[1. 序言 3](#_Toc28569)

[2. 准备 3](#_Toc1329)

[3. 如何嵌入网页? 4](#_Toc30911)

[4. 语言核心 4](#_Toc22089)

[4.5. 基本类型 5](#_Toc21883)

很详细的JS底层分析

序言

准备

如何嵌入网页?

写在网页中

在链接上使用伪协议

使用独立的js脚本

语言核心

关于分号

输出Html最简单的方式

alert : 弹出窗口,调试必备

命名规则

基本类型

字符串

十六进制

浮点

特殊常量

布尔值

基于对象

创建

构造函数

成员函数

继承

公共基类的属性

数组

创建

数组的函数

Date对象

null 和 undefined

函数

定义函数

一个函数,多个函数名

函数嵌套函数

函数参数

函数自身的length

arguments.callee : 递归无名函数

函数的静态变量

作用域

引用

等于

全等 (否定为!==)

+加

比较大小

in

instanceof

typeof

new

delete

void

[]

switch

label(标签)/break/continue

throw / try...catch...finally

正值表达式

浏览器中的Javascript概览

window对象

简单的对话框

状态栏

定时器

错误处理onerror

浏览器信息navigator

屏幕screen

打开窗口open

移动窗口/改变大小moveTo , moveBy , resizeTo , resizeBy

焦点控制 focus 和 blur

文档滚动

当前地址栏 location

历史History

框架

文档Documents

基本常识

属性

数组

DOM树

搜索文档的特定元素

动态改变文档内容

一个动态改变Document内容的演示

一个把文章中字母都变为大写的演示

把指定标签改为粗体

更改属性

临时容器DocumentFragment

选中内容 getSelection

CSS

当前的CSS设置

指定样式表

事件驱动

HTML中事件设置

Javascript中事件设置

事件列表

原始事件vs语义事件

同时执行原有动作和自定义动作

事件的返回值

this关键字

2级DOM标准的消息机制

Event事件对象

UIEvent事件对象

MouseEvent事件对象

MutationEvent

IE事件模型

Key Event事件对象

加载事件 onload Event

合成事件 Synthetic Events

cookie

JS库:prototype

$()等价getElementById()

$F()返回输入控件的值

$A()参数转Array

$H()转类似联合数组的Hash对象

Try.these()尝试直到一个成功

# **1. 序言**

现在学的东西很容易忘记,写这篇文章的目的是能让我在需要时快速找回当时的感觉.

入门学习和参考手册建议翻阅JavaScript.The.Definitive.Guide.5th.Edition的附录(有电子版).

# **2. 准备**

设置服务器\*.xml的MIME为text/xml ,Windows Xp自带的IIS的设置如下图 js\_0000.png

Firefox上有许多插件是开发必备的，用法大家Google，我列一下名称

FireBug

Web Developer Toolbar

GreaseMonkey 和 XmlHttpRequestDebugging

Html Validator

另外建议安装一下DreamWaver和EditPlus. EditPlus中可以快速切换IE和Js脚本,快捷键Ctrl+B 强烈建议学习时便学边调试

# **3. 如何嵌入网页?**

3.1. 写在网页中

<script type="text/javaScript">

//代码

</script>

3.2. 在链接上使用伪协议

<a href="javascript:void window.open();">Open</a>

3.3. 使用独立的js脚本

<script language="javascript" src="/js/dialog.js" />

# 4. 语言核心

4.1. 关于分号

javascript的分号是选加的,如果不加,javacript会在换行处自动加上分号.

但是,建议你不要省略分号,理由如下

加了分号的代码可以用软件压缩,来节省带宽(附带效果是加密你的Js代码:) )(Ajax基础教程 上的理由)

良好的编程习惯(Javascript权威指南 上的理由)

4.2. 输出Html最简单的方式

见例子1\_1

doucment.write("Hello World");

4.3. alert : 弹出窗口,调试必备

见例子1\_2

alert("弹出窗口,调试必备");

4.4. 命名规则

区分大小写,开头必须是字母或 $ 或 \_

# 4.5. 基本类型

4.5.1. 字符串

使用Unicode字符,如

"test"

'name="zsp"' //双引号可以放在单引号中,反之亦然

'You/'re right' //转义字符,和C++的转义相同

字符串的属性

str="zsp007@gmail.com"

str.length

//字符串的长度

str.charAt(str.length-1)

//字符串的最后一个字符(注意,第一个字符的索引是0,和C++数组类似)

str.indexOf("@")

//第一个 @ 字符的位置

str.substring(str.lastIndexOf(".")+1,str.length);

//提取后缀"com",提取规则是左开右闭(就是包括左边参数所代表的元素,而不包括右边的)

str.toUpperCase();

//返回大写形式

4.5.2. 十六进制

0xff //15\*16+15=255 , 0x为十六进制前缀

4.5.3. 浮点

-3.14 6.02E-23 //6.02 X 10-23(10的负23次方)

4.5.4. 特殊常量

Infinity 无穷大

Number.POSITIVE\_INFINITY 正无穷大

Number.NEGATIVE\_INFINITY 负无穷大

NaN 或 Number.NaN 非法的值

Number.MAX\_VALUE 可表示的最大值

Number.MIN\_VALUE 可表示的最接近0的值

4.5.5. 布尔值

true==1 false==0

4.5.6. 基于对象

4.5.6.1. 创建

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

var b = a.a ||a["a"] // 重要，访问对象的一个属性

alert(b)

方式1:

var o = new Object();

o.x=2.3;

o.y=-1.2;

方式2:

var o={x:2.3 , y:-1.2};

调用 ox //返回2.3

一个复杂点的例子 见例子1\_3

var point={x:2.3,y:-1.2};

var side=4;

var square={

        upperLeft : {x:point.x , y:point.y},

        lowerRight : {x:(point.x + side) , y:(point.y+side)}

};

alert(square["upperLeft"]["x"]);//显示2.3

备注:内建类型字串不是Object(),即用typeof得到的是"string",和"object"不等价,但在任何需要的地方他会自动转化为一个String对象

4.5.6.2. 构造函数

function Rectangle(w,h)//构造函数.不要返回值,因为返回值会使this失效

{

        this.width=w;

        this.height=h;

}

var rect1=new Rectangle(2,4);//创建对象

4.5.6.3. 成员函数

function Rectangle\_area(){return this.width \* this.height;}//声明函数

function Rectangle(w,h)

{

this.width=w;this.height=h;

this.area = Rectangle\_area;//关联函数为成员函数

}

var r=new Rectangle(3,2);

r.area();//返回6

4.5.6.4. 继承

Javascript的类都自动继承了原型对象 Object.prototype

继承的属性为只读的,既子类不可以给继承的属性赋值

子类的同名属性/函数将覆盖父类的同名属性/函数

String和Date等内部类也有原型对象

//定义一个父类

function Complex(real , imaginary){

        this.x=real;

        this.y=imaginary;

}

//为父类添加一个方法,通过定义父类原型prototype的函数

//该方法同时覆盖了Object.prototype中的同名函数

Complex.prototype.toString=function(){

        return this.x+"+"+this.y+"j";

}

//定义一个Complex函数,该函数不会被继承(这时不要加this了)

Complex.magnitude=function()

{

        return Math.sqrt(x\*x+y\*y);

}

//定义一个子类

function MoreComplex(real , imaginary){

        this.x=real;

        this.y=imaginary;

}

//继承父类,注意这时MoreComplex的构造函数被父类的构造函数覆盖

MoreComplex.prototype=new Complex(0,0);

//明确指定,使MoreComplex使用自己的构造函数

MoreComplex.prototype.constructor=MoreComplex;

//定义子类自己的函数

MoreComplex.prototype.swap=function(){

        var tmp=this.x;

        this.x=this.y;

        this.y=tmp;

}

a=new MoreComplex(11,33);

alert(a);//自动调用Complex.toString()

a.swap();

alert(a);

4.5.6.5. 公共基类的属性

Object.prototype是所有类的公共基类

constructor属性 构造器,由于构造函数确定了一个对象的类型,可以用如下代码确定一个对象的类型

if((typeof o=="object" )&& o.constructor == Date)//...........

 但不能保证constructor属性总是存在,类的创建者可以指定一个对象来取代原型的构造器,而指定的对象可以没有 constructor (PS:此句抄书,我看不懂)

toString() 返回一个字串,内容为对象的值或类型. 默认的toString对于对象返回

[object class]

//class为类名,如自定义对象为Object

//类似的还有Array,Function,Date,Math,Error,Window,Document,Form

由于可以重载toString.所以,如果你想明确的调用默认的toString来获得类型信息,你可以这样

Object.prototype.toString.apply(x);

toLocaleString() Object默认的toLocaleString返回值和toString相等;Array,Date,Number都绑定了自己的toLocaleString

valueof() Object默认定valueof不执行任何转换,Number和Boolean等有自定义的valueof(). 当你自定义了valueof时需要注意的是:valueOf()优先级比toString()高,所以,你常不得不明确的调用toString()

hasOwnProperty()

var o = new Object();

o.hasOwnProperty("undef"); //false:没有定义该属性

o.hasOwnProperty("toString"); //false:该属性是继承的

Math.hasOwnProperty("cos");//true:Math有cos属性

 \* propertylsEnumerable() 一个属性如果可以由for...in循环枚举出来就返回true,否则,返回false

var o = {x:1};

o.propertylsEnumerable("x");//true

o.propertylsEnumerable("y");//false,无该属性

o.propertylsEnumerable("valueOF");//false,该属性不可枚举

isPrototypeOf() 调用该函数的对象如果是参数的原型,就返回true

 var o=new Object();

 Object.prototype.isPrototypeOf(o);

 //true,o.constructor == Object

 Object.isPrototypeOf(o);//false

 o.isPrototypeOf(Object.prototype);//false

 Function.prototype.isPrototypeOF(Object);

 //true:Object.constructor == Function

4.5.7. 数组

4.5.7.1. 创建

方式1

var a = new Array();

a[0]=2;

a[1]="I'm a earthman ."

a[2]=true;

a[3]={x:1 , y:3}

方式2

var a=new Array(2,"I'm a earthman .",true,{x:1 , y:3});

var a2=new Array(10);//注意,这个10是表示a有10个未定义元素!(因为这时候只传给Array一个参数)

