

Javascript

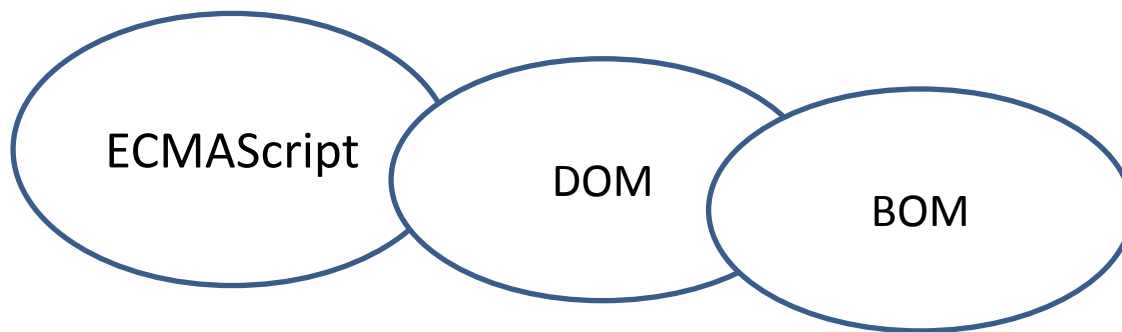
2015.9.20

isszym sysu.edu.cn

概述

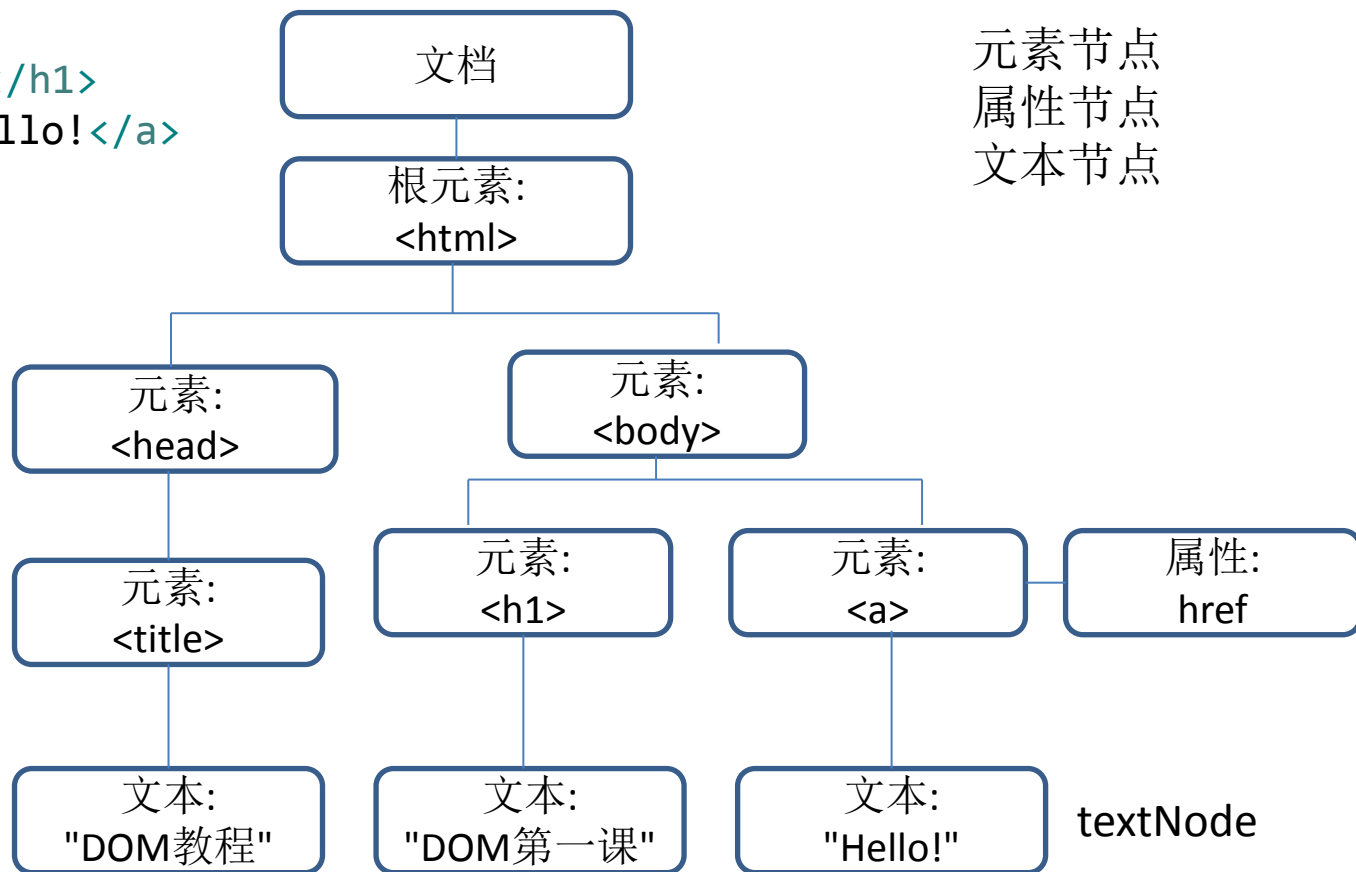
- 1995 年，Netscape Navigator 2.0 增加了一个由 Netscape 与 Sun 开发的称为 LiveScript 的脚本语言，当时的目的是在浏览器和服务端使用它。在 Netscape Navigator 2.0 即将正式发布前，Netscape 将其更名为 JavaScript，目的是为了利用 Java 这个因特网时髦词汇。
- 因为 JavaScript 1.0 如此成功，Netscape 在 Netscape Navigator 3.0 中发布了 1.1 版。此时微软发布了 IE 3.0 并搭载了一个 JavaScript 的克隆版，叫做 Jscript。这样命名是为了避免与 Netscape 产生潜在的许可纠纷。在微软进入后，有 3 种不同的 JavaScript 版本同时存在：Netscape Navigator 3.0 中的 **JavaScript**、IE 中的 **JScript** 以及 CEnv 中的 **ScriptEase**。
- 与 C 和其他编程语言不同的是，JavaScript 并没有一个标准来统一其语法或特性，而这 3 种不同的版本恰恰突出了这个问题。随着业界担心的增加，这个语言标准化显然已经势在必行。

- 1997 年，JavaScript 1.1 作为一个草案提交给欧洲计算机制造商协会（ECMA）。第 39 技术委员会（TC39）被委派来“标准化一个通用、跨平台、中立于厂商的脚本语言的语法和语义”(http://www.ecma-international.org/memento/TC39.htm)。由来自 Netscape、Sun、微软、Borland 和其他一些对脚本编程感兴趣的公司的程序员组成的 TC39 锤炼出了 ECMA-262，该标准定义了名为 **ECMAScript** 的全新脚本语言。
- 在接下来的几年里，国际标准化组织及国际电工委员会（ISO/IEC）也采纳 ECMAScript 作为标准（ISO/IEC-16262）。从此，Web 浏览器就开始努力（虽然有着不同的程度的成功和失败）将 ECMAScript 作为 JavaScript 实现的基础。
- 一个完整的 **JavaScript** 实现是由3 个部分组成的：**ECMAScript**，**DOM**，**BOM**。
ECMAScript 描述了该语言的基本语法
DOM (Document Object Model)以对象方式描述文档模型
BOM(Browser Object Model) 描述了与浏览器进行交互的方法和接口



文档树 (DOM节点树)

```
<html>
<head>
<title>DOM 教程</title>
</head>
<body>
  <h1>DOM 第一课</h1>
  <a href="#">Hello!</a>
</body>
</html>
```



元素节点
属性节点
文本节点

用JavaScript语言来修改元素（节点）的内容和属性

```
<html>
<head>
<title>DOM 教程</title>
<script type="text/javascript">
    function replace(){
        var x=document.getElementById("a1");
        x.innerHTML="DOM 第二课";
    }
</script>
</head>
<body>
    <h1 id="a1">DOM 第一课</h1>
    <p onclick="replace()">Hello!</p>
</body>
</html>
```

first.html

点击后会替换元素h1的内容

<script>可以定义多个，而且放在html的任何地方。它是作为一个整体，自上往下执行，上面定义的变量，下面依然可以可以使用。也可以使用外部文件的方法：

```
<script type="text/javascript" src="js1.js"></script>
```

把元素<script>的内容放在文件里面

不能有内容

如果替换replace函数为如下内容(ajax), 更可以不刷新页面而从网站获取一个页面替换<h1>的内容。

```
function replace(){
    var url="http://202.116.76.22:8080/html/index.jsp";
    var ctrlID="a1";
    var param=null;
    var xmlhttp = new XMLHttpRequest();

    xmlhttp.onreadystatechange = function () {
        if (xmlhttp.readyState == 4)    //读取服务器响应结束
        {
            if (xmlhttp.status >= 200 && xmlhttp.status < 300
                || xmlhttp.status >= 304) {
                alert(xmlhttp.responseText);
                var obj = document.getElementById(ctrlID)
                obj.innerHTML = xmlhttp.responseText;
            }
            else {
                alert("Request was unsuccessful:" + xmlhttp.status);
            }
        }
    }
    xmlhttp.open("get", url, true);
    xmlhttp.send(param);
}
```

JavaScript变量

- 定义

- ✓ **JavaScript**变量可以不经定义直接使用。为了确定生命期，最好对变量进行定义。
- ✓ 变量名以字母、下划线(_)和美元符号(\$)开头，其它部分还可以加上数字。变量名区分大小写。

```
<script type="text/javascript">  
    var x;                // 定义  
    var y=3;              // 定义并初始化  
    var a=3.5, b="123456"; // 一次定义多个变量  
    b=5;                  // 改变值和类型(不推荐)  
</script>
```

● 变量的基本类型

数字型： 整数和小数。070(8进制), 0xFF, 100, 0.345

布尔型： true 或 false。true转化为整数1， false为0

字符串： 引用类型。可以使用单引号或双引号。

数组： 属于引用类型

typeof

```
var i=10;           //typeof i ="number"  
var s="abcde";      //typeof s ="string"  
var b=false;        //typeof b ="boolean"  
var o={};           //typeof o ="object" 。还有数组、null  
                    //typeof x  ="undefined"
```


