01 JavaScript基础

JavaScript介绍

- **JavaScript简介**

JavaScript 是世界上最流行的脚本语言。 JavaScript 是属于 web 的语言，它适用于 PC、笔记本电脑、平板电脑和移动电话。 JavaScript 被设计为向 HTML 页面增加交互性。 许多 HTML 开发者都不是程序员，但是 JavaScript 却拥有非常简单的语法。几乎每个人都有能力将小的 JavaScript 片段添加到网页中。

完整的JavaScript实现包含三个部分：核心（ECMAScript），文档对象模型（DOM）和浏览器对象模型（BOM）。

- **JavaScript与java的区别**

基于对象和面向对象：

Java是一种真正的面向对象的语言，即使是开发简单的程序，必须设计对象。

JavaScript是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object Based）和事件驱动（Event Driver）的编程语言。因而它本身提供了非常丰富的内部对象供设计人员使用。

解释和编译：两种语言在其浏览器中所执行的方式不一样。

Java的源代码在传递到客户端执行之前，必须经过编译，因而客户端上必须具有相应平台上的仿真器或解释器（Java虚拟机）。

JavaScript是一种解释性编程语言，是将文本格式的字符代码发送给客户端，由浏览器解释执行。

JavaScript基本语法

JavaScript用法

JavaScript导入方式：内部直接使用，外部JavaScript导入

内部直接插入JavaScript时需使用<script>标签，<script></script>代表着JavaScript的开始和结束，中间代码为要执行的JavaScript内容。

script标签中有时会使用type="text/javascript"属性，也可以不用写，因为JavaScript是所有现代浏览器及HTML5中的默认脚本语言。

script可以出现在html中的任意位置，一般都会写在head标签中。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<title>JavaScript导入</title>
<script type="text/javascript"src="js/script.js"></script>
<script type="text/javascript">
    alert("内部导入，弹出框");
    document.write("内部导入，写入页面");
</script>
</head>
<body>
</body>
</html>
```

有时我们页面中会出现很多script，不同页面的有些JavaScript具有相同的脚本，为了是页面看起来美观，也为了是脚本共用，这是我们可以引入外部JavaScript来达到此目的。

外部脚本引入一个JavaScript文件，文件以.js结尾。

外部的.js文件不能包含script标签。

导入外部JavaScript文件的script标签中不能再写JavaScript代码。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<title>JavaScript导入</title>
<script type="text/javascript"src="js/script.js"></script>
</head>
<body>
</body>
</html>
```

JavaScript注释

JavaScript 注释可用于提高代码的可读性。

JavaScript 不会执行注释。

我们可以添加注释来对 JavaScript 进行解释，或者提高代码的可读性。

单行注释以 // 开头。

多行注释以 /* 开始，以 */ 结尾。

JavaScript变量

变量是存储信息的容器。

与代数一样，JavaScript 变量可用于存放值（比如 x=2）和表达式（比如 z=x+y）。

变量可以使用短名称（比如 x 和 y），也可以使用描述性更好的名称（比如 age, sum等）。

- 变量必须以字母开头
- 变量也能以 $ 和 _ 符号开头（不过我们不推荐这么做）
- 变量名称对大小写敏感（y 和 Y 是不同的变量）

提示：JavaScript 语句和 JavaScript 变量都对大小写敏感。

> **变量声明和使用**

在 JavaScript 中创建变量通常称为"声明"变量。

我们使用 var 关键词来声明变量，变量声明之后，该变量是空的（它没有值）。

如：var a;

变量声明后可进行赋值。

a="Volvo";

您也可以在声明变量时对其赋值。

var a="Volvo";

您可以在一条语句中声明很多变量。该语句以 var 开头，并使用逗号分隔变量即可。

如：var name="Gates", age=56, job="CEO";

在计算机程序中，经常会声明无值的变量。未使用值来声明的变量，其值实际上是 undefined。

```
<script>
  var x;
  x=4;
  var y=6;
  var a="Lily",b="joy",c="Tom";
</script>
```

JavaScript数据类型

JavaScript 拥有动态类型。这意味着相同的变量可用作不同的类型。

JavaScript数据类型包括字符串、数字、布尔、数组、对象、Null、Undefined。

> **JavaScript字符串**

字符串是存储字符的变量。

字符串可以是引号中的任意文本。您可以使用单引号或双引号。

您可以在字符串中使用引号，只要不匹配包围字符串的引号即可。

```
<script>
  var str="my script";
  var str='my script';
  var str="this is 'javascript'";
</script>
```

JavaScript 数字

JavaScript 只有一种数字类型。数字可以带小数点，也可以不带。

极大或极小的数字可以通过科学（指数）计数法来书写。

```
<script>
  var x=34.00;      //使用小数点来写
  var y=34;         //不使用小数点来写
  var a=12e5;      // 1200000
  var b=12e-5;     // 0.0012
</script>
```

JavaScript 布尔

布尔（逻辑）只能有两个值：true 或 false。

```
<script>
  var x=true;
  var y=false;
</script>
```

JavaScript 数组

数组下标是基于零的，所以第一个项目是 [0]，第二个是 [1]，以此类推。

```
<script>
  var cars=new Array();
  cars[0]="Audi";
  cars[1]="BMW";
  cars[2]="Volvo";
  //等同于
  var cars=new Array("Audi","BMW","Volvo");
  //等同于
  var cars=["Audi","BMW","Volvo"];
</script>
```

JavaScript 对象

对象由花括号分隔。在括号内部，对象的属性以名称和值对的形式 (name : value) 来定义。属性由逗号分隔。

```
<script>
  var person={firstname:"Bill", lastname:"Gates", id:5566};
  //等同于
  var person={
    firstname : "Bill",
    lastname  : "Gates",
    id        :  5566
    };
</script>
```

对象中空格和折行无关紧要。声明可横跨多行。

对象属性有两种获取方式：

```
name=person.lastname;
name=person["lastname"];
```

后续将详细介绍JavaScript对象

> **Undefined 和 Null**

Undefined 这个值表示变量不含有值。

可以通过将变量的值设置为 null 来清空变量。

```
var cars=null;
var person=null;
```

声明变量类型

当您声明新变量时，可以使用关键词 "new" 来声明其类型，如：

```
<script>
  var str=    new String;
  var x=      new Number;
  var y=      new Boolean;
  var cars=   new Array;
  var person= new Object;
</script>
```