方式3

var a=[2,"I'm a earthman .",true,{x:1 , y:3}];

var a2=[[1,2,3],[1,2,3]];//可以嵌套

var base=11;

var a3[base,base+1,base+2];//可以为变量

var a4=[1,,,,5];//有3个未定义的元素

4.5.7.2. 数组的函数

\* join() 把所有的元素转换为字串,参数作为分隔符(默认分隔符是 ,)

var a=[1,2,3];

a.join();//1,2,3

a.join(" : ");//1 : 2 : 3

\* reverse() 对数组反向排序 \* sort() 默认按照UTF-8码的顺序排序,可以传一个参数,如果返回一个小于等于0的数,第1个参数出现在第2个参数前面,反之同理.

 function randOrd(){

return (Math.round(Math.random())-0.5);

}

anyArray = new Array('3','a','5','F','x','47');

anyArray.sort( randOrd );       //把数组乱序排列

\* concat() 返回一个数组,包含了原数组和参数

anyArray = new Array(47,33);

anyArray.concat(8,99,88 );

//[47,33,8,99,88]

anyArray.concat([8,99],[88,53] );

//[47,33,8,99,88,53],会自动把参数中数组拆包一层

anyArray.concat(2,[3,[4,5]]);

//[47,33,2,3,[4,5]],拆包只拆一层

\* slice() 返回一个数组的切片,返回规则如下

var a = [1,2,3,4,5]

a.slice(0,3); //返回a[0]至a[2],即[1,2,3]

a.slice(3);        //返回a[3]至结尾,即[4,5]

a.slice(1,-1);//返回a[1]至最后一个元素[2,3,4]

a.slice(-3,-2);//返回[3]

\* splice() 可以删除/添加原数组元素,同时返回删除的值

var a = [1,2,3,4,5,6,7];

a.splice(4);

//删除a[4]至结尾,同时返回删除的值,即a=[1,2,3,4],返回[5,6,7]

var a = [1,2,3,4,5,6,7];

a.splice(3,3,"a","b");

//从a[3]开始,删除3个元素(包括a[3]),同时在从a[3]开始插入"a","b"作为新的元素,返回删除的值

//即a=[1,2,3,"a","b",7],返回[4,5,6]

var a = [1,2,3,4,5,6,7];

a.splice(3,0,["a",2],"k");

//返回[],a=[1,2,3,["a",2],"k",4,5,6,7] ,splice()不会对添加的数组拆包

\* push()和pop()

var stack=[];

stack.push(1,2);//stack=[1,2],return 2(2表示数组长度)

stack.pop();//stack=[1],return 2

stack.push(3);//stack=[1,3],return 2(2表示数组长度)

stack.pop();//stack=[1],return 3

stack.push([4,5]);//stack=[1,[4,5]],return 2

stack.pop();//stack=[1],return [4,5]

stack.pop();//stack=[],return 1

\* unshift()和shift() 和push()和pop()类似,不过是在数组的头进行操作

var a=[2];

a.unshift(1);

//a=[1,2]

//IE(包括IE7)和标准的规定返回值却不同，Gecko/Opera 中返回值为新数组的长度，而 IE 中没有返回值，也就是返回值是 undefined。因此，编程时不要依赖这个返回值。

a.unshift(3,[4,5]);//a=[3,[4,5],1,2]

a.shift();

//返回3

\* toString() toString等价于不加参数的join

4.5.8. Date对象

var now = new Date();

//默认是当前的时间和日期

var christmas = new Date(2000,11,25);

//月份从0开始计数,所以这是表示2000年12月25日

christmas.toLocaleString();//返回时间的本地表示,如果是中国大陆,这里返回"2000年12月25日 0:00:00 "

//如果不加toLocalString,christmas自动转换为string时会用GMT时间

//GMT时间这里就是"Mon Dec 25 00:00:00 UTC+0800 2000 "

christmas.getDay()

//返回该日期是星期几,比如这里返回1,注意星期天返回的是0

4.5.9. null 和 undefined

null : 无值,表示它的不是有效的Javascript类型

空对象 var a = null;

空的对象 var a = {}; //没有任何属性和方法的对象

undefined : 已经声不赋值的变量,对象的一个不存在的属性也是该值

==运算符将两者视为相等,

可以用===区别null和undefinedes

4.6. 函数

4.6.1. 定义函数

方式1:

function square(x){

        return x\*x;

}

方式2:

var square = function(x){return x\*x;}

方式3://比较笨拙

var square = new Funtion("x","return x\*x");

4.6.2. 一个函数,多个函数名

var square = function(x){return x\*x;}

var b=square;

var c=b(5);//c=25

4.6.3. 函数嵌套函数

function hypotenuse(){

        function square(x){

                return x\*x;

        }

        return Math.sqrt(square(a)+square(b));

}

4.6.4. 函数参数

尽管函数定义时参数个数是固定的,但调用时可以传递任意多个参数,这些参数可以通过arguments[]访问,即使是无名的.

arguments.length可以说明它所含的元素个数

注意:arguments不是数组(不可以用for...in循环),它是一个Aguments对象

arguments可以用于参数个数检查,或接受任意数目参数...

function max()

{

var m=Number.NEGATIVE\_INFINITY;

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

        if(arguments[i]>m)m=arguments[i];

return m;

}

4.6.5. 函数自身的length

返回函数声明中的参数的个数,通过arguments.callee.length访问

//一个检查参数是否相符的函数

function check(args){

        var actual=args.length;

        var expected=args.callee.length;

        if(actual!=expected){

                throw new Error("Wrong number of arguments");

        }

}

function f(x,y)

{

        check(arguments);

        return x+y;

}

4.6.6. arguments.callee : 递归无名函数

callee可以引用当前正在执行的函数

function(x)

{

if(x<1)return 1;

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

4.6.7. 函数的静态变量

类似在C++函数中声明一个static变量

uniqueInteger.counter=0;//静态变量

function uniqueInteger(){//每个返回一个不同的数

        return uniqueInteger.counter++;

}

4.7. 作用域

没有块级作用域,其他类似于Java和C++.

即函数中无论在那里声明的变量,在整个函数体中都是有定义的.如:

(见例1\_4)

var i="global";

function test(o){

document.write(i);//i为undefined,即函数中i已被声明,但没定义

var i="local";

if(typeof o == "object")

        {

                var j=0;        //j在整个函数中都有定义,不限于这个循环

                for(var k=0;k<10;k++){//k在整个函数中都有定义,不限于这个循环

                        document.write(k);

                }

                document.write("out loop k=",k);//k仍然有定义,值为10

        }

document.write("out loop j=",j);//仍然有定义,如果o不是object(),则j没有被初始化,值为undefined,否则为9

}

备注:把所有的变量集中在函数开头声明(不一定要定义)可以避免许多诡异的错误.

4.8. 引用

对与 数组 , 对象 , 函数 的拷贝只是拷贝了引用(相当于是个快捷方式),对任何一个副本的改动都会改动所有的副本

var a = [1 , 2 , 3];

var b = a;

a[0] = 99;

alert(b); //为99,2,3

a={x:1}

b=a;

a.x=2;

alert(b.x);//为2

4.9. ＝＝（等于）和＝＝＝（全等）

4.9.1. 等于

等于可以进行类型自动转换,如

"2"==2

true==1

一个对象和字串或数字比较,会尝试valueOf()和toString()转换

null == undinfined

4.9.2. 全等 (否定为!==)

如果类型不同,则不相同

NaN永远不与任何值全等,包括它本身(可以用IsNaN()测试一个值是否是NaN)

按字符原有编码进行比较(等于操作在比较前先统一了编码)

null != undinfined

对于数组,对象,函数等引用类型只有当他们是指向同一个实体时才相等,如

var a = [1,2,3];

var b = [1,2,3];

alert(a==b);//false

4.10. +加

加法是从左到右的 a=1+2+" blind mine" // a="3 blind mine" b="blind mine "+1+2 // a="blind mine 12"

4.11. 比较大小

符号> , < , >= , <=

任何和NaN的比较都返回false

字串和数字比较,会把字串转换为数字(如果转换不成功,则其值为NaN),再比较

如果对象可以被转换为数字和字串,将执行数字转换来比较

4.12. in

in运算符左边是一个字串,右边是一个对象,如果字串是对象的一个属性名则返回true

var point = {x:1};

var has\_x = "x" in point ; //true

var has\_y = "y" in point; //false

var ts = "toString" in point; //true,继承属性

4.13. instanceof

instanceof运算符要求左边是一个对象的实例,右边是对象名,如果两者相符返回true

var d=new Date();

d instanceof Date;//true

d instanceof Object;//true,所有的对象都是Object的实例

d instanceof Number;//false

4.14. typeof

返回对象的类型

typeof 1; //number

typeof "1";//string

typeof true;//boolean

typeof [1,2];//object

typeof {x:1};//object

typeof function(x){};//function

typeof xxx;//未定义的对象返回undefined

4.15. new

创建一个新的对象

o=new Date;

4.16. delete

删除对象的属性

var o={x:1,y:2};

delete o.x; //返回true

typeof o.x;//返回undefined

delete o.x;//删除不存在的属性,返回true

delete o;//不能删除var声明的对象,返回true

x=1;//不用var关键字,隐式声明一个变量

delete x;//返回true

x;//一个运行时错误,IE6显示为 " 'x'未定义 "

注意:delete只能删除属性值,不能删除引用对象 如

var my = new Object;

my.hire = new Date;

my.fire = my.hire;

delete my.hire;

alert(my.fire);//显示当前的时间

4.17. void

可以出现在任何操作数前,它作用是舍弃运算的结果,返回undefined.

