第16章 HTML5 脚本编程

本章内容

- 使用跨文档消息传递
- 拖放 API
- 音频与视频

本书前面讨论过，HTML5 规范定义了很多新 HTML 标记。为了配合这些标记的变化，HTML5 规范也用显著篇幅定义了很多 JavaScript API。定义这些 API 的用意就是简化此前实现起来困难重重的任务，最终简化创建动态 Web 界面的工作。

16.1 跨文档消息传递

跨文档消息传送（cross-document messaging），有时候简称为 XDM，指的是在来自不同域的页面间传递消息。例如，www.wrox.com 域中的页面与位于一个内嵌框架中的 p2p.wrox.com 域中的页面通信。在 XDM 机制出现之前，要稳妥地实现这种通信需要花很多工夫。XDM 把这种机制规范化，让我们能既稳妥又简单地实现跨文档通信。

XDM 的核心是 postMessage()方法。在 HTML5 规范中，除了 XDM 部分之外的其他部分也会提到这个方法名，但都是为了同一个目的：向另一个地方传递数据。对于 XDM 而言，“另一个地方”指的是包含在当前页面中的<iframe>元素，或者由当前页面弹出的窗口。

postMessage()方法接收两个参数：一条消息和一个表示消息接收方来自哪个域的字符串。第二个参数对保障安全通信非常重要，可以防止浏览器把消息发送到不安全的地方。来看下面的例子。

```
//注意：所有支持 XDM 的浏览器也支持 iframe 的 contentWindow 属性
var iframeWindow = document.getElementById("myframe").contentWindow;
iframeWindow.postMessage("A secret", "http://www.wrox.com");
```

最后一行代码尝试向内嵌框架中发送一条消息，并指定框架中的文档必须来源于"http://www.wrox.com"域。如果来源匹配，消息会传递到内嵌框架中；否则，postMessage()什么也不做。这一限制可以避免窗口中的位置在你不知情的情况下发生改变。如果传给 postMessage()的第二个参数是"*"，则表示可以把消息发送给来自任何域的文档，但我们不推荐这样做。

接收到 XDM 消息时，会触发 window 对象的 message 事件。这个事件是以异步形式触发的，因此从发送消息到接收消息（触发接收窗口的 message 事件）可能要经过一段时间的延迟。触发 message 事件后，传递给 onmessage 处理程序的事件对象包含以下三方面的重要信息。

- data：作为 postMessage()第一个参数传入的字符串数据。
- origin：发送消息的文档所在的域，例如"http://www.wrox.com"。

- source：发送消息的文档的 window 对象的代理。这个代理对象主要用于在发送上一条消息的窗口中调用 postMessage()方法。如果发送消息的窗口来自同一个域，那这个对象就是 window。

接收到消息后验证发送窗口的来源是至关重要的。就像给 postMessage()方法指定第二个参数，以确保浏览器不会把消息发送给未知页面一样，在 onmessage 处理程序中检测消息来源可以确保传入的消息来自已知的页面。基本的检测模式如下。

```
EventUtil.addHandler(window, "message", function(event){

    //确保发送消息的域是已知的域
    if (event.origin == "http://www.wrox.com"){

        //处理接收到的数据
        processMessage(event.data);

        //可选：向来源窗口发送回执
        event.source.postMessage("Received!", "http://p2p.wrox.com");
    }
});
```

还是要提醒大家，event.source 大多数情况下只是 window 对象的代理，并非实际的 window 对象。换句话说，不能通过这个代理对象访问 window 对象的其他任何信息。记住，只通过这个代理调用 postMessage()就好，这个方法永远存在，永远可以调用。

XDM 还有一些怪异之处。首先，postMessage()的第一个参数最早是作为“永远都是字符串”来实现的。但后来这个参数的定义改了，改成允许传入任何数据结构。可是，并非所有浏览器都实现了这一变化。为保险起见，使用 postMessage()时，最好还是只传字符串。如果你想传入结构化的数据，最佳选择是先在要传入的数据上调用 JSON.stringify()，通过 postMessage()传入得到的字符串，然后再在 onmessage 事件处理程序中调用 JSON.parse()。