JavaScript 变量均为对象。当您声明一个变量时，就创建了一个新的对象(具体参考JavaScript常用对象)。

> 综合案例

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JavaScript基本语法</title>
        <script type="text/javascript">
            //JavaScript变量赋值并输出
            var a = 1; //整数Number
            var a1 = "这是一个变量"; //字符串String
            var a2=1.02; //浮点型Number
            var a3=true; //boolean类型
            var a4; //undefined类型(未初始化变量)
            //var arr=["第一组","第二组","第三组"]; //数组类型
            var arr=new Array();
            arr[0]="第一组0";
            arr[1]="第二组1";
            arr[2]="第三组2";
            alert(a+" || "+a1+" || "+a2+" || "+a3+" || "+a4);
            alert(arr);
        </script>
    </head>
    <body>
    </body>
</html>
```

JavaScript运算符

运算符 = 用于给 JavaScript 变量赋值。

算术运算符

算术运算符用于执行变量与/或值之间的算术运算。

运算符和示例：y=5时；

运算符	描述	例子	结果
+	加	x=y+2	x=7
-	减	x=y-2	x=3
*	乘	x=y*2	x=10
/	除	x=y/2	x=2.5
%	求余数 (保留整数)	x=y%2	x=1
++	累加	x=++y	x=6
--	递减	x=--y	x=4

赋值运算符

赋值运算符用于给 JavaScript 变量赋值。

运算符和示例：x=10，y=5时；

运算符	例子	等价于	结果
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

字符串连接运算符

+ 运算符用于把文本值或字符串变量加起来（连接起来）。

如需把两个或多个字符串变量连接起来，请使用 + 运算符。

如果把数字与字符串相加，结果将成为字符串。

```
<script>
  var a="my";
  var b="JavaScript";
  var c=a+""+b;    //c结果为my JavaScript
  var x="3";
  var y=4;
  var z=x+y;    //z结果为34
</script>
```

比较运算符

比较运算符在逻辑语句中使用，以测定变量或值是否相等。

运算符和示例：x=5；

运算符	描述	例子
==	等于	x==8 为 false
===	全等（值和类型）	x===5 为 true；x==="5" 为 false
!=	不等于	x!=8 为 true
>	大于	x>8 为 false
<	小于	x<8 为 true
>=	大于或等于	x>=8 为 false
<=	小于或等于	x<=8 为 true

逻辑运算符

逻辑运算符用于测定变量或值之间的逻辑。

运算符及示例：x=6 以及 y=3；

运算符	描述	例子
&&	and	(x < 10 && y > 1) 为 true
||	or	(x==5 || y==5) 为 false
!	not	!(x==y) 为 true

条件运算符

JavaScript 还包含了基于某些条件对变量进行赋值的条件运算符。

语法：var name=(condition)?value1:value2;

```
var x=(y==5)?"今天上课":"今天自习";
```

> 综合案例

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JavaScript运算符</title>
        <script type="text/javascript">
            var a=5;
            var b=3;
            document.write("算术运算符：<br/>");
            var c=a+b;
            document.write("a+b="+c+"<br/>");
            document.write("a-b="+(a-b)+"<br/>");
```

```
            document.write("a*b="+a*b+"<br/>");
            document.write("a/b="+(a/b)+"<br/>");
            document.write("a%b="+a%b+"<br/>");
            document.write("a++="+(a++)+"<br/>"); //先赋值，再加1
            document.write("++a="+(++a)+"<br/>"); //先加1，在赋值
            document.write("b--="+(b--)+"<br/>"); //先赋值，再减1
            document.write("--b="+(--b)+"<br/>"); //先减1，在赋值
            document.write("赋值运算符：<br/>");
            var x=6;
            var y=x;
            document.write("y=x，    y="+y+",x="+x+"<br />");
             y+=x;
            document.write("y+=x，y="+y+",x="+x+"<br />");
             y-=x;
            document.write("y-=x，y="+y+",x="+x+"<br />");
             y*=x;
            document.write("y*=x，y="+y+",x="+x+"<br />");
             y/=x;
            document.write("y/=x，y="+y+",x="+x+"<br />");
             y%=x;
            document.write("y%=x，y="+y+",x="+x+"<br />");
            document.write("字符串链接运算符(+)：<br/>");
            document.write("java+网页设计="+("java"+"网页设计")+"<br/>");
            document.write("2+3="+(2+3)+"<br/>");
            document.write('"2"+"3"='+"2"+"3"+"<br/>");
            document.write('"2"+3='+"2"+"3"+"<br/>");
            document.write("比较运算符：i=12,j=10<br/>");
            var i=12;
            var j=10;
            document.write("i==j ->"+(i==j)+"<br/>");
            document.write("i!=j ->"+(i!=j)+"<br/>");
            document.write("i>j ->"+(i>j)+"<br/>");
            document.write("i&lt;j ->"+(i<j)+"<br/>");
            document.write("i>=j ->"+(i>=j)+"<br/>");
            document.write("i<=j ->"+(i<=j)+"<br/>");
            document.write("逻辑运算符：i=12,j=10<br/>");
            document.write("i<10 && j>1 ->"+(i<10 &&j>1)+"<br/>");
            document.write("i<10 || j>1 ->"+(i<10 || j>1)+"<br/>");
            document.write("!(i==j) ->"+(!(i==j))+"<br/>");
document.write("条件运算符：<br />");
            var week=7;
            document.write((week==7)?"今天休息":"今天上课");
        </script>
    </head>
    <body>
    </body>
</html>
```

JavaScript控制语句

选择语句

也叫条件语句，用于基于不同的条件来执行不同的动作。

在 JavaScript 中，我们可使用以下条件语句：

- if 语句 - 只有当指定条件为 true 时，使用该语句来执行代码
- if...else 语句 - 当条件为 true 时执行代码，当条件为 false 时执行其他代码
- if...else if....else 语句 - 使用该语句来选择多个代码块之一来执行
- switch 语句 - 使用该语句来选择多个代码块之一来执行

If 语句

只有当指定条件为 true 时，该语句才会执行代码。

语法：

```
if (条件)
{
  只有当条件为 true 时执行的代码
}
```

注意：请使用小写的 if。使用大写字母（IF）会生成 JavaScript 错误！

If...else 语句

请使用 if....else 语句在条件为 true 时执行代码，在条件为 false 时执行其他代码。

语法：

```
if (条件)
{
   当条件为 true 时执行的代码
}
else
{
   当条件不为 true 时执行的代码
}
```

If...else if...else 语句

使用 if....else if...else 语句来选择多个代码块之一来执行。

语法：

```
if (条件 1)
{
   当条件 1 为 true 时执行的代码
}
else if (条件 2)
{
   当条件 2 为 true 时执行的代码
}
else
{
   当条件 1 和 条件 2 都不为 true 时执行的代码
}
```

> **switch语句**

工作原理：首先设置表达式 n（通常是一个变量）。随后表达式的值会与结构中的每个 case 的值做比较。如果存在匹配，则与该 case 关联的代码块会被执行。请使用 break 来阻止代码自动地向下一个 case 运行。

