Javascript

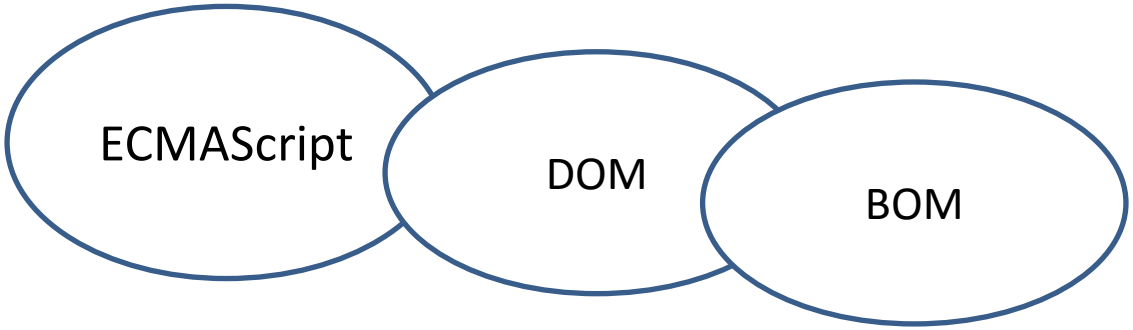
2016.1.13

isszym sysu.edu.cn

概述

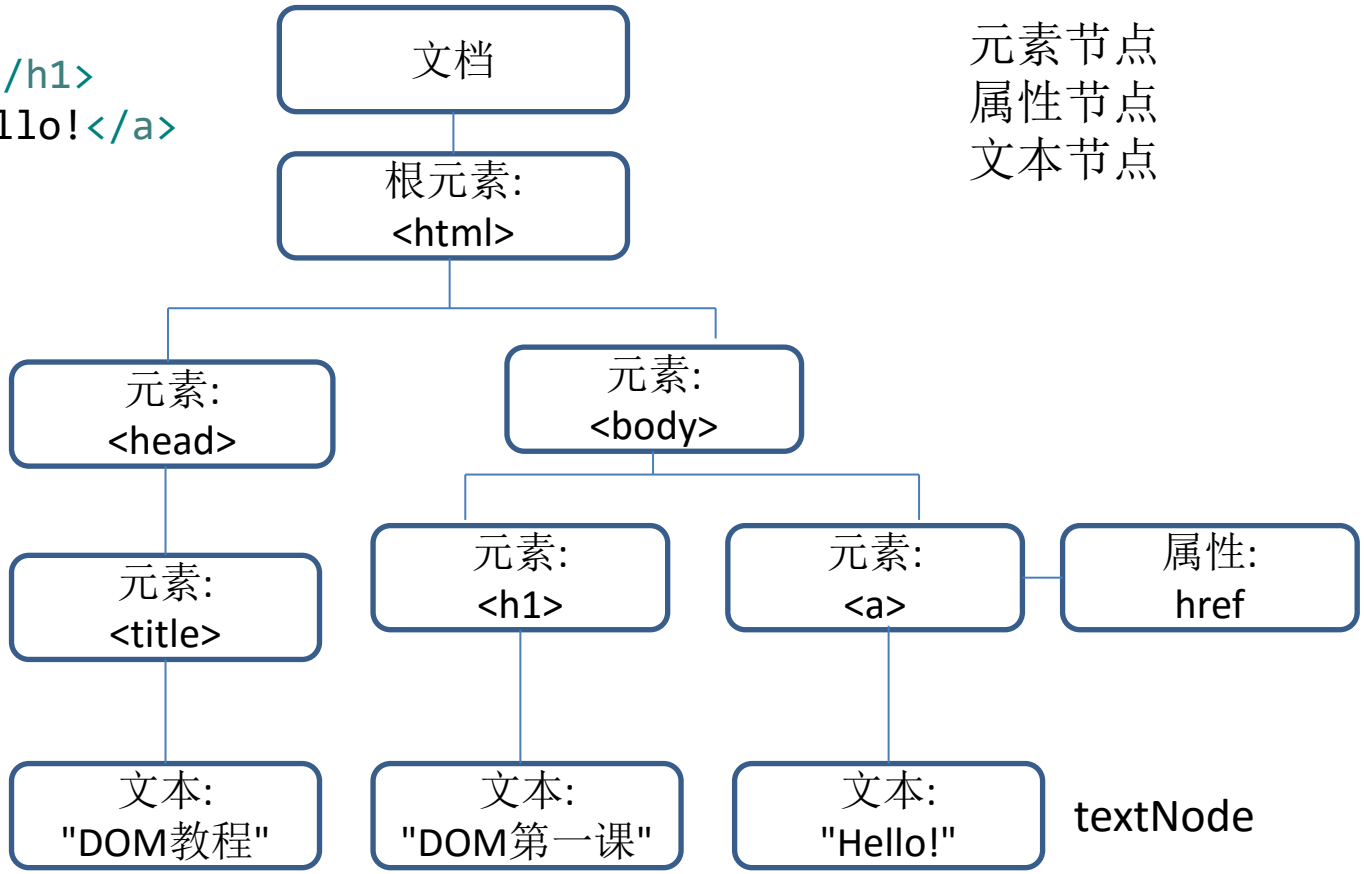
- 1995 年，Netscape Navigator 2.0 增加了一个由Netscape 与 Sun开发的称为 LiveScript 的脚本语言，当时的目的是在浏览器和服务器端使用它。在 Netscape Navigator 2.0 即将正式发布前，Netscape将其更名为 JavaScript，目的是为了利用 Java 这个因特网时髦词汇。
- 因为 JavaScript 1.0 如此成功，Netscape 在 Netscape Navigator 3.0 中发布了 1.1 版。此时微软发布了 IE 3.0 并搭载了一个 JavaScript 的克隆版，叫做 Jscript。这样命名是为了避免与 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 CEnvi 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，这个语言的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

first.html

```
<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>
```

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

TypeTrans.html

```
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)
```

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：　~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　x>>=a　x>>>=a　x<<=a