在通过内嵌框架加载其他域的内容时，使用 XDM 是非常方便的。因此，在混搭（mashup）和社交网络应用中，这种传递消息的方法极为常用。有了 XDM，包含<iframe>的页面可以确保自身不受恶意内容的侵扰，因为它只通过 XDM 与嵌入的框架通信。而 XDM 也可以在来自相同域的页面间使用。

支持 XDM 的浏览器有 IE8+、Firefox 3.5+、Safari 4+、Opera、Chrome、iOS 版 Safari 及 Android 版 WebKit。XDM 已经作为一个规范独立出来，现在它的名字叫 Web Messaging，官方页面是 http://dev.w3.org/html5/postmsg/。

16.2 原生拖放

最早在网页中引入 JavaScript 拖放功能的是 IE4。当时，网页中只有两种对象可以拖放：图像和某些文本。拖动图像时，把鼠标放在图像上，按住鼠标不放就可以拖动它。拖动文本时，要先选中文本，然后可以像拖动图像一样拖动被选中的文本。在 IE 4 中，唯一有效的放置目标是文本框。到了 IE5，拖放功能得到扩展，添加了新的事件，而且几乎网页中的任何元素都可以作为放置目标。IE5.5 更进一步，让网页中的任何元素都可以拖放。（IE6 同样也支持这些功能。）HTML5 以 IE 的实例为基础制定了拖放规范。Firefox 3.5、Safari 3+和 Chrome 也根据 HTML5 规范实现了原生拖放功能。

说到拖放，最有意思的恐怕就是能够在框架间、窗口间，甚至在应用间拖放网页元素了。浏览器对拖放的支持为实现这些功能提供了便利。

16.2.1 拖放事件

通过拖放事件，可以控制拖放相关的各个方面。其中最关键的地方在于确定哪里发生了拖放事件，有些事件是在被拖动的元素上触发的，而有些事件是在放置目标上触发的。拖动某元素时，将依次触发下列事件：

(1) dragstart
(2) drag
(3) dragend

按下鼠标键并开始移动鼠标时，会在被拖放的元素上触发 dragstart 事件。此时光标变成“不能放”符号（圆环中有一条反斜线），表示不能把元素放到自己上面。拖动开始时，可以通过 ondragstart 事件处理程序来运行 JavaScript 代码。

触发 dragstart 事件后，随即会触发 drag 事件，而且在元素被拖动期间会持续触发该事件。这个事件与 mousemove 事件相似，在鼠标移动过程中，mousemove 事件也会持续发生。当拖动停止时（无论是把元素放到了有效的放置目标，还是放到了无效的放置目标上），会触发 dragend 事件。

上述三个事件的目标都是被拖动的元素。默认情况下，浏览器不会在拖动期间改变被拖动元素的外观，但你可以自己修改。不过，大多数浏览器会为正被拖动的元素创建一个半透明的副本，这个副本始终跟随着光标移动。

当某个元素被拖动到一个有效的放置目标上时，下列事件会依次发生：

(1) dragenter
(2) dragover
(3) dragleave 或 drop

只要有元素被拖动到放置目标上，就会触发 dragenter 事件（类似于 mouseover 事件）。紧随其后的是 dragover 事件，而且在被拖动的元素还在放置目标的范围内移动时，就会持续触发该事件。如果元素被拖出了放置目标，dragover 事件不再发生，但会触发 dragleave 事件（类似于 mouseout 事件）。如果元素被放到了放置目标中，则会触发 drop 事件而不是 dragleave 事件。上述三个事件的目标都是作为放置目标的元素。

16.2.2 自定义放置目标

在拖动元素经过某些无效放置目标时，可以看到一种特殊的光标（圆环中有一条反斜线），表示不能放置。虽然所有元素都支持放置目标事件，但这些元素默认是不允许放置的。如果拖动元素经过不允许放置的元素，无论用户如何操作，都不会发生 drop 事件。不过，你可以把任何元素变成有效的放置目标，方法是重写 dragenter 和 dragover 事件的默认行为。例如，假设有一个 ID 为"droptarget"的 元素，可以用如下代码将它变成一个放置目标。

```
var droptarget = document.getElementById("droptarget");

EventUtil.addHandler(droptarget, "dragover", function(event){
    EventUtil.preventDefault(event);
});

EventUtil.addHandler(droptarget, "dragenter", function(event){
    EventUtil.preventDefault(event);
});
```

