于设置是否捕获其子元素的鼠标事件。

* 当参数是ture时 ，当前元素会捕获其内的所有子元素的鼠标事件，即指定元素内的子元素不会触发鼠标事件，也就是当前元素内的子元素与当前元素外的元素一致。
* 当参数为false时，当前元素不会捕获该其内的所有子元素的鼠标事件。容器内的对象能够正常地触发事件和取消冒泡。

# 鼠标滚轮

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

/\*

ie/chrome : onmousewheel

event.wheelDelta

上：120

下：-120

firefox : DOMMouseScroll 必须用addEventListener

event.detail

上：-3

下：3

return false阻止的是 obj.on事件名称=fn 所触发的默认行为

addEventListener绑定的事件需要通过event下面的preventDefault();

\*/

oDiv.onmousewheel = fn;

if (oDiv.addEventListener) {

oDiv.addEventListener('DOMMouseScroll', fn, false);

}

//alert(2);

function fn(ev) {

//alert(1);

var ev = ev || event;

//alert( ev.wheelDelta );

//alert(ev.detail)

var b = true;

if (ev.wheelDelta) {

b = ev.wheelDelta > 0 ? true : false;

} else {

b = ev.detail < 0 ? true : false;

}

//alert(b);

if ( b ) {

this.style.height = this.offsetHeight - 10 + 'px';

} else {

this.style.height = this.offsetHeight + 10 + 'px';

}

if (ev.preventDefault) {

ev.preventDefault();

}

return false;

}

/\*document.oncontextmenu = function() {

return false;

}\*/

document.attachEvent('oncontextmenu', function() {

return false;

});

/\*document.addEventListener('contextmenu', function(ev) {

ev.preventDefault();

//return false;

});\*/

# Cookie

/\*

cookie : 存储数据，当用户访问了某个网站（网页）的时候，我们就可以通过cookie来向访问者电脑上存储数据

1.不同的浏览器存放的cookie位置不一样，也是不能通用的

2.cookie的存储是以域名形式进行区分的

3.cookie的数据可以设置名字的

4.一个域名下存放的cookie的个数是有限制的，不同的浏览器存放的个数不一样

5.每个cookie存放的内容大小也是有限制的，不同的浏览器存放大小不一样

我们通过document.cookie来获取当前网站下的cookie的时候，得到的字符串形式的值，他包含了当前网站下所有的cookie。他会把所有的cookie通过一个分号+空格的形式串联起来

如果我们想长时间存放一个cookie。需要在设置这个cookie的时候同时给他设置一个过期的时间

cookie默认是临时存储的，当浏览器关闭进程的时候自动销毁

\*/

//document.cookie = '名字=值';

/\*document.cookie = 'username=leo';

document.cookie = 'age=32';\*/

//document.cookie = '名称=值;expires=' + 字符串格式的时间;

var oDate = new Date();

oDate.setDate( oDate.getDate() + 5 );

//alert(typeof oDate)

//alert(typeof oDate.toGMTString());

//内容最好编码存放,encodeURI

//alert( encodeURI('你好') );

//alert( decodeURI('%E4%BD%A0%E5%A5%BD') )

/\*document.cookie = 'username='+ encodeURI('leo\n你好') +';expires=' + oDate.toGMTString();

document.cookie = 'age=32';\*/

//document.cookie

//alert(decodeURI(document.cookie)); //username=leo; age=32

/\*document.cookie = 'username=leo;expires=' + oDate.toGMTString();

document.cookie = 'age=32';\*/

function setCookie(key, value, t) {

var oDate = new Date();

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

document.cookie = key + '=' + value + ';expires=' + oDate.toGMTString();

}

function getCookie(key) {

var arr1 = document.cookie.split('; ');

for (var i=0; i<arr1.length; i++) {

var arr2 = arr1[i].split('=');

if ( arr2[0] == key ) {

return decodeURI(arr2[1]);

}

}

}

function removeCookie(key) {

setCookie(key, '', -1);

}

//setCookie('sex','男', 10);

//alert( getCookie('age') );

removeCookie('username');

removeCookie('age');

removeCookie('sex');

alert(document.cookie)

# 面向对象

面向对象创建对象

function Person(name, sex)

{

this.name=name;

this.sex=sex;

}

Person.prototype.showName=function ()

{

alert(this.name);

};

Person.prototype.showSex=function ()

{

alert(this.sex);

};

var p=new Person('blue', '男');

p.showName();

p.showSex();

# 定时器和事件会改变this的指向

解决：用 var \_this = this;

function Aaa()

{

var \_this=this;

this.a=12;

setInterval(function (){

\_this.show();

}, 1000);

}

Aaa.prototype.show=function ()

{

alert(this.a);

};

var obj=new Aaa();

//obj.show();

function Bbb()

{

var \_this=this;

this.b=5;

document.getElementById('btn1').onclick=function ()

{

\_this.show();

};

}

Bbb.prototype.show=function ()

{

alert(this.b);

};

window.onload=function ()

{

new Bbb();

};

# Js继承

function Person(name, sex)

{

this.name=name;

this.sex=sex;

}

Person.prototype.showName=function ()

{

alert(this.name);

};

Person.prototype.showSex=function ()

{

alert(this.sex);

};

//-------------------------------------

function Worker(name, sex, job)

{

//this->new出来的Worker对象

//构造函数伪装 调用父级的构造函数——为了继承属性

Person.call(this, name, sex);

this.job=job;