常用于javascript : URL中.可以计算一个计算一个表达式的值,又不让浏览器显示它.如

<a href="javascript:void window.open();">Open</a><!-- 不加void,点击后链接会消失,变为一个"[object]"字串 -->

4.18. []

//用于数组

var a=[21,3];

a[0];//21

//用于对象,比"."运算符的优势是其要取的属性名可以动态生成

var b={x1:1,x2:"zsp"};

b['x1'];//1

b['x'+2];//zsp

for(f in o){

        alert('o.'+f+'='+o[f]);//由于f是动态生成的,所有不能用" . "运算符

}

== if/else/while/do...while/for== 和C++类似

== for...in === 可循环对象的所有用户定义属性,包括继承的(但是不包括内建属性)

var o={x:1,y:2};

var a=[];

var i=0;

for(a[i++] in o);//把o的属性名复制到一个数字中(注意,顺序不固定)

for (i in a)

{

alert(a[i]);//循环一个数组

}

4.19. switch

function convert(x){

        switch(typeof x){

                case 'number':  //case后可以接任意表达式,不限于常量

                        return x.toString(16);//转换为16进制

                case 'string':

                        return '"'+x+'"';

                case 'boolean':

                        return x.toSting().toUpperCase();

                default:

                        return x.toString();

        }

}

4.20. label(标签)/break/continue

break,continue的基本用法和C++中类似,不过他们可以和label配合使用(有的像Java中的label,功能限制版的goto)

outerloop:

        for(var i=0;i!=10;i++)

        {

                for (var j=0;j<10;j++){

                        if(j>3)break;

                        if(i==2)break outerloop;//跳出外层循环

                        alert("j = "+j);

                }

                alert("i = "+i);

        }

4.21. throw / try...catch...finally

throw用来抛出异常,异常对象通常是一个Error对象或其子类型

function factorial(x){

if(x<0)throw new Error("x must not be negative");//x是负数就抛出异常

return x;

}

try...catch...finally捕捉异常

try{

//代码

}

catch(e)

{

//仅当try块抛出异常,才会执行,e为异常实例

//这个块可以处理异常,也可以什么都不做

}

finally{

//无论是否有异常,这个块中的语句都将被执行

}

4.22. 正值表达式

//todo

5. 浏览器中的Javascript概览

window对象代表显示浏览器的窗口,document对象代表文档中窗口.

定义函数的js文件一般放在head区

<body>和<frameset>的onload在文档被完全加载完成时触发,onunload在文档被卸载时触发

5.1. window对象

window是全局对象,它的函数/属性一般可以不加前缀直接调用。有时为了避免名字冲突,可以加window前缀

5.1.1. 简单的对话框

注意:对话框中显示的是纯文本. \* alert() 见图js\_0001.png

\* confirm() 见图js\_0002.png

if(confirm("看我的非技术Blog吗?/n/n我写了不少诗哦 :)"))       location.replace("http://www.cnweblog.com/zuroc/");

\* prompt() 见图js\_0003.png

n=prompt("你叫什么名字呢?","");//第二个参数是输入的默认值

document.write("<h1>你好, "+n+"!<br/>我叫张沈鹏.</h1>");

5.1.2. 状态栏

\* status 浏览器状态栏的显示文本 \* defaultStatus 浏览器状态栏显示文本的默认值 该属性在firefox默认关闭,不介绍了

5.1.3. 定时器

\* setTimeout() 安排一个JavaScript代码在指定的时间后执行 \* clearTimerout() 取消setTimeout的执行 \* setInterval() 每隔指定的时间调用指定的函数 \* clearInterval() 取消Interval的执行

function hi(){alert("hi!");}

setInterval("hi()",2000);//2000ms=2s

5.1.4. 错误处理onerror

只要给这个属性赋了一个函数,那么这个窗口只要发生了javascript错误,该函数就会被调用.传给错误处理函数的参数有三个,第一个是错误类型,第二个是引发错误的js脚本的Url,第三是发生错误的行号.

如果onerror返回true,它将屏蔽系统的错误处理程序

//如果你不想错误消息打搅用户,无论你的代码有多Bug

window.onerror = function(){return true;}

//不过,不建议你这样做

5.1.5. 浏览器信息navigator

包含了浏览器的信息

\* appName Web浏览器的名称 \* userAgent 浏览器在USER-AGENT HTTP标题中发送的字符串,通常包含了appName和appVersion \* platform 平台,如Win32

//一个简单浏览器探测器

var browser = {

    version: parseInt(navigator.appVersion),

    isNetscape: navigator.appName.indexOf("Netscape") != -1,

    isMicrosoft: navigator.appName.indexOf("Microsoft") != -1

};

5.1.6. 屏幕screen

\* width,height 分辨率的宽和高 \* availWidth , availHeight 去除了任务栏这样的特性的宽和高 \* colorDepth 屏幕的色深 见例子1\_5.html

5.1.7. 打开窗口open

//注意,直接弹出窗口容易被浏览器广告拦截了

window.open("http://www.cnweblog.com/zuroc/","窗口名","width=200,height=200,status=yes,resizable=yes");

//具体属性希望那位大哥可以帮我列一下

//被打开的窗口可以用opener引用父窗口,如果是用户自己打开的opener为null

//用open时一般要明确加上window前缀,因为Document也有open属性

见例子1\_5.html

window.close可以关闭你用javascript打开的窗口

5.1.8. 移动窗口/改变大小moveTo , moveBy , resizeTo , resizeBy

\* moveTo 把窗口左上角移到指定的坐标 \* moveTo 把窗口相对现在的位置移动指定的象素 \* resizeTo 按照绝对大小调整窗口 \* resizeBy 按照相对大小调整窗口 见例子1\_5.html

5.1.9. 焦点控制 focus 和 blur

\* focus() 将焦点给指定窗口,同时把窗口移动到最前端,使窗口可见,当open()打开窗口,如果第二个参数指定的窗口名已经存在,open不会自动使那个窗口可见,这时就要调用该函数 \* blur() 放弃焦点,这时窗口将被至于窗口队列的最后(这就什么sohu背投广告的原理吧) 见例子1\_5.html

5.1.10. 文档滚动

\* scrollBy() 相对当前位置滚动指定的象素 \* scrollTo() 将文档滚动到一个绝对位置,文档的左上角的坐标为(0,0)

5.1.11. 当前地址栏 location

它本身代表当前窗口的URL,它还有各个属性如下 \* location.protocol 用的协议,如在本地直接打开网页则是file: \* location.host 用的域名 \* location.pathname 域名后的网址路径.如"/F:/Javascript绝对简明教程/example/test.html " \* location.search 查询的内容 \* location.reload() 刷新当前线 \* location.replace() 跳转到指定的URL,与直接给location赋值不同的是,这样不会在浏览历史中留下记录(也就是后退不会到该页)

//一个把URL串中查询内容解析为键值对的函数

function getArgs(){

        var args=new Object;

        var query=location.search.substring(1);//或取查询串

        var pairs=query.split(",");     //在逗号处分开

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

                var pos=pair[i].indexOf("=");

                var argname=pairs[i].substring(0,pos);

                var value=pairs[i].substring(pos+1);

                args[argname]=decodeURIComponent(value);

        }

        return args;

}

//调用可以这样

var args=getArgs();

if(args.x)x=parseInt(args.x);

//检查浏览器是否支持DOM,如果不支持则跳转

if (!document.getElementById) location = "staticpage.html";

5.1.12. 历史History

脚本不能真正访问History数组,但是可以调用它的函数 \* back() 后退 \* forward() 前进 \* go() 后退/前进指定的页数

5.1.13. 框架

一个窗口可以用frames,parent,top引用其他属性

每个窗口有一个frame[]数组(如果一个窗口没有框架则它的frame[]是空的,frames.length==0),frame[0]表示它的第一个子框架,如此类推

每个窗口还有一个parent属性,为包含这个窗口的window对象,这样一个窗口通过诸如parent.frames[1]的方式,引用他的兄弟窗口

对于顶层窗口,有parent==window

对于框架中的框架,可以用top直接引用最顶层的窗口

见图js\_0004.png

框架可以用name指定名字,这个名字可以为链接的target值.当这个链接被激活的时候会在指定的框架中显示内容

在javascript也可以通过名字访问框架,如一个框架如下

<frame name="table\_of\_contents" src="top.html">

 则可以通过parent.table\_of\_contents方法访问该框架

需要注意的在一个框架定义的函数/变量,在另一个框架中要通过框架名来调用

见例1\_6.html

5.2. 文档Documents

5.2.1. 基本常识

5.2.1.1. 属性

\* bgColor , fgColor 文档背景色,前景色(不赞成使用的属性) \* lastModified 文档的最后修改时间

//可以在文档末加入

  最后更新:<script>document.write(document.lastModified);</script>

\* referrer 如果该属性存在,则返回一个链接,表示用户是由哪个网页链接到该页的

<script>

//如果你想其他由网站链接到该页的用户先访问你的首页

if (document.referrer == "" || document.referrer.indexOf("mysite.com") == -1)

    window.location = "http://home.mysite.com";

</script>

5.2.1.2. 数组

\* title 文档标题 \* URL 等同于window的location.href \* anchors[] 所有有name属性的a标记 \* forms[] 所有form标记,每个form还有一个element[]数组 \* images[] 所有的<img>标签,可以改变它的src属性来改变图象 \* link[] 文档中所有的超链接,和window的location一样有protocol, hostname,pathname等属性

<form name="f1"><input type="button" value="Push Me" name="zipcode"></form>

//如果一个元素有name还可以这样访问

document.forms.f1

document.forms["f1"]

//对于它的子元素可以这样访问

document.shipping.zipcode

//如果有重名的元素,以上的引用方法将返回一个数组

5.2.2. DOM树

见图js\_0005.png 对于树上的节点可以用firstChild, lastChild, nextSibling, previousSibling, 和 parentNode属性来历遍

appendChild(), removeChild(), replaceChild(), 和 insertBefore() 可以在文档中动态的添加/删除节点

每个节点都有一个nodeType属性,表明了它的类型

Interface                       nodeType constant                       nodeType value

Element                 Node.ELEMENT\_NODE                        1

Text                            Node.TEXT\_NODE                           3

Document                        Node.DOCUMENT\_NODE                       9

Comment                 Node.COMMENT\_NODE                        8

DocumentFragment  Node.DOCUMENT\_FRAGMENT\_NODE 11

Attr                            Node.ATTRIBUTE\_NODE                      2

getAttribute(), setAttribute(), removeAttribute() 可以用来操作属性,标准的Html元素的属性也可以通过这种方式访问 getAttributeNode()可以返回一个Attr对象,它有一个specified属性,可以判断文档是否指定了该属性,或判断它是否是默认值

//历便一个DOM树

function countTags(n) {                         // n is a Node

    var numtags = 0;                            // Initialize the tag counter

    if (n.nodeType == 1 /\*Node.ELEMENT\_NODE\*/)  // Check if n is an Element

        numtags++;                              // Increment the counter if so

    var children = n.childNodes;                // Now get all children of n

    for(var i=0; i < children.length; i++) {    // Loop through the children

        numtags += countTags(children[i]);      // Recurse on each one

    }

    return numtags;                             // Return the total

}

alert('This document has ' + countTags(document) + ' tags');

5.2.2.1. 搜索文档的特定元素

\* document.getElementsByTagName() 返回指定标签名的数组

//虽然只有一个body,还是返回一个数组

document.getElementsByTagName("body")[0];

//有多少表格

var tables = document.getElementsByTagName("table");

alert("This document contains " + tables.length + " tables");

\* document.getElementById() 返回id号为指定值元素,因为每个id应该的独一无二的,所以该方法最多返回一个元素

<p id="specialParagraph">

var myParagraph = document.getElementById("specialParagraph");

\* document.getElementsByName() 返回name属性为指定值的元素的数组

// Find <a name="top">

var link = document.getElementsByName("top")[0];

// Find all <input type="radio" name="shippingMethod"> elements

var choices = document.getElementsByName("shippingMethod");

5.2.2.2. 动态改变文档内容

\* Document.createElement() \* Document.createTextNode() \* Node.appendChild() \* Node.insertBefore() \* Node.replaceChild()

5.2.2.3. 一个动态改变Document内容的演示

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>test</title>

<script>

function sortkids(e) {

     // This is the element whose children we are going to sort

     if (typeof e == "string") e = document.getElementById(e);

     // Transfer the element (but not text node) children of e to a real array

     var kids = [];

     for(var x = e.firstChild; x != null; x = x.nextSibling)

         if (x.nodeType == 1 /\* Node.ELEMENT\_NODE \*/) kids.push(x);

     // Now sort the array based on the text content of each kid.

