<video width="320" height="240" controls="controls">

<source src="movie.ogg" type="video/ogg">

<source src="movie.mp4" type="video/mp4">

Your browser does not support the video tag.

</video>

controls 属性供添加播放、暂停和音量控件。

video 元素允许多个 source 元素。source 元素可以链接不同的视频文件。浏览器将使用第一个可识别的格式：

**<video> 标签的属性**

|  |  |  |
| --- | --- | --- |
| **属性** | **值** | **描述** |
| [autoplay](http://www.w3school.com.cn/tags/att_video_autoplay.asp) | autoplay | 如果出现该属性，则视频在就绪后马上播放。 |
| [controls](http://www.w3school.com.cn/tags/att_video_controls.asp) | controls | 如果出现该属性，则向用户显示控件，比如播放按钮。 |
| [height](http://www.w3school.com.cn/tags/att_video_height.asp) | *pixels* | 设置视频播放器的高度。 |
| [loop](http://www.w3school.com.cn/tags/att_video_loop.asp) | loop | 如果出现该属性，则当媒介文件完成播放后再次开始播放。 |
| [preload](http://www.w3school.com.cn/tags/att_video_preload.asp) | preload | 如果出现该属性，则视频在页面加载时进行加载，并预备播放。  如果使用 "autoplay"，则忽略该属性。 |
| [src](http://www.w3school.com.cn/tags/att_video_src.asp) | *url* | 要播放的视频的 URL。 |
| [width](http://www.w3school.com.cn/tags/att_video_width.asp) | *pixels* | 设置视频播放器的宽度。 |

## HTML5 <video> - 方法、属性以及事件

下面列出了大多数浏览器支持的视频方法、属性和事件：

|  |  |  |
| --- | --- | --- |
| **方法** | **属性** | **事件** |
| play() | currentSrc | play |
| pause() | currentTime | pause |
| load() | videoWidth | progress |
| canPlayType | videoHeight | error |
|  | duration | timeupdate |
|  | ended | ended |
|  | error | abort |
|  | paused | empty |
|  | muted | emptied |
|  | seeking | waiting |
|  | volume | loadedmetadata |
|  | height |  |
|  | width |  |

**注释：**在所有属性中，只有 videoWidth 和 videoHeight 属性是立即可用的。在视频的元数据已加载后，其他属性才可用。

如需在 HTML5 中播放音频

<audio controls="controls">

<source src="song.ogg" type="audio/ogg">

<source src="song.mp3" type="audio/mpeg">

Your browser does not support the audio tag.

</audio>

## <audio> 标签的属性

|  |  |  |
| --- | --- | --- |
| **属性** | **值** | **描述** |
| [autoplay](http://www.w3school.com.cn/tags/att_audio_autoplay.asp) | autoplay | 如果出现该属性，则音频在就绪后马上播放。 |
| [controls](http://www.w3school.com.cn/tags/att_audio_controls.asp) | controls | 如果出现该属性，则向用户显示控件，比如播放按钮。 |
| [loop](http://www.w3school.com.cn/tags/att_audio_loop.asp) | loop | 如果出现该属性，则每当音频结束时重新开始播放。 |
| [preload](http://www.w3school.com.cn/tags/att_audio_preload.asp) | preload | 如果出现该属性，则音频在页面加载时进行加载，并预备播放。  如果使用 "autoplay"，则忽略该属性。 |
| [src](http://www.w3school.com.cn/tags/att_audio_src.asp) | *url* | 要播放的音频的 URL。 |

## HTML5 拖放实例

下面的例子是一个简单的拖放实例：

### 实例

<!DOCTYPE HTML>

<html>

<head>

<script type="text/javascript">

function allowDrop(ev)

{

ev.preventDefault();

}

function drag(ev)

{

ev.dataTransfer.setData("Text",ev.target.id);

}

function drop(ev)

{

ev.preventDefault();

var data=ev.dataTransfer.getData("Text");

ev.target.appendChild(document.getElementById(data));