}

//原型链 通过原型来继承父级的方法 不可取

//Worker.prototype=Person.prototype;

for(var i in Person.prototype)

{

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

}

Worker.prototype.showJob=function ()

{

alert(this.job);

};

var oP=new Person('blue', '男');

var oW=new Worker('blue', '男', '打杂的');

oP.showName();

oP.showSex();

oW.showName();

oW.showSex();

oW.showJob();

# 正则表达式

<http://www.w3school.com.cn/jsref/jsref_obj_regexp.asp>











# 表格

var oTab=document.getElementById('tab1');

//alert(oTab.getElementsByTagName('tbody')[0].getElementsByTagName('tr')[2].getElementsByTagName('td')[1].innerHTML);

alert(oTab.tBodies[0].rows[2].cells[1].innerHTML);

tBodies获取tBody rows获取所有行 cells获取所有单元格

# ajax

function ajax(mehtod,url,data,fn){

var xhr = null;

//打开浏览器

/\*

1.创建一个ajax对象

ie6以下new ActiveXObject('Microsoft.XMLHTTP')

\*/

if(window.XMLHttpRequest){

xhr = new XMLHttpRequest();

}else{

xhr = new ActiveXObject("Microsoft.XMLHTTP");

}

//等待服务器返回内容

xhr.onreadystatechange = function(){

if(xhr.readyState == 4){

if(xhr.status == 200){

fn && fn(xhr.responseText);

}else{

alert("出错了："+xhr.status);

}

}

}

if(mehtod.toLowerCase() == "get" && data){

url += "?" + data;

}

//在地址栏输入地址

/\*

open方法

参数

1.打开方式

2.地址

3.是否异步

异步:非阻塞 前面的代码不会影响后面代码的执行

同步:阻塞 前面的代码会影响后面代码的执行

\*/

xhr.open(mehtod,url,true);

//提交 发送请求

if(mehtod.toLowerCase() == "get"){

xhr.send();

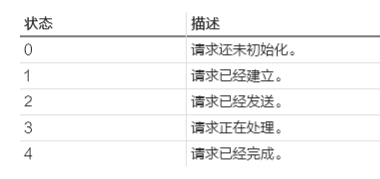
}else{

xhr.setRequestHeader("content-type","application/x-www-form-urlencoded");

xhr.send(data);

}

}



readyState = 0

在 XMLHttpRequest 对象创建之后，但是在我们调用 open() 方法之前。

readyState = 1

在调用 open() 方法之后，但是在调用 send() 之前。

readyState = 2

在我们调用 send() 之后。

readyState = 3

在浏览器建立与服务器的通信之后，但是在服务器完成响应之前。

readyState = 4

在请求完成以及响应数据已经完全从服务器接受之后。

# Ajax跨域解决方案 JSONP

JSONP : JSON with Padding

1.script标签

2.用script标签加载资源是没有跨域问题的

在资源加载进来之前定义好一个函数，这个函数接收一个参数（数据），函数里面利用这个参数做一些事情

然后需要的时候通过script标签加载对应远程文件资源，当远程的文件资源被加载进来的时候，就会去执行我们前面定义好的函数，并且把数据当作这个函数的参数传入进去

<script>

function fn(data) {

alert(data);

}

</script>

<script>

window.onload = function() {

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

oBtn.onclick = function() {

//当按钮点击的时候再去加载远程资源，让他执行

var oScript = document.createElement('script');

oScript.src = '2.txt';

//2.txt里的内容：fn([1,2,3]);

document.body.appendChild(oScript);

}

}

</script>

<!DOCTYPE HTML>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>无标题文档</title>

<script>

function fn1(data) {

var oUl1 = document.getElementById('ul1');

var html = '';

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

html += '<li>'+data[i]+'</li>';

}

oUl1.innerHTML = html;

}

function fn2(data) {

var oUl2 = document.getElementById('ul2');

var html = '';

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

html += '<li>'+data[i]+'</li>';

}

oUl2.innerHTML = html;

}

function fn3(data) {

var oUl3 = document.getElementById('ul3');

var html = '';

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

html += '<li>'+data[i]+'</li>';

}

oUl3.innerHTML = html;

}

</script>

<script>

window.onload = function() {

var oBtn1 = document.getElementById('btn1');

var oBtn2 = document.getElementById('btn2');

oBtn1.onclick = function() {

var oScript = document.createElement('script');

oScript.src = 'getData.php?callback=fn1';

document.body.appendChild(oScript);

}

var oBtn2 = document.getElementById('btn2');

oBtn2.onclick = function() {

var oScript = document.createElement('script');

oScript.src = 'getData.php?t=str&callback=fn2';

document.body.appendChild(oScript);

}

var oBtn3 = document.getElementById('btn3');

oBtn3.onclick = function() {

var oScript = document.createElement('script');

oScript.src = 'getData.php?callback=fn3';

document.body.appendChild(oScript);

}

}

</script>

</head>

<body>

<input type="button" id="btn1" value="加载数字" />

<ul id="ul1"></ul>

<input type="button" id="btn2" value="加载字母" />

<ul id="ul2"></ul>

<input type="button" id="btn3" value="加载字母" />

<ul id="ul3"></ul>

</body>

</html>

getData.php

<?php

$t = isset($\_GET['t']) ? $\_GET['t'] : 'num';

