第四章 音频和视频

内容安排

- 4.1 HTML5 Audio和Video概述
- 4.2 使用HTML5 Audio和Video API
- 4.3 课后思考
- 4.4 小结

4.1 HTML5 Audio和Video概述

- 视频容器
- 音频和视频编解码器
- HTML5 Audio和Video的限制
- audio元素和video元素的浏览器支持情况

4.1 HTML5 Audio和Video概述

- 20世纪90年代初，第一个网站由Tim Berners-Lee创立，只有文本和超链接
- 90 年代中期，一图胜千言，开始与图片结合
- 90 年代后期，Flash开始风靡，进入多媒体时代

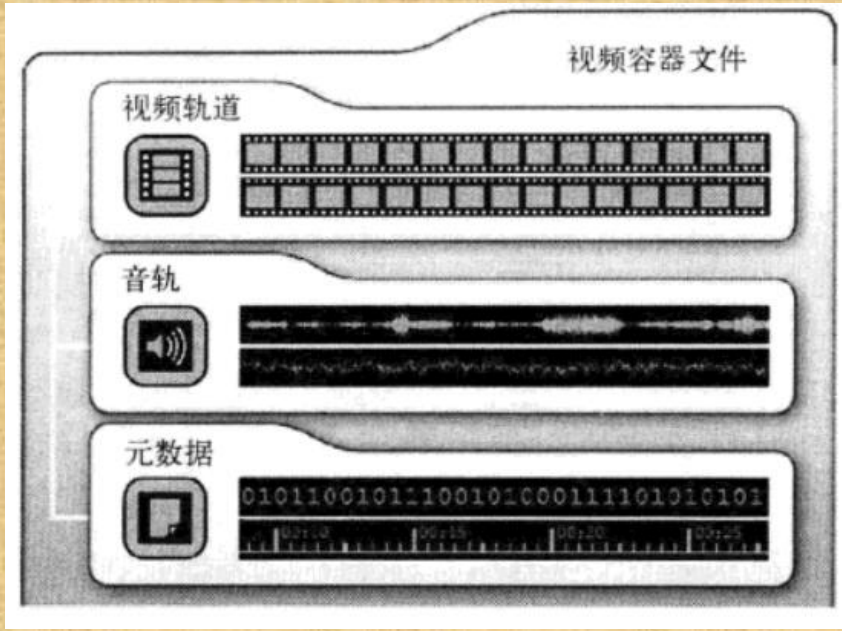
4.1 HTML5 Audio和Video概述

<audio>和<video>的出现让HTML5的媒体应用多了新选择，从而开发人员不必使用插件就能在网页中嵌入跨浏览器的音频和视频内容。

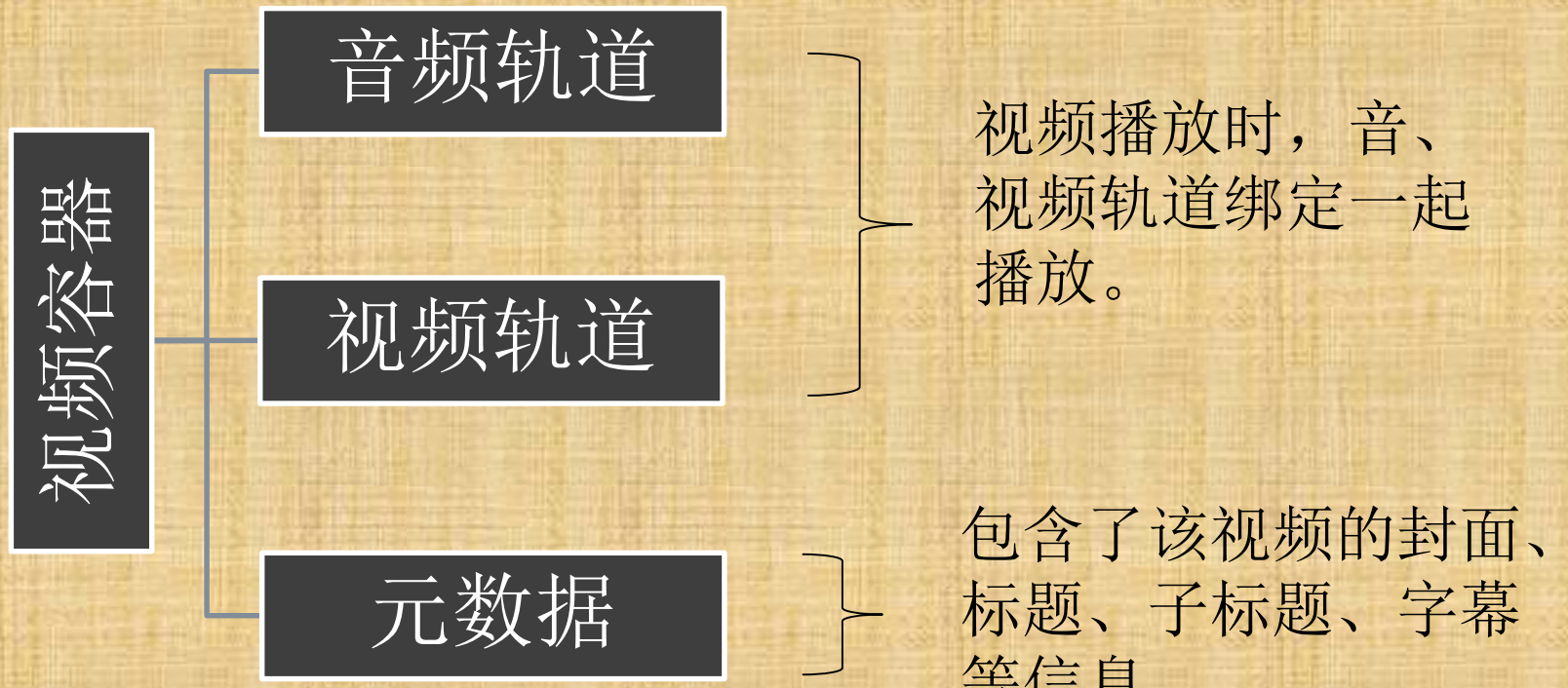
```
<!-- 嵌入视频 -->
<video src="视频资源">Video player not available</video>
<!-- 嵌入音频 -->
<audio src="音频资源">Audio player not available</audio>
```

