HTML5开发教程

专 业 的 I T 培 训 专 家

顾 翔

HTML5开发教程

- HTML 5的新特性
- HTML 5与HTML 4的区别
- HTML 5的结构
- HTML 5中的表单
- HTML 5中的文件与拖放
- 多媒体播放
- 绘制图形
- 数据存储
- 获取地理位置信息
- 练习

目录

HTML 5的新特性

- 谁在开发HTML 5
- HTML 5的新认识
- 无插件范式
- HTML 5的新特性

目录

谁在开发HTML 5

2014年10月29日，万维网联盟泪流满面地宣布，经过几乎8年的艰辛努力，HTML5标准规范终于最终制定完成了，并已公开发布。

在此之前的几年时间里，已经有很多开发者陆续使用了HTML5的部分技术，Firefox、Google Chrome、Opera、Safari 4+、Internet Explorer 9+都已支持HTML5，但直到今天，我们才看到“正式版”。

HTML5将会取代1999年制定的HTML 4.01、XHTML 1.0标准，以期能在互联网应用迅速发展的时候，使网络标准达到符合当代的网络需求，为桌面和移动平台带来无缝衔接的丰富内容。

HTML 5的新特性

- 谁在开发HTML 5
- HTML 5的新认识
- 无插件范式
- HTML 5的新特性

目录

HTML5的新认识

本节需要掌握的知识

1、兼容性
2、实用性和用户优先
3、化繁为简

HTML5的新认识

概述

任何新鲜事物的出现，都会带给人们惊喜，同时也会存在很多争议。虽然Web开发者普遍认为有了HTML5是比较好的，但是还是会很担心的，例如：新的HTML5在老的浏览器上能否正常运行，会不会产生错误等各种问题。HTML5是基于各种各样的理念进行设计的，这些设计理念体现了对可能行和可行性的新认识。

HTML5的新认识

✓ 兼容性

- 对HTML以前的版本开发的网站做了很好的兼容
- HTML5的一个核心理念就是保持一切新特性平滑过渡

✓ 实用性和用户优先

- HTML5规范是基于用户优先准则编写的
- HTML5内只封装了切实有用的功能，不封装复杂而没有实际意义的功能

✓ 化繁为简

- 以浏览器原生能力替代复杂的JavaScript代码。
- 新的简化的DOCTYPE
- 新的简化的字符集声明
- 简单而强大的HTML5API。

HTML 5的新特性

- 谁在开发HTML 5
- HTML 5的新认识
- 无插件范式
- HTML 5的新特性

目录

无插件范式

插件的副作用：

- 插件安装可能失败；
- 插件可能被禁用或屏蔽
- 插件自身会成为被攻击的对象
- 插件不容易与HTML文档其他部分集成

HTML 5的新特性

- 谁在开发HTML 5
- HTML 5的新认识
- 无插件范式
- HTML 5的新特性

目录

HTML5的新特性

本节需要掌握的知识

1、HTML5的新特性

HTML5的新特性

- 新特性应该基于HTML、CSS、DOM和JavaScript。
- 减少了对外部插件的需求（比如Flash）。
- 更优秀的错误处理。
- 更多取代脚本的标记。
- HTML5应该独立于设备。
- 用于绘画的canvas元素。
- 用于媒介回放的video和audio元素。
- 对本地离线存储的更好的支持。
- 新元素和表单控件。

而这些新特性，正在如今的浏览器最新版本中得到越来越普遍的实现，越来越多的开发者开始学习和使用这些新特性。

HTML5开发教程

- HTML 5的新特性
- HTML 5与HTML 4的区别
- HTML 5的结构
- HTML 5中的表单
- HTML 5中的文件与拖放
- 多媒体播放
- 绘制图形
- 数据存储
- 获取地理位置信息
- 练习

目录

HTML 5与HTML 4的区别

- 语法的改变
- 新增的元素和废除的元素
- 新增的属性和废除的属性
- 全局属性

目录

语法的改变

本节需要掌握的知识

1、HTML5的语法变化
2、HTML5中的标记方法
3、HTML5语法中需要掌握的3个要点
4、标记示例

语法的改变

HTML5中的标记方法

✓ 内容类型（ContentType）

HTML5文件的扩展名和内容类型（ContentType）没有发生变化。即扩展名还是“.html”或“.htm”，内容类型（ContentType）还“.text/html”。

✓ DOCTYPE声明

要使用HTML5标记，必须先进行如下的DOCTYPE声明。不区分大小写。语法如下：

```
<!DOCTYPE html>
```

另外，当使用工具时，也可以在DOCTYPE声明方式中加入SYSTEM标识。（不区分大小写。此外还可将双引号换为单引号来使用），声明方法如下面的代码：

```
<!DOCTYPE HTML SYSTEM "about:legacy-compat">
```

语法的改变

HTML5中的标记方法

✓ 字符编码的设置

字符编码的设置方法也有些新的变化。以前，设置HTML文件的字符编码时，要用到如下<meta>元素，如下所示：

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

在HTML5中，可以使用<meta>元素的新属性charset来设置字符编码。

```
<meta charset="UTF-8">
```

以上两种方法都有效。因此也可以继续使用前者的方法（通过content元素的属性来设置）。但要注意不能同时使用。如下所示：

```
<meta charset="UTF-8" http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

注意：从HTML5开始，文件的字符编码推荐使用UTF-8。

语法的改变

HTML5语法中需要掌握的3个要点

HTML5中规定的语法，在设计上兼顾了与现有HTML之间最大程度的兼容性。例如，在Web上充斥着“<p>没有结束标签”等HTML现象。HTML5不将这些视为错误，反而采取了“允许这些现象存在，并明确记录在规范中”的方法。因此，尽管与XHTML相比标记比较简洁，而在遵循HTML5的Web浏览器中也能保证生成相同的DOM。 那么下面就来看看具体的HTML5语法。

语法的改变

1)可以省略标签的元素

在HTML5中，有些元素可以省略标签。具体来讲有3种情况，

- 不允许写结束标记的元素有

area、base、br、col、command、embed、hr、img、input、keygen、link、meta、param、source、track、

wbr

不允许写结束标记的元素是指，不允许使用开始标记与结束标记将元素括起来的的形式，只允许使用“”<元素/>”的形式进行书写。例如：“
...</br>”的写法是错误的。应该写成“
”。当然，沿袭下来的“
”这种写法也是允许的。

语法的改变

1)可以省略标签的元素

- 可以省略结束标签

li、dt、dd、p、rt、rp、optgroup、option、colgroup、thead、tbody、tfoot、tr、td、th

- 可以省略整个标签（即连开始标签都不用写明）

html、head、body、colgroup、tbody需要注意的是，虽然这些元素可以省略，但实际上却是隐式存的例如“<body>”标签可以省略，但在DOM树上它是存在的，可以永恒访问“document.body”。上述元素中也包括了HTML5的新元素。有关这些新元素的用法，将在后面详细讲解。

语法的改变

2)取得boolean值的属性

取得布尔值（Boolean）的属性，例如disabled和readonly等，通过省略属性的值来表达“值为true”。如果要表达“值为false”，则直接省略属性本身即可。此外，在写明属性值来表达“值为true”时，可以将属性值设为属性名称本身，也可以将值设为空字符串。如下列所示：

```
<!-- 以下的checked属性值皆为true -->
<input type="checkbox" checked>
<input type="checkbox" checked="checked">
<input type="checkbox" checked="">
```

语法的改变

3)省略属性的引用符

设置属性值时，可以使用双引号或单引号来引用。HTML5语法则更进一步，只要属性值不包含空格、“<”、“>”、“'”、“"”、“`”、“=”等字符，都可以省略属性的引用符。如下例所示。

```
<!—请注意type属性的引用符 -->

<input type="text">

<input type='text'>

<input type=text>
```

HTML 5与HTML 4的区别

- 语法的改变
- 新增的元素和废除的元素
- 新增的属性和废除的属性
- 全局属性

目录

新增的元素和废除的元素

本节需要掌握的知识

1、新增的结构元素
2、新增的块级（block）的语义元素
3、新增的行内（inline）的语义元素
4、新增的嵌入多媒体元素与交互性元素
5、新增的input元素的类型
5、废除的元素

新增的元素和废除的元素

新增的结构元素

- **section元素**

section元素定义文档或应用程序中的一个区段,比如章节、页眉、页脚或文档中的其他部分。它可以与h1,h2,h3,h4,h5,h6元素结合起来使用，标示文档结构。

HTML5中代码示例：

```
<section>...</section>
```

HTML4中代码示例：

```
<div>...</div>
```

- **article元素**

article元素表示文档中的一块独立的内容，譬如博客中的一篇文章或报纸中的一篇文章。

HTML5中代码示例：

```
<article>...</article>
```

HTML4中代码示例：

```
<div class="article">...</div>
```

新增的元素和废除的元素

新增的结构元素

- **header元素**

header元素表示页面中一个内容区块或整个页面的标题。HTML5中代码示例：

```
<header>...</header>
```

HTML4中代码示例：

```
<div>...</div>
```

- **nav元素**

nav元素表示导航链接的部分。

HTML5中代码示例：

```
<nav>...</nav>
```

HTML4中代码示例：

```
<ul>...</ul>
```

新增的元素和废除的元素

新增的结构元素

- **footer元素**

footer元素表示整个页面或页面中一个内容区块的脚注。一般来说，它会包含创作者的姓名、文档的创作日期以及创建者联系信息。

HTML5中代码示例：

```
<footer>…</footer>
```

HTML4中代码示例：

```
<div>…</div>
```

新增的元素和废除的元素

新增的块级（block）的语义元素

- **aside元素**

aside元素表示article元素的内容之外的与article元素的内容相关的有关内容。

HTML5中代码示例：

```
<aside>...</aside>
```

HTML4中代码示例：

```
<div>...</div>
```

新增的元素和废除的元素

新增的块级（block）的语义元素

➢figure元素

figure元素表示一段独立的流内容，一般表示文档主体流内容中的一个独立单元。使用 <figcaption> 元素为figure元素组添加标题。

HTML5中代码示例：

```
<figure>
<figcaption>PRC</figcaption>
<p>The People's Republic of China was born in 1949...</p>
</figure>
```

HTML4中代码示例：

```
<dl>
<h1>PRC</h1>
<p>The People's Republic of China was born in 1949...</p>
</dl>
```

新增的元素和废除的元素

新增的块级（block）的语义元素

- **dialog元素**

  dialog标签定义对话，比如交谈。注意：对话中的每个句子都必须属于 <dt> 标签所定义的部分。

HTML5中代码示例：

