```
    } else {
      output.innerHTML = "Not an image.";
    }
  } else {
    output.innerHTML = "Your browser doesn't support object URLs.";
  }
});
```

如果把对象 URL 直接放到<img>标签，就不需要把数据先读到 JavaScript 中了。<img>标签可以直接从相应的内存位置把数据读取到页面上。

使用完数据之后，最好能释放与之关联的内存。只要对象 URL 在使用中，就不能释放内存。如果想表明不再使用某个对象 URL，则可以把它传给 window.URL.revokeObjectURL()。页面卸载时，所有对象 URL 占用的内存都会被释放。不过，最好在不使用时就立即释放内存，以便尽可能保持页面占用最少资源。

20.4.6　读取拖放文件

组合使用 HTML5 拖放 API 与 File API 可以创建读取文件信息的有趣功能。在页面上创建放置目标后，可以从桌面上把文件拖动并放到放置目标。这样会像拖放图片或链接一样触发 drop 事件。被放置的文件可以通过事件的 event.dataTransfer.files 属性读到，这个属性保存着一组 File 对象，就像文本输入字段一样。

下面的例子会把拖放到页面放置目标上的文件信息打印出来：

```
let droptarget = document.getElementById("droptarget");
function handleEvent(event) {
  let info = "",
    output = document.getElementById("output"),
    files, i, len;
  event.preventDefault();

  if (event.type == "drop") {
    files = event.dataTransfer.files;
    i = 0;
    len = files.length;

    while (i < len) {
      info += `${files[i].name} (${files[i].type}, ${files[i].size} bytes)<br>`;
      i++;
    }

    output.innerHTML = info;
  }
}
droptarget.addEventListener("dragenter", handleEvent);
droptarget.addEventListener("dragover", handleEvent);
droptarget.addEventListener("drop", handleEvent);
```

与后面要介绍的拖放的例子一样，必须取消 dragenter、dragover 和 drop 的默认行为。在 drop 事件处理程序中，可以通过 event.dataTransfer.files 读到文件，此时可以获取文件的相关信息。

20.5 媒体元素

随着嵌入音频和视频元素在 Web 应用上的流行，大多数内容提供商会强迫使用 Flash 以便达到最佳的跨浏览器兼容性。HTML5 新增了两个与媒体相关的元素，即<audio>和<video>，从而为浏览器提供了嵌入音频和视频的统一解决方案。

这两个元素既支持 Web 开发者在页面中嵌入媒体文件，也支持 JavaScript 实现对媒体的自定义控制。以下是它们的用法：

```
<!-- 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>
```

每个元素至少要求有一个 src 属性，以表示要加载的媒体文件。我们也可以指定表示视频播放器大小的 width 和 height 属性，以及在视频加载期间显示图片 URI 的 poster 属性。另外，controls 属性如果存在，则表示浏览器应该显示播放界面，让用户可以直接控制媒体。开始和结束标签之间的内容是在媒体播放器不可用时显示的替代内容。

由于浏览器支持的媒体格式不同，因此可以指定多个不同的媒体源。为此，需要从元素中删除 src 属性，使用一个或多个<source>元素代替，如下面的例子所示：

```
<!-- 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>
```

讨论不同音频和视频的编解码器超出了本书范畴，但浏览器支持的编解码器确实可能有所不同，因此指定多个源文件通常是必需的。

20.5.1 属性

<video>和<audio>元素提供了稳健的 JavaScript 接口。这两个元素有很多共有属性，可以用于确定媒体的当前状态，如下表所示。

属　性	数据类型	说　明
autoplay	Boolean	取得或设置 autoplay 标签
buffered	TimeRanges	对象，表示已下载缓冲的时间范围
bufferedBytes	ByteRanges	对象，表示已下载缓冲的字节范围
bufferingRate	Integer	平均每秒下载的位数
bufferingThrottled	Boolean	表示缓冲是否被浏览器截流