运算优先级：

高 []() →单目→ * / % → + - →<< >> >>>
→ < > <= >= →== != → & →^
→ | →&& →|| →三目运算 →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");//"Hellollo"
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));          // 返回 20
var asum = sum;             // 函数为对象，可以直接赋值
alert(asum(10,10));         // 返回20
sum = null;                 //  清除对象sum
alert(asum(10,10));         // 返回20。asum依然可用
asum=function(num1,num2){  // 再赋值一个函数给asum
    return num1-num2;
}
alert(asum(10,10));         //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!");
```

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)};
              };
```

还可以是对象

或

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

- 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Javasrcipt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

javascript1.jsp

```
<!DOCTYPE html>
<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)">
                                    <!--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>
```

Click

- 定义事件处理程序的第二种方法

javascript2.jsp

```
<!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>
```

要放在元素p1的后面，因为函数外的语句会顺序会执行。

javascript2a.jsp

```
<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);   // false表示事件在冒泡阶段触发
```

addEventListener的参数：

第一个参数为要捕捉的事件

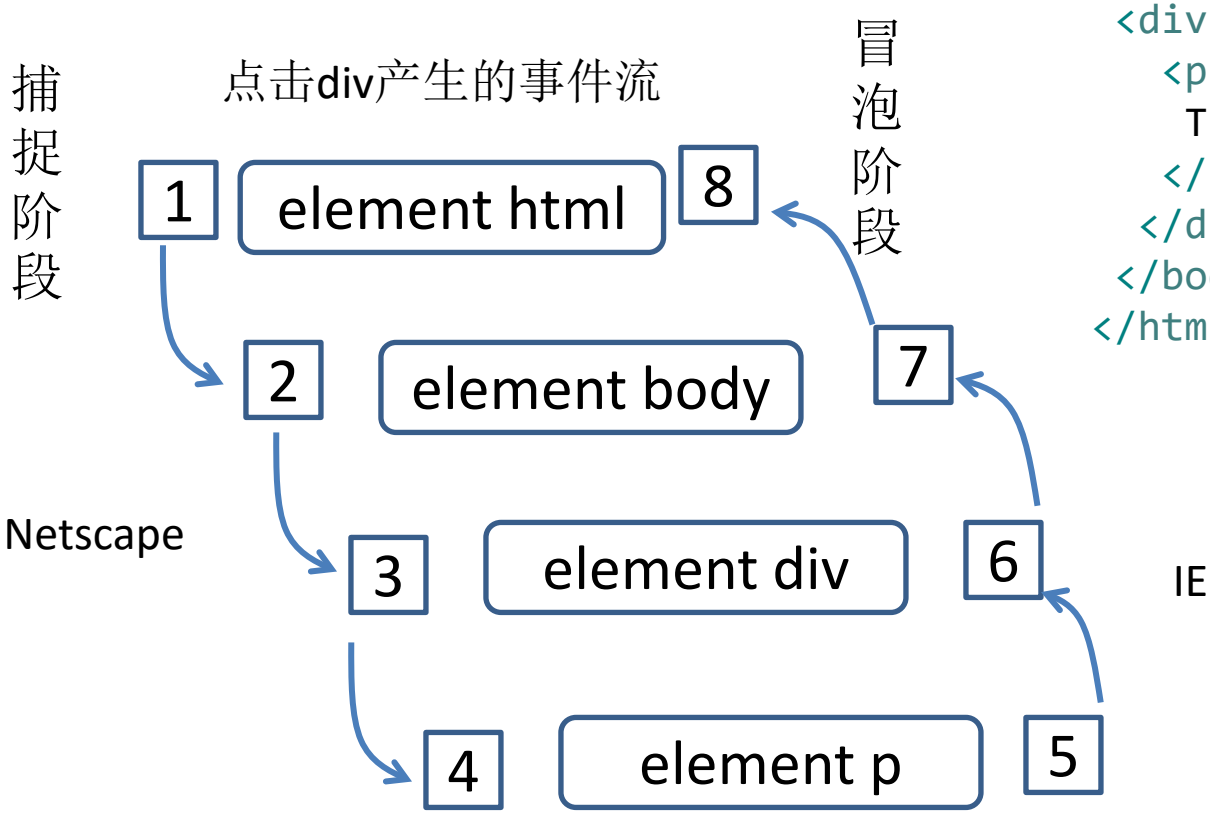
第二个参数为捕捉到事件后执行的函数

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。

其它例子：

```
p1.addEventListener("click",function(){changetext(this);});
p1.addEventListener("click",function(event){
                                alert(event.type+":"
                                +event.clientX+","+event.clientY
                                +event.screenX+","+event.screenY);
                              }
                    );
```

```
<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。

events

```
<!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);          //冒泡阶段
 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);           //捕捉阶段
 body.addEventListener("click", function() {alert("body");}, true);
 div.addEventListener("click", function() {alert("div");}, true);
 html.addEventListener("click", function() {alert("html");}, true);
</script>
</html>
```

- 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",hander, false);
btn.removeEventListener("click",hander, 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

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


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

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之前的版本用于取消默认行为

window.event.cancelBubble = true;// IE9之前的版本用于阻止事件传播：

• 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)">
```

