

Chapter 8

BOM

· BOM：浏览器对象模型 → 核心

BOM提供了很多对象，用于访问浏览器的功能，这些功能与任何网页内容无关。

从事实上的规范导致BOM既没有意义又有问题，因为浏览器提供商会影响各自的想法随意扩展它。

浏览器之间不同的对象就成为了事实上的标准。这些对象在浏览器中得以存在，很大程度上是由于它们提供了与浏览器的互操作性。

W3C为了把浏览器中JS最基本的部分标准化，已经将BOM的主要方面纳入了HTML5的规范中。

Windows 对象

· Windows 有双重角色： { 它既是通过JS访问浏览器窗口的一个接口，又是ECMAScript规定的Global对象。
→ 网页中定义的任何一个对象、变量和函数，都以window作为其Global对象，因此有方法 parseInt() 等方法。

所有在全局作用域中声明的变量、函数都会变成 window 对象的属性和方法。

! 定义全局变量与在 window 对象上直接定义属性还是有一点差别：全局变量不能通过 delete 操作符删除，而直接在 window 对象上的定义的属性可以。 语句会报错

! 尝试访问未声明的变量会抛出错误，但是通过查询 window 对象，可以知道某个可能未声明的变量是否存在。

```
// 这里会抛出错误，因为 oldValue 未定义  
var newValue = oldValue;  
  
// 这里不会抛出错误，因为这是一次属性查询  
// newValue 的值是 undefined  
var newValue = window.oldValue;
```

Windows mobile 平台的旧浏览器不支持通过 window.property = value 之类的形式，在 window 对象上创建新的属性或方法。可是，在全局作用域中声明的所有变量和函数，照样会变成 window 对象的成员。

窗口关系及框架

· 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 frames 集合中。在 frames 集合中，可以通过数值索引或框架名称来访问相应的 window 对象，每个 window 对象都有一个 name 属性，其中包含框架的名称。

所有对象始终指向最高（外层）的框架。对于在一个框架中编写的任何代码来说，其中的 window 对象指向的是那个框架的特定实例，而非最高层的框架。

window.frames[0] window.frames["topFrame"] top.frames[0] top.frames["topFrame"] frames[0] frames["topFrame"]	window.frames[2] window.frames["rightFrame"] top.frames[2] top.frames["rightFrame"] frames[2] frames["rightFrame"]
window.frames[1] window.frames["leftFrame"] top.frames[1] top.frames["leftFrame"] frames[1] frames["leftFrame"]	

→ 访问最高层的框架的方法

与 top 相对的另一个 window 对象是 parent。parent 对象始终指向当前框架的直接上层框架。
某些情况下，parent = top。在没有框架的情况下，parent 一定等于 top，都等于 window。

```

</head>
<frameset rows="100,*">
  <frame src="frame.htm" name="topFrame">
  <frameset cols="50%,50%">
    <frame src="anotherframe.htm" name="leftFrame">
    <frame src="anotherframeset.htm" name="rightFrame">
  </frameset>
</frameset>
</html>

```

```

<html>
  <head>
    <title>Frameset Example</title>
  </head>
  <frameset cols="50%,50%">
    <frame src="red.htm" name="redFrame">
    <frame src="blue.htm" name="blueFrame">
  </frameset>
</html>

```

窗口位置

- 用于不确定和修改 window 对象位置的属性和方法有很多
- IE, Safari, Opera, Chrome 都提供了 screenLeft 和 screenTop 属性，分别用于表示窗口相对屏幕左边缘和上边缘的位置。Firefox, Safari, Chrome 支持 screenX 和 screenY 提供相同的窗口位置信息。
- Opera 也支持这两个属性，但与 screenLeft 和 screenTop 属性并不对应，所以不在这里使用。

```

var leftPos = (typeof window.screenLeft == "number") ?
  window.screenLeft : window.screenX;
var topPos = (typeof window.screenTop == "number") ?
  window.screenTop : window.screenY;