$callback = isset($\_GET['callback']) ? $\_GET['callback'] : 'fn1';

$arr1 = array('111111','22222222','33333333','4444444','555555555555555555555');

$arr2 = array('aaaaaaaaaaaa','bbbbbbbb','cccccccccccc','ddddddddd','eeeeeeeeeeee');

if ($t == 'num') {

$data = json\_encode($arr1);

} else {

$data = json\_encode($arr2);

}

echo $callback.'('.$data.');';

?>

# Date

var d1 = new Date();

console.log(d1);

var d2 = new Date(Date.UTC(2018,3,4,22,22,22));

console.log(d2);

//获取的是毫秒值

var d3 = Date.now();

console.log(d3);

var d4 = +new Date();

console.log(d4);

console.log(d1.toLocaleString());

console.log(d4.toLocaleString());

console.log(d1.toUTCString());

console.log(d1.toDateString());

console.log(d1.toTimeString());

console.log(d1.toLocaleDateString());

console.log(d1.toLocaleTimeString());

console.log("年" + d1.getFullYear() + " 月" + (d1.getMonth() + 1) + " 日" + d1.getDate() + " 时" + d1.getHours() + " 分" + d1.getMinutes() + " 秒" + d1.getSeconds());

# 数组

var arr = [11,20,5,3,8,7,6];

//默认排序时会把元素先转换成字符串，再比较

arr.sort();

console.log(arr);

arr.sort(function(a,b){

return a - b;

})

console.log(arr);

arr.reverse();

// var arr2 = ["a","b","c"];

// arr = arr.concat(arr2);

// arr = arr.slice(2,6);

// arr.splice(1,3,"d","e","f");

// console.log(arr);

// 如果每个都符合，则返回true

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

return item > 5;

});

// 如果有一个符合，则返回true

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

return item > 5;

});

console.log(istrue);

var arr3 = arr.filter(function(item,index,array){

return item > 5;

});

console.log(arr3);

arr3.forEach(function(item,index,array){

console.log(index + " : " + item + "\n");

});

var arr4 = arr3.map(function(item,index,array){

return item + 8;

});

console.log(arr4);

var arr5 = [33,44,55,66,78];

var sum = arr5.reduce(function(pre,cur,index,array){

return pre + cur;

},10);

console.log(sum);

# function

function factorial(num){

if(num > 1){

return num \* arguments.callee(num - 1);

}else{

return 1;

}

}

// alert(factorial(15));

function sayColor(){

alert(this.color);

}

var color = "red";

// sayColor();

var o = {"color" : "blue"};

o.sayColor = sayColor;

// o.sayColor();

function outer(){

inner();

}

function inner(){

alert(arguments.callee.caller);

}

// outer();

function sum(num1,num2){

return sum1 + sum2;

}

//函数的形参的个数

// alert(sayColor.length);

// alert(sum.length);

// sayColor.call(o);

// sayColor.apply(o);

//返回一个函数的拷贝，该函数的this指向bind的第一个参数，第一个参数后可以接多个参数 args1 args2...

var sayColor2 = sayColor.bind(o);

sayColor2();

# string

var str = "bat";

var reg = /.at/g;

reg.test(str);

// alert(RegExp.lastMatch);

// $` 匹配匹配到的字符串的左侧字符串

// alert("(" + str.replace(/at/g,"word $`") + ")");

// $' 匹配匹配到的字符串的右侧字符串

// alert("(" + str.replace(/at/g,"word $'") + ")");

var reg2 = /(at)/g;

var str2 = "bat,cat,dat,fat";

alert(str2.replace(reg2,"(word $1)"));

# number

var n1 = 888.124565;

//保留几位小数

alert(n1.toFixed(2));

//保留几位小数 以e的几次方表示 即10的几次方

alert(n1.toExponential(3));

//自动选择转换成什么

alert(n1.toPrecision(2));

# 面向对象

// 创建对象方式：

// 1 工厂模式

function CreatPerson(name,age,job){

var o = new Object();

o.name = name;

o.age = age;

o.job = job;

o.sayName = function(){

alert(this.name);

}

return o;

}

var p1 = CreatPerson("tom",20,"doctor");

// 2 构造函数模式

function Person2(name,age,job){

this.name = name;

this.age = age;

this.job = job;

this.sayName = function(){

alert(this.name);

}

}

var p2 = new Person2("tom",20,"doctor");

// 3 原型模式

function Person3(){};

Person3.prototype.name = "tom";

Person3.prototype.age = 18;

Person3.prototype.job = "doctor";

Person3.prototype.sayName = function(){

alert(this.name);

}

var p3 = new Person3();

// 4 混合2 和 3

function Person4(name,age,job){

this.name = name;

this.age = age;

this.job = job;

}

Person4.prototype.sayName = function(){

alert(this.name);

}

var p4 = new Person4("tom",20,"doctor");

// 5 动态原型模式

function Person5(name,age,job){

this.name = name;

this.age = age;

this.job = job;

if(typeof this.sayName != "function"){

Person5.prototype.sayName = function(){

alert(this.name);

}

}

}

var p5 = new Person5("tom",20,"doctor");

//6 寄生构造函数模式

function Person6(name,age,job){

var o = new Object();

o.name = name;

o.age = age;

o.job = job;

o.sayName = function(){

alert(this.name);

}

return o;