```
<dialog>
<dt>老师</dt>
<dd>2+2 等于？ </dd>
<dt>学生</dt>
<dd>4</dd>
<dt>老师</dt>
<dd>答对了！ </dd>
</dialog>
```

新增的元素和废除的元素

新增的行内（inline）的语义元素

- **mark元素**

mark元素主要用来在视觉上向用户呈现那些需要**突出显示或高亮显示的文字**。mark元素的一个比较典型的应用就是在搜索结果中向用户高亮显示**搜索关键词**。

HTML5中代码示例：

```
<mark>...</mark>
```

HTML4中代码示例：

```
<span>...</span>
```

新增的元素和废除的元素

新增的行内（inline）的语义元素

- time元素

time元素表示**日期或时间**，也可以同时表示两者。

HTML5中代码示例：

<time>…</time>

HTML4中代码示例：

…

新增的元素和废除的元素

新增的行内（inline）的语义元素

- meter元素

meter元素表示度量衡。仅用于已知最大和最小值的度量。必须定义度量的范围，既可以在元素的文本中，也可以在 min/max 属性中定义。

HTML5中代码示例：

<meter>...</meter>

新增的元素和废除的元素

新增的行内（inline）的语义元素

- **progress元素**

progress元素表示运行中的进程。可以使用 progress元素来显示JavaScript 中耗费时间的函数的进程。

HTML5中代码示例：

```
<progress>...</progress>
```

新增的元素和废除的元素

新增的嵌入多媒体元素与交互性元素

➢ details元素

details元素表示用户要求得到并且可以得到的细节信息。它可以与summary元素配合使用。summary元素提供标题或图例。标题是可见的，用户点击标题时，会显示出details。summary元素应该是details元素的第一个子元素。

HTML5中代码示例：

```
<details><summary>HTML 5</summary>
This document teaches you everything you have to learn about HTML 5.
</details>
```

新增的元素和废除的元素

新增的嵌入多媒体元素与交互性元素

- **datagrid元素**

datagrid元素表示可选数据的列表，与input元素配合使用，可以制作出输入值的下拉列表。

HTML5中代码示例：

```
<datagrid>…</datagrid>
```

新增的元素和废除的元素

新增的嵌入多媒体元素与交互性元素

➢menu元素

menu元素表示菜单列表。当希望列出表单控件时使用该标签。

HTML5中代码示例：

```
<menu>
<li><input type="checkbox" />Red</li>
<li><input type="checkbox"/>blue</li>
</menu>
```

📣注意：HTML4中 menu元素不被推荐使用。

新增的元素和废除的元素

新增的嵌入多媒体元素与交互性元素

- **command元素**

command元素表示命令按钮，比如单选按钮、复选框或按钮。

HTML5中代码示例：

```
<command onclick=cut()" label="cut">
```

新增的元素和废除的元素

新增的input元素的类型

HTML5中，新增了很多input元素的类型，现列举如下：

- ✓email

  email类型用于应该包含e-mail 地址的输入域。

- ✓url

  url 类型用于应该包含URL 地址的输入域。

- ✓number

  number 类型用于应该包含数值的输入域。

- ✓range

  range 类型用于应该包含一定范围内数字值的输入域。

- ✓Date Pickers（数据检出器）

- ✓search

  search 类型用于搜索域，比如站点搜索或Google 搜索。search 域显示为常规的文本域。

新增的元素和废除的元素

新增的input元素的类型

HTML5 拥有多个可供选取日期和时间的新输入类型

- date - 选取日、月、年
- month - 选取月、年
- week - 选取周和年
- time - 选取时间（小时和分钟）
- datetime - 选取时间、日、月、年（UTC 时间）
- datetime-local - 选取时间、日、月、年（本地时间）

新增的元素和废除的元素

废除的元素

由于各种原因，在HTML5中废除了很多元素，下面简单介绍一下被废除的元素。

- 1.能使用css代替的元素

对于basefont、big、center、font、s、strike、tt、u这些元素，由于他们的功能都是纯粹为画面展示服务的，而在HTML5中提倡把画面展示性功能放在css样式表中统一编辑，所以将这些元素废除，并使用编辑css样式表的方式进行替代。

- 2.不在使用frame框架

对于frameset元素、frame元素与nofranes元素，由于frame框架对页面可以性存在负面影响，在html5中已不再支持frame框架，只支持iframe框架，或者用服务器方创建的由多个页面组成的复合页面的形式，同时将以上三个元素废除。

- 3.只有部分浏览器支持的元素

对于applet、bgsound、blink、marguee等元素，由于只有部分浏览器支持这些元素，所以在HTML5中被废除。其中applet元素可由embed元素替代，bgsound元素可由audio元素替代，marquee可以由JavaScript编程的方式所替代。

HTML 5与HTML 4的区别

- 语法的改变
- 新增的元素和废除的元素
- 新增的属性和废除的属性
- 全局属性

目录

新增的属性和废除的属性

本节需要掌握的知识

1、新增的属性
2、废除的属性

新增的属性和废除的属性

新增的属性

1.表单相关的属性

- **autocomplete 属性**

  autocomplete 属性规定 form 或 input 域应该拥有自动完成功能。

  注释

  autocomplete 适用于 <form> 标签，以及以下类型的 <input> 标签：text, search, url, telephone, email, password, datepickers, range 以及color

- **autofocus 属性**

  autofocus 属性规定在页面加载时，域自动地获得焦点。

  注释

  autofocus 属性适用于所有 <input> 标签的类型。

新增的属性和废除的属性

新增的属性

- **form 属性**

  form 属性规定输入域所属的一个或多个表单。

  注释

  form 属性适用于所有 <input> 标签的类型。

- **表单重写属性**

  表单重写属性（form override attributes）允许您重写form 元素的某些属性设定。表单重写属性有：

- formaction - 重写表单的 action 属性
- formenctype - 重写表单的 enctype 属性
- formmethod - 重写表单的 method 属性
- formnovalidate - 重写表单的 novalidate 属性
- formtarget - 重写表单的 target 属性

注释

表单重写属性适用于以下类型的 <input> 标签：submit 和 image。

新增的属性和废除的属性

新增的属性

- **height 和 width 属性**

  height 和 width 属性规定用于 image 类型的 input 标签的图像高度和宽度。

  注释

  height 和 width 属性只适用于 image 类型的 <input> 标签。

- **list 属性**

  list 属性规定输入域的 datalist。datalist 是输入域的选项列表。

  注释

  list 属性适用于以下类型的 <input> 标签：text, search, url, telephone, email, date pickers, number, range 以及 color。

新增的属性和废除的属性

新增的属性

- **min、max 和 step 属性**

min、max 和 step 属性用于为包含数字或日期的 input 类型规定限定（约束）。

✓max 属性规定输入域所允许的最大值。

✓min 属性规定输入域所允许的最小值。

✓step 属性为输入域规定合法的数字间隔（如果 step="3"，则合法的数是 -3,0,3,6 等）。

注释

min、max 和 step 属性适用于以下类型的 <input> 标签：date pickers、number 以及 range。

- **multiple 属性**

multiple 属性规定输入域中可选择多个值。

注释

multiple 属性适用于以下类型的 <input> 标签：email 和 file

新增的属性和废除的属性

新增的属性

- **novalidate 属性**

  novalidate 属性规定在提交表单时不应该验证 form 或 input域

注释

novalidate 属性适用于 **<form> 以及以下类型的<input>**

标签：text, search, url, telephone, email, password, date pickers, range 以及 color.

- **pattern 属性**

  pattern 属性规定用于验证 input 域的模式（pattern）。模式（pattern） 是正则表达式。您可以在我们的 JavaScript 教程中学习到有关正则表达式的内容。

注释

pattern 属性适用于以下类型的 **<input> 标签：text, search, url, telephone, email 以及 password**。

新增的属性和废除的属性

新增的属性

- **placeholder 属性**

  placeholder 属性提供一种提示（hint），描述输入域所期待的值。

注释

placeholder 属性适用于以下类型的 <input> 标签：text, search, url, telephone, email 以及 password。

- **required 属性**

  required 属性规定必须在提交之前填写输入域（不能为空）

注释

required 属性适用于以下类型的 <input> 标签：text, search, url, telephone, email, password, date pickers, number, checkbox, radio 以及 file。

新增的属性和废除的属性

新增的属性

2.链接相关属性

新增的与链接相关的属性如下：

- **media属性**

  为a与area元素增加了media属性，该属性规定目标URL是为什么类型的媒介/设备进行优化的。只能在 href 属性存在时使用。

- **hreflang属性与rel属性**

  为area元素增加了hreflang属性与rel属性，以保持与a元素，link元素的一致。

- **Sizes属性**

  为link元素增加了新属性sizes。该属性可以与icon元素结合使用（通过rel属性），该属性指定关联图标（icon元素）的大小。

新增的属性和废除的属性

新增的属性

- **target属性**

为base元素增加了target属性，主要目的是保持与a元素的一致性，同时target元素由于在Web应用程序中，尤其是在与iframe结合使用时，是非常有用的，所以不再是不赞成使用的元素了。

3.其他属性

除了上面介绍的与表单和链接相关的属性外，HTML5还增加了下面的属性:

- **reversed属性**

为ol元素增加属性reversed，它指定列表倒序显示。li元素的value属性与ol元素的start属性因为它不是被显示在界面上的，所以不再是不赞成使用的了。

新增的属性和废除的属性

新增的属性

- **charset属性**

  为meta元素增加charset属性，因为这个属性已经被广泛支持了，而且为文档的字符编码的指定提供了一种比较良好的方式。

- **type属性与label属性**

  为menu元素增加了两个新的属性type与label。label属性为菜单定义一个可见的标注，type属性让菜单可以以上下文菜单，工具条，与列表菜单三种形式出现。

- **scoped属性**

  为style元素增加scoped属性，用来规定样式的作用范围，譬如只对页面上某个树起作用。
  为script元素增加async属性，它定义脚本是否异步执行。

新增的属性和废除的属性

新增的属性

- **manifest属性**

为html元素增加属性manifest，开发离线Web应用程序时他与API结合使用，定义一个URL，在这个URL上描述文档的缓存信息。为iframe元素增加三个属性sandbox, seamless与srcdoc,用来提高页面安全性，防止不信任的Web页面执行某些操作。

新增的属性和废除的属性

废除的属性

HTML4中的一些属性在HTML5中不再被使用，而是采用其他属性或其他方案进行替换，具体如下表所示。

在HTML4中使用的属性	使用该属性的元素	在HTML5中的替代方案
rev	link、a	rel
charset	link、a	在被链接的资源的中使用HTTP Content-type头元素
shape、coords	a	使用area元素代替a元素
longdesc	img、iframe	使用a元素链接到较长描述
target	link	多余属性，被省略

新增的属性和废除的属性

废除的属性

nohref	area	多余属性，被省略
profile	head	多余属性，被省略
version	html	多余属性，被省略
name	img	id
scheme	meta	只为某个表单域使用scheme
archive、classid、codebase、codetype、declare、standby	object	使用data与type属性类调用插件。需要使用这些属性来设置参数时，使用param属性

新增的属性和废除的属性

废除的属性

valuetype、type	param	使用name与value属性，不声明值的mime类型
axis、abbr	td、th	使用以明确简洁的文字开头、后跟详述文字的形式。可以对更详细内容使用title属性，来使单元格的内容变得简短
scope	td	在被链接的资源的中使用HTTP content-type头元素
align	caption、input、legend、div、h1、h2、h3、h4、h5、h6、p	使用CSS样式表替代
alink、link、text、vlink、background、bgcolor	body	使用CSS样式表替代

新增的属性和废除的属性

废除的属性

align、bgcolor、border、cellpadding、cellspacing、frame、rules、width	table	使用CSS样式表替代
align、char、charoff、height、nowrap、valign	td、th	使用CSS样式表替代
align、bgcolor、char、charoff、valign	tr	使用CSS样式表替代
align、char、charoff、valign、width	col、colgroup	使用CSS样式表替代
align、border、hspace、vspace	object	使用CSS样式表替代

新增的属性和废除的属性

废除的属性

clear	br	使用CSS样式表替代
compact、type	ol、ul、li	使用CSS样式表替代
compact	dl	使用CSS样式表替代
compact	menu	使用CSS样式表替代
width	pre	使用CSS样式表替代
align、noshade、size、width	hr	使用CSS样式表替代
align、frameborder、scrolling、arginwidth	iframe	使用CSS样式表替代
autosubmit	menu	
align、hspace、vspace	img	使用CSS样式表替代

HTML 5与HTML 4的区别

- 语法的改变
- 新增的元素和废除的元素
- 新增的属性和废除的属性
- 全局属性

目录

全局属性

本节需要掌握的知识

1、contentEditable属性
2、designMode属性
3、hidden属性
4、spellcheck属性
5、tabindex属性

全局属性

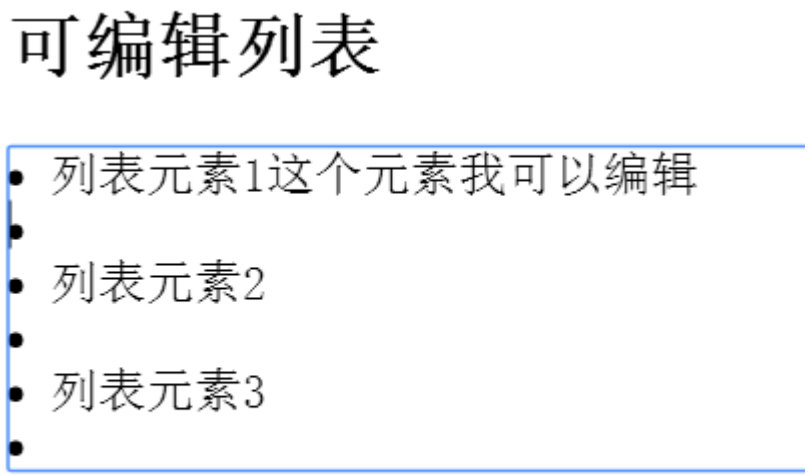
contentEditable属性

- 功能：允许用户编辑元素中的内容。
- 功能说明：该元素必须是可以获得鼠标焦点的元素，而且在点击鼠标后要向用户提供一个插入符号，提示用户该元素中的内容允许编辑。
- 注意： contentEditable是一个布尔类型属性，因此可以将其设置为true或false。
- 说明：该属性还有个隐藏的inherir（继承）状态，属性为true时，元素被指定位允许编辑；属性为false时，元素被指定为不允许编辑；未指定true或false时，则由inherir状态来决定，如果元素的父元素是可编辑的，则该元素就是可编辑的。另外，除了contentEditable属性外，元素还具有一个iscontentEditable属性，当元素可编辑时，该属性为true；当元素不可编辑时，该属性为false。

全局属性

contentEditable属性

