# JavaScript语法

## 基础语法

### 代码格式

|  |  |
| --- | --- |
| 内部 | **<script language="javascript">**  **<!--**  **......**  **//-->**  **</script>** |
| 外部 | <script language="javascript" src="Untitled-1.js" ></script> |
| 行内 | <a href="javascript: alert(new Date());">javascript</a>  <input type="button" value="test" onclick="alert(new Date());"> |

### 注释

### 标识符(变量名, 函数名, 数组名)命名规范

|  |  |
| --- | --- |
| 可以是 | 任意顺序的大小写字母, 数字, 下划线, 美元符号. |
| 不可以是 | 不能以数字开头.  不能有空格出现  不能是保留关键字 |

### 保留字(关键字)

A description...

## 简单数据类型

### 数值型

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| number | 不在引号内的数字 | |  | | --- | | JS中数值包含整数和浮点数, 所有数字都以双精度浮点型来表示.  备注: JS没有将整数和实数严格地分开, 他们可以自由转换. | | 八进制, 十进制, 十六进制  八进制: 以素质0开头, 后面紧跟八进制数.  十进制: 整数 和 浮点数, 都可以用科学计数法表示  十六进制: 数字0开头, 后面跟字母x不区分大小写. | | 科学计数法:  e不区分大小写: 3.14e-2  八进制/十六进制 没有指数形式表示法. | | 特殊值infinity(无穷大) :  infinity (超出JS能表示的数值范围, 表示无穷大)  -infinity(无穷小)  所有infinity和(-infinity)都相等, 并且infinity与其他数值进行运算的结果仍为infinity.  0/正数=Infinity 0/负数=-Infinity 0/0=NaN | | 特殊值NaN :  0/0 对Infinity求正选 Infinity与-Infinity相加  可通过isNaN(var)判断.  var iNumber = Number(value); // 返回一个数字或者NaN | | Number对象中特殊值属性   |  |  | | --- | --- | | Number.MAX\_VALUE | 可表示的最大值 | | Number.MIN\_VALUE | 可表示的最小值 | | Number.POSITIVE\_INFINITY | 无穷大Infinity | | Number.NEGATIVE\_INFINITY | 无穷小-Infinity | | Number.NaN | NaN | | |

### 字符串型

|  |  |  |
| --- | --- | --- |
| String | 引号内的一串字符 | 不区分单个字符和字符串, 均使用双引号或单引号引起来.  JavaScript为了提高效率, 在复制和传递字符串时使用的是引用, 但在对字符串进行比较时使用的却是值. |

### 正则表达式替换

|  |
| --- |
| 1. 替换全部字符 2. html = html.replace(**new** RegExp(/(&amp;)/g), "&"); |
|  |

**字符串常量**

|  |
| --- |
| 字符串常量必须使用 单引号或 双引号 括起来; 包括空字符串:""在内  字符串中的特殊字符, 需要用转义字符表示:  JavaScript中没有字符常量  字符串常量换行需要用加号连接. |

**转义字符**

|  |  |
| --- | --- |
| **\b** | **退格符** |
| **\f** | **换页符** |
|  |  |
| **\n** | **回车换行符** |
| **\r** | **换行符** |
|  |  |
| **\t** | **水平跳格, 制表符(tab)** |
| **\v** | **垂直跳格** |
|  |  |
| **\'** | **'** |
| **\"** | **"** |
| **\\** | **\** |
|  |  |
| **\ooo** | **八进制整数, 范围000~377** |
| **\xHH** | **十六进制整数, 范围00~FF** |
| **\uhhhh** | **十六进制编码的Unicode字符** |
|  |  |
| **备注** | **alert中用\n来换行**  **document.write中以 <br> 来换行** |

### 布尔值

|  |  |  |
| --- | --- | --- |
| boolean | 逻辑真或假 | 取值为true 和 false, 不能用1或0来表示. |

### null 和undefined

|  |  |  |
| --- | --- | --- |
| null | 空值 | 就是没有任何值, 什么也不表示.  专门用于指明一个已经创建但还没有初值的数据.  赋了一个空值 |
| undefined | 不存在的值  尚未赋值的变量. |  |

**编码规范**

|  |
| --- |
|  |

## 合成数据类型

### 函数

|  |  |  |
| --- | --- | --- |
| function | 函数 |  |

### 对象

|  |  |  |
| --- | --- | --- |
| object | 对象 |  |

### 数组

|  |  |  |
| --- | --- | --- |
| Array | 数组 | var a = new Array("youxuan", "luochuan", "wugang");  var l = a.length; |

**数据类型转换**

|  |
| --- |
| typeof操作符, 返回一个字符串, 表示操作数的类型. eg: typeof(age);  使用地方: 在JS脚本编程中, 当无法确认操作数的数据类型时, 可以使用typeof操作符, 根据返回值来确认其数据类型. |
| 将 字符串 --转换--> 数值   |  |  | | --- | --- | | parseFloat(numString, [radix]) | 将一个字符串转换成对应的小数 | | parseInt(numString, [radix]) |  |   将 其它类型 --转换--> 逻辑型   |  |  | | --- | --- | | undefined | false | | null | false | | 数值型 | 若为0或NaN, 则结果为false, 否则为true | | 字符串型 | 若长度为0(空字符串), 则结果为false, 否则为true. 备注: 空字符串长度为0 | | 其它对象 | true |   将 其它类型 --转换--> 数值型   |  |  | | --- | --- | | undefined | NaN | | null | 0 | | 逻辑型 | 值为true, 则结果为1, 值为false, 则结果为0 | | 字符串型 | 若内容为数值, 则结果为相应数字, 否则为NaN | | 其它对象 | NaN |   将 其它类型 --转换--> 字符串型   |  |  | | --- | --- | | undefined | "undefined" | | null | "NaN" | | 逻辑型 | 若为true, 则结果为"true", 值为false, 则为"false" | | 数值型 | NaN, 0或者与素质相对应的字符串 | | 其它类型 | 若存在, 则为toString()方法的值, 否则为"undefined" |  |  |  | | --- | --- | | 表达式 | 返回值 | | parseFloat("20") | 20 | | parseFloat("3.14") | 3.14 | | 3 + parseFloat("20") | 23 |   将 数值--转换--> 字符串  最简单的方法: 在数值中加入空字符串: ("" + 2500) |
| 整数和整数相加, 结果还是整数;  整数和浮点数相加, 结果是浮点数;  两个浮点数相加, 结果一般是浮点数, 但当结果为一个整数是, 就是整数形式. |

## 常量

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 表示在运行过程中保持不变的数据, 在JS脚本编程中可以直接输入这些值.  "youxuan"是字符型常量  134是数值型常量  true 或 false是逻辑型常量  整型常量   |  |  |  |  | | --- | --- | --- | --- | |  | 书写格式 | 例子 |  | | 八进制 | 零开头 | 0123 | 1\*82 + 2\*81 +3\*80 | | 十进制 | 第一位不能是零 | 334 | 3\*102 + 3\*101 + 4\*100 | | 十六进制 | 零X开头 | 0X8a |  |   实型常量   |  | | --- | | 12.32, 193.98, 5E7, 4e5 等 |   布尔常量 boolean   |  | | --- | | true 和 false |   null常量  表示一个什么都没有的占位符,  undefined常量  表示一个不存在的值或者尚未赋值的变量.  undefined表示一个变量尚未赋值; null表示一个变量被赋予一个空值.  特殊值:   |  |  | | --- | --- | | infinity | 无穷大 (infinity与其它数值运算的结果都为infinity) | | -infinity | 无穷小 (infinity = -infinity) | | NaN | 不是一个数值  使用举例: if(isNaN(x)){alert("x数NaN")} | | Number.MAX\_VALUE | 可表示的最大值 | | Number.MIN\_VALUE | 可表示到最小值 | | Number.POSTTIVE\_INFINITY | 无穷大Infinity | | Number.NEGATIVE\_INFINITY | 无穷小-Infinity | |

## 变量

|  |
| --- |
| 备注: 有经验的程序员习惯于同时定义变量和给变量赋值.  如果这个变量初始化时没有设定初始值, 那么他的值为null,  null值在if这类语句的条件判断中等同于false,  空字符串也被看作false  任何非0值等同于条件中的true.  eg:  var flag;  if(flag){  alert("flag是false");  } |
| 作用范围:  全局变量: 在函数外声明的变量．  局部变量: 在函数体内声明到变量，整个函数内都可见．不一定要先声明后使用. 也没有块级作用域限制, 函数中声明到变量无论是在哪里声明到, 整个函数中都有意义. |
| 基本类型 与 合成类型  基本数据类型在内存中具有固定大小, 而合成类型(对象,数组, 函数等类型)则可以具有任意长度.  基本类型存储的是实际数据, 合成类型存储的是引用地址. |
| 变量命名规范: 必须以字母开头, 后面可以是数字, 英文字母 或 下划线. 变量名中不能出现任何标点符号. |
| 定义一个变量, 系统就会为之分配一块内存, 程序可以用变量名来表示这块内存中的数据.  声明变量不必指定数据类型; 使用和赋值的时候由JavaScript引擎自动确定变量类型.  声明变量: var name="金万林";  可以使用一个var同时声明多个变量, 例如: var username, password;  var age; //此age默认为: undefined  也可以不声明变量, 而直接给一个变量赋值: x=124; |
| JS中没有块级作用域, 例如:  <html>  <head>  <script type="text/javascript" src="jquery.js"></script>  <script type="text/javascript">  <!--  function test(){  document.write("k=" +k + "<br />"); //必须先声明, 才能使用  var i=0;  if(true){  var j=0;  for(var k=0; k<3; k++){  document.write(k + "<br />");  }  document.write("k=" +k + "<br />"); //只要声明了, 在这个函数后面就都能访问. 不存在块级作用域, (变量k属于上面红色那个块, 在块外都可以访问)  }  document.write("j="+j + "<br />");  }  //-->  </script>  </head>  <body>  <a href="javascript:test">aaa</a> <br>  </body>  </html> |
| <script type="text/javascript">  <!--  var a=3.14;  var b=a; //这里是把变量a指向的地址赋值给了b, 也就是把数值3.14的地址赋值给了b.  a=4; //这里并没有修改3.14为4, 而是把变量a指向了数值4的地址.  document.write(a); //结果: 4  document.write(b) ; //结果: 3.14  //-->  </script>  **可以这样理解: 基本类型存放在栈中, 复合类型存放中堆中.** |

### 声明

|  |
| --- |
| var name;  var age; |
| var name, age; |

|  |
| --- |
| 1. 隐式声明: 不使用var关键字来声明, 而是直接为变量赋值. 2. 隐式声明的变量都是全局变量, 即使这个变量在方法中声明的, 也是全局变量. 3. 所以: 欲在方法或对象中声明局部变量, 必须使用var来声明 |
| 变量在使用前必须声明, 否则发生对象未定义的错误,  if(typeof str){ //当不确定变量是否声明时, 做一下判断, 保险一点  var str; //或者str=null  }  //接下来放心使用str变量 |
| 好的习惯是使用前都显示声明一下变量, 全局变量放在外面声明, 局部变量放在方法里面声明. |

### 类型

### 作用域

|  |
| --- |
| **全局变量**  作用范围是该变量定义后的所有语句, 包括其后定义的函数中的代码, 以及其后的<script></script>标签对中的代码  全局变量是window对象的属性 |
| **局部变量**  是定义在函数代码之内的变量, 只有在该函数中, 且位于该变量定义之后的程序代码, 才可以使用这个变量 |

## 表达式

## 运算符

**比较运算符**

|  |  |
| --- | --- |
| === | 绝对等于: 当操作对象相等并且类型相等, 返回true |
| !=== | 绝对不相等: 操作对象不相等, 并且不是同一类型 |
| == | 如果操作对象相等, 返回true;  如果操作对象不为同一类型, JavaScript将尝试转换他们为一个适当的类型 |
| != | 两个操作对象不相等 |

