Broadview
www.broadview.com.cn

王志刚作品系列

Canvas
CSS3
SVG
Video&Audio

拖放功能
脱机Web应用程序
WebSocket
Web Workers

Web Storage
Web SQL Database
Geolocation

HTML5 移动开发即学即用

·王志刚　王中元　江友华　编著·

HTML5已经广泛应用于各智能移动终端设备上，而且绝大部分技术已经被各种最新版本的浏览器所支持：

- 逐一剖析HTML5标准中包含的最新技术
- 详细介绍了HTML5新标准中提供的各种API
- 各种各样的应用实例，可以直接应用于自己的HTML5程序中。

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

内 容 简 介

HTML5 是取代 HTML4 的新一代 Web 技术，尽管正式标准还没有发布，但实际上已经被广泛应用于各智能移动终端设备上，而且绝大部分技术已经被各种最新版本的浏览器所支持。本书逐一剖析 HTML5 标准中包含的最新技术，全书分 11 章，详细介绍了 HTML5 新标准中提供的各种 API，同时附上了相关的应用实例，方便读者直接掌握这些 API 的使用，且大部分可以直接应用于自己的 HTML5 程序中。

前言

自从 1993 年互联网工程工作小组（IETF）发布了最初的 HTML 1.0 版本以来，期间经过了多次版本升级，现在广泛使用的最新正式版本是 1999 年 12 月发布的 HTML 4.01，至今已有 11 年了。

对于日新月异，分秒间都可能发生变化的 IT 世界来说，10 多年前可以说是很久以前了。尽管全世界的程序员开发出了各种各样的技术来扩展 HTML4 的功能，但这个“古老的”HTML4 标准越来越显示出其技术局限性，Web 世界呼唤一个崭新的标准来取代这个“古老的”HTML4 标准。新的 HTML5 标准可能于 2012 年正式公布。在这之前，在各种最新版的 Web 浏览器（如 FireFox 3.6 及以上、Opera10.5 及以上、Google Chrome 3 及以上、Internet Explorer 9、Safari 4 及以上）中已经支持大部分 HTML5 新标准中将要包括的内容了。尤其是在近年热卖的各种智能终端设备，如 iPhone、iPad、Android 手机等对 HTML5 的支持更迅速、更彻底，这是因为 HTML 5 在开发各种智能手机应用以及智能移动网站时具有巨大的优势。

本书内容

全书分 11 章，详细介绍了 HTML5 新标准中提供的各种 API，同时附上了相关的应用实例，方便读者直接掌握这些 API 的使用，且大部分可以直接应用于自己的 HTML5 程序中。

第 1 章是 HTML5 的概要，详细介绍了各种浏览器以及智能终端设备中对 HTML5 API 的支持现状。

第 2 章介绍 HTML5 新标准中最受关注的功能之一——图形/图像绘制技术 Canvas。尽管现在在 Canvas 中只能绘制 2D 图形/图像，但随着技术的发展将来完全可能取代 Flash 等技术。抛弃各种类似于 Flash 的插件正是 HTML5 新标准的目标之一。

第 3 章介绍 CSS 的最新版本 CSS3。编写 CSS3 样式单，能简单实现许多 CSS2 无法实现、或者实现起来很困难的效果，结合本书附录 C 中关于 CSS3 的动画实现方式的内容，读者可全面了解 CSS3 的具体应用。

第 4 章介绍在网页中绘制矢量图形的技术——SVG，帮助读者学习另一种不同于 Canvas 的图形绘制技术。

第 5 章介绍 HTML5 中另一种广受关注的 API，即 Video&Audio。在 HTML5 中只需要使用<video>/<audio>标签就可以实现视频/音频文件的播放，同时可以通过 JavaScript 脚本对其播放进行控制。

第 6 章介绍 HTML5 新标准提供的拖放功能。不仅可以轻松实现画面控件的拖放，还可以结合 File API 将桌面文件直接拖入到网页中，本章详细介绍了关于拖放功能的各种知识。

第 7 章介绍在 HTML5 中脱机实现 Web 应用程序的方式。脱机 Web 应用程序不仅允许用户在 Internet 环境中访问其网页，也可以在无法连接 Internet 时访问相应网页。

第 8 章介绍 HTML5 新标准中特色的双向通信（客户端与服务器）解决方案——

WebSocket。有了 WebSocket 技术，服务器端不再被动的接受客户端访问，还可以向客户端主动发送信息。

第 9 章介绍 HTML5 新标准中特色的多线程模式 Web Workers。使用 Web Workers 后，将画面中需要耗费大量时间运行的处理放在后台执行，前台画面不再像原来的网页一样，当进行耗时的处理时，画面只能“凝固”不动。

第 10 章介绍两种保存数据的 API——Web Storage 与 Web SQL Database。Web Storage 类似原来的 Cookie 与 Session，而使用 Web SQL Database 后，就可在 JavaScript 中直接操作数据库了。

第 11 章介绍在智能终端中受欢迎的定位功能 Geolocation API（当然普通网页中也能使用它），以及将 Geolocation API 与 Google Maps API 进行结合的具体应用。

本书阅读支持

本书可以作为 HTML5 开发的工具书以及 HTML5 入门学习读物，书中所有的源代码都可以从本书的支持网站（http://www.softechallenger.com）中下载。

武汉大学计算机学院王中元副教授以及上海电力学院计算机与信息工程学院的江友华副教授负责编写了本书中的部分内容。另外，朱蕾、罗伟、黄建峰、朱至濂参加了本书部分章节的审校及协助编写工作。在此特别感谢我父母在本书编写过程中的大力支持。

目　录

Video & Audio

在HTML5中可以很轻松的处理视频与音频，在HTML5中不仅可简单的播放视频/音频，而且可方便的进行变速播放，反向播放，等等。尤其是可将Canvas与视频进行结合，大大扩展显示效果。在HTML5中进行音频与视频的处理时，几乎可以使用相同的属性与方法，这是HTML5的一个优点之一。

5.1 <video> 与<audio>概要

本节介绍<video>与<audio>的产生背景以及<video>与<audio>的应用基础。

5.1.1 视频与音频处理革命

在没有HTML5以前，在页面中播放视频时必须通过Flash插件，HTML4中只能通过插件才能处理视频与音频。对于iPhone、Android等移动设备来说，并不支持Flash，通过Flash进行播放的视频网页将不能显示。自从移动设备的浏览器中支持HTML5的视频以及音频处理功能后，没有Flash也能够播放视频/音频。

HTML5中提供了<video>、<audio>这样的专用元素。首先我们看看HTML5中视频的处理方式。在HTML中显示视频时，在<video>标签中指定视频的URL。下面的例子就是显示名为sample.ogv的视频代码。

```
<video src="sample.ogv"></video>
```

现在只需要上述一行代码就可以播放视频sample.ogv了。当然还可以通过样式单自由的指定显示位置、视频屏幕尺寸大小等。另外，可以在同一页面内可以显示多个

视频，而且还可将这多个视频重叠显示。

我们还可以在视频上重叠字符串以及图像。这种重叠方式，可以轻松的实现与字幕的重叠显示。以前要追加字幕必须编辑视频，现在只要编辑HTML文档就可以了。而且在搜索引擎中也能搜索到字幕，将其翻译一下就可以实现多语言版本。本章后面将用具体的实例介绍实时字幕的实现方式。

HTML5中不仅可以播放视频，还可以将显示的视频送到Canvas中，进行绘制加工后再重新显示。这样可以实现效果更丰富的视频图像。通过此功能可编写简单的视频处理软件。

下面我们再看看音频。播放音频的实例代码如下。

```
<audio src="sample.mp3"></audio>
```

如果像上述一样能播放所有音频当然最理想了，但是实际上视频与音频都要涉及到文件形式以及encode形式的问题。这是因为不同的浏览器支持的视频/音频格式不同。因此，通常对同一视频或音频内容准备多种格式的文件，以兼容不同的浏览器。

● 知识专栏（Column）：视频格式与Theora/Ogg形式间的转换

业界有多种视频的数据格式。最古老的格式要算TGA、TIFF、以及JPEG连号系列图像文件等。这些都是拥有编号的静止图像文件，按帧数准备好。也并非是QuickTime或者AVI等容器。从互换性的角度来说，连号图像文件可放心的使用。

第二古老的是QuickTime。原来是在苹果电脑上使用的视频格式。后来移植到Windows上，在Windows上也支持此格式。QuickTime是放置视频的容器，不依赖特定的编解码器（Codec）。也就是说，QuickTime中实际容纳的视频可能是H.264或者MPEG2格式的视频。

只要追加了对应的编解码器，可支持任何格式的视频文件。

苹果电脑中有QuickTime，Windows电脑中也有此类型的AVI。这个与QuickTime一样也是容器，可支持各种格式的视频形式。

在个人计算机中，QuickTime以及AVI/Windows Media是主流，但是在摄像机等视频设备中则不太一样。现在使用最广泛的是AVCHD。AVCHD是注册商标，实际形式为H.264。也就是说，摄像机拍摄的视频格式为H.264，可以使用摄像机附属的编辑软件进行编辑。

H.264由于专利的关系，使用时需要支持专利许可（License）费。专利许可费随时可能涨价，随着规则的变化，H.264有可能不让使用。此时，出现了免费的编解

码器Theora。Theora是编解码器的名称，容器为Ogg。Ogg中不仅可容纳视频也可容纳音频数据。

Theora/Ogg没有搭载于摄像机中，与H.264不一样，它没有得到普及。可以进行上述两种格式间转换的软件也很少，能同时在MacOS X、UNIX、Windows中使用的只有ffmpeg2theora了。以下是其下载网址。

http://v2v.cc/~j/ffmpeg2theora/download.html

另外，可在Windows、UNIX运行的转换软件有OggConvert。下载地址如下。

http://oggconvert.tristanb.net/

ffmpeg2theora的使用方法比较简单，在命令行窗口下输入下述命令就可以了。

```
$ ffmpeg2theora movie30f.mov
$
```