● 变量的特殊取值和类型转换

| | |
|------------|--|
| undefined | 未初始化变量的取值 |
| null | 未初始化对象的取值 |
| N/A | Not Applicable |
| NaN | not a number。除以0、非数值字符串等的取值 |
| false | 0、空串、null、undefined、NaN |
| true | 其它 |
| isNaN() | 判断参数是否为数值 |
| isFinite() | 数值在Number.Min_VALUE~Number.Max_VALUE之间 |

```
var a=3;
var b="5";
var c1=a+b;           //35
var c2=b+a;           //53
b=4;                  //4.转换了类型，不提倡
var d=a+parseInt(b);  //7, 把字符串转化为整数
var e=a+(b-0);        //7
var f=parseFloat('a'); //NaN
var g=Number('abc');   //NaN
document.write(c1+" "+c2+" "+d+" "+e+" "+f+" "+g)
```

TypeTrans.html

运算符与表达式

➤算术表达式：用算术运算符形成的表达式，计算结果为整值

```
var x = 5,y = 6,z = 10;    // 一次定义多个变量，并赋初值
var exp = (y + 3) * z + x;  // 右边为算术表达式，exp得值95
```

➤关系表达式：用关系运算符形成的表达式，计算结果为真假值

```
var x = 300;
var r1 = x>10;    // x是否大于10。r1得值true
var r2 = x<=100;  // x是否小于等于100。r2得值false
```

➤逻辑表达式：用逻辑运算符形成的表达式，计算结果为真假值。

```
var y = 99;
var r3 = (y>10)&&(y<100); // y是否大于10并且小于100。 true
```

➤位表达式：按位运算的表达式，先转换整数(32位)再运算。

```
var z = 16;
var r4 = z | 0x1E0F; // 按位或。 0x1E1F (7711)
```

➤表达式计算

```
var x=20, y = 100.2;
document.write(eval("x+y")); //120.2. eval中可以写入语句
```

算术运算符: $a+b$ $a-b$ $a*b$ a/b (商) $a\%b$ (余数) i/j (整除,i和j为整数)
 关系运算符: $a>b$ $a<b$ $a>=b$ $a<=b$ $a==b$ (等于) $a===b$ (恒等) $a!=b$ (不等于)
 逻辑运算符: $a\&\&b$ (短路与) $a||b$ (短路或) $!a$ (非)
 位运算: $\sim a$ (按位非) $a\&b$ (按位与) $a|b$ (按位或) a^b (按位异或)
 移位运算: $b<<1$ (左移1位) $a>>2$ (带符号右移2位) $b>>>3$ (无符号右移3位)
 三目运算: $x<3?10:7$ (如果x小于3, 则取值10, 否则, 取值7)
 单目运算: $++x$ (x先加1, 再参与运算) $--x$ $x++$ $x--$ $-x$ (变符号)
 赋值运算: $x+=a$ ($x=x+a$) $x-=a$ $x*=a$ $x/=a$ $x\%=a$ $x\&=a$
 $x|=a$ $x\&=a$ $x|=a$ $x^a=a$ $x>>=a$ $x>>>=a$ $x<<=a$

运算优先级:

高 $[]()$ \rightarrow 单目 $\rightarrow * / \% \rightarrow + - \rightarrow << >> >>>$
 $\rightarrow < > <= >= \rightarrow == != \rightarrow \& \rightarrow ^$
 $\rightarrow | \rightarrow \&\& \rightarrow || \rightarrow$ 三目运算 \rightarrow 复杂赋值 低

* $==$ 会自动转换类型再判断值是否相等, $===$ 不会自动转换类型 (只要有null, undefined, NaN, 返回false)

基本语句

Statements.html

- 赋值语句

```
var    x;           //变量定义
x = 20;           //赋值语句，左边为变量，右边为表达式
var y = 100.2;
x=y;
z=y=30           //单行语句的结束处可以不加；
alert(x+" "+y+" "+z); //100.2 30 30
```

- 注释语句

```
//      注释一行
/* ..... */  注释若干行
```

• 分支控制语句

```
var bkcolor;
var item= "table";
if(item="table"){
    bkcolor="white";
}
else{
    bkcolor="black";
} //bkcolor="white"
var age = 20;
var state;
if(age<1)
    state="婴儿";
else if(age>=1 && age<10)
    state="儿童";
else if(age>=10 && age<18)
    state="少年";
else if(age>=18 && age<45)
    state="青年";
else
    state="中年或老年"; //state="青年"
```

*条件嵌套: else属于最靠近它的if

```
var cnt = 10;
var x;
switch(cnt){
    case 1: x=5.0;break;
    case 12: x=30.0;break;
    default: x=100.0;
} //x=100.0
```

• 循环控制语句

```
var sum=0;
for(var i=0;i<=100;i++){
    sum+=i;
} //sum=5050
```

```
sum=0;
cnt = 0;
var scores=[100.0, 90.2, 80.0, 78.0,93.5];
for(var score in scores){
    sum+=score;
    cnt++;
}
avg = sum/cnt; // avg=246.8
```

```
sum=0;
var k=0;
while(k<=10){
    sum=sum+k;
    k++;
} //55
```

```
sum=0;
k=0;
do{
    sum=sum+k;
    k++;
}while(k<=20);    // 210
```

```
/* 求距阵之和
 * 1 2 3 ... 10
 * 1 2 3 ... 10
 * .....
 * 1 2 3 ... 10
 */
```

```
sum=0;
for(var i=1;i<=10;i++){

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

        sum=sum+j;
    }
}    // sum=550
```



```

sum=0;
Label1:
for(var i=1;i<=10;i++){
    for(var j=1;j<=10;j++){
        if(j==i){
            continue; //跳到for结束处继续执行
        }
        sum=sum+j; //除去对角线的矩阵之和
    }             ← continue跳到这里
} //sum=495       ← continue Label1跳到这里

```

```

sum=0;
Label2:
for(var i=1;i<=10;i++){
    for(var j=1;j<=10;j++){
        if(j==i){
            break; //跳出for循环继续执行
        }
        sum=sum+j; //下三角加对角线矩阵之和
    }
} //sum=165       ← break跳到这里
                    ← break Label2跳到这里

```

字符串

字符串是 JavaScript 的一种基本的数据类型。"abc"就是一个字符串类型的常量。JavaScript 的字符串是不可变的(immutable)，String 类定义的方法返回的是全新的字符串，而原始字符串没有改动。**==可以直接判断内容是否相等。**

下面是常用的字符串函数：

//concat将两个或多个字符的文本组合起来

```
var a = "hello";
```

```
var b = ",world";
```

```
var c = a.concat(b); //"hello,world".与c=a+b结果相同
```

// indexOf返回字符串中一个子串第一处

// 出现的索引（从0开始）。如果没有

// 匹配项，则返回 -1 。lastIndexOf从后往前搜索

```
var index1 = a.indexOf("l");//2
```

```
var index2 = a.indexOf("l",3);//3
```

//charAt返回指定位置的字符。

```
var get_char = a.charAt(0); //"h"
```

```
//var a = "hello";  
//var b = ",world";  
//match检查一个字符串匹配一个正则表达式内容，如果不匹配则返回 null。  
var re = new RegExp(/^he/);  
var is_alpha1 = a.match(re); // "he"  
var is_alpha2 = b.match(re); // null  
  
//substring返回字符串的一个子串，传入参数是起始位置和结束位置。  
var sub_string1 = a.substring(1); // "ello"  
var sub_string2 = a.substring(1,4); // "ell"  
  
//substr返回字符串的一个子串，传入参数是起始位置和长度  
var sub_string3 = a.substr(1); // "ello"  
var sub_string4 = a.substr(1,4); // "ello"  
  
//replace把匹配正则表达式的字符串替换为新配的字符串。  
var result1 = a.replace(re, "Hello"); // "Helloello"  
var result2 = b.replace(re, "Hello"); // ",world"  
  
//search查找正则表达式，如果成功，返回匹配的索引值，否则返回 -1  
var index1 = a.search(re); // 0  
var index2 = b.search(re); // -1
```

```
//var a = "hello";  
//var b = ",world";  
//split将一个字符串做成一个字符串数组。join把数组变为字符串。  
var arr1 = a.split(""); // [h,e,l,l,o],可以指定间隔符，例如，","  
var s2=arr1.join(","); //返回Hello
```

```
//length返回字符串的长度，即其包含的字符的个数。  
len = a.length; // 5
```

```
//toLowerCase和将整个字符串转成小写字母和大写字母。  
var lower_string = a.toLowerCase();//"hello"  
var upper_string = a.toUpperCase();// "HELLO"
```

```
//parseInt把字符串转化为数值。数值转化为字符串:""+number  
int1=parseInt("1234blue"); // 1234  
int2=parseInt("0xA"); // 10  
int3=parseInt("22.5") // 22  
int4=parseInt("blue") // NaN
```

```
//返回链接字符串  
"Free Web Tutorials!".link("http://www.w3school.com.cn");
```

<http://blog.csdn.net/cyai/article/details/4213956>

函数

- 定义

在javascript中函数都是看成对象。每个函数都是Function类型的实例。函数名就是指向函数对象的一个指针。

```
function sum(num1,num2){  
    return num1+num2;  
}  
var sum = function(num1,num2){  
    return num1+num2;  
}
```

• 引用

例1

```
function sum(num1,num2){  
    return num1+num2;  
}  
alert(sum(10,10));  
var asum = sum;  
alert(asum(10,10));  
sum = null;  
alert(asum(10,10));  
asum=function(num1,num2){  
    return num1-num2;  
}  
alert(asum(10,10));
```

// 返回 20

// 函数为对象，可以直接赋值

// 返回20

// 清除对象sum

// 返回20。asum依然可用

// 再赋值一个函数给asum

//返回0 。 没有重载，直接覆盖

例2

情形1：调用正确（函数定义是全局的，可以在定义前引用）

```
alert(sum(10,10));           // 20
function sum(num1,num2){
    return num1+num2;
}
```

情形2：调用错误(sum作为函数对象，必须先定义再引用)

```
alert(sum(10,10));           // 出错
sum= function(num1,num2){
    return num1+num2;
}
```

例3

```
alert(sum(inc1,10));  
function inc1(num1){  
    if(num1<=1)  
        return 1;  
    return num1 * inc1(num1-1);  
}
```

```
// 函数对象可以作为参数  
function sum(inc,num2){  
    return inc(num2)+num2;  
}
```


例4

```
var sayHi =  
    new Function("sName", "sMessage",  
                 "alert(\"Hello \" + sName + sMessage);");  
var sayHello = sayHi;  
sayHello("Zhang", "come here!");
```



The diagram consists of two blue arrows. One arrow originates from the text '参数名' (Parameter Name) and points to the parameter 'sName' in the Function constructor. The second arrow originates from the text '函数体' (Function Body) and points to the function body string 'alert(\"Hello \" + sName + sMessage);'.