```
<!doctype html>
<head>
<meta charset=“utf-8”>
<title>contentEditable属性示例</title>
</head>
<h2>可编辑列表</h2>
<ui contentEditable=true>
<li>列表元素1<li>
<li>列表元素2<li>
<li>列表元素3<li>
</ui>
```



2-1.html

全局属性

designMode属性

- 功能：用来指定整个页面是否可编辑，当页面可编辑时，页面中任何支持上文所述的ontentEditable属性的元素都变成了可编辑状态。 IE8不允许，IE9允许

注意：designMode属性只能在JavaScript脚本里被编辑修改。

- designMode属性的两个值：该属性有两个值——“on”与“off”。当属性被指定为“on”时，页面可编辑；被指定为“off”时，页面不可编辑。

- 使用JavaScript脚本来指定designMode属性的方法如下所示document.designMode=”on”。

全局属性

hidden属性

- 适用范围：在HTML5中，所有的元素都允许有一个hidden 。
- 作用：hidden属性类似于aria-hidden，它告诉浏览器这个元素的内容不应该以任何方式显示。但是元素中的内容还是浏览器创建的，也就是说页面装载后允许使用JavaScript脚本将该属性取消，取消后该元素变为可见状态，同时元素中的内容也即时显示出来。
- 类型：Hidden属性是一个布尔值的属性，当设为true时，元素处于不可见状态；当设为false时，元素处于可见状态。

全局属性

spellcheck属性

- 适用范围：spellcheck属性 是HTML5针对input元素text与textarea这两个文本输入框提供的一个新属性
- 功能：为对用户输入的文本内容进行拼写和语法检查。
- 类型：Spellcheck属性是布尔型，具体true和false两种值，但是它在书写时有一个特殊的地方，就是必须明确声明属性值为true或false，书写方法如下所示：

```
<!--以下两种书写方法正确--!>
<textarea spellcheck="true">
<input type=text spellcheck=false />
<!--以下书写方法为错误--!>
< textarea  spellcheck >
```

- 注意：如果元素的readOnly属性或disabled属性设为true，则不执行拼写检查。

全局属性

tabindex属性

tabindex是开发中的一个基本概念，当不断敲击Tab键让窗口或页面中的控件获得焦点，对窗口或页面中的所有控件进行遍历的时候，每一个控件的tabindex 表示该控件是第几个被访问到的。

tabindex属性还有另外一个作用。在默认情况下，只有链接、表单元素和图像映射区域可以通过键盘获得焦点。如果对其他元素使用tabindex属性后，也能让改元素获得焦点，那么当脚本中执行focus()语句的时候，就可以让该元素获得焦点了。但是这样做会产生一个副作用：该元素也可以通过Tab键获得焦点，而这又时可能不是开发者想要的结果。把元素tabindex值设为负数（通常为-1）后就解决这个问题了。

tabindex的值为负数后，仍然可以通过编程的方式让元素获得焦点，但按下Tab键时该元素就不能获得焦点了，这在复杂的页面中或复杂的Web应用程序中时十分有用的。在HTML4中，-1是一个无用的属性值，但到了HTML5中，通过巧妙地运用让改属性值得到了极大地应用。

HTML5开发教程

目录

HTML 5的结构

- 新增的主体结构元素
- 新增的非主体结构元素

目录

新增的主体结构元素

本节需要掌握的知识

1、article元素
2、section元素
3、nav元素
4、aside元素
5、time元素
6、pubdate属性

新增的主体结构元素

article元素

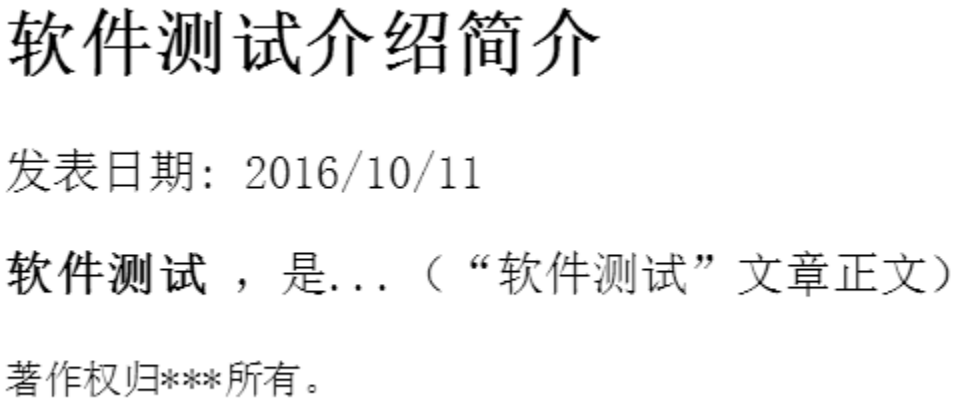
- article元素表示文档、页面或应用程序中独立的、完整的、可以独自被外部引用的内容。它可以是一篇博客或报刊中的文章、一篇论坛帖子、一段用户评论或独立的插件，或者他任何独立的内容。
- 除了内容部分，一个article元素通常有它自己的标题（一般放在一个header元素里面），有时还有自己的脚注。
- article元素是可以嵌套使用的，内层的内容在原则上需要与外层的内容相关联。例如，一篇博客文章中，针对该文章的评论就可以使用嵌套article元素的方式；用来呈现评论的arcticle元素被包含在表示整体内容的article元素里面。
- article元素也可以用来表示插件，它的作用是使插件看起来好像内嵌在页面中一样。

新增的主体结构元素

article元素

3-1.html