}

var p6 = new Person6("tom",20,"doctor");

// 稳妥构造函数模式： 没有公共属性，其方法也不引用this

function Person7(name,age,job){

var o = new Object();

var name = name;

var age = age;

var job = job;

o.sayName = function(){

alert(name);

}

return o;

}

var p7 = Person7("tom",20,"doctor");

# 面向对象—原型对象

function Person(){

}

Person.prototype.name = "tom";

Person.prototype.age = 20;

Person.prototype.sayName = function(){

alert(this.name);

}

var person1 = new Person();

var person2 = new Person();

// 检测某个原型对象是否是某个实例的原型对象

// alert(Person.prototype.isPrototypeOf(person1));

// alert(Person.prototype.isPrototypeOf(person2));

//获取实例的原型对象

// alert(Object.getPrototypeOf(person1).name);

// alert(Object.getPrototypeOf(person2) == Person.prototype);

// 检测某个实例是否拥有自己的属性（不是从原型对象那里得到的）

// alert(person1.hasOwnProperty("name"));

person2.name = "mike";

// alert(person2.hasOwnProperty("name"));

// in : 通过某个对象能够访问某个属性时返回true

// alert("name" in person2);

// 取得对象上所有可枚举的属性

var pros = Object.keys(Person.prototype);

// alert(pros);

var pros2 = Object.keys(person2);

// alert(pros2);

// 获取对象上的所有属性 包括不可枚举的

var pros3 = Object.getOwnPropertyNames(Person.prototype);

alert(pros3);

var pros4 = Object.getOwnPropertyNames(person2);

alert(pros4);

# 面向对象--数据、访问器属性

var person = new Object();

// 数据属性

Object.defineProperty(person,"name",{

configurable : true, // 是否可以配置（删除，修改特性等）

enumerable : true, //是否可以遍历

writable : true, // 是否可以修改值

value : "tom" // 属性的值

});

// alert(person.name);

// person.name = "mike";

// alert(person.name);

// 访问器属性

var person2 = {

\_name\_ : "mike",

age : 18

};

Object.defineProperty(person2,"name",{

contigurable : true,

enumerable : true,

get : function(){

return this.\_name\_;

},

set : function(newValue){

this.\_name\_ = newValue;

this.age += 1;

}

});

// alert(person2.\_name\_);

// alert(person2.name);

// person2.name = "amy";

// alert(person2.name);

// alert(person2.age);

//定义多个属性

var person3 = new Object();

Object.defineProperties(person3,{

\_year :{

value : 2018

},

edition : {

value : 5

},

year : {

get : function(){

return this.\_year;

},

set : function(newValue){

if(newValue > 2018){

this.\_year = newValue;

this.edition += newValue - this.edition;

}

}

}

});

// 读取属性的特性

var descriptor = Object.getOwnPropertyDescriptor(person3,"\_year");

alert(descriptor.configurable);

alert(descriptor.enumerable);

var descriptor2 = Object.getOwnPropertyDescriptor(person3,"year");

alert(descriptor2.set);

# 面向对象--继承

// 1 原型链

function superType(){

this.property = true;

}

superType.prototype.getSuperValue = function(){

return this.property;

}

function subType(){

this.subProperty = false;

}

subType.prototype = new superType();

subType.prototype.getSubValue = function(){

return this.subProperty;

}

var ins1 = new subType();

// alert(ins1.getSubValue());

// alert(ins1.getSuperValue());

// 2 借用构造函数

function superType2(name){

this.name = name;

this.colors = ["red","green","blue"];

}

function subType2(){

superType2.call(this);

this.age = 20;

}

var ins2 = new subType2();

// alert(ins2.colors);

ins2.colors.push("pink");

// alert(ins2.colors);

var ins22 = new subType2();

// alert(ins22.colors);

// 3 组合继承

function superType3(name){

this.name = name;

this.colors = ["red","green","blue"];

}

superType3.prototype.sayName = function(){

alert(this.name);

}

function subType3(name,age){

//继承属性

superType3.call(this,name);

this.age = age;

}

//继承方法

subType3.prototype = new superType3();

subType3.prototype.sayAge = function(){

alert(this.age);

}

// 4 原形式继承

function object(o){

function F(){};

F.prototype = o;

return new F();

}

var person = {

"name" : "Tom",

"friends" : ["mike","Amy","Jack"]

}

var anotherPerson = object(person);

anotherPerson.name = "greg";

anotherPerson.friends.push("rob");

var anotherPerson2 = object(person);

anotherPerson2.name = "linda";

anotherPerson2.friends.push("barbie");

// alert(person.friends);

// alert(anotherPerson.friends);

// 5 寄生式继承

function CreateAnther(original){

var clone = object(original);

clone.sayHi = function(){

alert("hi");

}

return clone;

}

// 6 寄生组合式继承 解决组合继承每次都会调用两次超类型构造函数的问题

function inheritPrototype(subType,superType){

var prototype = object(superType.prototype);

prototype.constructor = subType;

subType.prototype = prototype;

}

function superType4(name){

this.name = name;

this.colors = ["red","green","blue"];

}

superType4.prototype.sayName = function(){

alert(this.name);

}

function subType4(name,age){

superType4.call(this,name);

this.age = age;

}

inheritPrototype(subType4,superType4);

# history

//往后退一页

history.go(-1);

//前进一页

history.go(1);