函数体

例5

```
fucnton add()                                // 可变参数
{
    var c=0;
    for (var i=0; i<arguments.length;i++){
        var c=parseInt(arguments[i]) +c ;
    }
    return c;
}
document.write("<p> no param=" + add() + "</p>");
document.write("<p> four param=" + add(1,2,3,4) + "</p>");
```

闭包

- 闭包(closure), 指的是词法表示包括不被计算的变量的函数, 也就是说, 函数可以使用函数之外定义的变量。

```
var sMessage = "hello world";  
function sayHelloWorld() {  
    alert(sMessage);  
}  
sayHelloWorld();
```

- 在一个函数中定义另一个函数会使闭包变得更加复杂。

```
function addNum(iBaseNum) {  
    return function(iNum1, iNum2) {  
        return iNum1 + iNum2 + iBaseNum;  
    }  
}  
var add1 = addNum(100);    // 保存执行addNum时的环境。  
alert(add1(10,20));
```

对象

- 对象定义

```
var person = new object();    // 也可以 var person ={};  
person.name = "Nicholas";  
person.age = 26;  
person.print = function(){alert(person.name)};
```

或

```
var person = {    name: "Nicholas",  
                age: 26,  
                print: function(){alert(person.name)};  
};
```

或

```
var person = { "name": "Nicholas",  
               "age": 26,  
               5: true  
};
```

还可以是对象



- 对象访问

```
alert(person["name"]);  
alert(person.name);  
display({ name:"Nicholas", age:26}); //直接把对象作为参数  
function display(args){  
    if(typeof args.name=="string") alert("1");  
    if(typeof args.age=="number") alert("2");  
}
```

• Json的数据格式

```
var people={
  "programmers": [{
    "firstName": "Brett",
    "lastName": "McLaughlin",
    "email": "aaaa"
  }, {
    "firstName": "Jason",
    "lastName": "Hunter",
    "email": "bbbb"
  }, {
    "firstName": "Elliotte",
    "lastName": "Harold",
    "email": "cccc"
  }],
  "musicians": [{
    "firstName": "Eric",
    "lastName": "Clapton",
    "instrument": "guitar"
  }, {
    "firstName": "Sergei",
    "lastName": "Rachmaninoff",
    "instrument": "piano"
  }],
  "authors": [{
    "firstName": "Isaac",
    "lastName": "Asimov",
    "genre": "sciencefiction"
  }, {
    "firstName": "Tad",
    "lastName": "Williams",
    "genre": "fantasy"
  }, {
    "firstName": "Frank",
    "lastName": "Peretti",
    "genre": "christianfiction"
  }]
}

alert(people.authors[1].genre);           ?    // Value is "fantasy"
alert(people.musicians[1].lastName);      // Value is "Rachmaninoff"
alert(people.programmers[2].firstName);   // Value is "Elliotte"
```

原型

- 工厂模式

```
function createPerson(name,age,job) {  
    var o=new Object();  
    o.name=name;  
    o.age=age;  
    o.job=job;  
    return o;  
}  
var person1=createPerson("Nicholas",29,"Software Engineer");  
var person2=createPerson("Greg",27,"Doctor");  
alert(person1.name);
```

工厂模式可以用于产生对象。

● 构造函数

```
function Person(name,age,job) {  
    this.name=name;  
    this.age=age;  
    this.job=job;  
    this.sayName=function(){alert(this.name);};  
}  
// 作为构造函数使用  
var person1 = new Person("Nicholas",29,"Software Engineer");  
alert(person1.name);  
// 作为普通函数使用  
Person("Greg",27,"Doctor"); // 把方法添加到window中  
window.sayName();         // Greg  
// 在另一个对象的作用域中调用  
var o = new Object();  
Person.call(o, "Kristen", 25, "Nurse");  
o.sayName();               // Kristen
```

构造函数模式实际上是函数方式，因为函数就是对象，可以用new的方式产生对象，用this定义属性和方法。构造函数的缺点是每个实例都要重新创建它的方法，上面的sayName的定义与下面的定义等同：

```
this.sayName= new function(){alert(this.name);};
```


下面的方法可以克服这个问题，但是要定义很多全局函数：

```
function Person(name,age,job) {  
    this.name=name;  
    this.age=age;  
    this.job=job;  
    this.sayName=sayName;  
}  
  
function sayName(){  
    alert(this.name);  
}  
var person1 = new Person("Greg",27,"Doctor");  
person1.sayName();           // Greg
```

● 原型方法

所有类(函数)都有原型(**prototype**)，在原型上定义的函数和变量（数据域）被该类的所有对象所共享，并可以被这些对象直接使用，也可以被覆盖。

```
function Person() {  
}  
Person.prototype.name="Nicholas";  
Person.prototype.age=29;  
Person.prototype.job="Software Engineer";  
Person.prototype.sayName= function() {alert(this.name);};  
  
var person1 = new Person();  
person1.sayName(); // Nicholas--来自原型  
var person2 = new Person();  
person2.name = "Greg";  
person2.sayName(); // Greg--来自实例  
var person3 = new Person();  
person3.sayName(); // Nicholas--来自原型  
alert(person1.sayName==person2.sayName); //true
```

数组

- 定义

```
var a = new Array();           // var a=[];
var b = new Array(2);          // var b=[2];
var c = new Array("tom",3,"jerry");
var d = ["tom",3,"jerry"];      // 等同c;

document.write(a.length);      //=0
document.write(b.length);      //=2
document.write(c.length);      //=3
if (Array.isArray(a)) alert("OK");
```

• 引用

```
var colors=["red","green","blue"];
alert(colors.toString());           //red,blue,green.toLocalString
alert(colors.valueOf());            //red,blue,green
alert(colors);                      //red,blue,green
alert(colors.join(";"));            //red;blue;green
alert(colors.push("yellow","brown")); //5.red,blue,green,yellow,brown
alert(colors.pop());                //返回brown。red,blue,green,yellow
alert(colors.shift());              //返回red。blue,green,yellow
alert(colors.unshift("brown"));     //返回4。brown,blue,green,yellow
alert(colors.sort());               // blue,brown, green,yellow
alert(colors.reverse());            // yellow, green, brown, blue
```

```
var mycars = new Array()
mycars[0] = "Saab"
mycars[1] = "Volvo"
mycars[2] = "BMW"
for (var x in mycars){
    document.write(mycars[x] + "<br />")
}
```

• Javascript的数组常用方法

| | |
|----------|-------------------------|
| concat | 连接两个或更多的数组，并返回结果。 |
| join | 把数组所有元素放入一个字符串，可以指定分隔符。 |
| pop | 删除并返回数组最后一个元素。 |
| push | 向数组末尾添加一个或更多元素，并返回新长度。 |
| shift | 删除并返回数组第一个元素。 |
| unshift | 向数组的开头添加一个或更多元素，并返回新长度 |
| reverse | 颠倒数组中元素的顺序。 |
| slice | 从某个已有的数组返回选定元素 |
| sort | 对数组元素进行排序 |
| splice | 删除元素，并向数组添加新元素 |
| toString | 把数组转换为字符串 |

事件

- 概述

- ✓ 当点击页面元素或者输入字符时会发生鼠标事件和按键事件，浏览器可以捕捉到这些事件。利用这些事件，可以执行一段Javascript编写的代码来实现某些功能，例如，改变网页的面貌。
- ✓ Javascript通过Ajax技术可以在保持原有网页不刷新情况下取回或执行其它网页并把结果显示出来。
- ✓ Javascript可以用于提交前的合法性验证。
- ✓ Javascript也是实现HTML5新增功能的操作语言。

• 定义事件处理程序的第一种方法

```
<!DOCTYPE html>
```

javascript1.jsp

```
<html>
```

```
<head>
```

```
<title>Insert title here</title>
```

```
<script type = "text/javascript">
```

```
    function showMessage(){  
        alert("Hello wolrd!");  
    }
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<input type="button" value="Click Me" onclick="alert(event.type)">
```

```
<input type="button" value="Click Me" onclick="alert(this.value)">
```

 Click
<!--this为被点击的对象-->

```
<input type="button" value="Hello" onclick="showMessage();">
```

<!--调用函数 -->

```
<input type="button" value="Click Me"  
    onclick = "try {showmessage();}catch(ex){}"><!--屏蔽出错信息-->
```

```
</body>
```

```
</html>
```

• 定义事件处理程序的第二种方法

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
    function changetext(obj){
        obj.innerHTML="谢谢!";
    }
</script>
</head>
<body>
<div>
<p id="p1">请点击该文本</p>
<p id="p2">this is another header</p>
</div>
<script type="text/javascript">
    var p1=document.getElementById("p1");
    var f1=function(){changetext(this)};
    p1.onclick= f1;
    p2.onclick= f1;
</script>
</body>
</html>
```

javascript2.jsp

要放在元素p1的后面，因为函数外的语句会顺序会执行。


```
<head>
<script type="text/javascript">
    function bind(fn, context) {
        return function () {
            return fn.apply(context, arguments);
        };
    }
</script>
</head>
<body>
<p id="p1">请点击该文本</p>
<p id="p2">this is another header</p>
<script type="text/javascript">
var handler = {
    message: "Event handled",
    handleClick: function (event) {
        alert(this.message);
    }
};
var btn = document.getElementById("p1");
btn.onclick = bind(handler.handleClick, handler);
</script>
</body>
```

• 定义事件处理程序的第三种方法