```
<article>
  <header>
    <h1>软件测试介绍简介</h1>
    <p>发表日期: <time pubdate="pubdate">2016/10/11</time></p>
  </header>
  <p><b>软件测试</b>，是...（“软件测试”文章正文）</p>
  <footer>
    <p><small>著作权归***所有。</small></p>
  </footer>
</article>
```



新增的主体结构元素

article元素

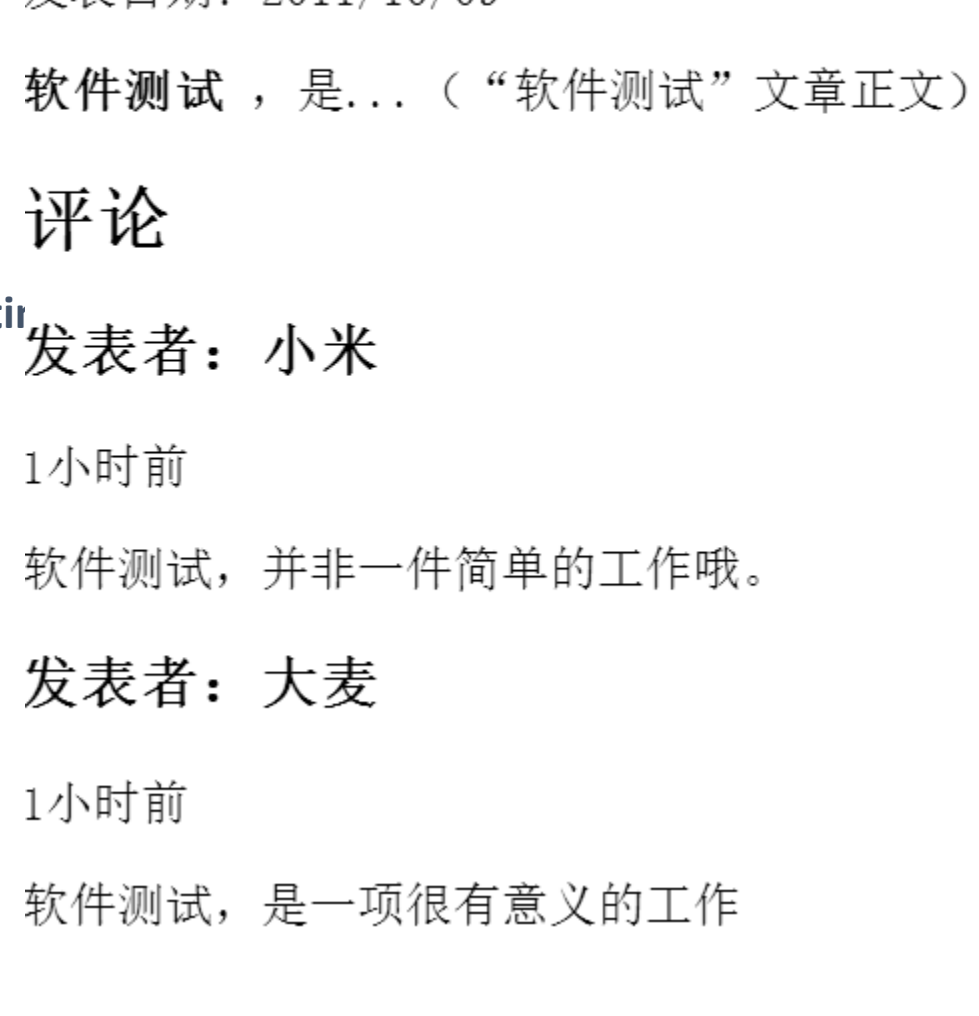
3-2.html

```
<article>
  <header>
    <h1>软件测试简介</h1>
    <p>发表日期:
      <time pubdate datetime="2010/10/09">2011/10/09</time>
    </p>
  </header>
  <p><b>软件测试</b>，是...（“软件测试”文章正文）</p>
  <section>
    <h2>评论</h2>
    <article>
      <header>
        <h3>发表者：小米</h3>
        <p><time pubdate datetime="2011-10-10T19:10-08:00">1小时前</time></p>
      </header>
```

新增的主体结构元素

article元素

```
<p>软件测试，并非一件简单的工作哦。</p>
    </article>
    <article>
      <header>
        <h3>发表者：大麦</h3>
        <p><time pubdate datetime="2011-10-10T19:15-08:00">1小时前</ti
      </header>
      <p>软件测试，是一项很有意义的工作</p>
    </article>
  </section>
</article>
```



新增的主体结构元素

article元素

3-3.html

```
<!DOCTYPE html><head>
<meta charset="utf-8 ">
<title>article元素表示插件</title>
</head>

<article>
   <h1>电子商务网站</h1>
   <object>
      <param name="allowFullScreen" value="true">
      <embed src="z/aa.avi"width="400" height="295"><
   </object>
</article>
```

电子商务网站

新增的主体结构元素

section元素

- ✓ Section元素用于对网站或应用程序中页面上的内容进行分块。一个section元素通常由内容及其标题组成。但section元素并非一个普通的容器元素；当一个容器需要被直接定义样式或通过脚本定义行为时，推荐使用div而非section元素。
- ✓ article元素与section元素的区别

在HTML5中，article元素可以看成是一种特殊种类的section元素，它比section元素更强调独立性。即section元素强调分段或分块，而article强调独立性。总结来说，如果一块内容相对来说比较独立、完整的时候，应该使用article元素，但是如果你想将一块内容分成几段的时候，应该使用section元素。

新增的主体结构元素

section元素

注意：在HTML5中，div元素变成了一种容器，当使用CSS样式的时候，可以对这个容器进行一个总替CSS样式的套用。

对section元素的注意事项进行总结：

- 不要将section元素用作设置样式的页面容器，那是div元素的工作。
- 当article元素、aside元素或nav元素更符合页面要求时，尽量不要使用section。
- 不要为没有标题的内容区块使用section元素。

新增的主体结构元素

section元素

3-4.html

```
<article>
<h1>葡萄</h1>
<p><b>葡萄</b>，植物类水果...</p>
<section>
    <h2>巨峰</h2>
    <p>欧美杂交,为四倍体葡萄品种...</p>
</section>
<section>
    <h2>刺霞珠</h2>
    <p>本身带有黑加仑、黑莓子等香味</p>
</section>
</article>
```

葡萄

葡萄，植物类水果…

巨峰

欧美杂交, 为四倍体葡萄品种…

刺霞珠

本身带有黑加仑、黑莓子等香味

新增的主体结构元素

section元素

3-5.html

```
<section>
    <h1>水果</h1>
    <article>
        <h2>苹果</h2>
        <p>苹果，植物类水果，多刺花果...</p>
    </article>
    <article>
        <h2>橘子</h2>
        <p>橘子，是芸香科柑橘属的一种水果...</p>
    </article>
    <article>
        <h2>香蕉</h2>
        <p>香蕉，属于芭蕉科芭蕉属植物，又指其水果...</p>
    </article>
</section>
```

水果

苹果

苹果，植物类水果，多刺花果…

橘子

橘子，是芸香科柑橘属的一种水果…

香蕉

香蕉，属于芭蕉科芭蕉属植物，又指其水果…

新增的主体结构元素

nav元素

Nav元素是一个可以用做页面导航的链接组，其中的导航元素链接到其他页面或当前页面的其他部分。并不是链接的每一个集合都是一个nav，只需要将主要的、基本的链接组放进nav元素即可。例如，在页脚中通常会有一组链接，包括服务条款、版权声明、联系方式等。对于这些footer元素就足够放置了。一个页面中可以拥有多个nav元素，作为页面整体或不同部分的导航。

新增的主体结构元素

nav元素

Nav元素是一个可以用做页面导航的链接组，其中的导航元素链接到其他页面或当前页面的其他部分。并不是链接的每一个集合都是一个nav，只需要将主要的、基本的链接组放进nav元素即可。例如，在页脚中通常会有一组链接，包括服务条款、版权声明、联系方式等。对于这些footer元素就足够放置了。一个页面中可以拥有多个nav元素，作为页面整体或不同部分的导航。

新增的主体结构元素

nav元素

```
<!DOCTYPE html>
<head>
<meta charset="utf-8 ">
<title>nav元素示例</title>
</head>
<body>
<h1>软件测试简介</h1>
<nav>
  <ul>
    <li><a href="/">主页</a></li>
    <li><a href="/mr">简介文档</a></li>
    ...more...
  </ul>
</nav>
<article>
```

```
<header>
  <h1>软件测试介绍</h1>
  <nav>
    <ul>
      <li><a href="#gl">手工测试</a></li>
      <li><a href="#kf">工具测试</a></li>
      ...more...
    </ul>
  </nav>
</header>
<section id="rum">
  <h1>手工测试</h1>
  <p>探索式测试</p>
</section>
<section id="kf">
  <h1>工具测试</h1>
  <p>LoadRunner</p>
</section>
...more...
<footer>
```

3-6.html

新增的主体结构元素

nav元素

```
 <p>
        <a href="?edit">编辑</a> |
        <a href="?delete">删除</a> |
        <a href="?rename">重命名</a>
      </p>
   </footer>
</article>
<footer>
   <p><small>版权所有：啄木鸟软件测试网</small></p>
</footer>
</body>
```

软件测试简介

- 主页
- 简介文档

...more...

软件测试介绍

- 手工测试
- 工具测试

...more...

手工测试

探索式测试

工具测试

LoadRunner

...more...

编辑 | 删除 | 重命名

版权所有：啄木鸟软件测试网

新增的主体结构元素

nav元素的应用场合

Nav元素可以用于以下这些场合：

- ✓ **传统导航条**：现在主流网站上都有不同层级的导航条，其作用是将当前页面跳转到网站的其他主要页面上去。
- ✓ **侧边框导航**：现在主流博客网站及商品网站上都有侧边栏导航，其作用是将页面从当前文章或当前商品跳转到其他文章或其他商品页面上去
- ✓ **页内导航**：页内导航的作用是在本页面几个主要的组成部分之间进行跳转
- ✓ **翻页操作**：翻页操作是指在多个页面的前后页或博客网站的前后篇文章滚动。

注意：**在HTML5中不要用menu元素代替nav元素**。因为menu元素是用在一系列发出命令的菜单上的，是一种**交互性的**元素，或者更确切地说是使用**在web应用程序中的**。

新增的主体结构元素

aside元素

aside元素用来表示当前页面或文章的附属信息部分，它可以包含与当前页面或主要内容相关的引用、侧边栏、广告、导航条，以及其他类似的有别于主要内容的部分。

aside元素主要有以下两种使用方法

（1）被包含在article元素中作为主要内容的附属信息部分，其中的内容可以是与当前文章有关信息、名词解释，等等。

（2）在article元素之外使用，可以作为页面或站点全局的附属信息部分。最典型的形式就是侧边栏，其中的内容可以是友情链接，博客中其他文章列表、广告单元等。

新增的主体结构元素

aside元素　　3-7.html

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>文章内部的aside元素示例</title>
</head>
<body>
<header>
  <h1>宋词赏析</h1>
</header>
<article>
  <h1><strong>水调歌头</strong></h1>
  <p>...但愿人长久，千里共婵娟(文章正文）</p>
  <aside>
    <!-- 因为这个aside元素被放置在一个article元素内部，所以分析器将这个aside元素的内容理解成是和article元素的内容相关联的。 -->
```

