HTML DOM 笔记

|  |  |  |
| --- | --- | --- |
| 修改时间 | 备注 | 修改人 |
| 2019/9/11 15:26 | Dom对象 | 杨庆文 |
| 2019/9/11 16:02 | 浏览器的工作原理 | 陈艺宇 |
| 2019/9/11 16:15 | dom操作(节点查找, 编辑） | 骆明睿 |
| 2019/9/11 16:28 | Dom/ jq 事件注册/解绑 | 袁鑫 |
| 2019/9/11 16:31 | dom操作（新增 , 删除(坑)) | 董青林 |
| 2019/9/11 17:05 | Event对象 / | 韦清源 |
| 2019/9/11 17:43 | window对象 window的事件 | 孙蒙 |

目录

[HTML DOM 笔记 1](#_Toc19172911)

[1 HTML DOM 教程 4](#_Toc19172912)

[1.1 HTML DOM 树形结构: 4](#_Toc19172913)

[1.2定义 5](#_Toc19172914)

[2.Document 对象 5](#_Toc19172915)

[2.1 Document 对象属性和方法: 6](#_Toc19172916)

[3. DOM操作 6](#_Toc19172917)

[3.1文献参考 6](#_Toc19172918)

[3.2新增 6](#_Toc19172919)

[3.3删除 7](#_Toc19172920)

[3.3.1 removeChild():从子节点列表中删除某个节点。 7](#_Toc19172921)

[3.3.2 removeChild()删除数组元素时的注 7](#_Toc19172922)

[3.4查找 HTML 元素 8](#_Toc19172923)

[3.41常用方法 8](#_Toc19172924)

[3.4.2 .通过 id 查找 HTML 元素 8](#_Toc19172925)

[3.4.3 通过标签名查找 HTML 元素 9](#_Toc19172926)

[3.5通过类名找到 HTML 元素 9](#_Toc19172927)

[3.6替换 HTML 元素 10](#_Toc19172928)

[3.7改变 HTML 输出流 10](#_Toc19172929)

[3.8改变 HTML 内容 11](#_Toc19172930)

[3.9改变 HTML 属性 11](#_Toc19172931)

[3.10改变 CSS 12](#_Toc19172932)

[4 event对象 （事件源） 12](#_Toc19172933)

[https://www.runoob.com/jquery/jquery-ref-events.html 12](#_Toc19172934)

[4.1 事件源 -- e 表示 Event对象(事件对象) 12](#_Toc19172935)

[4.2 以参数注入的方式 13](#_Toc19172936)

[5 dom事件 15](#_Toc19172937)

[参考文献 15](#_Toc19172938)

[拖放参考：https://www.runoob.com/html/html5-draganddrop.html 15](#_Toc19172939)

[https://blog.csdn.net/ssisse/article/details/52628739 15](#_Toc19172940)

[5.1 事件的绑定和解绑 15](#_Toc19172941)

[原生dom的绑定和解绑 15](#_Toc19172942)

[JQ 事件绑定： 16](#_Toc19172943)

[鼠标事件 16](#_Toc19172944)

[拖拽事件 18](#_Toc19172945)

[参考：https://www.runoob.com/html/html5-draganddrop.html 18](#_Toc19172946)

[https://blog.csdn.net/ssisse/article/details/52628739 18](#_Toc19172947)

[draggable="true"：设置元素为可拖放 18](#_Toc19172948)

[键盘事件 18](#_Toc19172949)

[浏览器事件 19](#_Toc19172950)

[表单事件 20](#_Toc19172951)

[6 . window对象 window的事件 20](#_Toc19172952)

[1.1 浏览器整个窗口与框架 20](#_Toc19172953)

[1.2 Window对象的属性 20](#_Toc19172954)

[1.3 Window对象的方法 21](#_Toc19172955)

[1.4 Screen对象说明 22](#_Toc19172956)

[1.5 History对象说明 22](#_Toc19172957)

[1.6 Location对象说明 22](#_Toc19172958)

[1.7 练习代码 22](#_Toc19172959)

[7 . 浏览器对象模型(bom) 的运行原理和过程 23](#_Toc19172960)

[1. 浏览器的组成 -- 两个引擎 23](#_Toc19172961)

[JavaScript 引擎 -- 主要作用是读取网页中的 JavaScript 代码，对其处理后运行。 24](#_Toc19172962)

[2. 工作原理 24](#_Toc19172963)

[3. 阻塞效应 24](#_Toc19172964)

[4. 脚本的动态加载 25](#_Toc19172965)

[5. 重绘 重流 25](#_Toc19172966)

目录:

[HTML DOM 笔记 4](#_Toc31289)

[1 HTML DOM 教程 4](#_Toc3089)

[1.1 HTML DOM 树形结构: 4](#_Toc516)

[1.2定义 5](#_Toc10855)

[2.Document 对象 5](#_Toc11381)

[2.1 Document 对象属性和方法: 6](#_Toc9943)