```
var p1=document.getElementById("p1");
var f1=function(){alert("hello");};
p1.addEventListener("click",f1,false);
p1.addEventListener("click",function(){changetext(this);});
p1.addEventListener("click",
                    function(event){
                        alert(event.type+": "
                        +event.clientX+", "+event.clientY
                        +event.screenX+", "+event.screenY);
                    });
```

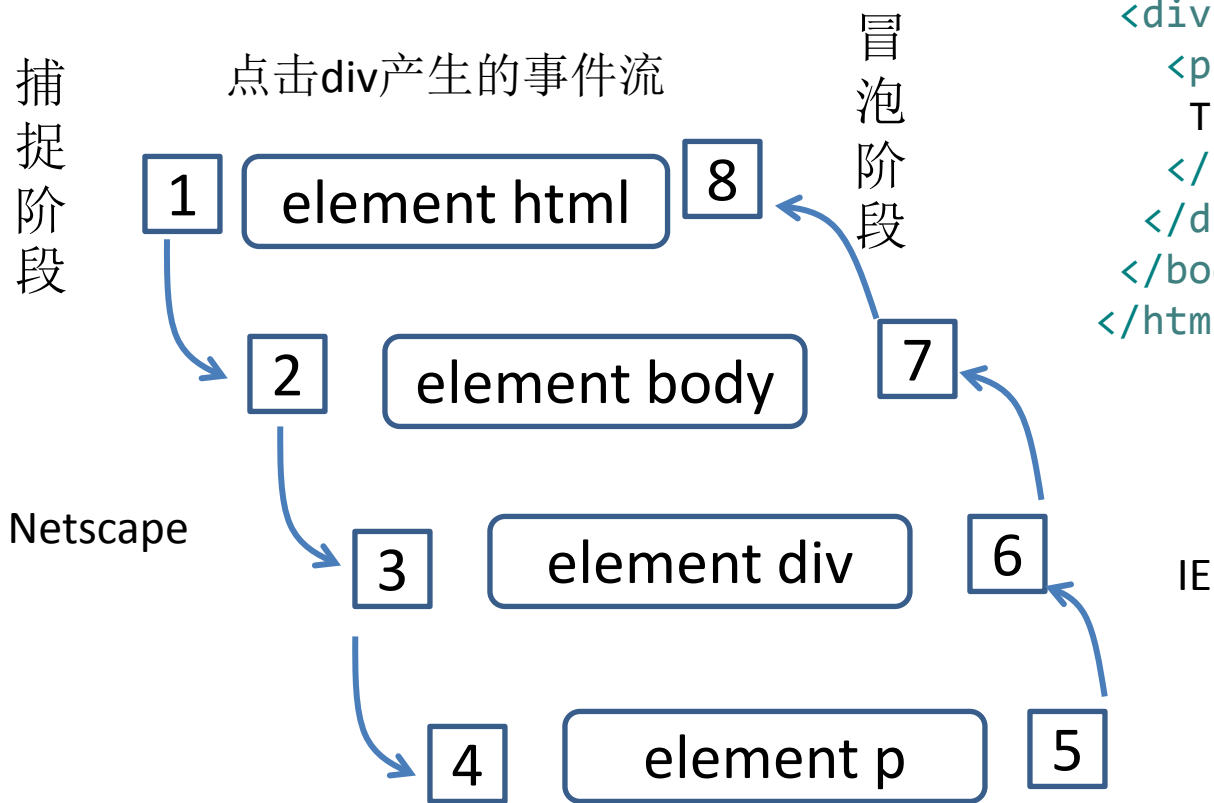
addEventListener的参数:

第一个参数为要捕捉的事件

第二个参数为捕捉到事件后执行的函数

第三个参数false表示事件在冒泡阶段触发, 为true则在捕捉阶段触发。

```
<html>
<head>
<title>events</title>
</head>
<h1>addListener</h1>
<body id="body">
<div id="div">
<p id="p1">
This is first paragraph.
</p>
</div>
</body>
</html>
```



大部分浏览器既有捕捉阶段也有冒泡阶段。除了第三种方法捕捉事件，其它方法只在冒泡阶段补捉事件。

```

<!DOCTYPE html>
<html id="html">
<head>
<meta charset="UTF-8">
<title>events</title>
</head>
<h1>addListener</h1>
<body id="body">
  <div id="div">
    <p id="p1">This is first paragraph.</p>
    <p id="p2">This is second paragraph.</p>
  </div>
</body>
<script>
  var p1 = document.getElementById("p1");
  p1.addEventListener("click", function() {alert("p1");}, false);
  var body = document.getElementById("body");
  body.addEventListener("click", function() {alert("body");}, false);
  var div = document.getElementById("div");
  div.addEventListener("click", function() {alert("div");}, false);
  var html = document.getElementById("html");
  html.addEventListener("click", function() {alert("html");}, false);
  p1.addEventListener("click", function() {alert("p1");}, true);
  body.addEventListener("click", function() {alert("body");}, true);
  div.addEventListener("click", function() {alert("div");}, true);
  html.addEventListener("click", function() {alert("html");}, true);
</script>
</html>

```

events

//冒泡阶段

//捕捉阶段

- 删除事件处理程序

```
btn.onclick =null;  
btn.removeEventListener("click",  
    function() {alert("Hello!");},  
    false);           //函数要完全一样才有效，否则，删除无效。
```

```
var handler = function(){alert(this.ad)};  
btn.addEventListener("click",handler, false);  
btn.removeEventListener("click",handler, false);  
// 第三个参数与定义时也要一样(冒泡还是捕捉)
```

- 阻止事件传播

```
var link = document.getElementById("myLink");
link.onclick = function(event){
    event.preventDefault();
};    // 取消链接的默认行为，即不会导航到url。
      //
```

```
var btn = document.getElementById("myBtn");
btn.onclick = function(event){
    event.stopPropagation();    //阻止该事件传播
};
```

```
document.body.onclick = function(event){
    alert("body clicked");
};    //不起作用
```

要处理IE8.0及更早版本的事件处理程序: javascript3.jsp
javascript4.jsp

```
var p1=document.getElementById("p1");
var f1=function(){alert("hello");};
if(p1.addEventListener){
    p1.addEventListener("click", function(event){
        alert(event.type+":"+event.clientX+","+event.clientY+"
            +event.screenX+","+event.screenY);
    });
}
else{
    p1.attachEvent("onclick", function(){ var event = window.event;
        alert(event.type+":"+event.clientX+","+event.clientY
            +event.screenX+","+event.screenY);
    });
}
```

p1.detachEvent("onclick", function(){ ... }); //删除事件处理程序

window.event.returnValue = false // IE9之前取消默认行为

window.event.cancelBubble = true; // IE9之前阻止事件传播 :

● 事件对象(event)

在触发某个事件时，会产生一个事件对象event，这个对象包含所有与事件相关的信息。IE8.0及更早版本的事件对象为window.event。

```
<input type="button" value="Click Me" onclick="alert(event.type)">
```

✓用户界面事件: load, unload, abort, error, select, resize

✓焦点事件: blur(失焦点), focus(不冒泡), focusin(冒泡) focusout

```
window.onload = function(){alert("Hello!")}  
//页面或图像完全加载后发生
```

```
<body onload = "alert('Hello!')">
```

```
<img onload = "alert('Hello!')">
```

```
<input type="text" onblur="alert('blur')"  
onfocus="alert('focus')" >
```


✓ 鼠标事件

```
<div onclick="alert('click')" > hello </div>
```

click 单击	dblclick 双击	mousedown 按下	mouseup 放开
mouseenter 进入	mouseout 离开	mousemove 移动	mouseover 悬浮

event.clientX, event.clientY	鼠标在客户区的坐标
event.screenX, event.screenY	鼠标在窗口的坐标
event.offsetX, event.offsetY	鼠标在点击对象上的坐标
event.pageX, event.pageY	鼠标在页面中的坐标
event.x, event.y	相对于上级元素的坐标(默认为BODY)
event.button	按下的鼠标键：0 没按键 1 按左键 2 按右键 3 按左右键 4 按中间键 5 按左键和中间键 6 按右键和中间键 7 按所有的键
event.fromElement, event.toElement	鼠标移动离开和进入的对象
event.srcElement	鼠标点击的元素

✓ 键盘事件

keydown	按下任意键
keyup, keypress	按下字符键，长按则连续触发
event.keyCode	按键内码
event.charCode	按键字符(可显示)
event.altKey,	
event.ctrlKey, event.shiftKey:	alt, ctrl或shift的键是否按下，按下为true

✓ 触摸屏事件

touchstart	开始触摸
touchend	结束触摸。触摸位置的参数与鼠标事件一样。
touchmove	手指移动

✓ 触摸属性

touches	跟踪Touch对象的数组
targetTouches	特定于时间目标的Touch对象数组
changeTouches	自从上次触摸以来发生了什么改变的Touch对象数组

✓ 手势事件

gesturestart
gesturechange
gestureend
event.scale
event.orientation

两个手指均触摸时发生
至少有一个手指位置发生变化。
至少有一个手指从触摸屏移开。
两指间距离,从1开始, 值越大距离越大
手指变化的旋转角度, 正(负)值为正(逆)时针旋转

✓ 触摸屏其它事件

orientationchange
devicemotion

方向: 0-肖像模式 90-左旋90度 90-右旋90度
设备正在移动

✓ 事件列表:

onabort	图像加载被中断
onblur	元素失去焦点
onchange	用户改变域的内容
onerror	当加载文档或图像时发生某个错误
onfocus	元素获得焦点
onload	某个页面或图像被完成加载
onreset	重置按钮被点击
onresize	窗口或框架被调整尺寸
onselect	文本被选定
onsubmit	提交按钮被点击
onunload	用户退出页面
onpropertychange	元素属性改变
event.srcElement	触发事件的元素
event.cancelBubble	设为true则取消冒泡，默认值为false (<=IE 8)
event.propertyName	设置或返回元素的变化了的属性的名称
event.returnValue	设置为 false， 源对象上事件的默认操作被取消(<=IE 8)

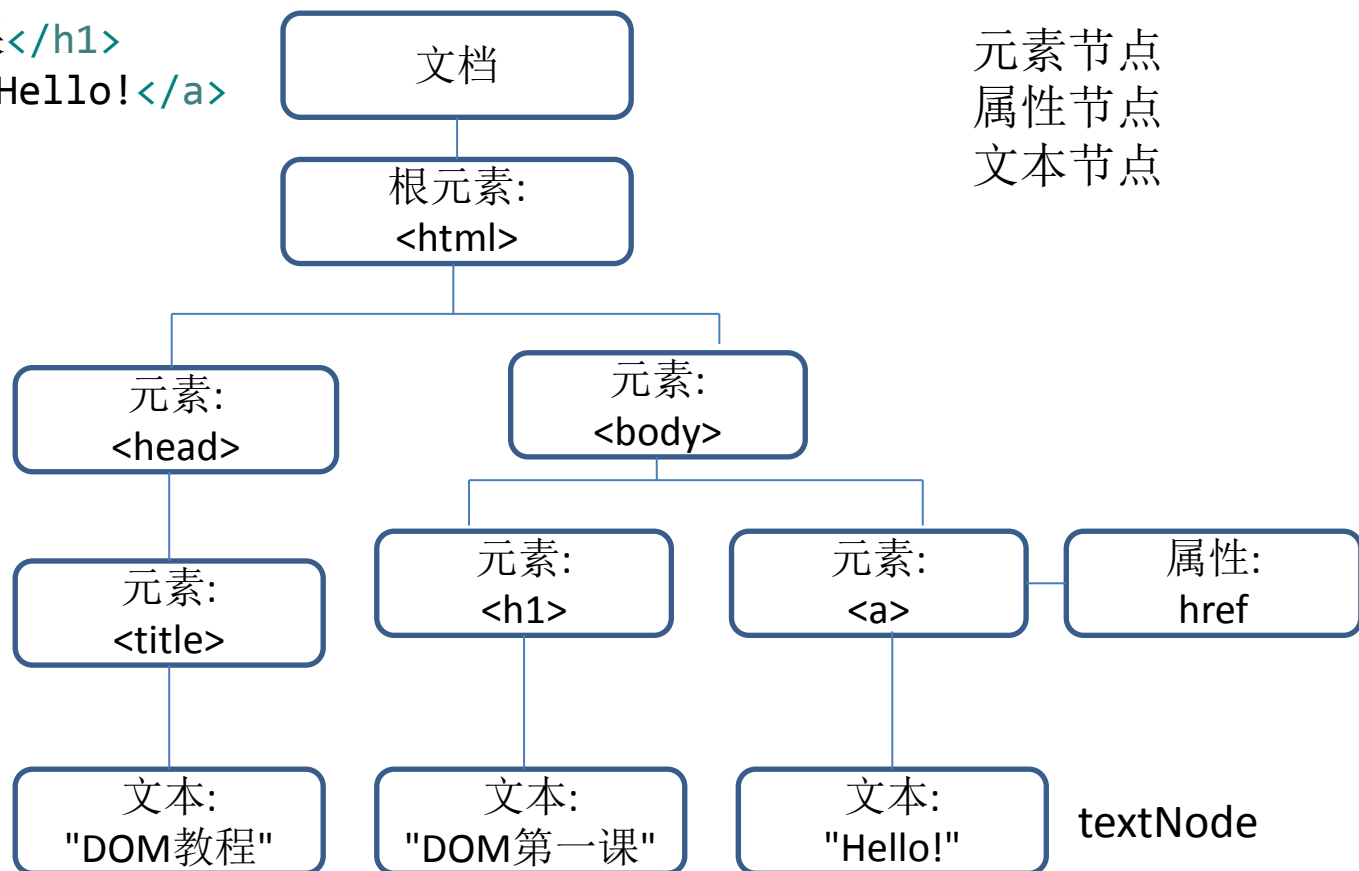
```
var btn = document.getElementById("myBtn");
var handler=function(event){
    switch(event.type){
        case "click":
            alert("Clicked"); break;
        case "mouseover":
            event.target.style.backgroundColor = "red";
        case "mouseout":
            event.target.style.backgroundColor = "";
    }
};
btn.onclick = handler;
btn.onmouseover = handler;
btn.onmouseout = handler;