//跳转到包含该字符串的第一个页面的位置

history.go("baidu.com");

//后退一页

history.back();

//前进一页

history.forward();

if(history.length == 0){

alert("这是你打开窗口后的第一个页面");

}

# Location

/\*

location所有属性：

属性名 例子 说明

hash "#contents" 返回URL中的hash（#号后跟零或多个字符），如果URL中不包含散列，则返回空字符串

host "www.wrox.com:80" 返回服务器名称和端口号（如果有）

hostname "www.wrox.com" 返回不带端口号的服务器名称

href "http://www.wrox.com" 返回当前加载页面的完整URL。toString()也返回这个值

pathname "/wileyCDA" 返回URL中的目录和（或）文件名

port "8080" 返回URL中指定的端口号。如果没有则返回空字符串

protocol "http:" 返回页面使用的协议。通常是http: 或https:

search "?q=javascript" 返回URL的查询字符串。这个字符串以问号开头

\*/

// 解析查询字符串

function searchString(){

var str = (location.search.length > 0 ? location.search.substring(1) : "");

var args = {};

var name = "";

var value = "";

var item = null;

var items = (str.length ? str.split("&") : []);

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

item = items[i].split("=");

name = decodeURIComponent(item[0]);

value = decodeURIComponent(item[1]);

if(name.length){

args[name] = value;

}

}

return args;

}

console.log(searchString());

//位置操作

// 以下三个方式结果都一样

// location.assign("http://www.baidu.com");

// location.href = "http://www.baidu.com";

// window.location = "http://www.baidu.com";

//使用location 下的方法改变URL都会在历史记录中改变，除非使用replace()

location.replace("http://www.google.com");

//重新加载页面 reload()

location.reload(); //如果自上次以来没有改变，则从缓存中加载

location.reload(true); //不论什么情况都强制从服务器中重新加载

# window

// 获取浏览器窗口相对于屏幕左上角的距离 不同浏览器的值不相同

var leftPos = (typeof window.screenLeft == "number") ? window.screenLeft : window.screenX;

var topPos = (typeof window.screenTop == "number") ? window.screenTop : window.screenY;

// alert(leftPos + " : " + topPos);

// 把窗口移动到那个位置上 chrome firefox被禁用 ie可以 调整窗口大小resizeTo() chrome firefox被禁用 ie可以

// window.moveTo(200,200);

// 移动窗口，相对于当前位置 chrome firefox被禁用 ie可以 调整窗口大小resizeBy() chrome firefox被禁用 ie可以

// window.moveBy(0,100);

//获取页面视口信息

var pageWidth = window.innerWidth;

var pageHeight = window.innerHeight;

if(typeof pageWidth != "number"){

if(document.compatMode == "CSS1Compat"){

pageWidth = document.documentElement.clientWidth;

pageHeight = document.documentElement.clientHeight;

}

else{

pageWidth = document.body.clientWidth;

pageHeight = document.body.clientHeight;

}

}

// alert(pageWidth + " : " + pageHeight);

// 导航和打开窗口

// window.open() 4个参数：要加载的URL、 窗口目标、 一个特性字符串、 一个表示新页面是否取代浏览器历史记录中当前加载页面的布尔值 通常只需传第一个参数

// window.open("http://www.baidu.com");

// window.open("http://www.baidu.com","\_blank");

// window.open("http://www.baidu.com","\_self");

// window.open("http://www.baidu.com","newWindow");

// window.open("http://www.baidu.com","\_blank","width=400,height=400,top=200,left=200,resizable=no");

//可以对通过window.open() 创建的窗口控制

// var newWindow = window.open("数组.html","\_blank","width=400,height=400,top=200,left=200,resizable=no");

// newWindow.resizeTo(600,600);

// newWindow.moveTo(400,400);

// newWindow.close();

//opener属性指向打开这个窗口的原始窗口

// alert(newWindow.opener == window);

//系统提示框 alert() confirm() prompt()

// alert(1);

// if(confirm("Are you sure?")){

// alert("you are sure");

// }else{

// alert("you are not sure");

// }

// var result = prompt("what's your name?","Mike");

// if(result){

// alert("Hello," + result);

// }

// 显示打印对话框

// window.print();

// 显示查找对话框

window.find();

# DOM1

//节点

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

// alert(box.nodeType);

if(box.nodeType == 1){

//如果是元素节点，则nodeName属性返回的是标签名，nodeValue值始终是null

// alert(box.nodeName);

}

// childNodes 返回一个nodeList 对象，是一个类数组，并且是动态的，随节点改变而改变 包含元素节点和非元素节点

var nodeList = box.childNodes;

// children 返回一个nodeList 对象，是一个类数组，并且是动态的，随节点改变而改变 只包含元素节点,

var childrenlist = box.children;

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

// 以下两种方式都可以获得 nodeList 的元素

// console.log(nodeList[i]);

console.log(nodeList.item(i).nodeType);

}

//将nodeList 转换成数组

function convertToArray(list){

var arr = null;

try{

arr = Array.prototype.slice.call(list,0); // IE8及以前会报错

}catch(e){

arr = new Array();

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

arr.push(list[i]);

}

}

return arr;

}

// 同胞节点

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

// if(nodeList[i].nextSibling === null ){

// alert("这是最后一个节点");

// }else if(nodeList[i].previousSibling === null){

// alert("这是第一个节点");

