### BOM

BOM模型Browser Object Model也称为文档对象模型，它包含浏览器相关的属性和方法，例如前进后退按钮，控制地址栏，关闭浏览器窗口，打开新窗口等等。

### window

window对象是JavaScript中的顶级对象。所有定义在全局作用域中的变量、函数都会变成window对象的属性和方法。全局变量是 window 对象的属性。全局函数是 window 对象的方法。如window.fn()

window是窗口对象，用来控制当前窗口，打开、关闭、放缩。

|  |  |
| --- | --- |
| window对象下的属性 | |
| window.name |  |
| window.top | 返回最顶层的先辈窗口（太祖） |
| window.innerHeight  This property is read only; it has no default value. | Height (in pixels) of the browser window **viewport** including, if rendered, the horizontal scrollbar.  A **viewport** represents a polygonal (normally rectangular) area in computer graphics that is currently being viewed. In web browser terms, it refers to **the part of the document you're viewing which is currently visible in its window**. Content outside the viewport is currently not displayed onscreen. （不包括工具栏和控制台，包括滚动条） |
| window.parent | 返回父窗口 |
| window.self | 返回对当前窗口的引用 |
| window.opener | 返回对创建此窗口的窗口的引用 |

|  |  |
| --- | --- |
| window对象下的对象 | |
| document对象 |  |
| **location对象** | 地址对象，用来获取当前URL的信息  location.href;location.search;location.reload |
| **history对象** | 浏览记录，有几个方法：back(); forward(); go(); |
| navigator对象 | 浏览器对象，包含有关访问者浏览器的信息，但不应该被用于检测浏览器版本  window.navigator.userAgent  //里面有用户的操作系统和浏览器的信息 |
| screen对象 | 获取屏幕的相关信息  window.screen.width / height //未刨除任务栏  window.screen.availHeight / availWidth |

|  |  |
| --- | --- |
| location对象的属性和方法 | |
| location.href | 当前页面的地址，对该值赋值可以跳转到另一个页面 |
| location.hash | 获取哈希值 |
| location.reload() | 刷新页面 |
| location.search() | 获取参数 |

|  |  |
| --- | --- |
| window.location.hash 锚点 # | window.location. |
| window.location.host | window.location.port 端口号 |
| window.location.hostname 服务器名 | window.location.protocol 协议名 |
| window.location.pathname 路径名 | window.location.search 参数 ？ |

|  |  |  |
| --- | --- | --- |
| window对象的方法 | | |
| window.open  返回新打开窗口的window对象 | window.open( “url”, “\_self ”, “width = 200, height = 300”);  url，窗口名，窗口的一些属性  window.open("http://www.baidu.com", "\_blank"); 新窗口 | |
| close() | window.close() | myWindow.close(); |
| setTimeout() | 定时器 | 定时器里面不要用this，因为这4个方法是被window调用的 |
| clearTimeout() |
| setInterval() | 间歇定时器 |
| clearInterval() |
| alert()  prompt()  confirm() |  |  |
| resizeTo() |  |  |

<script type="text/javascript">    
    window.onbeforeunload = onbeforeunload\_handler;    
    window.onunload = onunload\_handler;    
    function onbeforeunload\_handler(){    
        var warning="确认退出?";            
        return warning;    
    }    
        
    function onunload\_handler(){    
        var warning="谢谢光临";    
        alert(warning);    
    }    
</script>

window.closed属性可返回一个布尔值，该值声明了窗口是否已经关闭。该属性为只读。当浏览器窗口关闭时，表示该窗口的 window 对象并不会消失，它将继续存在，不过它的 closed 属性将设置为 true。

setInterval使用的时候是先等待设定的时间，再执行代码。

var sec = 5;  
timer = setInterval(function () {  
 btn.value = sec + "seconds after";

sec---;

if(sec===0){  
 clearInterval(timer);

btn.value="good"; btn.disabled=false;  
}  
},1000);

Q: 为什么显示 “2 seconds after” 以后直接变成“good”?为什么没有显示 “1 seconds after”？

A: 因为显示完 “1 seconds after” 之后，sec变成0，然后btn.value马上变成 "good" ，时间太快，眼睛反应不过来，所以看不到。

onload“加载完成”指的是包括外部资源（图片、样式、脚本）等所有资源都加载完。JS中涉及图片的都要写在onload里面, 因为浏览器有优化，所以很多外部资源在加载结构的时候并没有加载。