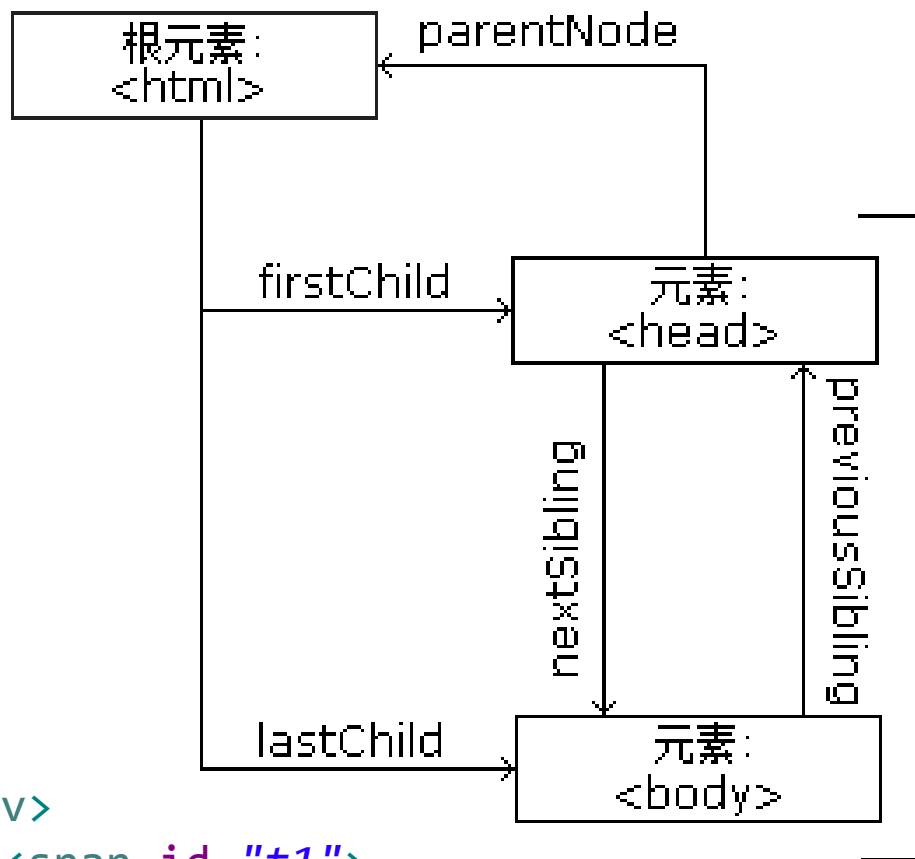
document.body.onclick=function(event){
    alert(event.currentTarget===document.body);
    alert(this===document.body);
    alert(event.Target===document.getElementById("myBtn"));
}; //点击了button
```

DOM节点树

```
<html>
<head>
  <title>DOM 教程</title>
</head>
<body>
  <h1>DOM 第一课</h1>
  <a href="#" >Hello!</a>
</body>
</html>
```

每个节点都是一个对象





<html>的子节点
同时，彼此互为同胞

```
<div>  
  <span id="t1">  
    My text  
  </span>  
</div>
```

```
alert(document.getElementById("t1").parentNode.nodeName)
```

节点的属性:

nodeName	节点名
nodeType	类型（元素 1 属性 2 文本 3 注释 8 文档 9）
nodeValue	节点值
parentNode	节点（元素）的父节点
childNodes	节点（元素）的所有子节点（数组）
children	同上。children只返回元素如input, span, script, div等，不会返回TextNode，而childNodes不但会返回元素，还会返回TextNode，注释等。
innerHTML	节点（元素）的内容。innerText--只有IE可用
outerHTML	节点（元素）的元素(包含自己的标签)。outerText
attributes	节点（元素）的属性节点
className	节点的属性class的值
classList	节点所有的类。可以用方法add(增加一个类)，remove（删除一个类），toggle（增/删类），contains（是否包含一个类）修改它。(HTML 5的功能)
data-	自定义属性名 html5

firstChild	节点（元素）的第一个子节点	(firstElementChild)
lastChild	节点（元素）的最后一个子节点	(lastElementChild)
nextSibling	节点的下一个兄弟	(nextElementSibling)
previousSibling	节点的下一个兄弟	(previousElementSibling)
		(childElementCount)

* 由于元素内容为空格时，与其它浏览器不同，IE10之前的版本不会返回文本节点，所以为了统一所有浏览器的行为，html5为节点重新定义了括号中的属性。

操作节点的方法：

<code>appendChild()</code>	把新的子节点添加到指定节点。
<code>removeChild()</code>	删除子节点。
<code>replaceChild()</code>	替换子节点。
<code>insertBefore()</code>	在指定的子节点前面插入新的子节点。
<code>createAttribute()</code>	创建属性节点。
<code>createElement()</code>	创建元素节点。
<code>createTextNode()</code>	创建文本节点。
<code>getAttribute()</code>	返回指定的属性值。
<code>setAttribute()</code>	把指定属性设置或修改为指定的值。
<code>focus()</code>	让输入类元素得焦点 html5
<code>insertAdjacentHTML()</code>	在元素的开始标签和结束标签的前后插入元素。 <code>e.insertAdjacentHTML("beforebegin", "<p>Welcome!</p>")</code>
<code>scrollIntoView()</code>	滚动页面让元素可见。

contains()

是否为后代元素。html5

compareDocumentPosition()

比较位置：1-无关 2-居前 3-居后 8-包含 16-被包含

通过document选择节点的方法：`document.getElementById("first")`

getElementById()

返回带有指定 ID 的元素。

getElementsByTagName()

返回包含带有指定标签名称的所有元素的节点列表（集合/节点数组）。

getElementsByClassName()

返回包含带有指定类名(可以是空格隔开的多个类名)的所有元素的节点列表(数组)。html5

querySelectorAll()

取得与CSS选择符匹配的所有节点。html5

querySelector()

取得与CSS选择符匹配的第一个节点。html5

matchesSelector()

看元素(第一个参数)与CSS选择符(第二个参数)是否匹配(目前浏览器支持都不高) html5

```
<html>
<head>
<script type="text/javascript">
    function changetext(){
        var oTest=document.getElementById("p1");
        var newNode1 = document.createElement("br");
        oTest.parentNode.insertBefore(newNode1,null);
        var newNode2 = document.createElement("br");
        oTest.parentNode.insertBefore(newNode2,null);
        var newNode3 = oTest.cloneNode(true);
        oTest.parentNode.insertBefore(newNode3,null);
    }
</script>
</head>
<body>
<form name="p3" >
<input id="p1" type="text" value="Hello!"/>
</form>
<script type="text/javascript">
    var p1=document.getElementById("p1");
    if(p1.addEventListener){
        p1.addEventListener("click",function(){changetext();});
    }
    else{
        p1.attachEvent("onclick",function(){changetext();});
    }
</script>
</body>
</html>
```

元素属性修改

- 第一种方法

```
<p id="hello" class="c1">Hello World!</p>
```

```
<script>
```

```
    document.getElementById(" hello").style.color="blue";
```

```
    document.getElementById(" hello").className="c2";
```

```
</script>
```

- 第二种方法

```
<script>
```

```
    var p1 = document.getElementById('hello');
```

```
    p1.setAttribute("style", "color:blue");
```

```
    p1.setAttribute("class", "c2 c3");
```

```
</script>
```

- 第三种方法

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