新增的主体结构元素

aside元素

```
<h1>名词解释</h1>
    <dl>
      <dt>宋词</dt>
      <dd>词，是我国古代诗歌的一种。它始于梁代，形成于唐代而极盛于宋代。（全部文章）</dd>    </dl>
    <dl>
      <dt>婵娟</dt>
      <dd>美丽的月光</dd>
    </dl>
  </aside>
</article>
</body>
```

宋词赏析

水调歌头

. . . 但愿人长久，千里共婵娟(文章正文)

名词解释

宋词
　　词，是我国古代诗歌的一种。它始于梁代，形成于唐代而极盛于宋代。（全部文章）

婵娟
　　美丽的月光

新增的主体结构元素

aside元素　　3-8.html

```
<nav>
        <h2>友情链接</h2>
        <ui>
                <li><a href="http://www.3testing.com">啄木鸟测试网</a></li>
                <li><a href="http://www.51testing.com">51测试网</a></li>
                <li><a href="http://www.lc.com">领测网</a></li>
                <li><a href="http://www.uml.org.cn">火龙果</a></li>
        </ui>
    </nav>
</aside>
```

友情链接

- 啄木鸟测试网
- 51测试网
- 领测网
- 火龙果

新增的主体结构元素

time元素

time是一个新元素，用于明确地对机器的日期和时间进行编码，并且以让人易读的方式来展现它。

time元素代表24小时中的某个时刻或某个日期，表示时刻允许带时差。它可以定义很多格式的日期和时间，如下所示：

3-9.html

- <time datetime="2011-10-12">2011年10月12日</time>
- <time datetime="2011-10-12">10月12日</time>
- <time datetime="1985-06-03">我的生日</time>
- <time datetime="2011-10-12T20:00">今天晚上8点吃饭</time>
- <time datetime="2011-10-12T20:00Z">今天晚上8点吃饭</time>
- <time datetime="2011-10-12T20:00+09:00">现在时晚上8点得美国时间</time>

2011年10月12日
10月12日
我的生日
今天晚上8点吃饭
今天晚上8点吃饭
现在时晚上8点得美国时间

新增的主体结构元素

pubdate属性

pubdate是一个布尔属性，用来表示这个特定的<time>是一篇<article>或整个<body>内容的发布日期。你可能会奇怪，为什么需要pubdate属性。为什么不假设一篇<article>的<header>中的任何一个<time>元素就是其发布日期呢？为了解决这个疑问，我们来看一下下面的这个实例。

3-10.html

```
<article>
<header>
  <h1>XXX公司<time datetime=2016-09-28>9月28日</time>发布
  <p>发布日期：<time datetime=2016-09-20 pubdate>9月20日</
</header>
<p>通知：由于公司....</p>
</article>
```

XXX公司9月28日发布放假通知

发布日期：9月20日

通知：由于公司….

HTML 5的结构

- 新增的主体结构元素
- 新增的非主体结构元素

目录

新增的非主体结构元素

本节需要掌握的知识

1、header元素
2、hgroup元素
3、footer元素
4、address元素

新增的非主体结构元素

header元素

header元素是一种具有引导和导航作用的结构元素，通常用来放置整个页面或页面内的一个内容区块的标题，但也可以包含其内容，例如搜索表单或相关的logo图片。

顾名思义，整个页面的标题应该放在页面的开头，我们可以用如下所示的形式书写页面的标题;

```
<header><h1>页面标题</h1></header>
```

一个网页内并未限制header元素的个数，可拥有多个，可以为每个内容区块加一个header元素。

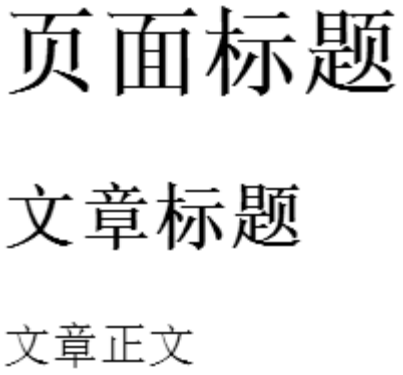
在HTML5中，一个header元素通常包括至少一个heading元素（h1~h6），也可以包括hgroup、table、from、nav。

新增的非主体结构元素

header元素

```
<header>
  <h1>页面标题</h1>
</header>
<article>
  <header>
  <h1>文章标题</h1>
  </header>
  <p>文章正文</p>
</article>
```

3-11.html



新增的非主体结构元素

hgroup元素

hgroup元素是将标题及其子标题进行分组的元素。hgroup元素通常会将h1~h6元素进行分组，例如一个内容区块的标题及其子标题算一组。通常，如果文章只有一个主标题，是不需要hgroup元素的。但是，**如果文章有主标题，主标题下有子标题，就需要使用hgroup元素了**。

新增的非主体结构元素

hgroup元素

```
<article>
    <header>
        <hgroup>
        <h1>主标题</h1>
        <h2>子标题</h2>
        </hgroup>
    <p><time datetime="2016-10-11">2016年10月11日</p>
    </header>
    <p>文章正文</p>
</article>
```

3-12.html

主标题

子标题

2016年10月11日

文章正文

新增的非主体结构元素

footer元素

footer元素可以作为其上层父级内容区块或是一个根区块的脚注。footer通常包括其相关区块的脚注信息，如作者、相关阅读链接及版权信息等。

与header元素一样，一个页面中也没有对footer元素的个数。同时，可以为article元素或section元素添加footer元素。

新增的非主体结构元素

address元素

address 元素用来在文档中呈现联系信息，包括文档作者或文档维护者的名字、他们的网站链接、电子邮箱、真实地址、电话号码等。Address应该不只是用来呈现电子邮箱或真实地址，还应用来展示跟文档相关的联系人的所有联系信息。

3-13.html

```
<address>
        <a href="http://blog.sina.com.cn/u/2329380554">顾翔的微博</a>
</address>
```

顾翔的微博

HTML5开发教程

- HTML 5的新特性
- HTML 5与HTML 4的区别
- HTML 5的结构
- HTML 5中的表单
- HTML 5中的文件与拖放
- 多媒体播放
- 绘制图形
- 数据存储
- 获取地理位置信息
- 练习

目录

HTML 5中的表单

- 新增表单元素与属性
- 对表单的验证
- 增加的页面元素

目录

新增元素与属性

本节需要掌握的知识

1、新增的属性
2、增加与改良的input元素的种类
3、output元素的添加
4、应用新增元素制作注册表单

新增元素与属性

新增属性——placeholder属性

placeholder是指当文本框（<input type="text">或<textarea>）处于未输入状态时文本框中显示的输入提示。当文本框处于未输入状态并且未获取光标焦点时，模糊显示输入提示文字。

实现方法非常简单，只要加上placeholder属性，然后指定提示文字就可以了。 Placeholder属性的使用方法如下所示。 4-1.html

```
<input type="text" placeholder="write me">
```

新增元素与属性

新增属性——autocomplete属性

对于autocomplete属性，可以指定“on”、“off”与“”（不指定）这三种值。不指定时，使用浏览器的认值（取决于各浏览器的决定）。把该属性设为on时，可以显示指定候补输入的数据列表。使用detailst素与list属性提供候补输入的数据列表，自动完成时，可以将该detailst元素中的数据作为候补输入的数据在文本框中自动显示。

Autocomplete属性的使用方法如下所示。

```
<input type="text" name="mr" autocomplete="on" list="mrs"/>
```

新增元素与属性

新增属性——autofocus属性

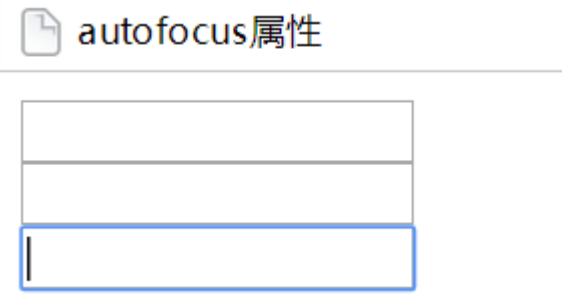
- 给文本框、选择框或按钮控件加上该属性，当画面打开时，该控件自动获得光标焦点。

  目前为止要做到这一点需要使用JavaScript，譬如“control.focus()”。

- autofocus属性的使用方法如下所示。

4-2.html

```
<input type="text" autofocus>
<input type="text" name="name1"><br>
<input type="text" name="name2"><br>
<input type="text" name="name3" autofocus>
```



新增元素与属性

新增属性——list属性

在HTML5中，为单行文本框（<input type="text">）增加了一个list属性，该属性的值为某个datalist元素的id。

Datalist元素也是HTML5中新增元素，该元素类似于选择框（select），但是当用户想要设置的值不在选择列表之内时，允许其自行输入。该元素本身并不显示，而是当文本框获得焦点时以提示输入的方式显示。为了避免在没有支持该元素的浏览器上出现显示错误，可以用CSS等将它设定为不显示。

注意：只有Opera 10浏览器支持list属性

新增元素与属性

新增属性——list属性

text:<input type="text" name="mr" autocomplete="on" list="mrs"/>　　4-3html

<datalist id="greetings" styles="display=none">

<option value="上海">上海</option >

<option value="北京">北京</option >

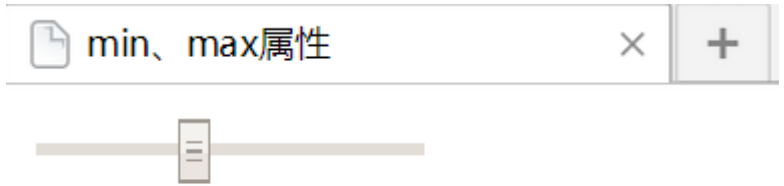
<option value="欢迎您">欢迎您</option >

</datalist>

新增元素与属性

新增属性——min和max

通过设置min和max特性，可以将range输入框的数值输入范围限定在最低值和最高值之间。这两个特性既可以只设置一个，也可以两个都设置，当然还可以都不设置，输入型控件会根据设置的参数对值范围做出相应调整。例如，创建一个表示型大小的range控件，值范围从0%至100%，代码如下所示：　　4-4.html




```
<input id="confidence" name="mr" type=“text" min="0" max="100" value="0">
```

说明：默认的min为0，max为100。

新增元素与属性

新增属性——step

对于输入型控件，设置其step特性能够制定输入值递增或递减的梯度。例如，按如下方式表示型大小range控件的step特性设置为5：

4-5.html


```
<input id="confidence" name="mr" type="range" min="0" max="100  step= 5  value= 0 >
```