如果想指定视频的品质（压缩率），需要附加-v的选项。另外，值的范围为0～10，值越大画面品质越高。当然画质越高数据容量也会越大。

```
$ ffmpeg2theora -v 值  视频文件路径
```


5.1.2 <video>与<audio>基础

上节中我们介绍过，使用<video>在HTML5中播放视频的基本语句是，将要播放的视频文件URL设置在其src属性中即可。

```
<video src="sample.mov"></video>
```

但是，在Firefox及Opera中不支持H.264编解码器，使用H.264编解码器的视频将不能显示。另外，如果将来出现新的编解码器，现在这样的方法必然行不通。HTML5是通过在<video>中追加<source>子元素的方式解决这种问题的。

浏览器会遍历<source>子元素，直至寻找到可播放的视频文件为止。也就是说，程序员只需为所有浏览器分别支持的视频格式文件追加一个<source>子元素，就可以实现在所有版本的浏览器中播放视频。<video>中还可追加当浏览器不支持<video>视频功能时显示的字符串，这一点与<object>元素的编写方式相同。

```
<video>
<source src="sample.mov">
```

```
<source src="sample.ogv">
<p>浏览器不支持<video>视频功能。</p>
</video>
```

<video>中定义了各种属性。例如当需要显示视频播放控制按钮时，其代码如下。

```
<video controls>
<source src="sample.mov">
<source src="sample.ogv">
<p>浏览器不支持<video>视频功能。</p>
</video>
```


在Firefox中视频数据不会被下载，可根据状况，指定autobuffer属性，使用数据缓冲功能。

```
<video autobuffer>
<source src="sample.mov">
<source src="sample.ogv">
<p>浏览器不支持<video>视频功能。</p>
</video>
```

另外，还可以通过width、height属性控制视频播放屏幕的尺寸。表5-1是<video>标签中定义的属性。

表5-1 <video>标签中定义的属性

属性	说明
src	视频数据的URL
poster	视频的缩略图
preload	自动缓冲（Firefox中为autobuffer）
autoplay	自动播放
loop	循环播放
controls	显示控制器
width	宽度
height	高度

<audio>与<video>一样，也定义了不少属性，但相对<video>标签来说，<audio>属性数量稍少。

如表5-2 所示为<audio>标签中定义的属性

属性	说明
src	视频数据的URL
preload	自动缓冲（Firefox中为autobuffer）
autoplay	自动播放
loop	循环播放
controls	显示控制器

5.2 Video和Audio的方法与属性

在HTML5中，视频与音频同属于媒体处理范畴。因此，其可使用的属性与方法是共同的。首先，就方法来讲，Video和Audio都有导入、播放、暂停、检查是否可播放等方法。主要方法以及属性如表5-3和表5-4所示。

表5-3 Video和Audio的主要方法

方法	说明
play()	播放处理
pause()	暂停处理
load()	导入视频/音频数据
canPlayType()	判断参数中指定的MIME类型的媒体文件是否可播放。返回空字符则不可播放。返回maybe则可播放

表5-4 Video和Audio的主要属性

属性	说明
currentTime	当前的播放时间
duration	视频/音频的时长
paused	是否暂停。暂停时为true
ended	是否播放到最后。播放到最后时值为true
playbackRate	播放速度。2时则表示2倍速。负数时逆向播放。可指定小数。另外，值太大时，视频有可能不能播放声音

下面看一个简单的例子，了解一下如何使用上述方法进行音乐播放的控制。下例中，点击相应按钮可以控制播放与暂停。如果想从头开始播放的话，先将currentTime属性值设为0，然后再调用play()方法。通过在currentTime属性中指定任意值，可以从

任意时刻开始音频/视频的播放，如图5-1所示。

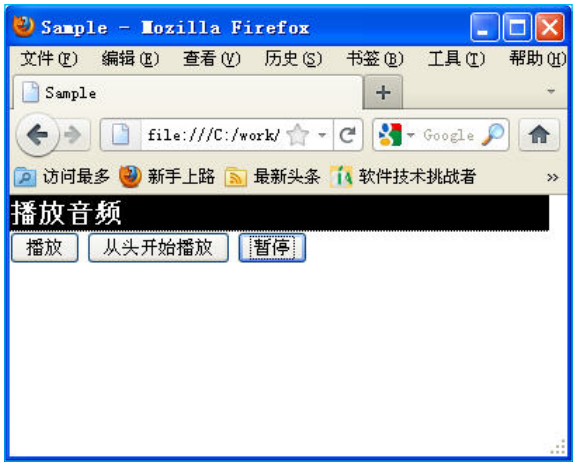

图5-1 控制音频播放

```
[HTML]01/index.html （可在Safari 4、Chrome 4 、Opera 10.5、Firefox3.6等以上版本中运行 ）
<!DOCTYPE html>
<html lang="zh">
   <head>
          <meta charset="utf-8">
          <title>Sample</title>
          <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
   </head>
   <body>
          <h1>播放音频</h1>
          <div id="contents">
                 <audio id="myAudio">
                        <source src="music/hydrancer.ogg">
                        <source src="music/hydrancer.mp4">
                        <p>请在支持audio标签的浏览器中播放</p>
                 </audio>
          </div>
```

```
        <button onclick="audioPlay()">播放</button>
        <button onclick="audioPlayFirst()">从头开始播放</button>
        <button onclick="audioPause()">暂停</button>
        <script>
            function audioPlay(){
                document.getElementById("myAudio").play();
            }
            function audioPlayFirst(){
                document.getElementById("myAudio").currentTime = 0;
                document.getElementById("myAudio").play();
            }
            function audioPause(){
                document.getElementById("myAudio").pause();
            }
        </script>
    </body>
</html>
```

● 知识专栏（Column）： Firefox 3.6的duration缺陷

在Firefox中，视频的时长刚好为15秒的情况下，也会显示14.965999603271484这样的时长。这是将1秒钟按照帧频进行分割，然后再乘以总帧数而换算得来的数值。极端的情况下，只存在一帧的视频在Firefox中不能播放，但在Chrome等浏览器中是可以播放的。

Firefox中duration的计算方式如下所示。

（1秒 ÷ 30fps）×（全帧数−1）

这里假设按15秒=450枚来计算。duration值如下。

14.96666666666667

Firefox中的计算也大致如此。

14.965999603271484

在 Firefox 4以后的版本中此缺陷已被修正。

5.3 事件以及事件的发生顺序

5.3.1 事件概要

Video与Audio中定义如表5-5所示的属性。到2011年9月为止，这些属性在各浏览器中的受支持情况各不相同。因此，此处只列举了现在受到稳定支持的属性。另外，数据在本地时与数据在服务器时，触发的事件有些不同。

表5-5 事件列表

事件	说明
loadstart	数据导入开始
progress	处理中
suspend	发生延迟
abort	数据传送中断
error	发生错误
emptied	数据为空
stalled	停止中（网络停止）
play	播放开始
pause	暂停
load	数据全部完成导入
loadedmetadata	meta数据完成导入
loadeddata	实体数据完成导入
waiting	待机中
playing	播放中
canplay	变成可播放状态时（但可能在播放途中中断）
canplaythrough	变成可一直播放到最后的状态（预测而非保证）
seeking	搜索中
timeupdate	搜索结束
ended	播放时间被更新
ratechange	播放速率发生变化
durationchange	播放时长发生变化
volumechange	音量发生变化

下面介绍一下在各浏览器中都可使用的主要事件。

● canplay

canplay事件在视频/音频数据被下载一定量后而达到可以播放的状态时发生。此事件在本地以及服务器中都可以响应。如果是自定义播放按钮，可捕捉此事件后再调用播放方法。

● play、playing

play事件在视频/音频将开始播放时发生，而playing事件在视频/音频已开始播放时发生。而播放中触发的事件并非playing，而是下面介绍的timeupdate。

● timeupadate

timeupadate事件在视频/音频播放后，播放时间更新时发生。timeupadate事件的发生时机随浏览器的不同而有所差异。在Firefox中，基本上是随着帧的变化而发生。而在Safari及Opera浏览器中1秒中会发生多次。实际测试的结果如表5-6所示。

表5-6 15秒中发生timeupadate事件次数。（包含450帧）

Firefox3.6	Chrome 5	Safari4	Opera 10.50
442	61	60	76

当将视频传送到Canvas中进行实时处理时，timeupdate事件并不实用。此时使用setTimeout()或者setInterval()，定时向Canvas传送视频的方式更好。特别是在Firefox中因为随着帧的变化会不断触发timeupdate事件，播放时消耗的内存资源会更多。

● ended

ended事件在视频播放结束时触发。另外如果在<video>中指定了loop属性，则ended事件不会发生。

5.3.2 事件的发生顺序

不同的浏览器中支持的事件不仅不同，而且发生顺序也有差异。详细差异已经在下面的系列表格（表5-7～表5-10）中列出。将来随着浏览器版本的升级有可能发生变化。另外，进行以下测试时没有设置autobuffer属性。

表5-7 视频导入时发生的事件及顺序（本地）

Firefox 3.6	Safari 4	Safari 5	Chrome 4
progress	loadstart	loadstart	loadstart
canplay	durationchange	durationchange	durationchange