以上代码执行后，你就会发现当拖动着元素移动到放置目标上时，光标变成了允许放置的符号。当然，释放鼠标也会触发 drop 事件。

在 Firefox 3.5+中，放置事件的默认行为是打开被放到放置目标上的 URL。换句话说，如果是把图像拖放到放置目标上，页面就会转向图像文件；而如果是把文本拖放到放置目标上，则会导致无效 URL 错误。因此，为了让 Firefox 支持正常的拖放，还要取消 drop 事件的默认行为，阻止它打开 URL：

```
EventUtil.addHandler(droptarget, "drop", function(event){
    EventUtil.preventDefault(event);
});
```


16.2.3 dataTransfer 对象

只有简单的拖放而没有数据变化是没有什么用的。为了在拖放操作时实现数据交换，IE 5 引入了 dataTransfer 对象，它是事件对象的一个属性，用于从被拖动元素向放置目标传递字符串格式的数据。因为它是事件对象的属性，所以只能在拖放事件的事件处理程序中访问 dataTransfer 对象。在事件处理程序中，可以使用这个对象的属性和方法来完善拖放功能。目前，HTML5 规范草案也收入了 dataTransfer 对象。

dataTransfer 对象有两个主要方法：getData()和 setData()。不难想象，getData()可以取得由 setData()保存的值。setData()方法的第一个参数，也是 getData()方法唯一的一个参数，是一个字符串，表示保存的数据类型，取值为"text"或"URL"，如下所示：

```
//设置和接收文本数据
event.dataTransfer.setData("text", "some text");
var text = event.dataTransfer.getData("text");

//设置和接收 URL
event.dataTransfer.setData("URL", "http://www.wrox.com/");
var url = event.dataTransfer.getData("URL");
```

IE 只定义了"text"和"URL"两种有效的数据类型，而 HTML5 则对此加以扩展，允许指定各种 MIME 类型。考虑到向后兼容，HTML5 也支持"text"和"URL"，但这两种类型会被映射为"text/plain"和"text/uri-list"。

实际上，dataTransfer 对象可以为每种 MIME 类型都保存一个值。换句话说，同时在这个对象中保存一段文本和一个 URL 不会有任何问题。不过，保存在 dataTransfer 对象中的数据只能在 drop 事件处理程序中读取。如果在 ondrop 处理程序中没有读到数据，那就是 dataTransfer 对象已经被销毁，数据也丢失了。

在拖动文本框中的文本时，浏览器会调用 setData()方法，将拖动的文本以"text"格式保存在 dataTransfer 对象中。类似地，在拖放链接或图像时，会调用 setData()方法并保存 URL。然后，在这些元素被拖放到放置目标时，就可以通过 getData()读到这些数据。当然，作为开发人员，你也可以在 dragstart 事件处理程序中调用 setData()，手工保存自己要传输的数据，以便将来使用。

将数据保存为文本和保存为 URL 是有区别的。如果将数据保存为文本格式，那么数据不会得到任何特殊处理。而如果将数据保存为 URL，浏览器会将其当成网页中的链接。换句话说，如果你把它放置到另一个浏览器窗口中，浏览器就会打开该 URL。

Firefox 在其第 5 个版本之前不能正确地将"url"和"text"映射为"text/uri-list"和"text/plain"。但是却能把"Text"（T 大写）映射为"text/plain"。为了更好地在跨浏览器的情况

下从 dataTransfer 对象取得数据，最好在取得 URL 数据时检测两个值，而在取得文本数据时使用 "Text"。

```
var dataTransfer = event.dataTransfer;

//读取 URL
var url = dataTransfer.getData("url") ||dataTransfer.getData("text/uri-list");

//读取文本
var text = dataTransfer.getData("Text");
```