视频容器（container）

不论音频还是视频文件，实际上都可以看成是一个容器文件。其结构如下图所示。



视频容器（container）



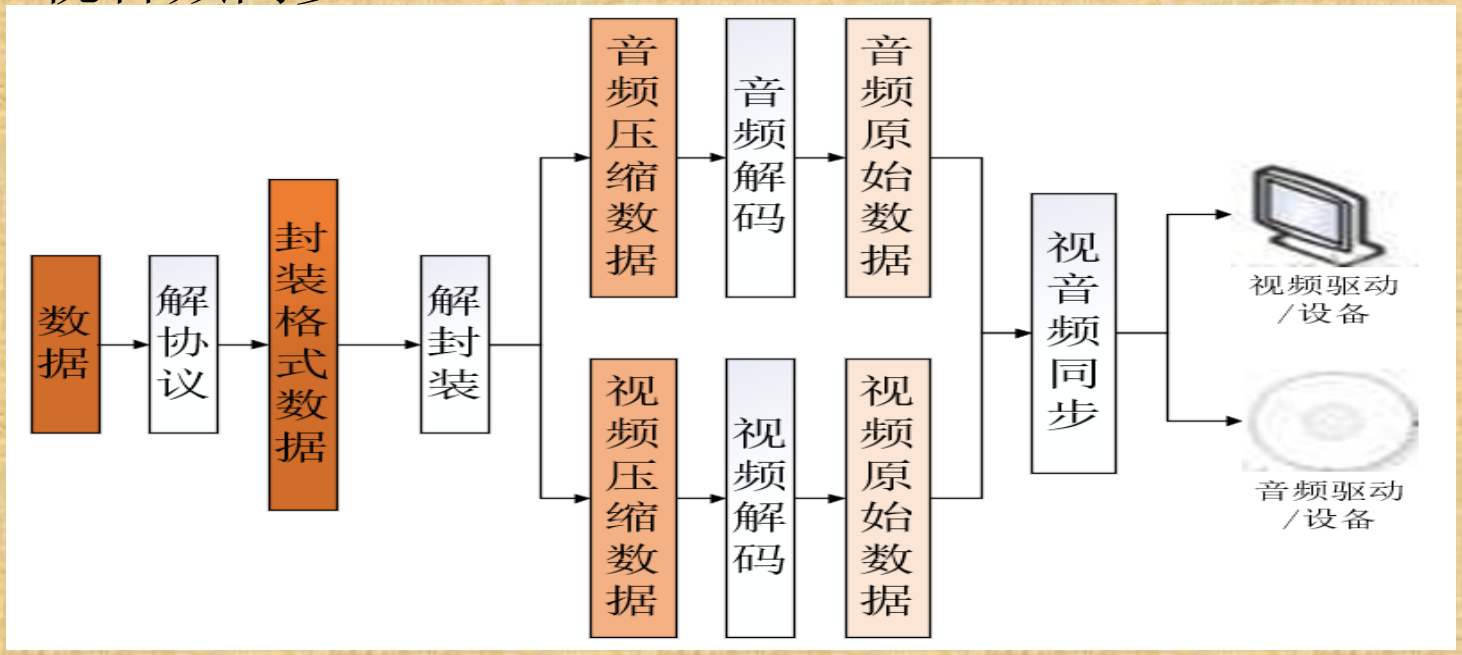
视频容器（container）

表 主流视频容器支持的视频格式

格式	文件	描述
Audio Video Interleave	.avi	微软开发。所有运行Windows的计算机都支持此格式。它是因特网上很常见的格式，但是非Window是计算机并不总是能够播放。
Flash Video	.flv	文件极小、加载速度极快
MPEG 4	.mp4	以存储数码音讯和数码视频为主，压缩比高，体积小，清晰度一般
Matroska	.mkv	能容纳多种不同编码的视频、音频及字幕流，同大小视频情况下最清晰
Ogg	.ogg	完全免费、开放和没有专利限制的