续 表

Firefox 3.6	Safari 4	Safari 5	Chrome 4
canplaythrough	loadedmetadata	loadedmetadata	loadedmetadata
suspend	loadeddata	loadeddata	loadeddata
	canplay	progress	canplay
	canplaythrough	canplay	canplaythrough
	load	canplaythrough	progress
		load	load

表5-8 视频导入时发生的事件及顺序（服务器）

Firefox 3.6	Safari 4/5*1	Chrome 4/5	iPad*2
suspend	loadstart	loadstart	durationchange
canplay	durationchange	suspend	loadedmetadata
progress	loadedmetadata	durationchange	loadeddata
	loadeddata	loadedmetadata	canplay
	canplay	loadeddata	canplaythrough

表5-9 视频播放时发生的事件及顺序（本地）

Firefox 3.6	Safari 4/5*1	Chrome 4/5
play	play	play
playing	playing	playing
timeupdate	timeupdate	timeupdate
ended	ended	ended

表5-10 视频播放时发生的事件及顺序（服务器）

Firefox 3.6	Safari 4/5	Chrome 4/5	iPad*3
play	play	play	play
playing	playing	playing	playing
timeupdate	timeupdate	timeupdate	timeupdate
progress	ended	ended	ended
waiting			
canplay			
playing			
canplaythrough			
suspend			
ended			

注意：

（1）数据在服务器上时，Safari 4/5保存相同的顺序。

（2）iPad上不能进行本地测试，只进行了服务器上的测试。

（3）iPad上不能进行本地测试，只进行了服务器上的测试。

5.4 使用Video实现实时字幕

本节使用Video实现实时字幕。例子中使用到了jQuery库，jQuery库只在读入字幕数据时使用。

本例中将字幕与视频重叠进行显示。即字幕显示在与<video>标签重叠的 中，与视频播放时间保持一致，如图5-2所示。

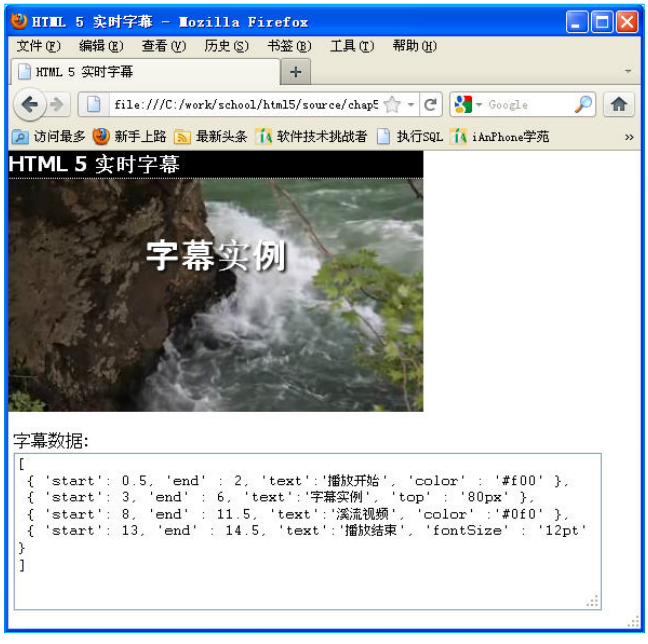

图5-2 实时字幕效果

● 知识专栏（Column）：jQuery库的使用

jQuery库在以下网站中进行下载，本例中使用了jQuery 1.3版本。

http://jquery.com

使用$.get()来读入JSON数据，其语法如下所示。

```
$.get(dataURL, callbackFunction);
```

第1个参数中指定读入数据的URL，第2个参数指定数据读入后的处理函数。本例中将读入的数据显示在画面下方。

5.4.1 HTML代码

在HTML中追加了显示视频的<video>标签，以及显示字幕用的 。在CSS中指定Z坐标，以使字幕显示在视频屏幕之上。

HTML5标准提供了用于在Firefox、Opera浏览器中显示的Theora/Ogg，用于在Safari、Chrome中显示H.264格式的视频。分别用了两个<source>，而且准备了针对其他不能显示视频的浏览器的信息。

因为本例是实时字幕，提供了在视频播放的同时可以编辑字幕的<textarea>区域，在此区域中显示读入的字幕数据，而且当用户编辑此区域内的数据时，可以实时的反映在字幕上。字幕数据的JSON形式如下所示。

```
JSON形式
[
 { 'start': 显示开始时刻秒, 'end' : 显示结束时刻秒, 'text':'字幕内容', 'color'
: '#f00' },
   :
  /*加入用于显示的各条字幕*/
]
```

字幕数据中可指定如下选项。此例中特别将选项名定义成与CSS属性相同的名称，可指定值与CSS相同，如表5-11所示。

表5-11 显示选项

选项	说明
color	字幕文字颜色
fontsize	字幕文字尺寸
top	离屏幕上端距离

```
    [HTML]02/index.html
    <!DOCTYPE html>
    <html lang="zh">
       <head>
              <meta charset="utf-8">
              <title>HTML 5 实时字幕</title>
              <link rel="stylesheet" href="css/main.css" type="text/css"
media="all">
              <script type="text/javascript" src="js/jquery-1.3.2.min.js"></script>
              <script type="text/javascript" src="js/telopper.js"></script>
       </head>
       <body>
              <h1>HTML 5 实时字幕</h1>
              <div id="monitor">
                     <video id="myVideo" controls>
                            <source src="movie/kiso.mov">
                            <source src="movie/kiso.ogg">
                            <p>请在支持video标签的浏览器中播放</p>
                     </video>
                     <div id="telop"></div>
              </div>
              <form>
                     字幕数据:<br>
                     <textarea id="telopCommand" cols="30" rows="8"></textarea>
              </form>
       </body>
    </html>
    02/data/telop.js
    [
     { 'start': 0.5, 'end' : 2, 'text':'播放开始', 'color' : '#f00' },
```

```
    { 'start': 3, 'end' : 6, 'text':'字幕实例', 'top' : '80px' },
    { 'start': 8, 'end' : 11.5, 'text':'溪流视频', 'color' :'#0f0' },
    { 'start': 13, 'end' : 14.5, 'text':'播放结束', 'fontSize' : '12pt' }
]
```

5.4.2 脚本代码

在脚本代码中，首先在onload事件中设置各种事件的处理代码。在一系列处理中读入字幕数据。这里使用jQuery库完成对字幕数据的读取，使用$.get()可以读取同一主机上的JSON数据。读入JSON数据后，再调用eval()方法将JSON数据保存到数组变量telopData中。

接着设置在视频中显示字幕用的timeupdate事件。触发timeupdate事件后，首先取得播放时间。播放时间在currentTime属性中，通过循环对比字幕数据中的字幕开始时间以及字幕结束时间，以判断是否显示字幕。

可指定显示字幕的字符尺寸以及颜色，此时将JSON数据中相应项目设置到对应的CSS属性中即可，代码如下。

```
tObj.style.color = telopData[i].color || "#fff";
```

最后设置文本区域内容变化时的处理。文本区域内容变化时触发change事件。此时与读入JSON数据后的处理一样，调用eval()方法同样将变化后的数据保存到数组变量telopData中。

这样就算完成了实时字幕显示实例，经过修改后还可以实现从左至右移动的字幕效果，这个交由读者自行完成。如图5-3所示是实时字幕的效果图。

```
[JavaScript]02/js/telopper.js
var vObj, tObj, cmdObj, telopData = [];
var telopURL = "data/telop.js";       // 字幕数据的URL
window.addEventListener("load", function(){
   vObj = document.getElementById("myVideo");
   tObj = document.getElementById("telop");
   cmdObj = document.getElementById("telopCommand");
```

```
    // 读入字幕数据
    $.get(telopURL, function(text){
            cmdObj.value = text;
            telopData = eval(text);
    });
    // 播放是发生timeupdate事件时显示字幕处理
    vObj.addEventListener("timeupdate", function(){
            var csec = vObj.currentTime;
            for(var i=0; i<telopData.length; i++){
                    if ((telopData[i].start <= csec) && (telopData[i].end >= csec)){
                            tObj.innerHTML = telopData[i].text;
                            tObj.style.color = telopData[i].color || "#fff";
                            // 字符颜色
                            tObj.style.top = telopData[i].top || "170px";
                            // 纵向显示位置
                            tObj.style.fontSize = telopData[i].fontSize || "24pt";
                            // 字符尺寸
                            return;
                    }
            }
            tObj.innerHTML = "";
    }, true);
    // 文本区域内的数据更新时显示字幕数据
    cmdObj.addEventListener("change", function(){
            telopData = eval(cmdObj.value);
    }, true);
}, true);
```

图5-3 实时显示字幕的效果图

5.5 视频与Canvas的组合技巧

HTML5 Video可以将视频传送/绘制到Canvas上，并可对播放中的视频进行加工后再显示。因为能自由的指定显示尺寸、位置，所以很容易实现在视频中显示另外的视频（即“画中画”功能）。向Canvas传送过程中还可进行像素单位的加工。本节中将介绍在向Canvas传送视频的过程中，进行灰色变换（即将彩色视频变成黑白视频）的例子。并进一步介绍了对灰色变换后的视频进行再合成的处理方法。