// }

// }

//nextElementSibling previousElementSibling

// firstElementChild lastElementChild

console.log(nodeList.firstChild === nodeList[0]);

console.log(nodeList.lastChild === nodeList[nodeList.length-1]);

// alert( box.hasChildNodes() );

//ownerDocument 表示这个节点所在的文档的文档节点

// alert(box.ownerDocument == document);

// 操作节点

// 添加节点到父元素子节点列表末尾，但不会允许同一个节点在列表中出现两次，所以添加原有的节点会将该节点剪切到列表末尾

// box.appendChild(box.firstElementChild);

// 同样执行剪切工作

// box.insertBefore(box.firstElementChild,box.lastElementChild);

// 同样执行剪切工作 并把被替换的节点删除

// box.replaceChild(box.firstElementChild,box.lastElementChild)

//删除节点

// box.removeChild(box.lastElementChild);

//其他方法

// cloneNode() 不会复制用JavaScript定义的属性

// 深度复制，复制本身及其整个子节点树

var deepClone = box.cloneNode(true);

// 浅度复制，只复制节点本身

var shallowClone = box.cloneNode(false);

// 由于解析器的实现或DOM操作等原因，可能会出现文本节点不包含文本，或者接连出现两个文本节点的情况，当在某个节点上调用这个方法是，就会在该节点的后代节点中查找这两种情况，如果找到了空文本节点，则删除他，如果找到相邻的文本节点，则将它们合并为一个文本节点

box.normalize();

// document document表示整个HTML页面

//document.ducumentElement 指向网页的<html>元素 获取html 元素的方法有两种

var html = document.documentElement;

var html2 = document.childNodes[1];

var html3 = document.firstElementChild;

// alert(html === html2);

// alert(html2 === html3);

// document.body 指向 <body> 元素

// document.title 获得<title>中的文本 可以获得或修改网页的标题，但不会改变<title> 标签

var initTitle = document.title;

document.title = "dom1";

//获取页面的URL

var url = document.URL;

//获取页面的域名 如URL为 http://www.baidu.com/2/124/index 时，域名为 www.baidu.com

var domain = document.domain;

// 获取链接到当前页面的那个页面的URL

var referrer = document.referrer;

var imgs = document.getElementsByTagName("img");

//imgs 是一个HTMLCollection 对象，可以通过索引值取得元素，也可以通过元素的name特性取得元素

//如某个img的name特性为 imgname 则可以这样获取

var img1 = imgs.namedItem("imgname");

//或

var img2 = imgs["imgname"];

//可以通过name特性（attribute）获取元素

var radioBoxs = document.getElementsByName("myRadio");

// 获取HTML中所有带name特性的 <a>

document.anchors;

//获取HTML中所有带有href特性的 <a>

document.links;

//Elememt

//HTML

box.id; //获取或修改元素id

box.className; //获取或修改元素的class

box.title; // 获取或修改元素的title

// 操作特性 可以操作原有的特性和自定义特性

box.getAttrbute("class");

box.setAttrbute("class","myClass");

box.removeAttribute("id");

// 创建节点

var newNode = document.createElement("div");

# DOM扩展

document.body.innerHTML += "<div class='div'>123</div>";

//outerHTML 在读模式，会获取调用这个属性的元素及其所有子节点的HTML便签，在写模式，会用根据指定的字符串创建DOM树，并用他替换自己

console.log(document.body.outerHTML);

// document.body.outerHTML = "<body><div class='div'>456</div></body>";

// insertAdjacentHTML() 接收两个参数 第一个只能是以下四个参数之一 ： beforebegin afterbegin beforeend afterend 第二个参数为要插入的HTML字符串

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

obox.insertAdjacentHTML("beforebegin","<p>beforebegin</p>"); // 在obox之前插入

obox.insertAdjacentHTML("afterbegin","<p>afterbegin</p>"); // 在obox第一个子元素之前插入

obox.insertAdjacentHTML("beforeend","<p>beforeend</p>"); // 在obox最后一个子元素之后插入

obox.insertAdjacentHTML("afterend","<p>afterend</p>"); // 在obox之后插入

// 通过类名获取元素 返回的是nodeList对象

document.getElementsByClassName("box active");

// classList() 方法

var odiv = document.getElementsByClassName("div")[0];

// div.classList.add("box"); // 添加 box 类

// div.classList.contains("active"); // 检测是否拥有 active 类

// div.classList.remove("box"); // 删除 box 类

// div.classList.toggle("active"); // 如果有 active 类 ，则删除，没有则添加

// 焦点管理

var acitveE = document.activeELement; // 获得当前获得了焦点的元素

console.log(document.hasFocus()); // 表示当前文档是否获得了焦点

//

console.log( document.readyState ); // loading 正在加载文档， complete 已经加载完文档

// 兼容模式

console.log( document.compatMode ); // CSS1Compat 标准模式 BackCompat 混杂模式

// 自定义属性 HTML5规定 自定义属性的格式为 data-name 以这个格式定义的自定义属性可以通过dataset获取

// 获取自定义属性 获取和设置时不用加前缀 data-

var data\_myName = odiv.dataset.myName;

// 设置

odiv.dataset.newname = "Mike"; // 以驼峰形式命名 如 newName 则标签中会以 data-new-name形式保存

// 检测一个节点是不是另一个节点的子节点

console.log(document.documentElement.contains( document.body ));