**逻辑操作符**

|  |  |
| --- | --- |
| 对 布尔变量进行运算. | |
| && | 逻辑与 |
| || | 逻辑或 |
| ! | 逻辑非 |

**位操作符**

|  |  |
| --- | --- |
| 对 二进制补码 作运算 (计算机中数字以二进制补码表示), 运算前将数字转换为二进制补码, 运算后将结果二进制补码转换为原来进制  在进行位运算前, 都先将操作数转换为32位的二进制整数, 然后再进行相应的运算, 最后的输出结果以十进制整数表示. | |
| 位逻辑运算符 | & 按位与  | 按位或  ~ 按位非  ^ 按位异或 |
| 位移动运算符 | << 左移, 补0  >> 算术右移, 补最高位  >>> 逻辑右移, 补0 |

**void运算符**

|  |
| --- |
| 用来指明一个表达式无返回结果. |
| var a, b, c, d;  var a, b, c, d;  a = void(b=5, c=7, d=56);  document.write('a=' + a);  结果: a=undefined |

**typeof(变量) 运算符**

|  |
| --- |
| 返回一个字符串, 指明其操作数到数据类型. |
| a=3; // number  b=null; // object  name="jin"; // string  alert("a的类型:" + typeof(a));  alert("a的类型:" + typeof(b));  alert("a的类型:" + typeof(name));  结果:  a的类型:number  b的类型:object  name的类型:string |

**对象运算符**

|  |
| --- |
| "." 点运算符: 连接对象及其属性 |
| "[]"运算符: 连接对象及其属性, 引用数组元素 |
| 例子: document["lastModified"] 等价于 document.lastModified |

**delete运算符**

|  |
| --- |
| 移除属性或元素  var myArray = ['1', '2', '3'];  delete myArray[2]; //移除数组的第三个元素 |

## 流程控制语句

### if else if

### switch

### while

### do-while

### for循环

### for in循环

|  |
| --- |
| //遍历数组  <script language="javascript">  <!--  var a = new Array(3);  a[0] = "吴刚";  a[1] = "罗川";  a[2] = "轩子";  document.write("<ul>");  for(var s **in** a){  document.write("<li>" + a[s] + "</li>");  }  document.write("</ul>");  //-->  </script> |

### break

**带标号的break**

|  |
| --- |
| st:while(true){  while(true){  break st;  }  } |

**continue**

|  |
| --- |
| for(var x=1; x<100; x++){  if(x%2 == 0){  continue; //执行continue, 将跳过循环体后面的语句, 去执行x++  }  alert(x);  } |

**with语句**

|  |
| --- |
| with(对象名称){  执行语句块里面直接使用方法名, 不需要此对象名.  } |

### var

### function

### return

## 函数

|  |
| --- |
| 如果函数中省略了return语句后的表达式, 或者函数结束时根本没有return语句, 函数就会返回一个undefineded值. |

**可变参数个数的函数**

|  |
| --- |
| **function** testparams(){ //定义参数可变的函数, 参数列表为空  **var** params = "";  **for**(**var** i=0; i<arguments.length; i++){ //在函数中从arguments数组中取传入的所有参数  params = params + "" + arguments[i];  alert(params);  }  }  //测试  testparams("abc", 123);  testparams(123, 456, "abc"); |

**动态函数**

|  |
| --- |
| **var** square = **new** Function("x", "y", "var sum; sum=x\*x+y\*y; return sum;"); //定义动态函数, 最后一个参数是函数体, 前面的所有参数是这个函数体到参数  alert(square(3, 2)); //测试动态函数 |

**函数的作用和定义**

|  |
| --- |
| function 函数名(参数列表){  程序代码......  return 表达式; //可以没有这条语句, 则会返回一个undefined的值  } |

**调用函数的两种方式**

|  |
| --- |
| 函数名(参数列表);  变量 = 函数名(参数列表); |

**全局函数与局部函数**

**其它函数**

|  |  |
| --- | --- |
| encodeURI(String) | URI的utf-8编码 |
| decodeURI(String) | URI解码 |
| isNaN方法 | 用于检测 parseInt 和 parseFloat 方法的返回值时候为NaN |
|  |  |
| eval(expr) | 将某个参数字符串作为一个JavaScript表达式执行.  在JavaScript中编写的每条表达式语句都是事先明确的, 不能根据运行时到情况进行变动, 但是使用eval函数可以实现这在运行时生成JS语句.  for(var i=0; i<n; i++){  eval("var a" + i + "=" + i); //动态生成JS表达式  } |
| isFinite(number) | 计算一个参数是否是一个有限数值,  是有限数: 值返回true  不是数字, 或 是正无穷数值, 或 是负无穷数值: 返回false |
| isNAN(value) | is not a number是否为非数字 |
| parseInt(numString, [radix]) | 将字符串转换成整数 |
| parseFloat | 将字符串转换成浮点数 |
| Number(objRef) | 将一个对象转换成一个数字 |
| String(objRef) | 将一个对象转换成一个字符串 |
| escape(str) | 将字符串进行Unicode编码 |
| unescape(str) | 将字符串进行Unicode解码 |
| prompt("提示信息") | str = prompt("提示信息"); //提示用户输入 |

### function构造函数 与 函数直接量

|  |
| --- |
| 1. 构造函数允许在运行时动态创建和变异JavaScript代码, 而函数直接量却是程序结构的一个静态部分, 就像函数语句一样.  2. 每次调用构造函数时都会解析函数体, 并且创建一个新的函数对象. 如果对构造函数的调用出现在一个循环中, 或出现在一个经常被调用的函数中, 这种方法的效率将非常低. 而函数直接量不论出现在循环中还是出现在嵌套函数中, 既不会在每次调用时都被重新编译, 也不会在每次遇到时都创建一个新的函数对象.  3. 使用构造函数创建的函数使用的不是静态作用域, 相反地, 该函数总是单作顶级函数来编译. |

## 对象

|  |
| --- |
| **用构造函数定义对象**  function Student(name, age, grade){  this.name = name;  this.age = age;  this.grade = grade;  } |
| **利用构造函数创建新对象**  student1 = new Student("youxuan", 23, 2); |
| **对象一旦建立, 就可以通过给新属性赋值来给对象加入新属性.**  student1.class = 7;//本语句只给student1加入新属性, 不会对其它对象造成影响. 若要给student对象的所有实例增加class属性, 需要在建立对象实例前在对象定义中增加如下语句.  function Student(name, age, grade){  this.name = name;  this.age = age;  this.grade = grade;  this.class = class;  } |
| **直接创建对象**  <script language="javascript">  <!--  var book = new Object();  book.title = "Book1";  book.chapter1 = new Object();  book.chapter1.title = "第一章";  book.chapter1.pages = 25;  book.chapter2 = {title:"第一章", pages:30};  alert("输出结果:" + "\n"  +book.title + "\n"  +book.chapter1.title + "\n"  +book.chapter1.pages + "\n"  +book.chapter2.title);  //-->  </script>  A description... |
| **对象的方法**  方法其实就是通过对象调用的JavaScript函数, 可以将函数赋值给任何变量, 甚至赋值给一个对象的任何属性.在方法的内部, 关键字this的值会变成调用该方法的对象.  方法用来对this对象进行操作.  函数通常是独立的, 并不需要使用this对象.  <script language="javascript">  <!--  //首先定义一些函数, 他们将被作为方法使用  function RectangleArea(){return this.width\*this.height;} //返回面积  function RectanglePerimeter(){return this.width\*2+this.height\*2} //返回周长  function RectangleSetsize(w, h){this.width\*=2; this.height\*=2;} //设置宽和高  //为Rectangle定义一个构造函数  function Rectangle(w, h){  //初始化对象的属性  this.width = w;  this.height = h;    //定义对象的方法  this.area = RectangleArea;  this.perimeter = RectanglePerimeter;  this.setsize = RectangleSetsize;  }  //调用Rectangle对象的方法  var r = new Rectangle(2,2);  var a = r.area();  r.setsize(3, 3);  var p = r.perimeter();  //-->  </script> |
| **原型对象 和 继承**  如果用构造函数把方法赋给药初始化的对象, 那么, 类的每一个对象都会有一个相同的方法属性的副本, 执行效率非常低.  在JS中, 可以使用一种更为有效的方式来声明方法, 常量以及能被类的所有对象共享的属性.  每一个对象都有一个原型对象, 而且一个对象可以继承其原型对象的所有属性. 要为一个对象的类指定原型对象, 需要将构造函数的propertye属性设置为适当的对象. 此后, 当用构造函数初始化一个新对象时, JS会自动将该对象作为新建对象的原型对象. |

**对象**

|  |  |
| --- | --- |
|  | 访问方式: |
| 动态对象 | 对象实例名.成员 |
| 静态对象 | 对象名.成员 |

### 属性

### 方法

### 系统内置函数

|  |
| --- |
| URI编码 解码 函数  var urlStr1 = encodeURI("http://www.swu.edu.cn");  var urlStr2 = decodeURI(urlStr1); |
| 数值处理函数 |
| onclick="**return** confirm('你确定要删除?');" |

**定时函数**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| setTimeout | | | 停顿指定的时间后  执行指定的函数  只执行一次 | |  | | --- | | <html>  <head>  <script type="text/javascript">  var t    function timedCount(){  document.getElementById('date').innerHTML=new Date; //给id='txt'的节点赋值  t=setTimeout("timedCount()",300); //休眠300毫秒再执行这行  }  </script>  </head>  <body onLoad='timedCount()'>  <div id='date'> </div>  <input type='button' value='stop' onclick='clearTimeout(t)'>  </body>  </html> | | <html>  <head>  <script language=*javascript*>  **var** f = 1;  **var** time = 2000;  **function** showTime(){  alert("1:" + time);  setTimeout("showTime()", time); //递归循环执行showTime(), 间隔时间可动态修改, 要停止执行, 只有修改间隔时间  alert("2:" + time);  f++;  time = f\*2000;  }  </script>  </head>  <body onload="showTime();">  </body>  </html> | | **循环修改title(有bug)**  <script language="javascript" >  <!--  var msgflag = true;  var tag = false;  loopRequest();  function loopRequest() {  if(msgflag){//循环修改title  if(tag){  document.getElementsByTagName("title").item(0).text = "00000";  tag=false;  }else{  document.getElementsByTagName("title").item(0).text = "11　　　　11";  tag=true;  }  //alert(document.getElementsByTagName("title").item(0).text);  }else{//修改title为默认值  document.getElementsByTagName("title").item(0).text = "00000";  }  setTimeout("loopRequest()", 500);  }  -->  </script> | |
| setInterval | 设置休息时间循环执行 | **var** d;  /\*\* 启动 \*/  **function** **start**(){  d=self.setInterval("clock()",50);//间隔时间(毫秒)不能在运行中修改  }  /\*\* 停止 \*/  **function** **stop**(){  window.clearInterval(d);  }  **function** **clock**(){  document.getElementById("clock").value=**new** Date();  } | | |

**创建自己的对象模版(相当于java中的类):**

|  |
| --- |
| function person(firstname,lastname,age,eyecolor){  this.firstname=firstname;  this.lastname=lastname;  this.age=age;  this.eyecolor=eyecolor;// 可以是变量 或 方法  } |

**其他内置函数**

|  |  |
| --- | --- |
| escape() | Unicode编码: 将一些特别字符编程ASCII编码, 这些特别的字符不包括字母, 数字以及\* @ - \_ + . /等字符 |
| unescape() | 与escape()相反, 用于将ASCII码变成字符. |

**对象与对象实例**

|  |
| --- |
| var person1 = new Person();  person1.name = "jinwanjiang"; //对象实例.成员变量 给成员变量赋值.  person1["name"] = "jinwanlin" //对象实例["成员变量"] 给成员变量赋值.  person1.say = sayFunc; //把一个函数赋值给person1的say成员方法.  person1.say(); //调用person1的say成员方法. |
| 可以使用delete语句来删除一个对象的属性.  delete person1.name; |

**正则表达式**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 元字符 ? 和 ()   |  |  | | --- | --- | | ? | 仅靠其前的元素是可选的(可有可无)  eg:  abc? 能匹配abc 和 ab  a(bc)? 能匹配 a 和 abc | | \d | 匹配一个任意的数字字符 | | {5} | 表示仅靠其前的元素相连续重复5次 | | () 子匹配 | ()除了能将多个元素组合成一个组合项外, 括起来的表达式部分还称为一个子匹配(也称子表达式)  子表达式匹配的内容按照从左到右的顺序存储在缓冲区, 该过程成为捕获.  子匹配捕获的内容可以在表达式中位于该括号之后的地方被反向引用, 第一个子匹配的内容用\1表示, 第二个用\2表示.  例如:  \d\d等同于(\d)\1  \d {5} 等同于 (\d)\1{4}  要匹配1221, 2553, 6776 等这样的对称数据, 可以用 (\d)(\d)\2\1 | |

