**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;）  
 }

**函数表达式可以存储在变量中：**var x = function (a, b) {return a \* b};  
 var z = x(4, 3);

## Function( ) 构造函数: 函数同样可以通过内置的 JavaScript 函数构造器（Function()）定义。

## 例：var myFunction = new Function("a", "b", "return a \* b"); var x = myFunction(4, 3); （x此时为12）

## 函数提升（Hoisting）：函数可以在声明之前调用

## 例：myFunction(5); function myFunction(y) { return y \* y; }

## 自调用：如果表达式后面紧跟 ( ) ，则会自动调用。

## 例：(function () {var x = "Hello!!"; })( );

## 函数是对象：****typeof****操作符判断函数类型将返回 "function"，但是函数描述为一个对象更加准确。

## JavaScript 函数有**属性**和**方法**。

## arguments.length 属性返回函数调用过程接收到的参数个数： return arguments.length;

## toString() 方法将函数整体作为一个字符串返回: return myFunction.toString();

# **默认参数：** 参数会默认设置为：undefined

y = y || 0; 如果y已经定义 ， y || 返回 y, 因为 y 是 true, 否则返回 0

## Arguments 对象： 函数内置arguments 对象，它包含了函数调用的参数数组。

## 可以通过遍历arguments数组访问所有参数。

# **函数调用：** 函数有 4 种调用方式。

**this：**在Javascript中，this指向函数执行时的当前对象。

1. **作为函数调用** function myFunction(a, b) { return a \* b; }

myFunction(10, 2); 或 window.myFunction(10, 2); //返回 20

在浏览器中的页面对象是浏览器窗口(window 对象)。以上函数会自动变为 window 对象的函数。

所以myFunction() 和 window.myFunction() 是一样的

1. **作为对象中的方法调用：**JavaScript 中你可以将函数定义为对象的方法。

var myObject = { firstName:"John", lastName: "Doe",

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

**fullName** 方法是一个函数。函数属于对象。 **myObject对象** 是函数的所有者。

return this; // 返回 [object Object] (所有者对象)

**3**. **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 函数是创建了新对象。

function myFunction(arg1, arg2) { this.firstName = arg1;

this.lastName = arg2; }

var x = new myFunction("John","Doe");

x.firstName; // 返回 "John"

构造函数的调用会创建一个新的对象。新对象会继承构造函数的属性和方法。

## 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，第一个参数必须是对象本身。

例： function myFunction(a, b) { return a \* b; }

myObject = myFunction.call(myObject, 10, 2); // 返回 20

myArray = [10, 2];

myObject = myFunction.apply(myObject, myArray); // 返回 20

两个方法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：call直接按顺序传入参数， apply传入的是二个参数数组。

在 严格模式(strict mode)下, 在调用函数时第一个参数会成为 **this** 的值， 即使该参数不是一个对象。

在 非严格模式(non-strict mode)下, 如果第一个参数的值是 null 或 undefined, 它将使用全局对象替代。

通过 call() 或 apply() 方法你可以设置 this 的值, 且作为已存在对象的新方法调用。

## 

# **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 内嵌函数：** 嵌套函数可以访问上一层的函数变量。

例：内嵌函数 **plus( )** 可以访问父函数（add）的 **counter**变量

function add() { var counter = 0;  
    function plus() {counter += 1;}  
    plus();

}

**闭包**：可以在外部访问 **plus( )** 函数

var add = (function () { var counter = 0;  
    return function () {return counter += 1;} } )( );  
  
 add(); add(); add(); // counter值为 3

变量 **add** 指定了函数自我调用的返回值（为一个函数）。

自我调用函数只执行一次。设置计数器为 0。并返回函数表达式。

add变量成为一个函数。它可以访问函数上一层作用域的计数器。

这个叫作 JavaScript **闭包。**它使得函数拥有私有变量变成可能。

计数器受匿名函数的作用域保护，只能通过 add 方法修改。

# **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：

（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：

（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!");

**JavaScript 错误**

**try**  测试代码块的错误。

**catch**  处理错误。

JavaScript 语句 **try** 和 **catch** 是成对出现的。

语法：try {

//在这里运行代码

} catch(err) {

//在这里处理错误

}

**throw**  创建自定义错误

语法：throw exception

例：try { var x=document.getElementById("demo").value;

if(x < 5) throw "太小"; }

catch(err) { var y=document.getElementById("mess");

y.innerHTML="错误：" + err + "。"; } }

如果输入2，则抛出错误：太小。

**JavaScript 调试**

很多浏览器都内置了调试工具，一般是按下 F12 键，并在调试菜单中选择 "Console" 。

## console.log( ) ：在调试窗口上打印 JavaScript 值。 例：console.log(10);

**debugger关键字** ：用于停止执行 JavaScript，并调用调试函数。

例：<script> var x = 15 \* 5;

debugger;

document.getElementById("demo").innerHTML = x;

</script> （开启 debugger ，代码在第三行前停止执行。）

**JavaScript 变量提升（hoisting）**

JavaScript 中，函数及变量的声明都将被提升到函数的最顶部。

JavaScript 中，变量可以在使用后声明，也就是变量可以先使用再声明。

## 初始化不会提升：只有变量声明（var x）会被提升，初始化（x=5）不会。

例： var x ; //声明 x

var x = 5; //初始化 x

**JavaScript 严格模式 (strict mode)：**严格模式下你不能使用未声明的变量

## **"use strict" 指令：**指定代码在严格条件下执行。

例："use strict";  
 x = 3.14;      //**调试模式会报错** (x 未定义)