//innerText 只包含节点内的文本，不包含节点元素，节点元素会被忽略掉，只保留文本，用它来设置时，设置的字符串会全部转为文本，不会被解析为节点,outerText 的作用与outerHTML相似

console.log(document.body.innerText);

obox.innerText = "<p>innerText</p>";

console.log(document.body.textContent);

# 事件

/\*

event属性

属性/方法 说明

bubbles 表明事件是否冒泡

cancelable 表明是否可以取消事件的默认行为

currentTarget 其事件处理程序当前正在处理事件的那个元素

defaultPrevented 表示是否调用了preventDefault()

detail 与事件相关的细节信息

eventPhase 调用事件处理程序的阶段：1表示捕获阶段，2表示“处于目标，3表示冒泡阶段

preventDefault() 取消事件的默认行为，如果cancelable是true，则可以使用 IE <= 8中为 return false; 或 returnValue = false;

stopImmediatePropagation() 取消事件的进一步捕获或冒泡，同时阻止任何事件处理程序被调用

stopPropagation() 取消事件的进一步捕获或冒泡，如果bubbles为true，则可以使用这个方法 IE中为 ev.cancelBubble = true

target 事件的目标 IE 中为ev.srcElement

trusted 为true表示事件是浏览器生成的。为false表示事件是由开发人员通过JavaScript创建的

type 被触发的事件的类型

view 与事件关联的抽象视图。等同于发生事件的window对象

\*/

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

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

// this 始终等于 currentTarget,他们都等于绑定事件的那个元素, 而target表示事件的实际目标，即通触发事件的那个目标

// btn.onclick = function(ev){

// var ev = ev || window.event;

// alert(ev.currentTarget == this);

// alert(ev.target == this);

// alert(ev.currentTarget == ev.target);

// }

// document.body.onclick = function(ev){

// var ev = ev || window.event;

// alert(ev.currentTarget == document.body);

// alert(this == document.body);

// alert(ev.target == btn);

// }

alink.onclick = function(ev){

var ev = ev || window.event;

// ev.preventDefault();

//ev.returnValue = false;

// return false;

}

var eventUtil = {

addHandler : function(element,type,handler){

if(element.addEventListener){

element.addEventListener(type,handler,false);

}else if(element.attachEvent){

element.attachEvent("on" + type,handler);

}else{

element["on"+type] = handler;

}

},

removeHandler : function(element,type,handler){

if(element.removeEventListener){

element.removeEventListener(type,handler,false);

}else if(element.detachEvent){

element.detachEvent("on" + type,handler);

}else{

element["on"+type] = null;

}

},

getEvent : function(ev){

return ev || window.event;

},

getTarget : function(ev){

return ev.target || ev.srcElement;

},

preventDefault : function(ev){

if(ev.preventDefault){

ev.preventDefault();

}else if(typeof ev.returnValue != "undefined"){

ev.returnValue = false;

}else{

return false;

}

},

stopPropagation : function(ev){

if(ev.stopPropagation){

ev.stopPropagation();

}else{

ev.cancelBubble = true;

}

},

// 在 onmouseout 和 onmouseover 中有代表相关元素的属性，IE9+ 为relatedTarget IE8及以下，onmouseout为toElement onmouseover 为fromElement

getRelatedTarget : function(ev){

if(ev.relatedTarget){

return ev.relatedTarget;

}else if(ev.toElement){

return ev.toElement;

}else if(ev.fromElement){

return ev.fromElement;

}

}

};

/\*

事件类型

1 UI类型

load 当页面完全加载后再window上触发，在图像加载完毕后在<img>元素上触发，当所有框架加载完毕后再框架集上触发，当嵌入的内容加载完毕是在<object>元素上触发

unload 当页面完全卸载后再window上触发（在页面跳转时），其他类似于load，

abort 当用户停止下载过程时，如果嵌入的内容没有加载完，则在<object>元素上触发

error 当JavaScript错误时在window上触发，在图像无法加载在<img>元素上触发，当不是所有框架都可以时在框架集上触发，当嵌入的内容无法加载时在<object>元素上触发