（续）

属　性	数据类型	说　明
controls	Boolean	取得或设置 controls 属性，用于显示或隐藏浏览器内置控件
currentLoop	Integer	媒体已经播放的循环次数
currentSrc	String	当前播放媒体的 URL
currentTime	Float	已经播放的秒数
defaultPlaybackRate	Float	取得或设置默认回放速率。默认为 1.0 秒
duration	Float	媒体的总秒数
ended	Boolean	表示媒体是否播放完成
loop	Boolean	取得或设置媒体是否应该在播放完再循环开始
muted	Boolean	取得或设置媒体是否静音
networkState	Integer	表示媒体当前网络连接状态。0 表示空，1 表示加载中，2 表示加载元数据，3 表示加载了第一帧，4 表示加载完成
paused	Boolean	表示播放器是否暂停
playbackRate	Float	取得或设置当前播放速率。用户可能会让媒体播放快一些或慢一些。与 defaultPlaybackRate 不同，该属性会保持不变，除非开发者修改
played	TimeRanges	到目前为止已经播放的时间范围
readyState	Integer	表示媒体是否已经准备就绪。0 表示媒体不可用，1 表示可以显示当前帧，2 表示媒体可以开始播放，3 表示媒体可以从头播到尾
seekable	TimeRanges	可以跳转的时间范围
seeking	Boolean	表示播放器是否正移动到媒体文件的新位置
src	String	媒体文件源。可以在任何时候重写
start	Float	取得或设置媒体文件中的位置，以秒为单位，从该处开始播放
totalBytes	Integer	资源需要的字节总数（如果知道的话）
videoHeight	Integer	返回视频（不一定是元素）的高度。只适用于<video>
videoWidth	Integer	返回视频（不一定是元素）的宽度。只适用于<video>
volume	Float	取得或设置当前音量，值为 0.0 到 1.0

上述很多属性也可以在<audio>或<video>标签上设置。

20.5.2　事件

除了有很多属性，媒体元素还有很多事件。这些事件会监控由于媒体回放或用户交互导致的不同属性的变化。下表列出了这些事件。

事　件	何时触发
abort	下载被中断
canplay	回放可以开始，readyState 为 2
canplaythrough	回放可以继续，不应该中断，readState 为 3
canshowcurrentframe	已经下载当前帧，readyState 为 1

（续）

事　件	何时触发
dataunavailable	不能回放，因为没有数据，readyState 为 0
durationchange	duration 属性的值发生变化
emptied	网络连接关闭了
empty	发生了错误，阻止媒体下载
ended	媒体已经播放完一遍，且停止了
error	下载期间发生了网络错误
load	所有媒体已经下载完毕。这个事件已被废弃，使用 canplaythrough 代替
loadeddata	媒体的第一帧已经下载
loadedmetadata	媒体的元数据已经下载
loadstart	下载已经开始
pause	回放已经暂停
play	媒体已经收到开始播放的请求
playing	媒体已经实际开始播放了
progress	下载中
ratechange	媒体播放速率发生变化
seeked	跳转已结束
seeking	回放已移动到新位置
stalled	浏览器尝试下载，但尚未收到数据
timeupdate	currentTime 被非常规或意外地更改了
volumechange	volume 或 muted 属性值发生了变化
waiting	回放暂停，以下载更多数据

这些事件被设计得尽可能具体，以便 Web 开发者能够使用较少的 HTML 和 JavaScript 创建自定义的音频/视频播放器（而不是创建新 Flash 影片）。

20.5.3 自定义媒体播放器

使用<audio>和<video>的 play() 和 pause() 方法，可以手动控制媒体文件的播放。综合使用属性、事件和这些方法，可以方便地创建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>
```

通过使用 JavaScript 创建一个简单的视频播放器，上面这个基本的 HTML 就可以被激活了，如下所示：