**用正则表达式判断输入字符串是否是 yyyy-MM-dd**

|  |
| --- |
| //用正则表达式判断输入字符串是否是 yyyy-MM-dd 并且是否符合日期要求.  function verifyDate(d){  var datePattern = /^\d{4}-(0?[1-9]|1[0-2])-(0?[1-9]|[1-2]\d|3[0-1])$/;  return datePattern.test(d);  } |

# 浏览器内置对象

|  |  |
| --- | --- |
| **window对象概述**  (浏览器窗口顶级对象)  window对象可以省略, 直接使用它的属性和方法. e.g. alert() | navigator对象: 用于读取当前文档的浏览器的厂商和版本号.[只读] |
| screen对象: 用于获取浏览器运行的物理环境, 例如监视器的有效像素. [只读] |
| history对象:  借助此对象, 脚本可以模拟浏览器的"前进" "后退"功能.  在网页中不真实存在的对象, 每个窗口维护一个浏览器最近访问网页的列表. |
| location对象: 表示载入窗口的URL, 只是简单的URL |
| document对象: |

**访问window对象的属性和方法:**

|  |  |
| --- | --- |
| 第一种:  window.propertyName  window.methodName([parameters]) |  |
| 第二种方式:  省略window这个对象名, 直接访问它的属性和方法:  propertyName  methodName([parameters]) |  |
| 第三种:  使用self代替window这个词, 更可读性更好.  self. propertyName  self. methodName([parameters]) |  |

### window对象:专用事件:

|  |  |
| --- | --- |
| 事件名 | 事件描述 |
| onload | 装载完网页代码后发生 |
| onunload | 卸载完网页代码后发生 |
| onbeforeunload | 卸载网页代码之前发生. //取消关闭窗口提示框. |

|  |  |  |
| --- | --- | --- |
| open/close | 打开关闭浏览器窗口 | |
| alert | 显示一个警告对话框 | |
| confirm | 显示一个OK和Cancel对话框 | |
| prompt | 显示一个输入值文本域的提示对话框 | |
| blur和focus | 在窗口中移去或得到一个焦点 | |
| scrollTo | 卷动窗口到指定的坐标 | |
| setInterval | 在每个指定时间段 对一个表达式赋值或调用一个函数 | |
| setTimeout | | 在指定的一段时间后 对一个表达式赋值或调用一个函数   |  | | --- | | <html>  <head>  <script type="text/javascript">  var c=0;  var t;  function timedCount() {  document.getElementById('txt').value=c //把C的值赋给id='txt"的元素  c=c+1  t=setTimeout("timedCount()",1000);//循环执行  }    function stopCount() {  clearTimeout(t)  }  </script>  </head>  <body>  <form>  <input type="button" value="Start count!" onClick="timedCount()">  <input type="text" id="txt">  <input type="button" value="Stop count!" onClick="stopCount()">  </form>  </body>  </html> | |

**window对象创建子窗口**

|  |
| --- |
| var subwindow = window.open("define.html", "def", "height=200, width=300");  //第一个参数: 新窗口的URL,  //第二个参数: 窗口引用的名称;  //第三个参数: 物理外观  关闭刚才创建的子窗口: subwindow.close();  关闭主窗口: window.close(), self.close(), close() |

**window的一个属性和三个方法**

|  |  |
| --- | --- |
| window.status属性: **修改状态栏值.**   |  | | --- | | window.status="自定义显示的字符串";  return true; //必须返回true, 表名它是事件处理的一部分. | |
| window.alert()方法: **弹出警告对话框**  使用脚本不能修改对话框标题和按钮上的文字, 只能修改文本框提示消息内容.  A description... |
| window.confirm()方法: **确认对话框**(有两个按钮:"确定", "取消")  该方法有返回值, 返回true 或 false  A description... |
| window.prompt(提示信息, 输入默认值)方法: **输入提示对话框**  它显示预先设置的信息并提供输入内容. 有确定和取消两个按钮.  单击确定, 返回输入的字符串  单击取消, 返回null  A description... |

**History对象:**

|  |  |
| --- | --- |
| 去指定页面: | go(index) //index=正数 或 负数 |
| 后退: | back() |
| 前进: | forward() |

**Navigator对象**

Navigator对象, 是window对象的属性, 它封装了当前浏览器的相关信息.

|  |  |
| --- | --- |
| appName | 返回浏览器名字 |
| appVersion | 浏览器的版本 |
| appCodeName属性 | 浏览器厂商内部发行代码名 |
| language |  |
| platform |  |
| userAgent属性 | 返回浏览器和操作系统详细信息的字符串文本. |

**location对象:**

|  |
| --- |
| location对象表示载入窗口的URL, 这种URL是包含协议和主机信息的完整URL.  它和document对象不同, document对象是实际的内容.  location对象具有基于当前URL的属性  href属性: 就是<a fref=""></a>这类href属性.  hostname属性: 当前文档服务器和域名, 他有两个方法:  reload强制窗口重载当前文档;  replace从当前历史记录装载指定的URL; 也可用go方法来重定向到任何历史记录:history.go(-2) |

**document对象:**

|  |
| --- |
| 每一页都有一个document对象, 是文档页Anchor, applet, area, form, image, layer, link, plugin对象的父对象. |
| 访问document对象的属性和方法:  [window.]document.propertyName;  [window.]document.methodName([parameters]); |
| |  |  |  | | --- | --- | --- | | document.form**s**[]属性 | 返回表单数组 | | | document.image**s**[]属性 | 返回图像数组 | | | document.form**s**["formName "] 或 document.formName | 访问当前文档的表单 | | | document.form**s**.length; | 表单数组长度 | | | document.title | 网页标题 | | | document.write(写入内容可以是变量);  document.close(); //document.write(value)完后必须document.close() | 生成文本内容 | | | document.createElement("p"); //创建节点  document.createTextNode("这是生成的文本"); //创建文本 | | 说明: 创建元素或文本节点的动作不能影响文档的节点树,  必须调用一种插入或替换方法 将新的文本节点放置到其元素中并将元素放置到文档中.   |  | | --- | | var board = document.getElementById("board");  var e = document.createElement("input");  e.type = "button";  e.value = "这是测试加载的小例子";  var object = board.appendChild(e); | | var board = document.getElementById("board");  var e2 = document.createElement("select");  e2.options[0] = new Option("加载项1", "");  e2.options[1] = new Option("加载项2", "");  e2.size = "2";  var object = board.appendChild(e2); | | var board = document.getElementById("board");  var e3 = document.createElement("input");  e4.setAttribute("type", "text");  e4.setAttribute("name", "q");  e4.setAttribute("value", "使用setAttribute");  e4.setAttribute("onclick", "javascript:alert('This is a test!');");  var object = board.appendChild(e3); | | |

**DOM(Document Object Model, 文档对象模型)**

在DOM眼中, HTML是由很多个不同类型的节点组成的, 这些节点属于node对象.

node对象有一个nodeType属性, 可以返回节点类型.

|  |  |  |  |
| --- | --- | --- | --- |
| 接口 | nodeType常量 | nodeType值 | 备注 |
| Element | Node.ELEMENT\_NODE | 1 | 元素节点 |
| Attr | Node.ATTRIBUTE\_NODE | 2 | 节点属性 |
| Text | Node.TEXT\_NODE | 3 | 文本节点 |
| Comment | Node.COMMENT\_NODE | 8 | 注释的文本 |
| Document | Node.DOCUMENT\_NODE | 9 | Document根节点 |

利用ID属性, 找到那个节点, 然后把那个节点中的相关数据取出来.

A description... A description...

**DOM模型中的节点**

|  |
| --- |
| 元素节点  文本节点  属性节点 |

**使用DOM**

|  |
| --- |
| **获取节点:**  var nodes1 = document.getElementById(); //返回元素  var node2 = document.getElementsByTagName("a"); //返回document对象下所有<a></a>元素组成的数组 |
| **根据节点获取子节点**  var nodes3 = node2.childNodes; //返回node2的所有子标记组成的数组 |
| **根据节点 访问节点名称, 节点文本, 节点属性值**  node2.tagName; //返回元素的名称, 例如a 或者 li 或者 ul  node2.nodeValue; //返回元素的文本值  node2.getAttrubute("href"); //返回元素node2的href属性值. |
| **创建新节点:**  var oP = document.createElements("p"); //创建元素节点  var oText = document.createTextNode("这是创建的文本内容");  oP.appendChild(oText); //把文本绑定到元素节点  document.body.appendChild(oP); //把元素节点绑定到DOM |

**JS核心对象**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 使用核心对象之前, 需要new对象名(); 每个对象中都包含toString方法   |  | | --- | | Array | | Boolean | | Date | | Function | | Math | | Number | | RegExp 正则表达式 | | String | |

**String**

|  |
| --- |
| var name1 = "youxuan";  var name2 = new String("youxuan"); |
| 常用字符串方法:   |  |  | | --- | --- | | var result1 = string1.toUpperCase(); | 将字符串中的所有字符转换成大写, 这里不对原始字符串做任何操作. | | var result2 = string2.toLowerCase(); | 将字符串中的所有字符转换成小写, 这里不对原始字符串做任何操作. | | indexOf()  eg: var name = "youxuan";  var result = name.indexOf("you"); | indexOf()方法返回短字符串在长字符串中起始位置的索引值, 位置从0开始;  当短字符串不在长字符串中时, 返回-1. | |
| **从字符串中提取一个字符**: charAt(索引号从0开始)  这里不对原始字符串做任何操作.  eg: var name = "youxuan";  var result = name.substring(3); |
| **从字符串中提取子字符串:** 起始位置和终止位置, 从0开始计数.  **注意:** 其实位置包含着其中, 终止位置不包括在其中.  eg: var name = "youxuan";  var result = name.substring(3, 6); |
| **字符串长度:** var l = name.length; |