[3. DOM操作 6](#_Toc8064)

[3.1文献参考 6](#_Toc461)

[3.2新增 6](#_Toc11004)

[3.3删除 7](#_Toc24667)

[3.3.1 removeChild():从子节点列表中删除某个节点。 7](#_Toc2538)

[3.3.2 removeChild()删除数组元素时的注 7](#_Toc23276)

[3.4查找 HTML 元素 8](#_Toc16070)

[3.41常用方法 8](#_Toc3778)

[3.4.2 .通过 id 查找 HTML 元素 8](#_Toc27555)

[3.4.3 通过标签名查找 HTML 元素 9](#_Toc28723)

[3.5通过类名找到 HTML 元素 9](#_Toc19422)

[3.6替换 HTML 元素 10](#_Toc3070)

[3.7改变 HTML 输出流 10](#_Toc25768)

[3.8改变 HTML 内容 11](#_Toc6202)

[3.9改变 HTML 属性 11](#_Toc27330)

[3.10改变 CSS 12](#_Toc31135)

[4 event对象 （事件源） 12](#_Toc3861)

[https://www.runoob.com/jquery/jquery-ref-events.html 12](#_Toc18496)

[4.1 事件源 -- e 表示 Event对象(事件对象) 12](#_Toc12557)

[4.2 以参数注入的方式 13](#_Toc30444)

[5 dom事件 15](#_Toc11496)

[参考文献 15](#_Toc6871)

[拖放参考：https://www.runoob.com/html/html5-draganddrop.html 15](#_Toc27987)

[https://blog.csdn.net/ssisse/article/details/52628739 15](#_Toc32051)

[5.1 事件的绑定和解绑 15](#_Toc25352)

[原生dom的绑定和解绑 15](#_Toc11998)

[JQ 事件绑定： 16](#_Toc23117)

[鼠标事件 16](#_Toc22109)

[拖拽事件 18](#_Toc9606)

[参考：https://www.runoob.com/html/html5-draganddrop.html 18](#_Toc29360)

[https://blog.csdn.net/ssisse/article/details/52628739 18](#_Toc3382)

[draggable="true"：设置元素为可拖放 18](#_Toc25029)

[键盘事件 18](#_Toc14626)

[浏览器事件 18](#_Toc12187)

[表单事件 20](#_Toc9076)

[6 . window对象 window的事件 20](#_Toc6987)

[1.1 浏览器整个窗口与框架 20](#_Toc28177)

[1.2 Window对象的属性 20](#_Toc27028)

[1.3 Window对象的方法 21](#_Toc29563)

[1.4 Screen对象说明 22](#_Toc6308)

[1.5 History对象说明 22](#_Toc30717)

[1.6 Location对象说明 22](#_Toc26408)

[1.7 练习代码 22](#_Toc17647)

[7 . 浏览器对象模型(bom) 的运行原理和过程 23](#_Toc28916)

[1. 浏览器的组成 -- 两个引擎 23](#_Toc1870)

[JavaScript 引擎 -- 主要作用是读取网页中的 JavaScript 代码，对其处理后运行。 24](#_Toc30134)

[2. 工作原理 24](#_Toc21512)

[3. 阻塞效应 24](#_Toc17054)

[4. 脚本的动态加载 25](#_Toc1960)

[5. 重绘 重流 25](#_Toc15836)

参考网站:<https://www.runoob.com/htmldom/htmldom-tutorial.html>

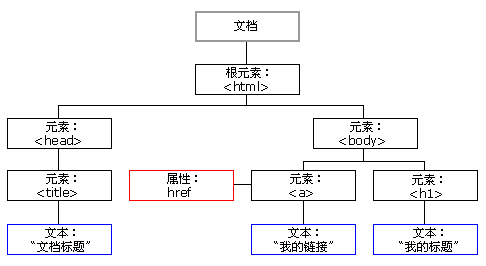
# 1 HTML DOM 教程

DOM (Document Object Model) 译为文档对象模型，是 HTML 和 XML 文档的编程接口。HTML DOM 定义了访问和操作 HTML 文档的标准方法。DOM 以树结构表达 HTML 文档。

## 1.1 HTML DOM 树形结构:

通过 HTML DOM，JavaScript 能够访问和改变 HTML 文档的所有元素。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 模型被结构化为对象树：



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。

1. JavaScript 能够改变页面中的所有 HTML 元素。
2. JavaScript 能够改变页面中的所有 HTML 属性。
3. JavaScript 能够改变页面中的所有 CSS 样式。
4. JavaScript 能够对页面中的所有事件做出反应。

html 为根节点 没有父节点 最大的是浏览器

html标签皆 节点

## 1.2定义

DOM(文档对象模型)结构是一个树型结构，当一个HTML元素产生一个事件时，该事件会在元素节点与根结点之间的路径传播，路径所经过的结点都会收到该事件，这个传播过程可称为DOM事件流。