DataTransferExample01.htm

注意，一定要把短数据类型放在前面，因为 IE 10 及之前的版本仍然不支持扩展的 MIME 类型名，而它们在遇到无法识别的数据类型时，会抛出错误。

16.2.4 dropEffect 与 effectAllowed

利用 dataTransfer 对象，可不光是能够传输数据，还能通过它来确定被拖动的元素以及作为放置目标的元素能够接收什么操作。为此，需要访问 dataTransfer 对象的两个属性：dropEffect 和 effectAllowed。

其中，通过 dropEffect 属性可以知道被拖动的元素能够执行哪种放置行为。这个属性有下列 4 个可能的值。

- "none"：不能把拖动的元素放在这里。这是除文本框之外所有元素的默认值。
- "move"：应该把拖动的元素移动到放置目标。
- "copy"：应该把拖动的元素复制到放置目标。
- "link"：表示放置目标会打开拖动的元素（但拖动的元素必须是一个链接，有 URL）。

在把元素拖动到放置目标上时，以上每一个值都会导致光标显示为不同的符号。然而，要怎样实现光标所指示的动作完全取决于你。换句话说，如果你不介入，没有什么会自动地移动、复制，也不会打开链接。总之，浏览器只能帮你改变光标的样式，而其他的都要靠你自己来实现。要使用 dropEffect 属性，必须在 ondragenter 事件处理程序中针对放置目标来设置它。

dropEffect 属性只有搭配 effectAllowed 属性才有用。effectAllowed 属性表示允许拖动元素的哪种 dropEffect，effectAllowed 属性可能的值如下。

- "uninitialized"：没有给被拖动的元素设置任何放置行为。
- "none"：被拖动的元素不能有任何行为。
- "copy"：只允许值为"copy"的 dropEffect。
- "link"：只允许值为"link"的 dropEffect。
- "move"：只允许值为"move"的 dropEffect。
- "copyLink"：允许值为"copy"和"link"的 dropEffect。
- "copyMove"：允许值为"copy"和"move"的 dropEffect。
- "linkMove"：允许值为"link"和"move"的 dropEffect。
- "all"：允许任意 dropEffect。

必须在 ondragstart 事件处理程序中设置 effectAllowed 属性。

假设你想允许用户把文本框中的文本拖放到一个 元素中。首先，必须将 dropEffect 和 effectAllowed 设置为"move"。但是，由于 元素的放置事件的默认行为是什么也不做，所以文本不可能自动移动。重写这个默认行为，就能从文本框中移走文本。然后你就可以自己编写代码将文本插入到 中，这样整个拖放操作就完成了。如果你将 dropEffect 和 effectAllowed 的值设置为"copy"，那就不会自动移走文本框中的文本。

Firefox 5 及之前的版本在处理 effectAllowed 属性时有一个问题，即如果你在代码中设置了这个属性的值，那不一定会触发 drop 事件。

16.2.5 可拖动

默认情况下，图像、链接和文本是可以拖动的，也就是说，不用额外编写代码，用户就可以拖动它们。文本只有在被选中的情况下才能拖动，而图像和链接在任何时候都可以拖动。

让其他元素可以拖动也是可能的。HTML5 为所有 HTML 元素规定了一个 draggable 属性，表示元素是否可以拖动。图像和链接的 draggable 属性自动被设置成了 true，而其他元素这个属性的默认值都是 false。要想让其他元素可拖动，或者让图像或链接不能拖动，都可以设置这个属性。例如：

```
<!-- 让这个图像不可以拖动 -->
<img src="smile.gif" draggable="false" alt="Smiley face">

<!-- 让这个元素可以拖动 -->
<div draggable="true">...</div>
```

支持 draggable 属性的浏览器有 IE 10+、Firefox 4+、Safari 5+和 Chrome。Opera 11.5 及之前的版本都不支持 HTML5 的拖放功能。另外，为了让 Firefox 支持可拖动属性，还必须添加一个 ondragstart 事件处理程序，并在 dataTransfer 对象中保存一些信息。

在 IE9 及更早版本中，通过 mousedown 事件处理程序调用 **dragDrop()**能够让任何元素可拖动。而在 Safari 4 及之前版本中，必须额外给相应元素设置 CSS 样式 -khtml-user-drag: element。