```
<span onclick="javascript:document.getElementById('css').href = 'ie.css'">  
点我改变样式</span>
```

- 第四种方法

```
<STYLE>
```

```
    .theforever {width:50px;color:red;}
```

```
    #theforever {width:150px;color:silver;}
```

```
</STYLE>
```

```
<div class="theforever">这里应该是红色的，但它会被下面的JS  
    通过改变CSS样式设置而变成黄色</div>
```

```
<div id="theforever">这里应该是银灰色的，的确这个不会发生  
    变化，只作为对比</div>
```

```
<script> document.styleSheets[0].cssText  
    =document.styleSheets[0].cssText.replace(/red/g,"yellow");
```

```
</script>
```

document.styleSheets是个数组，可以取到页面定义的所有样式。

Ajax技术(get)

(Asynchronous Javascript And XML)

javascript7.jsp

```
<html><head>
<script type="text/javascript">
    function show(){
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function(){
            if(xmlhttp.readyState==4){
                if(xmlhttp.status >=200 && xmlhttp.status<300
                || xmlhttp.status >=304){
                    alert(xmlhttp.responseText);
                    var oTest=document.getElementById("p4");
                    oTest.innerHTML=xmlhttp.responseText;
                }
                else{
                    alert("error");
                }
            }
        };
        xmlhttp.open("get","http://202.116.76.22:8080/js/ajaxGet.jsp", true);
        xmlhttp.send(("x=6&y=7"); // 参数。没有参数就用null。
    }
</script> </head>
```

```
<body>
  <form name="p3">
    <p id="p4">hello world!</p>
    <input id="p1" type="text" value="Hello!" />
    <input id="p2" type="button" value="showHTML" />
  </form>
  <script type="text/javascript">
    p2=document.getElementById("p2");
    p2.addEventListener("click",show);
  </script>
</body>
</html>
```

ajaxGet.jsp

```
<h1>DOM 1st Class </h1>
<p>Hello world!</p>
```



```
xmlhttp.open("get","http://202.116.76.22:8080/js/ajaxGet.jsp", true);  
xmlhttp.send(("x=6&y=7"));    // 参数,没有参数就用null。
```

第一个参数: get-取回网页, post-提交表单

第二个参数: 请求的URL

第三个参数: 是否异步发送

xmlhttp的属性:

responseText: 作为响应主体被返回的文本

responseXML: 如果响应的内容类型是"text/xml"或
"application/xml",这里保存着响应数据。

status: 响应的HTTP状态

statusText: HTTP状态说明。

readyState: 请求或响应过程的当前活动阶段。

xmlhttp.readyState

- 0: 未初始化。尚未调用open
- 1: 启动。已调用open，未调用send
- 2: 发送。已调用send
- 3: 接收。已接收到部分响应数据
- 4: 完成。已接收到所有响应数据

xmlhttp.status

- 200: 成功。响应有效，可以取到responseText。
- 304: 请求的资源没有更改，可以直接使用缓存数据。
可以取到responseText。

.....

http头部信息

```
xmlhttp.setRequestHeader("Content-Type",  
    "application/x-www-form-urlencoded");
```

Accept: 浏览器能够处理的内容类型

Accept-Charset: 浏览器能够显示的字符集

Accept-Language: 浏览器当前设置的语言

Cookie: 当前页面设置的任何Cookie

Host: 发出请求的页面所在域(domain, 例如:www.sohu.com)

Referer: 发出请求的页面的URL

User-Agent: 浏览器的用户代理字符串

设定超时

```
xmlhttp.timeout=1000; //ms  
xmlhttp.ontimeout=function(){  
    alert("Request did not return in a second.")  
};
```

Ajax技术(post)

```
<script type="text/javascript">
    function show(){
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function(){
            if(xmlhttp.readyState==4){
                if(xmlhttp.status >=200 && xmlhttp.status<300
                    || xmlhttp.status >=304){
                    alert(xmlhttp.responseText);
                    var oTest=document.getElementById("p4");
                    oTest.innerHTML=xmlhttp.responseText;
                } else{
                    alert("error");
                }
            }
        };
        var v1=document.getElementById("p1").value;
        var param = "p1="+v1+"&p2=345";
        xmlhttp.open("post","http://202.116.76.22:8080/js/ajaxPost.jsp", true);
        xmlhttp.setRequestHeader("Content-Type",
                                "application/x-www-form-urlencoded");
        xmlhttp.send(param);    // 没有参数就用null。
    }
</script>
```

javascrip8.jsp

```

<form name="p3">
  <p id="p4">hello world!</p>
  <input id="p1" type="text" value="Hello!" />
  <input id="p2" type="button" value="showHTML" />
</form>
<script type="text/javascript">
  p2=document.getElementById("p2");
  if(p2.addEventListener){
    p2.addEventListener("click",show);
  }
  else{
    p2.addEventListener("click",show);
  }
</script>
</body>
</html>

```

ajaxPost.jsp

```

<%
  String p1=request.getParameter("p1");
  String p2=request.getParameter("p2");
  out.print("eerrwe:"+p1+" "+p2);
%>

```

采用函数的提交表单

```
<html>
<head>
<script type="text/javascript">
    function show(){
        // document.p3.submit();
    }
    function submit1(){
        return false;
        var p1=document.getElementById("p1");
        if(p1==""){
            alert("error!");
            return false;
        }
        else{
            return true;
        }
        document.p3.submit();
    }
</script>
</head>
```

```
<body>
<form name="p3" method="post" action="post.jsp">
  <p id="p4">hello world!</p>
  <input id="p1" type="text" value="Hello!" />
  <input id="p2" type="button" value="showHTML" />
  <!--      <input id="p5"
            type="submit" value="submit"/> -->
</form>
<script type="text/javascript">
  p2=document.getElementById("p2");
  if(p2.addEventListener){
    p2.addEventListener("click",show);
  }
  else{
    p2.addEventListener("click",show);
  }
</script>
</body>
</html>
```

浏览器对象

- javascript可以通过一组对象获得浏览器属性或操作浏览器：

通过window对象访问浏览器当前打开的窗口

通过document 对象访问每个载入浏览器的 HTML 文档

通过screen对象获得显示屏的特性

通过location对象得到和设置浏览器地址栏的内容

通过history对象操作浏览历史

通过navigator 对象获得浏览器本身的信息。

通过global对象可以使用全局变量和全局函数

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

location对象

- location 对象用于获得当前页面的地址 (URL)，并把浏览器重定向到新的页面。通过location.href可以返回当前页面的url，设置它可以转到 相应页面。

Location 对象属性

属性	描述
hash	设置或返回从井号 (#) 开始的 URL（锚）。
host	设置或返回主机名和当前 URL 的端口号。
hostname	设置或返回当前 URL 的主机名。
href	设置或返回完整的 URL。
pathname	设置或返回当前 URL 的路径部分。
port	设置或返回当前 URL 的端口号。
protocol	设置或返回当前 URL 的协议。
search	设置或返回从问号 (?) 开始的 URL（查询部分）。

Location 对象方法

属性	描述
assign()	加载新的文档。
reload()	重新加载当前文档。
replace()	用新的文档替换当前文档。

history对象

- 通过history 对象可以访问浏览器历史。

History 对象属性

属性

[length](#)

描述

返回浏览器历史列表中的 URL 数量。

History 对象方法

方法

[back\(\)](#)

[forward\(\)](#)

[go\(\)](#)

描述

加载 history 列表中的前一个 URL。

加载 history 列表中的下一个 URL。

加载 history 列表中的某个具体页面。history.go(-1)与 history.back()作用相同。

window对象

window 对象表示浏览器窗口。所有 JavaScript 全局对象、函数以及变量均自动成为 **window** 对象的成员。全局变量是 **window** 对象的属性。全局函数是 **window** 对象的方法。HTML DOM 的 **document** 也是 **window** 对象的属性之一。

确定浏览器窗口的尺寸（不包括工具栏和滚动条）。