```

→ 浏览器端上边缘
窗口左边缘和上边缘
位置。

但上面方法令人捉摸不透，无法在跨浏览器的条件下取得窗口左边缘和上边缘的精确坐标值

moveTo() 和 moveBy() 方法倒有可能将窗口精确地移动到一个新位置。

{ moveTo() : 接收新位置的 x 和 y 坐标值。
moveBy() : 接收的是在水平和垂直方向上移动的距离数。 }
→ 可能会被禁用
这两个方法都不适用于框架，只能对最外层的 window 对象使用。

窗口大小

- 跨浏览器确定一个窗口的大小不是一件简单的事。

IE9+, Firefox, Safari, Opera, Chrome 均提供了 4 个属性：innerWidth, innerHeight, outerWidth, outerHeight。

→ { - - - - - . Height } 嵌套于页面视口的信息。

IE6 中，上面必须在标准模式。

混杂模式的话，必须通过 document.body.clientWidth 和 document.body.clientHeight。

- 使用 resizerTo() 和 reszeBy() 可以调整浏览器窗口大小。

↓ 捕获新宽度和高度 ↓ 捕获新窗口与原窗口的宽度和高度之差。 → 可能被禁用 不适用于框架，只能对最外层的 window 对象使用

②弹出和打开窗口

- `window.open` 通常以导航到一个特定的 URL，也可以打开一个新浏览器窗口。
四个参数：URL、窗口目标、一个特性字符串，以及一个表示新页面是否取代浏览器历史记录中当前加载页面的布尔值。通常只传递第一个参数，最后一个参数只在不打开新窗口的情况下使用。

①弹出窗口

如果传递第二个参数，而且该参数是已有窗口或框架的名称，那么就在具有该名称的窗口或框架中加载第一个参数指定的 URL。
→ 也可以是 `_self`、`_parent`、`_top` 或 `_blank`。

如果第二个参数不是已存在的窗口或框架，就会根据参数三位位置上插入的字符串创建一个新窗口或新标签页。如果没有参数三，打开一个具有全部默认设置的新窗口。

第三个参数是一个逗号分隔的设置字符串，表示显示的特性。可以出现在这个字符串的设置选项：

设 置	值	说 明
<code>fullscreen</code>	<code>yes</code> 或 <code>no</code>	表示浏览器窗口是否最大化。仅限IE
<code>height</code>	数值	表示新窗口的高度。不能小于100
<code>left</code>	数值	表示新窗口的左坐标。不能是负值
<code>location</code>	<code>yes</code> 或 <code>no</code>	表示是否在浏览器窗口中显示地址栏。不同浏览器的默认值不同。如果设置为 <code>no</code> ，地址栏可能会隐藏，也可能会被禁用（取决于浏览器）
<code>menubar</code>	<code>yes</code> 或 <code>no</code>	表示是否在浏览器窗口中显示菜单栏。默认值为 <code>no</code>
<code>resizable</code>	<code>yes</code> 或 <code>no</code>	表示是否可以通过拖动浏览器窗口的边框改变其大小。默认值为 <code>no</code>
<code>scrollbars</code>	<code>yes</code> 或 <code>no</code>	表示如果内容在视口中显示不下，是否允许滚动。默认值为 <code>no</code>
<code>status</code>	<code>yes</code> 或 <code>no</code>	表示是否在浏览器窗口中显示状态栏。默认值为 <code>no</code>
<code>toolbar</code>	<code>yes</code> 或 <code>no</code>	表示是否在浏览器窗口中显示工具栏。默认值为 <code>no</code>
<code>top</code>	数值	表示新窗口的上坐标。不能是负值
<code>width</code>	数值	表示新窗口的宽度。不能小于100

如下：

```
window.open("http://www.wrox.com/", "wroxWindow",
            height=400, width=400, top=10, left=10, resizable=yes");
```