}

</script>

</head>

<body>

<div id="div1" ondrop="drop(event)"

ondragover="allowDrop(event)"></div>

<img id="drag1" src="img\_logo.gif" draggable="true"

ondragstart="drag(event)" width="336" height="69" />

</body>

</html>

## 设置元素为可拖放

首先，为了使元素可拖动，把 draggable 属性设置为 true ：

<img draggable="true" />

## 拖动什么 - ondragstart 和 setData()

然后，规定当元素被拖动时，会发生什么。

在上面的例子中，ondragstart 属性调用了一个函数，drag(*event*)，它规定了被拖动的数据。

dataTransfer.setData() 方法设置被拖数据的数据类型和值：

function drag(ev)

{

ev.dataTransfer.setData("Text",ev.target.id);

}

在这个例子中，数据类型是 "Text"，值是可拖动元素的 id ("drag1")。

## 放到何处 - ondragover

ondragover 事件规定在何处放置被拖动的数据。

默认地，无法将数据/元素放置到其他元素中。如果需要设置允许放置，我们必须阻止对元素的默认处理方式。

这要通过调用 ondragover 事件的 *event*.preventDefault() 方法：

event.preventDefault()

## 行放置 - ondrop

当放置被拖数据时，会发生 drop 事件。

在上面的例子中，ondrop 属性调用了一个函数，drop(*event*)：

function drop(ev)

{

ev.preventDefault();

var data=ev.dataTransfer.getData("Text");

ev.target.appendChild(document.getElementById(data));

}

### 代码解释：

* 调用 preventDefault() 来避免浏览器对数据的默认处理（drop 事件的默认行为是以链接形式打开）
* 通过 dataTransfer.getData("Text") 方法获得被拖的数据。该方法将返回在 setData() 方法中设置为相同类型的任何数据。
* 被拖数据是被拖元素的 id ("drag1")
* 把被拖元素追加到放置元素（目标元素）中

**canvas (画布)元素用于在网页上绘制图形。**

## 创建 Canvas 元素

向 HTML5 页面添加 canvas 元素。

规定元素的 id、宽度和高度：

<canvas id="myCanvas" width="200" height="100"></canvas>

## 通过 JavaScript 来绘制

canvas 元素本身是没有绘图能力的。所有的绘制工作必须在 JavaScript 内部完成：

<script type="text/javascript">

**var c=document.getElementById("myCanvas");**

**var cxt=c.getContext("2d");**

cxt.fillStyle="#FF0000";

cxt.fillRect(0,0,150,75);

</script>

JavaScript 使用 id 来寻找 canvas 元素：

var c=document.getElementById("myCanvas");

然后，创建 context 对象：

var cxt=c.getContext("2d");

getContext("2d") 对象是内建的 HTML5 对象，拥有多种绘制路径、矩形、圆形、字符以及添加图像的方法。

下面的两行代码绘制一个红色的矩形：

cxt.fillStyle="#FF0000";

cxt.fillRect(0,0,150,75);

fillStyle 方法将其染成红色，fillRect 方法规定了形状、位置和尺寸。

## 理解坐标

上面的 fillRect 方法拥有参数 (0,0,150,75)。

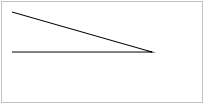
意思是：在画布上绘制 150x75 的矩形，从左上角开始 (0,0)。

## 更多 Canvas 实例

下面的在 canvas 元素上进行绘画的更多实例：

### 实例 - 线条

通过指定从何处开始，在何处结束，来绘制一条线：



JavaScript 代码：

<script type="text/javascript">

**var c=document.getElementById("myCanvas");**

**var cxt=c.getContext("2d");**

cxt.moveTo(10,10);

cxt.lineTo(150,50);

cxt.lineTo(10,50);

cxt.stroke();

</script>

canvas 元素：

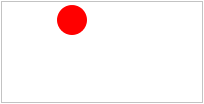
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #c3c3c3;">

Your browser does not support the canvas element.