使用 switch 语句来选择要执行的多个代码块之一。

使用 default 关键词来规定匹配不存在时做的事情。

语法：

```
switch(n)
{
  case 1:
    执行代码块 1
    break;
  case 2:
    执行代码块 2
    break;
  default:
    n 与 case 1 和 case 2 不同时执行的代码
}
```

循环语句

循环可以将代码块执行指定的次数。

JavaScript 支持不同类型的循环：

- for - 循环代码块一定的次数
- for/in - 循环遍历对象的属性
- while - 当指定的条件为 true 时循环指定的代码块
- do/while - 同样当指定的条件为 true 时循环指定的代码块

> **for 循环**

语法：

```
for (语句 1; 语句 2; 语句 3)
{
  被执行的代码块
}
```

语句 1 在循环（代码块）开始前执行，通常我们会使用语句 1 初始化循环中所用的变量。

语句 2 定义运行循环（代码块）的条件，通常语句 2 用于评估初始变量的条件。

语句 3 在循环（代码块）已被执行之后执行，通常语句 3 会增加初始变量的值。

> **for in 循环**

语法：

```
for (对象中的变量 in 对象)
{
   要执行的代码
}
```

> **while循环**

语法：

```
while (条件)
{
   需要执行的代码
}
```

> **do/while 循环**

do/while 循环是 while 循环的变体。该循环会执行一次代码块，在检查条件是否为真之前，然后如果条件为真的话，就会重复这个循环。

语法：

```
do
{
  需要执行的代码
}
while (条件);
```

- **跳转语句**

break 语句用于跳出循环。

continue 用于跳过循环中的一个迭代。

break 语句

break 语句可用于跳出循环。

break 语句跳出循环后，会继续执行该循环之后的代码。

```
var x ="";
for (var i=0;i<4;i++)
     {
if (i==2)
          {
break;
          }
      x=x + "The number is "+ i + "<br>";
     }
```

执行结果：

```
The number is 0
The number is 1
```

continue 语句

continue 语句中断循环中的迭代，如果出现了指定的条件，然后继续循环中的下一个迭代。

```
var x ="";
for (var i=0;i<4;i++)
     {
if (i==2)
          {
continue;
          }
      x=x + "The number is "+ i + "<br>";
     }
```

执行结果：

```
The number is 0
The number is 1
The number is 3
```

综合案例

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JavaScript控制语句</title>
        <script type="text/javascript">
            var arr=new Array();
            arr[0]="第一组";
```

```
            arr[1]="第二组";
            arr[2]="第三组";
            document.write("选择语句：<br/>");
            document.write("if..else语句：<br/>");
            if(arr.length<=3){
                document.write("数组长度为"+arr.length+"<br />");
            } else {
                document.write("数组越界，数组长度为"+arr.length+"<br />");
            }
            document.write("switch..case语句：<br/>");
            var i=2;
            switch(i) {
                case 0:
                    document.write("arr[0]="+arr[0]+"<br />");
                    break;
                case 1:
                    document.write("arr[1]="+arr[1]+"<br />");
                    break;
                case 2:
                    document.write("arr[2]="+arr[2]+"<br />");
                    break;
                default:
                    document.write("无数据");
            }

            document.write("循环语句：<br />");
            document.write("for循环：<br />");
            for (var i=0;i<arr.length;i++){
                document.write(arr[i] + "<br />");
            }
            document.write("for..in循环：<br />");
            var x=0;
            for (x in arr){
                document.write(arr[x] + "<br />");
            }
            document.write("while循环：<br />");
            var i=0;
            while (i<arr.length){
                document.write(arr[i] + "<br>");
                i++;
            }
            document.write("do..while循环：<br />");
            var j=0;
            do{
                document.write(arr[j] + "<br>");
                j++;
            }while(j<arr.length);
            document.write("跳转语句：<br/>");
            for (var i=0;i<arr.length;i++){
                if(arr[i]=="第二组"){
                    //continue; //继续执行下一个
                    break;//跳出循环
                }
                document.write(arr[i] + "<br />");
            }

        </script>
    </head>
```

```
    <body>
    </body>
</html>
```

02 JavaScript函数

函数是由事件驱动的或者当它被调用时执行的可重复使用的代码块。

JavaScript函数创建与使用

函数就是包裹在花括号中的代码块，前面使用了关键词 function：

```
function functionName()
{
  这里是要执行的代码
}
```

当调用该函数时，会执行函数内的代码。

提示：JavaScript 对大小写敏感。关键词 function 必须是小写的，并且必须以与函数名称相同的大小写来调用函数。

> **调用带参数的函数**

在调用函数时，您可以向其传递值，这些值被称为参数。

这些参数可以在函数中使用。

您可以发送任意多的参数，由逗号 (,) 分隔：myFunction(argument1,argument2)

当您声明函数时，请把参数作为变量来声明：

```
function myFunction(var1,var2)
{
   这里是要执行的代码
}
```

变量和参数必须以一致的顺序出现。第一个变量就是第一个被传递的参数的给定的值，以此类推。

> **带有返回值的函数**

有时，我们会希望函数将值返回调用它的地方。

通过使用 **return** 语句就可以实现。

在使用 return 语句时，函数会停止执行，并返回指定的值。

语法：

```
function myFunction()
{
    执行的代码内容;
    return 返回值;
}
```

注释：整个 JavaScript 并不会停止执行，仅仅是函数。JavaScript 将继续执行代码，从调用函数的地方。

函数调用将被返回值取代：var myVar=myFunction();

JavaScript函数调用

函数间相互调用

JavaScript 函数有 4 种调用方式。

每种方式的不同在于 this 的初始化。

一般而言，在JavaScript中，this指向函数执行时的当前对象。

- **作为一个函数调用**

实例：

```
function myFunction(a, b) {
    return a * b;
 }
myFunction(10, 2);
// myFunction(10, 2) 返回 20
```

- **函数作为方法调用**

在 JavaScript 中你可以将函数定义为对象的方法。

实例：创建了一个对象 (myObject), 对象有两个属性 (firstName 和 lastName), 及一个方法 (fullName):

```
var myObject = {
    firstName:"John",
    lastName: "Doe",
    fullName: function () {
    return this.firstName + ""+ this.lastName; }
}
 myObject.fullName(); // 返回 "John Doe"
```

- **使用构造函数调用函数**

如果函数调用前使用了 new 关键字, 则是调用了构造函数。

这看起来就像创建了新的函数，但实际上 JavaScript 函数是重新创建的对象：

实例：