在函数内部声明"use strict"; 表示局部作用域 (只在函数内使用严格模式)

# **JavaScript 表单**: **HTML 表单验证可以通过 JavaScript 来完成。**

# **获取html表单中的值:**

# 第一种：<input type="text" value="123" name="name1"/> document.forms[表单name属性值][name1].value 第二种：<input type="text" value="123" id="id1"/> document.getElementById("id1").value

**表单验证**

# 必填（或必选）项目: 如果为空 需要返回false，表示数据有问题。

# var x=document.forms["myForm"]["fname"].value;

# if (x==null || x=="") { alert("姓必须填写"); return false; }

## E-mail 验证: 输入的数据必须包含 @ 符号和点号(.)，同时，@ 不可以是邮件地址的首字符，并且 @ 之后需有至少一个点号。

var x=document.forms["myForm"]["email"].value;

var atpos=x.indexOf("@");

var dotpos=x.lastIndexOf(".");

if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)

{ alert("不是一个有效的 e-mail 地址"); return false; }

**JavaScript 验证 API：**

# **约束验证 DOM 方法：**

checkValidity( )：如果 input 元素中的数据是合法的返回 true，否则返回 false。

setCustomValidity( )：设置 input 元素的 validationMessage 属性，用于自定义错误提示信息 的方法。

# **约束验证 DOM 属性：**

validity：布尔属性值，返回 input 输入值是否合法

validationMessage：浏览器错误提示信息

willValidate：指定 input 是否需要验证

例：<input id="id1" type="number" min="100" max="300" required>

var inpObj = document.getElementById("id1");

if (inpObj.checkValidity() == false) {

document.getElementById("demo").innerHTML = inpObj.validationMessage; }

**validity：**包含一系列关于数据的属性:

rangeOverflow ：为 true, 如果元素的值大于设置的最大值。

rangeUnderflow ：为true, 如果元素的值小于它的最小值。

tooLong ：为 true, 如果元素的值超过了 maxLength 属性设置的长度。

valid ：为 true，如果元素的值是合法的。

例：var inpObj = document.getElementById("id1");

if (inpObj.validity.rangeUnderflow) { txt = "输入的值太小了";}

**JavaScript JSON （** **JavaScript Object Notation）**

JSON 是用于存储和传输数据的格式。通常用于服务端向网页传递数据 。

JSON 是一种轻量级的数据交换格式。

JSON是独立的语言 ：使用 JavaScript 语法，但是 JSON 格式仅仅是一个文本。  
 文本可以被任何编程语言读取及作为数据格式传递。

**JSON 语法规则**：1.数据为 键:值 对。 例："name":"Runoob"

2.数据由逗号分隔。

3.大括号保存对象。 例：{"name":"Runoob", "url":"www.runoob.com"}

4.方括号保存数组。 例：对象 "sites" 是一个数组，包含了三个对象。

"sites":[{"name": "Runoob", "url":"www.runoob.com"},

{"name": "Google", "url":"www.google.com"},

{"name": "Taobao", "url":"www.taobao.com"}]

## JSON 字符串转换为 JavaScript 对象

通常我们从服务器中读取 JSON 数据，并在网页中显示数据。

1. 简单起见创建 JavaScript 字符串，字符串为 JSON 格式的数据：

var text = '{ "sites" : [' +

'{ "name":"Runoob" , "url":"www.runoob.com" },' +

'{ "name":"Google" , "url":"www.google.com" },' +

'{ "name":"Taobao" , "url":"www.taobao.com" } ]}';

1. 使用 JavaScript 内置函数 JSON.parse() 将字符串转换为 JavaScript 对象

var obj = JSON.parse(text);

1. 在页面中使用新的 JavaScript 对象

例：document.write = obj.sites[1].name + " " + obj.sites[1].url;

结果：Google [www.google.com](http://www.google.com)

## 相关函数

[JSON.parse( )](http://www.runoob.com/js/javascript-json-parse.html)：用于将 JSON 字符串转换为 JavaScript 对象。

[JSON.stringify( )](http://www.runoob.com/js/javascript-json-stringify.html)：用于将 JavaScript 值转换为 JSON 字符串。

**Javascript:void(0)**

javascript:void(0) 中最关键的是 void 关键字，该操作符要计算括号内表达式但是不返回值。

语法格式如下：<head>

<script type="text/javascript">

void func( ) 或者：void(func( ))

javascript:void func( ) javascript:void(func( ))

</script>

</head>

例：创建了一个超级链接，当用户点击以后不会发生任何事：

<a href="javascript:void(0)">单击此处什么也不会发生</a>

在用户点击链接后显示警告信息：

<a href="javascript:void(alert('Warning!!!'))">点我!</a>

参数 a 将返回 undefined （结果：a = undefined b = 5 c = 7）:

function getValue(){  
   var a,b,c;  
   a = void ( b = 5, c = 7 );  
   document.write('a = ' + a + ' b = ' + b +' c = ' + c ); }

## href="#"与href="javascript:void(0)"的区别

**#** 包含了一个位置信息，默认的锚是**#top** 也就是网页的上端。而javascript:void(0), 仅仅表示

一个死链接。在页面很长的时候会使用 **#** 来定位页面的具体位置，格式为：**# + id**。如果要定

义一个死链接请使用 javascript:void(0) 。

例：<a href="#pos">点我定位到指定位置!</a>

<p id="pos">尾部定位点</p>

## HTML 载入外部 JavaScript 文件

<script src="myscript.js">

## 使用 JavaScript 访问 HTML 元素

var obj = getElementById("Demo")