```
var w=window.innerWidth || document.documentElement.clientWidth ||  
    document.body.clientWidth;  
var h=window.innerHeight || document.documentElement.clientHeight  
    || document.body.clientHeight;
```

Internet Explorer 8、7、6、5

对浏览器窗口的操作：

window.open() - 打开新窗口

window.close() - 关闭当前窗口

window.moveTo() - 移动当前窗口

window.resizeTo() - 调整当前窗口的尺寸

属性	描述
<u>closed</u>	返回窗口是否已被关闭。
<u>defaultStatus</u>	设置或返回窗口状态栏中的默认文本。
<u>document</u>	对 Document 对象的只读引用。请参阅 <u>Document 对象</u> 。
<u>history</u>	对 History 对象的只读引用。请参阅 <u>History 对象</u> 。
<u>innerheight</u>	返回窗口的文档显示区的高度。
<u>innerwidth</u>	返回窗口的文档显示区的宽度。
<u>location</u>	用于窗口或框架的 Location 对象。请参阅 <u>Location 对象</u> 。
<u>Navigator</u>	对 Navigator 对象的只读引用。请参阅 <u>Navigator 对象</u> 。
<u>outerheight</u>	返回窗口的外部高度。
<u>outerwidth</u>	返回窗口的外部宽度。
<u>Screen</u>	对 Screen 对象的只读引用。请参阅 <u>Screen 对象</u> 。
<u>self</u>	返回对当前窗口的引用。等价于 Window 属性。
<u>status</u>	设置窗口状态栏的文本。
<u>top</u>	返回最顶层的先辈窗口。

方法	描述
<u>alert()</u>	显示带有一段消息和一个确认按钮的警告框。
<u>blur()</u>	把键盘焦点从顶层窗口移开。
<u>clearInterval()</u>	取消由 setInterval() 设置的 timeout。
<u>clearTimeout()</u>	取消由 setTimeout() 方法设置的 timeout。
<u>close()</u>	关闭浏览器窗口。
<u>confirm()</u>	显示带有一段消息以及确认按钮和取消按钮的对话框。
<u>createPopup()</u>	创建一个 pop-up 窗口。
<u>focus()</u>	把键盘焦点给予一个窗口。
<u>moveBy()</u>	可相对窗口的当前坐标把它移动指定的像素。
<u>moveTo()</u>	把窗口的左上角移动到一个指定的坐标。
<u>open()</u>	打开一个新的浏览器窗口或查找一个已命名的窗口。
<u>print()</u>	打印当前窗口的内容。
<u>prompt()</u>	显示可提示用户输入的对话框。
<u>resizeBy()</u>	按照指定的像素调整窗口的大小。
<u>resizeTo()</u>	把窗口的大小调整到指定的宽度和高度。
<u>scrollBy()</u>	按照指定的像素值来滚动内容。
<u>scrollTo()</u>	把内容滚动到指定的坐标。
<u>setInterval()</u>	按照指定的周期（以毫秒计）来调用函数或计算表达式。
<u>setTimeout()</u>	在指定的毫秒数后调用函数或计算表达式。

事件：onload

document对象

每个载入浏览器的 HTML 文档都会成为 Document 对象。

Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

document 对象的集合

集合	描述	IE	F	O	W3C	属性	描述	IE	F	O	W3C
all[]	提供对文档中所有 HTML 元素的访问。	4	1	9	No	body	提供对 <body> 元素的直接访问。				
anchors[]	返回对文档中所有 Anchor 对象的引用。	4	1	9	Yes		对于定义了框架集的文档，该属性引用最外层的 <frameset>。				
applets	返回对文档中所有 Applet 对象的引用。	-	-	-	-	cookie	设置或返回与当前文档有关的所有 cookie。	4	1	9	Yes
forms[]	返回对文档中所有 Form 对象引用。	4	1	9	Yes	domain	返回当前文档的域名。	4	1	9	Yes
images[]	返回对文档中所有 Image 对象引用。	4	1	9	Yes	lastModified	返回文档被最后修改的日期和时间。	4	1	No	No
links[]	返回对文档中所有 Area 和 Link 对象引用。	4	1	9	Yes	referrer	返回载入当前文档的文档的 URL。	4	1	9	Yes
						title	返回当前文档的标题。	4	1	9	Yes
						URL	返回当前文档的 URL。	4	1	9	Yes

document 对象的方法

方法	描述
close()	关闭用 <code>document.open()</code> 方法打开的输出流，并显示选定的数据。
getElementById()	返回对拥有指定 id 的第一个对象的引用。
getElementsByName()	返回带有指定名称的对象集合。
getElementsByTagName()	返回带有指定标签名的对象集合。
getElementsByClassName()	返回带有指定类名的对象集合
open()	打开一个流，以收集来自任何 <code>document.write()</code> 或 <code>document.writeln()</code> 方法的输出。
write()	向文档写 HTML 表达式 或 JavaScript 代码。
writeln()	等同于 <code>write()</code> 方法，不同的是在每个表达式之后写一个换行符。

事件：

`window.onload`和`document.ready`的不同

<http://www.cnblogs.com/a546558309/p/3478344.html>

html5

属性	描述
<code>readyState</code>	取值"loading"和"complete"，表示正在加载文档和文档加载完毕。
<code>head</code>	返回文档的head元素。 <code>document.body</code> 返回body元素。
<code>charset</code>	文档中实际使用的字符集
<code>documentElement</code>	把document当成元素对待

方法	描述
<code>querySelectorAll()</code>	返回与CSS选择符匹配的所有节点。
<code>querySelector()</code>	返回与CSS选择符匹配的第一个节点。
<code>matchSelector()</code>	判断元素与CSS选择符是否匹配。很多浏览器不支持
<code>hasFocus()</code>	判断文档是否获得了焦点

screen对象

Screen 对象包含有关客户端显示屏幕的信息。

Screen 对象属性

属性	描述
<u>availHeight</u>	返回显示屏幕的高度 (除 Windows 任务栏之外)。
<u>availWidth</u>	返回显示屏幕的宽度 (除 Windows 任务栏之外)。
<u>bufferDepth</u>	设置或返回调色板的比特深度。
<u>colorDepth</u>	返回目标设备或缓冲器上的调色板的比特深度。
<u>deviceXDPI</u>	返回显示屏幕的每英寸水平点数。
<u>deviceYDPI</u>	返回显示屏幕的每英寸垂直点数。
<u>fontSmoothingEnabled</u>	返回用户是否在显示控制面板中启用了字体平滑。
<u>height</u>	返回显示屏幕的高度。
<u>logicalXDPI</u>	返回显示屏幕每英寸的水平方向的常规点数。
<u>logicalYDPI</u>	返回显示屏幕每英寸的垂直方向的常规点数。
<u>pixelDepth</u>	返回显示屏幕的颜色分辨率（比特每像素）。
<u>updateInterval</u>	设置或返回屏幕的刷新率。
<u>width</u>	返回显示器屏幕的宽度。

Navigator 对象

Navigator 对象包含有关浏览器的信息。

Navigator 对象属性

属性	描述
<u>appName</u>	返回浏览器的代码名。
<u>appMinorVersion</u>	返回浏览器的次级版本。
<u>appName</u>	返回浏览器的名称。
<u>appVersion</u>	返回浏览器的平台和版本信息。
<u>browserLanguage</u>	返回当前浏览器的语言。
<u>cookieEnabled</u>	返回指明浏览器中是否启用 cookie 的布尔值。
<u>cpuClass</u>	返回浏览器系统的 CPU 等级。
<u>onLine</u>	返回指明系统是否处于脱机模式的布尔值。
<u>platform</u>	返回运行浏览器的操作系统平台。
<u>systemLanguage</u>	返回 OS 使用的默认语言。
<u>userAgent</u>	返回由客户机发送服务器的 user-agent 头部的值。
<u>userLanguage</u>	返回 OS 的自然语言设置。

Navigator 对象方法

方法	描述
<u>javaEnabled()</u>	规定浏览器是否启用 Java 。
<u>taintEnabled()</u>	规定浏览器是否启用数据污点 (data tainting)。

Global

函数

[decodeURI\(\)](#)

[decodeURIComponent\(\)](#)

[encodeURI\(\)](#)

[encodeURIComponent\(\)](#)

[escape\(\)](#)

[eval\(\)](#)

[getClass\(\)](#)

[isFinite\(\)](#)

[isNaN\(\)](#)

[Number\(\)](#)

[parseFloat\(\)](#)

[parseInt\(\)](#)

[String\(\)](#)

[unescape\(\)](#)

描述

解码某个编码的 URI。

解码一个编码的 URI 组件。

把字符串编码为 URI。

把字符串编码为 URI 组件。

对字符串进行编码。

计算 JavaScript 字符串，并把它作为脚本代码来执行。

返回一个 `JavaScript` 的 `JavaClass`。

检查某个值是否为有穷大的数。

检查某个值是否是数字。

把对象的值转换为数字。

解析一个字符串并返回一个浮点数。

解析一个字符串并返回一个整数。

把对象的值转换为字符串。

对由 `escape()` 编码的字符串进行解码。

属性

[Infinity](#)

[java](#)

[NaN](#)

[Packages](#)

[undefined](#)

描述

代表正的无穷大的数值。

代表 `java.*` 包层级的一个 `JavaPackage`。

指示某个值是不是数字值。

根 `JavaPackage` 对象。

指示未定义的值。

其它对象

- Date对象
- Math对象
- cookie对象
- Number对象
- String对象
- RegExp对象
- DOM对象

Date对象

```
var myDate=new Date()  
document.write(myDate.getMonth())
```

Date 对象方法

方法	描述
Date()	返回当日的日期和时间。
getDate()	从 Date 对象返回一个月中的某一天 (1 ~ 31)。
getDay()	从 Date 对象返回一周中的某一天 (0 ~ 6)。
getMonth()	从 Date 对象返回月份 (0 ~ 11)。
getFullYear()	从 Date 对象以四位数字返回年份。
getYear()	请使用 getFullYear() 方法代替。
getHours()	返回 Date 对象的小时 (0 ~ 23)。
getMinutes()	返回 Date 对象的分钟 (0 ~ 59)。
getSeconds()	返回 Date 对象的秒数 (0 ~ 59)。
getMilliseconds()	返回 Date 对象的毫秒(0 ~ 999)。
getTime()	返回 1970 年 1 月 1 日至今的毫秒数。
getTimezoneOffset()	返回本地时间与格林威治标准时间 (GMT) 的分钟差。
getUTCDate()	根据世界时从 Date 对象返回月中的一天 (1 ~ 31)。
getUTCDay()	根据世界时从 Date 对象返回周中的一天 (0 ~ 6)。
getUTCMonth()	根据世界时从 Date 对象返回月份 (0 ~ 11)。
getUTCFullYear()	根据世界时从 Date 对象返回四位数的年份。
getUTCHours()	根据世界时返回 Date 对象的小时 (0 ~ 23)。
getUTCMinutes()	根据世界时返回 Date 对象的分钟 (0 ~ 59)。
getUTCSeconds()	根据世界时返回 Date 对象的秒钟 (0 ~ 59)。
getUTCMilliseconds()	根据世界时返回 Date 对象的毫秒(0 ~ 999)。
parse()	返回1970年1月1日午夜到指定日期（字符串）的毫秒数。

方法	描述
setDate()	设置 Date 对象中月的某一天 (1 ~ 31)。
setMonth()	设置 Date 对象中月份 (0 ~ 11)。
setFullYear()	设置 Date 对象中的年份（四位数字）。
setYear()	请使用 setFullYear() 方法代替。
setHours()	设置 Date 对象中的小时 (0 ~ 23)。
setMinutes()	设置 Date 对象中的分钟 (0 ~ 59)。
setSeconds()	设置 Date 对象中的秒钟 (0 ~ 59)。
setMilliseconds()	设置 Date 对象中的毫秒 (0 ~ 999)。
setTime()	以毫秒设置 Date 对象。
setUTCDate()	根据世界时设置 Date 对象中月份的一天 (1 ~ 31)。
setUTCMonth()	根据世界时设置 Date 对象中的月份 (0 ~ 11)。
setUTCFullYear()	根据世界时设置 Date 对象中的年份（四位数字）。
setUTCHours()	根据世界时设置 Date 对象中的小时 (0 ~ 23)。
setUTCMinutes()	根据世界时设置 Date 对象中的分钟 (0 ~ 59)。
setUTCSeconds()	根据世界时设置 Date 对象中的秒钟 (0 ~ 59)。
setUTCMilliseconds()	根据世界时设置 Date 对象中的毫秒 (0 ~ 999)。
toSource()	返回该对象的源代码。
toString()	把 Date 对象转换为字符串。
toTimeString()	把 Date 对象的时间部分转换为字符串。
toDateString()	把 Date 对象的日期部分转换为字符串。
toGMTString()	请使用 toUTCString() 方法代替。
toUTCString()	根据世界时，把 Date 对象转换为字符串。
toLocaleString()	根据本地时间格式，把 Date 对象转换为字符串。
toLocaleTimeString()	根据本地时间格式，把 Date 对象的时间部分转换为字符串。
toLocaleDateString()	根据本地时间格式，把 Date 对象的日期部分转换为字符串。
UTC()	根据世界时返回 1970 年 1 月 1 日到指定日期的毫秒数。
valueOf()	返回 Date 对象的原始值。

Math对象

Math 对象方法

