**第8章 BOM**

本章内容

* 理解window对象---BOM核心
* 控制窗口、框架和弹出窗口
* 利用location对象中的页面信息
* 使用navigator对象了解浏览器

ECMAScript是JavaScript的核心，但如果要在Web中使用JavaScript，那么BOM（浏览器对象模型）怎无疑才是真正的核心。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。多年来，缺少事实上的规范导致BOM既有意思又有问题，因为浏览器提供的商会按照各自的想法随意去扩展它。于是，浏览器之间共有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为了把浏览器中的JavaScript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。

8.1 window对象

BOM的核心对象是window，它表示浏览器的一个实例。在浏览器中，window对象有双重角色，它既是JavaScript访问浏览器窗口的一个接口，又是ECMAScript规定的Global对象。这意味在网页中定义的任何一个对象、变量和函数，都以window作用Global对象，因此有权访问parseInt()等方法。

8.1.1 全局作用域

由于window对象同时扮演着ECMAScript中Global对象的角色，因此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。来看下面的例子。

1.       var age = 33;

2.       function sayAge() {

3.         alert(this.age);

4.       }

5.

6.       alert(window.age);    // 33

7.       sayAge();             // 33

8.       window.sayAge();      // 33

9.

我们在全局作用域中定义了一个变量age和一个函数sayAge()，它们被自动归在了window对象名下。于是，可以通过window.age访问age，可以通过window.sayAge()访问函数sayAge()。 由于sayAge()存在于全局作用域中，因此this.age被映射到window.age，最终显示仍然是正确的结果。

抛开全局变量会成为window对象的属性不谈，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window对象上定义的属性可以。

例如：

1.       var age = 29;

2.       window.color = "red";

3.

4.       // 在IE < 9时抛出错误，在其它所有浏览器中都返回false

5.       delete window.age;

6.

7.       // 在IE < 9时抛出错误，在其它所有浏览器中都返回true

8.       delete window.color;

9.

10.       alert(window.age); // 29

11.       alert(window.color); // undefined

12.

DeleteOperatorExample01.html

刚才使用var语句添加的window属性有一个名为[[Configurable]]的特性，这个特性的值被设置为false，因此这样定义的属性不可以通过delete操作符删除。IE8及更早的版本在遇到使用delete删除window属性的语句时，不管该属性最初是如何创建的，都会抛出错误，以示警告。IE9及更高版本不会抛出错误。

另外，还要记住一件事：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。例如：

1.       // 这里抛出错误，因为oldvalue未定义

2.       var newValue = oldValue;

3.

4.       // 这里不会抛出错误，因为这是一次属性查询

5.       // newValue的值是undefined

6.       var newValue = window.oldValue;

7.

本章后面将要讨论的很多全局JavaScript对象（如location和navigator）实际上都是window对象的属性。

Windows Mobile平台的IE浏览器不允许通过window.property = value之类的形式，直接在window对象上创建新的属性和方法。可是，在全局作用域中声明的所有变量和函数，照样会编程window对象的成员。

8.1.2 窗口关系及框架

如果页面中包含框架，则每个框架都拥有自己的window对象，并且保存在frame集合中。在frames集合中，可以通过数值索引（从0开始，从左至右，从上到下）或者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。下面是一个包含框架的页面：

1. <html>

2.   <head>

3.     <title>Frameset Example</title>

4.   </head>

5.   <frameset rows="160,\*">

6.     <frame src="./frame.html" name="topFrame">

7.       <frameset cols="50%,50%">

8.         <frame src="./anotherFrame.html" name="lefFrame">

9.         <frame src="./yetAnotherFrame.html" name="rightFrame">

10.       </frameset>

11.     </frameset>

12. </html>

13.

FramesetExample01.html

以上代码创建了一个框架集，其中一个框架居上，两个框架居下。对这个例子而言，可以通过window.frames[0]或者window.frames[“topFrame”]来引用上方的框架。不过，恐怕你最好使用top而非window来引用这些框架（例如，通过top.frames[0]）。

我们知道，top对象始终执行最高（最外）层的框架，也就是浏览器窗口。使用它可以确保在一个框架中正确地访问另一个框架。因为对于在一个框架中编写的任何代码来说，其中的window对象指向的都是那个框架的特定实例，而非最高层的框架。图8-1展示了在最高层窗口中，通过代码来访问前面例子中每个框架的不同方式。