另外，关于Canvas的有关知识请参照本书的第2章。灰色变换处理请参考本书的2.3.2 节像素处理，本节的灰色变换使用了与此节相同的代码。

5.5.1 在Canvas上绘制视频影像

首先实现在Canvas上实时传送并显示视频的功能。画面上分别拥有播放视频的<video>标签，以及<canvas>标签。都赋予适当ID，便于在脚本中访问。其HTML代码如下。

```
[HTML]videoCanvas1/index.html
<!DOCTYPE html>
<html lang="zh">
   <head>
          <meta charset="utf-8">
          <title>HTML5 实时视频效果</title>
          <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
          <script type="text/javascript" src="js/videoCanvas.js"></script>
   </head>
   <body>
          <h1>HTML5 实时视频效果</h1>
          <video id="myVideo" controls autobuffer>
                 <source src="movie/matsumoto.mov">
                 <source src="movie/matsumoto.ogv">
                 <p>请在支持HTML5 video标签的浏览器中</p>
          </video>
          <canvas id="myCanvas" width="384" height="216"></canvas>
```

```
    </body>
</html>
```

其中<video>标签的ID为myVideo，<canvas>标签的ID为myCanvas。

在Canvas中绘制视频影像时，使用Canvas上下文的drawImage()方法。将drawImage()方法的第1个参数指定为<video>元素即可。第2个参数指定为横坐标，第3个参数为纵坐标。

drawImage()方法中仅对当前正在播放视频的帧进行描绘。因此每次响应timeupdate事件时，必须使用drawImage()方法在Canvas中进行视频影像绘制。实际脚本如下所示。


```
[JavaScript]videoCanvas1/js/videoCanvas.js
window.addEventListener("load", function(){
    var videoObj = document.getElementById("myVideo");
    var canvasObj = document.getElementById("myCanvas");
    videoObj.addEventListener("timeupdate", function(){
            canvasObj.getContext("2d").drawImage(videoObj, 0, 0);
    }, true);
}, true);
```

运行上述脚本后，在Firefox（3.6以后版本）中可以在Canvas里实时绘制视频影像，如图5-4所示。但是与其他浏览器相比较，播放速度有显著下降。这是由于在不同的浏览器中，timeupdate事件发生的时机与次数有差异而造成的。

因此如果想让任何浏览器上都显示相同的效果，这里就不要使用timeupdate事件了，而使用timer[setTimeout()或者setInterval()]来实现相同的处理，如图5-5所示。

假设视频的播放帧率为30fps，则在timer方法的第2个参数中设置为1000/30。到底能以多高的速度进行播放与环境有关，而且需要注意的是，这里不保证能在1秒钟内能播放指定的帧数。修改后的脚本代码如下。

```
[JavaScript]videoCanvas2/js/videoCanvas.js
window.addEventListener("load", function(){
    var fps = 1000/30;   // 1/30秒
    var videoObj = document.getElementById("myVideo");
    var canvasObj = document.getElementById("myCanvas");
    setInterval(function(){
```

```
            canvasObj.getContext("2d").drawImage(videoObj, 0, 0);
        }, fps);
    }, true);
```

图5-4 使用timeupdate事件实现的实时视频传送（在Firefox中速度下降）

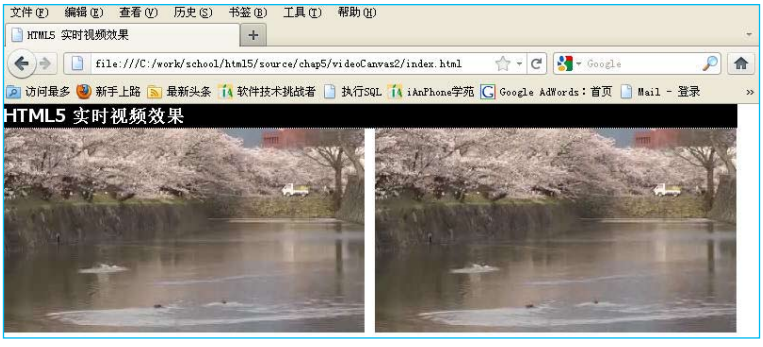

图5-5 使用timer实现的实时视频传送

5.5.2 对视频进行黑白影像变换

在实现了实时向Canvas传送视频影像后，接着完成下一步，即在传送中对视频进行黑白影像变换。关于像素变换的脚本说明请参照2.3.2节。

因为需要对整个Canvas区域进行变换。首先取得Canvas的宽带与高度值。

```
var w = canvasObj.width;        // 画布的宽度
var h = canvasObj.height;       // 画布的高度
```

接着调用变换函数。此函数的参数为容纳Canvas内像素数据的数组，因此在之前

需要调用上下文的getImageData()方法取得容纳Canvas内像素数据的数组。变换完成后再调用putImageData方法重新将像素数据的数组设置到Canvas中。完整代码如下。

```
var imageData = ctx.getImageData(0, 0, w, h);
convert _ image _ to _ gray _ scale(imageData.data);
ctx.putImageData(imageData,0, 0);
```

这样视频影像就被成功的进行了黑白变换，如图5-6所示。但是，如果需要更复杂的特效，处理速度可能会影响实时播放的效果，出现播放速度下降的问题，播放速度还与具体使用的浏览器以及硬件软件环境相关。

图5-6 黑白变换后的播放效果（Opera浏览器）

```
[JavaScript]videoCanvas3/js/videoCanvas.js
window.addEventListener("load", function(){
   var fps = 1000/30;   // 1/30秒
   var videoObj = document.getElementById("myVideo");
   var canvasObj = document.getElementById("myCanvas");
   var w = canvasObj.width;    // 画布的宽度
   var h = canvasObj.height;   // 画布的高度
   setInterval(function(){
          var ctx = canvasObj.getContext("2d");
          ctx.drawImage(videoObj, 0, 0);      // 将视频设置到画布中
          var imageData = ctx.getImageData(0, 0, w, h);
          convert _ image _ to _ gray _ scale(imageData.data);
          ctx.putImageData(imageData, 0, 0);
   }, fps);
```

```
}, true);
// ----------------------------------------
// 灰色转换
// ----------------------------------------
function convert_image_to_gray_scale(data) {
   var len = data.length;
   var pixels = len / 4;
   for(var i=0; i<pixels; i++){
          var r = data[i*4];
          var g = data[i*4+1];
          var b = data[i*4+2];
          var gray = parseInt((11*r + 16*g + 5*b) / 32);
          data[i*4] = gray;
          data[i*4+1] = gray;
          data[i*4+2] = gray;
   }
}
```

注意：上述代码只能在Safari（Mac，版本≥4）,以及Opera浏览器中才能正常运行。

5.5.3 显示加工后的视频

上节的例子中，视频（<video>）与Canvas都显示在同一画面中。通常最好不要再显示原来的视频。只显示经过黑白变换后的视频。这时可以通过使用样式单将<video>变成隐藏的。当视频（<video>）隐藏起来后，其上的播放控制按钮也不能显示了，因此需要创建新的控制按钮。

```
[JavaScript]videoCanvas4/index.html
<!DOCTYPE html>
<html lang="zh">
   <head>
          <meta charset="utf-8">
```

```
        <title>HTML5 实时视频效果</title>
        <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
        <script type="text/javascript" src="js/videoCanvas.js"></script>
    </head>
    <body>
        <h1>HTML5 实时视频效果</h1>
        <video id="myVideo" controls autobuffer>
            <source src="movie/matsumoto.mov">
            <source src="movie/matsumoto.ogv">
            <p>请在支持HTML5 video标签的浏览器中</p>
        </video>
        <canvas id="myCanvas" width="384" height="216"></canvas>
        <div id="myController">
            <img src="images/playButton.png" id="playVideo">
        </div>
    </body>
</html>
```

播放按钮的点击事件通过addEventListener来追加，事件发生后，先将currentTime设为0，接着调用play()开始视频播放。这样当点击播放按钮时，将总是从头播放视频。

整个脚本代码如下所示。点击播放按钮后，实时播放经过黑白变换后的视频如图5–7所示。

图5–7 只显示Canvas中经过黑白变换后的视频

```
[JavaScript]videoCanvas4/js/videoCanvas.js
window.addEventListener("load", function(){
   var fps = 1000/30;          // 1/30秒
   var videoObj = document.getElementById("myVideo");
   var canvasObj = document.getElementById("myCanvas");
   var w = canvasObj.width;    // 画布的宽度
   var h = canvasObj.height;   // 画布的高度
   setInterval(function(){
          var ctx = canvasObj.getContext("2d");
          ctx.drawImage(videoObj, 0, 0);     // 将视频设置到画布中
          var imageData = ctx.getImageData(0, 0, w, h);
          convert _ image _ to _ gray _ scale(imageData.data);
          ctx.putImageData(imageData, 0, 0);
   }, fps);
   // 设置播放按钮
   var playButton = document.getElementById("playVideo");
   playButton.addEventListener("click", function(){
          videoObj.currentTime = 0;          // 回退
          videoObj.play();                   // 播放
   }, true)
}, true);
// ----------------------------------------
// 灰色转换
// ----------------------------------------
function convert _ image _ to _ gray _ scale(data) {
   var len = data.length;
   var pixels = len / 4;
   for(var i=0; i<pixels; i++){
          var r = data[i*4];
          var g = data[i*4+1];
```

```
            var b = data[i*4+2];
            var gray = parseInt((11*r + 16*g + 5*b) / 32);
            data[i*4] = gray;
            data[i*4+1] = gray;
            data[i*4+2] = gray;
        }
    }
```


最后，我们试着实现在上述黑白视频的中央部分显示原来的彩色视频，如图5-8所示。这种特效有时会在电视机上看到，这里使用HTML5 Video与Canvas的组合很容易就可以实现，即在黑白视频对应的位置调用drawImage()方法绘制指定大小的视频。例如如下的代码就是在坐标（42，23）处绘制尺寸为宽300像素，高169像素的视频。

```
ctx. drawImage(videoObj,42,23,300,169);
```

通过Video与Canvas的组合，可以很方便的对视频进行加工。Canvas中绘制的数据可以以文件的形式保存，通过合适的使用方法可以实现如视频编辑软件一样的效果。