DOM同时支持两种事件模型：[捕获型事件](https://baike.baidu.com/item/%E6%8D%95%E8%8E%B7%E5%9E%8B%E4%BA%8B%E4%BB%B6)和[冒泡型事件](https://baike.baidu.com/item/%E5%86%92%E6%B3%A1%E5%9E%8B%E4%BA%8B%E4%BB%B6/7508003)，但是，捕获型事件先发生。两种事件流会触发DOM中的所有对象，从[document对象](https://baike.baidu.com/item/document%E5%AF%B9%E8%B1%A1/6553996)开始，也在document对象结束。

DOM事件模型的最独特的性质是，文本节点也触发事件（在IE中不会）

# 2.Document 对象

当浏览器载入 HTML 文档, 它就会成为 Document 对象。Document 对象是 HTML 文档的根节点。Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

提示：Document 对象是 Window 对象的一部分，可通过 window.document 属性对其进行访问。

## 2.1 Document 对象属性和方法:

这个参考:<https://www.runoob.com/jsref/dom-obj-document.html>

// document.images[0].src=""; // 通过doucument检索标签(节点)

// 用法: document.getElementById("id").要用的东西

# 3. DOM操作

## 3.1文献参考

**1.1.1** <https://www.runoob.com/htmldom/htmldom-elements.html>

**1.1.2** <https://www.runoob.com/jsref/met-node-insertbefore.html>

## 3.2新增

**3.2.1 appendChild():**向节点的子节点列表的末尾添加新的子节点。(这个函数，得先去找到父元素).

<script>

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

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

para.appendChild(node);// para是node的父元素

</script>

**3.2.2 insertBefore():**在已有的子节点前插入一个新的子节点。

<div id="div1">

<p id="p1">这是一个段落。</p>

</div>

<script>

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

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

para.appendChild(node);

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

var child=document.getElementById("p1");

element.insertBefore(para,child);

</script>

**3.2.3 createAttribute():**创建一个指定名称的属性，并返回Attr 对象属性。

// 创建 class 属性, 属性值为 "democlass", 并将clas属性添加到 H1 元素中

var att=document.createAttribute("class");

att.value="democlass";

document.getElementsByTagName("H1")[0].setAttributeNode(att);

**3.2.4 createElement():**创建元素节点。

var para=document.createElement("p");// p可以替换成其他标签名

**3.2.5 createTextNode():**创建文本节点。

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

## 3.3删除

### 3.3.1 removeChild():从子节点列表中删除某个节点。

var list=document.getElementById("myList");

list.removeChild(list.childNodes[0]);

3.3.2 removeChild()删除数组元素时的注**意事项:**

⑴错误代码示范: removeChild()方法删除数组元素时直接使用for循环会导致数组长度发生变化,从而无法将数组元素全部删除.

var allStudents = [

{

name: "姓名",

id: 03

},

];

var allguys = document.getElementsByClassName("student");

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

allguys[i].remove();

}

⑵正确代码示范:

var allStudents = [

{

name: "姓名",

id: 03

},

];

// 方法一:

var allguys = document.getElementsByClassName("student");

while (allguys.length > 0) {

allguys[0].remove();

}

// 方法二:

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

var targetHtmlNode=document.getElementById(allStudents[i].id);

if (targetHtmlNode) {

targetHtmlNode.remove();

}

}

## 3.4查找 HTML 元素

3.41常用方法 :