文本

描述已自动生成

图8-1

与top相对的另一个window对象是parent。顾名思义，parent（父）对象始终指向当前框架的直接上层框架。在某些情况下，parent有可能等于top；但在没有框架的情况下，parent一定等于top（此时它们都等于window）。再看下面的例子。

1. <!DOCTYPE html>

2. <html lang="en">

3. <head>

4.   <meta charset="UTF-8">

5.   <meta name="viewport" content="width=device-width, initial-scale=1.0">

6.   <title>Document</title>

7. </head>

8. <frameset rows="100,\*">

9.   <frame src="./frame.html" name="topFrame">

10.   <frameset cols="40%,60%">

11.       <frame src="./anotherFrame.html" name="lefFrame">

12.       <frame src="./anotherFrameset.html" name="rightFrame">

13.   </frameset>

14. </frameset>

15. </html>

16.

frameset1.html

这个框架集中的一个框架包含了另一个框架集，该框架集的代码如下所示。

1. <!DOCTYPE html>

2. <html lang="en">

3. <head>

4.   <meta charset="UTF-8">

5.   <meta name="viewport" content="width=device-width, initial-scale=1.0">

6.   <title>Document</title>

7. </head>

8. <frameset cols="40%,60%">

9.     <frame src="./red.html" name="redFrame">

10.     <frame src="./blue.html" name="blueFrame">

11. </frameset>

12. </html>

13.

anotherframeset.html

浏览器在加载完第一个框架集以后，会继续将第二个框架集加载到rightFrame中。如果代码位于redFrame（或blueFrame）中，那么parent对象指向的是rightFrame。可是，如果代码位于topFrame中，则parent指向的是top，因为topFrame的直接上层框架就是最外层框架。图8-2展示了在将前面例子加载到浏览器之后，不同window对象的值。

图片包含 表格

描述已自动生成

图8-2

注意，除非最高层窗口是通过window.open()打开的（本章后面将会讨论），否则其window对象的name属性不会包含任何值。

与框架有关的最后一个对象self，它始终指向window；实际上，self和window对象可以互换使用。引入self对象的目的只是为了与top和parent对象对应起来，因此它不格外包含其他值。所有这些对象都是window对象的属性，可以通过window.parent、window.top等形式来访问。同时，这也意味着可以将不同层次的window对象连缀起来，例如window.parent.parent.frames[0]。

在使用框架的情况下，浏览器中会存在多个Global对象。在每个框架中定义的全局变量会自动成为框架中window对象的属性。由于每个window对象都包含原生类型的构造函数，因此每个框架都有一套自己的构造函数，这些构造函数一一对应，但并不相等。例如，top.Object并不等于top.frames[0].Object。这个问题会影响到对跨框架传递的对象使用instanceof操作符。

8.1.3 窗口位置

用来确定和修改window对象位置的属性和方法有很多。IE、Safari、Opera和Chrome都提供了screenLeft和screenTop属性，分别用于表示窗口相对于屏幕左边和上边的位置。Firefox则在screenX和screenY属性中提供相同的窗口位置信息，Safari和Chrome也同时支持这两个属性。Opera虽然也支持screenX和screenY属性，但与screenLeft和screenTop属性并不对应，因此建议大家不要在Opera中使用它们。使用下列代码可以跨浏览器取得窗口左边和上边的位置。

1.       var leftPos =

2.         typeof window.screenLeft == "number"

3.           ? window.screenLeft

4.           : window.screenX;

5.

6.       var topPos =

7.         typeof window.screenTop == "number"

8.           ? window.screenTop

9.           : window.screenY;

10.

WindowPositionExample01.html

这个例子运用二元操作符首先确定screenLeft和screenTop属性是否存在，如果是（在IE、Safari、Opera和Chrome），则取得这两个属性的值。如果不存在（Firefox中），则取得screenX和screenY的值。

在使用这些值的过程中，还必须注意一些小问题。在IE、Opera中，screenLeft和screenTop中保存的是从屏幕左边和上边到由window对象表示的页面可见区域的距离。换句话说，如果window对象是最外层对象，而且浏览器窗口紧贴屏幕最上端---即y轴坐标为0，那么screenTop的值就是位于页面可见区域上方的浏览器工具栏的像素高度。但是，在Chrome、Firefox和Safari中，screenY或screenTop中保存的是整个浏览器窗口相对于屏幕的坐标值，即在窗口的y轴坐标为0时返回0。

更让人捉摸不透是，Firefox、Safari和Chrome始终返回页面中每个框架的top.screenX和top.screenY值。即使在页面由于被设置了外边距而发生偏移的情况下，相对于window对象使用screenX和screenY每次也都会返回相同的值。而IE和Opera则会给出框架相对于屏幕边界的精确坐标值。

最终结果，就是无法在跨浏览器的条件下取得窗口左边和上边的精确坐标值。然而，使用moveTo()和moveBy()方法倒是有可能将窗口精确地移动到一个新位置。这两个方法都接受两个参数，其中moveTo()接受的是新位置的x和y坐标值，而moveBy()接受的是在水平和垂直方向上移动的像素数。下面来看几个例子：

1.       // 将窗口移动到屏幕左上角

2.       window.moveTo(0, 0);

3.