图5-8 黑白视频中央绘制彩色视频

```
[JavaScript]videoCanvas5/js/videoCanvas.js
window.addEventListener("load", function(){
    var fps = 1000/30;    // 1/30秒
    var videoObj = document.getElementById("myVideo");
    var canvasObj = document.getElementById("myCanvas");
```

```
    var w = canvasObj.width;    // 画布的宽度
    var h = canvasObj.height;   // 画布的高度
    setInterval(function(){
            var ctx = canvasObj.getContext("2d");
            ctx.drawImage(videoObj, 0, 0);      // 将视频设置到画布中
            var imageData = ctx.getImageData(0, 0, w, h);
            convert _ image _ to _ gray _ scale(imageData.data);
            ctx.putImageData(imageData, 0, 0);
            ctx.drawImage(videoObj, 42, 23, 300, 169);
                                                // 缩小视频尺寸显示在画布上
    }, fps);
    // 设置播放按钮
    var playButton = document.getElementById("playVideo");
    playButton.addEventListener("click", function(){
            videoObj.currentTime = 0;           // 回退
            videoObj.play();                    // 播放
            return false;
    }, true)
}, true);
// ----------------------------------------
// 灰色转换
// ----------------------------------------
function convert _ image _ to _ gray _ scale(data) {
    var len = data.length;
    var pixels = len / 4;
    for(var i=0; i<pixels; i++){
            var r = data[i*4];
            var g = data[i*4+1];
            var b = data[i*4+2];
            var gray = parseInt((11*r + 16*g + 5*b) / 32);
```

```
            data[i*4] = gray;
            data[i*4+1] = gray;
            data[i*4+2] = gray;
        }
    }
```

5.6 创建简易音频播放器

本节使用HTML5的Audio功能创建简易音频播放器。点击音频播放器乐曲列表中的任意曲目（文件名/URL）时，对应曲目将被演奏。另外还提供了回退、播放、停止等三个按钮，并显示现在播放到第几秒，以及该曲目总时长为多少秒。因为是简易音频播放器，没有提供与音频一起展示的视频效果，但是可以在本节简易音频播放器的基础上与Canvas或Video、SVG的功能一起结合，创建拥有视觉特效的播放器。图5-9为简易播放器的外观。

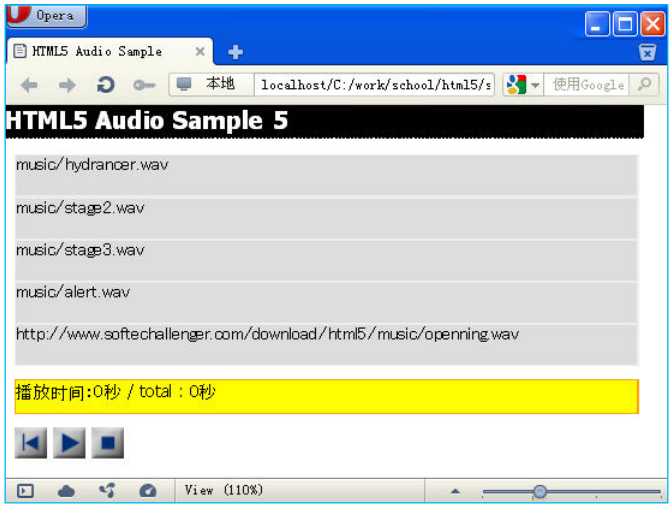

图5-9 简易音频播放器

在播放音频时，随着浏览器的不同所支持的音频格式也不相同。WAVE格式（后缀为.wav）在Google Chrome中是不能播放的，而Ogg格式（.ogg）在Safari中也不能播放。因此，我们需要准备几种各浏览器特有的音频格式文件。

本节中，我们准备了WAVE以及Ogg两种格式的音频文件。表5-12为各常用浏览

器支持的音频格式。

表5-12 各浏览器支持的音频格式列表

	wave	ogg	aiff	mp3	au
Firefox	○	○	×	×	×
Google Chrome	×	○	×	○	×
Safari	○	×	○	△	○
Opera	○	○	×	×	×

△：有些音频数据播放时会花费很长时间

5.6.1 播放音频

在播放音频数据时，使用如下的audio元素，在其src属性中指定音频文件名或其URL（音频地址）。

```
<audio src="sample.wav">
```

但是，因各种浏览器支持的音频格式不同，指定的一种音频文件有可能在其他浏览器上不能播放。此时与<video>的处理方式一样，在<video>元素中追加多个source子元素，如下所示。

```
<audio>
<source src="sample.ogg">
<source src="sample.wav">
<source src="sample.mp3">
<source src="sample.aif">
</audio>
```

需要特别强调的是，在Firefox中，如果第一个文件不能播放，此浏览器将不会向下搜索，因此需要将受Firefox支持的Ogg格式的放在第一的位置。

如果要控制audio元素中指定的音频，首先使用document.getElementById等取得具体元素对象，然后与video元素一样，调用play()、pause()等方法进行相关处理。而且currentTime等属性的用法也与video元素一样。

在以下的Audio/sample1中，当页面导入后，通过脚本以一半的音量进行音频的播放。

```
[HTML]Audio/sample1/index.html
<!DOCTYPE html>
<html lang="zh">
   <head>
          <meta charset="utf-8">
          <title>HTML5 Audio Sample</title>
          <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
   </head>
   <body>
          <h1>HTML5 Audio Sample</h1>
          <div id="contents">
                 <audio id="myAudio" controls>
                        <source src="music/hydrancer.ogg">
                        <source src="music/hydrancer.wav">
                        <source src="music/hydrancer.mp3">
                        <source src="music/hydrancer.aif">
                 </audio>
          </div>
          <script type="text/javascript" src="js/audio.js"></script>
   </body>
</html>

[JavaScript]Audio/sample1/js/audio.js
var audio1 = document.getElementById("myAudio");
audio1.volume = 0.5;    // 一半音量
audio1.play();
```


5.6.2 在脚本中控制音频

播放音频时，不使用audio元素，只通过脚本就可以控制音频文件的播放。本小

节将制作一个不定义audio标签，只使用脚本实现简易音频播放器，如图5-10所示。

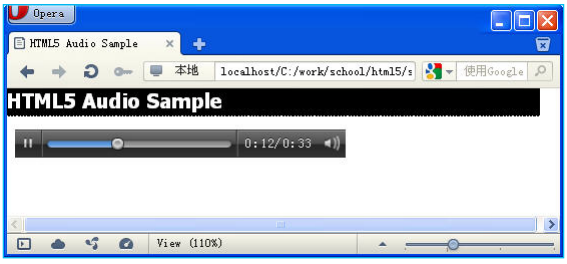

图5-10 不使用<audio>的简易播放器

首先我们看看WAVE格式数据的播放处理。画面音频文件列表中有多个WAVE格式的音频文件，当点击其中一个音频文件时，会触发点击事件，点击事件的处理代码由addEventListener()方法完成追加，其代码形式如下。

```
var ele = document.querySelectorAll("li");
for(var i=0; i<ele.length; i++){
   ele[i].addEventListener("click",function(){
          //点击后的处理
   }, true);
}
```

首先读入音频文件列表中的所有音频文件的<li>对象，对每一个音频文件追加点击事件处理代码。

播放音频文件时，如下述代码一样，使用new Audio()，在其参数中指定播放文件名或者其URL。调用由new Audio()创建的对象的play()方法，就可以完成数据的读入并进行播放。

```
(new Audio("文件名或URL")).play();
```

实际代码在Audio/sample2中，当点击列表中的任何曲目时，此曲目将被播放。因为只有WAVE格式的音频文件，此简易音频播放器在Chrome中不能运行。

```
[HTML]Audio/sample2/index.html
<!DOCTYPE html>
<html lang="zh">
   <head>
          <meta charset="utf-8">
```

```
        <title>HTML5 Audio Sample</title>
        <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
        <script type="text/javascript" src="js/audio.js"></script>
    </head>
    <body>
        <h1>HTML5 Audio Sample 2 (wav)</h1>
        <ul>
            <li>music/hydrancer.wav</li>
            <li>music/stage2.wav</li>
            <li>music/stage3.wav</li>
            <li>music/alert.wav</li>
        </ul>
    </body>
</html>

[JavaScript]Audio/sample2/js/audio.js
window.addEventListener("load",function(){
    // 根据列表设置播放曲目
    var ele = document.querySelectorAll("li");
    for(var i=0; i<ele.length; i++){
        ele[i].addEventListener("click",function(){
            var audioName = this.firstChild.textContent;
                                                    // 取得曲目的文件名
            (new Audio(audioName)).play();          // 播放音乐
        }, true);
    }
}, true);
```


5.6.3 检查音频文件是否可播放

上小节中的实例Audio/sample2在Google Chrome中是不能播放的。就算追加了在

Google Chrome中可播放的音频文件，今后如果出现其他新的浏览器中，还是有可能出现不能播放的问题。这时如果能追加一个检查音频文件是否能在特定浏览器中播放的功能，将方便多了。在HTML5 Audio中提供了canPlayType()的方法。在其参数中指定MIME类型后，就能返回该浏览器是否支持此音频格式的结果。

如果浏览器支持对应的MIME类型，将返回字符串“maybe”。本例中使用canPlayType()方法检查WAVE格式是否可播放。不能播放的情况下，重新读入Ogg格式的视频文件。并且准备了文件名相同，后缀不同的音频文件，所处目录也相同。即类似下述文件。

```
sample.wav
```