- event.type的取值：
  - ✓用户界面事件：load，unload，abort error，select，resize
  - ✓焦点事件：blur(失焦点)，focus(得到焦点)

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

- load事件： 当页面或图像完全加载后发生。

```
window.onload = function(){alert("Hello!")}
<body  onload = "alert('Hello!')">
<img  onload = "alert('Hello!')">
```

* focusin和focusout分别为通过冒泡方式得到和失去焦点事件。

onclick	元素被点击
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.propertyName设置或返回元素的变化了的属性的名称）

•鼠标事件

- 事件列举：

```
<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

- 事件属性：

event.screenX, event.screenY	鼠标在窗口的坐标
event.clientX,event.clientY	鼠标在客户区的坐标
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	鼠标移动离开的对象
event.toElement	鼠标移动进入的对象
event.srcElement	触发事件(鼠标点击)的元素
event.target	冒泡事件中的最小范围元素
event.currentTarget	当前事件的元素（同this）

•其它事件

✓ 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（扫描码）
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 。
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	两个手指均触摸时发生
gesturechange	至少有一个手指位置发生变化。
gestureend	至少有一个手指从触摸屏移开。
event.scale	两指间距离,从1开始，值越大距离越大
event.oritation	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	方向：0-肖像模式 90-左旋90度 90-右旋90度
devicemotion	设备正在移动

```
<!DOCTYPE  html>
<html>
<body>
        <p id="myBtn">第一段</p>
        <p>第二段</p>
        <p>第三段</p>
        <script>
            var btn = document.getElementById("myBtn");
            var hander = function (event) {
                switch (event.type) {
                    case "click":
                        alert("Clicked"); break;                          //点击了myBtn之后显示Clicked
                    case "mouseover":
                        event.target.style.color = "red"; break;          //这里的this===event.target
                    case "mouseout":
                        event.target.style.color = "";
                }
            };
            btn.onclick = hander;
            btn.onmouseover = hander;
            btn.onmouseout = hander;
            document.body.onclick = function (event) {                    //点击了myBtn之后
                alert(event.currentTarget === document.body);             //true
                alert(this === document.body);                            //true
                alert(event.target === event.currentTarget);              //false
                alert(event.target === document.getElementById("myBtn")); //true
            };
        </script>
</body>
</html>
```

通过事件提交表单

submit.html

```
<%@ page language="java" import="java.util.*"  contentType="text/html; charset=utf-8"%>
<% request.setCharacterEncoding("utf-8");%>
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
   function submit1() {
      var p1 = document.getElementById("p1");
      if (p1.value == "") {
         alert("姓名不能为空!"); return false;
      }
      else {
         document.forms["p3"].submit(); //或document.p3.submit()
         return true;
      }
   }
</script>
</head>
<body>
<!--没有action会提交给当前url -->
<form name="p3" method="post" action="post.jsp">
  <p id="p4">hello world!</p>
  <input id="p1" name="p1" type="text" value="Hello!" />
  <input id="p2" type="submit" value="showHTML" />
</form>
<script type="text/javascript">
     p2 = document.getElementById("p2");
     p2.addEventListener("click", submit1);  // 也可以用 “function(){submit();}”代替submit.   见 submit2.html
</script>                                    // 利用form的submit事件也可以调用函数提交(返回true) 见 submit3.html
</body>
</html>
```

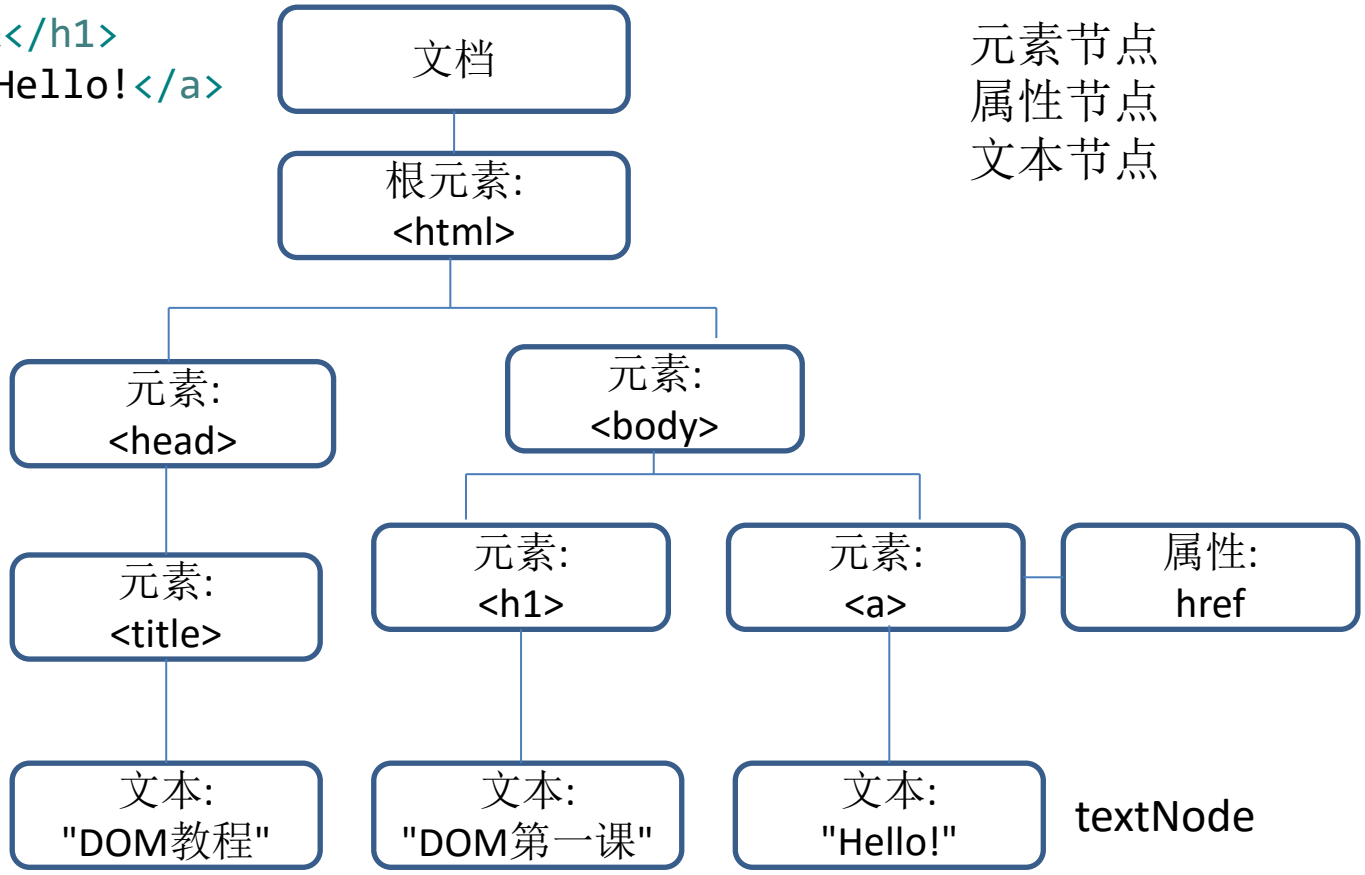
post.jsp