16.2.6 其他成员

HTML5 规范规定 dataTransfer 对象还应该包含下列方法和属性。

- addElement(*element*)：为拖动操作添加一个元素。添加这个元素只影响数据（即增加作为拖动源而响应回调的对象），不会影响拖动操作时页面元素的外观。在写作本书时，只有 Firefox 3.5+ 实现了这个方法。
- clearData(*format*)：清除以特定格式保存的数据。实现这个方法的浏览器有 IE、Fireforx 3.5+、Chrome 和 Safari 4+。
- setDragImage(*element*, *x*, *y*)：指定一幅图像，当拖动发生时，显示在光标下方。这个方

法接收的三个参数分别是要显示的 HTML 元素和光标在图像中的 x、y 坐标。其中，HTML 元素可以是一幅图像，也可以是其他元素。是图像则显示图像，是其他元素则显示渲染后的元素。实现这个方法的浏览器有 Firefox 3.5+、Safari 4+和 Chrome。

- types：当前保存的数据类型。这是一个类似数组的集合，以"text"这样的字符串形式保存着数据类型。实现这个属性的浏览器有 IE10+、Firefox 3.5+和 Chrome。

16.3 媒体元素

随着音频和视频在 Web 上的迅速流行，大多数提供富媒体内容的站点为了保证跨浏览器兼容性，不得不选择使用 Flash。HTML5 新增了两个与媒体相关的标签，让开发人员不必依赖任何插件就能在网页中嵌入跨浏览器的音频和视频内容。这两个标签就是<audio>和<video>。

这两个标签除了能让开发人员方便地嵌入媒体文件之外，都提供了用于实现常用功能的 JavaScript API，允许为媒体创建自定义的控件。这两个元素的用法如下。

```
<!-- 嵌入视频 -->
<video src="conference.mpg" id="myVideo">Video player not available.</video>

<!-- 嵌入音频 -->
<audio src="song.mp3" id="myAudio">Audio player not available.</audio>
```

使用这两个元素时，至少要在标签中包含 src 属性，指向要加载的媒体文件。还可以设置 width 和 height 属性以指定视频播放器的大小，而为 poster 属性指定图像的 URI 可以在加载视频内容期间显示一幅图像。另外，如果标签中有 controls 属性，则意味着浏览器应该显示 UI 控件，以便用户直接操作媒体。位于开始和结束标签之间的任何内容都将作为后备内容，在浏览器不支持这两个媒体元素的情况下显示。

因为并非所有浏览器都支持所有媒体格式，所以可以指定多个不同的媒体来源。为此，不用在标签中指定 src 属性，而是要像下面这样使用一或多个<source>元素。

```
<!-- 嵌入视频 -->
<video id="myVideo">
  <source src="conference.webm" type="video/webm; codecs='vp8, vorbis'">
  <source src="conference.ogv" type="video/ogg; codecs='theora, vorbis'">
  <source src="conference.mpg">
  Video player not available.
</video>

<!-- 嵌入音频 -->
<audio id="myAudio">
  <source src="song.ogg" type="audio/ogg">
  <source src="song.mp3" type="audio/mpeg">
  Audio player not available.
</audio>
```

关于视频和音频编解码器的内容超出了本书讨论的范围。作者在此只想告诉大家，不同的浏览器支持不同的编解码器，因此一般来说指定多种格式的媒体来源是必需的。支持这两个媒体元素的浏览器有 IE9+、Firefox 3.5+、Safari 4+、Opera 10.5+、Chrome、iOS 版 Safari 和 Android 版 WebKit。