String对象属性和方法

|  |
| --- |
| 属性 |
| 1.constructor 指定创建一个对象的函数。 |
| constructor 属性是每个具有原型的对象的原型成员。这包括除了 arguments、Enumerator、Error、Global、Math、RegExp、Regular Expression 和 VBArray 对象以外的所有内部 JScript 对象。constructor 属性包含了对某种函数的引用，此种函数构造了特定对象的实例。  基于类的对象没有 constructor 属性。 |
| 2.length 返回字符串的长度。 |
| length 属性包含一个整数，该整数指示 String 对象中的字符数。String 对象中的最后一个字符的索引为 length - 1。 |
| 3.prototype 为对象的类返回原型的引用。 |
| 用prototype属性为对象的类提供一组基本功能。对象的新的实例“继承”了赋予该对象的原型的行为。  所有内部 JScript 对象都有一个只读的 prototype 属性。可以像该例中那样，为原型添加功能，但不可以向对象赋予另外一个原型。但是，可以向用户定义的对象赋予新的原型。  本语言参考中，每个内部对象的方法和属性列表都指示了哪些是对象原型的一部分，哪些不是。  注意：当以快速模式（JScript .NET 的默认模式）运行时，不能修改内置对象的 prototype 属性。若要从命令行编译使用 prototype 属性的程序，必须使用 /fast- 关闭快速选项。由于线程处理问题，在 ASP.NET 中关闭快速选项是不安全的。 |
|  |
| 方法 |
| 1.function anchor(anchorString : String ) : String |
| 返回对象中指定文本周围带有 HTML 定位点（具有 NAME 属性）的字符串。  参数：anchorString是要放在 HTML 定位点 NAME 属性中的文本。  备注：调用 anchor 方法在 String 对象外创建一个命名定位点。  未进行任何检查来查看此标记是否已应用于该字符串。 |
| 2.function big() : String |
| 返回 String 对象中文本周围带有 HTML ＜BIG＞ 标记的字符串。  备注：未进行任何检查来查看此标记是否已应用于该字符串。 |
| 3.function blink() : String |
| 返回 String 对象中文本周围带有 HTML ＜BLINK＞ 标记的字符串。  备注：未进行任何检查来查看此标记是否已应用于该字符串。在 Microsoft Internet Explorer 中不支持 ＜BLINK＞ 标记 |
| 4.function bold() : String |
| 返回 String 对象中文本周围带有 HTML ＜B＞ 标记的字符串。  备注:未进行任何检查来查看此标记是否已应用于该字符串。 |
| 5.function charAt(index : Number) : String |
| 返回 String 对象的指定索引处的字符  参数：index 必选。所需字符的从零开始的索引。有效值为 0 到字符串长度减 1 的数字。  备注：charAt 方法返回一个字符值，该字符值等于指定索引位置的字符。一个字符串中的第一个字符位于索引位置 0，第二个字符位于索引位置 1，依此类推。超出有效范围的 index 返回空字符串。 |
| 6.function charCodeAt(index : Number) : String |
| 返回一个整数，该整数表示 String 对象中指定位置处的字符的 Unicode 编码。  参数：index 必选。所需字符的从零开始的索引。有效值为 0 到字符串长度减 1 的数字。  备注：一个字符串中的第一个字符位于索引位置 0，第二个字符位于索引位置 1，依此类推。如果指定 index 没有字符，将返回 NaN。 |
| 7.function concat([string1 : String [， ... [， stringN : String]]]]) : String |
| 返回一个字符串值，该值包含当前字符串与提供的任何字符串的连接。  参数：string1， ...， stringN 可选。要连接到当前字符串末尾的 String 对象或文本。  备注：concat 方法的结果等同于：result = curstring + string1 + ... + stringN。curstring 是指对象中存储的用于提供 concat 方法的字符串。源字符串中或结果字符串中的值的变化都不会影响另一个字符串中的值。如果有不是字符串的参数，则它们在连接到 curstring 之前将首先被转换为字符串。 |
| 8.function fixed() : String |
| 返回 String 对象中文本周围带有 HTML ＜TT＞ 标记的字符串。  备注：未进行任何检查来查看此标记是否已应用于该字符串。 |
| 9.function fontcolor(colorVal : String) : String |
| 返回 String 对象中文本周围带有 HTML ＜FONT＞ 标记（具有 COLOR 属性）的字符串。  参数：colorVal 必选。包含颜色值的字符串值。可以是颜色的十六进制值，也可以是颜色的预定义名。  备注：有效的预定义颜色名取决于 JScript 主机（浏览器、服务器等）。它们也可能随主机版本的不同而不同。详细信息请查阅主机文档。未进行任何检查来查看此标记是否已应用于该字符串。 |
| 10.function fontsize(intSize : Number) : String |
| 返回 String 对象中文本周围带有 HTML ＜FONT＞ 标记（具有 SIZE 属性）的字符串。  参数：intSize 必选。用来指定文本大小的整数值。  备注：有效的整数值取决于 Microsoft JScript 主机。详细信息请参见主机文档。未进行任何检查来查看此标记是否已应用于该字符串。 |
| 11.function fromCharCode([code1 : Number [， ... [， codeN : Number]]]]) : String |
| 从一些 Unicode 字符值中返回一个字符串。  参数：code1， ...， codeN 可选。要转换为字符串的 Unicode 字符值序列。如果没有给出参数，结果为空字符串。  备注：fromCharCode 方法是从全局 String 对象中调用的。 |
| 12.function indexOf(subString : String [， startIndex : Number]) : Number |
| 返回 String 对象内第一次出现子字符串的字符位置。  参数：subString 必选。在 String 对象中搜索的子字符串。  startIndex 可选。该整数值指定在 String 对象内开始搜索的索引。若省略此参数，则搜索从字符串的起始处开始。  备注：indexOf 方法返回一个整数值，该值指示 String 对象内子字符串的起始位置。如果未找到子字符串，则返回 -1。如果 startindex 为负，则将 startindex 视为零。如果它比最大字符位置索引还大，则将它视为可能的最大索引。搜索将从左向右执行。否则，此方法与 lastIndexOf 相同。 |
| 13.function italics() : String |
| 返回字符串对象中文本周围带有 HTML ＜I＞ 标记的字符串。  备注：未进行任何检查来查看此标记是否已应用于该字符串。 |
| 14.function lastIndexOf(substring : String [， startindex : Number ]) : Number |
| 返回 String 对象中某个子字符串的最后一个匹配项的索引。  参数：substring 必选。在 String 对象内将被搜索的子字符串。  startindex 可选。该整数值指定在 String 对象内开始搜索的索引。若省略该参数，则搜索将从字符串的结尾开始。  备注：lastIndexOf 方法返回一个整数值，指示 String 对象内子字符串的开始位置。如果未找到子字符串，则返回 -1。如果 startindex 为负，则将 startindex 视为零。如果它比最大字符位置索引还大，则将它视为可能的最大索引。从右向左执行搜索。否则，该方法和 indexOf 相同。 |
| 15.function link(linkstring : String) : String |
| 返回 String 对象中文本周围带有 HTML 定位点和 HREF 属性的字符串。  备注：调用 link 方法来创建 String 对象外部的超级链接。未进行任何检查来查看此标记是否已应用于该字符串。 |
| 16.function localeCompare(stringExp : String) : Number |
| 返回一个值，指示两个字符串在当前区域设置中是否相等。  参数：stringExp 必选。要与当前字符串对象进行比较的字符串。  备注：localeCompare 对当前字符串对象和 stringExp 进行区分区域设置的字符串比较，并返回 -1、0 或 +1，这取决于系统中默认的区域设置的排序顺序。如果当前字符串对象排在 stringExp 之前，则 localeCompare 返回 -1；如果当前字符串排在 stringExp 之后，则返回 +1。如果返回值为零，则说明这两个字符串是相等的。 |
| 17.function match(rgExp : RegExp) : Array |
| 使用正则表达式模式对字符串执行搜索，并返回一个包含该搜索结果的数组。  参数：rgExp 必选。包含正则表达式模式和适用标志的 Regular Expression 对象的实例。也可以是包含正则表达式模式和标志的变量名或字符串。  备注：如果 match 方法没有找到匹配，将返回 null。如果找到匹配，则 match 方法返回一个数组，并将更新全局 RegExp 对象的属性以反映匹配结果。  match 方法返回的数组有三个属性：input、index 和 lastIndex。Input 属性包含整个被搜索的字符串。Index 属性包含了在整个被搜索字符串中匹配的子字符串的位置。LastIndex 属性包含了前一次匹配中最后一个字符的下一个位置。  如果没有设置全局标志 (g)，数组的零元素包含整个匹配，而第 1 到第 n 个元素包含了匹配中曾出现过的任一个子匹配。此行为与没有设置全局标志的 exec 方法的行为相同。如果设置了全局标志，则元素 0 到 n 中包含所有出现的匹配。 |
| 18.function replace(rgExp : RegExp， replaceText : String) : String |
| 返回一个字符串的副本，该字符串的文本已被使用正则表达式或搜索字符串替换。  参数：rgExp 必选。Regular Expression 对象的实例包含正则表达式模式和适用标志。也可以是 String 对象或文本。如果 rgExp 不是 Regular Expression 对象的实例，它将被转换为字符串，并对结果进行精确的搜索；字符串将不会被试图转化为正则表达式。  replaceText 必选。一个 String 对象或字符串文本，包含用于替换当前字符串对象中 rgExp 的每个成功匹配的文本。在 Jscript 5.5 或更高版本中，replaceText 参数也可是返回替换文本的函数。  备注：在完成指定的替换之后，replace 方法的结果是当前字符串对象的副本。下面任意的匹配变量都可用于识别最新的匹配及匹配字符串。在需要动态确定替换字符串的文本替换中可以使用匹配变量。   |  |  | | --- | --- | | **字符** | **含义** | | **$$** | **$**（JScript 5.5 或更高版本） | | **$&** | 指定当前字符串对象中与整个模式相匹配的部分。（JScript 5.5 或更高版本） | | **$`** | 指定当前字符串对象中位于 **$&** 所描述的匹配前面的部分。（JScript 5.5 或更高版本） | | **$＇** | 指定当前字符串对象中位于 **$&** 所描述的匹配后面的部分。（JScript 5.5 或更高版本） | | **$*n*** | 第 *n* 个捕获到的子匹配，这里 *n* 为从 1 到 9 的十进制一位数。（JScript 5.5 或更高版本） | | **$*nn*** | 第 *nn* 个捕获到的子匹配，这里 *nn* 为从 01 到 99 的十进制二位数。（JScript 5.5 或更高版本） |   如果 replaceText 是一个函数，对于每个匹配的子字符串，调用该函数时带有下面的 m + 3 个参数，这里 m 是在 rgExp 中用于捕获的左括弧的个数。第一个参数是匹配的子字符串。接下来的 m 个参数是搜索中捕获到的全部结果。参数 m + 2 是当前字符串对象中发生匹配位置的偏移量，而参数 m + 3 是当前字符串对象。结果为将每一匹配的子字符串替换为函数调用的相应返回值后的字符串值。Replace 方法更新全局 RegExp 对象的属性。 |
| 19.function search(rgExp : RegExp) : Number |
| 返回正则表达式搜索中第一个子字符串匹配项的位置。  参数：rgExp 必选。Regular Expression 对象的实例包含正则表达式模式和适用标志。  备注：search 方法表示是否找到了匹配项。如果找到一个匹配项，则 search 方法将返回一个整数值，该值指示发生匹配处距字符串开头的偏移量。如果没有找到匹配项，则返回 -1。 |
| 20.function slice(start : Number [， end : Number]) : String |
| 返回字符串的片段。  参数：start 必选。指向字符串指定部分的开头的索引。 end 可选。指向字符串指定部分的结尾的索引。  备注：slice 方法返回一个包含字符串指定部分的 String 对象。  slice 方法一直复制到 end 所指示的元素，但是不包括该元素。如果 start 为负，则将其视为 length + start，此处 length 为字符串的长度。如果 end 为负，则将其视为 length + end，此处 length 为字符串的长度。如果省略 end，则将一直提取到字符串的结尾。如果 end 出现在 start 之前，则不会将任何字符复制到新字符串中。 |
| 21.function small() : String |
| 返回 String 对象中文本周围带有 HTML ＜SMALL＞ 标记的字符串。  备注：未进行任何检查来查看此标记是否已应用于该字符串。 |
| 22.function split([ separator : { String | RegExp } [， limit : Number]]) : Array |
| 返回一个字符串拆分为若干子字符串时所产生的字符串数组。  参数：separator 可选。字符串或正则表达式对象的实例，它标识用于分隔字符串的一个或多个字符。如果忽略该参数，将返回包含整个字符串的单元素数组。  limit 可选。一个用于限制数组中返回的元素数的值。  备注：split 方法的结果是在字符串中出现 separator 的每个位置分隔字符串后产生的字符串数组。separator 将不作为任何数组元素的一部分返回。 |

**Math**

Math对象是静态对象. 属性通常是常量.

|  |  |  |
| --- | --- | --- |
| Math.E | 2.718 281 828 459 045 | 欧拉常数 |
| Math.LN2 | 0.6 931 471 805 599 453 | 2的自然对数 |
| MathLN10 | 2.302 585 092 994 046 | 10的自然对数 |
| Math.LOG2E | 1.4 426 950 408 889 633 | 以2为底E的对数 |
| Math.PI | 3.141 592 653 589 793 | 圆周率 |
| Math.SQRT2 | 1.4 142 135 6233 730 951 | 2的平方根 |
| Math.SQRT1\_2 | 0.7 071 067 811 865 476 | 0.5的平方根 |
|  |  |  |
| abs(n) |  |  |
| Math.max(value1, value2) | 取最大值 |  |
| Math.pow(value1, 10) | value1的10次方. |  |
| Math.round(value1) | 把一个值四舍五入成一个整数 |  |
| Math.random() | 生成随机数, 返回0-1之间的浮点数.  Math.floor(Math.random() \* n)+m ;//返回m到n之间的随机数, 包含m和n. |  |
| Math.floor() | 返回任意浮点数的整数部分. |  |

**Number对象**

Number.MAX\_VALUE

最大值(1.79E+308)

Number.MIN\_VALUE

最小值(2.22E-308)

Number.PASITIVE\_VALUE

比最大值还大的数

Number.NEGATIVE\_VALUE

比最小值还小的数

Number.toExponential(小数点后面的位数)

强制数值用指数符号显示, 整数常数指明返回的数值在小数点后面有多少位, 即使是0, 小数点有的所有数字都必须返回.

