**JavaScript**

**1.HTML**定义了网页的内容

**2.CSS**描述了网页的布局

**3.JavaScript**网页的行为

**用法:**

HTML 中的(JavaScript)脚本语句必须位于 <script> 与 </script> 标签之间。

<body>: <body> 文本中的 JavaScript 语句会在页面加载时执行。

<head>: 通常的做法是把函数放入 <head> 部分中，就可以在事件发生时调用该函数。

外部： 外部 JavaScript 文件的文件扩展名是 .js

如需使用外部文件：<body> < script src="myScript.js" > </script> </body>

**输出：**

**window.alert()：** 弹出警告框。

**document.write()** ：将内容直接写到 HTML 文档中。

例：document.write(Date());

如果在文档已完成加载后执行 document.write，整个 HTML 页面将被覆盖。

**innerHTML：** 写入到 HTML 元素。

例：用过ID更改元素内容 document.getElementById("demo").innerHTML="段落已修改。"

**document.getElementById("demo")** 是用 id 属性来查找 HTML 元素

**console.log()** ：写入到浏览器的控制台（F12开启调试模式，Console为调试窗口）。

例：console.log(5); 窗口将显示输出 2

**JavaScript 变量:** 使用关键字 **var** 来定义变量

例： var length; length = 6;

# **JavaScript 数据类型：** JavaScript 拥有动态类型

字符串： var carname="Volvo XC60";

数字： var x1=34.00;

Boolean： var x=true;

数组： var cars=new Array(); cars[0]="Saab";

或： var cars=new Array("Saab","Volvo","BMW");

或： var cars=["Saab","Volvo","BMW"];

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。

例：var person={firstname:"John", lastname:"Doe", id:5566};

对象寻址：person.lastname; 或 person["lastname"]; 结果都为Doe

清空变量：cars=null;

声明变量类型：使用关键词 "new"  例：var carname=new String;

# **JavaScript 对象：** 对象也是一个变量，但对象可以包含多个值（多个变量）。

例：var person={firstname:"John", lastname:"Doe", id:5566};

访问对象属性：person.lastname; 或 person["lastname"];

访问对象方法：person.fullName(); （不加括号返回函数表达式）

# **JavaScript 函数：**使用关键词 function：

function functionname(var1,var2)  
 {  
 执行代码

（return x;）  
 }

# **JavaScript 作用域：**

局部变量：只能在函数内部访问。函数执行完后局部变量会自动销毁。

全局变量：定义在函数外。 网页中所有脚本和函数均可使用。

如果变量在函数内没有声明（没有使用 var 关键字），该变量为全局变量。

例： function myFunction() { carName = "Volvo"; }

全局变量是 window 对象: 所有数据变量都属于 window 对象。

上例carName 可以使用  window.carName

定义的全局变量，或者函数，可以覆盖 window 对象的变量或者函数。  
 局部变量，包括 window 对象可以覆盖全局变量和函数

# **JavaScript 事件：**事件触发时 JavaScript 可以执行一些代码。

HTML 事件的实例：

1. HTML 页面完成加载
2. HTML input 字段改变时
3. HTML 按钮被点击

常见的HTML事件的列表:

onchange HTML 元素改变

onclick 用户点击 HTML 元素

onmouseover 用户在一个HTML元素上移动鼠标

onmouseout 用户从一个HTML元素上移开鼠标

onkeydown 用户按下键盘按键

onload 浏览器已完成页面的加载

例：<button onclick='getElementById("demo").innerHTML=Date()'>11</button>

如果想修改元素本身：(使用 **this**.innerHTML) 🡺 onclick="this.innerHTML=Date()"

## JavaScript 字符串

## var carname = "Volvo XC60";

可以使用索引位置来访问字符串中的每个字符：var character = carname[7];

字符串长度：var sln = carname.length;

转义字符：使用反斜杠 (\) 来转义。 如 \' 表示单引号

字符串也可以是对象：  **var firstName = new String("John")**

var x = "John"; typeof x // 返回 String  
 var y = new String("John"); typeof y // 返回 Object

(x === y) //结果为 false，因为 x 是字符串，y 是对象

(=== 为绝对相等，即数据类型与值都必须相等。) (!==为值和类型至少有一个不等)

## For/In 循环

## var person={fname:"John",lname:"Doe",age:25};

## for (x in person) 🡺 键：x ，值：person[x]

## JavaScript 标签: 在语句之前加上冒号 例： label: statements

## break 和 continue 语句仅仅是能够跳出代码块的语句。 例：break labelname;

## continue labelname;

continue 语句（带有或不带标签引用）只能用在循环中。

break 语句（不带标签引用），只能用在循环或 switch 中。

通过标签引用，break 语句可用于跳出任何 JavaScript 代码块：

## JavaScript typeof, null, 和 undefined

## typeof :检测变量的数据类型。 例：typeof [1,2,3,4]   // 返回 object

（在JavaScript中，数组是一种特殊的对象类型。 因此 typeof [1,2,3,4] 返回 object。）

## Null：表示一个空对象引用。用 typeof 检测 null 返回是object。

## 例：var person = null;   // 值为 null(空), 但类型为object

## undefined ：没有设置值的变量。

## 例：var person = undefined; // 值为 undefined, 类型是undefined

## Undefined 和 Null 的区别：

## typeof undefined             // undefined typeof null                  // object null === undefined           // false null == undefined            // true