音频和视频编解码器（codec）

音频和视频的编码/解码器是一组算法，用来对一段特定音频或视频进行解码和编码，以便音频和视频能够播放。播放视频的基本流程是：解协议 → 解封装 → 解码 → 视音频同步。



音频和视频编解码器（codec）

原始的媒体文件体积非常大，如果不对其进行编码，数据就会非常庞大，从而在因特网上传播的时间也就无法忍受。如果没有解码器，则接受方就不能将数据重组为媒体数据进行播放。

主流音频编解码器：

- AAC
- MPEG-3
- Ogg Vorbis

主流视频编解码器：

- H.264
- VP8
- Ogg Theora

HTML5 Audio和Video的限制

有些功能是HTML5规范所不支持的:

- 流式音频和视频。HTML5视频规范中暂时没有比特率切换标准，所以对视频的支持只限于加载的全部媒体文件。
- HTML5的媒体受到HTTP跨源资源共享的限制。
- 全屏视频无法通过脚本控制。但浏览器可以提供其他控制手段。

audio元素和video元素的浏览器支持情况

<audio>和<video>现在已经被大多浏览器支持，例如Internet Explorer 9+、Firefox、Opera、Chrome和Safari。

ps:Internet Explorer 8及更早的IE版本并不支持<audio>和<video>元素。

当前<video>元素支持的三种视频格式：MP4，WebM和Ogg:

浏览器	版本	支持的格式
IE	6.0+	MP4
Chrome	6.0+	MP4、WebM、Ogg
Firefox	3.6+	WebM、Ogg
Safari	5.0+	MP4
Opera	10.0+	WebM、Ogg

- MP4=带有H.264视频编码和AAC音频编码的MPEG4文件
- WebM=带有VP8视频编码和Vorbis音频编码的WebM文件
- Ogg=带有Theora视频编码和Vorbis音频编码的Ogg文件

当前<audio>元素支持的三种视频格式：MP3，Wav和Ogg:

浏览器	版本	支持的格式
IE	9.0+	MP3
Chrome	6.0+	Ogg、MP3、Wav
Firefox	3.6+	Ogg、Wav
Safari	5.0+	MP3、Wav
Opera	10.0+	Ogg、Wav

大部分浏览器支持HTML5的<audio>标签，但并没有一种统一的音频格式。从支持的格式来看，要让所有浏览器可以播放<audio>元素上的音频，推荐使用MP3和Ogg两种格式（Wav接近无损音乐，文件相对较大）。

4.2 使用HTML5 Audio和Video API

- 浏览器支持的检测
- 可访问性
- 理解媒体元素
- 使用audio元素
- 使用video元素
- 进阶功能

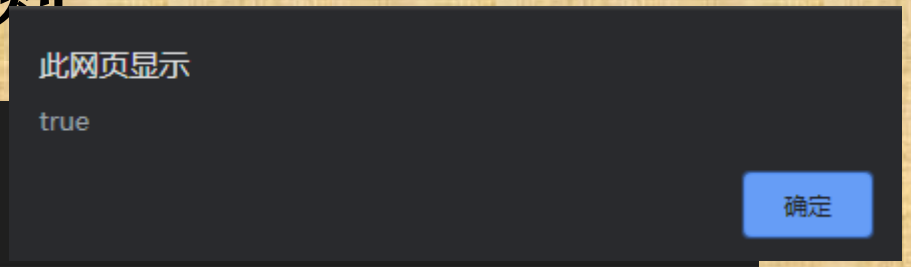
浏览器支持的检测

检测浏览器是否支持audio元素或video元素最简单的方式是用脚本动态创建它。

```
!!document.createElement('video').canPlayType
```

这段脚本会动态创建一个video元素，然后检查canPlayType()函数是否存在。通过“!!”运算符将结果转换成布尔值，就可以反应出视频对象是否已创建成功。

浏览器支持的检测




```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>检测浏览器对audio和vedio元素的支持</title>
</head>
<body>
    <script>
        // 对video进行检测
        alert(!!document.createElement('video').canPlayType);
        // 对audio进行检测
        // alert(!!document.createElement('audio').canPlayType);
    </script>
</body>
</html>
```

如果浏览器支持该元素，则会弹出一个为“true”的窗口。

浏览器支持的检测

如果检测结果是浏览器不支持（一般是因为浏览器版本太低），则需要使用其他方式来控制媒体，例如Flash等其他播放技术。

另外，可以在audio或video元素中放入备选内容。如果浏览器都不支持该元素，这些备选内容就会显示在元素对应的位置。一般情况下有两种备选内容：

浏览器支持的检测

（1）显示一条文本形式提示信息代替本应显示的内容。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>向网页中添加视频</title>
    </head>
    <body>
        <video src="video/movie.mp4" controls>
            <p>您的浏览器不支持此格式</p>
        </video>
    </body>
</html>
```

显示结果：

如果浏览器不支持则会显示<p>标签内容。

controls属性：规定浏览器为该视频或音频提供播放控件。

浏览器支持的检测

（2）以插件方式播放视频的代码作为备选内容。