     // Assume that each kid has only a single child and it is a Text node

     kids.sort(function(n, m) { // This is the comparator function for sorting

                   var s = n.firstChild.data; // text of node n

                   var t = m.firstChild.data; // text of node m

                   if (s < t) return -1;      // n comes before m

                   else if (s > t) return 1;  // n comes after m

                   else return 0;             // n and m are equal

               });

     // Now append the kids back into the parent in their sorted order.

     // When we insert a node that is already part of the document, it is

     // automatically removed from its current position, so reinserting

     // these nodes automatically moves them from their old position

     // Note that any text nodes we skipped get left behind, however.

     for(var i = 0; i < kids.length; i++) e.appendChild(kids[i]);

}

</script>

</head>

<body>

<ul id="list"> <!-- This is the list we'll sort -->

<li>one<li>two<li>three<li>four <!-- items are not in alphabetical order -->

</ul>

<!-- this is the button that sorts the list -->

<button onclick="sortkids('list')">Sort list</button>

</body>

</html>

5.2.2.4. 一个把文章中字母都变为大写的演示

// This function recursively looks at Node n and its descendants,

// converting all Text node data to uppercase

function upcase(n) {

    if (n.nodeType == 3 /\*Node.TEXT\_NODE\*/) {

        // If the node is a Text node, change its text to uppercase.

        n.data = n.data.toUpperCase();

        //你也可以用appendData(), insertData(), deleteData(), replaceData()来更改数据

    }

    else {

        // If the node is not a Text node, loop through its children

        // and recursively call this function on each child.

        var kids = n.childNodes;

        for(var i = 0; i < kids.length; i++) upcase(kids[i]);

    }

}

5.2.2.5. 把指定标签改为粗体

<script>

// This function takes a Node n, replaces it in the tree with an Element node

// that represents an HTML <b> tag, and then makes the original node the

// child of the new <b> element.

function embolden(n) {

    if (typeof n == "string") n = document.getElementById(n); // Lookup node

    var b = document.createElement("b"); // Create a new <b> element

    var parent = n.parentNode;           // Get the parent of the node

    parent.replaceChild(b, n);           // Replace the node with the <b> tag

    b.appendChild(n);                    // Make the node a child of the <b> element

}

</script>

<!-- A couple of sample paragraphs -->

<p id="p1">This <i>is</i> paragraph #1.</p>

<p id="p2">This <i>is</i> paragraph #2.</p>

<!-- A button that invokes the embolden() function on the element named p1 -->

<button onclick="embolden('p1');">Embolden</button>

5.2.2.6. 更改属性

var headline = document.getElementById("headline");  // Find named element

headline.setAttribute("align", "center");            // Set align='center'

5.2.2.7. 临时容器DocumentFragment

DocumentFragment是一个用来装载DOM对象的临时容器,将他写入文档其实是将它的子元素写入节点

// 翻转元素

function reverse(n) {

    // Create an empty DocumentFragment as a temporary container

    var f = document.createDocumentFragment();

    // Now loop backward through the children, moving each one to the fragment.

    // The last child of n becomes the first child of f, and vice-versa.

    // Note that appending a child to f automatically removes it from n.

    while(n.lastChild) f.appendChild(n.lastChild);

    // Finally, move the children of f all at once back to n, all at once.

    n.appendChild(f);

}

==== 直接插入Html代码innerHTML ==== 虽然不是W3C的标准,但是事实上的标准,类似的还有outerHTML, innerText, 和outerText

var table = document.createElement("table");  // Create the <table> element

table.border = 1;                             // Set an attribute

// Add a Name|Type|Value header to the table

5.2.2.8. 选中内容 getSelection

还不是标准,但还是可用的,参考 http://www.quirksmode.org/js/selected.html

Hi , everybody . My name is Zsp .

function getSelectedText() {

    if (window.getSelection) {

        // This technique is the most likely to be standardized.

        // getSelection() returns a Selection object, which we do not document.

        return window.getSelection().toString();

    }

    else if (document.selection) {

        // This is the IE-specific technique.

        // We do not document the IE selection property or TextRange objects.

        return document.selection.createRange().text;

    }

    //现在可以省略了

    else if (document.getSelection) {

        // This is an older, simpler technique that returns a string

        return document.getSelection();

    }

}

//一个实际使用的演示

<a href="javascript:

    var q;

    if (window.getSelection) q = window.getSelection().toString();

    else if (document.getSelection) q = document.getSelection();

    else if (document.selection) q = document.selection.createRange().text;

    void window.open('http://en.wikipedia.org/wiki/' + q);

">

Look Up Selected Text In Wikipedia

</a>

5.2.3. CSS

//一个用来隐藏广告代码的演示

var imgs = document.getElementsByTagName("img");  // Find all images

for(var i = 0; i < imgs.length; i++) {            // Loop through them

    var img=imgs[i];

    if (img.width == 468 && img.height == 60)     // If it's a 468x60 banner...

        img.style.visibility = "hidden";          // hide it!

}

\* javascript与css的名字映射,如font-family映射为fontFamily,诸如此类.另外css中的float映射为cssFloat,如此类推

e.style.position = "absolute";

e.style.fontFamily = "sans-serif";

e.style.backgroundColor = "#ffffff";

e.style.left = "300px";

e.style.margin = topMargin + "px " + rightMargin + "px " +

                 bottomMargin + "px " + leftMargin + "px";

5.2.3.1. 当前的CSS设置

W3C规定了getComputedStyle()函数来获取CSS元素的经过各种效果复合过后的当前值

但是IE不支持标准,但它有每个元素都有一个currentStyle属性,可以取得同样的效果

//一段兼容的代码

<p>

Look Up Selected Text In Wikipedia

</p>

<script>

var p = document.getElementsByTagName("p")[0]; // Get first paragraph of doc

var typeface = "";                             // We want its typeface

if (p.currentStyle)                            // Try simple IE API first

    typeface = p.currentStyle.fontFamily;

else if (window.getComputedStyle)              // Otherwise use W3C API

    typeface = window.getComputedStyle(p, null).fontFamily;

alert(typeface);

</script>

5.2.3.2. 指定样式表

<head>

<!-- Here we define four stylesheets, using <link> and <style> tags. -->

<!-- Two of the <link>ed sheets are alternate and so disabled by default. -->

<!-- All have id attributes so we can refer to them by name. -->

<link rel="stylesheet" type="text/css" href="ss0.css" id="ss0">

<link rel="alternate stylesheet" type="text/css" href="ss1.css"

      id="ss1" title="Large Type">

<link rel="alternate stylesheet" type="text/css" href="ss2.css"

      id="ss2" title="High Contrast">

<style id="ss3" title="Sans Serif">

body { font-family: sans-serif; }

</style>

<script>

// This function enables or disables a stylesheet specified by id.

// It works for <link> and <style> elements.

function enableSS(sheetid, enabled) {

    document.getElementById(sheetid).disabled = !enabled;

}

</script>

</head>

<body>

<!-- This is a simple HTML form for enabling and disabling stylesheets -->

<!-- It is hardcoded to match the sheets in this document but could be -->

<!-- dynamically generated using stylesheet titles instead. -->

<form>

<input type="checkbox" onclick="enableSS('ss0', this.checked)" checked>Basics

<br><input type="checkbox" onclick="enableSS('ss1', this.checked)">Large Type

<br><input type="checkbox" onclick="enableSS('ss2', this.checked)">Contrast

<br><input type="checkbox" onclick="enableSS('ss3', this.checked)" checked>

Sans Serif

</form>

5.2.4. 事件驱动

5.2.4.1. HTML中事件设置

html元素对事件进行响应(不要放在<script>块中)

<input type="checkbox" name="opts" value="ignore-case" onclick="ignore-case=this.checked;">

其中onclick就是事件触发器,常用的还有onmousedown,onmouseup,onmouseover,onmouseout,onchange,onsubmit,onreset

5.2.4.2. Javascript中事件设置

<form name="myform" onsubmit="return validateform();">...</form>

//以上事件设置在javascript中的等效代码如下,注意,javascript中的所有事件名都是小写的

document.myform.onsubmit = validateform;

//可以在脚本中批量设置事件触发器

function confirmAllLinks( ) {

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

        document.links[i].onclick = confirmLink;

    }

}

5.2.4.3. 事件列表

事件名                     支持该事件的元素

onabort                 <img>

Image loading interrupted.

onblur                   <button>, <input>, <label>, <select>, <textarea>, <body>

 Element loses input focus.

onchange                        <input>, <select>, <textarea>

Selection in a <select> element or other form element loses focus, and its value has changed since it gained focus.

 onclick                        Most elements

 Mouse press and release; follows mouseup event. Return false to cancel default action (i.e., follow link, reset, submit).

 ondblclick                     Most elements

 Double-click.

 onerror                        <img>

 Error when loading image.

onfocus                 <button>, <input>, <label>, <select>, <textarea>, <body>

 Element gains input focus.

 onkeydown               Form elements and <body>

 Key pressed down. Return false to cancel.

 onkeypress             Form elements and <body>

 Key pressed; follows keydown. Return false to cancel.

onkeyup                 Form elements and <body>

Key released; follows keypress.

onload                  <body>, <frameset>, <img>

Document load complete.

onmousedown             Most elements

Mouse button pressed.

onmousemove             Most elements

Mouse moved.

onmouseout              Most elements

Mouse moves off element.

onmouseover             Most elements

Mouse moves over element.

onmouseup               Most elements

Mouse button released.

onreset                 <form>

Form reset requested. Return false to prevent reset.

 onresize                       <body>, <frameset>

 Window size changes.

 onselect                       <input>, <textarea>

 Text selected.

onsubmit                        <form>

Form submission requested. Return false to prevent submission.

onunload                        <body>, <frameset>

离开页面时

5.2.4.4. 原始事件vs语义事件

\* 原始事件:如键盘输入,鼠标按下 \* 语义事件:如Sumbit事件,他可以由提交按钮上的onmousedown, onmouseup和onclick产生的.也可以由键盘上的回车产生.总的来说,语义事件往往与产生事件的设备无关.