**JavaScript 类型转换**

5 种不同的数据类型：

String / number / Boolean / object / function

3 种对象类型（**typeof** 都是 返回 Object）：

Object / Date / Array

2不包含任何值的数据类型：

Null / undefined

constructor 属性返回变量或对象的构造函数。

例："john".constructor 返回function String() { [native code] }

可以使用 constructor 属性来判断对象是否为数组 (利用字符串是否包含 "Array"字符)

myArray.constructor.toString().indexOf("Array") > -1; 若为数组则返回true

**类型转换：**使用 JavaScript 函数 或通过 JavaScript 自身自动转换

## 数字转换为字符串：全局方法 **String()** 可以将数字转换为字符串。 例：String(123)

Number 方法 **toString()** 也是有同样的效果。 例：(123).toString()

## 布尔值转换为字符串：全局方法 **String()**  例：String(false)  // 返回 "false"

Boolean 方法 **toString() 例：**false.toString()

## 日期转换为字符串：全局方法 **String()**  例：String(Date())

**toString()**  例：Date().toString()

## 字符串转换为数字： 全局方法 **Number()**  例：Number("3.14")    // 返回 3.14

## 空字符串转换为 0。 其他的字符串会转换为 NaN (不是个数字)。

## 一元运算符 +：将变量转换为数字 例：var y = "5"; var x = + y;  // x 是一个数字

var y = "John"; var x = + y;  // x 值为NaN

## 布尔值转换为数字：全局方法 **Number()** ：Number(false) //返回0 Number(true) //返回1

## 日期转换为数字：全局方法 **Number()**  例：d = new Date(); Number(d)

日期方法 **getTime()**  例：d = new Date(); d.getTime()

## 自动转换类型

5 + null    // 返回 5         null 转换为 0  
 "5" + null  // 返回"5null"   null 转换为 "null"  
 "5" + 1     // 返回 "51"      1 转换为 "1"    
 "5" - 1     // 返回 4         "5" 转换为 5

## 自动转换为字符串：自动调用变量的 toString() 方法

document.getElementById("demo").innerHTML = myVar;  
 // if myVar = {name:"Fjohn"}  // toString 转换为 "[object Object]"  
 // if myVar = [1,2,3,4]       // toString 转换为 "1,2,3,4"  
 // if myVar = new Date()  // toString 转换为 "Fri Jul 18 2014 09:08:55 GMT

数字和布尔值也经常相互转换：

// if myVar = 123             // toString 转换为 "123"  
 // if myVar = true            // toString 转换为 "true"  
 // if myVar = false           // toString 转换为 "false"

**JavaScript 正则表达式（Regular Expression）**

语法： /正则表达式主体/修饰符(可选)

JavaScript 中，正则表达式通常用于两个字符串方法 : search() 和 replace()。

**search( ) ：**用于检索字符串中的子字符串，并返回子串的起始位置。

例：搜索 "Runoob" 字符串，且不区分大小写：（**i**  是一个**修饰符** (搜索不区分大小写））

var str = "Visit Runoob!";

var n = str.search(/Runoob/i); 结果为：6

search( ) 也可使用字符串作为参数：var n = str.search("Runoob");结果为：6

**replace( ) ：**用于在字符串中用一些字符（串）替换另一些字符（串）。

例：将字符串中的 Microsoft 替换为 Runoob :

var txt = str.replace(/microsoft/i,"Runoob");

## replace( ) 也可使用字符串： var txt = str.replace("Microsoft","Runoob");

## 修饰符****：****可以在全局搜索中不区分大小写修饰符-描述

i：执行对大小写不敏感的匹配。

g：执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。

m：执行多行匹配。

## 正则表达式模式

方括号用于查找某个范围内的字符：**描述**

（1）[abc] 查找方括号之间的任何字符。

（2）[0-9] 查找任何从 0 至 9 的数字。

（3）(x|y) 查找任何指定的选项。

元字符是拥有特殊含义的字符：**-描述**

（1）\d 查找数字。

（2）\s 查找空白字符。

（3）\b 匹配单词边界。

（4）\uxxxx 查找以十六进制数 xxxx 规定的 Unicode 字符。

量词:

（1）n+ 匹配任何包含至少一个 *n* 的字符串。

（2）n\* 匹配任何包含零个或多个 *n* 的字符串。

（3）n? 匹配任何包含零个或一个 *n* 的字符串。

**使用 RegExp 对象**

## test( )：检测一个字符串是否匹配某个模式（返回 true或false。）

例：/e/.test("The best things in life are free!")

字符串中含有 "e"，所以该实例输出为：true

## exec( )：检索字符串中的正则表达式的匹配。

该函数返回一个存放匹配结果的数组。如果未找到匹配，则返回值为 null。

例：/e/.exec("The best things in life are free!");