```
<video src="video.webm" controls>
    <object data="videoplayer.swf" type="application/x-shockwave-flash">
        <param name="movie" value="video.swf" />
    </object>
    您的浏览器不支持HTML5 video 元素
</video>
```

在video元素中嵌入显示Flash视频的object元素之后，如果浏览器支持HTML5视频，那么HTML5视频会优先显示，Flash视频作为后备。

可访问性

我们设计网页时应该考虑到尽可能的满足各类人群都可以使用我们的Web程序。因此，对于视力或听力不好的用户，Web应用程序应该能呈现替代内容以满足用户的需求。

视频可访问性出现的标准时WebVTT，其使用简单二的文本文件（*.vtt），该文件第一行必须以单词WEBVTT开头。右图为vtt示例文件：

```
1  WEBVTT
2
3  1
4  00:00:15.000 --> 00:00:18.000 a:start
5  <v Proog>At the <i>left</i> we can see...</v>
6
7  2
8  00:00:18.167 --> 00:00:20.083
9  At the <i>right</i> we can <u>see</u> the...
```

可访问性

使用指向vtt文件的track元素来支持对话字幕：

```
<video src="elephants-dream-medium.webm" controls>
    <track label="English" kind="subtitles"
    srclang="en" src="english-subtitles.vtt" default>
    您的浏览器不支持HTML5 video元素
</video>
```

显示结果：

可访问性

表 track元素属性值

属性	描述
default	规定该轨道是默认的
kind	表示轨道属于什么文本类型
label	轨道的标签或标题
src	轨道的URL
srclang	轨道的语言

WebVTT标准支持的不仅仅是对话字幕，还能设计屏幕文字（caption）和线索（cue）等。详情参考：https://w3c.github.io/webvtt/。

理解媒体元素

HTML5中的audio元素和video元素有很多相同之处。两者都支持的操作有播放、暂停、静音/消除静音、加载等，因此通用的动作被从video和audio元素中剥离出来并放到媒体元素部分。

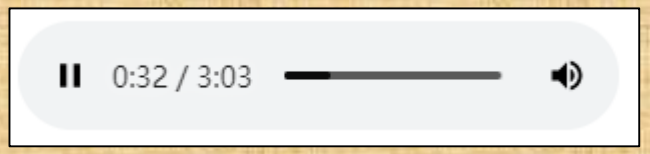
1、基本操作：声明媒体元素

简单的音频控制代码：

```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>HTML5 Audio </title>
  <audio controls src="johann_sebastian_bach_air.ogg">
    An audio clip from Johann Sebastian Bach.
  </audio>
</html>
```

理解媒体元素

上述段代码运行结果如下所示，可以看到一个带有播放按钮的播放控制条。



代码中的controls特性告诉浏览器显示通用的用户控件，包括开始、停止、跳播以及音量控制。如果不指定controls特性，用户将无法播放网页上的音频。

□理解媒体元素

2、使用source元素

上述代码中利用src特性直接指向媒体文件，但如果遇到浏览器不支持相关容器或编解码器（如Ogg和Vorbis），我们需要做好备用声明。

备用声明中包含**多种来源**，浏览器可以从中进行选择其支持的容器。代码示例如下：

理解媒体元素

```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>HTML5 Audio </title>
  <audio controls>
    <source src="johann_sebastian_bach_air.ogg">
    <source src="johann_sebastian_bach_air.mp3">
    An audio clip from Johann Sebastian Bach.
  </audio>
</html>
```

其中用两个**source**元素替代了src特性，这样可以让浏览器根据自身播放能力自动选择最佳播放来源。浏览器是按照声明顺序判断，如果支持多种格式，浏览器默认选择第一个来源播放。

理解媒体元素

此外有一种方式可以暗示浏览器播放哪种来源。即为source元素添加type特性：

```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>HTML5 Audio </title>
  <audio controls>
    <source src="johann_sebastian_bach_air.ogg" type="audio/ogg; codecs=vorbis">
    <source src="johann_sebastian_bach_air.mp3" type="audio/mpeg">
    An audio clip from Johann Sebastian Bach.
  </audio>
</html>
```

如果type特性中指定的类型与源文件不匹配，浏览器直接拒绝播放；如果省略了type特性，浏览器需要自己检测编码方式。因此添加type特性更为明智。

理解媒体元素

表 常用的媒体类型和特性值组合

类型	特性值
在Ogg容器中的Theora视频和Vorbis音频	type=“video/ogg;codecs=“theora,vorbis”
在Ogg容器中的Vorbis音频	type=“audio/ogg;codecs=“vorbis”
在Matroska容器中的WebM视频	type=“video/webm;codecs=“vp8,vorbis”
在MP4容器中的H.264视频和AAC音频	type=“video/mp4;codecs=“avc1.42E01E,mp4a.40.2”
在MP4容器中的MPEG-4简单视频和简单ACC音频	type=“video/mp4;codecs=“mp4v.20.8,mp4a.40.2”

理解媒体元素

3、媒体的控制

我们已经知道controls特性可以控制媒体的播放，如果不加这个特性，音频无法显示，视频则无法播放。此时可以在audio元素或video元素使用autoplay特性：