16.3.1 属性

<video>和<audio>元素都提供了完善的 JavaScript 接口。下表列出了这两个元素共有的属性，通过这些属性可以知道媒体的当前状态。

属　性	数据类型	说　明
autoplay	布尔值	取得或设置autoplay标志
buffered	时间范围	表示已下载的缓冲的时间范围的对象
bufferedBytes	字节范围	表示已下载的缓冲的字节范围的对象
bufferingRate	整数	下载过程中每秒钟平均接收到的位数
bufferingThrottled	布尔值	表示浏览器是否对缓冲进行了节流
controls	布尔值	取得或设置controls属性，用于显示或隐藏浏览器内置的控件
currentLoop	整数	媒体文件已经循环的次数
currentSrc	字符串	当前播放的媒体文件的URL
currentTime	浮点数	已经播放的秒数
defaultPlaybackRate	浮点数	取得或设置默认的播放速度。默认值为1.0秒
duration	浮点数	媒体的总播放时间（秒数）
ended	布尔值	表示媒体文件是否播放完成
loop	布尔值	取得或设置媒体文件在播放完成后是否再从头开始播放
muted	布尔值	取得或设置媒体文件是否静音
networkState	整数	表示当前媒体的网络连接状态：0表示空，1表示正在加载，2表示正在加载元数据，3表示已经加载了第一帧，4表示加载完成
paused	布尔值	表示播放器是否暂停
playbackRate	浮点数	取得或设置当前的播放速度。用户可以改变这个值，让媒体播放速度变快或变慢，这与defaultPlaybackRate只能由开发人员修改的defaultPlaybackRate不同
played	时间范围	到目前为止已经播放的时间范围
readyState	整数	表示媒体是否已经就绪（可以播放了）。0表示数据不可用，1表示可以显示当前帧，2表示可以开始播放，3表示媒体可以从头到尾播放
seekable	时间范围	可以搜索的时间范围
seeking	布尔值	表示播放器是否正移动到媒体文件中的新位置
src	字符串	媒体文件的来源。任何时候都可以重写这个属性
start	浮点数	取得或设置媒体文件中开始播放的位置，以秒表示
totalBytes	整数	当前资源所需的总字节数
videoHeight	整数	返回视频（不一定是元素）的高度。只适用于<video>
videoWidth	整数	返回视频（不一定是元素）的宽度。只适用于<video>
volume	浮点数	取得或设置当前音量，值为0.0到1.0

其中很多属性也可以直接在<audio>和<video>元素中设置。

16.3.2 事件

除了大量属性之外，这两个媒体元素还可以触发很多事件。这些事件监控着不同的属性的变化，这些变化可能是媒体播放的结果，也可能是用户操作播放器的结果。下表列出了媒体元素相关的事件。

事　件	触发时机
abort	下载中断
canplay	可以播放时；readyState值为2
canplaythrough	播放可继续，而且应该不会中断；readyState值为3
canshowcurrentframe	当前帧已经下载完成；readyState值为1
dataunavailable	因为没有数据而不能播放；readyState值为0
durationchange	duration属性的值改变
emptied	网络连接关闭
empty	发生错误阻止了媒体下载
ended	媒体已播放到末尾，播放停止
error	下载期间发生网络错误
load	所有媒体已加载完成。这个事件可能会被废弃，建议使用canplaythrough
loadeddata	媒体的第一帧已加载完成
loadedmetadata	媒体的元数据已加载完成
loadstart	下载已开始
pause	播放已暂停
play	媒体已接收到指令开始播放
playing	媒体已实际开始播放
progress	正在下载
ratechange	播放媒体的速度改变
seeked	搜索结束
seeking	正移动到新位置
stalled	浏览器尝试下载，但未接收到数据
timeupdate	currentTime被以不合理或意外的方式更新
volumechange	volume属性值或muted属性值已改变
waiting	播放暂停，等待下载更多数据

这些事件之所以如此具体，就是为了让开发人员只使用少量 HTML 和 JavaScript（与创建 Flash 影片相比）即可编写出自定义的音频/视频播放器。

16.3.3 自定义媒体播放器

使用<audio>和<video>元素的 play()和 pause()方法，可以手工控制媒体文件的播放。组合使用属性、事件和这两个方法，很容易创建一个自定义的媒体播放器，如下面的例子所示。

```
<div class="mediaplayer">
    <div class="video">
        <video id="player" src="movie.mov" poster="mymovie.jpg"
               width="300" height="200">
            Video player not available.
        </video>
    </div>
    <div class="controls">
        <input type="button" value="Play" id="video-btn">
        <span id="curtime">0</span>/<span id="duration">0</span>
    </div>
</div>
```