5.2.4.5. 同时执行原有动作和自定义动作

var b=document.myform.mybutton;//我们需要的按钮

var olderHandler=b.onclick;//HTML原有自定义事件

function newHandler(){

//自定义动作

}

b.onclick=function(){olderHandler();newHandler();}

5.2.4.6. 事件的返回值

事件默认返回true,会继续执行默认动作.

onsubmit , onclick , onkeydown , onkeypress , onmousedown , onmouseup 和 onreset 返回false会阻止他们的默认行为发生.

5.2.4.7. this关键字

button.onclick=o.mymethod;//这时o.mymethod中的this指代的是button

button.onclick=function(){o.mymethod();}//这时o.mymethod中的this指代是o

5.2.4.8. 2级DOM标准的消息机制

这种事件处理方式被Firefox系列等支持.

浏览器中,当事件被触发时的流程

1.capturing(捕捉):事件按照Document树从上往下传播直到目标节点,如果目标的 任何一个父元素注册了caputure该事件的函数, 则会运行这些函数

2.运行目标节点的该事件处理函数

3.bubbling(起泡):事件将从目标元素回传至Document根节点(语义事件一般不起泡).

4.如果存在默认事件,执行它

2级DOM中事件名称是原始事件名把on前缀去除.如"onmousedown"在2级DOM中叫"mousedown"

\* 在事件传播过程中,任何事件处理函数都可以调用stopPropagation()来停止事件的传播.

\* addEventListener来注册事件,removeEventListener()来移除事件,你也可以给Text节点注册事件处理函数

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

//捕捉<div>中所有的mousedown事件

//一个元素的同一个事件可以注册多个处理函数,但是他们的调用顺序不一定是注册顺序

mydiv=addEventListener(

"mousedown",

function(event){

                //函数体

        },

false//为true则在传播阶段捕捉,为false在起泡阶段捕捉

);

//Firefox中一个对象实现了EventListener接口和handleEvent()方法,便可以直接作为一个参数传给addEventHandler

//一个简单的函数,可以实现类似效果

function registerObjectEventHandler(element, eventtype, listener, captures) {

    element.addEventListener(

                             eventtype,

                             function(

                                  event//一个事件对象

                             )

                             { listener.handleEvent(event); }

                             captures

                             );

}

一个事件被触发时会产生一个相应的事件对象,这个对象提供了关于这个事件的详细信息,如:发生坐标 如在e.addEventListener("click", f, false);中,事件对象会作为参数传给f

事件对象的类型分 \* 模块名:HTMLEvents

事件接口:Event

触发事件:abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload

\* 模块名:MouseEvents

事件接口:MouseEvent

触发事件:click, mousedown, mousemove, mouseout, mouseover, mouseup

\* 模块名:UIEvents

事件接口:UIEvent 触发事件:DOMActivate, DOMFocusIn, DOMFocusOut

                        事件<->类型表

C:表示能否用preventDefault()取消默认动作.

事件                      类型                      起泡              C               由...支持

abort           Event            yes            no               <img>, <object>

blur                    Event             no            no               <a>, <area>, <button>, <input>, <label>, <select>, <textarea>

change          Event            yes            no               <input>, <select>, <textarea>

click                   MouseEvent       yes            yes              screenX, screenY, clientX, clientY,altKey,

                                                                                 ctrlKey, shiftKey, metaKey, button, detail

error                   Event            yes            no               <body>, <frameset>, <img>, <object>

focus           Event             no            no               <a>, <area>, <button>, <input>, <label>, <select>, <textarea>

load                    Event             no            no               <body>, <frameset>, <iframe>, <img>, <object>

mousedown       MouseEvent       yes            yes              screenX, screenY, clientX, clientY, altKey,

                                                                                 ctrlKey, shiftKey, metaKey, button, detail

mousemove       MouseEvent       yes             no              screenX, screenY, clientX, clientY, altKey,

                                                                                 ctrlKey, shiftKey, metaKey

mouseout                MouseEvent       yes             yes            screenX, screenY, clientX, clientY, altKey,

                                                                                ctrlKey, shiftKey, metaKey, relatedTarget

mouseover       MouseEvent       yes             yes             screenX, screenY, clientX, clientY, altKey,

                                                                                 ctrlKey, shiftKey, metaKey, relatedTarget

mouseup         MouseEvent       yes             yes             screenX, screenY, clientX, clientY, altKey,

                                                                                 ctrlKey, shiftKey, metaKey, button, detail

reset                   Event            yes             no              <form>

resize          Event            yes             no              <body>, <frameset>, <iframe>

scroll                  Event            yes             no              <body>

select          Event            yes             no              <input>, <textarea>

submit          Event            yes             yes             <form>

unload          Event            no              no              <body>, <frameset>

DOMActivate     UIEvent          yes            yes               detail

DOMFocusIn      UIEvent          yes            no                none

DOMFocusOut   UIEvent            yes            no                none

5.2.4.8.1. Event事件对象

Event是其他所有的事件父类,所有的事件都实现了这个接口.

支持属性:

type 触发的事件的名称,如"click" 或 "mouseover"

target 发生事件的节点

currentTarget 当前正在处理该事件的节点,在捕捉阶段或起泡阶段他的值与target的值不同.在事件处理函数中应该用这个属性而不是this来指代当前处理的节点

eventPhase 一个数字,指定了当前事件所处的传播过程的阶段.它的值是常量,可以是Event.CAPTURING\_PHASE或Event.AT\_TARGET或Event.BUBBLING\_PHASE.

timeStamp 一个Date对象,表明事件的发生时间

bubbles 布尔值,表明该事件是否起泡

cancelable 布尔值,表明事件是否可以用preventDefault()方法来取消默认的动作.

stopPropagation( ) 停止事件的传播

preventDefault( ) 阻止默认事件发生,和0级DOM中返回false的作用类似

5.2.4.8.2. UIEvent事件对象

UIEvent是Event的一个子类

支持属性:

view 视图,发生事件的window对象

detail 一个数字,提供关于该事件的额外信息.对于click事件,mousedown事件和mouseup事件表示连续点击的次数,如1表示单击,2表示双击,3表示连续点3次(注意:每次点击还是生成一个事件,in a word,detail为2的鼠标事件前总有一个detail为1的鼠标事件,detail为3的鼠标事件前总有一个detail为2的鼠标事件).

对于DOMActivate事件,1表示正常激活,2表示超级激活(如双击鼠标或同时按下Shift和Enter键)

5.2.4.9. MouseEvent事件对象

MouseEvent是UIEvent的一个子类

支持属性:

button 一个数字,表明在mousedown,mouseup和click中,那个鼠标键改变了状态.0表示左键,1表示中键,2表示右键.

altKey , ctrlKey, metaKey, shiftKey 表明对应的键盘上的键的状态(PC上,Meta键通常对应 Alt 键)

screenX , screenY 鼠标指针相对于屏幕左上角X,Y坐标,如果你想在鼠标附近弹出一个窗口,这个值很有用

clientX, clientY 鼠标指针相对于浏览器页面显示部分左上角的X,Y坐标,注意,这个值不考虑文档滚动.如果需要相对于文档的位置,IE中可以加上document.body.scrollLeft和document.body.scrollTop,其他浏览器可以加上window.pageXOffset and window.pageYOffset

relatedTarget 该属性仅对mouseover和mouseout生效.

对于mouseover是它离开的节点,对于mouseout是它进入的节点.

5.2.4.9.1. MutationEvent

不常用,不介绍

5.2.4.10. IE事件模型

IE不完全支持标准事件模型,它使用IE的Event模型,它有以下重要属性.

type 兼容DOM

srcElement 等价DOM中的target

button 和DOM的MouseEvent的同名属性类似,但1为左键,2为右键,4为中键,如果多个键同时按下,会把他们的属性值相加.如同时按下左右键,值为3

clientX , clientY 兼容DOM

offsetX, offsetY 鼠标指针相对于源元素的位置,用他们可以确定点击了Image的哪个象素,2级DOM无等价事件

altKey , ctrlKey, shiftKey 等价于DOM,无metaKey属性

keyCode 声明了keydown,keyup的键代码,和keypress事件的Unicode字符.用String.fromCharCode()方法可以把字符代码转换为字符.()

fromElement, toElement 类似DOM的relatedTarget

cancelBubble 设置为True时,阻止事件的起泡.类似DOM的stopPropagation( )

returnValue 设置为false时,阻止默认行为.类似DOM的preventDefault( )

/\*

 \* Handler.js -- Portable event-handler registration functions

 \*

 \* This module defines event-handler registration and deregistration functions

 \* Handler.add( ) and Handler.remove( ). Both functions take three arguments:

 \*

 \*   element: the DOM element, document, or window on which the handler

 \*      is to be added or removed.

 \*

 \*   eventType: a string that specifies the type of event for which the

 \*      handler is to be invoked. Use DOM-standard type names, which do

 \*      not include an "on" prefix. Examples: "click", "load", "mouseover".

 \*

 \*   handler: The function to be invoked when an event of the specified type

 \*      occurs on the specified element. This function will be invoked as

 \*      a method of the element on which it is registered, and the "this"

 \*      keyword will refer to that element. The handler function will be

 \*      passed an event object as its sole argument. This event object will

 \*      either be a DOM-standard Event object or a simulated one. If a

 \*      simulated event object is passed, it will have the following DOM-

 \*      compliant properties: type, target, currentTarget, relatedTarget,

 \*      eventPhase, clientX, clientY, screenX, screenY, altKey, ctrlKey,

 \*      shiftKey, charCode, stopPropagation( ), and preventDefault( )

 \*

 \* Handler.add( ) and Handler.remove( ) have no return value.