```
<audio autoplay>
  <source src="johann_sebastian_bach_air.ogg" type="audio/ogg; codecs=vorbis">
  <source src="johann_sebastian_bach_air.mp3" type="audio/mpeg">
  An audio clip from Johann Sebastian Bach.
</audio>
```

autoplay特性支持音频和视频加载完后自动播放。因此该特性主要应用于背景音乐来控制背景氛围，另一场景是强制用户接受广告。

理解媒体元素

有时内置的控件不适应用户界面的布局，我们可以借助一些内置的JavaScript函数和特性来控制视频或音频文件。具体示例见下一节。

表 常用的控制函数

函数	动作
load()	在播放前预加载资源文件
play()	加载并播放视频/音频文件
pause()	暂停处于播放的资源文件

当然，能控制媒体播放的函数和特性还有很多，深入学习可以参考https://www.w3school.com.cn/。

使用audio元素

利用上述audio元素所能提供的功能，则可以利用脚本开控制audio元素。

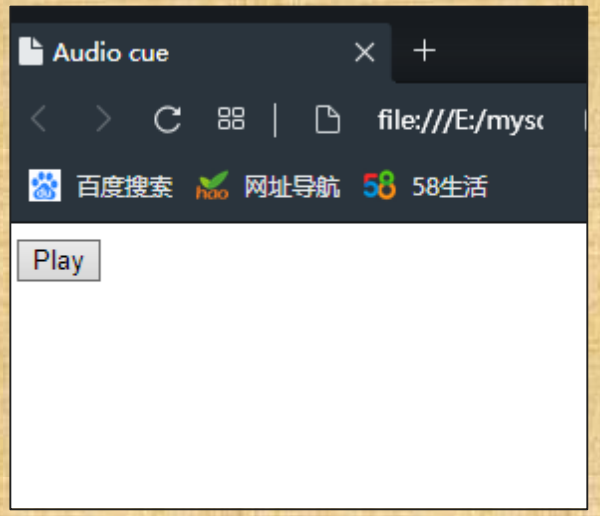
audio元素会为页面提供播放控件，在实际的开发过程中，这些控件与我们的UI设计的冲突时有发生。因此常常会将默认**控件隐藏**。

下面示例隐藏了用户控制界面和自动播放（即不设置controls特性或将其设置为false），而是利用功能按钮以脚本方式实现对音频的控制。

代码：

```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>Audio cue</title>

  <audio id="clickSound">
    <source src="johann_sebastian_bach_air.ogg">
    <source src="johann_sebastian_bach_air.mp3">
  </audio>

  <button id="toggle" onclick="toggleSound()">Play</button>

  <script type="text/javascript">
    function toggleSound() {
        var music = document.getElementById("clickSound");
        var toggle = document.getElementById("toggle");

        if (music.paused) {
          music.play();
          toggle.innerHTML = "Pause";
        }
        else {
          music.pause();
          toggle.innerHTML ="Play";
        }
    }
  </script>
</html>
```

运行结果：



```
<button id="toggle" onclick="toggleSound()">Play</button>
```

按钮初始化提示用户单击播放音频，每次单击都会出发toggleSound()函数。

```
var music = document.getElementById("clickSound");
var toggle = document.getElementById("toggle");
```

分别通过的id获取元素。