```
// 取得元素的引用
let player = document.getElementById("player"),
  btn = document.getElementById("video-btn"),
  curtime = document.getElementById("curtime"),
  duration = document.getElementById("duration");
// 更新时长
duration.innerHTML = player.duration;

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

// 周期性更新当前时间
setInterval(() => {
  curtime.innerHTML = player.currentTime;
}, 250);
```

这里的 JavaScript 代码简单地为按钮添加了事件处理程序，可以根据当前状态播放和暂停视频。此外，还给<video>元素的 load 事件添加了事件处理程序，以便显示视频的时长。最后，重复的计时器用于更新当前时间。通过监听更多事件以及使用更多属性，可以进一步扩展这个自定义的视频播放器。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。

20.5.4　检测编解码器

如前所述，并不是所有浏览器都支持<video>和<audio>的所有编解码器，这通常意味着必须提供多个媒体源。为此，也有 JavaScript API 可以用来检测浏览器是否支持给定格式和编解码器。这两个媒体元素都有一个名为 canPlayType()的方法，该方法接收一个格式/编解码器字符串，返回一个字符串值："probably"、"maybe"或""（空字符串），其中空字符串就是假值，意味着可以在 if 语句中像这样使用 canPlayType()：

```
if (audio.canPlayType("audio/mpeg")) {
  // 执行某些操作
}
```

"probably"和"maybe"都是真值，在 if 语句的上下文中可以转型为 true。

在只给 canPlayType()提供一个 MIME 类型的情况下，最可能返回的值是"maybe"和空字符串。这是因为文件实际上只是一个包装音频和视频数据的容器，而真正决定文件是否可以播放的是编码。在同时提供 MIME 类型和编解码器的情况下，返回值的可能性会提高到"probably"。下面是几个例子：

```
let audio = document.getElementById("audio-player");
// 很可能是"maybe"
if (audio.canPlayType("audio/mpeg")) {
  // 执行某些操作
}
// 可能是"probably"
```