与

```
sample.ogg
```

这里只有WAVE及Ogg两种格式能在常用的几种浏览器中完成播放。也可以追加其他新的格式音频的检查处理。

上述处理已都在Audio/sample3中实现了，但是此例中有一个缺点。即当一个曲目在播放过程中，点击另一个曲目后，播放中曲目不会停止，而是两个曲目同时播放。这对于那些需要背景音乐的应用来说没有任何问题，但对于音频播放器来说是不合适的。

经过修改在Audio/sample4中实现了同时只播放一首曲目的功能。修改Audio/sample3中创建音频对象处的代码，Audio/sample4中将只能同时创建一个音频对象。具体修改的地方如下。

```
(new Audio(audioName)).play();
```

改为

```
(myAudio = new Audio(audioName)).play();
```

Audio/sample3以及Audio/sample4的代码如下，其中包含了详细的代码注释，请参考注释理解脚本代码的含义。

```
[HTML]Audio/sample3/index.html
<!DOCTYPE html>
<html lang="zh">
  <head>
        <meta charset="utf-8">
```

```
        <title>HTML5 Audio Sample</title>
        <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
        <script type="text/javascript" src="js/audio.js"></script>
    </head>
    <body>
        <h1>HTML5 Audio Sample 3</h1>
        <ul>
            <li>music/hydrancer.wav</li>
            <li>music/stage2.wav</li>
            <li>music/stage3.wav</li>
            <li>music/alert.wav</li>
        </ul>
    </body>
</html>

[JavaScript]Audio/sample3/js/audio.js
window.addEventListener("load",function(){
    var myAudio = new Audio();
    // 根据列表设置播放曲目
    var ele = document.querySelectorAll("li");
    for(var i=0; i<ele.length; i++){
        ele[i].addEventListener("click",function(){
            myAudio.pause();                    // 播放暂停
            var audioName = this.firstChild.textContent;
                                                // 取得曲目文件名
            if(myAudio.canPlayType("audio/wav") != "maybe") {
                // 不能播放wav形式音频的情况下播放ogg格式
                audioName = audioName.replace(/\.wav/,".ogg");
            }
            (new Audio(audioName)).play();      // 播放音乐
```

```
            }, true);
    }
}, true);

[HTML]Audio/sample4/index.html
<!DOCTYPE html>
<html lang="zh">
    <head>
            <meta charset="utf-8">
            <title>HTML5 Audio Sample</title>
            <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
            <script type="text/javascript" src="js/audio.js"></script>
    </head>
    <body>
            <h1>HTML5 Audio Sample 4</h1>
            <ul>
                    <li>music/hydrancer.wav</li>
                    <li>music/stage2.wav</li>
                    <li>music/stage3.wav</li>
                    <li>music/alert.wav</li>
                    <li>http://www.openspc2.org/book/HTML5/music/openning.wav</li>
            </ul>
    </body>
</html>
Audio/sample4/js/audio.js
window.addEventListener("load",function(){
    var myAudio = new Audio();
    // 根据列表设置播放曲目
    var ele = document.querySelectorAll("li");
    for(var i=0; i<ele.length; i++){
```

```
            ele[i].addEventListener("click",function(){
                    myAudio.pause();      // 播放暂停
                    var audioName = this.firstChild.textContent;
                                          // 取得曲目文件名
                    if(myAudio.canPlayType("audio/wav") != "maybe") {
                            // 不能播放wav形式音频的情况下播放ogg格式
                            audioName = audioName.replace(/\.wav/,".ogg");
                    }
                    (myAudio = new Audio(audioName)).play();  // 播放音乐
            }, true);
    }
}, true);
```


5.6.4 显示播放时间

最后追加播放按钮以及显示播放时间的功能。播放与停止时与video元素中使用的方法完全相同。播放使用play()方法、停止使用pause()方法。pause()方法是暂停播放，当再次调用play()方法时，将从上次中断的地方开始播放。回退处理是将currentTime属性重新设置为0，再调用play()方法时会从头开始播放。

通过使用timeupdate事件来实现播放时间的显示。与视频的简易字幕处不同，这里细微时间单位不进行处理与显示，因此使用timeupdate事件没有问题。timeupdate事件发生时，从currentTime属性中读取当前播放时间，然后显示在画面上。总播放时间通过duration属性读取即可。

在Audio/sample5中实现了上述功能。点击音频文件列表中的曲目后，开始播放。各种按钮可以控制其播放、暂停、以及回退，如图5-11所示。如果列表部分以XML或者JSON数据形式读入的话，可以实现类似于MediaPlayer或者iTunes样的播放器，有兴趣的读者可以在本例的基础上尝试一下。

```
[HTML]Audio/sample5/index.html
<!DOCTYPE html>
<html lang="zh">
```

```
    <head>
        <meta charset="utf-8">
        <title>HTML5 Audio Sample</title>
        <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
        <script type="text/javascript" src="js/audio.js"></script>
    </head>
    <body>
        <h1>HTML5 Audio Sample 5</h1>
        <ul>
            <li>music/hydrancer.wav</li>
            <li>music/stage2.wav</li>
            <li>music/stage3.wav</li>
            <li>music/alert.wav</li>
            <li>http://www.softechallenger.com/download/html5/music/
openning.wav</li>
        </ul>
        <div id="stat">播放时间: 0秒 / total : 0秒</div>
        <div id="control">
            <img src="images/rewind.gif" id="rewindButton">
            <img src="images/play.gif" id="playButton">
            <img src="images/stop.gif" id="stopButton">
        </div>
    </body>
</html>

[JavaScript]Audio/sample5/js/audio.js
window.addEventListener("load",function(){
    var myAudio = new Audio();
    // 根据列表设置播放曲目
    var ele = document.querySelectorAll("li");
    for(var i=0; i<ele.length; i++){
```

```
            ele[i].addEventListener("click",function(){
                    myAudio.pause();      // 播放暂停
                    var audioName = this.firstChild.textContent;
                                          // 取得曲目文件名
                    if(myAudio.canPlayType("audio/wav") != "maybe") {
                            // 不能播放wav形式音频的情况下播放ogg格式
                            audioName = audioName.replace(/\.wav/,".ogg");
                    }
                    (myAudio = new Audio(audioName)).play();  // 播放音乐
                    // 显示播放时间
                    myAudio.addEventListener("timeupdate", function(){
                            var ct = parseInt(myAudio.currentTime);
                    // 当前的播放时间(舍弃小数点以下数据)
                            var total = parseInt(myAudio.duration);
                    // 整体播放时间(舍弃小数点以下数据)
                            document.getElementById("stat").innerHTML =
"播放时间:" + ct + "秒 / total : " + total + "秒";
                    }, true);
            }, true);
    }
    // 播放按钮、停止按钮、返回按钮的设置
    document.getElementById("playButton").addEventListener("click", function(){
            myAudio.play();
    }, true);
    document.getElementById("stopButton").addEventListener("click", function(){
            myAudio.pause();
    }, true);
    document.getElementById("rewindButton").addEventListener("click", function(){
            myAudio.currentTime = 0;
    }, true);
}, true);
```

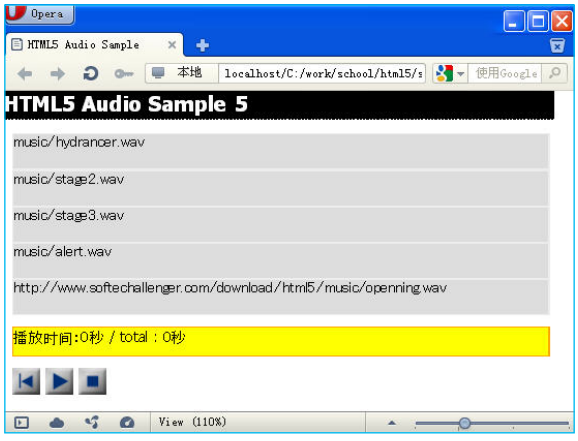

图5-11 简易音频播放器

5.7 制作乐器演奏程序

本节将制作一个乐器演奏程序。使用HTML5 audio实现爵士鼓（鼓、 钹、高帽等）的功能。乐器的原理都差不多，用同一种方法可以实现钢琴或者笛子等自己喜欢的乐器。本例中只是将各种分乐器布置在矩形区域内，如果使用合适的图片可以制作出更有真实感的乐器。如图5-12所示是本乐器演奏程序的画面外观。

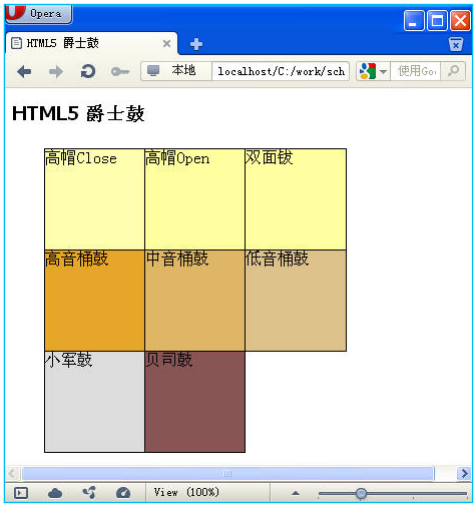

图5-12 HTML5 audio实现的爵士鼓外观

此乐器演奏程序会根据点击的区域来判断子乐器名，然后播放相应的音频。与上一节制作简易音频播放器一样，都事先准备了各种子乐器的音频文件，WAVE格式或者Ogg格式。

在上一节的简易音频播放器中我们将WAVE格式的音频文件放在了优先播放的位置上，其实WAVE格式的音频文件因为数据量比较大，不适合频繁出声的乐器演奏。因此这里将Ogg格式放在了优先的位置，不支持Ogg格式时才使用WAVE格式。检查方法如下所示。

```
var ext = '.wav';
if((new Audio()).canPlayType('audio/ogg') == 'maybe') { ext = '.ogg'; }
```