VideoPlayerExample01.htm

以上基本的 HTML 再加上一些 JavaScript 就可以变成一个简单的视频播放器。以下就是 JavaScript 代码。

```
//取得元素的引用
var player = document.getElementById("player"),
    btn = document.getElementById("video-btn"),
    curtime = document.getElementById("curtime"),
    duration = document.getElementById("duration");

//更新播放时间
duration.innerHTML = player.duration;

//为按钮添加事件处理程序
EventUtil.addHandler(btn, "click", function(event){
    if (player.paused){
        player.play();
        btn.value = "Pause";
    } else {
        player.pause();
        btn.value = "Play";
    }
});

//定时更新当前时间
setInterval(function(){
    curtime.innerHTML = player.currentTime;
}, 250);
```

VideoPlayerExample01.htm

以上 JavaScript 代码给按钮添加了一个事件处理程序，单击它能让视频在暂停时播放，在播放时暂停。通过<video>元素的 load 事件处理程序，设置了加载完视频后显示播放时间。最后，设置了一个计时器，以更新当前显示的时间。你可以进一步扩展这个视频播放器，监听更多事件，利用更多属性。而同样的代码也可以用于<audio>元素，以创建自定义的音频播放器。

16.3.4 检测编解码器的支持情况

如前所述，并非所有浏览器都支持<video>和<audio>的所有编解码器，而这基本上就意味着你必须提供多个媒体来源。不过，也有一个 JavaScript API 能够检测浏览器是否支持某种格式和编解码器。

这两个媒体元素都有一个 canPlayType()方法，该方法接收一种格式/编解码器字符串，返回"probably"、"maybe"或""（空字符串）。空字符串是假值，因此可以像下面这样在 if 语句中使用 canPlayType()：

```
if (audio.canPlayType("audio/mpeg")){
    //进一步处理
}
```

而"probably"和"maybe"都是真值，因此在 if 语句的条件测试中可以转换成 true。

如果给 canPlayType()传入了一种 MIME 类型，则返回值很可能是"maybe"或空字符串。这是因为媒体文件本身只不过是音频或视频的一个容器，而真正决定文件能否播放的还是编码的格式。在同时传入 MIME 类型和编解码器的情况下，可能性就会增加，返回的字符串会变成"probably"。下面来看几个例子。

```
var audio = document.getElementById("audio-player");

//很可能"maybe"
if (audio.canPlayType("audio/mpeg")){
    //进一步处理
}

//可能是"probably"
if (audio.canPlayType("audio/ogg; codecs=\"vorbis\"")){
    //进一步处理
}
```

注意，编解码器必须用引号引起来才行。下表列出了已知的已得到支持的音频格式和编解码器。

音　频	字　符　串	支持的浏览器
AAC	audio/mp4; codecs="mp4a.40.2"	IE9+、Safari 4+、iOS版Safari
MP3	audio/mpeg	IE9+、Chrome
Vorbis	audio/ogg; codecs="vorbis"	Firefox 3.5+、Chrome、Opera 10.5+
WAV	audio/wav; codecs="1"	Firefox 3.5+、Opera 10.5+、Chrome

当然，也可以使用 canPlayType()来检测视频格式。下表列出了已知的已得到支持的音频格式和编解码器。

视　频	字　符　串	支持的浏览器
H.264	video/mp4; codecs="avc1.42E01E, mp4a.40.2"	IE9+、Safari 4+、iOS版Safari、Android版WebKit
Theora	video/ogg; codecs="theora"	Firefox 3.5+、Opera 10.5、Chrome
WebM	video/webm; codecs="vp8, vorbis"	Firefox 4+、Opera 10.6、Chrome

16.3.5　Audio 类型

<audio>元素还有一个原生的 JavaScript 构造函数 Audio，可以在任何时候播放音频。从同为 DOM 元素的角度看，Audio 与 Image 很相似，但 Audio 不用像 Image 那样必须插入到文档中。只要创建一

个新实例，并传入音频源文件即可。

```
var audio = new Audio("sound.mp3");
EventUtil.addHandler(audio, "canplaythrough", function(event){
    audio.play();
});
```