 \*

 \* Handler.add( ) ignores duplicate registrations of the same handler for

 \* the same event type and element. Handler.remove( ) does nothing if called

 \* to remove a handler that has not been registered.

 \*

 \* Implementation notes:

 \*

 \* In browsers that support the DOM standard addEventListener( ) and

 \* removeEventListener( ) event-registration functions, Handler.add( ) and

 \* Handler.remove( ) simply invoke these functions, passing false as the

 \* third argument (meaning that the event handlers are never registered as

 \* capturing event handlers).

 \*

 \* In versions of Internet Explorer that support attachEvent( ), Handler.add( )

 \* and Handler.remove() use attachEvent( ) and detachEvent( ). To

 \* invoke the handler function with the correct this keyword, a closure is

 \* used. Since closures of this sort cause memory leaks in Internet Explorer,

 \* Handler.add( ) automatically registers an onunload handler to deregister

 \* all event handlers when the page is unloaded. To keep track of

 \* registered handlers, Handler.add( ) creates a property named \_allHandlers on

 \* the window object and creates a property named \_handlers on any element on

 \* which a handler is registered.

 \*/

var Handler = {};

// In DOM-compliant browsers, our functions are trivial wrappers around

// addEventListener( ) and removeEventListener( ).

if (document.addEventListener) {

    Handler.add = function(element, eventType, handler) {

        element.addEventListener(eventType, handler, false);

    };

    Handler.remove = function(element, eventType, handler) {

        element.removeEventListener(eventType, handler, false);

    };

}

// In IE 5 and later, we use attachEvent( ) and detachEvent( ), with a number of

// hacks to make them compatible with addEventListener and removeEventListener.

else if (document.attachEvent) {

    Handler.add = function(element, eventType, handler) {

        // Don't allow duplicate handler registrations

        // \_find( ) is a private utility function defined below.

        if (Handler.\_find(element, eventType, handler) != -1) return;

        // To invoke the handler function as a method of the

        // element, we've got to define this nested function and register

        // it instead of the handler function itself.

        var wrappedHandler = function(e) {

            if (!e) e = window.event;

            // Create a synthetic event object with partial compatibility

            // with DOM events.

            var event = {

                \_event: e,    // In case we really want the IE event object

                type: e.type,           // Event type

                target: e.srcElement,   // Where the event happened

                currentTarget: element, // Where we're handling it

                relatedTarget: e.fromElement?e.fromElement:e.toElement,

                eventPhase: (e.srcElement==element)?2:3,

                // Mouse coordinates

                clientX: e.clientX, clientY: e.clientY,

                screenX: e.screenX, screenY: e.screenY,

               // Key state

                altKey: e.altKey, ctrlKey: e.ctrlKey,

                shiftKey: e.shiftKey, charCode: e.keyCode,

                // Event-management functions

                stopPropagation: function( ) {this.\_event.cancelBubble = true;},

                preventDefault: function( ) {this.\_event.returnValue = false;}

            }

            // Invoke the handler function as a method of the element, passing

            // the synthetic event object as its single argument.

            // Use Function.call( ) if defined; otherwise do a hack

            if (Function.prototype.call)

                handler.call(element, event);

            else {

                // If we don't have Function.call, fake it like this.

                element.\_currentHandler = handler;

                element.\_currentHandler(event);

                element.\_currentHandler = null;

            }

        };

        // Now register that nested function as our event handler.

        element.attachEvent("on" + eventType, wrappedHandler);

        // Now we must do some record keeping to associate the user-supplied

        // handler function and the nested function that invokes it.

        // We have to do this so that we can deregister the handler with the

        // remove( ) method and also deregister it automatically on page unload.

        // Store all info about this handler into an object.

        var h = {

            element: element,

            eventType: eventType,

            handler: handler,

            wrappedHandler: wrappedHandler

        };

        // Figure out what document this handler is part of.

        // If the element has no "document" property, it is not

        // a window or a document element, so it must be the document

        // object itself.

        var d = element.document || element;

        // Now get the window associated with that document.

        var w = d.parentWindow;

        // We have to associate this handler with the window,

        // so we can remove it when the window is unloaded.

        var id = Handler.\_uid( );  // Generate a unique property name

        if (!w.\_allHandlers) w.\_allHandlers = {};  // Create object if needed

        w.\_allHandlers[id] = h; // Store the handler info in this object

        // And associate the id of the handler info with this element as well.

        if (!element.\_handlers) element.\_handlers = [];

        element.\_handlers.push(id);

        // If there is not an onunload handler associated with the window,

        // register one now.

        if (!w.\_onunloadHandlerRegistered) {

            w.\_onunloadHandlerRegistered = true;

            w.attachEvent("onunload", Handler.\_removeAllHandlers);

        }

    };

    Handler.remove = function(element, eventType, handler) {

        // Find this handler in the element.\_handlers[] array.

        var i = Handler.\_find(element, eventType, handler);

        if (i == -1) return;  // If the handler was not registered, do nothing

        // Get the window of this element.

        var d = element.document || element;

        var w = d.parentWindow;

        // Look up the unique id of this handler.

        var handlerId = element.\_handlers[i];

        // And use that to look up the handler info.

        var h = w.\_allHandlers[handlerId];

        // Using that info, we can detach the handler from the element.

        element.detachEvent("on" + eventType, h.wrappedHandler);

        // Remove one element from the element.\_handlers array.

        element.\_handlers.splice(i, 1);

        // And delete the handler info from the per-window \_allHandlers object.

        delete w.\_allHandlers[handlerId];

    };

    // A utility function to find a handler in the element.\_handlers array

    // Returns an array index or -1 if no matching handler is found

    Handler.\_find = function(element, eventType, handler) {

        var handlers = element.\_handlers;

        if (!handlers) return -1;  // if no handlers registered, nothing found

        // Get the window of this element

        var d = element.document || element;

        var w = d.parentWindow;

        // Loop through the handlers associated with this element, looking

        // for one with the right type and function.

        // We loop backward because the most recently registered handler

        // is most likely to be the first removed one.

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

            var handlerId = handlers[i];        // get handler id

            var h = w.\_allHandlers[handlerId];  // get handler info

            // If handler info matches type and handler function, we found it.

            if (h.eventType == eventType && h.handler == handler)

                return i;

        }

        return -1;  // No match found

    };

    Handler.\_removeAllHandlers = function( ) {

        // This function is registered as the onunload handler with

        // attachEvent. This means that the this keyword refers to the

        // window in which the event occurred.

        var w = this;

        // Iterate through all registered handlers

        for(id in w.\_allHandlers) {

            // Get handler info for this handler id

            var h = w.\_allHandlers[id];

            // Use the info to detach the handler

            h.element.detachEvent("on" + h.eventType, h.wrappedHandler);

            // Delete the handler info from the window

            delete w.\_allHandlers[id];

        }

    }

    // Private utility to generate unique handler ids

    Handler.\_counter = 0;

    Handler.\_uid = function( ) { return "h" + Handler.\_counter++; };

}

5.2.4.11. Key Event事件对象

由于DOM中没有相应的标准,这里的用法是相当混乱的,我将分别介绍一下IE和Firefox系列的模型.

通常,一次按键有3个过程keydown, keypress, 和 keyup.

keypress为输入的ASCII字符,如"@"等.不可打印字符如空格,回车, 退格,方向键,F1到F12,如果没有对应的ASCII码,不产生该事件

keypress,keyup为"virtual keycode" ,如同时按下 "shift"和"2", 发生事件"shift-2"

altKey, ctrlKey, shiftKey属性可以用来查询这三个键的状态(ALT在IE中不产生keypress事件)

为了获得嵌入的字符,Firefox定义了两个属性keyCode(返回"virtual keycode")和charCode(返回字符编码),对于不可打印字符,charCode是0.

IE中只有keyCode属性,为keydown事件时, keyCode是virtual keycode;为keypress事件时, keyCodes是字符编码

String.fromCharCode( )可以把字符编码转换为字符.

<input type="text" class="spellInput" onKeyPress="if(event.keyCode==13 && this.value!=''){this.value='you have enter something'}" />

A Keymap class for keyboard shortcuts

/\*

 \* Keymap.js: bind key events to handler functions.

 \*

 \* This module defines a Keymap class. An instance of this class represents a

 \* mapping of key identifiers (defined below) to handler functions. A

 \* Keymap can be installed on an HTML element to handle keydown and keypress

 \* events. When such an event occurs, the Keymap uses its mapping to invoke

 \* the appropriate handler function.

 \*

 \* When you create a Keymap, pass a JavaScript object that represents the

 \* initial set of bindings for the Keymap. The property names of this object

 \* are key identifers, and the property values are the handler functions.

 \*

 \* After a Keymap has been created, you can add new bindings by passing a key

 \* identifer and handler function to the bind( ) method.

You can remove a

 \* binding by passing a key identifier to the unbind( ) method.

 \*

 \* To make use of a Keymap, call its install( ) method, passing an HTML element,

 \* such as the document object. install( ) adds an onkeypress and onkeydown

 \* event handler to the specified object, replacing any handlers previously set

 \* on those properties. When these handlers are invoked, they determine the

 \* key identifier from the key event and invoke the handler function, if any,

 \* bound to that key identifier. If there is no mapping for the event, it uses

 \* the default handler function (see below), if one is defined. A single

 \* Keymap may be installed on more than one HTML element.

 \*

 \* Key Identifiers

 \*

 \* A key identifier is a case-insensitive string representation of a key plus

 \* any modifier keys that are held down at the same time. The key name is the

 \* name of the key: this is often the text that appears on the physical key of

 \* an English keyboard. Legal key names include "A", "7", "F2", "PageUp",

 \* "Left", "Delete", "/", "~". For printable keys, the key name is simply the

 \* character that the key generates. For nonprinting keys, the names are

 \* derived from the KeyEvent.DOM\_VK\_ constants defined by Firefox. They are

 \* simply the constant name, with the "DOM\_VK\_" portion and any underscores

 \* removed. For example, the KeyEvent constant DOM\_VK\_BACK\_SPACE becomes

 \* BACKSPACE. See the Keymap.keyCodeToFunctionKey object in this module for a

 \* complete list of names.