</canvas>

### 实例 - 圆形

通过规定尺寸、颜色和位置，来绘制一个圆：



JavaScript 代码：

<script type="text/javascript">

**var c=document.getElementById("myCanvas");**

**var cxt=c.getContext("2d");**

cxt.fillStyle="#FF0000";

cxt.beginPath();

cxt.arc(70,18,15,0,Math.PI\*2,true);

cxt.closePath();

cxt.fill();

</script>

canvas 元素：

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #c3c3c3;">

Your browser does not support the canvas element.

</canvas>

### 实例 - 渐变

使用您指定的颜色来绘制渐变背景：



JavaScript 代码：

<script type="text/javascript">

var c=document.getElementById("myCanvas");

var cxt=c.getContext("2d");

var grd=cxt.createLinearGradient(0,0,175,50);

grd.addColorStop(0,"#FF0000");

grd.addColorStop(1,"#00FF00");

cxt.fillStyle=grd;

cxt.fillRect(0,0,175,50);

</script>

canvas 元素：

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #c3c3c3;">

Your browser does not support the canvas element.

</canvas>

### 实例 - 图像

把一幅图像放置到画布上：



JavaScript 代码：

<script type="text/javascript">

var c=document.getElementById("myCanvas");

var cxt=c.getContext("2d");

var img=new Image()

img.src="flower.png"

cxt.drawImage(img,0,0);

</script>

canvas 元素：

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #c3c3c3;">

Your browser does not support the canvas element.

</canvas>

# HTML5 内联 SVG

## 把 SVG 直接嵌入 HTML 页面

在 HTML5 中，您能够将 SVG 元素直接嵌入 HTML 页面中：

### 实例

<!DOCTYPE html>

<html>

<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1" height="190">

<polygon points="100,10 40,180 190,60 10,60 160,180"

style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;" />

</svg>

</body>

</html>

# HTML5 地理定位

## HTML5 - 使用地理定位

请使用 getCurrentPosition() 方法来获得用户的位置。

下例是一个简单的地理定位实例，可返回用户位置的经度和纬度。

### 实例

<script>

var x=document.getElementById("demo");

function getLocation()

{

if (navigator.geolocation)

{

navigator.geolocation.getCurrentPosition(showPosition);

}

else{x.innerHTML="Geolocation is not supported by this browser.";}

}

function showPosition(position)

{

x.innerHTML="Latitude: " + position.coords.latitude +

"<br />Longitude: " + position.coords.longitude;

}

</script>

## 处理错误和拒绝

getCurrentPosition() 方法的第二个参数用于处理错误。它规定当获取用户位置失败时运行的函数：

### 实例

function showError(error)

{

switch(error.code)

{

case error.PERMISSION\_DENIED:

x.innerHTML="User denied the request for Geolocation."

break;

case error.POSITION\_UNAVAILABLE:

x.innerHTML="Location information is unavailable."

break;

case error.TIMEOUT:

x.innerHTML="The request to get user location timed out."

break;

case error.UNKNOWN\_ERROR:

x.innerHTML="An unknown error occurred."

break;

}

}

## 在地图中显示结果

如需在地图中显示结果，您需要访问可使用经纬度的地图服务，比如谷歌地图或百度地图：

### 实例

function showPosition(position)

{

var latlon=position.coords.latitude+","+position.coords.longitude;

var img\_url="http://maps.googleapis.com/maps/api/staticmap?center="

+latlon+"&zoom=14&size=400x300&sensor=false";

document.getElementById("mapholder").innerHTML="<img src='"+img\_url+"' />";

