JS学习

边看边敲，菜鸟教程，不要眼高手低

1. 初步理解DOM和BOM
2. 自己使用JS获取DOM，操控DOM树
3. 能自己写一个网页端的四则运算

## JavaScript：改变 HTML 内容

使用 JavaScript 来处理 HTML 内容是非常强大的功能。

## 实例

x=document.getElementById("demo"); //查找元素 x.innerHTML="Hello JavaScript"; //改变内容

**[尝试一下 »](https://www.runoob.com/try/try.php?filename=tryjs_intro_inner_html" \t "https://www.runoob.com/js/_blank)**

您会经常看到 **document.getElementById("some id")**。这个方法是 HTML DOM 中定义的。

DOM (**D**ocument **O**bject **M**odel)（文档对象模型）是用于访问 HTML 元素的正式 W3C 标准。

您将在本教程的多个章节中学到有关 HTML DOM 的知识。

<body>

<script>

  function changeImage(){

    element=document.getElementById('myimage')

    if （element.src.march("bulbon")）{

      element.src="/images/pic-bulboff.gif";

    }

    else

    {

      element.src="/images/pic-bulbon.gif";

    }

  }

</script>

  <img id="myimage" onclick="changeImage()" src="/images/pic-bulboff.gif"width="100" height="80">

<p>点击图片切换图片</p>

</body>

JS DOM

# JavaScript HTML DOM - 改变 HTML

HTML DOM 允许 JavaScript 改变 HTML 元素的内容。

## 改变 HTML 输出流

JavaScript 能够创建动态的 HTML 内容：实时时间

<script>  
document.write(Date());  
</script>

## 改变 HTML 内容

<p id="p1">Hello World!</p>  
<script>  
document.getElementById("p1").innerHTML="新文本!";  
</script>

## 改变 CSS 样式

如需改变 CSS 元素的样式，请使用这个语法：

document.getElementById(*id*).style.*property*=*新样式*

# JavaScript HTML DOM 事件

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

本例从事件处理器调用一个函数：

<script>  
function changetext(id)  
{  
    id.innerHTML="Ooops!";  
}  
</script>  
</head>  
<body>  
<h1 onclick="changetext(this)">点击文本!</h1>  
</body>

## onload 和 onunload 事件

onload 和 onunload 事件会在用户进入或离开页面时被触发。

onload 事件可用于检测访问者的浏览器类型和浏览器版本，并基于这些信息来加载网页的正确版本。

onload 和 onunload 事件可用于处理 cookie。

## onchange 事件

onchange 事件常结合对输入字段的验证来使用。

下面是一个如何使用 onchange 的例子。当用户改变输入字段的内容时，会调用 upperCase() 函数。

## 实例

<input type="text" id="fname" **onchange**="upperCase()">

## onmouseover 和 onmouseout 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。

<div onmouseover="mOver(this)" onmouseout="mOut(this)" style="background-color:#D94A38;width:120px;height:20px;padding:40px;">Mouse Over Me</div>

<script>

function mOver(obj){

obj.innerHTML="Thank You"

}

function mOut(obj){

obj.innerHTML="Mouse Over Me"

## onmousedown、onmouseup 以及 onclick 事件

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。

## addEventListener() 方法

在用户点击按钮时触发监听事件：

document.getElementById("myBtn").addEventListener("click", displayDate);

addEventListener() 方法添加的事件句柄不会覆盖已存在的事件句柄。

你可以向一个元素添加多个事件句柄。

你可以向同个元素添加多个同类型的事件句柄，如：两个 "click" 事件。

你可以向任何 DOM 对象添加事件监听，不仅仅是 HTML 元素。如： window 对象。

你可以使用 removeEventListener() 方法来移除事件的监听。

button.removeEventListener("click", handleClick);

在此示例中，handleClick 函数最初添加为按钮上 “click” 事件的事件侦听器。然后使用 removeEventListener 方法删除此事件侦听器，确保对按钮的后续单击不会触发 handleClick 函数[2](https://www.w3schools.com/jsref/met_element_removeeventlistener.asp" \t "https://cn.bing.com/_blank).