```
function myFunction(arg1, arg2) {
   this.firstName = arg1;
   this.lastName = arg2;
}
var x = new myFunction("John","Doe");
x.firstName;   // 返回 "John"
```

- **作为函数方法调用函数**

在 JavaScript 中, 函数是对象。JavaScript 函数有它的属性和方法。

call() 和 apply() 是预定义的函数方法。 两个方法可用于调用函数，两个方法的第一个参数必须是对象本身。

实例1：

```
function myFunction(a, b) {
   return a * b;
}
var myObject = myFunction.call(myObject, 10, 2);
// 返回 20
```

实例2：

```
function myFunction(a, b) {
   return a * b;
}
myArray = [10, 2];
myObject = myFunction.apply(myObject, myArray);
// 返回 20
```

两个方法实例都使用了对象本身作为第一个参数。 两者的区别在于第二个参数： apply传入的是一个参数数组，也就是将多个参数组合成为一个数组传入，而call则作为call的参数传入（从第二个参数开始）。

JavaScript函数中的变量

- **局部 JavaScript 变量**

在 JavaScript 函数内部声明的变量（使用 var）是局部变量，所以只能在函数内部访问它。（该变量的作用域是局部的）。

您可以在不同的函数中使用名称相同的局部变量，因为只有声明过该变量的函数才能识别出该变量。

只要函数运行完毕，本地变量就会被删除。

- **全局 JavaScript 变量**

在函数外声明的变量是全局变量，网页上的所有脚本和函数都能访问它。

- **JavaScript 变量的生存期**

JavaScript 变量的生命期从它们被声明的时间开始。

局部变量会在函数运行以后被删除。

全局变量会在页面关闭后被删除。

> **综合案例**

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JavaScript函数</title>
        <script type="text/javascript">
            function myFunction(){
                alert("这是一个click函数");
            }
            function myFirst(){
                alert("这是第一个函数");
                mySecond();
            }
            function mySecond(){
                alert("这是第二个函数");
            }
            function myParam(i,j) {
                alert(i+"+"+j+"="+(i+j));
            }
        </script>
    </head>
    <body>
        <p>事件函数：</p>
        <input type="button"onclick="myFunction()"value="点击事件函数"/>
        <p>函数间调用：</p>
        <input type="button"onclick="myFirst()"value="两个函数间调用"/>
        <p>参数传递：</p>
        3+2 = <input type="button"onclick="myParam(3,2)"value="点击产出结果"/>
    </body>
</html>
```

03 JavaScript对象

JavaScript对象介绍

JavaScript 中的所有事物都是对象：字符串、数值、数组、函数...

此外，JavaScript 允许自定义对象。

JavaScript 提供多个内建对象，比如 String、Date、Array 等等。

对象只是带有属性和方法的特殊数据类型。

- **访问对象的属性**

属性是与对象相关的值。

访问对象属性的语法是：objectName.propertyName

- **访问对象的方法**

方法是能够在对象上执行的动作。

您可以通过以下语法来调用方法：objectName.methodName()

- **创建 JavaScript 对象**

通过 JavaScript，您能够定义并创建自己的对象。

创建新对象有两种不同的方法：

定义并创建对象的实例

使用函数来定义对象，然后创建新的对象实例

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JavaScript对象</title>
        <script type="text/javascript">
            var str = new String("myScript");
            //String对象方法concat链接字符串
            alert(str.concat("Object"));
//创建一个person对象并赋值
            var person = {
                userId:"1001",
                userName:"Joy",
                password:"123456",
                myObject: function myObject(){
                    alert("这是person对象的一个方法");
                }
                };
            //获取对象属性
            var username = person.userName;
            alert(username);
            //执行person对象的方法
            person.myObject();
        </script>
    </head>
    <body>
    </body>
</html>
```

下面将介绍几种常用的对像。

Boolean（布尔）对象

Boolean 对象表示两个值："true" 或 "false"。

创建 Boolean 对象的语法：

```
new Boolean(value); //构造函数
Boolean(value);     //转换函数
```

> **参数**

参数 *value* 由布尔对象存放的值或者要转换成布尔值的值。

> **返回值**

当作为一个构造函数（带有运算符 new）调用时，Boolean() 将把它的参数转换成一个布尔值，并且返回一个包含该值的 Boolean 对象。

如果作为一个函数（不带有运算符 new）调用时，Boolean() 只将把它的参数转换成一个原始的布尔值，并且返回这个值。

如果省略 value 参数，或者设置为 0、-0、null、""、false、undefined 或 NaN，则该对象设置为 false。否则设置为 true（即使 value 参数是字符串 "false"）。

Boolean 对象常用方法

方法	描述
toString()	把逻辑值转换为字符串，并返回结果。
valueOf()	返回 Boolean 对象的原始值。

Number 对象

Number 对象是原始数值的包装对象。

创建 Number 对象的语法：

```
var myNum=new Number(value);
var myNum=Number(value);
```

> **参数**

参数 *value* 是要创建的 Number 对象的数值，或是要转换成数字的值。

> **返回值**

当 Number() 和运算符 new 一起作为构造函数使用时，它返回一个新创建的 Number 对象。如果不用 new 运算符，把 Number() 作为一个函数来调用，它将把自己的参数转换成一个原始的数值，并且返回这个值（如果转换失败，则返回 NaN）。

Number 对象常用属性

属性	描述
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。

Number 对象常用方法

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。

Math（算数）对象

Math 对象用于执行数学任务。

使用 Math 的属性和方法的语法：

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

Math 对象并不像 Date 和 String 那样是对象的类，因此没有构造函数 Math()，像Math.sin() 这样的函数只是函数，不是某个对象的方法。您无需创建它，通过把 Math 作为对象使用就可以调用其所有属性和方法。

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。
valueOf()	返回 Math 对象的原始值。

字符串（String）对象

String 对象用于处理文本（字符串）。

创建 String 对象的语法：

```
new String(s);
 String(s);
```

参数

参数 *s* 是要存储在 String 对象中或转换成原始字符串的值。

返回值

当 String() 和运算符 new 一起作为构造函数使用时，它返回一个新创建的 String 对象，存放的是字符串 *s* 或 *s* 的字符串表示。

当不用 new 运算符调用 String() 时，它只把 *s* 转换成原始的字符串，并返回转换后的值。

String 对象属性

属性	描述
length	字符串的长度

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。
concat()	连接字符串。
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。
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。
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。
toString()	返回字符串。
valueOf()	返回某个字符串对象的原始值。

Date（日期）对象

Date 对象用于处理日期和时间。

创建 Date 对象的语法：

```
var myDate=new Date();
```

Date 对象会自动把当前日期和时间保存为其初始值。

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。
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。
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)。

方法	描述
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)。
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。
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。

Array（数组）对象

Array 对象用于在单个的变量中存储多个值。

创建 Array 对象的语法：