创建新的 Audio 实例即可开始下载指定的文件。下载完成后，调用 play()就可以播放音频。

在 iOS 中，调用 play()时会弹出一个对话框，得到用户的许可后才能播放声音。如果想在一段音频播放后再播放另一段音频，必须在 onfinish 事件处理程序中调用 play()方法。

16.4 历史状态管理

历史状态管理是现代 Web 应用开发中的一个难点。在现代 Web 应用中，用户的每次操作不一定会打开一个全新的页面，因此“后退”和“前进”按钮也就失去了作用，导致用户很难在不同状态间切换。要解决这个问题，首选使用 hashchange 事件（第 13 章曾讨论过）。HTML5 通过更新 history 对象为管理历史状态提供了方便。

通过 hashchange 事件，可以知道 URL 的参数什么时候发生了变化，即什么时候该有所反应。而通过状态管理 API，能够在不加载新页面的情况下改变浏览器的 URL。为此，需要使用 history.pushState()方法，该方法可以接收三个参数：状态对象、新状态的标题和可选的相对 URL。例如：

```
history.pushState({name:"Nicholas"}, "Nicholas' page", "nicholas.html");
```

执行 pushState()方法后，新的状态信息就会被加入历史状态栈，而浏览器地址栏也会变成新的相对 URL。但是，浏览器并不会真的向服务器发送请求，即使状态改变之后查询 location.href 也会返回与地址栏中相同的地址。另外，第二个参数目前还没有浏览器实现，因此完全可以只传入一个空字符串，或者一个短标题也可以。而第一个参数则应该尽可能提供初始化页面状态所需的各种信息。

因为 pushState()会创建新的历史状态，所以你会发现“后退”按钮也能使用了。按下“后退”按钮，会触发 window 对象的 popstate 事件[①]。popstate 事件的事件对象有一个 state 属性，这个属性就包含着当初以第一个参数传递给 pushState()的状态对象。

```
EventUtil.addHandler(window, "popstate", function(event){
    var state = event.state;
    if (state){    //第一个页面加载时 state 为空
        processState(state);
    }
});
```

得到这个状态对象后，必须把页面重置为状态对象中的数据表示的状态（因为浏览器不会自动为你做这些）。记住，浏览器加载的第一个页面没有状态，因此单击“后退”按钮返回浏览器加载的第一个页面时，event.state 值为 null。

要更新当前状态，可以调用 replaceState()，传入的参数与 pushState()的前两个参数相同。调用这个方法不会在历史状态栈中创建新状态，只会重写当前状态。

① popstate 事件发生后，事件对象中的状态对象（event.state）是当前状态。

```
history.replaceState({name:"Greg"}, "Greg's page");
```

支持 HTML5 历史状态管理的浏览器有 Firefox 4+、Safari 5+、Opera 11.5+和 Chrome。在 Safari 和 Chrome 中，传递给 pushState()或 replaceState()的状态对象中不能包含 DOM 元素。而 Firefox 支持在状态对象中包含 DOM 元素。Opera 还支持一个 history.state 属性，它返回当前状态的状态对象。

在使用 HTML5 的状态管理机制时，请确保使用 pushState()创造的每一个“假”URL，在 Web 服务器上都有一个真的、实际存在的 URL 与之对应。否则，单击“刷新”按钮会导致 404 错误。

16.5　小结

HTML5 除了定义了新的标记规则，还定义了一些 JavaScript API。这些 API 是为了让开发人员创建出更好的、能够与桌面应用媲美的用户界面而设计的。本章讨论了如下 API。

- 跨文档消息传递 API 能够让我们在不降低同源策略安全性的前提下，在来自不同域的文档间传递消息。
- 原生拖放功能让我们可以方便地指定某个元素可拖动，并在操作系统要放置时做出响应。还可以创建自定义的可拖动元素及放置目标。
- 新的媒体元素<audio>和<video>拥有自己的与音频和视频交互的 API。并非所有浏览器支持所有的媒体格式，因此应该使用 canPlayType()检查浏览器是否支持特定的格式。
- 历史状态管理让我们不必卸载当前页面即可修改浏览器的历史状态栈。有了这种机制，用户就可以通过“后退”和“前进”按钮在页面状态间切换，而这些状态完全由 JavaScript 进行控制。