getElementById() 返回带有指定 ID 的元素。

getElementsByTagName() 返回包含带有指定标签名称的所有元素的节点列表（集合/节点数组）。

getElementsByClassName() 返回包含带有指定类名的所有元素的节点列表。

appendChild() 把新的子节点添加到指定节点。

removeChild() 删除子节点。

replaceChild() 替换子节点。

insertBefore() 在指定的子节点前面插入新的子节点。

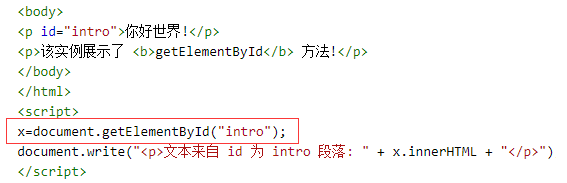
createAttribute() 创建属性节点。

createElement() 创建元素节点。

createTextNode() 创建文本节点。

### 3.4.2 .通过 id 查找 HTML 元素

本例查找 id="intro" 元素：

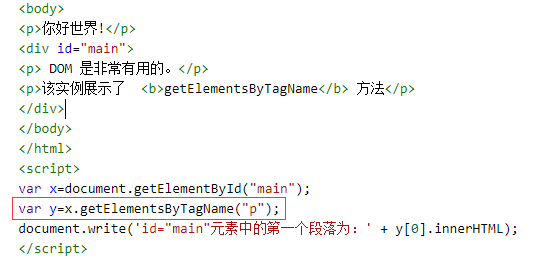


如果找到该元素，则该方法将以对象（在 x 中）的形式返回该元素。

如果未找到该元素，则 x 将包含 null。

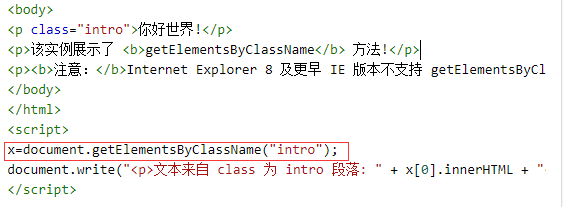
### 3.4.3 通过标签名查找 HTML 元素

本例查找 id="main" 的元素，然后查找 id="main" 元素中的所有 <p> 元素：



## 3.5通过类名找到 HTML 元素

本例通过 [getElementsByClassName](https://www.runoob.com/jsref/met-document-getelementsbyclassname.html) 函数来查找 class="intro" 的元素：



## 3.6替换 HTML 元素

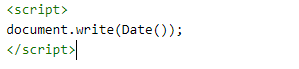
如需替换元素的，请使用 replaceChild() 方法：



## 3.7改变 HTML 输出流

JavaScript 能够创建动态的 HTML 内容：

在 JavaScript 中，document.write() 可用于直接向 HTML 输出流写内容。

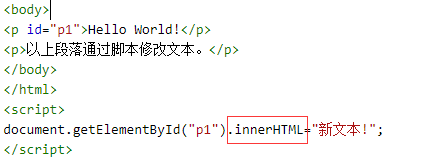


注意：绝对不要在文档（DOM）加载完成之后使用document.write()。这会覆盖文档。

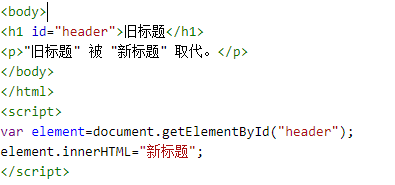
## 3.8改变 HTML 内容

修改 HTML 内容的最简单的方法是使用 innerHTML 属性。

本例改变了 <p>元素的内容：



本例改变了 <h1> 元素的内容：



实例讲解：

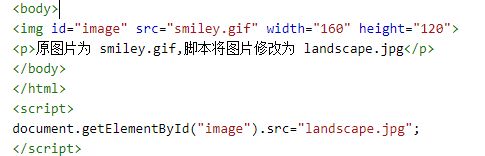
1. 上面的 HTML 文档含有 id="header" 的 <h1> 元素
2. 我们使用 HTML DOM 来获得 id="header" 的元素
3. JavaScript 更改此元素的内容 (innerHTML)

## 3.9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).attribute=新属性值

本例改变了 <img> 元素的 src 属性：



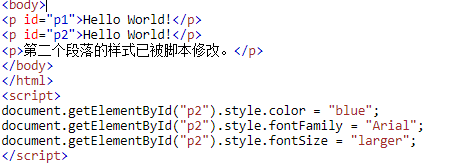
实例讲解：

1. 上面的 HTML 文档含有 id="image" 的 <img> 元素
2. 我们使用 HTML DOM 来获得 id="image" 的元素
3. JavaScript 更改此元素的属性（把 "smiley.gif" 改为 "landscape.jpg"）

## 3.10改变 CSS

如需更改 HTML 元素的样式，请使用此语法：

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



# 4 event对象 （事件源）

# <https://www.runoob.com/jquery/jquery-ref-events.html>

## 4.1 事件源 -- e 表示 Event对象(事件对象)

\* 认识event对象实例:

document.getElementById("myButton1").onmouseover = function(e){

//alert("lu");

console.info(e)

}

e.target; // 就是这个按钮本生 == 事件源

document.getElementById("myButton1") = e;

## 4.2 以参数注入的方式

// Assuming myButton is a button element

myButton.onclick = function(event){alert('Hello world');};

该函数在定义时，可以传入一个 event 形式的参数。

4.3 课堂实操

注册事件:

1. 内部 调用函数(注意: 这个函数必须是全局函数)

2. 外部找节点,注册事件

3. 对象addEventLister (没有on类型事件,命名函数, false) --> 绑定事件

1.1 鼠标事件续

// 为什么点击body的空白处不行,为点击body中的文字就可以? 答: 因为 "事件源"

// window 是最顶级的

// window有无数个事件对象,只有一个事件

// innerWidth , innerHeight

//点击整个页面都会执行whichButton(event) 因为innerWidth , innerHeight沾满了整个窗口

//document.getElementsByClassName("main")[0].style.width = window.innerWidth + "px";

//document.getElementsByClassName("main")[0].style.height = window.innerHeight + "px";

// 对整个窗口执行时间注册

// onmousedown 点击开始触发 是 事件函数

// onmouseout 出,onmouseover经过

window.onmousedown = function(e){

//whichButton(e);

console.info("onmousedown");

}

// onmouseup 点击结束触发

window.onmouseup = function(e) {

//whichButton(e);

console.info("onmouseup");

}

window.onclick = function(e) {

console.info("onclick");