```
if (audio.canPlayType("audio/ogg; codecs=\"vorbis\"")) {
  // 执行某些操作
}
```

注意，编解码器必须放到引号中。同样，也可以在视频元素上使用 canPlayType()检测视频格式。

20.5.5 音频类型

<audio>元素还有一个名为 Audio 的原生 JavaScript 构造函数，支持在任何时候播放音频。Audio 类型与 Image 类似，都是 DOM 元素的对等体，只是不需插入文档即可工作。要通过 Audio 播放音频，只需创建一个新实例并传入音频源文件：

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

创建 Audio 的新实例就会开始下载指定的文件。下载完毕后，可以调用 play()来播放音频。

在 iOS 中调用 play()方法会弹出一个对话框，请求用户授权播放声音。为了连续播放，必须在 onfinish 事件处理程序中立即调用 play()。

20.6 原生拖放

IE4 最早在网页中为 JavaScript 引入了对拖放功能的支持。当时，网页中只有两样东西可以触发拖放：图片和文本。拖动图片就是简单地在图片上按住鼠标不放然后移动鼠标。而对于文本，必须先选中，然后再以同样的方式拖动。在 IE4 中，唯一有效的放置目标是文本框。IE5 扩展了拖放能力，添加了新的事件，让网页中几乎一切都可以成为放置目标。IE5.5 又进一步，允许几乎一切都可以拖动（IE6 也支持这个功能）。HTML5 在 IE 的拖放实现基础上标准化了拖放功能。所有主流浏览器都根据 HTML5 规范实现了原生的拖放。

关于拖放最有意思的可能就是可以跨窗格、跨浏览器容器，有时候甚至可以跨应用程序拖动元素。浏览器对拖放的支持可以让我们实现这些功能。

20.6.1 拖放事件

拖放事件几乎可以让开发者控制拖放操作的方方面面。关键的部分是确定每个事件是在哪里触发的。有的事件在被拖放元素上触发，有的事件则在放置目标上触发。在某个元素被拖动时，会（按顺序）触发以下事件：

(1) dragstart

(2) drag

(3) dragend

在按住鼠标键不放并开始移动鼠标的那一刻，被拖动元素上会触发 dragstart 事件。此时光标会变成非放置符号（圆环中间一条斜杠），表示元素不能放到自身上。拖动开始时，可以在 ondragstart 事件处理程序中通过 JavaScript 执行某些操作。

dragstart 事件触发后，只要目标还被拖动就会持续触发 drag 事件。这个事件类似于 mousemove，即随着鼠标移动而不断触发。当拖动停止时（把元素放到有效或无效的放置目标上），会触发 dragend 事件。

所有这 3 个事件的目标都是被拖动的元素。默认情况下，浏览器在拖动开始后不会改变被拖动元素的外观，因此是否改变外观由你来决定。不过，大多数浏览器此时会创建元素的一个半透明副本，始终跟随在光标下方。

在把元素拖动到一个有效的放置目标上时，会依次触发以下事件：

(1) dragenter

(2) dragover

(3) dragleave 或 drop

只要一把元素拖动到放置目标上，dragenter 事件（类似于 mouseover 事件）就会触发。dragenter 事件触发之后，会立即触发 dragover 事件，并且元素在放置目标范围内被拖动期间此事件会持续触发。当元素被拖动到放置目标之外，dragover 事件停止触发，dragleave 事件触发（类似于 mouseout 事件）。如果被拖动元素被放到了目标上，则会触发 drop 事件而不是 dragleave 事件。这些事件的目标是放置目标元素。

20.6.2　自定义放置目标

在把某个元素拖动到无效放置目标上时，会看到一个特殊光标（圆环中间一条斜杠）表示不能放下。即使所有元素都支持放置目标事件，这些元素默认也是不允许放置的。如果把元素拖动到不允许放置的目标上，无论用户动作是什么都不会触发 drop 事件。不过，通过覆盖 dragenter 和 dragover 事件的默认行为，可以把任何元素转换为有效的放置目标。例如，如果有一个 ID 为"droptarget"的 元素，那么可以使用以下代码把它转换成一个放置目标：

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

droptarget.addEventListener("dragover", (event) => {
  event.preventDefault();
});

droptarget.addEventListener("dragenter", (event) => {
  event.preventDefault();
});
```

执行上面的代码之后，把元素拖动到这个 上应该可以看到光标变成了允许放置的样子。另外，drop 事件也会触发。

在 Firefox 中，放置事件的默认行为是导航到放在放置目标上的 URL。这意味着把图片拖动到放置目标上会导致页面导航到图片文件，把文本拖动到放置目标上会导致无效 URL 错误。为阻止这个行为，在 Firefox 中必须取消 drop 事件的默认行为：

```
droptarget.addEventListener("drop", (event) => {
  event.preventDefault();
});
```

20.6.3　dataTransfer 对象

除非数据受影响，否则简单的拖放并没有实际意义。为实现拖动操作中的数据传输，IE5 在 event 对象上暴露了 dataTransfer 对象，用于从被拖动元素向放置目标传递字符串数据。因为这个对象是 event 的属性，所以在拖放事件的事件处理程序外部无法访问 dataTransfer。在事件处理程序内部，

可以使用这个对象的属性和方法实现拖放功能。dataTransfer 对象现在已经纳入了 HTML5 工作草案。

dataTransfer 对象有两个主要方法：getData()和 setData()。顾名思义，getData()用于获取 setData()存储的值。setData()的第一个参数以及 getData()的唯一参数是一个字符串，表示要设置的数据类型："text"或"URL"，如下所示：

```
// 传递文本
event.dataTransfer.setData("text", "some text");
let text = event.dataTransfer.getData("text");

// 传递 URL
event.dataTransfer.setData("URL", "http://www.wrox.com/");
let url = event.dataTransfer.getData("URL");
```