```
new Array();
new Array(size);
new Array(element0, element1, ..., elementn);
```

参数

参数 *size* 是期望的数组元素个数。返回的数组，length 字段将被设为 *size* 的值。

参数 *element1 ..., elementn* 是参数列表。当使用这些参数来调用构造函数 Array() 时，新创建的数组的元素就会被初始化为这些值。它的 length 字段也会被设置为参数的个数。

返回值

返回新创建并被初始化了的数组。

如果调用构造函数 Array() 时没有使用参数，那么返回的数组为空，length 字段为 0。

当调用构造函数时只传递给它一个数字参数，该构造函数将返回具有指定个数、元素为 undefined 的数组。

当其他参数调用 Array() 时，该构造函数将用参数指定的值初始化数组。

当把构造函数作为函数调用，不使用 new 运算符时，它的行为与使用 new 运算符调用它时的行为完全一样。

Array 对象属性

属性	描述
length	设置或返回数组中元素的数目。

Array 对象方法

方法	描述
concat()	连接两个或更多的数组，并返回结果。
join()	把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。
pop()	删除并返回数组的最后一个元素
push()	向数组的末尾添加一个或更多元素，并返回新的长度。
reverse()	颠倒数组中元素的顺序。
shift()	删除并返回数组的第一个元素
slice()	从某个已有的数组返回选定的元素
sort()	对数组的元素进行排序
splice()	删除元素，并向数组添加新元素。
toString()	把数组转换为字符串，并返回结果。
unshift()	向数组的开头添加一个或更多元素，并返回新的长度。
valueOf()	返回数组对象的原始值

> **综合案例**

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JavaScript对象</title>
        <script type="text/javascript">
            function boolDom(){
                var mybool1 = new Boolean("123"); //构造函数
                var mybool2 = Boolean("");  //转换函数
                alert("作为构造函数返回："+mybool1 + "，作为一个函数返回:"+mybool2);
            }
            function numDom(){
                var myNum1 = new Number("123"); //构造函数
                var myNum2 = Number("abc"); //转换函数(错误信息返回NaN)
```

```
                alert("作为构造函数返回："+myNum1 + "，作为一个函数返回:"+myNum2);
                alert("获取最大Number："+Number.MAX_VALUE);
                alert("Number构造函数对象转成String"+myNum1.toString());
            }
            function strDom(){
                var myStr1 = new String("myscript"); //构造函数
                var myStr2 = String("abc"); //转换函数
                alert("作为构造函数返回："+myStr1 + "，作为一个函数返回:"+myStr2);
                alert("获取myscript长度："+myStr1.length);
                alert("字符串相加："+myStr2.concat("123"));
            }
            function dateDom(){
                var myDate = new Date();
                alert("返回当前时间："+myDate.getHours()+"时"
                    +myDate.getMinutes()+"分"+myDate.getSeconds()+"秒");
            }
            function mathDom(){
                var pi = Math.PI;
                alert("圆周率值为："+pi);
                var x=25;
                alert("返回25的平方根："+Math.sqrt(x));
            }
            function arrayDom(){
                var arr=new Array("tom",2,"joy");
                alert("数组长度："+arr.length);
                alert("原数组："+arr.toString()+"数组排序后:"+arr.sort());
            }
        </script>
    </head>
    <body>
        <p><input type="button"onclick="boolDom()"value="boolean对象"/></p>
        <p><input type="button"onclick="numDom()"value="Number对象"/></p>
        <p><input type="button"onclick="strDom()"value="String对象"/></p>
        <p><input type="button"onclick="dateDom()"value="Date对象"/></p>
        <p><input type="button"onclick="mathDom()"value="Math对象"/></p>
        <p><input type="button"onclick="arrayDom()"value="Array对象"/></p>
    </body>
</html>
```

Event对象

event代表事件的状态，例如触发event对象的元素、鼠标的位置及状态、按下的键等等。event对象只在事件发生的过程中才有效。event的某些属性只对特定的事件有意义。比如，fromElement 和 toElement 属性只对 onmouseover 和 onmouseout 事件有意义。

Event属性

属性名	描述	值	说明
altKey	检查alt键的状态	当alt键按下时，值为True否则为False	只读
shiftKey	检查shift键的状态	当shift键按下时，值为True否则为False	只读
ctrlKey	检查ctrl键的状态	当ctrl键按下时，值为True否则为False	只读
keyCode	检测键盘事件相对应的内码		可读写，可以是任何一个Unicode键盘内码。如果没有引发键盘事件，则该值为0
srcElement	返回触发事件的元素	Object	只读
x,y	鼠标相对于当前浏览器的位置	px	只读
clientX,clientY	鼠标当前相对于网页的位置	px	只读
offsetX,offsetY	鼠标当前相对于网页中的某一区域的位置	px	只读
screenX,screenY	相对于用户显示器的位置	px	只读
returnValue	设置或检查从事件中返回的值	true 事件中的值被返回 false 源对象上事件的默认操作被取消	可读写
button	检查按下的鼠标键	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	仅用于 onmousedown，onmouseup 和 onmousemove 事件。对其他事件，不管鼠标状态如何，都返回0（比如 onclick）

属性名	描述	值	说明
srcElement	检测onmouseover和onmouseout事件发生时，鼠标所离开的元素	Object	只读
toElement	检测onmouseover和onmouseout事件发生时，鼠标所进入的元素	Object	只读
type	返回事件名		返回没有“on”作为前缀的事件名，比如，onclick事件返回的type是click

属性相关案例

- 点击按钮时显示几个特殊键按下的状态

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <input type="button"value="点击" onkeydown="showState()"/>
        <script>
            function showState(){
             alert("keyCode: "+window.event.keyCode
              +"altKey: "+window.event.altKey
              +"\nshiftKey: "+window.event.shiftKey
              +"\nctrlKey: "+window.event.ctrlKey);
            }
        </script>
    </body>
</html>
```

以上代码在Firefox中执行，没有任何提示。IE 和 Chrome中执行显示正确结果。由此说明window.event 只能在IE下运行，而不能在Firefox下运行，这是因为Firefox的event只能在事件发生的现场使用。

> **解决浏览器兼容方法**

(1) **window.event问题**

a. 在 函数中传递event参数

在函数中传递event参数，这样我们就可以兼容IE和FF的event的获取了。

上面代码可改成如下的函数：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <input type="button"value="点击"onclick="showState(event)"/>
        <script>
            function showState(evt){
                if(typeof(window.event) == 'undefined') {
                    alert("altKey: "+evt.altKey
                        +"\nshiftKey: "+evt.shiftKey
                        +"\nctrlKey: "+evt.ctrlKey);
                }else {
                    alert("altKey: "+window.event.altKey
                        +"\nshiftKey: "+window.event.shiftKey
                        +"\nctrlKey: "+window.event.ctrlKey);
                }
            }
        </script>
    </body>
</html>
```

b. 在函数调用中不传递event对象

虽然在函数中没有传递参数，这个在IE下没有任何影响，因为window.event是全局对象，在什么地方都可以直接调用的，而在Firefox下就不行了。所以我们这里要使用另外一种方式来获取了，如下：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <input type="button"value="点击"
onclick="showState()"/>
        <script>
            function showState(){
                var evt = window.event || arguments.callee.caller.arguments[0];
// 获取event对象
                alert("altKey: "+evt.altKey
                    +"\nshiftKey: "+evt.shiftKey
                    +"\nctrlKey: "+evt.ctrlKey);
            }
        </script>
    </body>
```