```
var pi_value=Math.PI;  
var sqrt_value=Math.sqrt(15);
```

Math 对象属性

属性	描述
<u>E</u>	返回算术常量 e ，即自然对数的底数（约等于 2.718 ）。
<u>LN2</u>	返回 2 的自然对数（约等于 0.693 ）。
<u>LN10</u>	返回 10 的自然对数（约等于 2.302 ）。
<u>LOG2E</u>	返回以 2 为底的 e 的对数（约等于 1.414 ）。
<u>LOG10E</u>	返回以 10 为底的 e 的对数（约等于 0.434 ）。
<u>PI</u>	返回圆周率（约等于 3.14159 ）。
<u>SQRT1_2</u>	返回返回 2 的平方根的倒数（约等于 0.707 ）。
<u>SQRT2</u>	返回 2 的平方根（约等于 1.414 ）。

方法	描述
<u>abs(x)</u>	返回数的绝对值。
<u>acos(x)</u>	返回数的反余弦值。
<u>asin(x)</u>	返回数的反正弦值。
<u>atan(x)</u>	以介于 $-\pi/2$ 与 $\pi/2$ 弧度之间的数值来返回 x 的反正切值。
<u>atan2(y,x)</u>	返回从 x 轴到点 (x,y) 的角度（介于 $-\pi/2$ 与 $\pi/2$ 弧度之间）。
<u>ceil(x)</u>	对数进行上舍入。
<u>cos(x)</u>	返回数的余弦。
<u>exp(x)</u>	返回 e 的指数。
<u>floor(x)</u>	对数进行下舍入。
<u>log(x)</u>	返回数的自然对数（底为 e ）。
<u>max(x,y)</u>	返回 x 和 y 中的最高值。
<u>min(x,y)</u>	返回 x 和 y 中的最低值。
<u>pow(x,y)</u>	返回 x 的 y 次幂。
<u>random()</u>	返回 $0 \sim 1$ 之间的随机数。
<u>round(x)</u>	把数四舍五入为最接近的整数。
<u>sin(x)</u>	返回数的正弦。
<u>sqrt(x)</u>	返回数的平方根。
<u>tan(x)</u>	返回角的正切。
<u>toSource()</u>	返回该对象的源代码。
<u>valueOf()</u>	返回 Math 对象的原始值。

RegExp对象

直接量: /pattern/attributes

attributes

修饰符	描述
i	执行对大小写不敏感的匹配。
g	执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。
m	执行多行匹配。

方括号

方括号用于查找某个范围内的字符:

表达式	描述
[abc]	查找方括号之间的任何字符。
[^abc]	查找任何不在方括号之间的字符。
[0-9]	查找任何从 0 至 9 的数字。
[a-z]	查找任何从小写 a 到小写 z 的字符。
[A-Z]	查找任何从大写 A 到大写 Z 的字符。
[A-z]	查找任何从大写 A 到小写 z 的字符。
[adgk]	查找给定集合内的任何字符。
[^adgk]	查找给定集合外的任何字符。
(red blue green)	查找任何指定的选项。

元字符

元字符（Metacharacter）是拥有特殊含义的字符:

元字符	描述
.	查找单个字符，除了换行和行结束符。
\w	查找单词字符。
\W	查找非单词字符。
\d	查找数字。
\D	查找非数字字符。
\s	查找空白字符。
\S	查找非空白字符。
\b	匹配单词边界。
\B	匹配非单词边界。
\0	查找 NUL 字符。
\n	查找换行符。
\f	查找换页符。
\r	查找回车符。
\t	查找制表符。
\v	查找垂直制表符。
\xxx	查找以八进制数 xxx 规定的字符。
\xdd	查找以十六进制数 dd 规定的字符。
\uxxxx	查找以十六进制数 xxxx 规定的 Unicode 字符。

量词

量词	描述
<u>n+</u>	匹配任何包含至少一个 n 的字符串。
<u>n*</u>	匹配任何包含零个或多个 n 的字符串。
<u>n?</u>	匹配任何包含零个或一个 n 的字符串。
<u>n{X}</u>	匹配包含 X 个 n 的序列的字符串。
<u>n{X,Y}</u>	匹配包含 X 或 Y 个 n 的序列的字符串。
<u>n{X,}</u>	匹配包含至少 X 个 n 的序列的字符串。
<u>n\$</u>	匹配任何结尾为 n 的字符串。
<u>^n</u>	匹配任何开头为 n 的字符串。
<u>?=n</u>	匹配任何其后紧接指定字符串 n 的字符串。
<u>?!n</u>	匹配任何其后没有紧接指定字符串 n 的字符串。

RegExp 对象属性

属性	描述
<u>global</u>	RegExp 对象是否具有标志 g 。
<u>ignoreCase</u>	RegExp 对象是否具有标志 i 。
<u>lastIndex</u>	一个整数，标示开始下一次匹配的字符位置。
<u>multiline</u>	RegExp 对象是否具有标志 m 。
<u>source</u>	正则表达式的源文本。

RegExp 对象方法

方法	描述
<u>compile</u>	编译正则表达式。
<u>exec</u>	检索字符串中指定的值。返回找到的值，并确定其位置。
<u>test</u>	检索字符串中指定的值。返回 true 或 false 。

支持正则表达式的 **String** 对象的方法

方法	描述
<u>search</u>	检索与正则表达式相匹配的值。
<u>match</u>	找到一个或多个正则表达式的匹配。
<u>replace</u>	替换与正则表达式匹配的子串。
<u>split</u>	把字符串分割为字符串数组。

Number 对象

Number 对象是原始数值的包装对象。

Number 对象属性

属性

[constructor](#)

[MAX_VALUE](#)

[MIN_VALUE](#)

[NaN](#)

[NEGATIVE_INFINITY](#)

[POSITIVE_INFINITY](#)

prototype

描述

返回对创建此对象的 Number 函数的引用。

可表示的最大的数。

可表示的最小的数。

非数字值。

负无穷大，溢出时返回该值。

正无穷大，溢出时返回该值。

使您有能力向对象添加属性和方法。

Number 对象方法

方法

[toString](#)

[toLocaleString](#)

[toFixed](#)

[toExponential](#)

[toPrecision](#)

[valueOf](#)

描述

把数字转换为字符串，使用指定的基数。

把数字转换为字符串，使用本地数字格式顺序。

把数字转换为字符串，结果的小数点后有指定位数的数字。

把对象的值转换为指数计数法。

把数字格式化为指定的长度。

返回一个 Number 对象的基本数字值。

String 对象

String 对象用于处理文本（字符串）。`var s1=new String("abc")`

String 对象属性

属性	描述
constructor	对创建该对象的函数的引用
length	字符串的长度
prototype	允许您向对象添加属性和方法

String 对象方法

方法	描述
anchor()	创建 HTML 锚。
big()	用大号字体显示字符串。
blink()	显示闪动字符串。
bold()	使用粗体显示字符串。
charAt()	返回在指定位置的字符。
charCodeAt()	返回在指定的位置的字符的 Unicode 编码。
concat()	连接字符串。
fixed()	以打字机文本显示字符串。
fontcolor()	使用指定的颜色来显示字符串。
fontsize()	使用指定的尺寸来显示字符串。
fromCharCode()	从字符编码创建一个字符串。
indexOf()	检索字符串。

方法	描述
italics()	使用斜体显示字符串。
lastIndexOf()	从后向前搜索字符串。
link()	将字符串显示为链接。
localeCompare()	用本地特定的顺序来比较两个字符串。
match()	找到一个或多个正则表达式的匹配。
replace()	替换与正则表达式匹配的子串。
search()	检索与正则表达式相匹配的值。
slice()	提取字符串的片断，并在新的字符串中返回被提取的部分。
small()	使用小字号来显示字符串。
split()	把字符串分割为字符串数组。
strike()	使用删除线来显示字符串。
sub()	把字符串显示为下标。
substr()	从起始索引号提取字符串中指定数目的字符。
substring()	提取字符串中两个指定的索引号之间的字符。
sup()	把字符串显示为上标。
toLocaleLowerCase()	把字符串转换为小写。
toLocaleUpperCase()	把字符串转换为大写。
toLowerCase()	把字符串转换为小写。
toUpperCase()	把字符串转换为大写。
toSource()	代表对象的源代码。
toString()	返回字符串。
valueOf()	返回某个字符串对象的原始值。

cookie对象

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
function getCookie(c_name){
    if (document.cookie.length>0) {
        c_start=document.cookie.indexOf(c_name + "=")
        if (c_start!=-1) {
            c_start=c_start + c_name.length+1
            c_end=document.cookie.indexOf(";",c_start)
            if (c_end==-1) c_end=document.cookie.length
            return unescape(document.cookie.substring(c_start,c_end))
        }
    }
    return ""
}
>
```

```

function setCookie(c_name,value,expiredays){
    var exdate=new Date()
    exdate.setDate(exdate.getDate()+expiredays)
    document.cookie=c_name+ "=" +escape(value)+
        ((expiredays==null) ? "" : ";expires="+exdate.toGMTString())
}
function checkCookie(){
    username=getCookie('username')
    if (username!=null && username!="")
        {alert('Welcome again '+username+'!')}
    else {
        username=prompt('Please enter your name:', "")
        if (username!=null && username!="") {
            setCookie('username',username,365)
        }
    }
}
</script>
</head>
<body onLoad="checkCookie()">
</body>
</html>

```

DOM对象

[Document](#)

[Anchor](#)

[Area](#)

[Base](#)

[Body](#)

[Button](#)

[Canvas](#)

[Event](#)

[Form](#)

[Frame](#)

[Frameset](#)

[IFrame](#)

[Image](#)

[Input Button](#)

[Input Checkbox](#)

[Input File](#)

[Input Hidden](#)

[Input Password](#)

[Input Radio](#)

[Input Reset](#)

[Input Submit](#)

[Input Text](#)

[Link](#)

[Meta](#)

[Object](#)

[Option](#)

[Select](#)

[Style](#)

[Table](#)

[TableCell](#)

[TableRow](#)

[Textarea](#)