4.       // 将窗口向下移动100像素

5.       window.moveBy(0, 100);

6.

7.       // 将窗口移动到（200，300）

8.       window.moveTo(200, 300);

9.

10.       // 将窗口像左移动50像素

11.       window.moveBy(-50, 0);

12.

需要注意的是，这两个方法可能会被浏览器禁用；在Opera和IE7（及更高版本）中默认就是禁用的。另外，这两个方法都不适用于框架，只能对最外层的window对象使用。

8.1.4 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和Chrome均为此提供了4个属性：innerWidth、innerHeight、outerWidth和outerHeight。在IE9+、Safari和Firefox中，outerWidth和outerHeight返回浏览器窗口本身的尺寸（无论是从最外层的window对象还是从某个框架访问）。在Opera中，这两个属性的值表示页面视图容器的大小。而innerWidth和innerHeight则表示该容器中页面视图区的大小（减去边框宽度）。在Chrome中，outerWidth，outerHeight与innerWidth，innerHeight返回相同的值，即视口（viewport）大小而非浏览器窗口大小。

IE8及更早版本没有提供取得当前浏览器窗口尺寸的属性：不过，它通过DOM提供了页面可见区域的相关信息。

在IE9+、Firefox、Safari、Opera和Chrome中，document.documentElement.clientWidth和document.documentElement.clientHeight中保存了页面视口信息。在IE6中，这些属性必须在标准模式下才有效；如果是混杂模式，就必须通过document.body.clientWidth和document.body.clientHeight取得形同信息。而对于混杂模式下的Chrome，则无论通过document.documentElement还是document.body中的clientWidth和clientHeight属性，都可以取得视口的大小。

虽然最终无法确定浏览器窗口本身的大小，但却可以取得页面视口的大小，如下所示。

1.       var pageWidth = window.innerWidth, pageHeight = window.innerHeight;

2.

3.       if (typeof pageWidth != "number") {

4.         if (document.compatMode == "CSS1Compat") {

5.           pageWidth = document.documentElement.clientWidth;

6.           pageHeight = document.documentElement.clientHeight;

7.         } else {

8.           pageWidth = document.body.clientWidth;

9.           pageHeight = document.body.clientHeight;

10.         }

11.       }

12.

13.       console.log(pageWidth, pageHeight);

14.

WindowSizeExample01.html

在以上代码中，我们首先将window.innerWidth和window.innerHeight的值分别赋给了pageWidth和pageHeight。然后检查pageWidth中保存的是不是一个数值；如果不是，则通过检查document.compatMode（这个属性将在第10章全面讨论）来确定页面是否处于标准模式。如果是，则分别使用document.documentElement.clientWidth和document.documentElement.clientHeight的值。否则，就是用document.body.clientWidth和document.body.clientHeight的值。

对于移动设备，window.innerWidth和window.innerHeight保存着可见视口，也就是屏幕上可见页面区域的大小。移动IE浏览器不支持这些属性，但通过document.documentElement.clientWidth和document.documentElement.clientHeight提供了相同的信息。随着页面的缩放，这些值也会相应变化。

在其他移动浏览器中，document.documentElement度量的是布局视口，即渲染后页面的实际大小（与可见视口不同，可见视口只是整个页面中的一小部分）。移动IE浏览器把布局视口的信息保存在document.body.clientWidth和document.body.clientHeight中。这些值不会随着页面缩放变化。

由于与桌面浏览器间存在这些差异，最好是先检查一下用于是否在使用移动设备，然后再决定使用哪个属性。

有关移动设备视口的话题比较复杂，有很多非常规的情形，也有各种各样的建议。移动开发咨询师Peter-Paul Koch记述了他对这个问题的研究：<http://t.cn/zOZs0Tz>。如果你在做移动web开发，推荐你读一下这篇文章。

另外，使用resizeTo()和resizeBy()方法可以调整浏览器窗口的大小。这两个方法都接受两个参数，其中resizeTo()接收浏览器窗口的新宽度和新高度，而resizeBy()接收新窗口与原窗口的宽度和高度之差。来看下面的例子。

1. // 调整到100x100

2. window.resizeTo(100, 100);

3.