## 语法

element.addEventListener(event, function, useCapture);

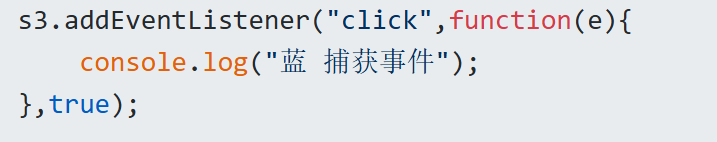
第一个参数是事件的类型 (如 "click" 或 "mousedown").///////例子为event////

第二个参数是事件触发后调用的函数。///例子中为function///

## **事件捕获与事件冒泡同时存在-先捕获后冒泡**

## 第三个参数是个布尔值用于描述事件是冒泡(事件会从最内层的元素开始发生)还是捕获(事件会从最外层开始发生)///例子中为useCapture///

addEventListener第三个参数默认值是false，表示在事件冒泡阶段调用事件处理函数;如果参数为true，则表示在事件捕获阶段调用处理函数。



## 向原元素添加事件句柄

内置函数

element.addEventListener("click", function(){ alert("Hello World!"); });

链接外部函数

element.addEventListener("click", myFunction);  
function myFunction() {  
    alert ("Hello World!");  
}

## 向 Window 对象添加事件句柄

当用户重置窗口大小时添加事件监听：

window.addEventListener("resize", function(){  
    document.getElementById("demo").innerHTML = sometext;  
});

## 传递参数

<button id="myBtn">点我</button>

<p id="demo"></p>

<script>

var p1 = 7;

var p2 = 99;

document.getElementById("myBtn").addEventListener("click",function(){

myFunction(p1,p2);

});

function myFunction(p1,p2){

var result=p1\*p2;

document.getElementById("demo").innerHTML=result;}

</script>

## 创建新的 HTML 元素 (节点) - appendChild()

## 添加新元素到尾部

以下代码是用于创建 <p> 元素:

var para = document.createElement("p");

为 <p> 元素创建一个新的文本节点：

var node = document.createTextNode("这是一个新的段落。");

将文本节点添加到 <p> 元素中：

para.appendChild(node);

最后，在一个已存在的元素中添加 p 元素。

查找已存在的元素：

var element = document.getElementById("div1");

添加到已存在的元素中:

element.appendChild(para);

## 移除已存在的元素

要移除一个元素，你需要知道该元素的父元素。

<div id="div1">

<p id="p1">这是一个段落。</p>

<p id="p2">这是另外一个段落。</p>

</div>

<script>

var parent = document.getElementById("div1");

Var child=document.getElementById("p1"); parent.removeChild(child);

## 替换 HTML 元素 - replaceChild()

parent.replaceChild(para, child);

## HTMLCollection 对象()

<script>

var myCollection = document.getElementsByTagName("p");

document.getElementById("demo").innerHTML = "第二个段落的内容为:<span style='color:red;'> " + myCollection[1].innerHTML + '</span>';

**修改所有 <p> 元素的背景颜色:**

<p>点击按钮修改 p 元素的背景颜色。</p>

<button onclick="myFunction()">点我</button>

<script>

function myFunction() {

var myCollection = document.getElementsByTagName("p");

var i;

for (i = 0; i < myCollection.length; i++) {

myCollection[i].style.color = "red";

}

}

获取 <p> 元素的集合：var myCollection = document.getElementsByTagName("p");

显示集合元素个数：document.getElementById("demo").innerHTML = myCollection.length;

集合 length 属性常用于遍历集合中的元素。

## Window 尺寸

有三种方法能够确定浏览器窗口的尺寸。

实用的 JavaScript 方案（涵盖所有浏览器，包含 IE8 及以下版本的浏览器）：

var w=window.innerWidth || document.documentElement.clientWidth || document.body.clientWidth; var h=window.innerHeight || document.documentElement.clientHeight || document.body.clientHeight;

全局变量不能通过 delete 操作符删除；而 window 属性上定义的变量可以通过 delete 删除

## Window Screen

**window.screen**对象在编写时可以不使用 window 这个前缀。

一些属性：

* screen.availWidth - 可用的屏幕宽度
* screen.availHeight - 可用的屏幕高度

## Window Screen 可用宽度

## Window Screen 可用高度

1.screen.availHeight 属性返回访问者屏幕的高度

2.screen.availWidth 属性返回访问者屏幕的宽度，以像素计，减去界面特性，比如窗口任务栏。

<script>  
document.write("可用宽度: " + screen.availWidth);  
</script>

## Window Location

**window.location** 对象在编写时可不使用 window 这个前缀。 一些例子：

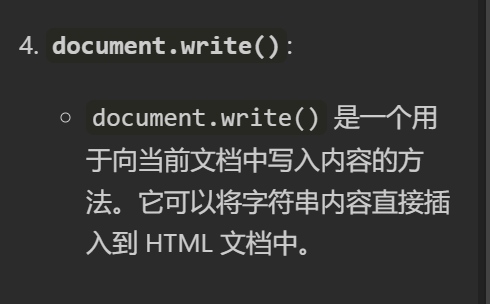
一些实例:

* location.hostname 返回 web 主机的域名
* location.pathname 返回当前页面的路径和文件名
* location.port 返回 web 主机的端口 （80 或 443）
* location.protocol 返回所使用的 web 协议（http: 或 https:）

## Window Location href

**location.href** 属性返回当前页面的 URL。

<script>  
  
document.write(location.href);  
  
</script>



## Window Location assign

**location.assign()** 方法加载新的文档。

window.location.assign(url) ： 加载 URL 指定的新的 HTML 文档。 就相当于一个链接，跳转到指定的url，当前页面会转为新页面内容，可以点击后退返回上一个页面。

window.location.replace(url) ： 通过加载 URL 指定的文档来替换当前文档 ，这个方法是替换当前窗口页面，前后两个页面共用一个窗口，所以是没有后退返回上一页的

<script>

Function newDoc(){

window.location.assign("https://www.runoob.com") } </script>

</head> <body>

<input type="button" value="加载新文档" onclick="newDoc()"> </body>

## Window History

**window.history**对象在编写时可不使用 window 这个前缀。

为了保护用户隐私，对 JavaScript 访问该对象的方法做出了限制。

一些方法：

* history.back() - 与在浏览器点击后退按钮相同
* history.forward() - 与在浏览器中点击向前按钮相同

function goBack() {

window.history.back() } </script>

</head> <body>

<input type="button" value="Back" onclick="goBack()">

除此之外可以用 **history.go()** 这个方法来实现向前，后退的功能。

function a(){

history.go(1); // go() 里面的参数表示跳转页面的个数 例如 history.go(1) 表示前进一个页面}function b(){

history.go(-1); // go() 里面的参数表示跳转页面的个数 例如 history.go(-1) 表示后退一个页面}

history.go(0); // go() 里面的参数为0,表示刷新页面

## Window Navigator

**window.navigator** 对象在编写时可不使用 window 这个前缀。

## 实例

<script> txt = "<p>浏览器代号: " + navigator.appCodeName + "</p>";

txt+= "<p>浏览器名称: " + navigator.appName + "</p>";

txt+= "<p>浏览器版本: " + navigator.appVersion + "</p>";

txt+= "<p>启用Cookies: " + navigator.cookieEnabled + "</p>";

txt+= "<p>硬件平台: " + navigator.platform + "</p>";

txt+= "<p>用户代理: " + navigator.userAgent + "</p>";

txt+= "<p>用户代理语言: " + navigator.language + "</p>";

document.getElementById("example").innerHTML=txt; </script>

来自 navigator 对象的信息具有误导性，不应该被用于检测浏览器版本，这是因为：

* navigator 数据可被浏览器使用者更改
* 一些浏览器对测试站点会识别错误
* 浏览器无法报告晚于浏览器发布的新操作系统

## 确认框

确认框通常用于验证是否接受用户操作。

当确认框弹出时，用户可以点击 "确认" 或者 "取消" 来确定用户操作。

当你点击 "确认", 确认框返回 true， 如果点击 "取消", 确认框返回 false。

### **语法**

window.confirm("*sometext*");

**window.confirm()** 方法可以不带上window对象，直接使用**confirm()**方法。

<p>点击按钮，显示确认框。</p><button onclick="myFunction()">点我</button>

<p id="demo"></p><script>

function myFunction(){

var x;

var r=confirm("按下按钮!");

if (r==true){ x="你按下了\"确定\"按钮!"; }

else{ x="你按下了\"取消\"按钮!";

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

}</script>

## JavaScript 计时事件

JavaScript 中使用计时事件是很容易的，两个关键方法是:

* setInterval() - 间隔指定的毫秒数不停地执行指定的代码。
* setTimeout() - 在指定的毫秒数后执行指定代码。

setInterval() 方法设置每秒钟执行一次代码，就是手表一样。

var myVar=setInterval(function(){myTimer()},1000);

function myTimer() {

var d=new Date();

var t=d.toLocaleTimeString();

document.getElementById("demo").innerHTML=t; }

learInterval() 方法用于停止 setInterval() 方法执行的函数代码。

window.clearInterval(intervalVariable)