Number.toFixed(小数点后面的位数)

将数值格式化为小数点后面指定的数值, 使用该方法显示具有单位和百分制单位(如美元和美分)的货币计算结果.

如果格式化的数值右边的位数比指定参数大, 方法将后面的值四舍五入(但只对未进行四舍五入的下一个数字进行处理)

第一个参数表示:小数点后面显示的位数.

Number.toPrecision(精度)

定义数值中共显示多少位数(包括整数和小数部分,

包括小数点), 即定义数值的精度.

Number.toString()

如果初始值不是十进制数, 默认将值转换为十进制数, 或指定其它基数

|  |  |  |
| --- | --- | --- |
| toString(2) | 二进制 | |
| toString(8) | | 八进制 |
| toString() | | 十进制 |
| toString(16) | | 十六进制 |

Number.

## Boolean对象

|  |
| --- |
| var va1 = new Boolean(布尔值) |

|  |  |
| --- | --- |
| Boolean.constructor | 创建该对象的构造对象 |
| Boolean.prototype | 对象原型 |
| Boolean.toString() | 返回该布尔值的文本形式: 即"true" , "false" |
| Boolean.valueOf() | 返回对象的基本布尔值. |

**Date**对象

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Date对象是静态对象.  由于JS脚本中内部使用毫秒数跟踪日期和时间, 因此, 在最后将出现准确的日期, 即使新的日期在下一个月. JS脚本能自动计算闰年及一个月有多少天.  **构造函数**   |  | | --- | | var today = new Date() | | var today = new Date(3143243243); //毫秒数作为参数. | | var someDate = new Date("Month dd, yyyy hh:mm:ss") | | var someDate = new Date("Month dd, yyyy") | | var someDate = new Date("yy,mm,dd,hh,mm,ss") | | var someDate = new Date("yy,mm,dd") | | var someDate = new Date("GMT milliseconds from 1/1/1970") |   **Date对象的get方法**   |  |  |  | | --- | --- | --- | | dateObj.**getTime**() | 0~... | **从GMT 1970年1月1日00:00:00开始计算的毫秒数**. | | dateObj.getYear() | 70~... | 指定的年份减去1900, 2000年后为4位数表示的年份. | | dateObj.getFullYear() | 1970~... | 4位数年份, 适用于版本4以上的浏览器. | | dateObj.getMonth() | 0~11 | 一年中的月份, 一月为0 | | dateObj.getDate() | 1~31 | 一个月中的日期 | | dateObj.getDay() | 0~6 | 星期(星期日为0) | | dateObj.getHours() | 0~23 | 一天中的小时数, 采用24小时制. | | dateObj.getMinutes() | 0~59 | 指定小时内的分钟数 | | dateObj.getSeconds() | 0~59 | 指定分钟内的秒数. | | dateObj.setTime(val) | 0~... | 从GMT 1970年1月1日00:00:00开始计算的毫秒数. |   **Date对象的set方法**   |  |  |  | | --- | --- | --- | | dateObj.setTime () | 0~... | 从GMT 1970年1月1日00:00:00开始计算的毫秒数. | | dateObj.setYear() | 70~... | 指定的年份减去1900, 2000年后为4位数表示的年份. | | dateObj.setFullYear() | 1970~... | **没有setFullYear这个方法** | | dateObj.setMonth() | 0~11 | 一年中的月份, 一月为0 | | dateObj.setDate() | 1~31 | 一个月中的日期 | | dateObj.setDay() | 0~6 | 星期(星期日为0) | | dateObj.setHours() | 0~23 | 一天中的小时数, 采用24小时制. | | dateObj.setMinutes() | 0~59 | 指定小时内的分钟数 | | dateObj.setSeconds() | 0~59 | 指定分钟内的秒数. | |
| 日期计算: 要求在Date对象的毫秒值上进行.  eg: 获取系统当前日期, 求7天后的日期.  function nextWeek(){  var todayInMS = today.getTime();  var nextWeekInMS = todayInMS + (60\*60\*24\*7\*1000);  return new Date(nextWeekInMS);  }  也可以这样写:  function nextWeek(){  var nextWeek= today.setDate(today.getDate() + 7);  return nextWeek;  } |

**JS格式化日期(输出yyyy-MM-dd)**

|  |
| --- |
| **function** formatDate(date){  **var** year = date.getFullYear();  **var** month = date.getMonth()+1;  **var** date = date.getDate();  **if**(month<10){  month = "0" + month;  }  **if**(date<10){  date = "0" + date;  }  **var** dateStr = year+"-"+month+"-"+date;  **return** dateStr;  } |

**某年某月第一天 & 某年某月最后一天**

|  |
| --- |
| **var** begin = **new** Date(2012, 5, 1);  **var** end = **new** Date(2012, parseInt(5)+1, 0); |

**Array**

|  |
| --- |
| 索引值从0开始,  可以使用字符串作为索引代替数字.  同一个JS数组的不同元素可以包含不同的数据类型. |
| var students = new Array(52);  students[0] = "youxuan";  students[1] = "wugang"; |
| Array对象构造方法:   |  | | --- | | var arr = new Array();  arr[0] = 1;  arr[1] = "king"; | | var arr = Array(2); //括号里必须是一个整数  arr[0] = 1;  arr[1] = "king"; | | var arr = Array(3.5, "abc", 3); | |
| 元素类型是任意的,  可以是空, 即逗号之间没有内容, // var arr = ["jin", , , "wanlin"]  可以是JavaScript表达式  可以是数组. //数组的数组var arr = [[1, 2, 3], ["jin", "wanlin"], 3, 4] 访问方式 arr[0][1] |
| var arr = ["jin", 123, "wanlin", 23]; |
| |  |  | | --- | --- | | 方法 | 描述 | | indexOf(str,position) | 从position(可选)位置开始,搜索字符串中的第一个str所出现的位置.例:  "hello,jack".indexOf("hello")返回0  "abcabc".indexOf("a",1)返回3 | | join() | 将一个数组到所有元素转换为字符串, 默认以逗号分隔, 也可指定分隔符: join(",")  a.join();  a.join(","); | | reverse() | 对数组元素进行重排, 颠倒数组元素的顺序.  var a = [1, 2, 3];  var b = a.reverse(); | | sort() | 用来对数组元素进行排序. 如果不带参数, 输出结果将按照字母顺序排列.  如果要按其他顺序排列, 需要将一个比较函数作为参数传递给sort方法. | | alice() | 返回指定数组的一个子数组, 起止点:包含第一个参数数组位置, 不包含第二个参数数组位置  如果只有一个参数, 则表示返回:包含第一个参数至后面所有元素到子数组.  如果两个参数中有一个是负数, 则表示相对于数组最有一个元素而言到元素.(-1代表最后一个元素, -3代表倒数第三个元素.) | | lastIndexOf(str,position) | 从position(可选)位置开始,搜索字符串中的最后一个str所出现的位置.例:  "abcabc".lastIndexOf("b")返回4 | | charAt(pos) | 返回字符串中位置为pos的字符,例:  "abc".charAt(1)返回"b" | | charCodeAt(pos) | 返回字符串中位置为pos的字符的ASCII码,例:  "abc".charCodeAt(0)返回97 | | slice(start,end) | 返回字符串起始位置为start,结束位置为end(不包括end)的子字符串. | | substr(start,length) | 将字符串以separator作为分隔符切割成多个子字符串,并返回一个数组.  linmit(可选)表示数组的最大长度.例:  "a1,b1,c1".split(",")将返回["a1","b1","c1"]  "a,b,c".split(",",2)返回["a","b"]  "a,b,c".split("")返回["a",",","b",",","c"]  "ab,c".split()返回["ab,c"] | | split(separator,linmit) | 返回字符串中起始位置为start,长度为length的子字符串,例:  "abcdefg".substr(1,3)返回"bcd" | | splice() | 可以删除数组中的元素, 也可以向数组中插入新元素. | | substring(start,end) | 返回字符串中起始位置为start,结束位置为end(包括end)的子字符串. | | replace(value,reValue) | 将字符串中第一个出现的value子字符串替换为reValue,并返回新的字符串.  var str1="aaaa";  var str2=str1.replace("a","b");  alert(str2);  alert(str1);  str2=str1.replace(/a/g,"b");//替换多个需要使用正则表达式  alert(str2);  alert(str1); | | match(regExp) | 从字符串中搜索出匹配regExp正则表达式的所有子字符串,作为数组返回.还可以判断一个字符串是否匹配regExp表示的正则表达式。例:  var strInput=prompt("输入一个数字:",0);  while(!strInput.match(/\d+/)){  strInput=prompt("输入一个数字:",0);  } | | search(regExp) | 从字符串中搜索出匹配regExp正则表达式的第一个子字符串,返回其索引位置.例:  var str="aabcabcabc";  alert(str.search(/abc/g)); | | toLowerCase() | 将字符串中的英文字母全部转换为小写,返回一个新的字符串.原字符串不受影响. | | toUpperCase() | 将字符串中的英文字母全部转换为大写,返回一个新的字符串.原字符串不受影响. | |

# DHTML

A description...

**form对象**

|  |
| --- |
| 访问表单的属性(name, target, action, method, enctype)  document.forms[0].action = "http://www.swu.edu.cn"; 或者 |
| document.forms[0].elements[]// 返回表单的子节点数组  document.forms[0].elements[0].type; //返回第一个节点的type值(text 或 password 等) |

IE的事件监听

|  |
| --- |
| function |

**事件**

**事件监听**

|  |
| --- |
| IE浏览器中事件对象是window对象的一个属性event  oP.onclick = function(){  var oEvent = window.event;  }  标准DOM规定event对象必须作为唯一的参数传给事件处理函数, 显然IE是特立独行.  oP.onclick = function(oEvent){  //...  } |
| 为了兼容IE浏览器和其他遵循标准DOM标准的浏览器, 采用以下做法.  oP.onclick = function(oEvent){  if(window.event){  oEvent = window.event; //如果window.event返回true, 则表明是IE浏览器, 那么采用IE浏览器的时间监听器, 如果返回false, 则使用传入的实参.  }  //...  } |
| **事件类型**  if(oEvent.type == "click"){...} |
| **文档载入后运行指定代码**  window.onload = function(){...} |

**事件绑定**

|  |
| --- |
| **function** handle(oEvent){  **var** d2Node = document.getElementById("d2");  **if**(window.event){  oEvent = window.event;  }  **if**(oEvent.type == "click"){ //判断事件的类型, 注意, 这里去掉了事件前面的on  d1Node.innerHTML += "你点击我了!";  }  **if**(oEvent.type == "mouseover"){ //判断事件的类型, 注意, 这里去掉了事件前面的on  d1Node.innerHTML += "你移动到我上方了!";  }  }  window.onload = **function**(){  d1Node = document.getElementById("d1");  d1Node.onclick = handle; //当节点发生onclick事件时, 绑定handle函数  d1Node.onmouseover = handle; //当节点发生onmouseover事件时, 绑定handle函数  } |
| **为一个节点添加多个事件和移除事件**  **var** oP;  **function** fnClick(){  alert("我被点击了!");  oP.detachEvent("onclick", fnClick); //【IE中】删除oP节点上绑定的监听函数  //oP.removeEventListener("click", fnClick, **false**); //【FireFox】中删除oP节点上绑定的监听函数  }  window.onload = **function**(){  oP = document.getElementById("myP");  oP.attachEvent("onclick", fnClick); //【IE中】添加oP节点上的监听函数, 可以绑定多个监听函数  //oP.addEventListener("click", fnClick, **false**); //【FireFox】中添加oP节点上的监听函数, 可以绑定多个监听函数  } |