```
if (music.paused) {
  music.play();
  toggle.innerHTML = "Pause";
}
```

```
else {
  music.pause();
  toggle.innerHTML ="Play";
}
```

利用paused特性，可以检测用户是否已经暂停播放，如果未播放则调用play()函数，并修改按钮文字，再次点击则暂停，反之亦然。

使用video元素

HTML5 video元素同audio元素非常相似，但比audio元素多了一些特性。

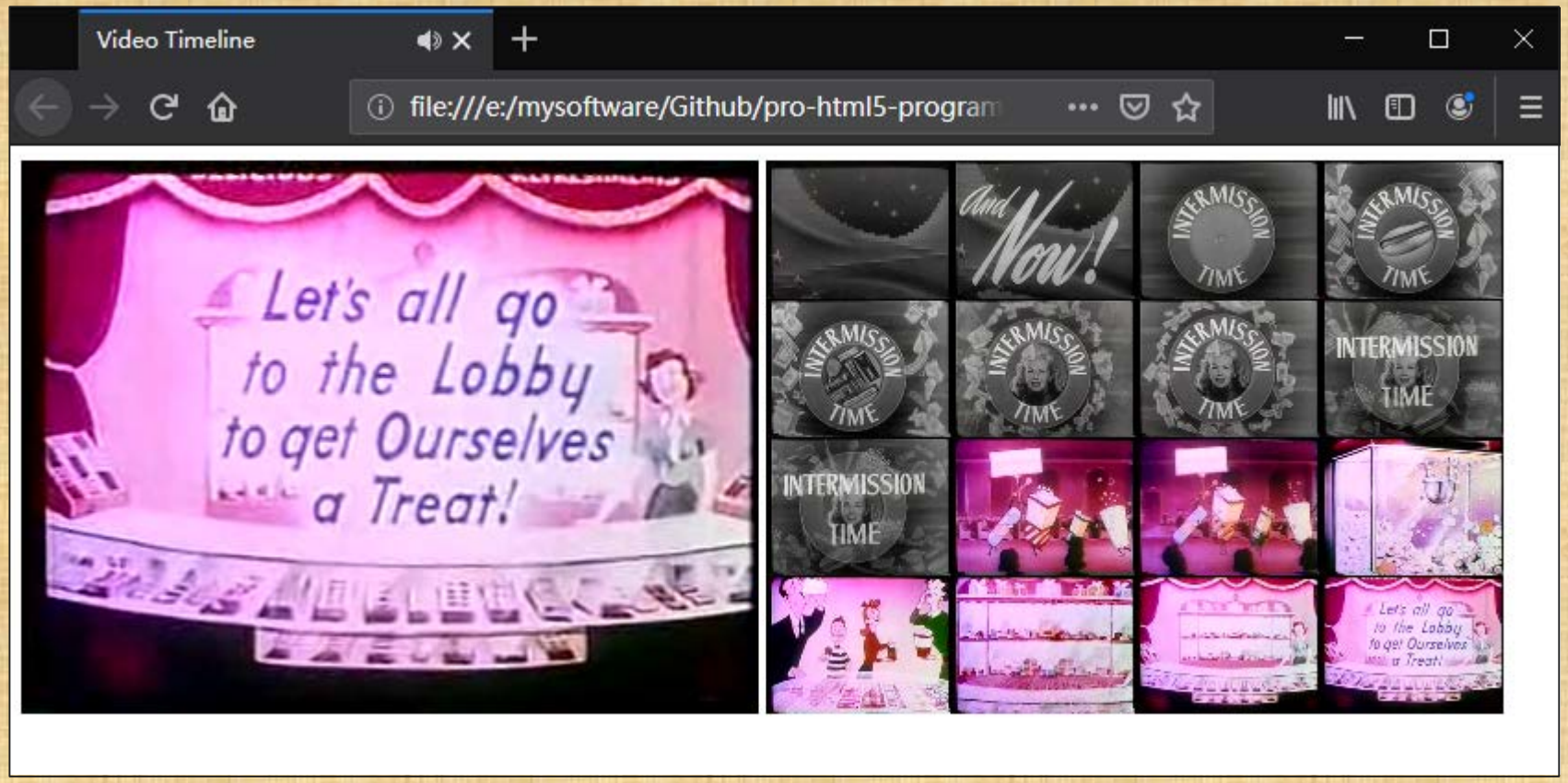
表 video元素的额外特性

特性	值
poster	视频加载完成之前，代表视频内容的图片的URL地址。
Width、height	读取或设置显示尺寸。
videoWidth、videoHeight	返回视频固有或自适应的宽度和高度。只读

*另外，video元素还有一个audio元素不支持的关键特性：可被HTML5 Canvas的函数调用。

使用video元素

抓取video元素中的帧并显示在动态canvas上示例效果预览：



使用video元素

1、添加一段代码来显示视频：

```
<video id="movies" controls oncanplay="startVideo()" onended="stopTimeline()"
       autobuffer="true" width="400px" height="300px" poster="timg.jpg" >
  <source src="Intermission-Walk-in.ogv" type='video/ogg;
          codecs="theora, vorbis"'>
  <source src="Intermission-Walk-in_512kb.mp4" type='video/mp4;
          codecs="avc1.42E01E, mp4a.40.2"'>
</video>
```

使用controls和poster特性，播放后会触发oncanplay函数来执行我们预设的动作，当视频播放结束后会触发onended函数以停止帧的创建。

使用video元素

2、创建canvas, 之后会以固定的时间间隔在上面绘制视频帧。

```
<canvas id="timeline" width="400px" height="300px">
```

3、添加变量

```
// 当前帧
var frameCount = 0;

// 播放完后取消计时器
var intervalId;

var videoStarted = false;
```

frameCount：跟中当前播放的帧。

intervalId：停止控制计时器。

videoStarted：保证只创建一个计时器。

使用video元素

```
//抓取帧的时间间隔，单位是ms
var updateInterval = 5000;

//时序中帧的尺寸
var frameWidth = 100;
var frameHeight = 75;