要使用 clearInterval() 方法, 在创建计时方法时你必须使用全局变量：

myVar=setInterval("*javascript function*",*milliseconds*);

然后你可以使用 clearInterval() 方法来停止执行。

## setTimeout() 方法

setTimeout(function(){alert("Hello")},3000);

## 如何停止执行?

clearTimeout() 方法用于停止执行setTimeout()方法的函数代码。（同样需要全局变量）

# JavaScript Cookie

## 使用 JavaScript 创建Cookie

JavaScript 可以使用 **document.cookie** 属性来创建 、读取、及删除 cookie。

document.cookie="username=John Doe; expires=Thu, 18 Dec 2043 12:00:00 GMT";

## 使用 JavaScript 读取 Cookie

在 JavaScript 中, 可以使用以下代码来读取 cookie：

var x = document.cookie;

如果您设置了新的 cookie，旧的 cookie 不会被覆盖。 新 cookie 将添加到 document.cookie 中，所以如果您重新读取document.cookie，您将获得如下所示的数据：

cookie1=value; cookie2=value;

## 设置 cookie 值的函数

首先，我们创建一个函数用于存储访问者的名字：

function setCookie(cname,cvalue,exdays) { var d = new Date(); d.setTime(d.getTime()+(exdays\*24\*60\*60\*1000)); var expires = "expires="+d.toGMTString(); document.cookie = cname + "=" + cvalue + "; " + expires; }

## 获取 cookie 值的函数

然后，我们创建一个函数用于返回指定 cookie 的值：

function getCookie(cname) { var name = cname + "="; var ca = document.cookie.split(';'); for(var i=0; i<ca.length; i++) { var c = ca[i].trim(); if (c.indexOf(name)==0) return c.substring(name.length,c.length); } return ""; }

cookie 名的参数为 cname。

创建一个文本变量用于检索指定 cookie :cname + "="。

使用分号来分割 document.cookie 字符串，并将分割后的字符串数组赋值给 ca (ca = document.cookie.split(';'))。

循环 ca 数组 (i=0;i<ca.length;i++)，然后读取数组中的每个值，并去除前后空格 (c=ca[i].trim())。

如果找到 cookie(c.indexOf(name) == 0)，返回 cookie 的值 (c.substring(name.length,c.length)。

如果没有找到 cookie, 返回 ""。

## 检测 cookie 值的函数

最后，我们可以创建一个检测 cookie 是否创建的函数。

如果设置了 cookie，将显示一个问候信息。

如果没有设置 cookie，将会显示一个弹窗用于询问访问者的名字，并调用 setCookie 函数将访问者的名字存储 365 天：

function checkCookie() { var username=getCookie("username"); if (username!="") { alert("Welcome again " + username); } else { username = prompt("Please enter your name:",""); if (username!="" && username!=null) { setCookie("username",username,365); } } }

const 声明一个只读的常量，一旦声明，常量的值就不能改变。

let 声明的变量只在 let 命令所在的代码块内有效。

let 声明的变量只在 let 命令所在的代码块 **{}** 内有效，在 **{}** 之外不能访问。

{

let x = 2;}// 这里不能使用 x 变量

## Const 关键字

const 用于声明一个或多个常量，声明时必须进行初始化，且初始化后值不可再修改：

## 实例

const PI = 3.141592653589793; PI = 3.14; // 报错 PI = PI + 10; // 报错

const定义常量与使用let 定义的变量相似：

* 二者都是块级作用域
* 都不能和它所在作用域内的其他变量或函数拥有相同的名称

两者还有以下两点区别：

* const声明的常量必须初始化，而let声明的变量不用
* const 定义常量的值不能通过再赋值修改，也不能再次声明。而 let 定义的变量值可以修改。

const 定义的对象或者数组，其实是可变的。下面的代码并不会报错：

## 实例

// 创建常量对象 const car = {type:"Fiat", model:"500", color:"white"}; // 修改属性: car.color = "red"; // 添加属性 car.owner = "Johnson";

paeseFloat(获取值转化为浮点数)

 const num2 = parseFloat(document.getElementById('num2').value);

Placeholder(文本框内引导字)

<input type="number" id="num2" placeholder="输入第二个数字">