**表格的添加和删除**

|  |
| --- |
| //添加一行表格  window.onload = **function** addTrTd(){  **var** oTr = document.getElementById("member").insertRow(2); //插入一行, 2代表新添加的行, 行号为2, 即作为第三行.  **var** aText = **new** Array();  aText[0] = document.createTextNode("aaaaaaaaa"); //第一个单元格的内容  aText[1] = document.createTextNode("aaaaaaaaa");  aText[2] = document.createTextNode("aaaaaaaaa");  aText[3] = document.createTextNode("aaaaaaaaa");  **for**(**var** i=0; i<aText.length; i++){  **var** oTd = oTr.insertCell(i);//添加一个单元格  oTd.appendChild(aText[i]); //在刚才添加的单元格中添加文本内容  }  }  //删除表格中的一行  **function** delTr(){  **var** oTable = document.getElementById("member");  oTable.deleteRow(2);//删除第二行, 后面的行自动补齐  }    //删除表格中的一个单元格  **function** delTd(){  **var** oTable = document.getElementById("member");  oTable.rows[2].deleteCell(1); //删除第二行的第一个单元格,后面的单元格自动补齐.  } |

**用JS控制textarea可输入的字符个数**

|  |
| --- |
| **function** LessThan(oTextArea){  **return** oTextArea.value.length < oTextArea.getAttribute("maxlength");  }  <textarea name=*"comments"* id=*"comments"* cols=*"40"* rows=*"4"* maxlength=*"5"* onkeypress="return LessThan(this);">  </textarea>  **备注:** <textarea>的maxlength属性是用户自定义的. |

**单行文本输入框可输入的字符个数**

|  |
| --- |
| <input type=*"text"* name=*"textfield"* id=*"textfield"* maxlength=*"5"* /> |

**输入框获得焦点, 相关特效**

|  |
| --- |
| <input type=*"text"* name=*"username"* value=*"wanlinjin"* onmouseover="this.focus()" onfocus=*"this.select()"*> |
| <input type=*"text"* value=*"搜索关键字..."* onfocus=*"if(this.value=='搜索关键字...'){this.value=''}"* onblur=*"if(this.value==''){this.value='搜索关键字...'}"* /> |

**错误处理**

|  |
| --- |
| **公用通用错误处理函数**  window.onerror = **function**(){ //只要JS一出错, 就自动执行这个函数  alert("出错了!");  } |
| **使用try..catch..捕获异常**  **try**{  //....  }**catch**(exception){  **var** sError = "";  **for**(**var** i **in** exception){  sError = sError + i + ":" + exception[i] + "\n";  }  alert(sError);  } |
| **浏览器调试器**  Firefox的debug 和 Venkman插件  IE的Microsoft Script Debugger |

## 事件驱动:

|  |  |
| --- | --- |
| onLoad | 载入窗口 |
| onUnload | 关闭网页 |
| location.reload() | 重新载入当前页面 |
| window.location=”www.swu.edu.cn” | 打开指定页面 |
|  |  |
| onFocus | 获得焦点 |
| onBlur | 失去焦点 |
|  |  |
| event.x | 鼠标的x坐标 |
| event.y | 鼠标的y坐标 |
|  |  |
| onClick | 鼠标单击 |
| onDbclick | 鼠标双击 |
| onMouseMove | 鼠标移动 |
| onChange | 值被改变 |
| onmousedown | 鼠标键按下 |
| onmouseout | 鼠标从某个元素上方移出 |
| onmouseover | 鼠标停留在某个元素上方 |
| onmouseup | 鼠标键被释放 |
|  |  |
| onkeyup | 释放按键 |
| onkeydown | 按下鼠标键 |
| onkeypress | 按下并释放鼠标键 |
|  |  |
| onreset | 表单被复位(重置) |
| onsubmit | 表单将被提交 |
| onselect | 用户选择文本, 通常会高亮显示. |
|  |  |
| onReturn |  |

# 事件处理

如何编写事件处理程序:

|  |
| --- |
| 1. 直接在JavaScript代码中设置元素对象的事件属性, 让事件属性值等于处理该事件的函数名或程序代码. |
| 2.  <script language="JavaScript" for="document" event="oncontextmenu">  window.event.returnValue=false;  </script> |
| 3. |

|  |  |
| --- | --- |
| 动作 | 促发事件名称 |
| 右键 | oncontextmenu |
| 禁用右键菜单 | window.event.returnValue = false; |

**常用JS对HTML的操作**

|  |  |
| --- | --- |
| 创建新节点 | var op = document.createElement("p"); |
| 创建节点文本 | var otext = document.createTextNode("节点文本内容"); |
| 文本添加到节点 | op.appendChild(otext); |
| 节点添加到节点 | document.body.appendChild(op); |
| 获取innerHTML内容 | var text = myDiv.innerHTML |
| 设置innerHTML的值 | myDiv.innerHTML = "<img src='01.jpg' title='hehe' >" |
| 给id="id\_1"的节点, 插入HTML代码 | document.getElementById("id\_1").innerHTML=xmlHttp.responseText; |
| 给name="time"的节点, 设置value值 | document.myForm.time.value = "2010-04-28" |
| 获取id="time"的节点的value值 | document.getElementById("username").value |

# 获取浏览器相关数据(地址/宽/高/滚动条位置)

|  |
| --- |
| 浏览器地址  **var** url = document.location+""; |
|  |

# 代码摘抄

## 改变浏览器 位置和大小的函数

|  |
| --- |
| <html>  <head>  <script type="text/javascript" language="javascript">  function ok()  {  window.resizeTo(400, 300);  x = 200, y = 150;  window.moveTo(x, y);  for (i = 0; i < 5; i ++) {  window.moveTo(x,y);  window.moveTo(x,y + 10);  window.moveTo(x + 10,y + 10);  // window.moveTo(x + 10,y);  }  }  </script>  </head>  <body>  <input type = "button" value = "······" onclick="ok();">  介绍一个用window的小技巧，做个能够实现类似QQ的窗口抖动的功能，其实也不算是技巧，就是玩来着。  天天都用QQ，而且经常和lp用那个抖动的功能，感觉挺有意思，也不知道是谁学谁的，后来msn甚至飞信都有了抖动的这个功能。  不多废话，  window.moveTo这个函数，就是单纯的将网页窗口的左上角移动到一个绝对位置，  window.moveBy是能够相对移动的。  window.resizeTo，能够重新设置窗口大小。    <center>  </center>  </body>  </html> |

## JS实现页面跳转

|  |
| --- |
| **按钮式**  <input id="sub" type="submit" value="登　陆" onclick="**window.location.href='welcome.html**'"/> |
| <script type="text/javascript">  url="http://202.202.96.44/login/logout1.jsp";  //window.location.href=url;  window.location.replace(url);  </script> |
| **<body onload="document.form1.submit();">**  **<form name='form1' action='http://202.202.96.53/index.jsp' method='post'>**  **</form>**  **</body>** |
| **开新窗口：**  　　<a href="javascript:" onClick="window.open('http://www.h591.com/blog/guestbook.asp','','height=500,width=611,scrollbars=yes,status=yes')">祥子网屋</a> |
| **JS跳转页面参考代码**  第一种：  <script language="javascript" type="text/javascript">  window.location.href="login.jsp?backurl="+window.location.href;  </script>  第二种：  <script language="javascript">  alert("返回");  window.history.back(-1);  </script>  第三种：  <script language="javascript">  window.navigate("top.jsp");  </script>  第四种：  <script language="JavaScript">  self.location=’top.htm’;  </script>  第五种：  <script language="javascript">  alert("非法访问！");  top.location=’xx.jsp’;  </script> |
| ＝＝＝＝＝javascript中弹出选择框跳转到其他页面＝＝＝＝＝  <script language="javascript">  <!--  function logout()...{  if (confirm("你确定要注销身份吗？是－选择确定，否-选择取消"))...{  window.location.href="logout.asp?act=logout"  }  }  -->  </script> |

**倒计时**

|  |
| --- |
| <HTML>  <HEAD>  <meta http-equiv="page-enter" content=“revealtrans(duration=3,transition=12)">  <meta http-equiv="page-exist" content=“revealtrans(duration=3,transition=23)">  <TITLE>离校--倒计时</TITLE>    <script language=JavaScript>      var theDay = new Date("july 5, 2010")  var DayALL    function DownCount(){  var today = new Date()  var seconds = Math.floor((theDay.getTime() - today.getTime())/1000)  var minutes = Math.floor(seconds/60)  var hours = Math.floor(minutes/60)  var days = Math.floor(hours/24)  CDay= days  CHour= hours % 24  CMinute= minutes % 60  CSecond= seconds % 60  DayALL = CDay + " 天 " + CHour + " 小时 " + CMinute + " 分钟 " + CSecond + " 秒 "    document.getElementById("count").innerHTML = DayALL;  var counter = setTimeout("DownCount()", 100)  }    </script>  </HEAD>    <BODY bgcolor="#fef4d2" onLoad=DownCount()>  <center><h1>  <BASEFONT size="n">  <font color="ffaafa">离校<br>(2010-07-05 00:00:00)--倒计时</font>  <hr>  <font size=+15>    <div id="count" />    </font>  </h1>  </BODY>    </HTML> |

**文本框校验**

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <title>无标题文档</title>  </head>  <body>  <script language="javascript">  function checkUserName(username){  if(username.length<3 || username.length>18){  document.getElementById("usernameErr").innerHTML = "<font color='red'>请输入正确用户名,用户名长度为3-18位!</font>";  return false;  }  document.getElementById("usernameErr").innerHTML = "";  return true;  }  </script>  <form>  <input type="text" id="username" name="username" onblur="checkUserName(this.value.toLowerCase())"/>  <span id="usernameErr"/>  </form>  </body>  </html> |

**json级联表单**

|  |
| --- |
| <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>MyHtml.html</title>  <script type="text/javascript">  var data = [{name:'湖北',id:99,cities:[{name:'武汉',id:3},{name:'宜昌',id:9},{name:'荆州',id:6}]},  {name:'湖南',id:77,cities:[{name:'长沙',id:4},{name:'岳阳',id:7},{name:'郴州',id:10}]},  {name:'四川',id:77,cities:[{name:'成都',id:5},{name:'宜宾',id:11},{name:'锦阳',id:8},{name:'自贡',id:12}]}];  //填充所有的省份  function fillProvince(){  for(var i=0; i  <data.length  ; i++){  var provincedObj = data[i];  var optProvince = new Option(provincedObj.name, provincedObj.id);  document.getElementById("provincId").add(optProvince);  }  }  //填充省份下的城市  function fillCity(provinceIndex){  document.getElementById("cityId").options.length=0; //简单的清空子元素办法  //clearOptions(document.getElementById("cityId")); //用JS代码实现真正的清空子元素  var provinceObj = data[provinceIndex];  for(var i=0; i  <provinceObj.cities.length  ; i++){ //i从1开始的原因是:脚标为0的数据为省份  var cityObj = provinceObj.cities[i];  var optCity = new Option(cityObj.name, cityObj.id); //创建city节点, 第一个参数是value, 第二个参数是key  document.getElementById("cityId").add(optCity);  }  }  //清除options的内容  function clearOptions(selElement){  var options = selElement.options;  var len = options.length;  for(var i=0; i  <len  ; i++){  //selElement.remove(i); //父元素的remove方法  options.remove(0); //子元素集合的remove方法  }  }  </script>  </head>  <body>  <select id="provincId" name="provincId" onchange="fillCity(this.selectedIndex)"></select>  <select id="cityId" name="cityId"></select>  <script type="text/javascript">  fillProvince();//显示省份  fillCity(document.getElementById("provincId").selectedIndex); //页面载入时, 级联显示省份下的城市  </script>  </body>  </html> |

**JS添加到收藏夹 设为首页**

|  |
| --- |
| <a onclick="window.external.addFavorite('http://www.199xp.com/','自定义网站名字')" href="#">加入收藏</a> |
| <a href="http://www.yijing365.com" onclick="this.style.behavior='url(#default#homepage)';this.setHomePage('http://www.yijing365.com');return(false);" style="">设为首页</a> |

**状态栏跑马灯效果**

|  |
| --- |
| <script Language="JavaScript">  var msg="这是一个跑马灯效果的JavaScript文档";  var interval = 100;  var spacelen = 120;  var space10=" ";  var seq=0;  function Scroll() {  len = msg.length;  window.status = msg.substring(0, seq+1);  seq++;  if ( seq >= len ) {  seq = spacelen;  window.setTimeout("Scroll2();", interval );  }else  window.setTimeout("Scroll();", interval );  }    function Scroll2() {  var out="";  for (i=1; i<=spacelen/space10.length; i++) out +=  space10;  out = out + msg;  len=out.length;  window.status=out.substring(seq, len);  seq++;  if(seq >= len){  seq = 0;  }  window.setTimeout("Scroll2();", interval );  }  Scroll();  </script> |