```
<%@ page language="java"  import="java.util.*"
         contentType="text/html; charset=utf-8"
%><%
  request.setCharacterEncoding("utf-8");
%><%
  String p1=request.getParameter("p1");
  out.print("Get:"+p1);
%>
```

DOM文档树

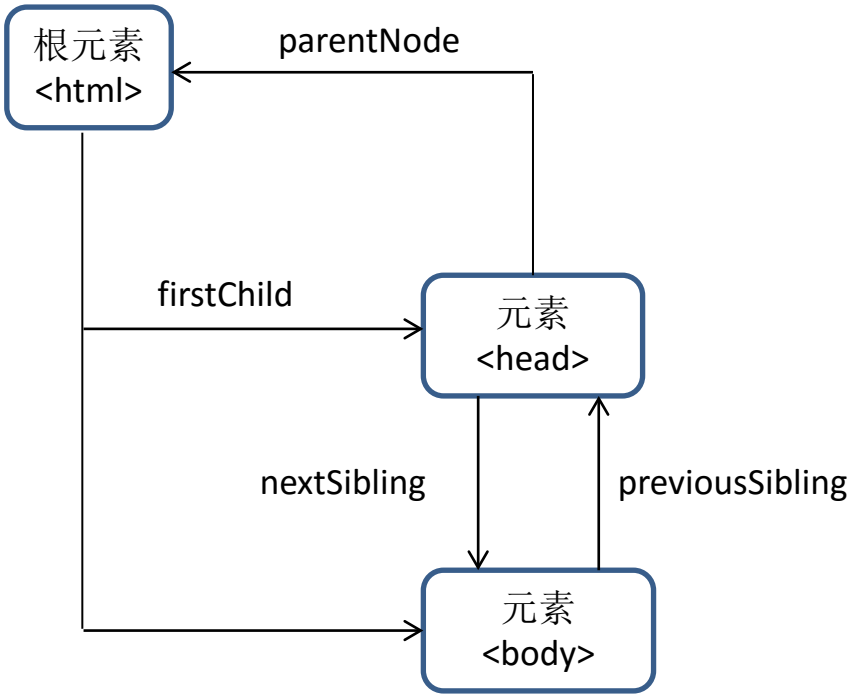
```
<html>
<head>
   <title>DOM 教程</title>
</head>
   <body>
       <h1>DOM 第一课</h1>
       <a href="#" >Hello!</a>
    </body>
</html>
```

每个节点都是一个对象



元素节点
属性节点
文本节点

textNode




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

var s = document.getElementById("t1");
alert(s.parentNode.nodeName);
```

nodeName 返回节点名：标签名(元素节点)或属性名(属性节点)或#text(文本节点)或#document(文档节点)

nodeType 取到节点类型：1-元素 2-属性 3-文本 8-注释 9-文档

nodeValue 取到节点值：文本(文本节点)，属性包含属性值(属性节点)，null(文档节点和元素节点)

innerHTML 取到节点（元素）的内容。**outerHTML**，取到本元素(包含内容)。**innerText** 类似innerHTML，只是当成文本返回，只有IE支持。**outerText** 类似outerHTML，只是当成文本返回，只有IE支持。

childNodes 会返回所有子节点。children类似，只是不会返回TextNode。

attributes 取到节点（元素）的所有属性

className 获取节点的属性class的值

classList 获得节点的所有类名。可以用以下方法修改它：add(增加类)，remove（删除类），toggle（切换类），contains（是否包含一个类）(HTML 5支持)

data- 自定义属性名（ HTML5支持）

* 当元素内容为空格时， IE10之前的版本不会返回文本节点。为了统一所有浏览器的行为，html5为节点重新定义了一套遍历节点的属性：firstElementChild，lastElementChild，nextElementSibling，preivousElementSibling，childElementCount。