设置完成后，控件可接受的输入值只能是初始值与5的倍数之和。也就是说只能输入0、5、10.....100，至于是输入框还是滑动条输入则由浏览器决定。

Step特性的默认值取决于控件的类型。对于range控件，step默认值为1。为了配合step特性，HTML5引入了stepUp和stepDown两个函数对其进行控制。这两个函数的作用分别是根据step特性的值来增加或减少控件的值。

新增元素与属性

新增属性——required

一旦为某输入型控件设置了required特性，那么此项必填，否则无法提交表单。以文本输入框为例，要将其设置为必填项，按照如下方式添加required特性即可：

4-6.html

```
<input type="text" id="firstname" name="mr" required>
```

说明：required属性是最简单的一种表单验证方式。

*firstname:
Secondname:
*pasword:
*repasword:
Email:
提交

新增元素与属性

增加与改良的input元素

在HTML5中，大幅度地增加与改良了input元素的种类，可以简单地使用这些元素来实现HTML5之前需要使用JavaScript才能实现的许多功能。

到目前为止，对于这些input的种类来说，支持得最多、最全面的是Opera10浏览器。对于不支持新增input元素的浏览器来说，统一将这些input元素视为text类型。另外，HTML5中也没有规定这些元素在各浏览器中的外观形式，所以同样的input元素在不同的浏览器中可能会有不同的外观。

新增元素与属性

input元素 ——email类型

- 作用：email类型的input元素是一种专门用来输入email地址的文本框。
- 作用说明：提交时如果该文本框中内容不是email地址格式的文字则不允许提交，但是它不检查email地址是否存在。
- email类型属性：email类型的文本框具有一个multiple属性，它允许在该文本框中是用逗号隔开的有效email地址的一个列表。
- email类型的input元素的使用方法如下所示。

```
<input type="email" name="email" value="mingrisoft@yahoo.com.cn"/>
```

新增元素与属性

input元素 ——email类型

4-7.html

```
<form action="a.jsp">
<input type="email" name="emai1" value="xianggu625@126.com"/><br>
<input type="email" name="emai2" value="1111"/><br>
<input type="submit" id="submit" name="mr">
</form>
```

新增元素与属性

input元素 ——url输入类型

✓作用：url类型的input元素是一种专门用来输入url地址的文本框。

✓作用说明：提交时如果该文本框中内容不是url地址格式的文字，则不允许提交。

✓url类型的input元素的使用方法如下所示

```
<input name="url1" type="url"  value="http://www.mingribook.com" />
```

新增元素与属性

input元素 ——url输入类型

4-8.html

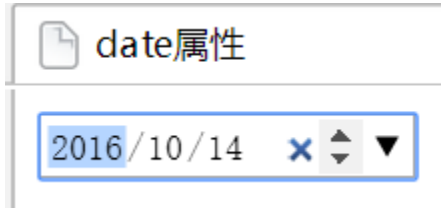
```
<form action="a.jsp">
<form action="a.jsp">
<input type="url" name="url1" value="http://www.3testing.com"/><br>
<input type="url" name="url2" value="www.3testing.com"/><br>
<input type="submit" id="submit" name="mr">
</form>
```

新增元素与属性

input元素 ——data

- 作用：data类型的input元素以日历的形式方便用户输入。
- 作用说明：在Opera浏览器中，当该文本框获得焦点时，显示日历，可以在日历中选择日期进行输入。

4-9.html



- Data类型的input元素的使用方法如下所示：

```
<input name="data1" type="date" value="2016-10-14"/>
```

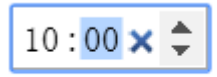
新增元素与属性

input元素 ——time输入类型

- 作用：time类型的input元素是一种专门用来输入时间的文本框，并且在提交时会对输入时间的有效性进行检查。它的外观取决于浏览器，可能是简单的文本框，只在提交时检查是否在其中输入了有效的时间，也可以以时钟形式出现，还可以携带时区。

- time类型的input元素的使用方法如下所示：　　4-10.html

```
<input name="time1" type="time" value="10:00" />
```



新增元素与属性

input元素 ——datatime输入类型

- 作用：datetime类型的input元素是一种专门用来输入UTC日期和时间的文本框，并且在提交时会对输入的日期和时间进行有效性检查。

- datetime类型的input元素的使用方法如下所示：

4-11.html

```
<input name="datetime1" type="datetime" />
```

本浏览器不支持

新增元素与属性

input元素 ——datetime-local输入类型

➢作用：datetime-local类型的input元素是一种专门用来输入本地日期和时间的文本框，并且在提交时会对输入的日期和时间进行有效性检查。 4-12.html

➢datetime-local类型的input元素的使用方法如下所


```
<input name="datetime-local" type="datetime-local" ,-
<form action="demo-form.php">
  生日 (日期和时间): <input type="datetime-local" name="bdaytime">
  <input type="submit">
</form>
```

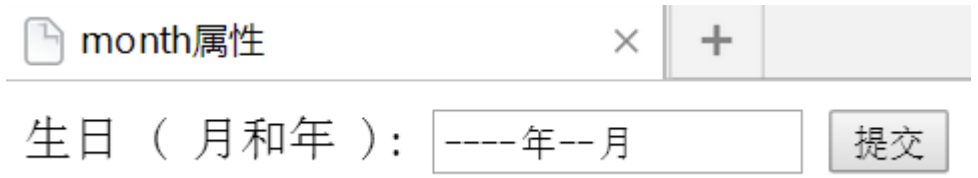
新增元素与属性

input元素 ——month输入类型

➢作用：month类型的input元素是一种专门用来输入月份的文本框，并且在提交时会对输入的月份的有效性进行检查。

4-13.html

➢Month类型的input元素的使用方法如下所示




```
<input name="month1" type="month" value="2011-10" />
<form action="demo-form.php">
  生日 ( 月和年 ): <input type="month" name="bdaymonth">
  <input type="submit">
</form>
```

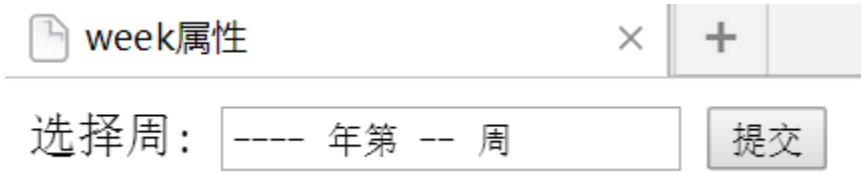
新增元素与属性

input元素——week输入类型

- 作用：Week类型的input元素是一种专门用来输入周号的文本框，并且在提交时会对输入的周号有效性进行检查。它可能是一个简单的输入文本框，允许用户输入一个数字；也可能更复杂。更精确。例如，2016-W07，它代表的是2016年第7个周。
- Opera浏览器中提供了一个辅助输入的日历，可以在该日历中选取日期，选取完毕文本框中自动显示周号。
- Week类型的input元素的使用方法如下所示。

```
<input name="week1" type="week" value="2016-w10" />
<form action="demo-form.php">
  选择周: <input type="week" name="year_week">
  <input type="submit">
</form>
```

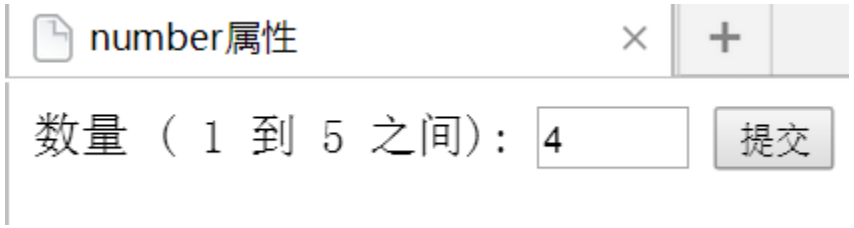
4-14.html



新增元素与属性

input元素——number输入类型

- 作用：Number类型的input元素是一种专门用来输入数字的文本框，并且在提交时会检查其中的内容是否为数字。它与min、max、step属性能很好地协作。
- 显示：在Opear中，它显示为一个微调器控件，将不能超出最大限制和最小限制（如果指定了的话），并且根据step中指定的增量来增加，当然用户也可以输入一个值。
- number类型的input元素的使用方法如下所示：




```
<input name="number1" type="number" value="54" min="10" max="100" step="5" />
<form action="demo-form.php">
  数量 ( 1 到 5 之间): <input type="number" name="quantity" min="1" max="5">
  <input type="submit">
</form>
```

4-15.html

新增元素与属性

input元素——range输入类型

- 作用：range类型的input元素是一种只允许输入一段范围内数值的文本框，它具有min属性与max属性，可以设定最小值与最大值（默认为0与100），它还具有step属性，可以指定每次拖动的步幅。在Opera浏览器中，用滑动条的方式进行值的指定

- range类型的input元素的使用方法如下所示。

```
<input name="range1" type="range" value="25" min="0" max="100" step="5" />
```

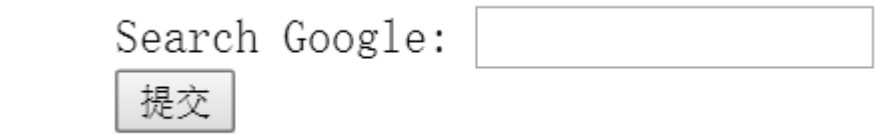
4-4.html

新增元素与属性

input元素——search输入类型

➢作用：Search类型的input元素是一种专门用来输入搜索关键词的文本框。Search类型与text类型仅仅在外观上有区别。

在Safari4浏览器中，它的外观为操作系统默认的圆角矩形文本框，但这个外观可以用CSS样式表进行改写。在其他浏览器中，它的外观暂与text类型的文本框外观相同，但可以用CSS样式表进行改写，如下所示。 4-16.html




```
input[type="search"]{-webkit-appearance:textfield;}
```