**将日期字符串转换为日期对象**

|  |
| --- |
| var arrayD1 = d1.split("-"); //将日期字符串(yyyy-MM--dd)转换为数组, 这个数组包含三个元素:年月日  var date1 = new Date(arrayD1[0],arrayD1[1],arrayD1[2]); //创建日期对象 |

**获取本机IP和MAC地址**

|  |
| --- |
| 方法一:  <script language="JavaScript">  **function** GetLocalIPAddr() {  **var** oSetting = **null**;  **var** ip = **null**;  **try** {  oSetting = **new** ActiveXObject("rcbdyctl.Setting");  ip = oSetting.GetIPAddress;  alert(ip);  **if** (ip.length == 0) {  **return** "没有连接到Internet";  }  oSetting = **null**;  } **catch** (e) {  **return** ip;  }  **return** ip;  }  alert(GetLocalIPAddr())  </script> |
| 方法二:  <!doctype html public "-//w3c//dtd html 4.01 transitional//en">  <html>  <head>  <title>wmi scripting html</title>  <meta http-equiv=*content-type* content=*"text/html; charset=utf-8"*>  <meta content=*"mshtml 6.00.2800.1106"* name=*generator*>    <script language=*jscript* event=*"oncompleted(hresult,perrorobject, pasynccontext)"* for=*foo*>  document.forms[0].txtmacaddr.value = unescape(macaddr);  document.forms[0].txtipaddr.value = unescape(ipaddr);  document.forms[0].txtdnsname.value = unescape(sdnsname);  </script>    <script language=*jscript* event=*onobjectready(objobject,objasynccontext)* for=*foo*>  **if** (objobject.ipenabled != **null** && objobject.ipenabled != "undefined" && objobject.ipenabled == **true**) {  **if** (objobject.macaddress != **null** && objobject.macaddress != "undefined")  macaddr = objobject.macaddress;  **if** (objobject.ipenabled && objobject.ipaddress(0) != **null** && objobject.ipaddress(0) != "undefined")  ipaddr = objobject.ipaddress(0);  **if** (objobject.dnshostname != **null** && objobject.dnshostname != "undefined")  sdnsname = objobject.dnshostname;  }  </script>  </head>    <body>  <!--  -->  <object id=*locator* classid=*clsid:76a64158-cb41-11d1-8b02-00600806d9b6* ></object>  <object id=*foo* classid=*clsid:75718c9a-f029-11d1-a1ac-00c04fb6c223*></object>    <script language=*jscript*>  **var** service = locator.connectserver();  **var** macaddr;  **var** ipaddr;  **var** domainaddr;  **var** sdnsname;  service.security\_.impersonationlevel = 3;  service.instancesofasync(foo, 'win32\_networkadapterconfiguration');  </script>    <form id=*formfoo* name=*formbar* action=*nicpost.asp* method=*post*>  <table>  <tr><td>MAC地址</td><td><input value=*""* name=*txtmacaddr*></td></tr>  <tr><td>IP地址</td><td><input value=*""* name=*txtipaddr*></td></tr>  <tr><td>????</td><td><input value=*""* name=*txtdnsname*></td></tr>  </table>  </form>  </body>  </html> |

**使用JavaScript运行bat文件获取IP, 并读取运行后的bat文件.**

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <title>获得ip</title>  <script>  //使用JavaScript获得客户端的的ip配置信息通过使用bat文件运行后获得一个配置信息文件  //在通过读取这个配置信息拿到客户端的信息，可见js的功能确实很强大，可是由于现在的浏览器安全限制是不能够让用户这样使用的但是可以参考这个方法。以后会用到。  function getIP(bat,txt){  var fso=new ActiveXObject("Scripting.FileSystemObject")  var f=fso.createTextFile(bat,1);  f.writeLine("ipconfig>"+txt);  f.Close();  var wsh=new ActiveXObject("wscript.shell");  wsh.run(bat)  setTimeout(display(),1000) }  function display(){  var f=fso.OpenTextFile(txt);  alert(f.ReadAll());  f.Close();  fso.DeleteFile(bat);  fso.DeleteFile(txt);  }  getIP("qswhIP.bat","clientipconfig.txt")  </script>  </head>  <body>  </body>  </html> |

## JS修改CSS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 当需要批量修改某些元素的样式时，可以直接用JS修改CSS的定义   |  | | --- | | <style type="text/css">  div{  height:300px;  width:300px;  border:1px #003399 solid;  background-color:#006633;  }  </style> | | <script language="JavaScript" type="text/javascript">  if(document.all){//兼容IE  document.styleSheets[0].rules[0].style.height="100px";  }else{//兼容firefox  document.styleSheets[0].cssRules[0].style.height="100px";  }  //alert(document.getElementsByTagName('style')[0].innerHTML);  </script> |   写成函数，由于样式名只能用数字来索引，所以用了DOM遍历（此函数来源于：http://tb.blog.csdn.net/TrackBack.aspx?PostId=1885195）   |  | | --- | | <style>  .exampleA{}  .exampleB{}  </style> | | <script>  **function** changecss(theClass,element,value) {  **var** cssRules;  **if** (document.all) {  cssRules = 'rules';  }  **else** **if** (document.getElementById) {  cssRules = 'cssRules';  }  **for** (**var** S = 0; S < document.styleSheets.length; S++){  **for** (**var** R = 0; R < document.styleSheets[S][cssRules].length; R++) {  **if** (document.styleSheets[S][cssRules][R].selectorText == theClass) {  document.styleSheets[S][cssRules][R].style[element] = value;  }  }  }  }  </script> | | <span A</span>  <span B</span>  <span A</span>  <input type="button" value="Change A Red" onclick="changecss('.exampleA','color','red')"/>  <input type="button" value="Change A Black" onclick="changecss('.exampleA','color','black')"/> | | **第二种写法:**  document.getElementById('province').display='block'; | |

## JS验证复选框是否被选中

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>无标题文档</title>  <script language=*"javascript"*>  **function** check(){  **var** v1=document.getElementsByName("checkbox");  **var** check=**false**;  //alert(v1.length);  **for**(**var** i=0;i<v1.length;i++){  **if**(v1[i].type=='checkbox' && v1[i].checked){  check=**true**;  }  }  **if**(!check){  alert("请至少选择一个您最理想的团购");  **return** **false**;  }    //返回所选值  **for**(i=0;i<v1.length;i++){  **if**(v1[i].checked){  alert(v1[i].value);  //return true;  }  }  **return** **false**;  }  **function** getValue(v1){  }  </script>  </head>  <body>  <form id=*"form1"* name=*"form1"* method=*"post"* action=*""*>  <label>  <input type=*"checkbox"* name=*"checkbox"* id=*"checkbox"* value=*"1"*/>1  </label>  <label>  <input type=*"checkbox"* name=*"checkbox"* id=*"checkbox6"* value=*"2"* />2  </label>  <label>  <input type=*"checkbox"* name=*"checkbox"* id=*"checkbox2"* value=*"3"* />3  </label>  <label>  <input type=*"checkbox"* name=*"checkbox"* id=*"checkbox3"* value=*"4"* />4  </label>  <label>  <input type=*"checkbox"* name=*"checkbox"* id=*"checkbox4"* value=*"5"* />5  </label>  <label>  <input type=*"checkbox"* name=*"checkbox"* id=*"checkbox5"* value=*"6"* />6  </label>  </form>  <input type=*"button"* onclick="check();" value=*"验证是否选择, 并取出所选值"* />  </body>  </html> |

## 使某个元素获得焦点

|  |
| --- |
| document.getElementById("text").focus(); |

## JS结合正则表达式 替换html文本

|  |
| --- |
| //过滤HTML文本  **function** htmlFilter(htmlText){  **var** htmlText;  htmlText = htmlText.replace(/&/g, "&amp;");  htmlText = htmlText.replace(/</g, "&lt;");  htmlText = htmlText.replace(/>/g, "&gt;");  htmlText = htmlText.replace(/ /g, "&nbsp;");  htmlText = htmlText.replace(/'/g, "&#39;");  htmlText = htmlText.replace(/\"/g, "&quot;");  htmlText = htmlText.replace(/\n/gi, "<\br>");  **return** htmlText;  } |

点击链接之执行JS, 不跳转

|  |
| --- |
| <a href=*"javascript:void(0);"* class=*"MagicZoomPlus"*><img src=*"../picture/*<%=pic%>*"* width=*"50px"* height=*"50px"*/></a> |

## JS三种弹出对话框

|  |
| --- |
| <script language="javascript">  **//1：只是提醒，不能对脚本产生任何改变；**  function ale()  {//这个基本没有什么说的，就是弹出一个提醒的对话框  alert("我敢保证，你现在用的是演示一");  }  **2：一般用于确认，返回 true 或者 false ，所以可以轻松用于 if...else...判断**  function firm()  {//利用对话框返回的值 （true 或者 false）  if(confirm("你确信要转去 天轰穿的博客？"))  {//如果是true ，那么就把页面转向thcjp.cnblogs.com  location.href="http://thcjp.cnblogs.com";  }  else  {//否则说明下了，赫赫  alert("你按了取消，那就是返回false");  }  }  **例子**  **<a href="delete.do?id=23" onclick="return confirm('你确定要删除?');" >删除</a>**  **3： 一个带输入的对话框，可以返回用户填入的字符串，常见于某些留言本或者论坛输入内容那里的 插入UBB格式图片**  function prom()  {  var name=prompt("请输入您的名字","");//将输入的内容赋给变量 name ，  //这里需要注意的是，prompt有两个参数，前面是提示的话，后面是当对话框出来后，在对话框里的默认值  if(name)//如果返回的有内容  {  alert("欢迎您："+ name)  }  }  </script> |

## 点击按钮, 复制制定字符串到剪贴板(IE可以, FF不兼容)

|  |
| --- |
| ﻿<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <title>ipv6代理服务器</title>  </head>  <body>  <SCRIPT>  function funCopy(code)  {  alert(code +" 已复制到剪贴板");  var arr=new Array();  arr.push(code);  var str=arr.join("");  window.clipboardData.setData('text',str);  alert(code + " 已经复制到剪贴板");  }  </SCRIPT>  <center><h2>使用说明: 单击按钮, 按钮上的字符串就复制到剪贴板了.</h2></center>  </p></p>  <h1>  ipv6代理服务器:<input type="button" value="proxytest.mucip.net" onclick="funCopy('proxytest.mucip.net')" /><p>  端口号:<input type="button" value="3128" onclick="funCopy('3128')" /><p>  </h1>  </body>  </html> |

## JS让某输入框获得焦点

|  |
| --- |
| admin.loginUserName.focus(); //admin:form的名字, loginUserName是input的名字 |

## 用JS修改浏览器的标题栏

|  |
| --- |
| 修改浏览器状态栏的脚本很多，修改浏览器标题栏的程序还不多见，偶然发现，特拿出来共享！  我是通过JS来实现的！演示程序：http://www.85time.com首页  JS文件：myjs.js  <!-- Begin  var clocktext;  var pagetitle = document.title;  function scroll() {  today = new Date();  sec = today.getSeconds();  hr = today.getHours();  min = today.getMinutes();  if (hr <= 9) hr = "0" + hr;  if (min <= 9) min = "0" + min;  if (sec <= 9) sec = "0" + sec;  var clocktext = " 现在时间是: " + hr + ":" + min + ":" + sec;  clocktimer = setTimeout("scroll()", 1000);  document.title = pagetitle + clocktext;  }  if (document.all) scroll();  // End -->  在需要调用的网页中，用如下代码：  <script src="myjs.js"></script> |