获得元素的方法

除了通过从一个元素遍历可以取到所有元素（实际上还可以取到属性节点和文本节点），还可以通过document 的一些方法得到所需的元素：

getElementById()	得到具有某个id值的元素。
getElementsByTagName()	返回包含带有指定标签名称的所有元素（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(数组)。html5
querySelectorAll()	取得与CSS选择符匹配的所有元素（数组）。html5
querySelector()	取得与CSS选择符匹配的第一个节点。html5

```
var s = document.querySelectorAll("a");  //返回文档中所有的a元素(数组)
document.write(s.length);
```

修改文档树的方法

通过对节点数据域（例如，nodeValue，innerHTML和outerHTML）赋值，或者直接通过修改元素属性和样式，可以改变文档树。通过增删改子节点等方法，还可以直接改变文档树。

javascript6.jsp

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
  function changetext() {
    var oTest = document.getElementById("p1");
    var newNode1 = document.createElement("br");
    // null表示前面没有子节点，即插入到最前面
    oTest.parentNode.insertBefore(newNode1, null);
    var newNode2 = oTest.cloneNode(true);
    oTest.parentNode.insertBefore(newNode2, null);
  }
</script>
</head>
<body>
<input id="p1" type="text" value="Hello!"/>
<input id="p2" type="button" value="showHTML"/>
<script type="text/javascript">
  var p1 = document.getElementById("p1");
  p1.addEventListener("click", function () {
      changetext(); });
</script>
</body>
</html>
```

appendChild()	把新的子节点添加到指定节点。
removeChild()	删除子节点。
replaceChild()	替换子节点。
insertBefore()	在指定的子节点前面插入新的子节点。
createAttribute()	创建属性节点。
createElement()	创建元素节点。
createTextNode()	创建文本节点。
cloneNode()	克隆节点。参数：是否克隆属性
focus()	让输入类元素得焦点 htm5。
scrollIntoView()	滚动页面让元素可见；
contains()	是否为后代元素；
insertAdjacentHTML()	在元素的开始标签和结束标签的前后插入元素。
compareDocumentPosition()	比较位置：1-无关 2-居前 3-居后 8-包含 16-被包含 。

修改元素属性

通过属性节点的nodeValue可以获得或修改属性值，通过元素的方法getAttribute(name)和setAttribute(name,value)可以获得或设置属性值，还可以把属性名作为元素的数据域获得或修改属性值。

```
<div id="div1" class="hd">
   <img id="logo" />
   <a id="a1" href="http://www.sysu.edu.cn">中大</a>
</div>
var   div = document.getElementById("div1");
var   img = document.getElementById("logo");
var   a = document.getElementById("a1");
//取得元素特性
alert(div.id);
alert(div.getAttribute("id"));
alert(div.attributes["id"].nodeValue);
alert(div.className);              //不用div.class是因为class是保留关键字
alert(div.getAttribute("class")); //注意这里是class而不是className
//设置元素特性
div.id = "div2";
 div.setAttribute("id", "div2");
 div.attributes["id"].nodeValue = "div2";
 img.src = "images/img1.gif";

div.className = "ft";
div.setAttribute("class", "ft");
```

```
//移除元素特性
 div.removeAttribute("class");
div.attributes.removeNamedItem("class");

a.href = "http://www.sina.com.cn";    //重新设置超链接
div.title = "header";                 //title改为"header"
a.innerHTML = "华工";                 //"中大"改为"华工"
```

修改元素样式

- 想修改元素的样式可以通过修改样式文件名、修改style属性值、修改原子样式的值和修改类名来实现。文档中的每个样式表定义(外部css文件和style元素)都可以取到和修改。

```
<p  id="hello" class="c1">Hello World!</p>
<link rel = "stylesheet" type="text/css" id="css" href="firefox.css" />
<span onclick="javascript:document.getElementById('css').href = 'ie.css'"> <!--改变样式文件名-->
      点我改变样式
</span>
<script>
        document.getElementById(" hello").style.color="blue";  //修改原子样式
        document.getElementById(" hello").className="c2";     //改变类属性值
        var p1 = document.getElementById('hello');
        p1.setAttribute("style", "color:blue");                    //style属性
        p1.setAttribute("class", "c2  c3");                        //改变类属性值
       // 设置样式表（外部css文件、style元素）的全部内容
       document.styleSheets[0].cssText = document.styleSheets[0].cssText.replace(/red/g, "yellow");
</script>
```

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 () {  // 当http请求的状态变化时执行
       if (xmlhttp.readyState == 4) {           //  4-收到http响应
         if (xmlhttp.status >= 200 && xmlhttp.status < 300 // http响应的状态：
             || xmlhttp.status >= 304) {     // 200-OK，304-unmodified
           alert(xmlhttp.responseText);  // http响应的正文
           var oTest = document.getElementById("p4");
           oTest.innerHTML = xmlhttp.responseText;
         }
         else {
           alert("error");
         }
       };
     };
     // 打开http请求（open）的参数：get或post，url，是否异步发送
     xmlhttp.open("get", "ajaxGet.jsp", true);
     xmlhttp.send(null); //发送http请求。参数为要传送的输入值，null表示没有。
   }
</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。

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(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

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。
……

设定超时时间

```
xmlhttp.timeout=1000;    //ms
xmlhttp.ontimeout=function(){
     alert("no reponse received in a second.")
};
```

Ajax技术(post)

javasrcipt8.jsp

```
<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=" + encodeURIComponent(v1) + "&p2=345";//汉字需要编码
      xmlhttp.open("post", "ajaxPost.jsp", true);
      xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
      xmlhttp.send(param);   // 没有参数就用null。
   }
</script>
```