}

// document.oncontextmenu // 页面上的右键事件

/\* document.oncontextmenu = function(e){

return false; // false禁用右键菜单

return true; // true启用右键菜单

} \*/

// 鼠标的移动 onmousemove

var counter1 = 0;

window.onmousemove = function(e) {

counter1++;

//console.info(e);

console.info(counter1);

if( e.movementX < 0){

console.info("<---左移");

}else{

console.info("向右移----");

}

if( e.movementY < 0){

console.info("向上移===");

}else{

console.info("下移\*\*\*");

}

}

// 课堂练习1: 对body 写一个鼠标监听事件,控制台输出按下了鼠标的那个键

// 为什么点击body的空白处不行,为点击body中的文字就可以? 答: 因为 "事件源"

function whichButton(event) {

var btnNum = event.button;

if (btnNum == 2) {

alert("您点击了鼠标右键！")

} else if (btnNum == 0) {

alert("您点击了鼠标左键！")

} else if (btnNum == 1) {

alert("您点击了鼠标中键！");

} else {

alert("您点击了" + btnNum + "号键，我不能确定它的名称。");

}

}

# 5 dom事件

事件 /需要做的四件事 1.元素 2.属性 3.方法 4.事件

## 参考文献

各种事件可参考：<https://www.runoob.com/tags/ref-eventattributes.html>

## 拖放参考：<https://www.runoob.com/html/html5-draganddrop.html>

## <https://blog.csdn.net/ssisse/article/details/52628739>

## 5.1 事件的绑定和解绑

### 原生dom的绑定和解绑

事件的绑定：1.对象.on事件名字=事件处理函数--->绑定事件

2.对象.addEventListener("没有on的事件类型",命名函数,false);---绑定事件

3. 对象.attachEvent("on事件类型",命名函数);---绑定事件

事件的解绑：1. 对象.on事件名字=null;

2. 对象.removeEventListener("没有on的事件类型",函数名字,false);

3. 对象.detachEvent("on事件类型",函数名字);

## JQ 事件绑定：

绑定方法1 - 直接绑定

$("#bt1").click(function() {

btClick();

});

$("#bt1").click( btClick );

绑定方法2 - on函数

$("#bt1").on( "click" , function(){ console.info("触发了点击事件"); } );

$("#bt1").on( "click" , btClick );

绑定方法3 - 原生方式

$("#bt1")[0].onclick = btClick ;

$("#bt1")[0].onclick = function(e){

btClick();

};

绑定方法 4 - bind 函数

$("#bt1").bind("click", function() {

btClick();

});  
 绑定方法5 - 通过父元素操作 （针对未来元素）  
 $("#father").delegate("p", "click", function() {

 $(this).css({

 color: "blue"

 });

 });

解绑  
unbind()

## 鼠标事件

onclick：单击

ondblclick：双击

onmousedown：鼠标按下

onmouseup：鼠标抬起

onmouseover：鼠标悬浮

onmouseout：鼠标离开

onmousemove：鼠标移动

onmouseenter：鼠标进入

onmouseleave：鼠标离开

onmousewheel：滚轮调整

oncontextmenu：右键菜单

*<button onclick="myClick()">鼠标单击</button>*

*<button ondblclick="myDBClick()">鼠标双击</button>*

*<button onmousedown="myMouseDown()" onmouseup="myMouseUp()">鼠标按下和抬起</button>*

*<button onmouseover="myMouseOver()" onmouseout="myMouseOut()">鼠标悬浮和离开</button>*

*<button onmousemove="myMouseMove()">鼠标移动</button>*

*<button onmouseenter="myMouseEnter()" onmouseleave="myMouseLeave()">鼠标进入和离开</button>*

*<script>*

*function myClick() {*

*console.log("你单击了按钮！");*

*}*

*function myDBClick() {*

*console.log("你双击了按钮！");*

*}*

*function myMouseDown() {*

*console.log("鼠标按下了！");*

*}*

*function myMouseUp() {*

*console.log("鼠标抬起了！");*

*}*

*function myMouseOver() {*

*console.log("鼠标悬浮！");*

*}*

*function myMouseOut() {*

*console.log("鼠标离开！")*

*}*

*function myMouseMove() {*

*console.log("鼠标移动！")*

*}*

*function myMouseEnter() {*

*console.log("鼠标进入！")*

*}*

*function myMouseLeave() {*

*console.log("鼠标离开！")*

*}*

## 拖拽事件

## 参考：<https://www.runoob.com/html/html5-draganddrop.html>

## <https://blog.csdn.net/ssisse/article/details/52628739>

## draggable="true"：设置元素为可拖放

ondrop：放置被拖数据时

ondragover：规定在何处放置被拖动的数据

ondragstart：开始拖动元素

ondragend：完成拖动元素

ondrag：拖动时

## 键盘事件

onkeydown：按键按下

onkeyup：按键抬起

onkeypress：按键按下抬起

次序为: onkeydown, onkeyup, onkeypress (鼠标事件同)