## 统一处理提示信息

|  |
| --- |
| /\*\*  \* 淡入淡出提示div  \* 使用方法: JS调用notice(message)方法, 参数message为提示文字.  \*/  **function** notice(message){  $("body").prepend('<div id="noticeDiv" style="color:red; font-weight:bolder; text-align:center; margin:auto; vertical-align:bottom; padding-top:5px; display:block; height:20px; letter-spacing:4; width:300px; background-color:#FFF68F; position:absolute; top:2px; z-index:999;"></div>');    **var** noticeDiv = $("#noticeDiv");  **var** left = (document.body.clientWidth+noticeDiv.width() - noticeDiv.width())/2 + document.body.scrollLeft - 180 + 'px';    noticeDiv.css("left", left);  noticeDiv.html(message);  noticeDiv.fadeIn(500);  noticeDiv.fadeOut(1000000);  }  $(**function**(){  **if** (document.cookie == "") {  setCookie("isBack", "0");  }  **var** isBack = getCookie("isBack");//读取cookie中名为isBack的值  **if**(isBack == 0){  **var** message = $("#messageDiv").html();  **if**(message != **null**){  notice(message);  }    //处理地址栏名字为message的提示信息  //**var** url = document.location.search;  //**var** arr = url.split('&');  //**for**(**var** i=0;i<arr.length;i++){  // **var** ar = arr[i].split('=');  // **if**(ar[0] == 'message' || ar[0] == '?message'){  // notice(decodeURI(ar[1]));  // }  //}    }**else**{  setCookie("isBack", "0"); //每次进入页面, 显示完提示信息后. 都设置标志isBack为0, 表示不是返回的页面  }  })  /\*\*返回上一个页面调用的方法\*/  **function** getBack(){  setCookie("isBack", "1"); //设置cookie名为isBack的值为1, 表示执行了返回动作)  history.go(-1);  }  /\*\*设置cookie\*/  **function** setCookie(name,value){  **var** exp = **new** Date();  exp.setTime(exp.getTime() + 30\*24\*60\*60\*1000);  document.cookie = name + "="+ escape (value) + ";expires=" + exp.toGMTString();  }    /\*\*获取cookie\*/  **function** getCookie(name){  **var** arr = document.cookie.match(**new** RegExp("(^| )"+name+"=([^;]\*)(;|$)"));  **if**(arr != **null**){  **return** unescape(arr[2]);  }**else**{  **return** "";  }  }  /\*\*删除cookie\*/  **function** delCookie(name){  **var** exp = **new** Date();  exp.setTime(exp.getTime() - 1);  **var** cval=getCookie(name);  **if**(cval!=**null**) document.cookie= name + "="+cval+";expires="+exp.toGMTString();  } |

## 读 - 写 - 删cookie

|  |
| --- |
| /\*\*读cookie\*/  **function** getCookie(name){  **var** arr = document.cookie.match(**new** RegExp("(^| )"+name+"=([^;]\*)(;|$)"));  **if**(arr != **null**){  **return** unescape(arr[2]);  }**else**{  **return** "";  }  }  /\*\*写cookie\*/  **function** setCookie(name,value){  **var** exp = **new** Date();  exp.setTime(exp.getTime() + 30\*24\*60\*60\*1000);  document.cookie = name + "="+ escape (value) + ";expires=" + exp.toGMTString();  }  /\*\*删cookie\*/  **function** delCookie(name){  **var** exp = **new** Date();  exp.setTime(exp.getTime() - 1);  **var** cval=getCookie(name);  **if**(cval!=**null**) document.cookie= name + "="+cval+";expires="+exp.toGMTString();  } |

## 处理框架页面, 退出系统后登陆页面不是最外层问题

|  |
| --- |
| <script type=*"text/javascript"*>  **if**(window.parent!=window){  window.parent.location=window.location;  }  </script> |

## ifram内部文档高度改变, 自动调整ifram框架高度, 而不是出现滚动条

|  |
| --- |
| <script>  **function** **SetCwinHeight**(obj)  {  **var** cwin=obj;  **if** (document.getElementById)  {  **if** (cwin && !window.opera)  {  **if** (cwin.contentDocument && cwin.contentDocument.body.offsetHeight)  cwin.height = cwin.contentDocument.body.offsetHeight + 20; //FF NS  **else** **if**(cwin.Document && cwin.Document.body.scrollHeight)  cwin.height = cwin.Document.body.scrollHeight + 10;//IE  }  **else**  {  **if**(cwin.contentWindow.document && cwin.contentWindow.document.body.scrollHeight)  cwin.height = cwin.contentWindow.document.body.scrollHeight;//Opera  }  }  }  </script> |
| <iframe onload=*"****SetCwinHeight****(this);"* style=*"height:auto !important;height:700px;min-height:700px;"* name=*"body"* width=*"100%"* height=*"100%"* scrolling=*"no"* frameborder=*"0"* marginheight=*"0"* marginwidth=*"0"* src="<c:url value=*'/navigation/main.do'* />"></iframe> |

## 子页面修改父页面中的信息

|  |
| --- |
| **function** **setDepartment**(){  **if**(**undefined** == name)  opener.document.forms[0].departmentName.value = "";  **else**  opener.document.forms[0].departmentName.value = name;  opener.document.forms[0].departmentId.value = id;  window.close();  } |

## 鼠标经过事件, 修改样式

|  |
| --- |
| $(function(){  //鼠标经过评论, 显示该评论外边框; 鼠标离开评论, 隐藏该评论外边框  $(".common").mouseover(showOutline);  $(".common").mouseout(hideOutline);  }) |
| //显示外边框  function showOutline(){  $(this).css("outline", "solid 1px #CDBA96").css("background-color", "#DBDBDB");  }  //隐藏外边框  function hideOutline(){  $(this).css("outline", "none").css("background-color", "#fff");  } |

## onclick提交form

方法1:

document.register.submit();**return** **false**;// register为form的name值

方法2:

document.forms[0].submit(); **return** **false**;

## 页面弹出提示的top和left

|  |
| --- |
| **var** top = ($(window).height()/ 2 - $("#" + obj).height() / 2)+ "px" ;  **var left =** ($(window).width() / 2 - $("#" + obj).width() / 2) + "px" ; |

## 一直显示滚动条的底部

document.getElementById("logbox").scrollTop = document.getElementById("logbox").scrollHeight;

## cookieUtil.js

|  |
| --- |
| /\*\*设置cookie\*/  **function** setCookie(name,value){  **var** exp = **new** Date();  exp.setTime(exp.getTime() + 30\*24\*60\*60\*1000);  document.cookie = name + "="+ escape(value) + ";expires=" + exp.toGMTString()+";path=/";  }    /\*\*获取cookie\*/  **function** getCookie(name){  **var** arr = document.cookie.match(**new** RegExp("(^| )"+name+"=([^;]\*)(;|$)"));  **if**(arr != **null**){  **return** unescape(arr[2]);  }**else**{  **return** "";  }  }  /\*\*删除cookie\*/  **function** delCookie(name){  **var** exp = **new** Date();  exp.setTime(exp.getTime() - 1);  **var** cval=getCookie(name);  **if**(cval!=**null**){  document.cookie = name + "="+escape(value) + ";expires=" + exp.toGMTString()+";path=/";  }  } |

## notice.js

|  |
| --- |
| **var** id = **null**;  **var** noticeDiv = **null**;  **var** hideTime = **null**;  **var** removeTime = **null**;  /\*\*  \* 淡入淡出提示div  \* 使用方法: JS调用notice(message)方法, 参数message为提示文字.  \*  \* 参数cssStyle: success/failure/warning  \*/  **function** notice(message, cssStyle){    **if**(noticeDiv != **null**){  clearTimeout(hideTime);//取消之前设置的隐藏定时器  clearTimeout(removeTime);//取消之前设置的删除定时器  remove\_notice();//直接删除之前的提示框  }    id = "notice" + Math.floor(Math.random() \* 10000)+1 ;//返回m到n之间的随机数, 包含m和n.    $("body").prepend('<div id="'+id+'" class="noticeDiv"><div class="'+cssStyle+' msg-left"></div><div id="msg" class="msg"></div><div class="msg-right"></div></div>');    noticeDiv = $("#"+id);    **var** top = getObjTop();  **var** left = getObjLeft();  noticeDiv.css("top", top);  noticeDiv.css("left", left);    $("#msg").html(message);  noticeDiv.fadeIn(500);    hideTime = window.setTimeout("hide\_notice()",1800);  }  //隐藏提示div  **function** hide\_notice(){  noticeDiv.fadeOut(500);  removeTime = window.setTimeout("remove\_notice()",550);  }  //移除提示div  **function** remove\_notice(){  noticeDiv.remove();  }  $(window).resize(**function**() {  **var** top = getObjTop();  **var** left = getObjLeft();  noticeDiv.css("top", top);  noticeDiv.css("left", left);  });  $(**function**(){  **if** (document.cookie == "") {  setCookie("isBack", "0");  }  **var** isBack = getCookie("isBack");//读取cookie中名为isBack的值  **if**(isBack == 0){  //成功  **if**(**typeof** success!="undefined" && success != **null**){  notice(success, 'success');  }    //失败  **if**(**typeof** failure!="undefined" && failure != **null**){  notice(failure, 'failure');  }    //警告  **if**(**typeof** warning!="undefined" && warning != **null**){  notice(warning, 'warning');  }    }**else**{  setCookie("isBack", "0"); //每次进入页面, 显示完提示信息后. 都设置标志isBack为0, 表示不是返回的页面  }  });  /\*\*返回上一个页面调用的方法\*/  **function** getBack(){  setCookie("isBack", "1"); //设置cookie名为isBack的值为1, 表示执行了返回动作)  history.go(-1);  }  /\*\* 获取居中div的top值\*/  **function** getObjTop() {  **var** winH = $(window).height();//屏幕的高度  **var** objH = noticeDiv.height();//浮动对象的高度  **var** h = ((Number(winH) - Number(objH))) / 2;  **return** h +"px";  }  /\*\* 获取居中div的left值\*/  **function** getObjLeft() {  **var** winW = $(window).width();//屏幕的宽度  **var** objW = noticeDiv.width();//浮动对象的宽度  **var** w = ((Number(winW)) - Number(objW)) / 2 - 100;  **return** w +"px";  }  **function** developing(){  notice("开发中....", "failure");  **return** **false**;  } |

## notice.css

|  |
| --- |
| */\*\* 所用背景图片是QQ使用的提示背景图片 \*/*  *.noticeDiv*{  height:*53px*;  padding-left:*45px*;  position: *fixed*;  z-index:*999*;  }  *.noticeDiv* *.msg-left*{  width:*45px*;  height:*53px*;  float:*left*;  }  *.noticeDiv* *#msg*{  background:*url(noticeDiv\_bg.png)* *repeat-x* *0* *-161px*;  padding:*15px*;  float:*left*;  font-size: *18px*;  font-weight: *700*;  }  *.noticeDiv* *.msg-right*{  background:*url(noticeDiv\_bg.png)* *repeat-x* *0* *0*;  height:*53px*;  width: *6px*;  float:*left*;  }  *.noticeDiv* *.success*{  background:*url(noticeDiv\_bg.png)* *no-repeat* *-6px* *0px*;  }  *.noticeDiv* *.failure*{  background:*url(noticeDiv\_bg.png)* *no-repeat* *-6px* *-108px*;  }  *.noticeDiv* *.warning*{  background:*url(noticeDiv\_bg.png)* *no-repeat* *-6px* *-54px*;  } |

页面内跳转的两种方式   
1 使用超链接 锚点   
2 使用window.location.hash = '#id';

## 判断浏览器类型

|  |
| --- |
| /\*\* 判断浏览器类型 \*/  **function** browserType(){  **if** ((navigator.userAgent.indexOf('MSIE') >= 0) && (navigator.userAgent.indexOf('Opera') < 0)) {  **return** 'IE';  } **else** **if** (navigator.userAgent.indexOf('Firefox') >= 0) {  **return** 'Firefox';  } **else** **if** (navigator.userAgent.indexOf('Opera') >= 0) {  **return** 'Opera';  } **else** {  **return** 'Other';  }  } |

## 禁止浏览器后退

|  |
| --- |
| 方法一：  <script type="text/javascript">  jQuery(document).ready(function ($) {  if (window.history && window.history.pushState) {  　　　　$(window).on('popstate', function () {  　　 window.history.forward(1);  　　　　});  　　　　}  　　　　});  　　</script>    　　方法二：（未测试）  　　$(function() {  　　if (window.history && window.history.pushState) {  　　$(window).on('popstate', function () {  　　window.history.pushState('forward', null, '#');  　　window.history.forward(1);  　　});  　　}  　　window.history.pushState('forward', null, '#'); //在IE中必须得有这两行  　　window.history.forward(1);  　　}) |