```
<form name="p3">
   <p id="p4">hello world!</p>
   <input id="Text1" 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>
```

ajaxPost.jsp

```
<%@ page language="java" import="java.util.*"  contentType="text/html; charset=utf-8"%>
<% request.setCharacterEncoding("utf-8");%>
<%     String p1=request.getParameter("p1");
       String p2=request.getParameter("p2");
       out.print("Get:"+p1+" "+p2);
%>
```

早期Ajax技术是采用一个隐藏的iframe实现的。通过改变iframe的src来生成http请求，当收到http响应时会触发onload事件，此时可以取到http响应的正文。

javascript10.jsp

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
 function LoadContent(src, iframeID) {
       var content = document.getElementById(iframeID);
       content.setAttribute('src', src);
 }
 function swipContent(src, to) {
      var bobo=document.getElementById(src);
      var content=bobo.contentDocument?bobo.contentDocument.body.innerHTML
                  :bobo.Document.body.innerHTML;
   document.getElementById(to).innerHTML = content;
 }
</script>
</head>
<body>
   <a href="#" onclick="LoadContent('ajaxGet.jsp','icontent')">Link Button</a>
  <iframe id="icontent" width="600" height="0" scrolling="auto" frameborder="0"
        onload="swipContent('icontent','dest')"></iframe>
  <div id="dest" />
</body>
</html>
```

串行化

串行化就是把form中每个输入字段的值全部取到并形成如下key-value格式的数据：

x1=v1&x2=v2&x3=v3...

如果x和v使用了汉字，则要使用函数encodeURIComponent()进行编码。采用ajax技术就可以把这个串行化的数据提交给后台处理。

```
function serialize(form){
  var parts = [], field = null,i,len,j,optLen,option,optValue;
  for (i=0, len=form.elements.length; i < len; i++){
        field = form.elements[i];
        switch(field.type){
              ...

         }//switch
    }//for
    return parts.join("&");
 }//function
```

javascript11.jsp

```
switch(field.type){
    case "select-one":
    case "select-multiple":
        if (field.name.length){
                    for (j=0, optLen = field.options.length; j < optLen; j++){
                        option = field.options[j];
                        if (option.selected){
                              optValue = "";
                              if (option.hasAttribute){
                                 optValue = (option.hasAttribute("value")?option.value:option.text);
                              }
                              else {
                                 optValue = (option.attributes["value"].specified?option.value:option.text);
                              } //if
                              parts.push(encodeURIComponent(field.name) + "=" + encodeURIComponent(optValue));
                        } //if
                     } //for
        } //if
        break;
    case undefined:              //字段集
    case "file":                 //文件输入
    case "submit":               //提交按钮
    case "reset":                //重置按钮
    case "button":               //自定义按钮
        break;
    case "radio":                //单选按钮
    case "checkbox":             //复选框
        if (!field.checked){ break;}
    /* 执行默认操作 */
    default:
       //不包含没有名字的表单字段
       if (field.name.length){
          parts.push(encodeURIComponent(field.name) + "=" + encodeURIComponent(field.value));
       }
}//switch
```

富文本编辑

富文本编辑可以在输入框中直接设置文本颜色、字体大小、显示图片等。有两种实现富文本编辑的方法，一种是通过设置iframe的document.designMode 为on就可以使用富文本编辑。

richEdit10.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Blank Page for Rich Text Editing</title>
</head>
<body>
  <iframe name="edit" style="height:200px;width:400px;">
  </iframe><br>
  <input type="button" value="加粗" onclick="bold()">
  <script>
     frames["edit"].document.designMode = "on";
     function bold() {
         frames["edit"].document.execCommand("bold",
                                              false, null);
     }
  </script>
</body>
</html>
```

另一种实现富文本编辑的方法是通过设置div的属性contenteditable的值为true。

```
<!DOCTYPE html>
<html>
<head>
    <title>Rich Text Editing</title>
</head>
<body>
    <div class="editable" id="richedit" contenteditable="true"
                style="padding:10px;width:400px;height:200px;border:solid 1px black"></div><br>
    <input type="button" value="斜体" onclick="bold()">
    <script>
        function bold() {
            document.execCommand("italic", false, null);
        }
    </script>
</body>
</html>
```

定时器

语句setTimeOut(f, t)用于在t毫秒之后执行一次函数f。如果每个t毫秒都要执行一次f，可以在函数f中执行setTimeOut(f, t)。 setTimeOut(f, t)返回的值为id，可以用于停止执行clearTimeout(id)。也可以使用setInterval(f, t)实现周期性执行函数f的功能。clearInterval(id)可以用setInterval返回的id停止周期性执行函数f。

timer.html

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
        var i = 0;
        function hello() {
           i++;
           var ctrl = document.getElementById("timer");
           ctrl.innerHTML = i;
           window.setTimeout(hello, 1000);
        }
         var id=setTimeout(hello, 1000);
</script>
</head>
<body>
        timer:<span id="timer">0</span>
</body>
<html>
```

* 采用setTimeout实现定时器不会因为每次执行时间超出间隔时间而出现异常失常。

cookie对象

下面的网页在加载之后立即检查是否存在username的cookie，如果存在，则取出其值，如果不存在，则要求输入值然后创建该cookie。

cookie.html

```
<!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() {
                var 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 >
```