```
</html>
```

(2) **event.x与event.y问题**

IE下,event对象有x,y属性,但是没有pageX,pageY属性;Firefox下,event对象有pageX,pageY属性,但是没有x,y属性。

解决方法:使用mX(mX = event.x ? event.x : event.pageX;)来代替IE下的event.x或者Firefox下的event.pageX。

(3) **event.srcElement问题**

IE下,event对象有srcElement属性,但是没有target属性;Firefox下,event对象有target属性,但是没有srcElement属性。

解决方法:使用obj(obj = event.srcElement ? event.srcElement : event.target;)来代替IE下的event.srcElement或者Firefox下的event.target.

- 按回车键让下一组件得到焦点，相当按Tab键

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <input type="text"onKeyDown="nextBlur()"/>
        <input type="text"/>
        <script>
        function nextBlur(){
        var evt = window.event || arguments.callee.caller.arguments[0]; // 获取
event对象
        if(evt.keyCode==13)//回车键的 code
          evt.keyCode=9;//Tab键的code
        }
        </script>
    </body>
</html>
```

- 点击按钮时显示按钮的name值

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <input type="button"value="闽"name="福建"onClick="show()"/>
        <input type="button"value="赣"name="江西"onClick="show()"/>
        <script>
        function show(){
```

```
          var evt = window.event || arguments.callee.caller.arguments[0]; // 获取event对象
          alert(evt.srcElement.name);
            }
        </script>
    </body>
</html>
```

- 屏蔽鼠标右键、Ctrl+n、shift+F10、F5刷新、退格键

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <input type="text"onKeyDown="KeyDown()"/>
        <script>
            function KeyDown(){
            var evt = window.event || arguments.callee.caller.arguments[0]; //获取event对象
              //屏蔽鼠标右键、Ctrl+N、Shift+F10、F5刷新、退格键
              if ((evt.altKey)&&
                  ((evt.keyCode==37)||   //屏蔽 Alt+ 方向键 ←
                   (evt.keyCode==39))){  //屏蔽 Alt+ 方向键 →
                 evt.returnValue=false;//防止使用ALT+方向键前进或后退网页
              }
              if ((evt.keyCode==8) ||       //屏蔽退格删除键
                  (evt.keyCode==116)||   //屏蔽 F5 刷新键
                  (evt.keyCode==112)||   //屏蔽 F1 刷新键 bitsCN.com中国网管联盟
                  (evt.ctrlKey &&evt.keyCode==82)){ //Ctrl + R
                 evt.keyCode=0;
                 evt.returnValue=false;
              }
              if ((evt.ctrlKey)&&(evt.keyCode==78))   //屏蔽Ctrl+N
                 evt.returnValue=false;
              if ((evt.shiftKey)&&(evt.keyCode==121)) //屏蔽Shift+F10
                 evt.returnValue=false;
              if (evt.srcElement.tagName == "A"&&evt.shiftKey)
                  evt.returnValue = false;  //屏蔽 shift 加鼠标左键新开一网页
              if ((evt.altKey)&&(evt.keyCode==115)){ //屏蔽Alt+F4

 window.showModelessDialog("about:blank","","dialogWidth:1px;dialogHeight:1px");
                  return false;}
            }
        </script>
    </body>
</html>
```

04 JavaScript DOM使用

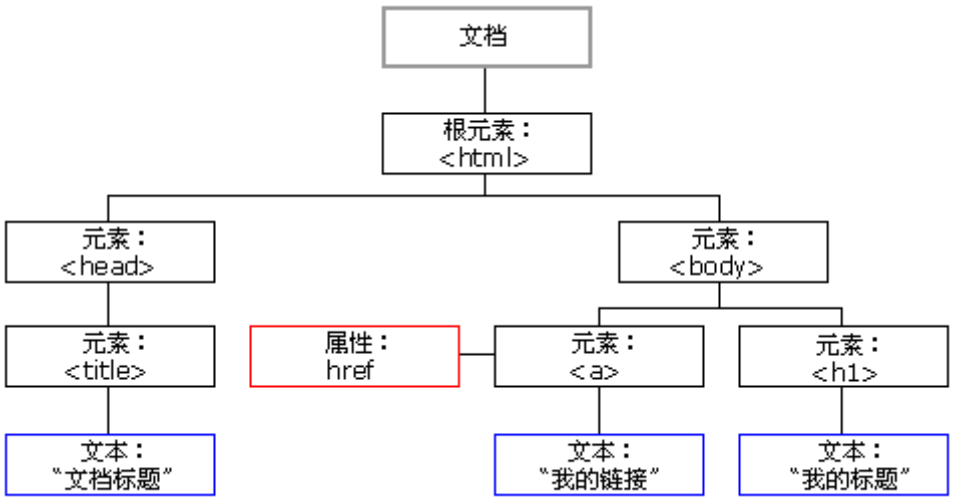
HTML DOM介绍

HTML DOM （文档对象模型）

当网页被加载时，浏览器会创建页面的文档对象模型（Document Object Model）。

HTML DOM 模型被构造为对象的树。

HTML DOM 树



通过可编程的对象模型，JavaScript 获得了足够的能力来创建动态的 HTML。

- JavaScript 能够改变页面中的所有 HTML 元素
- JavaScript 能够改变页面中的所有 HTML 属性
- JavaScript 能够改变页面中的所有 CSS 样式
- JavaScript 能够对页面中的所有事件做出反应

查找 HTML 元素

通常，通过 JavaScript，您需要操作 HTML 元素。

为了做到这件事情，您必须首先找到该元素。有四种方法来做这件事：

- 通过 id 找到 HTML 元素
- 通过标签名找到 HTML 元素
- 通过类名找到 HTML 元素
- 通过name找到HTML元素

通过 id 查找 HTML 元素

在 DOM 中查找 HTML 元素的最简单的方法，是通过使用元素的 id。

```
var x=document.getElementById("id属性名");
```