虽然这两种数据类型是 IE 最初引入的，但 HTML5 已经将其扩展为允许任何 MIME 类型。为向后兼容，HTML5 还会继续支持"text"和"URL"，但它们会分别被映射到"text/plain"和"text/uri-list"。

dataTransfer 对象实际上可以包含每种 MIME 类型的一个值，也就是说可以同时保存文本和 URL，两者不会相互覆盖。存储在 dataTransfer 对象中的数据只能在放置事件中读取。如果没有在 ondrop 事件处理程序中取得这些数据，dataTransfer 对象就会被销毁，数据也会丢失。

在从文本框拖动文本时，浏览器会调用 setData()并将拖动的文本以"text"格式存储起来。类似地，在拖动链接或图片时，浏览器会调用 setData()并把 URL 存储起来。当数据被放置在目标上时，可以使用 getData()获取这些数据。当然，可以在 dragstart 事件中手动调用 setData()存储自定义数据，以便将来使用。

作为文本的数据和作为 URL 的数据有一个区别。当把数据作为文本存储时，数据不会被特殊对待。而当把数据作为 URL 存储时，数据会被作为网页中的一个链接，意味着如果把它放到另一个浏览器窗口，浏览器会导航到该 URL。

直到版本 5，Firefox 都不能正确地把"url"映射为"text/uri-list"或把"text"映射为"text/plain"。不过，它可以把"Text"（第一个字母大写）正确映射为"text/plain"。在通过 dataTransfer 获取数据时，为保持最大兼容性，需要对 URL 检测两个值并对文本使用"Text"：

```
let dataTransfer = event.dataTransfer;
// 读取 URL
let url = dataTransfer.getData("url") || dataTransfer.getData("text/uri-list");
// 读取文本
let text = dataTransfer.getData("Text");
```

这里要注意，首先应该尝试短数据名。这是因为直到版本 10，IE 都不支持扩展的类型名，而且会在遇到无法识别的类型名时抛出错误。

20.6.4 dropEffect 与 effectAllowed

dataTransfer 对象不仅可以用于实现简单的数据传输，还可以用于确定能够对被拖动元素和放置目标执行什么操作。为此，可以使用两个属性：dropEffect 与 effectAllowed。

dropEffect 属性可以告诉浏览器允许哪种放置行为。这个属性有以下 4 种可能的值。

- "none"：被拖动元素不能放到这里。这是除文本框之外所有元素的默认值。
- "move"：被拖动元素应该移动到放置目标。
- "copy"：被拖动元素应该复制到放置目标。

- "link"：表示放置目标会导航到被拖动元素（仅在它是 URL 的情况下）。

在把元素拖动到放置目标上时，上述每种值都会导致显示一种不同的光标。不过，是否导致光标示意的动作还要取决于开发者。换句话说，如果没有代码参与，则没有什么会自动移动、复制或链接。唯一不用考虑的就是光标自己会变。为了使用 dropEffect 属性，必须在放置目标的 ondragenter 事件处理程序中设置它。

除非同时设置 effectAllowed，否则 dropEffect 属性也没有用。effectAllowed 属性表示对被拖动元素是否允许 dropEffect。这个属性有如下几个可能的值。

- "uninitialized"：没有给被拖动元素设置动作。
- "none"：被拖动元素上没有允许的操作。
- "copy"：只允许"copy"这种 dropEffect。
- "link"：只允许"link"这种 dropEffect。
- "move"：只允许"move"这种 dropEffect。
- "copyLink"：允许"copy"和"link"两种 dropEffect。
- "copyMove"：允许"copy"和"move"两种 dropEffect。
- "linkMove"：允许"link"和"move"两种 dropEffect。
- "all"：允许所有 dropEffect。

必须在 ondragstart 事件处理程序中设置这个属性。

假设我们想允许用户把文本从一个文本框拖动到一个 元素。那么必须同时把 dropEffect 和 effectAllowed 属性设置为"move"。因为 元素上放置事件的默认行为是什么也不做，所以文本不会自动地移动自己。如果覆盖这个默认行为，文本就会自动从文本框中被移除。然后是否把文本插入 元素就取决于你了。如果是把 dropEffect 和 effectAllowed 属性设置为"copy"，那么文本框中的文本不会自动被移除。