// 时序的总帧数
var frameRows = 4;
var frameColumns = 4;
var frameGrid = frameRows * frameColumns;
```

updateInterval：

用来控制抓取帧的频率。

frameWidth、frameHeight：

指定帧的大小。

frameGrid、frameRows和frameColumns：

决定在显示多少帧。

使用video元素

4、添加updateFrame函数

```
// 把帧绘制到canvas上
function updateFrame() {
    var video = document.getElementById("movies");
    var timeline = document.getElementById("timeline");

    var ctx = timeline.getContext("2d");

    // 根据帧数计算出当前播放位置
    // 然后以视频为输入参数
    // 绘制图像
    var framePosition = frameCount % frameGrid;
    var frameX = (framePosition % frameColumns) * frameWidth;
    var frameY = (Math.floor(framePosition / frameRows)) * frameHeight;
    ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY, frameWidth, frameHeight);

    frameCount++;
}
```

使用video元素

5、添加startVideo函数

```
function startVideo() {

    // 只在视频第一次播放时
    // 设置计时器
    if (videoStarted)
      return;

    videoStarted = true;

    // 计算初始帧，然后以规定时间间隔
    // 创建其他帧
    updateFrame();
    intervalId = setInterval(updateFrame, updateInterval);
```

视频播放时触发startVideo函数，并抓取第一帧，然后会启用间隔计时器(即5000ms)来定期调用updateFrame()函数。

6、处理用户输入

当用户单机时序查看器上的某一帧时，系统该怎么处理？

```
// 创建事件处理函数，用来在用户
// 单机某帧后定位视频
var timeline = document.getElementById("timeline");
timeline.onclick = function(evt) {
    var offX = evt.layerX - timeline.offsetLeft;
    var offY = evt.layerY - timeline.offsetTop;

    // 计算以零为基准索引的网络中
    // 哪帧被单机
    var clickedFrame = Math.floor(offY / frameHeight) * frameRows;
    clickedFrame += Math.floor(offX / frameWidth);

    // 视频开始后已经播放到多少帧
    var seekedFrame = (((Math.floor(frameCount / frameGrid)) *
                        frameGrid) + clickedFrame);

    // 如果用户单机的帧在当前帧之前
    // 则假定上上一轮的帧
    if (clickedFrame > (frameCount % 16))
        seekedFrame -= frameGrid;

    // 不允许跳出当前视频
    if (seekedFrame < 0)
      return;
```

使用video元素

7、添加stopTimeline函数

整个视频时序查看器示例中，设置视频播放完成时，停止帧的抓取。如果不添加此函数，示例会不停的抓取帧，最后会变成一片空白。

```
// 停止绘制时序的帧
function stopTimeline() {
    clearInterval(intervalId);
}
```

当视频播放完毕时会出发onended()函数，stopTimeline函数会在此时被调用。

□使用video元素　　完整代码：

```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>Video Timeline</title>

  <video id="movies" controls oncanplay="startVideo()" onended="stopTimeline()"
                autobuffer="true" width="400px" height="300px" poster="timg.jpg" >
    <source src="Intermission-Walk-in.ogv" type='video/ogg;
                codecs="theora, vorbis"'>
    <source src="Intermission-Walk-in_512kb.mp4" type='video/mp4;
                codecs="avc1.42E01E, mp4a.40.2"'>
  </video>

  <canvas id="timeline" width="400px" height="300px">

  <script type="text/javascript">

    //抓取帧的时间间隔，单位是ms
    var updateInterval = 5000;
```

```
//时序中帧的尺寸
var frameWidth = 100;
var frameHeight = 75;

// 时序的总帧数
var frameRows = 4;
var frameColumns = 4;
var frameGrid = frameRows * frameColumns;

// 当前帧
var frameCount = 0;

// 播放完后取消计时器
var intervalId;

var videoStarted = false;