变量ext中保存着可演奏的音频格式。

5.7.1 通过点击演奏

音频文件准备好了后，接着就可以编辑HTML文件代码了。为了让脚本代码变得更简单，这里下了一点小功夫。首先，将子乐器名显示在拥有ID的DIV元素中。如下所示。

```
<div id="snare _ drum">小军鼓</div>
```

上述子乐器的ID名称为snare_drum。这里特别将音频文件名称保持与此ID名称一致。即相应准备了如下两个音频文件。

```
snare _ drum.ogg
snare _ drum.wav
```

这样当点击相应区域时可以很简单的取得对应音频文件的名称。处理代码只需要如下两行就可以了。

```
var audioObj = new Audio("sound/"+this.id + ext);
audioObj.play();
```

实际HTML代码以及脚本代码如下。

```
[HTML]drumset _ sample/sample1/index.html
<!DOCTYPE html>
<html lang="zh">
```

```
    <head>
        <meta charset="utf-8">
        <title>HTML5 爵士鼓</title>
        <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
    </head>
    <body>
        <h1>HTML5 爵士鼓</h1>
        <div id="drumset">
            <div id="high _ hat _ close">高帽Close</div>
            <div id="high _ hat _ open">高帽Open</div>
            <div id="crash _ cymbal">双面钹</div>
            <div id="high _ tom">高音桶鼓</div>
            <div id="mid _ tom">中音桶鼓</div>
            <div id="low _ tom">低音桶鼓</div>
            <div id="snare _ drum">小军鼓</div>
            <div id="bass _ drum">贝司鼓</div>
        </div>
        <script type="text/javascript" src="js/drumset.js"></script>
    </body>
</html>
[JavaScript]drumset _ sample/sample1/js/drumset.js
var ext = '.wav';
if((new Audio()).canPlayType('audio/ogg') == 'maybe') { ext = '.ogg'; }
// 设置乐器对应的声音
var drumlist = document.getElementById("drumset").getElementsByTag
Name("div");
for(var i=0; i<drumlist.length; i++){
// 设置点击事件
    drumlist[i].addEventListener("click", function(){
        var audioObj = new Audio("sound/"+this.id + ext);
        audioObj.play();
```

```
    }, true);
}
```

5.7.2 通过按键演奏

上面实现了点击的演奏方式，本小节将其修改为通过按键方式演奏爵士鼓。表5-13为键与子乐器的对照表。

表5-13 键与子乐器的对照

键	子乐器名
Q	高帽Close
W	高帽Open
E	双面钹
A	高音桶鼓
S	中音桶鼓
D	低音桶鼓
Z	小军鼓
X	贝司鼓

HTML代码与上一节的sample1基本一致。只是为了方便用户按键，在每个子乐器前加上了对应的字母键。下面定义了键与对应演奏音频文件的对应表。数组中有字母（对应于键盘上的按键），以及无后缀的文件名。

```
var keydata = [
    { key : "Z", sound : "snare_drum" },
    { key : "X", sound : "bass_drum" },
    { key : "A", sound : "high_tom" },
    { key : "S", sound : "mid_tom" },
    { key : "D", sound : "low_tom" },
    { key : "Q", sound : "high_hat_close" },
    { key : "W", sound : "high_hat_open" },
    { key : "E", sound : "crash_cymbal" }
];
```

注意：

key后为字母按钮，sound后为文件名。

通过for循环来判断点击的键是否在定义的数组中，如果判断在数组中，则播放相应的音频文件。这里允许同时播放多个音频文件。代码如下。

```
var kc = String.fromCharCode(e.keyCode);
for(var i=0; i<keydata.length; i++){
   if (kc == keydata[i].key){
          var audioObj = new Audio("sound/"+keydata[i].sound+ext);
          audioObj.play();
   }
}
```


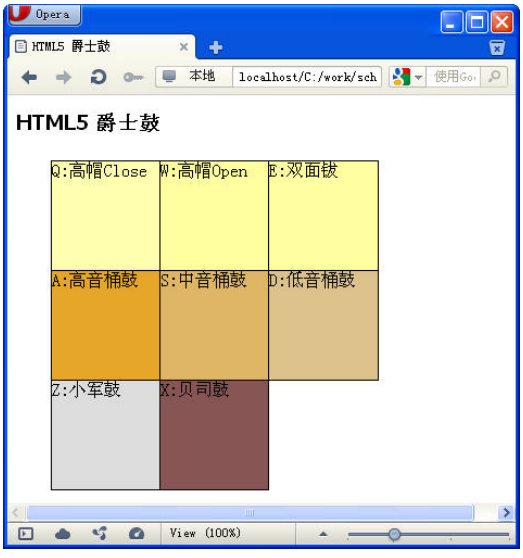

图5-13 按键演奏的爵士鼓

如图5-13所示的效果实际代码如下所示。请注意，在便携式智能设备iPhone/iPad中，由于内存比较小，同时只能演奏一种乐器。

```
[HTML]drumset _ sample/sample2/index.html
<!DOCTYPE html>
<html lang="zh">
```

```
    <head>
        <meta charset="utf-8">
        <title>HTML5 爵士鼓</title>
        <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
    </head>
    <body>
        <h1>HTML5 爵士鼓</h1>
        <div id="drumset">
            <div id="high_hat_close">Q:高帽Close</div>
            <div id="high_hat_open">W:高帽Open</div>
            <div id="crash_cymbal">E:双面钹</div>
            <div id="high_tom">A:高音桶鼓</div>
            <div id="mid_tom">S:中音桶鼓</div>
            <div id="low_tom">D:低音桶鼓</div>
            <div id="snare_drum">Z:小军鼓</div>
            <div id="bass_drum">X:贝司鼓</div>
        </div>
        <script type="text/javascript" src="js/drumset.js"></script>
    </body>
</html>

[JavaScript]drumset_sample/sample2/js/drumset.js
var ext = '.wav';
if((new Audio()).canPlayType('audio/ogg') == 'maybe') { ext = '.ogg'; }
// 设置乐器对应的声音
var drumlist = document.getElementById("drumset").getElementsByTagName
("div");
for(var i=0; i<drumlist.length; i++){
    // 设置点击事件
    drumlist[i].addEventListener("click", function(){
```

```
            var audioObj = new Audio("sound/"+this.id + ext);
            audioObj.play();
    }, true);
}
// 设置键的事件
document.addEventListener("keydown", function(e){
    var keydata = [
            { key : "Z", sound : "snare _ drum" },
            { key : "X", sound : "bass _ drum" },
            { key : "A", sound : "high _ tom" },
            { key : "S", sound : "mid _ tom" },
            { key : "D", sound : "low _ tom" },
            { key : "Q", sound : "high _ hat _ close" },
            { key : "W", sound : "high _ hat _ open" },
            { key : "E", sound : "crash _ cymbal" }
    ];
    var kc = String.fromCharCode(e.keyCode);
    for(var i=0; i<keydata.length; i++){
            if (kc == keydata[i].key){
                    var audioObj = new Audio("sound/"+keydata[i].sound+ext);
                    audioObj.play();
            }
    }
}, true);
```

将上例中按键处理部分使用Javascript中的数组，可以将代码简化成下述形式。

```
[JavaScript]drumset _ sample/sample3/js/drumset.js
//…省略…
// 设置键的事件
document.addEventListener("keydown", function(e){
    var keydata = [];
```

```
    keydata["Z"] = "snare _ drum";
    keydata["X"] = "bass _ drum";
    keydata["A"] = "high _ tom";
    keydata["S"] = "mid _ tom";
    keydata["D"] = "low _ tom";
    keydata["Q"] = "high _ hat _ close";
    keydata["W"] = "high _ hat _ open";
    keydata["E"] = "crash _ cymbal";
    var kc = String.fromCharCode(e.keyCode);
    if(keydata[kc]) (new Audio("sound/"+keydata[kc]+ext)).play();
}, true);
```


5.8 制作可变速视频播放器

HTML5中视频播放速度是可以自由改变的。也就是说，可以显示视频的慢放及快放功能。本节制作一个可变速的视频播放器。

使用滑块来进行播放速度调节。向左移动降低速度，向右移动增加速度。播放速度的调整随滑块的移动进行实时调整。

另外此例中使用到jQuery库的UI滑块，本章没有详细的介绍jQuery的相关应用，读者可以参考笔者所著的《jQurey即学即用》一书，或者参考jQuery的官方网站。

- jQuery的网址

http://jquery.com

- jQuery UI的网址

http://jqueryui.com

如图5-14所示是可变速视频播放器的画面外观，调整下面的滑块可以改变视频的播放速度。

图5-14 可变速视频播放器

5.8.1 HTML代码

首先将播放视频放置在<video>以及<source>标签中。本例中播放的视频是固定的，表示视频位置的任意URL或路径都可以。此部分代码如下。

```
<video id="myVideo">
   <source src="movie/sample.mov">
   <source src="movie/sample.ogv">
   <p>请在支持video标签的浏览器中播放</p>
</video>
```

接着编写视频控制部分，各个控件的ID、功能如表5-14所示。

表5-14 可变速视频播放器的控件列表

ID名称	功能
rewindButton	回退按钮
playButton	播放/停止按钮
ctime	播放时间
speed	播放速度
slider	速度控制滑块

如表5-14所示，本例中将使用自定义视频控制按钮。在video元素中去掉controls属性。这样默认控制按钮将不显示。

按钮部分使用div元素定义即可。另外创建滑块时会使用到如下的jQuery/UI库。

```
<link rel="stylesheet" href="css/ui-lightness/jquery-ui-1.7.3.custom.
css" type="text/css" media="all">
<script type="text/javascript" src="js/jquery-1.3.2.min.js"></script>
<script type="text/javascript" src="js/ui/ui.core.js"></script>
<script type="text/javascript" src="js/ui/ui.slider.js"></script>
<script type="text/javascript" src="js/player.js"></script>
```