如果找到该元素，返回对拥有指定 id 的第一个对象的引用。

如果未找到该元素，则 x 将包含 null。

通过标签名查找 HTML 元素

```
var y=x.getElementsByTagName("标签名");
```

如果存在该元素，返回带有指定标签名的对象集合。

通过类名找到 HTML 元素

```
var x=document.getElementsByClassName("class属性名");
```

如果存在该元素，返回返回文档中所有指定类名的元素集合

提示：通过类名查找 HTML 元素在 IE 5,6,7,8 中无效。

通过name找到HTML元素

getElementsByName() 方法可返回带有指定名称的对象的集合。

> 语法：

~~~~
~~~js
  document.getElementsByName("name属性);
~~~
~~~~

该方法与 getElementById() 方法相似，但是它查询元素的 name 属性，而不是 id 属性。

另外，因为一个文档中的 name 属性可能不唯一（如 HTML 表单中的单选按钮通常具有相同的 name 属性），所有 getElementsByName() 方法返回的是元素的数组，而不是一个元素。

HTML DOM内容

HTML DOM 允许 JavaScript 改变 HTML 元素的内容。

改变HTML内容

JavaScript可通过**innerHTML** 属性获取和修改 HTML 内容 。

```
var x = document.getElementById(id).innerHTML; //获取内容

document.getElementById(id).innerHTML=新的 HTML; //修改内容
```

改变 HTML 属性

如需改变 HTML 元素的属性，请使用这个语法：

```
document.getElementById(id)setAttribute('属性','属性名');
```

如： document.getElementById(id).src="img.jpg";

改变 HTML 样式

如需改变 HTML 元素的样式，请使用这个语法

```
document.getElementById(id).style.property=new style;
```

如： document.getElementById("id").style.color="red";

> **综合案例**

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JavaScript DOM HTML</title>
    </head>
    <body>
        <div id="idIntro">通过id获取html内容</div>
        <div name="nameIntro">通过name获取html内容</div>
        <p>通过标签获取html内容</p>
        <div class="classIntro">通过class获取html内容</div>
        <img src="images/beijing.jpg"id="img"width="100px"height="90px"/><br />
        <input type="button"onclick="changeHtml()"value="改变html元素内容属性和样
式"/>
        <!--
    html页面加载方式是按顺序从上到下加载，如果将script放在head中，则会报错
       -->
        <script>
            //获取html内容
            var x=document.getElementById("idIntro");
            var y=document.getElementsByTagName("p")[0];
            var z=document.getElementsByClassName("classIntro")[0];
            var a=document.getElementsByName("nameIntro")[0];
            alert(a.innerHTML);
            function changeHtml() {
```

```
                //改变html内容
                x.innerHTML = "内容被改变了";
                // 改变html属性
                document.getElementById("img").src="images/renwu.jpg";
                //改变html样式
                x.style.color="red";
            }
        </script>
    </body>
</html>
```

JavaScript DOM事件

HTML DOM 使 JavaScript 有能力对 HTML 事件做出反应。

HTML 事件的例子：

- 当用户点击鼠标时
- 当鼠标移动到元素上时
- 当网页已加载时
- 当用户触发按键时
- 当输入字段被改变时
- 当提交 HTML 表单时

鼠标事件

鼠标事件包括onclick事件、ondblclick事件、onmouseout事件、onmouseover事件、onmousedown事件、onmouseup事件。

onmouseover 和 onmouseout 事件可用于在用户的鼠标移至 HTML 元素上方或移出元素时触发函数。

onmousedown, onmouseup 以及 onclick 构成了鼠标点击事件的所有部分。首先当点击鼠标按钮时，会触发 onmousedown 事件，当释放鼠标按钮时，会触发 onmouseup 事件，最后，当完成鼠标点击时，会触发 onclick 事件。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JavaScript事件</title>
        <style type="text/css">
            #mouse{
                background-color:green;
                width:200px;
                height:100px;
                color:#ffffff;
                text-align: center;
                display:table-cell;
                vertical-align:middle;
                font-size:20px;
            }
        </style>
```

```
        <script type="text/javascript">
            function clickDom(){
                alert("这是一个click函数");
            }
            function dblClickDom(){
                alert("这是一个dblclick函数");
            }
            function mOver(obj){
                obj.innerHTML="欢迎光临";
            }

            function mOut(obj){
                obj.innerHTML="谢谢惠顾";
            }
            function mDown(obj){
                obj.style.backgroundColor="red";
            }

            function mUp(obj){
                obj.style.backgroundColor="green";
            }
        </script>
    </head>
    <body>
        <p><input type="button"onclick="clickDom()"value="单击事件"/></p>
        <p><input type="button"ondblclick="dblClickDom()"value="双击击事件"/></p>
        <p>下面鼠标移动看看？再点击一下试试？</p>
        <div id="mouse"onmouseover="mOver(this)"onmouseout="mOut(this)"
            onmousedown="mDown(this)"onmouseup="mUp(this)"></div>
    </body>
</html>
```

键盘事件

键盘事件包括onkeydown事件和onkeyup事件，onkeydown事件为按下键盘时触发的事件，onkeyup为松开键盘时触发的事件。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JavaScript事件</title>
        <script type="text/javascript">
            function kUp(obj){
                alert("键盘按键被松开"+obj.value);
            }
            function kDown(obj){
                alert("键盘按键被按下"+obj.value);
            }
        </script>
    </head>
    <body>
        <p>键盘事件,观察键盘被按下和松开时的value值：</p>
        <p>按下键盘：<input type="text"onkeydown="kDown(this)"value="keyDown"/>
</p>
```

```
        <p>松开键盘：<input type="text"onkeyup="kUp(this)"value="keyUp"/></p>
    </body>
</html>
```

表单事件

表单事件包括onblur事件：当输入字段失去焦点时触发的事件；onchange事件：输入字段发生改变时触发的事件；onfocus事件：输入字段获取焦点时触发的事件；oninput事件：字段输入时触发的事件；onreset事件：按钮被点击时重置表单内容事件；onsubmit事件：按钮被点击时提交表单内容触发的事件。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JavaScript事件</title>
        <script type="text/javascript">
            function blurDom(obj){
                alert("光标移除："+obj.value);
            }
            function focusDom(obj){
                alert("光标获取："+obj.value);
            }
            function changeDom(obj){
                alert("内容改变后："+obj.value);
            }
            function inputDom(obj){
//              alert("输入信息时："+obj.value);
                document.getElementById("content").innerHTML="您输入的内容
为："+obj.value;
            }
            function resetDom(){
                alert("表单重置");
                document.getElementById("content").innerHTML="";
            }
            function submitDom(){
                alert("表单提交");
            }
        </script>
    </head>
    <body>
        <p>表单事件：</p>
            <p><input type="text"onblur="blurDom(this)"value="失去焦点"/></p>
            <p><input type="text"onfocus="focusDom(this)"value="获取焦点"/></p>
            <p><input type="text"onchange="changeDom(this)"value="值发生变化"/>
</p>
<p>表单提交：</p>
        <form action="#"onreset="resetDom()"onsubmit="submitDom()">
            <p><input type="text"oninput="inputDom(this)"/>
                <span id="content"></span></p>
            <p><input type="reset"value="重置事件"/></p>
            <p><input type="submit"value="提交事件"/></p>
        </form>
    </body>
</html>
```