Javascript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/

•对象说明

通过location.href可以取到当前浏览器地址栏中的地址(URL)，通过location.hostname和location.port可以取到URL中的主机名和端口号。执行history.back()、 history.forward()和history.go(-2)可以回退一页、前进一页和回退两页。

window对象还可以用来确定浏览器窗口客户区大小，详细做法见附录。执行window.open(" http://www.w3school.com.cn ")可以用来打开一个新窗口并控制其外观，还可以用来控制打开窗口的外观，例如，

window.open("http://www.w3school.com.cn","_blank","toolbar=yes, location=yes, directories=no, status=no, menubar=yes, scrollbars=yes, resizable=no, copyhistory=yes, width=400, height=400")

-- toolbar=yes, location=yes , menubar=yes , scrollbars=yes表示浏览器显示工具栏、地址栏

-- menubar=yes , scrollbars=yes表示浏览器显示菜单和滚动条

-- directories=no, status=no, resizable=no表示不显示收藏夹、状态栏，不可变化大小

document对象可以取到当前网页的内容。 document.anchors[] 、document.forms[]和document.images[]可以得到当前网页的所有a元素、所有form和所有img元素。document.referrer可以得到载入当前网页的网页的 URL。 document。 document. write("...")可以直接往网页中输出一个字符串。

采用document的方法如何获得元素，见前面“获取元素的方法” 。

```
var s = document.querySelectorAll("a");
 document.write(s.length);
```

日期操作采用Date对象，数学函数采用Math对象，数据值转换使用Number对象， RegExp对象用于操作正则表达式，文档中很多元素都属于DOM对象，具体内容见附录。

附录

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	描述
length	返回浏览器历史列表中的 URL 数量。

History 对象方法

方法	描述
back()	加载 history 列表中的前一个 URL。
forward()	加载 history 列表中的下一个 URL。
go()	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。

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

方法	描述
alert()	显示带有一段消息和一个确认按钮的警告框。
blur()	把键盘焦点从顶层窗口移开。
clearInterval()	取消由 setInterval() 设置的 timeout。
clearTimeout()	取消由 setTimeout() 方法设置的 timeout。
close()	关闭浏览器窗口。
confirm()	显示带有一段消息以及确认按钮和取消按钮的对话框。
createPopup()	创建一个 pop-up 窗口。
focus()	把键盘焦点给予一个窗口。
moveBy()	可相对窗口的当前坐标把它移动指定的像素。
moveTo()	把窗口的左上角移动到一个指定的坐标。
open()	打开一个新的浏览器窗口或查找一个已命名的窗口。
print()	打印当前窗口的内容。
prompt()	显示可提示用户输入的对话框。
resizeBy()	按照指定的像素调整窗口的大小。
resizeTo()	把窗口的大小调整到指定的宽度和高度。
scrollBy()	按照指定的像素值来滚动内容。
scrollTo()	把内容滚动到指定的坐标。
setInterval()	按照指定的周期（以毫秒计）来调用函数或计算表达式。
setTimeout()	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
all[]	提供对文档中所有 HTML 元素的访问。	4	1	9	No
anchors[]	返回对文档中所有 Anchor 对象的引用。	4	1	9	Yes
applets	返回对文档中所有 Applet 对象的引用。	-	-	-	-
forms[]	返回对文档中所有 Form 对象引用。	4	1	9	Yes
images[]	返回对文档中所有 Image 对象引用。	4	1	9	Yes
links[]	返回对文档中所有 Area 和 Link 对象引用。	4	1	9	Yes

属性	描述	IE	F	O	W3C
body	提供对 <body> 元素的直接访问。 对于定义了框架集的文档，该属性引用最外层的 <frameset>。				
cookie	设置或返回与当前文档有关的所有 cookie。	4	1	9	Yes
domain	返回当前文档的域名。	4	1	9	Yes
lastModified	返回文档被最后修改的日期和时间。	4	1	No	No
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 document.open() 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 document.write() 或 document.writeln() 方法的输出。
write()	向文档写 HTML 表达式 或 JavaScript 代码。
writeln()	等同于 write() 方法，不同的是在每个表达式之后写一个换行符。

html5

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

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

事件：

window.onload和document.ready的不同：一个要所有页面对象加载完毕加载完毕才发生，一个只要页面加载完毕就发生。 http://www.cnblogs.com/a546558309/p/3478344.html

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
availHeight	返回显示屏幕的高度 (除 Windows 任务栏之外)。
availWidth	返回显示屏幕的宽度 (除 Windows 任务栏之外)。
bufferDepth	设置或返回调色板的比特深度。
colorDepth	返回目标设备或缓冲器上的调色板的比特深度。
deviceXDPI	返回显示屏幕的每英寸水平点数。
deviceYDPI	返回显示屏幕的每英寸垂直点数。
fontSmoothingEnabled	返回用户是否在显示控制面板中启用了字体平滑。
height	返回显示屏幕的高度。
logicalXDPI	返回显示屏幕每英寸的水平方向的常规点数。
logicalYDPI	返回显示屏幕每英寸的垂直方向的常规点数。
pixelDepth	返回显示屏幕的颜色分辨率（比特每像素）。
updateInterval	设置或返回屏幕的刷新率。
width	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
appCodeName	返回浏览器的代码名。
appMinorVersion	返回浏览器的次级版本。
appName	返回浏览器的名称。
appVersion	返回浏览器的平台和版本信息。
browserLanguage	返回当前浏览器的语言。
cookieEnabled	返回指明浏览器中是否启用 cookie 的布尔值。
cpuClass	返回浏览器系统的 CPU 等级。
onLine	返回指明系统是否处于脱机模式的布尔值。
platform	返回运行浏览器的操作系统平台。
systemLanguage	返回 OS 使用的默认语言。
userAgent	返回由客户机发送服务器的 user-agent 头部的值。
userLanguage	返回 OS 的自然语言设置。