容纳滑块的也是div元素，ID名称定义为slider。各控制按钮的显示位置，尺寸通过样式单来控制。全部HTML代码如下所示。

```
[HTML]HTML5 _ player/index.html
```

```
<!DOCTYPE html>
<html lang="zh">
   <head>
          <meta charset="utf-8">
          <title>HTML5 Video Player</title>
          <link rel="stylesheet" href="css/main.css" type="text/css" media="all">
          <link rel="stylesheet" href="css/ui-lightness/jquery-ui-1.7.3.
custom.css" type="text/css" media="all">
          <script type="text/javascript" src="js/jquery-1.3.2.min.js"></script>
          <script type="text/javascript" src="js/ui/ui.core.js"></script>
          <script type="text/javascript" src="js/ui/ui.slider.js"></script>
          <script type="text/javascript" src="js/player.js"></script>
   </head>
   <body>
          <h1>HTML5 Video Player</h1>
          <div id="viewArea">
                 <video id="myVideo">
                        <source src="movie/sample.mov">
                        <source src="movie/sample.ogv">
                        <p>请在支持video标签的浏览器中播放</p>
                 </video>
          </div>
          <div id="ctrl">
                 <img src="images/rewind.png" id="rewindButton">
                 <img src="images/on.png" id="playButton">
                 <div id="ctime">播放时间: 0秒</div><br>
                 <div id="speed">1倍速</div><br>
                 <div id="slider"></div>
          </div>
   </body>
</html>
```

5.8.2 实现各按钮功能

首先实现播放按钮的功能。点击播放按钮时将自动进行播放、停止按钮的切换。这里将播放与停止放在同一按钮中实现，为了区别分别准备了播放与停止的按钮图片，点击时切换按钮对应的图片即可。即视频播放时显示停止的图片，停止时显示播放的图片。通过paused属性来判定视频是否停止，paused等true时为停止中，false时为播放中。实际代码如下。

```
// 设置播放按钮的事件
pButton.addEventListener("click", function(){
   if (videoObj.paused == true){      // 停止时
          videoObj.play();            // 播放
          pButton.src = 'images/off.png';
   }else{
          videoObj.pause();
          pButton.src = 'images/on.png';
   }
}, true);
```

接着实现回退功能。点击回退按钮时将currentTime属性设置为0即可。

```
// 设置回退按钮的事件
rewButton.addEventListener("click", function(){
   videoObj.currentTime = 0;
}, true);
```

下一步完成播放时间显示的功能。当前的播放时间保存在currentTime属性中，将此值设置在播放时间标签中。但是此处想显示0.1秒单位的时间，因此将currentTime属性值乘以10倍，四舍五入后再除以10就可以显示0.1秒单位的时间了。

```
   document.getElementById("ctime").innerHTML = "播放时间:"+Math.floor
(videoObj.currentTime*10)/10+"秒";
```

作为HTML5中的视频播放器，我们需要编写当视频播放结束时的处理。一般，视频播放结束后，自动返回到开始位置，调用pause()方法停止视频播放。在此例中将原来显示为停止的按钮切换为显示播放的按钮。实际代码如下。

```
videoObj.addEventListener("ended", function(){
    videoObj.currentTime = 0;
    videoObj.pause();
    pButton.src = 'images/on.png';
}, true);
```


5.8.3 制作控制速度的滑块

最后追加调节播放速度的滑块。HTML5中由下述两种属性来控制视频的播放速度。

当前的播放速度	playbackRate
默认播放速度	defaultPlaybackRate

指定为0时停止，0.5时以一半的速度播放，1时等倍速，2时2倍速。还可以指定负数，指定负数的情况下，将逆向播放。

变速播放的同时，声音也会变速播放，但是到底能适应多少倍速的播放水平，与具体的浏览器有关，太快或者太慢的情况下，有的浏览器上可能播放不出声音。

可变速滑块相关代码如下所示。

```
$("#slider").slider({
    max : 4,        // 滑块的最大值
    min: 0,         // 滑块的最小值
    step : 0.05,    // 滑块的间隔
    value : 1,      // 滑块初始值
    change: function(evt, ui){
        videoObj.playbackRate = ui.value;    // 根据滑块值设置播放速度
        videoObj.defaultPlaybackRate = ui.value;
                                             // 根据滑块的默认值设置播放速度
        document.getElementById("speed").innerHTML = ui.value+"倍速";
    }
});
```

在$("#slider").slider()中进行滑块动作的设置，是jQuery中特有的代码编写方式，

将滑块处理以参数的形式设置到其中。当拖动滑块时将触发change事件。在change事件的处理中进行播放速度的设置。将拖动停止处的值ui.value设置到playbackRate以及defaultPlaybackRate属性中，即完成了改变播放速度的处理。

了解上述基本原理后，大家可以在本节实例的基础上开发有自己特色的视频播放器，有兴趣的朋友可以挑战一下。

图5-15是拖动滑块，改变播放速度的效果图。注意以下是在Google Chrome中运行的效果。本程序在FireFox（3.6版本以上）中还无法实现变速播放。

图5-15 改变视频播放速度

```
[JavaScript]HTML5 _ player/js/player.js
window.addEventListener("load", function(){
   // 时间设置
   var videoObj = document.getElementById("myVideo");
   var crtTime = document.getElementById("ctime");
   var pButton = document.getElementById("playButton");
   var rewButton = document.getElementById("rewindButton");
   // 设置播放按钮的事件
   pButton.addEventListener("click", function(){
          if (videoObj.paused == true){      // 停止时
                 videoObj.play();            // 播放
                 pButton.src = 'images/off.png';
          }else{
                 videoObj.pause();
```

```
                    pButton.src = 'images/on.png';
                }
        }, true);
        // 播放结束时触发的事件
        videoObj.addEventListener("ended", function(){
                videoObj.currentTime = 0;
                videoObj.pause();
                pButton.src = 'images/on.png';
        }, true);
        // 显示播放时间
        videoObj.addEventListener("timeupdate", function(){
                document.getElementById("ctime").innerHTML = "播放时间:"+Math.floor
(videoObj.currentTime*10)/10+"秒";
        }, true);
        // 设置回退按钮的事件
        rewButton.addEventListener("click", function(){
                videoObj.currentTime = 0;
        }, true);
        // 设置速度滑块
        $("#slider").slider({
                max : 4,        // 滑块的最大值
                min: 0,         // 滑块的最小值
                step : 0.05,    // 滑块的间隔
                value : 1,      // 滑块初始值
                change: function(evt, ui){
                        videoObj.playbackRate = ui.value;
                                // 根据滑块值设置播放速度
                        videoObj.defaultPlaybackRate = ui.value;
                                // 根据滑块的默认值设置播放速度
                        document.getElementById("speed").innerHTML = ui.value+"倍速";
                }
        });
    }, true);
```

Broadview
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

HTML5移动开发即学即用

书中几乎涵盖了HTML5规范中涉及的所有技术（还在酝酿中的Indexed Database除外），是您能找到的真正的HTML5技术大全。包含作者精心编写的，运行于常用PC浏览器以及智能手机上的应用实例，操作性与实用性俱佳，可以让您做到即学即用。讲解兼顾常用的PC浏览器以及当前流行的各种智能移动设备，作者丰富的移动开发经验会让你受益匪浅。双色印刷，既美观大方，又方便您的阅读。

关于作者

·王志刚· 1998年大学毕业后进入青岛海尔集团，经历过IT泡沫那个激动人心的时代。2000年后去日本工作，历经程序员、系统工程师、项目经理、开发部长等职。在十四年的开发生涯中，参加过日立、富士通等公司主导的大型项目的开发，获得过日本国专利的成绩。擅长各种移动开发技术，在十多年的工作中，积累了不少大型项目的开发经验，并不断与他人分享。

·王中元· 博士，武汉大学计算机学院副教授，硕士研究生导师。
研究方向为多媒体信息处理和网络通信，承担科研课题10余项，其中主持两项国家自然科学基金，是973项目、新一代宽带无线移动通信网国家重大科技专项的主要学术骨干。在包含IEEE Trans期刊和多媒体领域顶级国际会议的刊物上发表论文30余篇，16篇被EI索引。申报发明专利10项，3项授权。向国家数字音视频编解码技术标准工作组提交技术提案9项，3项被采纳，是公安部城市监控报警联网系统系列标准中“关键设备通用技术要求”分标准的执笔人。担任知名国际会议PCM2009、ICME2012的程序委员会委员，获得教育部科技进步奖、湖北省科技进步奖等省部级奖励6项。2010、2009、2008连续三个年度被授予武汉大学“优秀员工”称号。

·江友华· 博士后，上海电力学院计算机与信息工程学院副院长，副教授。
从事电力企业信息化、数据分析与软件设计方面的教学和研究工作。主持和承担过多项上海市重点及创新项目，近年来在国内外期刊发表30多篇论文，其中EI检索10篇。此外，还申请了多项专利及软件著作权。

本书相关实例代码下载地址及本书服务网站：
http://www.softechallenger.com/download/?ISBN=978-7-121-15685-4

上架建议：计算机 > HTML5

ISBN 978-7-121-15685-4
9 787121 156854 >
定价：59.00元

策划编辑：孙学瑛
责任编辑：葛　娜
封面设计：侯士卿