 \*

 \* A key identifier may also include modifier key prefixes. These prefixes are

 \* Alt\_, Ctrl\_, and Shift\_. They are case-insensitive, but if there is more

 \* than one, they must appear in alphabetical order. Some key identifiers that

 \* include modifiers include "Shift\_A", "ALT\_F2", and "alt\_ctrl\_delete". Note

 \* that "ctrl\_alt\_delete" is not legal because the modifiers are not in

 \* alphabetical order.

 \*

 \* Shifted punctuation characters are normally returned as the appropriate

 \* character. Shift-2 generates a key identifier of "@", for example. But if

 \* Alt or Ctrl is also held down, the unshifted symbol is used instead.

 \* We get a key identifier of Ctrl\_Shift\_2 instead of Ctrl\_@, for example.

 \*

 \* Handler Functions

 \*

 \* When a handler function is invoked, it is passed three arguments:

 \*   1) the HTML element on which the key event occurred

 \*   2) the key identifier of the key that was pressed

 \*   3) the event object for the keydown event

 \*

 \* Default Handler

 \*

 \* The reserved key name "default" may be mapped to a handler function. That

 \* function will be invoked when no other key-specific binding exists.

 \*

 \* Limitations

 \*

 \* It is not possible to bind a handler function to all keys. The operating

 \* system traps some key sequences (Alt-F4, for example). And the browser

 \* itself may trap others (Ctrl-S, for example). This code is browser, OS,

 \* and locale-dependent. Function keys and modified function keys work well,

 \* and unmodified printable keys work well. The combination of Ctrl and Alt

 \* with printable characters, and particularly with punctuation characters, is

 \* less robust.

 \*/

// This is the constructor function

function Keymap(bindings) {

    this.map = {};    // Define the key identifier->handler map

    if (bindings) {   // Copy initial bindings into it, converting to lowercase

        for(name in bindings) this.map[name.toLowerCase( )] = bindings[name];

    }

}

// Bind the specified key identifier to the specified handler function

Keymap.prototype.bind = function(key, func) {

    this.map[key.toLowerCase( )] = func;

};

// Delete the binding for the specified key identifier

Keymap.prototype.unbind = function(key) {

    delete this.map[key.toLowerCase( )];

};

// Install this Keymap on the specified HTML element

Keymap.prototype.install = function(element) {

    // This is the event-handler function

    var keymap = this;

    function handler(event) { return keymap.dispatch(event); }

    // Now install it

    if (element.addEventListener) {

        element.addEventListener("keydown", handler, false);

        element.addEventListener("keypress", handler, false);

    }

    else if (element.attachEvent) {

        element.attachEvent("onkeydown", handler);

        element.attachEvent("onkeypress", handler);

    }

    else {

        element.onkeydown = element.onkeypress = handler;

    }

};

// This object maps keyCode values to key names for common nonprinting

// function keys. IE and Firefox use mostly compatible keycodes for these.

// Note, however that these keycodes may be device-dependent and different

// keyboard layouts may have different values.

Keymap.keyCodeToFunctionKey = {

    8:"backspace", 9:"tab", 13:"return", 19:"pause", 27:"escape", 32:"space",

    33:"pageup", 34:"pagedown", 35:"end", 36:"home", 37:"left", 38:"up",

    39:"right", 40:"down", 44:"printscreen", 45:"insert", 46:"delete",

    112:"f1", 113:"f2", 114:"f3", 115:"f4", 116:"f5", 117:"f6", 118:"f7",

    119:"f8", 120:"f9", 121:"f10", 122:"f11", 123:"f12",

    144:"numlock", 145:"scrolllock"

};

// This object maps keydown keycode values to key names for printable

// characters. Alphanumeric characters have their ASCII code, but

// punctuation characters do not. Note that this may be locale-dependent

// and may not work correctly on international keyboards.

Keymap.keyCodeToPrintableChar = {

    48:"0", 49:"1", 50:"2", 51:"3", 52:"4", 53:"5", 54:"6", 55:"7", 56:"8",

    57:"9", 59:";", 61:"=", 65:"a", 66:"b", 67:"c", 68:"d",

    69:"e", 70:"f", 71:"g", 72:"h", 73:"i", 74:"j", 75:"k", 76:"l", 77:"m",

    78:"n", 79:"o", 80:"p", 81:"q", 82:"r", 83:"s", 84:"t", 85:"u", 86:"v",

    87:"w", 88:"x", 89:"y", 90:"z", 107:"+", 109:"-", 110:".", 188:",",

    190:".", 191:"/", 192:"'", 219:"[", 220:"//", 221:"]", 222:"/""

};

// This method dispatches key events based on the keymap bindings.

Keymap.prototype.dispatch = function(event) {

    var e = event || window.event;  // Handle IE event model

    // We start off with no modifiers and no key name

    var modifiers = ""

    var keyname = null;

    if (e.type == "keydown") {

        var code = e.keyCode;

        // Ignore keydown events for Shift, Ctrl, and Alt

        if (code == 16 || code == 17 || code == 18) return;

        // Get the key name from our mapping

        keyname = Keymap.keyCodeToFunctionKey[code];

        // If this wasn't a function key, but the ctrl or alt modifiers are

        // down, we want to treat it like a function key

        if (!keyname && (e.altKey || e.ctrlKey))

            keyname = Keymap.keyCodeToPrintableChar[code];

        // If we found a name for this key, figure out its modifiers.

        // Otherwise just return and ignore this keydown event.

        if (keyname) {

            if (e.altKey) modifiers += "alt\_";

            if (e.ctrlKey) modifiers += "ctrl\_";

            if (e.shiftKey) modifiers += "shift\_";

        }

        else return;

    }

    else if (e.type == "keypress") {

        // If ctrl or alt are down, we've already handled it.

        if (e.altKey || e.ctrlKey) return;

        // In Firefox we get keypress events even for nonprinting keys.

        // In this case, just return and pretend it didn't happen.

        if (e.charCode != undefined && e.charCode == 0) return;

        // Firefox gives us printing keys in e.charCode, IE in e.charCode

        var code = e.charCode || e.keyCode;

        // The code is an ASCII code, so just convert to a string.

        keyname=String.fromCharCode(code);

        // If the key name is uppercase, convert to lower and add shift

        // We do it this way to handle CAPS LOCK; it sends capital letters

        // without having the shift modifier set.

        var lowercase = keyname.toLowerCase( );

        if (keyname != lowercase) {

            keyname = lowercase;    // Use the lowercase form of the name

            modifiers = "shift\_";   // and add the shift modifier.

        }

    }

    // Now that we've determined the modifiers and key name, we look for

    // a handler function for the key and modifier combination

    var func = this.map[modifiers+keyname];

    // If we didn't find one, use the default handler, if it exists

    if (!func) func = this.map["default"];

    if (func) {  // If there is a handler for this key, handle it

        // Figure out what element the event occurred on

        var target = e.target;              // DOM standard event model

        if (!target) target = e.srcElement; // IE event model

        // Invoke the handler function

        func(target, modifiers+keyname, e);

        // Stop the event from propagating, and prevent the default action for

        // the event. Note that preventDefault doesn't usually prevent

        // top-level browser commands like F1 for help.

        if (e.stopPropagation) e.stopPropagation( );  // DOM model

        else e.cancelBubble = true;                  // IE model

        if (e.preventDefault) e.preventDefault( );    // DOM

        else e.returnValue = false;                  // IE

        return false;                                // Legacy event model

    }

};

/\*\*

 \* InputFilter.js: unobtrusive filtering of keystrokes for <input> tags

 \*

 \* This module finds all <input type="text"> elements in the document that

 \* have a nonstandard attribute named "allowed". It registers an onkeypress

 \* event handler for any such element to restrict the user's input so that

 \* only characters that appear in the value of the allowed attribute may be

 \* entered. If the <input> element also has an attribute named "messageid",

 \* the value of that attribute is taken to be the id of another document

 \* element. If the user types a character that is not allowed, the messageid

 \* element is made visible. If the user types a character that is allowed,

 \* the messageid element is hidden. This message id element is intended to

 \* offer an explanation to the user of why her keystroke was rejected. It

 \* should typically be styled with CSS so that it is initially invisible.

 \*

 \* Here is some sample HTML that uses this module.

 \*   Zipcode:

 \*   <input id="zip" type="text" allowed="0123456789" messageid="zipwarn">

 \*   <span id="zipwarn" style="color:red;visibility:hidden">Digits only</SPAN>  \*

 \* In browsers such as IE, which do not support addEventListener( ), the

 \* keypress handler registered by this module overwrites any keypress handler

 \* defined in HTML.

 \*

 \* This module is purely unobtrusive: it does not define any symbols in

 \* the global namespace.

 \*/

(function( ) {  // The entire module is within an anonymous function

    // When the document finishes loading, call the init( ) function below

    if (window.addEventListener) window.addEventListener("load", init, false);

    else if (window.attachEvent) window.attachEvent("onload", init);

    // Find all the <input> tags we need to register an event handler on

    function init( ) {

        var inputtags = document.getElementsByTagName("input");

        for(var i = 0 ; i < inputtags.length; i++) { // Loop through all tags

            var tag = inputtags[i];

            if (tag.type != "text") continue; // We only want text fields

            var allowed = tag.getAttribute("allowed");

            if (!allowed) continue;  // And only if they have an allowed attr

            // Register our event handler function on this input tag

            if (tag.addEventListener)

                tag.addEventListener("keypress", filter, false);

            else {

                // We don't use attachEvent because it does not invoke the

                // handler function with the correct value of the this keyword.

                tag.onkeypress = filter;

            }

        }

    }

    // This is the keypress handler that filters the user's input

    function filter(event) {

        // Get the event object and character code in a portable way

        var e = event || window.event;         // Key event object

        var code = e.charCode || e.keyCode;    // What key was pressed

        // If this keystroke is a function key of any kind, do not filter it

        if (e.charCode == 0) return true;       // Function key (Firefox only)

        if (e.ctrlKey || e.altKey) return true; // Ctrl or Alt held down

        if (code < 32) return true;             // ASCII control character

        // Now look up information we need from this input element

        var allowed = this.getAttribute("allowed");     // Legal chars

        var messageElement = null;                      // Message to hide/show

        var messageid = this.getAttribute("messageid"); // Message id, if any

        if (messageid)  // If there is a message id, get the element

            messageElement = document.getElementById(messageid);

        // Convert the character code to a character

        var c = String.fromCharCode(code);

        // See if the character is in the set of allowed characters

        if (allowed.indexOf(c) != -1) {

            // If c is a legal character, hide the message, if any

            if (messageElement) messageElement.style.visibility = "hidden";

            return true; // And accept the character

        }

        else {

            // If c is not in the set of allowed characters, display message

            if (messageElement) messageElement.style.visibility = "visible";

            // And reject this keypress event

            if (e.preventDefault) e.preventDefault( );

            if (e.returnValue) e.returnValue = false;

            return false;

        }

    }

})( ); // Finish anonymous function and invoke it.