新增元素与属性

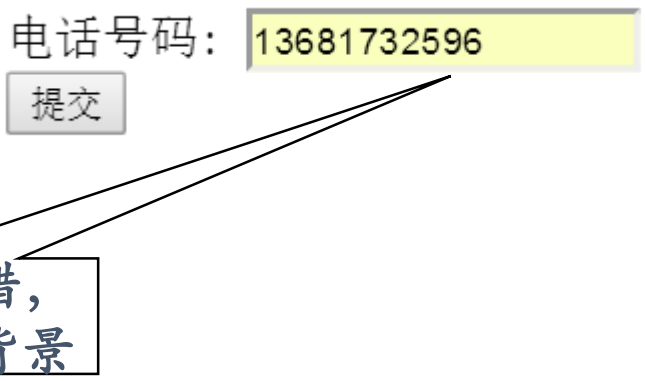
input元素——tel、color输入类型

- tel类型的input元素被设计为用来输入电话号码的专用文本框。它没有特殊的校验规则，它甚至不强调只输入数字，因为很多电话号码常常带有额外的字符，例如44-1234567。但是在实际开发中可以通过pattern属性来指定对于输入的电话号码格式的验证。

- color类型的input元素用来选取颜色，它提供了一个颜色选取器。现在，它只在Black Berry浏览器中被支持。

4-17.html

```
<form action="demo-form.php">
  电话号码: <input type="tel" name="usrtel"><br>
  <input type="submit">
</form>
```



新增元素与属性

output元素的添加

output元素显示出一些计算的结果或者脚本的其他结果。output元素必须从属于某个表单，

也就是说，必须将它书写在表单内部，或对它添加from属性。目前为止该元素只被

Opera10浏览器支持。 4-18.html

```
<form action="demo-form.php">
<input id="confidence" name="mr" type="range" min="0" max="100" value="0">
<output onforminput="value= mr.value">50</output>
</form>
```

新增元素与属性

对新的表单元素使用样式说明

HTML5中新增加了上面提到的这么多表单元素，那么怎么对这些元素使用样式呢？基本上与其他元素一样，字体、颜色等对于样式的编辑，基本上采取同样方法。但是如果你想把日历的背景改成浅蓝色，或者把number元素的增减调节按钮改大一点，或者修改错误信息的字体改成深蓝色之类的，这些修改是不可能的。**到目前为止，还没有可以针对新元素的局部区域进行修改的样式。**

新增元素与属性

案例

4-19.html

```
<body>
<h1>注册表单</h1>
<form id=regForm onsubmit="return chkForm();" method=post>
<fieldset>
<ol>
 <li><label for=username>用户昵称： </label><input id=username name=username
 autofocus required>
 <li><label for=uemail>Email： </label><input id=uemail type=email name=uemail
 required placeholder="example@domain.com">
 <li><label for=age>工作年龄： </label><input id=age type=range  name=range1 max="60" min="18"><output
   onforminput="value=range1.value">30</output>
 <li><label for=age2>年龄:</label><input id=age2 type=number required  placeholder="your age">
```

新增元素与属性

```
<li><label for=birthday>出生日期： </label><input id=birthday type=date>
 <li><label for=search>个人主页： </label><input id=search type=url
 required list="searchlist">
 <datalist id=searchlist>
 <option label="Google" value="http://www.google.com" />
 <option label="Yahoo" value="http://www.yahoo.com" />
 <option label="Bing" value="http://www.bing.com" />
 <option label="Baidu" value="http://www.baidu.com" />
 </datalist></li>
</ol>
</fieldset>
<div><button type=submit>注册</button> </div></form>
</body>
```

HTML 5中的表单

- 新增表单元素与属性
- 对表单的验证
- 增加的页面元素

目录

对表单的验证

本节需要掌握的知识

1、自动验证
2、checkValidity显式验证法
3、避免验证
4、使用setCustomValidity方法自定义错误信息

对表单的验证

自动验证 ——required属性

- 1、required属性

required属性主要目的是确保表单控件中的值已填写。在提交时，如果元素中内容为空白，则不允许提交，同时在浏览器中显示信息提示文字，提示用户这个元素中必须输入内容。

4-6.html

对表单的验证

自动验证 ——pattern属性

- 2、pattern属性

pattern属性主要目的是根据表单控件上设置的格式规则验证输入是否为有效格式。对input元素使用pattern属性，并且将属性值设为某个格式的正则表达式，在提交时会检查其内容是否符合给定格式。当输入的内容不符合给定格式时，则不允许提交，同时在浏览器中显示信息提示文字，提示输入的内容必须符合给定格式。

4-20.html

例如下面所示，要求输入的内容必须为一个数字与三个大写字母。

```
<input pattern="[0-9][A-Z]{3}" name="mr" size="40“ placeholder="输
" />
```

对表单的验证

自动验证——min、max属性

- 3、min属性与max属性

min与max这两个属性是数值类型或日期类型的input元素的专用属性，它们限制了在input元素中输入的数值与日期的范围。

4-4.html

对表单的验证

自动验证 ——step属性

- step属性

step属性控制input元素中的值增加或减少时的增量。例如当你想让用户输入的值在0与100之间，但必须是5的倍数时，你可以指定step为5。

4-5.html

对表单的验证

checkValidity显式验证法

在HTML5中，form元素与input元素（包括select元素与textarea元素）都具有一个checkValidity方法。调用该方法，可以显式地对表单内所有元素内容或单个元素内容进行有效性验证。checkValidity方法以boolean的形式返回验证结果。

另外还要提到的是，在HTML5中，form元素与input元素都还存在一个validitystate属性，该属性返回一个validitystate对象。该对象具有很多属性，但最简单、最重要的属性为valid属性，它表示了表单内所有元素内容是否有效或单个input元素内容是否有效。

对表单的验证

checkValidity显式验证法

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>checkValidity显式验证法</title>
<script language="javascript">
function check(){
    var email=document.getElementById("email");
    if (email.value=="")
    {
        alert("请输入Email地址");
        return false;
    }else if(!email.checkValidity())
    {
        alert("请输入正确的Email地址");
```

对表单的验证

checkValidity显式验证法

```
    }else{
        alert("输入了正确的Email地址");
        return true;
    }
}
</script>
</head>
<body>
<form id="testform" onsubmit="mycheck();">
<input type="email" id="email"><br>
<input type="submit">
</form>
</body>
```

4-21.html

本浏览器不支持

对表单的验证

使用setCustomValidity方法自定义错误信息

HTML5中许多新的input元素都带有对于输入内容的有效性的检查，如果检查不通过，浏览器会针对该元素提供错误信息。但有时开发者不想使用这些默认的错误信息提示，而想使用自己定义的错误信息提示。或者有时，想给某个文本框增加一种错误信息提示，例如密码与确认密码不一致时用浏览器错误信息提示方式提供关于密码不一致的错误信息。

在HTML5中，可以使用Javascript调用各input元素的setCustomValidity方法来自定义错误信息。

对表单的验证

使用setCustomValidity方法自定义错误信息

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>自定义错误信息示例</title>
<script language="javascript">
function check()
{
   var pass1=document.getElementById("pass1");
   var pass2=document.getElementById("pass2");
   if(pass1.value!=pass2.value)
      pass2.setCustomValidity("密码不一致。");
   else
      pass2.setCustomValidity("");
   var email=document.getElementById("email");
     if(!email.checkValidity())
        email.setCustomValidity("请输入正确的Email地址。");
}
</script>

<form id="testform" onsubmit="return check();">
密码：<input type=password name="pass1" id="pass1" /><br/>
确认密码：<input type=password name="pass2"  id="pass2"/><br/>
Email:<input type=email name="email1" id="email"/><br/>
<div><input type="submit" /></div>
</form>
```

4-22.html

本浏览器不支持

HTML 5中的表单

- 新增表单元素与属性
- 对表单的验证
- 增加的页面元素

目录

增强的页面元素

本节需要掌握的知识

1、新增的figure元素
2、新增的details元素
3、新增的mark元素
4、新增的progress元素
5、新增的meter元素
6、改良的ol列表
7、改良的dl列表
8、加以严格限制的cite元素
9、重新定义的small元素

增强的页面元素

新增的figure元素

figure 元素用来表示网页上一块独立的内容，将其从网页上移除后不会对网页上的其他内容产生任何影响。

figcaption元素表示figure 元素的标题。它从属于figure 元素，所以其必须书写在figure 元素内部，可以书写在figure 元素内的其他从属元素的前面或后面。一个figure 元素内最多只允许放置一个figcaption元素，但允许放置多个其他元素。

4-23.html

```
<figure>

<img src="img/1.PNG">

<figcaption>啄木鸟软件测试培训网</figcaption>

<figure>
```

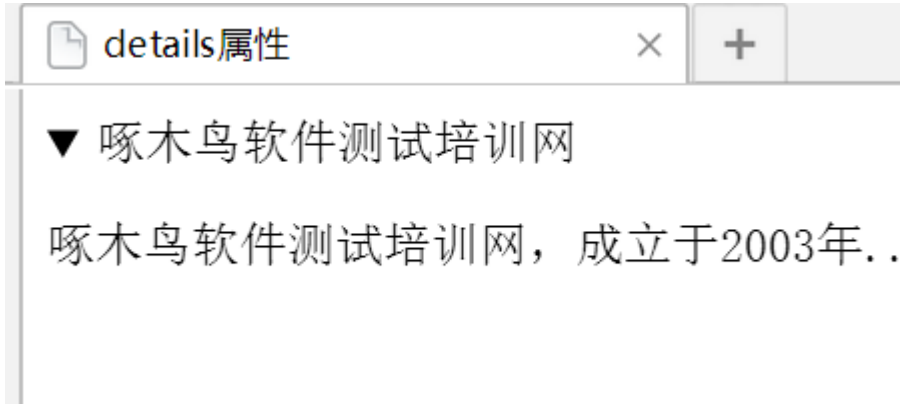
增强的页面元素

新增的details元素

details元素提供了一种替代JavaScrip的，它主要是提供了一个展开/收缩区域。例如如下的应用代码：

4-24.html

```
<details>
  <summary>啄木鸟软件测试培训网</summary>
  <p>啄木鸟软件测试培训网，成立于2003年....</p>
</details>
```



Details元素具有一个属性“open”，使用<details open> 语句对details 添加该属性，添加该属性后在画面打开时details元素所表示的局部区域则会处于展开状态。

说明：details元素内并不限于放置文字，也可以在其内部放置表单、插件或对于一个统计图提供的详细数据表格。

增强的页面元素

新增的mark元素

mark元素表示页面中需要突出显示或高亮显示的，对于当前用户具有参考作用的一段文字。它通常使用于引用原文的时候，目的是引起读者的注意。mark元素的作用相当于使用一支荧光笔在打印的纸张上标出一些文字。