05 JavaScript正则表达式

正则表达式介绍

- **什么是正则表达式？**

正则表达式，又称正规表达式、规则表达式（英语：Regular Expression，在代码中常简写为regex、regexp或RE），是计算机科学的一个概念。正则表达式使用单个字符串来描述、匹配一系列语法规则的字符串。在很多文本编辑器中，正则表达式通常被用来检索、替换那些匹配某种规则的文本。

Regular Expression的“Regular”一般被译为“正则”、“正规”、“常规”。此处的“Regular”即是“规则”、“规律”的意思，Regular Expression即“描述某种规则的表达式”之意。

- **正则表达式特点**

减少代码量；提高查询效率；

可以迅速地用极简单的方式达到字符串的复杂控制；

对于刚接触的人来说，比较晦涩难懂。

- **什么情况下使用正则表达式？**

数据验证

字符串查找

字符串替换

正则表达式匹配规则

- 普通匹配符：

  能够匹配与之对应的字符（默认正则区分大小写）

- 能够匹配多个字符中的其中一个匹配符:

  \d ---- 0~9

  \w ---- 字母、数字、下划线

  . ----- 匹配除换行的所有字符(\d \w . 都只能匹配一个)

- 能够自定义规则的匹配符：

  [] ,如果在[^]代表取反

- 用来修饰匹配次数的匹配符 ：

  {n}: 代表前面匹配符出现n次

  {n,m} ---- 出现次数 n~m之间

  {n,} ---- n~max

  +---- 1~max

  ? ---- 0~1

  *---- 0~max

- 正则表达式的完整匹配：

  ^ ---- []中^代表取反,但是在外面代表的是从开始匹配

  $ ---- 持续匹配的结束

- 特殊符号如何匹配在正则中：

  ^ $. \ [] 这些符号在匹配的时候需要加\

- 条件分支：

  | ---- 或

  () ---- 1: 括号中的内容,成为一个独立的整体!

  2: 括号的内容可以进行分组,单独匹配,不需要此功能则(?:)

- 数值的匹配 ：

  1:把合法的数值写出并分析规律

  2:根据规律编写正则,并且测试非法数值

- 中文的处理：

  默认中文采用的是双字节, 在计算机中通过ASCII对应表来输入汉字

  \u4E00-\u9FA5 ---- 中文取值范围

- 贪婪与懒惰(find()):

  在正则中默认是贪婪模式(尽可能多的匹配)

  可以在修饰数量的匹配符(* + ? {})后面添加? 则代表懒惰

Javascript正则表达式案例

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>正则基本语法</title>
<script type="text/javascript">
    // 说明: 正则在不同语言的支持方法语法略有不同  js、java、php ...
    var str = "Hello Java Hello java";
    // 1:在js中如果要引入正则表达式,则需要:/.../
    // 2:string.match(/表达式/) 匹配成功返回数组,否则为null
    // 3:普通匹配符: 能够匹配以之对应的字符
    // 4:默认正则区分大小写
    // 5: i、g、m 称为正则标记符(参数): i: 不区分大小写 g:全局匹配  m: 多行匹配
    var r=str.match(/java/gi);
    // 6:能够匹配多个字符中的其中一个匹配符: \d: 0~9  \w: 字母、数字、下划线  .:匹配除换
行的所有字符(\d \w . 都只能匹配一个)
    str = "hello 2016";
    r=str.match(/\w.\d/g);

    // 7:能够自定义规则的匹配符[],如果在[^]代表取反
    str = "198"// 第二数值 3|5|8|9
    r=str.match(/1[3589]\d/);
    // 8:用来修饰匹配次数的匹配符  {n}: 代表前面匹配符出现n次
```

```
    /*
        {n,m}: 出现次数 n~m之间   {n,}: n~max   {,n}: min~n
         +: 1~max  ?: 0~1  *: 0~max
    */
    str = "18312345678";
    r=str.match(/1[3589]\d{9}/);
    // 9:正则表达式的完整匹配  ^: []中^代表取反,但是在外面代表的是从开始匹配  $:持续匹配的
结束
    // 10:特殊符号如何匹配,在正则中: ^ $ . \ [] 这些符号在匹配的时候需要加\
    str = "15312345678";
    r=str.match(/^1[3589]\d{9}$/);
    str = "153^.$[]";
    r=str.match(/\^\.\$\[\]$/);
    // 11: 条件分支 |  2:(): 1: 括号中的内容,成为一个独立的整体! 2: 括号的内容可以进行分
组,单独匹配,不需要此功能则(?:)
    // 在添加^$ 完整匹配模式下如果完整匹配成功后面才会有分组匹配的功能.
    str = "12&3.jpeg";
    r=str.match(/\.(?:png|gif|jpe?g)$/);
    // 12: 数值的匹配   1:把合法的数值写出并分析规律   2:根据规律编写正则,并且测试非法数值
    str = "-12.34E5";
    r = str.match(/^(-?)(0|[1-9]\d*)(\.\d+)?([eE][-+]?\d+)?$/);
    // 13: 中文的处理 默认中文采用的是双字节, 在计算机中通过ASCII对应表来输入汉字
    //  来设置中文的范围即可 escape() 可以把字符串转化为ASCII编码   unescape() 可以把编
码转化为中文
    // console.info(escape('一') + "," + unescape(escape('一')) + "," +
escape('龥'));
    str = '欕eom硷eos碒fenwa覅fiao猤fui甴gad嗧geo喖geu嗰gib啯go兝gongfen兣';
    r=str.match(/[\u4E00-\u9FA5]/g);
    // 14: 贪婪与懒惰: 在正则中默认是贪婪模式(尽可能多的匹配) 可以在修饰数量的匹配符(* + ?
{})后面添加? 则代表懒惰
    str = "aabab";
    r=str.match(/a.*?b/g);
    console.info(r);
</script>
</head>
<body>
</body>
</html>
```