20.6.5　可拖动能力

默认情况下，图片、链接和文本是可拖动的，这意味着无须额外代码用户便可以拖动它们。文本只有在被选中后才可以拖动，而图片和链接在任意时候都是可以拖动的。

我们也可以让其他元素变得可以拖动。HTML5 在所有 HTML 元素上规定了一个 draggable 属性，表示元素是否可以拖动。图片和链接的 draggable 属性自动被设置为 true，而其他所有元素此属性的默认值为 false。如果想让其他元素可拖动，或者不允许图片和链接被拖动，都可以设置这个属性。例如：

```
<!-- 禁止拖动图片 -->
<img src="smile.gif" draggable="false" alt="Smiley face">
<!-- 让元素可以拖动 -->
<div draggable="true">...</div>
```

20.6.6　其他成员

HTML5 规范还为 dataTransfer 对象定义了下列方法。

- addElement(*element*)：为拖动操作添加元素。这纯粹是为了传输数据，不会影响拖动操作的外观。在本书写作时，还没有浏览器实现这个方法。

- ❑ clearData(*format*)：清除以特定格式存储的数据。
- ❑ setDragImage(*element, x, y*)：允许指定拖动发生时显示在光标下面的图片。这个方法接收 3 个参数：要显示的 HTML 元素及标识光标位置的图片上的 *x* 和 *y* 坐标。这里的 HTML 元素可以是一张图片，此时显示图片；也可以是其他任何元素，此时显示渲染后的元素。
- ❑ types：当前存储的数据类型列表。这个集合类似数组，以字符串形式保存数据类型，比如"text"。

20.7 Notifications API

Notifications API 用于向用户显示通知。无论从哪个角度看，这里的通知都很类似 alert()对话框：都使用 JavaScript API 触发页面外部的浏览器行为，而且都允许页面处理用户与对话框或通知弹层的交互。不过，通知提供更灵活的自定义能力。

Notifications API 在 Service Worker 中非常有用。渐进 Web 应用（PWA，Progressive Web Application）通过触发通知可以在页面不活跃时向用户显示消息，看起来就像原生应用。

20.7.1 通知权限

Notifications API 有被滥用的可能，因此默认会开启两项安全措施：

- ❑ 通知只能在运行在安全上下文的代码中被触发；
- ❑ 通知必须按照每个源的原则明确得到用户允许。

用户授权显示通知是通过浏览器内部的一个对话框完成的。除非用户没有明确给出允许或拒绝的答复，否则这个权限请求对每个域只会出现一次。浏览器会记住用户的选择，如果被拒绝则无法重来。

页面可以使用全局对象 Notification 向用户请求通知权限。这个对象有一个 requestPemission()方法，该方法返回一个期约，用户在授权对话框上执行操作后这个期约会解决。

```
Notification.requestPermission()
    .then((permission) => {
      console.log('User responded to permission request:', permission);
    });
```

"granted"值意味着用户明确授权了显示通知的权限。除此之外的其他值意味着显示通知会静默失败。如果用户拒绝授权，这个值就是"denied"。一旦拒绝，就无法通过编程方式挽回，因为不可能再触发授权提示。

20.7.2 显示和隐藏通知

Notification 构造函数用于创建和显示通知。最简单的通知形式是只显示一个标题，这个标题内容可以作为第一个参数传给 Notification 构造函数。以下面这种方式调用 Notification，应该会立即显示通知：

```
new Notification('Title text!');
```

可以通过 options 参数对通知进行自定义，包括设置通知的主体、图片和振动等：

```
new Notification('Title text!', {
  body: 'Body text!',
  image: 'path/to/image.png',
  vibrate: true
});
```