*<body>*

*<input id="name" type="text" οnkeydοwn="myKeyDown(this.id)" οnkeyup="myKeyUp(this.id)">*

*</body>*

*<script>*

*/\*输出输入的字符\*/*

*function myKeyDown(id) {*

*console.log(document.getElementById(id).value);*

*}*

*/\*按键结束，字体转换为大写\*/*

*function myKeyUp(id) {*

*var text = document.getElementById(id).value;*

*document.getElementById(id).value = text.toUpperCase();*

*}*

*</script>*

## 浏览器事件

onload：文档加载完成

onselect：被选中的时候

onchange：内容被改变

onfocus：得到光标

onresize：窗口尺寸变化

onscroll：滚动条移动

*<body οnlοad="loaded()">*

*<div style="height: 3000px" ></div>*

*<input type="text" id="name" οnselect="mySelect(this.id)">*

*<input type="text" id="name2" οnchange="myChange(this.id)">*

*<input type="text" id="name3" οnfοcus="myFocus()">*

*</body>*

*<script>*

*window.onload = function () {*

*console.log("文档加载完毕！");*

*};*

*/\*window.onunload = function () {*

*alert("文档被关闭！");*

*};\*/*

*/\*打印选中的文本\*/*

*function mySelect(id) {*

*var text = document.getElementById(id).value;*

*console.log(text);*

*}*

*/\*内容被改变时\*/*

*function myChange(id) {*

*var text = document.getElementById(id).value;*

*console.log(text);*

*}*

*/\*得到光标\*/*

*function myFocus() {*

*console.log("得到光标！");*

*}*

*/\*窗口尺寸变化\*/*

*window.onresize = function () {*

*console.log("窗口变化！")*

*};*

*/\*滚动条移动\*/*

*window.onscroll = function () {*

*console.log("滚动"); }</script>*

## 表单事件

on[blur](https://www.runoob.com/tags/ev-onblur.html)：失去焦点时

onfocus：获得焦点时

onchange：内容改变时

onselect：选取元素时

onsubmit：提交表单时

oninput：获取用户输入时

oninvalid：元素无效时

# 6 . window对象 window的事件

查询资料网址: <https://www.runoob.com/jsref/obj-window.html>

Window 对象表示浏览器中打开的窗口,所有浏览器都对其支持.它是 bom中的顶级对象,他的子类有 document--console--math—webSocket--location等。

## 浏览器整个窗口与框架

1. 没有子框架时：window、self、top和parent都相等，都是指最顶层框架，即浏览器窗口。
2. 有子框架时：window、self 和 top 都为最顶层框架，parent指向当前框架的直接上层框架。
3. 如果页面中包含框架，则每个框架都有自己的window对象，并且保存在父窗口的frames集合中。
4. 可以通过索引（从0开始，从左至右，从上至下）或者框架名称name属性访问相应的window对象。每个window对象都有一个name属性，表示所处框架的名称。

## Window对象的属性

下边中是window 对象的一些常用的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| closed | 返回窗口是否已被关闭。 |
| [document](https://www.runoob.com/jsref/dom-obj-document.html) | 对 Document 对象的只读引用 |
| frames | 返回窗口中所有命名的框架。该集合是 Window 对象的数组，每个 Window 对象在窗口中含有一个框架。 |
| [history](https://www.runoob.com/jsref/obj-history.html) | 对 History 对象的只读引用。请参数 |
| [innerHeight](https://www.runoob.com/jsref/prop-win-innerheight.html) | 返回窗口的文档显示区的高度。 |
| [innerWidth](https://www.runoob.com/jsref/prop-win-innerheight.html) | 返回窗口的文档显示区的宽度。 |
| [localStorage](https://www.runoob.com/jsref/prop-win-localstorage.html) | 在浏览器中存储 key/value 对。没有过期时间。 |
| [location](https://www.runoob.com/jsref/obj-location.html) | 用于窗口或框架的 Location 对象 |
| [navigator](https://www.runoob.com/jsref/obj-navigator.html) | 对 Navigator 对象的只读引用 |
| [opener](https://www.runoob.com/jsref/prop-win-opener.html) | 返回对创建此窗口的窗口的引用。 |
| [parent](https://www.runoob.com/jsref/prop-win-parent.html) | 返回父窗口。 |
| [screen](https://www.runoob.com/jsref/obj-screen.html) | 对 Screen 对象的只读引用。 |
| [screenX](https://www.runoob.com/jsref/prop-win-screenx.html) | 返回相对于屏幕窗口的x坐标 |
| [self](https://www.runoob.com/jsref/prop-win-self.html) | 返回对当前窗口的引用。等价于 Window 属性。 |
| [status](https://www.runoob.com/jsref/prop-win-status.html) | 设置窗口状态栏的文本。 |
| [top](https://www.runoob.com/jsref/prop-win-top.html) | 返回最顶层的父窗口。 |