### 事件队列

JavaScript 是**单线程**语言，但它允许通过设置超时值和间歇时间值来调度代码在特定的时刻执行。  
JavaScript 是一个单线程序的解释器，因此一定时间内只能执行一段代码。为了控制要执行的代码，就有一个JavaScript 任务队列。这些任务会按照将它们添加到队列的顺序执行。setTimeout() 的第二个参数告诉JavaScript 再过多长时间把当前任务添加到队列中。如果队列是空的，那么添加的代码会**立即**

**执行**；如果队列不是空的，那么它就要等前面的代码执行完了以后再执行。

比如：

for (var i = 0; I < 5; ++i) {

setTimeout(function() {

console.log("i= " + i);

}, 1000\*i)

}

有3个同学同时去食堂吃饭，A同学只能等1分钟，B同学能等2分钟，C同学能等3分钟。但是饭等了2.5分钟才做好，这时AB已经等得不耐烦了，立即开吃，而C还可以再等半分钟再吃。

一般认为，使用超时调用来模拟间歇调用是一种最佳模式。在开发环境下，很少使用真正的间歇调用，原因是后一个间歇调用可能会在前一个间歇调用结束之前启动。

setTimeout(function() {

if(flag \*定时器结束的条件) {

......

setTimeout(arguments.callee, interval);

}

}, interval)

### setInterval 中的this

一般情况下，间歇定时器中的函数是在interval之后被window调用的，所以this指向window

但是如果定时器中用的是箭头函数，this的指向就依赖于外层的函数，比如：

function fn() {

setTimeout(()=>console.log(this), 500);

}

fn.call({a:1}); //打印结果是：{a:1}

### 时间对象

var date = new Date(2005,10,1);

可以接受三种参数：

1. 2005，10，1日期的每一部分
2. "2005-10-1" 字符串的日期格式
3. 表示日期的毫秒形式1128096000000

var date = Date.parse("2005-10-1");

把字符串转换成日期的毫秒形式，如果字符串的格式不是时间的正确格式返回NaN

|  |  |
| --- | --- |
| 获取日期中指定的部分 | |
| getTime() | 返回毫秒数，和valueOf()结果一样 |
| getMilliseconds() |  |
| getSeconds() | 返回0-59 |
| getMinutes() | 返回0-59 |
| getHours() | 返回0-23 |
| getDay() | 返回星期几，周日是0，周6是6 |
| getDate() | 返回当前月的第几天 |
| getMonth() | 返回月份，0-11 |
| getFullYear() | 返回4位的年份，如2016 |

console.log(date.toDateString()); 打印字符串类型的日期  
console.log(date.toTimeString()); 打印字符串类型的时间

date.valueOf()); 这是一个13位的数字，表示从1970-1-1 00:00:00:000 开始到现在的毫秒值。

transform: rotate(deg); 旋转的度数

创建指定时间的日期对象：var date = new Date(2005,10,1) 参数还可以是“2005-10-1”毫秒数

Date parse()：将字符串转化为日期对象，var date = Date.parse(“2005-10-1”);若参数格式不正确就返回NaN

|  |  |  |
| --- | --- | --- |
| 事件 | window.onbeforeunload | window.onunload |
| 调用时机 | 去服务器读取新的页面时调用，此时还没开始读取 | 已经从服务器上读到了需要加载的新的页面，在即将替换掉当前页面时调用 |
|  | 可以阻止页面更新/关闭 | 无法阻止页面的更新和关闭 |

function Num(value) {

//Num既有强制类型转换的功能，也有构造函数的功能。

this.value = parseInt(value);

this.name = "number";

var num = parseInt(value);

return num;  
}  
var num1 = Num("1");  
var num2 = new Num("1"); //new能够使函数中的this返回，但return返回的值类型的值就不返回了。因为有new就既不需要new Object也不需要return。

### 特效

|  |  |  |
| --- | --- | --- |
|  | demo.style.height | demo.offsetHeight |
| 所得数值 | 带单位的字符串 | 没有单位的数值 |
| 数值意义 |  | 包括height / padding / border，不包括margin，也就是说offsetHeight 包括整个盒子 |
| 性质 | 可读可写，可以设置行内样式 | 只读属性，不可以设置样式 |
| 作用 | 能获取行内样式 | 行内样式、内部样式、外部样式都能获取 |

offsetLeft位移以border左上角为基准，即父盒子的padding + 定位的left + 子盒子的margin

style.left以margin左上角为基准，即定位时的left

bshare.com

事件：

|  |  |  |  |
| --- | --- | --- | --- |
| onclick | onmouseover | onmousedown | onmousemove |
| ondblclick | onmouseout | onmouseup | oninput |