**getCurrentPosition() 方法 - 返回数据**

若成功，则 getCurrentPosition() 方法返回对象。始终会返回 latitude、longitude 以及 accuracy 属性。如果可用，则会返回其他下面的属性。

|  |  |
| --- | --- |
| **属性** | **描述** |
| coords.latitude | 十进制数的纬度 |
| coords.longitude | 十进制数的经度 |
| coords.accuracy | 位置精度 |
| coords.altitude | 海拔，海平面以上以米计 |
| coords.altitudeAccuracy | 位置的海拔精度 |
| coords.heading | 方向，从正北开始以度计 |
| coords.speed | 速度，以米/每秒计 |
| timestamp | 响应的日期/时间 |

## Geolocation 对象 - 其他有趣的方法

watchPosition() - 返回用户的当前位置，并继续返回用户移动时的更新位置（就像汽车上的 GPS）。

clearWatch() - 停止 watchPosition() 方法

下面的例子展示 watchPosition() 方法。您需要一台精确的 GPS 设备来测试该例（比如 iPhone）：

### 实例

<script>

var x=document.getElementById("demo");

function getLocation()

{

if (navigator.geolocation)

{

navigator.geolocation.watchPosition(showPosition);

}

else{x.innerHTML="Geolocation is not supported by this browser.";}

}

function showPosition(position)

{

x.innerHTML="Latitude: " + position.coords.latitude +

"<br />Longitude: " + position.coords.longitude;

}

</script>

# HTML 5 Web 存储

HTML5 提供了两种在客户端存储数据的新方法：

* localStorage - 没有时间限制的数据存储
* sessionStorage - 针对一个 session 的数据存储
* <script type="text/javascript">
* if (localStorage.pagecount)
* {
* localStorage.pagecount=Number(localStorage.pagecount) +1;
* }
* else
* {
* localStorage.pagecount=1;
* }
* document.write("Visits "+ localStorage.pagecount + " time(s).");
* </script>

## sessionStorage 方法

sessionStorage 方法针对一个 session 进行数据存储。当用户关闭浏览器窗口后，数据会被删除。

如何创建并访问一个 sessionStorage：

<script type="text/javascript">

if (sessionStorage.pagecount)

{

sessionStorage.pagecount=Number(sessionStorage.pagecount) +1;

}

else

{

sessionStorage.pagecount=1;

}

document.write("Visits "+sessionStorage.pagecount+" time(s) this session.");

</script>

## **使用 HTML5，通过创建 cache manifest 文件，可以轻松地创建 web 应用的离线版本。**

## Cache Manifest 基础

如需启用应用程序缓存，请在文档的 <html> 标签中包含 manifest 属性：

<!DOCTYPE HTML>

<html manifest="demo.appcache">

...

</html>

每个指定了 manifest 的页面在用户对其访问时都会被缓存。如果未指定 manifest 属性，则页面不会被缓存（除非在 manifest 文件中直接指定了该页面）。

manifest 文件的建议的文件扩展名是：".appcache"。

请注意，manifest 文件需要配置**正确的 MIME-type**，即 "text/cache-manifest"。必须在 web 服务器上进行配置。

## Manifest 文件

manifest 文件是简单的文本文件，它告知浏览器被缓存的内容（以及不缓存的内容）。

manifest 文件可分为三个部分：

* **CACHE MANIFEST** - 在此标题下列出的文件将在首次下载后进行缓存
* **NETWORK** - 在此标题下列出的文件需要与服务器的连接，且不会被缓存
* **FALLBACK** - 在此标题下列出的文件规定当页面无法访问时的回退页面（比如 404 页面）

### CACHE MANIFEST

第一行，CACHE MANIFEST，是必需的：

CACHE MANIFEST

/theme.css

/logo.gif

/main.js

上面的 manifest 文件列出了三个资源：一个 CSS 文件，一个 GIF 图像，以及一个 JavaScript 文件。当 manifest 文件加载后，浏览器会从网站的根目录下载这三个文件。然后，无论用户何时与因特网断开连接，这些资源依然是可用的。

### NETWORK

下面的 NETWORK 小节规定文件 "login.asp" 永远不会被缓存，且离线时是不可用的：

NETWORK:

login.asp

可以使用星号来指示所有其他资源/文件都需要因特网连接：

NETWORK:

\*

### FALLBACK

下面的 FALLBACK 小节规定如果无法建立因特网连接，则用 "offline.html" 替代 /html5/ 目录中的所有文件：

FALLBACK:

/html5/ /404.html

**注释：**第一个 URI 是资源，第二个是替补。

## 更新缓存

一旦应用被缓存，它就会保持缓存直到发生下列情况：

* 用户清空浏览器缓存
* manifest 文件被修改（参阅下面的提示）
* 由程序来更新应用缓存

### 实例 - 完整的 Manifest 文件

CACHE MANIFEST

# 2012-02-21 v1.0.0

/theme.css

/logo.gif

/main.js

NETWORK:

login.asp

FALLBACK:

/html5/ /404.html

**重要的提示：**以 "#" 开头的是注释行，但也可满足其他用途。应用的缓存会在其 manifest 文件更改时被更新。如果您编辑了一幅图片，或者修改了一个 JavaScript 函数，这些改变都不会被重新缓存。更新注释行中的日期和版本号是一种使浏览器重新缓存文件的办法。

## 关于应用程序缓存的注释

请留心缓存的内容。

一旦文件被缓存，则浏览器会继续展示已缓存的版本，即使您修改了服务器上的文件。为了确保浏览器更新缓存，您需要更新 manifest 文件。

**注释：**浏览器对缓存数据的容量限制可能不太一样（某些浏览器设置的限制是每个站点 5MB）。

**web worker 是运行在后台的 JavaScript，不会影响页面的性能。**

## 什么是 Web Worker？

当在 HTML 页面中执行脚本时，页面的状态是不可响应的，直到脚本已完成。

web worker 是运行在后台的 JavaScript，独立于其他脚本，不会影响页面的性能。您可以继续做任何愿意做的事情：点击、选取内容等等，而此时 web worker 在后台运行。

## 浏览器支持

所有主流浏览器均支持 web worker，除了 Internet Explorer。

## HTML5 Web Workers 实例

下面的例子创建了一个简单的 web worker，在后台计数：

计数： 4

启动 Worker 停止 Worker

[亲自试一试](http://www.w3school.com.cn/tiy/t.asp?f=html5_webworker)

## 检测 Web Worker 支持

在创建 web worker 之前，请检测用户的浏览器是否支持它：

if(typeof(Worker)!=="undefined")

{

// Yes! Web worker support!

// Some code.....

}

else

{

// Sorry! No Web Worker support..

}

## 创建 web worker 文件

现在，让我们在一个外部 JavaScript 中创建我们的 web worker。

在这里，我们创建了计数脚本。该脚本存储于 "demo\_workers.js" 文件中：

var i=0;

function timedCount()

{

i=i+1;

postMessage(i);

setTimeout("timedCount()",500);

}

timedCount();

以上代码中重要的部分是 **postMessage()** 方法 - 它用于向 HTML 页面传回一段消息。

**注释：**web worker 通常不用于如此简单的脚本，而是用于更耗费 CPU 资源的任务。

## 创建 Web Worker 对象

我们已经有了 web worker 文件，现在我们需要从 HTML 页面调用它。

下面的代码检测是否存在 worker，如果不存在，- 它会创建一个新的 web worker 对象，然后运行 "demo\_workers.js" 中的代码：

if(typeof(w)=="undefined")

{

w=new Worker("demo\_workers.js");

}

然后我们就可以从 web worker 发生和接收消息了。

向 web worker 添加一个 "onmessage" 事件监听器：

w.onmessage=function(event){

document.getElementById("result").innerHTML=event.data;

};

当 web worker 传递消息时，会执行事件监听器中的代码。event.data 中存有来自 event.data 的数据。

## 终止 Web Worker

当我们创建 web worker 对象后，它会继续监听消息（即使在外部脚本完成之后）直到其被终止为止。

如需终止 web worker，并释放浏览器/计算机资源，请使用 terminate() 方法：

w.terminate();