4. // 调整到200x150

5. window.resizeBy(100, 50);

6.

7. // 调整到300x300

8. window.resizeTo(300, 300);

9.

需要注意的是，这两个方法与移动窗口的方法类似，也有可能被浏览器禁用；而且，在Opera和IE7（及更高版本）中默认就是禁用的。另外，这两个方法同样不适用于框架，而且只能对最外层的window对象使用。

8.1.5 导航和打开窗口

使用window.open()方法既可以导航到一个特定的URL，也可以打开一个新的浏览器窗口。这个方法可以接收4个参数：要加载的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。通常只须传递第一个参数，最后一个参数只在打开新窗口的情况下使用。

如果为window.open()传递第二个参数，而且该参数是已有窗口或框架的名称，那么就会在具有该名称的窗口或框架中加载第一个参数指定的URL。看下面的例子。

1.       // 等同于 <a href="http://www.baidu.com" target="topFrame"></a>

2.       window.open("http://www.wrox.com/", "topFrame");

3.

调用这行代码，就如同用户点击了href属性为http://www.wrox.com，target属性为“topFrame”的链接。如果有一个名叫“topFrame”的窗口或者框架，就会在该窗口或框架加载这个URL；否则，就会创建一个新窗口并将其命名为“topFrame”。此外，第二个参数也可以是下列任何一个特殊的窗口名称；\_self、\_parent、\_top和\_blank。

1. **弹出窗口**

如果给window.open()传递的第二个参数并不是一个已经存在的窗口或框架，那么该方法就会根据在第三个参数位置上传入的字符串创建一个新窗口或新标签页。如果没有传入第三个参数，那么就会打开一个带有全部默认设置（工具栏、地址栏和状态栏等）的新浏览器窗口（或者打开一个新标签页-根据浏览器设置）。在不打开新窗口的情况下，会忽略第三个参数。

第三个参数是一个逗号分隔的设置字符串，表示在新窗口中都显示哪些特性。下表列出了可以出现在这个字符串中的设置选项。

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

表中所列的部分或全部设置选项，都可以通过逗号分隔的名值对列表来指定。其中，名值对以等号表示（注意，整个特性字符串中不允许出现空格），如下面的例子所示。

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

2.

这行代码会打开一个新的可以调整大小的窗口，窗口初始化大小为400x400像素，并且距屏幕上沿和左边各10个像素。

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，但我们可以对其进行更多的控制。例如，这些浏览器在默认情况下可能不允许我们针对主浏览器窗口调整大小或移动位置，但却允许我们针对通过window.open()创建的窗口调整大小或移动位置。通过这个返回的对象，可以像操作其他窗口一样操作新打开的窗口，如下所示。

1.       // 等同于 <a href="http://www.baidu.com" target="topFrame"></a>

2.       var wroxWin = window.open(

3.         "http://www.wrox.com/",

4.         "wroxWindow",

5.         "height=400,width=400,top=10,left=10,resizable=yes"

6.       );

7.

8.       // 移动位置

9.       wroxWin.moveTo(500, 500);

10.

11.       // 调整大小

12.       wroxWin.resizeTo(500, 500);

13.

14.       **wroxWin.close()**;

15.

但是，这个方法仅适用于通过window.open()打开的弹出窗口，对于浏览器的主窗口，如果没有得到用户的允许是不能关闭它的。不过，弹出窗口倒是可以调用top.close()在不经用户允许的情况下关闭自己。弹出窗口关闭之后，窗口的引用仍然还在，但除了像下面这样检测其closed属性之外，已经没有其他用处了。

1. wroxWin.close();

2. alert(wroxWin.closed); // true

3.

新创建的window对象有一个opener属性，其中保存着打开它的原始窗口对象。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。例如：

1. var wroxWin = window.open("http://www.wrox.com/", "wroxWindow", "height=400,width=400,top=0,left=0,resizable=yes,location=no");

2. alert(wroxWin.opener == window); // true

3.

虽然弹出窗口中有一个指针指向打开它的原始窗口，但原始窗口中并没有这样的指针指向弹出窗口。窗口并不跟踪记录它们打开的弹出窗口，因此我们只能在必要的时候自己来手动实现跟踪。

1. **安全限制**