Navigator 对象方法

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

Global

函数	描述
decodeURI()	解码某个编码的 URI。
decodeURIComponent()	解码一个编码的 URI 组件。
encodeURI()	把字符串编码为 URI。
encodeURIComponent()	把字符串编码为 URI 组件。
escape()	对字符串进行编码。
eval()	计算 JavaScript 字符串，并把它作为脚本代码来执行。
getClass()	返回一个 JavaObject 的 JavaClass。
isFinite()	检查某个值是否为有穷大的数。
isNaN()	检查某个值是否是数字。
Number()	把对象的值转换为数字。
parseFloat()	解析一个字符串并返回一个浮点数。
parseInt()	解析一个字符串并返回一个整数。
String()	把对象的值转换为字符串。
unescape()	对由 escape() 编码的字符串进行解码。

属性	描述
Infinity	代表正的无穷大的数值。
java	代表 java.* 包层级的一个 JavaPackage。
NaN	指示某个值是不是数字值。
Packages	根 JavaPackage 对象。
undefined	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

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

Math 对象属性

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

Math 对象方法

方法	描述
abs(x)	返回数的绝对值。
acos(x)	返回数的反余弦值。
asin(x)	返回数的反正弦值。
atan(x)	以介于 -PI/2 与 PI/2 弧度之间的数值来返回 x 的反正切值。
atan2(y,x)	返回从 x 轴到点 (x,y) 的角度（介于 -PI/2 与 PI/2 弧度之间）。
ceil(x)	对数进行上舍入。
cos(x)	返回数的余弦。
exp(x)	返回 e 的指数。
floor(x)	对数进行下舍入。
log(x)	返回数的自然对数（底为e）。
max(x,y)	返回 x 和 y 中的最高值。
min(x,y)	返回 x 和 y 中的最低值。
pow(x,y)	返回 x 的 y 次幂。
random()	返回 0 ~ 1 之间的随机数。
round(x)	把数四舍五入为最接近的整数。
sin(x)	返回数的正弦。
sqrt(x)	返回数的平方根。
tan(x)	返回角的正切。
toSource()	返回该对象的源代码。
valueOf()	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
n+	匹配任何包含至少一个 n 的字符串。
n*	匹配任何包含零个或多个 n 的字符串。
n?	匹配任何包含零个或一个 n 的字符串。
n{X}	匹配包含 X 个 n 的序列的字符串。
n{X,Y}	匹配包含 X 或 Y 个 n 的序列的字符串。
n{X,}	匹配包含至少 X 个 n 的序列的字符串。
n$	匹配任何结尾为 n 的字符串。
^n	匹配任何开头为 n 的字符串。
?=n	匹配任何其后紧接指定字符串 n 的字符串。
?!n	匹配任何其后没有紧接指定字符串 n 的字符串。

RegExp 对象方法

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

RegExp 对象属性

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

支持正则表达式的 String 对象的方法

方法	描述
search	检索与正则表达式相匹配的值。
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。
split	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	描述
constructor	返回对创建此对象的 Number 函数的引用。
MAX_VALUE	可表示的最大的数。
MIN_VALUE	可表示的最小的数。
NaN	非数字值。
NEGATIVE_INFINITY	负无穷大，溢出时返回该值。
POSITIVE_INFINITY	正无穷大，溢出时返回该值。
prototype	使您有能力向对象添加属性和方法。

Number 对象方法

方法	描述
toString	把数字转换为字符串，使用指定的基数。
toLocaleString	把数字转换为字符串，使用本地数字格式顺序。
toFixed	把数字转换为字符串，结果的小数点后有指定位数的数字。
toExponential	把对象的值转换为指数计数法。
toPrecision	把数字格式化为指定的长度。
valueOf	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。
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。

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

TextArea要用.value改变值，
innerHTML在chrome中无效

富文本编辑框命令

下表列出了那些被支持最多的命令:

命 令 值	（第三个参数）	说 明
backcolor	颜色字符串	设置文档的背景颜色
bold	null	将选择的文本转换为粗体
copy	null	将选择的文本复制到剪贴板
createlink	URL字符串	将选择的文本转换成一个链接，指向指定的URL
cut	null	将选择的文本剪切到剪贴板
delete	null	删除选择的文本
fontname	字体名称	将选择的文本修改为指定字体
fontsize	1～7	将选择的文本修改为指定字体大小
forecolor	颜色字符串	将选择的文本修改为指定的颜色
formatblock	要包围当前文本块的HTML标签；如<h1>	使用指定的HTML标签来格式化选择的文本块
indent	null	缩进文本
inserthorizontalrule	null	在插入字符处插入一个<hr>元素
insertimage	图像的URL	在插入字符处插入一个图像
insertorderedlist	null	在插入字符处插入一个 元素
insertunorderedlist	null	在插入字符处插入一个 元素
insertparagraph	null	在插入字符处插入一个<p>元素
italic	null	将选择的文本转换成斜体
justifycenter	null	将插入光标所在文本块居中对齐
justifyleft	null	将插入光标所在文本块左对齐
outdent	null	凸排文本（减少缩进）
paste	null	将剪贴板中的文本粘贴到选择的文本
removeformat	null	移除文本块的块级格式。这是撤销formatblock命令的操作
selectall	null	选择文档中的所有文本
underline	null	为选择的文本添加下划线
unlink	null	移除文本的链接。这是撤销createlink命令的操作