`window.open()` 方法会返回一个指向新窗口的引用。引用的对象与其他 `window` 对象大致相似，但可以进行更多控制。

`opener` 保存着打开它的原生窗口对象，这个属性只在弹出窗口中的最外层 `window` 对象 (`top`) 中有定义，而且指向调用 `window.open()` 的窗口或框架。

当一个标签页打开另一个标签页时（如果两个 `window` 对象之间需要通信，那么新标签页就不能运行在独立的进程中）。将 `opener = null`，那表示在单独的进程中运行新标签页 / 告诉浏览器新创建的标签页不要与打开它的标签页通信。

2. 安全限制

每个浏览器对弹窗限制不同。

3. 弹出窗口屏蔽程序

两种可能阻止浏览器内置的屏蔽程序阻止的弹出窗口。`window.open()` 返回 `null`。

两种可能阻止浏览器拦截或其他程序阻止的弹出窗口。→ 抛出一个错误

检测弹出窗口是否被屏蔽只是一方面，它并不会阻止浏览器显示与被屏蔽的弹出窗口有关的信息。

间歇调用和超时调用

· JS是单线程语言，允许设置超时值和间歇时间值来调度代码在特定的时刻执行。

↑ 在指定的时间
↓ 之后执行代码

↑ 每隔规定的
时间就执行一次代码

//不建议传递字符串!
setTimeout("alert('Hello world!')", 1000); → 传递字符串可能导致性能损失

//推荐的调用方式
setTimeout(function() {
 alert("Hello world!");
}, 1000); ✓

↳ 延缓多长时间的离散数，但经过去后指定的代码不一定能执行。

JS是单线程语言的解释器，一定时间内只能执行一段代码。为了控制需要执行的代码，就必须有一个JS队列。这些任务会按顺序将它们添加到队列的顺序执行。

调用 setTimeout 后，创建一个ID。表示超时调用，可以通过ID取消尚未执行的超时调用计划。

//设置超时调用
var timeoutId = setTimeout(function() {
 alert("Hello world!");
}, 1000);

//注意：把它取消
clearTimeout(timeoutId);

超时调用的代码都在全局作用域中。

非严格模式：this = window

严格模式：this = undefined

· 间歇调用与超时调用类似，只不过它会按照指定的时间间隔重复执行代码，直至间歇调用被取消或页面被卸载。

//不建议传递字符串！
setInterval("alert('Hello world!')", 10000);

//推荐的调用方式
setInterval(function() {
 alert("Hello world!");
}, 10000);

var num = 0;
var max = 10;
var intervalId = null;

function incrementNumber() {
 num++;

 //如果执行次数达到了 max 设定的值，则取消后续尚未执行的调用
 if (num == max) {
 clearInterval(intervalId);
 alert("Done");
 }
}

intervalId = setInterval(incrementNumber, 500);

↓ 这种模式也可以使用超时调用来实现

使用超时调用时，没有必要跟跟超时功。因为每次执行代码之后，如果不设置另一次超时调用，调用就会自动停止。

使用超时调用的模式从间歇调用的是一种最佳模式。

最好不要使用间歇调用！因为后一个间歇调用可能会在前一个间歇调用结束后启动。

var num = 0;
var max = 10;

function incrementNumber() {
 num++;

 //如果执行次数未达到 max 设定的值，则设置另一次超时调用
 if (num < max) {
 setTimeout(incrementNumber, 500);
 } else {
 alert("Done");
 }
}

setTimeout(incrementNumber, 500); ✓

系统对话框

系统对话框与在浏览器中显示的网页无关，也不包含HTML。它们的外观由OS及浏览器设置决定，而不是由CSS决定。

`alert("Hello world!")`

```
if (confirm("Are you sure?")) {  
    alert("I'm so glad you're sure! ");  
} else {  
    alert("I'm sorry to hear you're not sure. ");  
}
```



`prompt()`

`prompt("What's your name?", "Michael")`

chrome引进了新特性：执行过程打开
2个或多个对话框，从第二个开始每个都会
显示一个复选框：



• `print()` 和 `read()` 是异步显示的，chrome对话框计数器不计算它们在内！