## Window对象的方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| [alert()](https://www.runoob.com/jsref/met-win-alert.html) | 显示带有一段消息和一个确认按钮的警告框 |
| [atob()](https://www.runoob.com/jsref/met-win-atob.html) | 解码一个 base-64 编码的字符串。 |
| [btoa()](https://www.runoob.com/jsref/met-win-btoa.html) | 创建一个 base-64 编码的字符串。 |
| [blur()](https://www.runoob.com/jsref/met-win-blur.html) | 把键盘焦点从顶层窗口移开。 |
| [clearInterval()](https://www.runoob.com/jsref/met-win-clearinterval.html) | 取消由 setInterval() 设置的 timeout。 |
| [clearTimeout()](https://www.runoob.com/jsref/met-win-cleartimeout.html) | 取消由 setTimeout() 方法设置的 timeout。 |
| [close()](https://www.runoob.com/jsref/met-win-close.html) | 关闭浏览器窗口。 |
| [confirm()](https://www.runoob.com/jsref/met-win-confirm.html) | 显示带有一段消息以及确认按钮和取消按钮的对话框。 |
| [createPopup()](https://www.runoob.com/jsref/met-win-createpopup.html) | 创建一个 pop-up 窗口。 |
| [focus()](https://www.runoob.com/jsref/met-win-focus.html) | 把键盘焦点给予一个窗口。 |
| [getComputedStyle()](https://www.runoob.com/jsref/jsref-getcomputedstyle.html) | 获取指定元素的 CSS 样式。 |
| [moveTo()](https://www.runoob.com/jsref/met-win-moveto.html) | 把窗口的左上角移动到一个指定的坐标。 |
| [moveBy()](https://www.runoob.com/jsref/met-win-moveby.html) | 可相对窗口的当前坐标把它移动指定的像素。 |
| [open()](https://www.runoob.com/jsref/met-win-open.html) | 打开一个新的浏览器窗口或查找一个已命名的窗口。 |
| [print()](https://www.runoob.com/jsref/met-win-print.html) | 打印当前窗口的内容。 |
| [prompt()](https://www.runoob.com/jsref/met-win-prompt.html) | 显示可提示用户输入的对话框。 |
| [resizeBy()](https://www.runoob.com/jsref/met-win-resizeby.html) | 按照指定的像素调整窗口的大小。 |
| [scrollTo()](https://www.runoob.com/jsref/met-win-scrollto.html) | 把内容滚动到指定的坐标。 |
| [setInterval()](https://www.runoob.com/jsref/met-win-setinterval.html) | 按照指定的周期（以毫秒计）来调用函数或计算表达式。 |
| [setTimeout()](https://www.runoob.com/jsref/met-win-settimeout.html) | 在指定的毫秒数后调用函数或计算表达式。 |
| [stop()](https://www.runoob.com/jsref/met-win-stop.html) | 停止页面载入。 |

## Screen对象说明

Screen 对象包含有关客户端显示屏幕的信息。

主要属性有: [availHeight](https://m.runoob.com/jsref/prop-screen-availheight.html) / [availWidth](https://m.runoob.com/jsref/prop-screen-availwidth.html)/ [colorDepth](https://m.runoob.com/jsref/prop-screen-colordepth.html)/ [height](https://m.runoob.com/jsref/prop-screen-height.html)/ [pixelDepth](https://m.runoob.com/jsref/prop-screen-pixeldepth.html)/ [width](https://m.runoob.com/jsref/prop-screen-width.html)

## History对象说明

History 对象包含用户（在浏览器窗口中）访问过的 URL。

History 对象是 window 对象的一部分，可通过 window.history 属性对其进行访问。

history对象的主要属性和方法有：

|  |  |
| --- | --- |
| 方法 | 说明 |
| [back()](https://www.runoob.com/jsref/met-his-back.html) | 加载 history 列表中的前一个 URL |
| [forward()](https://www.runoob.com/jsref/met-his-forward.html) | 加载 history 列表中的下一个 URL |
| [go()](https://www.runoob.com/jsref/met-his-go.html) | 加载 history 列表中的某个具体页面 |

## Location对象说明

Location 对象包含有关当前 URL 的信息。

Location 对象是 window 对象的一部分，可通过 window.Location 属性对其进行访问。

## 练习代码

document.getElementById("bt1").onclick = function() {

window.location = "https://baidu.com";

}

document.getElementById("windowClose").onclick = function() {

window.close(); // 关闭当前浏览器窗口

}

// open 参数列表 ( url ,name , xxx)

// 关于name参数,可以自定义,也可以使用这几个值: \_self 当前窗口打开

// \_blank(默认值) 新窗口打开 \_top 表示顶层窗口

document.getElementById("windowOpen").onclick = function() {

\_son = window.open("windows相关2.html","\_self");// 打开浏览器窗口

}

document.getElementById("windowCloseSon").onclick = function() {

//\_son.open("http://www.hao123.com");