5.2.4.12. 加载事件 onload Event

onload回在文档完全加载完成后运行

一个跨平台的版本

function runOnLoad(f) {

    if (runOnLoad.loaded) f( );    // If already loaded, just invoke f( ) now.

    else runOnLoad.funcs.push(f); // Otherwise, store it for later

}

runOnLoad.funcs = []; // The array of functions to call when the document loads

runOnLoad.loaded = false; // The functions have not been run yet.

// Run all registered functions in the order in which they were registered.

// It is safe to call runOnLoad.run( ) more than once: invocations after the

// first do nothing. It is safe for an initialization function to call

// runOnLoad( ) to register another function.

runOnLoad.run = function( ) {

    if (runOnLoad.loaded) return;  // If we've already run, do nothing

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

        try { runOnLoad.funcs[i]( ); }

        catch(e) { /\* An exception in one function shouldn't stop the rest \*/ }

    }

    runOnLoad.loaded = true; // Remember that we've already run once.

    delete runOnLoad.funcs;  // But don't remember the functions themselves.

    delete runOnLoad.run;    // And forget about this function too!

};

// Register runOnLoad.run( ) as the onload event handler

 for the window

if (window.addEventListener)

    window.addEventListener("load", runOnLoad.run, false);

else if (window.attachEvent) window.attachEvent("onload", runOnLoad.run);

else window.onload = runOnLoad.run;

5.2.4.13. 合成事件 Synthetic Events

5.2.5. cookie

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

6. JS库:prototype

6.1. $()等价getElementById()

$() 方法是在DOM中使用过于频繁的 document.getElementById() 方法的简写，这个方法返回参数传入的id的那个元素。

你可以传入多个id作为参数然后 $() 返回一个带有所有要求的元素的一个 Array 对象。

    function test2()

    {

        var divs = $('myDiv','myOtherDiv');

        for(i=0; i<divs.length; i++)

        {

            alert(divs[i].innerHTML);

        }

    }

</script>

<div id="myDiv">

        <p>This is a paragraph</p>

</div>

<div id="myOtherDiv">

        <p>This is another paragraph</p>

</div>

<input type="button" value=Test2 onclick="test2();">

6.2. $F()返回输入控件的值

<script>

    function test3()

    {

        alert(  $F('userName')  );

    }

</script>

<input type="text" id="userName" value="Joe Doe"><br>

<input type="button" value=Test3 onclick="test3();"><br>

6.3. $A()参数转Array

这个方法加上对Array类的扩展，可以很容易的转换或者复制任意的列举列表到Array对象， 一个被推荐使用的用法就是转换DOM的NodeLists到一个普通的数组里，可以被更广泛高效的使用， 看下面的例子。

<script>

    function showOptions(){

        var someNodeList = $('lstEmployees').getElementsByTagName('option');

        var nodes = $A(someNodeList);

        nodes.each(

        function(node){

        alert(node.nodeName + ': ' + node.innerHTML);

        }

        );

    }

</script>

<select id="lstEmployees" size="10" >

    <option value="5">Buchanan, Steven</option>

    <option value="8">Callahan, Laura</option>

    <option value="1">Davolio, Nancy</option>

</select>

<input type="button" value="Show the options" onclick="showOptions();"

6.4. $H()转类似联合数组的Hash对象

$H()方法把对象转化成可枚举的貌似联合数组Hash对象。

<script>

    function testHash()

    {

        //let's create the object

        var a = {

            first: 10,

            second: 20,

            third: 30

            };

        //now transform it into a hash

        var h = $H(a);

        alert(h.toQueryString()); //displays: first=10&second=20&third=30

    }

</script>

==$R()简单的遍历 == 用方法$R()方法是new ObjectRange(lowerBound, upperBound, excludeBounds)的一个简单快捷的使用方式。

<script>

    function demoDollar\_R(){

        var range = $R(10, 20, false);

        range.each(function(value, index){

            alert(value);

        });

    }

</script>

<input type="button" value="Sample Count" onclick="demoDollar\_R();" >

6.5. Try.these()尝试直到一个成功

方法Try.these() 方法使得实现当你想调用不同的方法直到其中的一个成功正常的这种需求变得非常容易， 他把一系列的方法作为参数并且按顺序的一个一个的执行这些方法直到其中的一个成功执行，返回成功执行的那个方法的返回值。

在下面的例子中， xmlNode.text在一些浏览器中好用，但是xmlNode.textContent在另一些浏览器中正常工作。 使用Try.these()方法我们可以得到正常工作的那个方法的返回值。

<script>

function getXmlNodeValue(xmlNode){

    return Try.these(

        function() {return xmlNode.text;},

        function() {return xmlNode.textContent;)

        );

}

</script>

== Ajax.RequestXML响应==

如果你不使用任何的帮助程序包，你很可能编写了整个大量的代码来创建XMLHttpRequest对象并且异步的跟踪它的进程，然后解析出响应 然后处理它。当你不需要支持多于一种类型的浏览器时你会感到非常的幸运。

为了支持 AJAX 功能。这个包定义了 Ajax.Request 类。

假如你有一个应用程序可以通过url http://yoursever/app/get\_sales?empID=1234&year=1998与服务器通信。它返回下面这样的XML响应。

<?xml version="1.0" encoding="utf-8" ?>

<ajax-response>

<response type="object" id="productDetails">

<monthly-sales>

<employee-sales>

<employee-id>1234</employee-id>

<year-month>1998-01</year-month>

<sales>$8,115.36</sales>

</employee-sales>

<employee-sales>

<employee-id>1234</employee-id>

<year-month>1998-02</year-month>

<sales>$11,147.51</sales>

</employee-sales>

</monthly-sales>

</response>

</ajax-response>

用 Ajax.Request对象和服务器通信并且得到这段XML是非常简单的。下面的例子演示了它是如何完成的。

<script>

function searchSales()

{

var empID = $F('lstEmployees');

var y = $F('lstYears');

var url = 'http://yoursever/app/get\_sales';

var pars = 'empID=' + empID + '&year=' + y;

var myAjax = new Ajax.Request( url, { method: 'get', parameters: pars, onComplete: showResponse });

}

function showResponse(originalRequest)

{

//put returned XML in the textarea

$('result').value = originalRequest.responseText;

}

</script>

<select id="lstEmployees" size="10" onchange="searchSales()">

<option value="5">Buchanan, Steven</option>

<option value="8">Callahan, Laura</option>

<option value="1">Davolio, Nancy</option>

</select>

<select id="lstYears" size="3" onchange="searchSales()">

<option selected="selected" value="1996">1996</option>

<option value="1997">1997</option>

<option value="1998">1998</option>

</select>

<br><textarea id=result cols=60 rows=10 ></textarea>

你注意到传入 Ajax.Request构造方法的第二个对象了吗？ 参数{method: 'get', parameters: pars, onComplete: showResponse} 表示一个匿名对象的真实写法。他表示你传入的这个对象有一个名为 method 值为 'get'的属性，另一个属性名为 parameters 包含HTTP请求的查询字符串，和一个onComplete 属性/方法包含函数showResponse。

还有一些其它的属性可以在这个对象里面定义和设置，如 asynchronous，可以为true 或 false 来决定AJAX对服务器的调用是否是异步的（默认值是 true）。

这个参数定义AJAX调用的选项。在我们的例子中，在第一个参数通过HTTP GET命令请求那个url，传入了变量 pars包含的查询字符串， Ajax.Request 对象在它完成接收响应的时候将调用showResponse 方法。

也许你知道， XMLHttpRequest在HTTP请求期间将报告进度情况。这个进度被描述为四个不同阶段：Loading, Loaded, Interactive, 或 Complete。你可以使 Ajax.Request 对象在任何阶段调用自定义方法 ，Complete 是最常用的一个。想调用自定义的方法只需要简单的在请求的选项参数中的名为 onXXXXX 属性/方法中提供自定义的方法对象。 就像我们例子中的 onComplete 。你传入的方法将会被用一个参数调用，这个参数是 XMLHttpRequest 对象自己。你将会用这个对象去得到返回的数据并且或许检查包含有在这次调用中的HTTP结果代码的 status 属性。

还有另外两个有用的选项用来处理结果。我们可以在onSuccess 选项处传入一个方法，当AJAX无误的执行完后调用， 相反的，也可以在onFailure选项处传入一个方法，当服务器端出现错误时调用。正如onXXXXX 选项传入的方法一样，这两个在被调用的时候也传入一个带有AJAX请求的XMLHttpRequest对象。

我们的例子没有用任何有趣的方式处理这个 XML响应， 我们只是把这段XML放进了一个文本域里面。对这个响应的一个典型的应用很可能就是找到其中的想要的信息，然后更新页面中的某些元素， 或者甚至可能做某些XSLT转换而在页面中产生一些HTML。

在1.4.0版本中，一种新的事件回传外理被引入。如果你有一段代码总是要为一个特殊的事件执行，而不管是哪个AJAX调用引发它，那么你可以使用新的Ajax.Responders对象。

假设你想要在一个AJAX调用正在运行时，显示一些提示效果，像一个不断转动的图标之类的，你可以使用两个全局事件Handler来做到，其中一个在第一个调用开始时显示图标，另一个在最后一个调用完成时隐藏图标。看下面的例子。

<script>

var myGlobalHandlers = {

onCreate: function(){

Element.show('systemWorking');

},

onComplete: function() {

if(Ajax.activeRequestCount == 0){

Element.hide('systemWorking');

}

}

};

Ajax.Responders.register(myGlobalHandlers);

</script>

<div id='systemWorking'><img src='spinner.gif'>Loading...</div>

更完全的解释，请参照 Ajax.Request 参考 和 Ajax选项参考。

MoinMoin PoweredPython PoweredValid HTML 4.01

推荐使用支持W3C标准的浏览器进行阅读，获得最佳页面效果，谢绝IE的混乱解析！   Searching all CPUG sites and mailist::

 Web

getACL = 0.006s getPageList = 0.204s loadLanguage = 0.005s load\_multi\_cfg = 0.003s run = 0.364s send\_page = 0.112s send\_page\_content = 0.052s total = 0.378s