## 完整的 Web Worker 实例代码

我们已经看到了 .js 文件中的 Worker 代码。下面是 HTML 页面的代码：

### 实例

<!DOCTYPE html>

<html>

<body>

<p>Count numbers: <output id="result"></output></p>

<button onclick="startWorker()">Start Worker</button>

<button onclick="stopWorker()">Stop Worker</button>

<br /><br />

<script>

var w;

function startWorker()

{

if(typeof(Worker)!=="undefined")

{

if(typeof(w)=="undefined")

{

w=new Worker("demo\_workers.js");

}

w.onmessage = function (event) {

document.getElementById("result").innerHTML=event.data;

};

}

else

{

document.getElementById("result").innerHTML="Sorry, your browser

does not support Web Workers...";

}

}

function stopWorker()

{

w.terminate();

}

</script>

</body>

</html>

# HTML 5 服务器发送事件

* [HTML5 Web Workers](http://www.w3school.com.cn/html5/html_5_webworkers.asp)
* [HTML5 输入类型](http://www.w3school.com.cn/html5/html_5_form_input_types.asp)

**HTML5 服务器发送事件（server-sent event）允许网页获得来自服务器的更新。**

## Server-Sent 事件 - 单向消息传递

Server-Sent 事件指的是网页自动获取来自服务器的更新。

以前也可能做到这一点，前提是网页不得不询问是否有可用的更新。通过服务器发送事件，更新能够自动到达。

例子：Facebook/Twitter 更新、估价更新、新的博文、赛事结果等。

## 浏览器支持

所有主流浏览器均支持服务器发送事件，除了 Internet Explorer。

## 接收 Server-Sent 事件通知

EventSource 对象用于接收服务器发送事件通知：

### 实例

var source=new EventSource("demo\_sse.php");

source.onmessage=function(event)

{

document.getElementById("result").innerHTML+=event.data + "<br />";

};

[亲自试一试](http://www.w3school.com.cn/tiy/t.asp?f=html5_sse)

例子解释：

* 创建一个新的 EventSource 对象，然后规定发送更新的页面的 URL（本例中是 "demo\_sse.php"）
* 每接收到一次更新，就会发生 onmessage 事件
* 当 onmessage 事件发生时，把已接收的数据推入 id 为 "result" 的元素中

## 检测 Server-Sent 事件支持

在上面的 TIY 实例中，我们编写了一段额外的代码来检测服务器发送事件的浏览器支持情况：

if(typeof(EventSource)!=="undefined")

{

// Yes! Server-sent events support!

// Some code.....

}

else

{

// Sorry! No server-sent events support..

}

## 服务器端代码实例

为了让上面的例子可以运行，您还需要能够发送数据更新的服务器（比如 PHP 和 ASP）。

服务器端事件流的语法是非常简单的。把 "Content-Type" 报头设置为 "text/event-stream"。现在，您可以开始发送事件流了。

### PHP 代码 (demo\_sse.php)：

<?php

header('Content-Type: text/event-stream');

header('Cache-Control: no-cache');

$time = date('r');

echo "data: The server time is: {$time}\n\n";

flush();

?>

### ASP 代码 (VB) (demo\_sse.asp):

<%

Response.ContentType="text/event-stream"

Response.Expires=-1

Response.Write("data: " & now())

Response.Flush()

%>

代码解释：

* 把报头 "Content-Type" 设置为 "text/event-stream"
* 规定不对页面进行缓存
* 输出发送日期（始终以 "data: " 开头）
* 向网页刷新输出数据

## EventSource 对象

在上面的例子中，我们使用 onmessage 事件来获取消息。不过还可以使用其他事件：

|  |  |
| --- | --- |
| **事件** | **描述** |
| onopen | 当通往服务器的连接被打开 |
| onmessage | 当接收到消息 |
| onerror | 当错误发生 |