\_son.close();

}

//关闭网页之前要做的事情

window.onunload = function(){

//\_son.close();

}

// 打开页面的时候顺带打开新的子页面

window.onload = function(){

// \_son = window.open("windows相关2.html");

}

// 刷新页面

document.getElementById("reload").onclick = function() {

location.reload();

}

# 7 . 浏览器对象模型(bom) 的运行原理和过程

参考链接:<http://www.wangdoc.com/javascript/bom/engine.html>

-- BOM（Brower Object Model浏览器对象模型）

JavaScript 是浏览器的内置脚本语言。也就是说，浏览器内置了 JavaScript 引擎，并且提供各种接口，让 JavaScript 脚本可以控制浏览器的各种功能。一旦网页内嵌了 JavaScript脚本，浏览器加载网页，就会去执行脚本，从而达到操作浏览器的目的，实现网页的各种动态效果。

## 浏览器的组成 -- 两个引擎

**渲染引擎** -- 主要作用是将网页代码渲染为用户视觉可以感知的平面文档。

渲染引擎处理网页，通常分成四个阶段。

1.解析代码：HTML 代码解析为 DOM，CSS 代码解析为 CSSOM（CSS Object Model）。

2.对象合成：将 DOM 和 CSSOM 合成一棵渲染树（render tree）。

3.布局：计算出渲染树的布局（layout）。

4.绘制：将渲染树绘制到屏幕。

### JavaScript 引擎 -- 主要作用是读取网页中的 JavaScript 代码，对其处理后运行。

## 工作原理

浏览器加载 JavaScript 脚本，主要通过<script>元素完成的.

因为浏览器是单线程,且逐行翻译执行的,浏览器一边下载 HTML 网页，一边开始解析。也就是说，不等到下载完，就开始解析。解析过程中，浏览器发现<script>元素，就暂停解析，把网页渲染的控制权转交给 JavaScript 引擎。如果<script>元素引用了外部脚本，就下载该脚本再执行，否则就直接执行代码。JavaScript 引擎执行完毕，控制权交还渲染引擎，恢复往下解析 HTML 网页。

加载外部脚本时，浏览器会暂停页面渲染，等待脚本下载并执行完成后，再继续渲染。原因是 JavaScript 代码可以修改 DOM，所以必须把控制权让给它，否则会导致复杂的线程竞赛的问题。

## 阻塞效应

如果外部脚本加载时间很长（一直无法完成下载），那么浏览器就会一直等待脚本下载完成，造成网页长时间失去响应，浏览器就会呈现“假死”状态，这被称为“阻塞效应”.

为了避免这种情况，较好的做法是将<script>标签都放在页面底部，而不是头部。脚本文件都放在网页尾部加载，还有一个好处。因为在 DOM 结构生成之前就调用 DOM 节点，JavaScript 会报错，如果脚本都在网页尾部加载，就不存在这个问题，因为这时 DOM 肯定已经生成了。

3.1 解决方法 -- 设定DOMContentLoaded事件的回调函数。



## 脚本的动态加载

<script>元素还可以动态生成，生成后再插入页面，从而实现脚本的动态加载。这种方法的好处是，动态生成的script标签不会阻塞页面渲染，也就不会造成浏览器假死。



## 重绘 重流

渲染树转换为网页布局，称为“布局流”（flow）；布局显示到页面的这个过程，称为“绘制”（paint）。它们都具有阻塞效应，并且会耗费很多时间和计算资源。

页面生成以后，脚本操作和样式表操作，都会触发“重流”（reflow）和“重绘”（repaint）。用户的互动也会触发重流和重绘，比如设置了鼠标悬停（a:hover）效果、页面滚动、在输入框中输入文本、改变窗口大小等等。

重流和重绘并不一定一起发生，重流必然导致重绘，重绘不一定需要重流。比如改变元素颜色，只会导致重绘，而不会导致重流；改变元素的布局，则会导致重绘和重流。

大多数情况下，浏览器会智能判断，将重流和重绘只限制到相关的子树上面，最小化所耗费的代价，而不会全局重新生成网页。

作为开发者，应该尽量设法降低重绘的次数和成本。比如，尽量不要变动高层的 DOM 元素，而以底层 DOM 元素的变动代替；再比如，重绘table布局和flex布局，开销都会比较大。var foo = document.getElementById('foobar');

优化技巧:

读取 DOM 或者写入 DOM，尽量写在一起，不要混杂。不要读取一个 DOM 节点，然后立刻写入，接着再读取一个 DOM 节点。

缓存 DOM 信息。

不要一项一项地改变样式，而是使用 CSS class 一次性改变样式。

使用documentFragment操作 DOM

动画使用absolute定位或fixed定位，这样可以减少对其他元素的影响。

只在必要时才显示隐藏元素。

使用window.requestAnimationFrame()，因为它可以把代码推迟到下一次重流时执行，而不是立即要求页面重流。

使用虚拟 DOM（virtual DOM）库。