function startVideo() {

    // 只在视频第一次播放时设置计时器
    if (videoStarted)
      return;
    videoStarted = true;
```

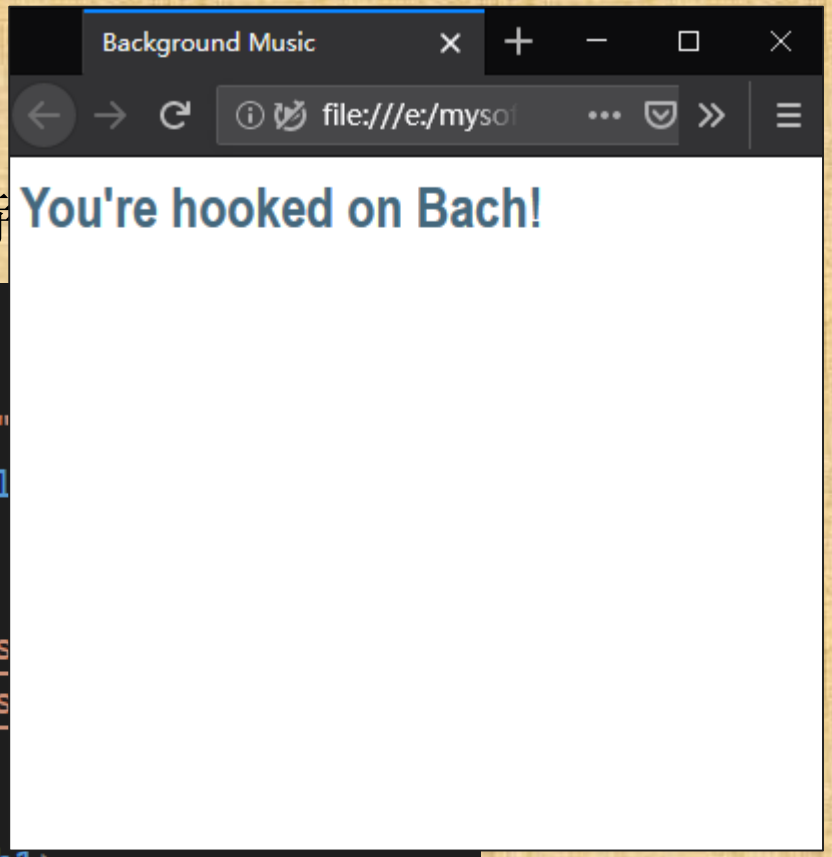
```
// 计算初始帧，然后以规定时间间隔创建其他帧
updateFrame();
intervalId = setInterval(updateFrame, updateInterval);

// 单机帧时设置处理器搜索视频
var timeline = document.getElementById("timeline");
timeline.onclick = function(evt) {
    var offX = evt.layerX - timeline.offsetLeft;
    var offY = evt.layerY - timeline.offsetTop;

    // 计算以零为基准索引的网络中哪帧被单击
    var clickedFrame = Math.floor(offY / frameHeight) * frameRows;
    clickedFrame += Math.floor(offX / frameWidth);

    // 视频开始后已经播放到多少帧
    var seekedFrame = (((Math.floor(frameCount / frameGrid)) *
                        frameGrid) + clickedFrame);

    // 如果用户单机的帧在当前帧之前则假定上上一轮的帧
    if (clickedFrame > (frameCount % 16))
        seekedFrame -= frameGrid;
```

```
        // 不允许跳出当前视频
        if (seekedFrame < 0)
          return;

        // 寻找到该帧的视频(以秒为单位)
        var video = document.getElementById("movies");
        video.currentTime = seekedFrame * updateInterval / 1000;

        // 然后将帧数设置为我们的目标
        frameCount = seekedFrame;
    }
}

// 把帧绘制到canvas上
function updateFrame() {
    var video = document.getElementById("movies");
    var timeline = document.getElementById("timeline");

    var ctx = timeline.getContext("2d");
```

```
        // 根据帧数计算出当前播放位置
        // 然后以视频为输入参数
        // 绘制图像
        var framePosition = frameCount % frameGrid;
        var frameX = (framePosition % frameColumns) * frameWidth;
        var frameY = (Math.floor(framePosition / frameRows)) * frameHeight;
        ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY, frameWidth, frameHeight);

        frameCount++;
    }

    // 停止绘制时序的帧
    function stopTimeline() {
        clearInterval(intervalId);
    }

</script>

</html>
```

进阶功能

（1）使用loop和autoplay特




```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="
  <title>Background Music</titl

  <audio autoplay loop>
      <source src="johann_sebas
      <source src="johann_sebas
      </audio>

  <h1>You're hooked on Bach!</h1>

</html>
```

很多网页为所有访问者自动播放音乐，即页面中的背景噪音，经常出现在广告页面中。

进阶功能

（2）鼠标悬停播放视频

```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>Mouseover Video</title>

  <video id="movies" onmouseover="this.play()" onmouseout="this.pause()" autobuffer="true"
    width="400px" height="300px">
    <source src="Intermission-Walk-in.ogv" type='video/ogg; codecs="theora, vorbis"'>
    <source src="Intermission-Walk-in_512kb.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  </video>

  <h1>Point at the video to play it!</h1>
</html>
```

通过视频上移动的鼠标来触发play和pause功能，即鼠标放在视频上播放，移开则暂停，单击视频则会播放完整视频。该功能主要应用与页面包含多个视频，且由用户来选择播放某个视频时（如直播等）。

4.3 课后思考

- 思考如果音视频没有编解码器会造成什么样的后果？
- 编解码器可以对视频容器中的元数据进行编码、解码吗？为什么？
- audio和video怎么引用网上的资源文件？
- audio和video还有哪些特性值？并分析其主要用在哪些场景。

4.4 小结

- HTML5引入了新元素video和audio, 这两个元素用于向网页添加视频和音频，但只有最新的浏览器兼容性好
- 支持HTML5元素的浏览器有很多种，它们所支持的视频和音频格式并不相同。因此需要提供不同格式的文件，以保证所有人都可以观看或收听。
- 最后我们通过编程方式使用API控制audio和video元素。并探讨了在实际中的应用。