select 当用户选择文本框（<input> <textarea>）中的一个或多个字符时触发

resize 当窗口或框架的大小变化时在window或框架集上触发

scroll 当用户滚动带滚动条的元素中的内容时，在该元素上触发

2 焦点事件

blur 在元素失去焦点时触发，这个事件不冒泡

focus 在元素获得焦点时触发，这个事件不冒泡

focusin 在元素获得焦点时触发，这个事件冒泡

focusout 在元素失去焦点时触发，这个事件冒泡

3 鼠标与滚轮事件

click 在用户点击主鼠标按钮或按下回车键时触发

dbclick 双击鼠标时触发

mousedown 在用户按下了任意鼠标按钮时触发

mouseenter 在鼠标光标从元素外部首次移动到元素范围之内时触发。不冒泡。而且在光标移动到后代元素上不会触发

mouseleave 在位于元素上方的鼠标光标移动到元素范围之外时触发。不冒泡。而且在光标移动到后代元素上不会触发

mousemove 当鼠标指针在元素内部移动时重复触发

mouseout 在鼠标位于一个元素上方，然后用户将其移入另一个元素时触发，移入的另一个元素可能位于前一个元素的外部，也可能是这个元素的子元素。

mouseover 在鼠标位于一个元素外部，然后用户将其首次移入另一个元素边界之内时触发

除了mouseenter 和mouseleave，所有鼠标事件都会冒泡，也可以取消

ev.button 0 表示鼠标主键被按下 1表示鼠标中键 2表示鼠标次键被按下 IE8及以下又更复杂的

鼠标滚轮 onmousewheel

ev.wheelDelta 向上滚一次为120 向下滚动一次为-120

火狐： DOMMouseScroll事件

ev.detail 向上滚动一次为-3 向下滚动一次为3

4 键盘和文本事件

keydown 当用户按下键盘上的任意按键时触发，如果按住不放则重复触发

keypress 当用户按下键盘上的字符键时触发， 如果按住不放则重复触发 输入法在英文状态下按下字母键就会触发，中文状态下不触发 按下数字键都会触发

keyup 当用户释放键盘上的按键时触发

5 变动事件

DOMSubtreeModified 在DOM结构中发生任何变化时触发。这个事件在其他任何事件触发后都会触发

DOMNodeInserted 在一个节点作为子节点被插入到另一个节点中时触发

DOMNodeRemoved 在一个节点从其父节点中被移除时触发

6 HTML5事件

contextmenu 右键菜单

beforeunload 在卸载（关闭）页面前触发 E6，IE7，IE8 中 刷新页面、关闭浏览器之后、页面跳转之后都会执行；IE9 刷新页面 会执行，页面跳转、关闭浏览器不能执行；firefox(包括firefox3.6) 关闭标签之后、页面跳转之后、刷新页面之后能执行，但关闭浏览器不能执行；Safari 刷新页面、页面跳转之后会执行，但关闭浏览器不能执行；Opera、Chrome 任何情况都不执行

\*/

// window.onbeforeunload = function (ev){

// var ev = ev || window.event;

// var message = "are you sure?";

// ev.returnValue = message;

// return message;

// }

/\*

DOMContentLoaded 文档被加载和解析，并且DOM被完全构造，但链接的资源（例如图像，样式表和子框架（subframes））可能尚未被加载。

hashchange URL的参数列表（以及#后面的所有字符串）改变时触发 必须绑定在window 有ev.oldURL 和 ev.newURL 分别保存变化前后的完整URL IE没有这两个属性，只能通过 location来访问

\*/

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

textinput.onkeypress = function(){

console.log(new Date().toLocaleString());

var ospan = document.createElement("sapn");

ospan.innerHTML = "LLL";

document.body.appendChild(ospan);

}

document.onDOMSubtreeModified = function(){

console.log(1);

}

// alert(document.onDOMNodeInserted);

// alert("oninput" in textinput);

# 表单

// 可以通过 document.forms 获取页面所有表单，然后通过索引或者 name 值来取得特定的表单

var allForm = document.forms;

var firstForm = document[0];

var form1 = allForm["form1"];

//可以通过 elements 属性获取表单中所有元素，然后通过索引或者 name 值来取得特定的元素 如果有多个元素的name属性相同，则返回一个包含所有这些元素的nodelist，nodelist可以通过索引访问里面的值

var allElememts = form1.elements;

var minput = allElememts["minput"];

var secondE = allElememts[1];

/\*

共有的表单字段属性

disabled 布尔值，表示当前表单是否被禁用

form 指向当前字段所属表单的指针，只读

name 当前字段的名称

readOnly 表示当前字段是否只读

tabIndex 表示当前字段的切换序号

type 当前字段的类型，如“checkbox”，“radio”。

value 当前字段将被提交给服务器的值

共有的表单字段方法

focus() 使获取焦点，也可以用HTML5 的autofoucus

blur() 使失去焦点

共有的表单字段事件

blur 当字段失去焦点时触发

change 对于<input> 和<textarea> ,当他们失去焦点且value值发生改变时触发，对于<select> 当其选项改变时触发

focus 当字段获得焦点时触发

\*/

//文本框脚本

minput.select(); // 选择文本框中的文本

minput.onselect = function(){} // 当用户选择文本时触发，调用select() 时也会触发

// 获取选择的文本

var start = minput.selectionStart;

var end = minput.selecttionEnd;

var selectStr = minput.value.substring(start,end);

//IE8及以下

var selectStr2 = document.selection.createRange().text;

// 选择部分文本

minput.setSelectionRange(0,3); // 选择前3个文本 IE8及下不支持

/\*

操作剪贴板

beforecopy 在发生复制操作之前触发

copy 在发生复制操作时触发

beforecut 在发生剪切操作之前触发

cut 在发生剪切操作时触发

beforepaste 在发生粘贴操作之前触发

paste 在发生粘贴操作时触发

要访问剪贴板中的数据，使用clipboardData 在IE中，它是window的，其他为event的

他有3个方法

getData() 接收1个参数，表示数据格式，IE中有 text 和 URL，其他的为 text/plain,不过可以用text代替

setDate() 接收2个参数，第一个表示数据格式，IE中有 text 和 URL，其他的为 text/plain,不可以用text代替 第二个参数为要设置的值

clearDate()

\*/

function getClipboardText(ev){

var ev = ev || window.event;

var clipboardText = ev.clipboardText || window.clipboardText;

return clipboardText.getData("text");

}

function setClipboardText(ev,value){

var ev = ev || window.event;

if(ev.clipboardText){

return ev.clipboardText.setDate("text/plain",value);

}else if(window.clipboardText){

return window.clipboardText.setDate("text",value);

}

}