说明：mark元素最主要的目的是吸引当前用户的注意，因为标示出来的文字与用户的当前操作有关，通常该元素对于当前用户具有很好的帮助作用。

能够体现mark元素作用的最好例子是对网页全文检索某个关键词时显示的检索结果，现在许多检索引擎用其他方法实现了mark元素所要达到的功能。

mark元素的另一个主要作用是在引用原文的时候，为了某种特殊目的而把原文作者没有特别重点标示的内容给标示出来。

增强的页面元素

新增的mark元素

```
<!DOCTYPE html>
<meta charset="UTF-8" />
<title> mark元素应用在网页检索时的示例</title>
<h1>搜索"<mark>HTML 5</mark>",找到相关网页约10,210,000篇，用时0.041秒</h1>
<section id="search-results">
  <article>
    <h2>
      <a href="http://developer.51cto.com/art/200907/133407.htm">
        专题：<mark>HTML 5</mark> 下一代Web开发标准详解_51CTO.COM - 技术成就梦想 ...
      </a>
    </h2>
    <p><mark>HTML 5</mark>是近十年来Web开发标准最巨大的飞跃</p>
  </article>
```

4-25.html

增强的页面元素

新增的mark元素

```
<article>
    <h2>
      <a href="http://paranimage.com/list-of-html-5/">
        <mark>HTML 5</mark>一览 | 帕兰映像
      </a>
    </h2>
    <p><mark>html 5</mark>最近被讨论的越来越多，越来越烈...</p>
  </article>
  <article>
    <h2>
      <a href="http://www.chinabyte.com/keyword/HTML+5/">
        <mark>html 5</mark>_比特网
      </a>
```

增强的页面元素

新增的mark元素

```
<p><mark>HTML 5</mark>提供了一些新的元素和属性，反映典型...</p>
  </article>
  <article>
    <h2>
      <a href="http://www.slideshare.net/mienflying/html5-4921810">
        <mark>HTML 5</mark>表单
      </a>
    </h2>
    <p>about <mark>HTML 5</mark> Form,the web form 2.0 tech
  </article>
</section>
```

增强的页面元素

mark元素与em、strong元素的区别

在HTML4中，开发者可能已经习惯于用em元素或strong元素来突出显示文字，但要注意mark元素的作用与这两个元素的作用是有区别的，不能混同使用。**Mark元素的标示目的与原文作者无关**，或者说它不是原文作者用来标示文字的，而是在后面引用的时候添加上去的，它的目的是吸引当前用户的注意力，提供给用户作参考，希望能够对用户有帮助。而strong是**原文作者用来强调一段文字的重要性**，例如警告信息、错误信息等，em元素是作者为了突出文章重点而使用的。

增强的页面元素

新增的progress元素

Progress是HTML5标准草案中新增的元素之一。它表示一个任务的完成进度，这进度可以是不确定的，只是表示进度正在进行，但是不清楚还有多少工作量没有完成，也可以用0到某个最大数字（例如100）之间的数字来表示准确的进度完成情况（例如进度百分比）

该元素主要有两个属性：

- value属性表示已经完成了多少工作量，
- max属性表示总共有多少工作量。

工作量的单位是随意的，不用指定。

注意：value和max属性的值必须大于0，value的值小于或等于max属性的值。

增强的页面元素

新增的progress元素

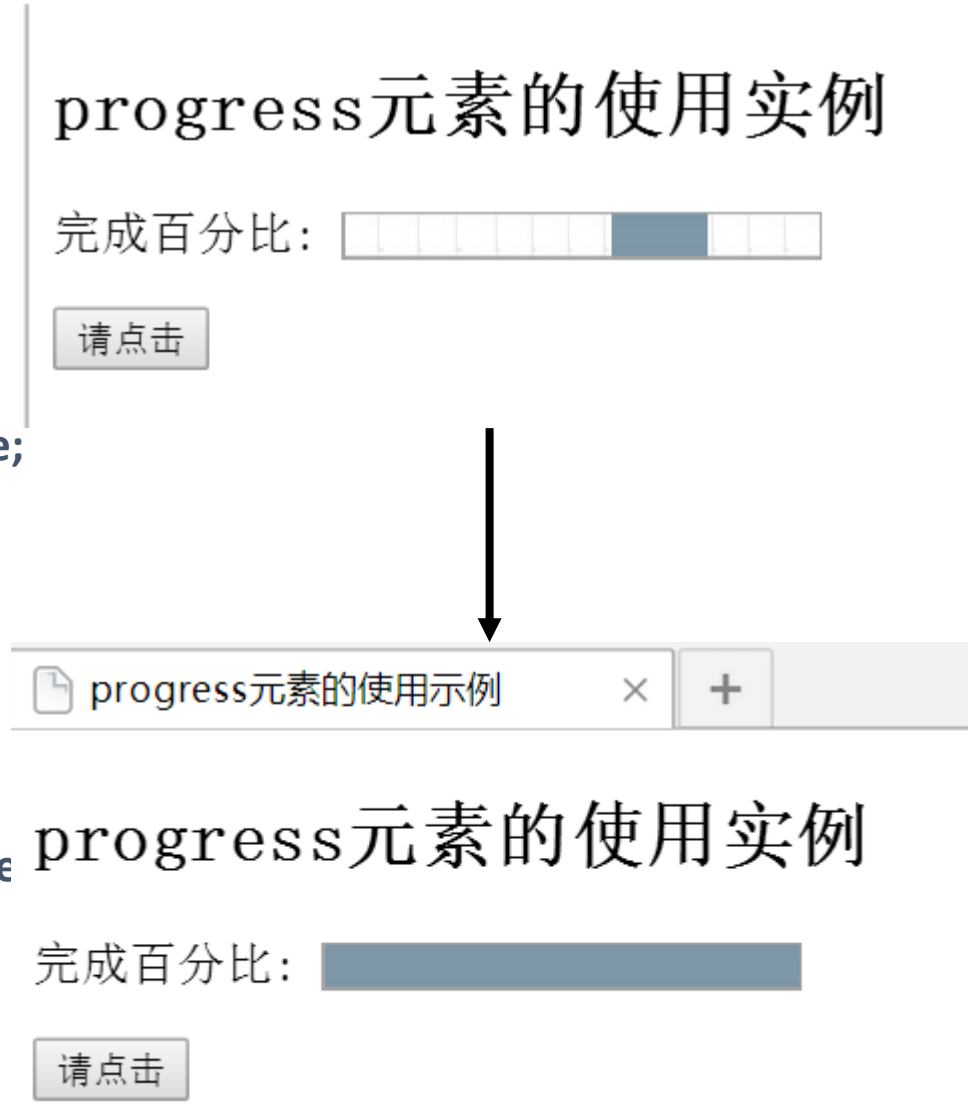
```
<!DOCTYPE html>
<meta charset="UTF-8"/>
<title>progress元素的使用示例</title>
<script>
var progressBar = document.getElementById('p');
function button_onclick()
{
   var progressBar = document.getElementById('p');
   progressBar.getElementsByTagName('span')[0].textContent ="0";
   for(var i=0;i<=100;i++)
      updateProgress(i);
}
```

4-26.html

增强的页面元素

新增的progress元素

```
function updateProgress(newValue)
{
    var progressBar = document.getElementById('p');
    progressBar.value = newValue;
    progressBar.getElementsByTagName('span')[0].textContent = newValue;
}
</script>
<section>
    <h2>progress元素的使用实例</h2>
    <p>完成百分比: <progress id="p" max=100><span>0</span>%</progre
    <input type="button" onclick="button_onclick()"  value="请点击"/>
</section>
```



增强的页面元素

新增的meter元素

Meter元素表示规定范围内的数量值。例如，磁盘使用量，对某个候选者的投票人数占总投票人数的比例等。

注意：<meter>不可以用来表示那些没有已知范围的任意值，例如重量、高度，除非已经设定了它们值的范围。

Meter元素有如下六个属性

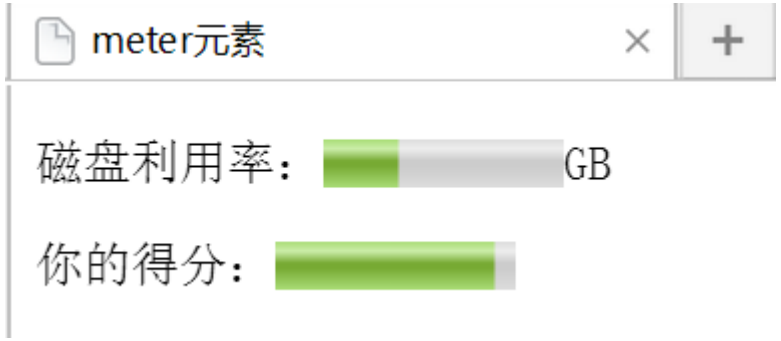
- **value**：表示当前标量的实际值；该属性的默认值为0，可以给该属性指定一个浮点小数值
- **min**：当前标量的最小值；如不做指定则为0。
- **max**：当前标量的最大值；如不做指定则为1；如果指定的最大值小于最小值，那么最小值会被认为是最大值。
- **low**：当前标量的低值区；必须小于或等于标量的高值区数字；如果低值区数字小于标量最小值，那么它会被认为是最小值。
- **high**：当前标量的高值区。
- **optimum**：最佳值；其范围在最小值与最大值区间当中，并且可以处于高值区

增强的页面元素

新增的meter元素

```
<p>磁盘利用率：<meter value="50" min="0" max="160">50/160</meter>GB</p>
<p>你的得分：<meter value="91" min="0" max="100"  low="10" high="90" optimum="100">A+</meter></p>
```

4-27.html



增强的页面元素

改良的ol列表

在HTML5中，将ol列表进行了改良，为它添加了**start**属性与**reversed**属性。

- 如果你不想ol元素所代表的列表编号从1开始，那么可以使用start属性来自定义编号的初始值
- 如果你想对列表进行反向排序，那么你可以使用ol列表的reversed属性，但是，**现在还没有任何浏览器对该属性提供支持**。

```
<h3></h3>
<ol start="5">
<li>列表5</li>
<li>列表6 </li>
<li>列表7 </li>
<li>列表8 </li>
<li>列表9 </li>
<li>列表10 </li>
</ol>
```

4-28.html

ol的Start属性列表

5. 列表5
6. 列表6
7. 列表7
8. 列表8
9. 列表9
10. 列表10

增强的页面元素

改良的dl列表

在HTML4中，dl元素是一个定义列表，包含了一个术语及其一个或多个定义。这个定义不明确而且容易令人混淆。

在HTML5中，将该元素进行了重新定义，重新定义后的dl列表包含多个带名字的列表项。每一项包含一条或多条带名字的dt元素，用来表示术语，dt元素后面紧跟一个或多个dd元素，用来表示定义。在一个元素内，不允许有相同名字的dt元素，不允许有重复的术语。

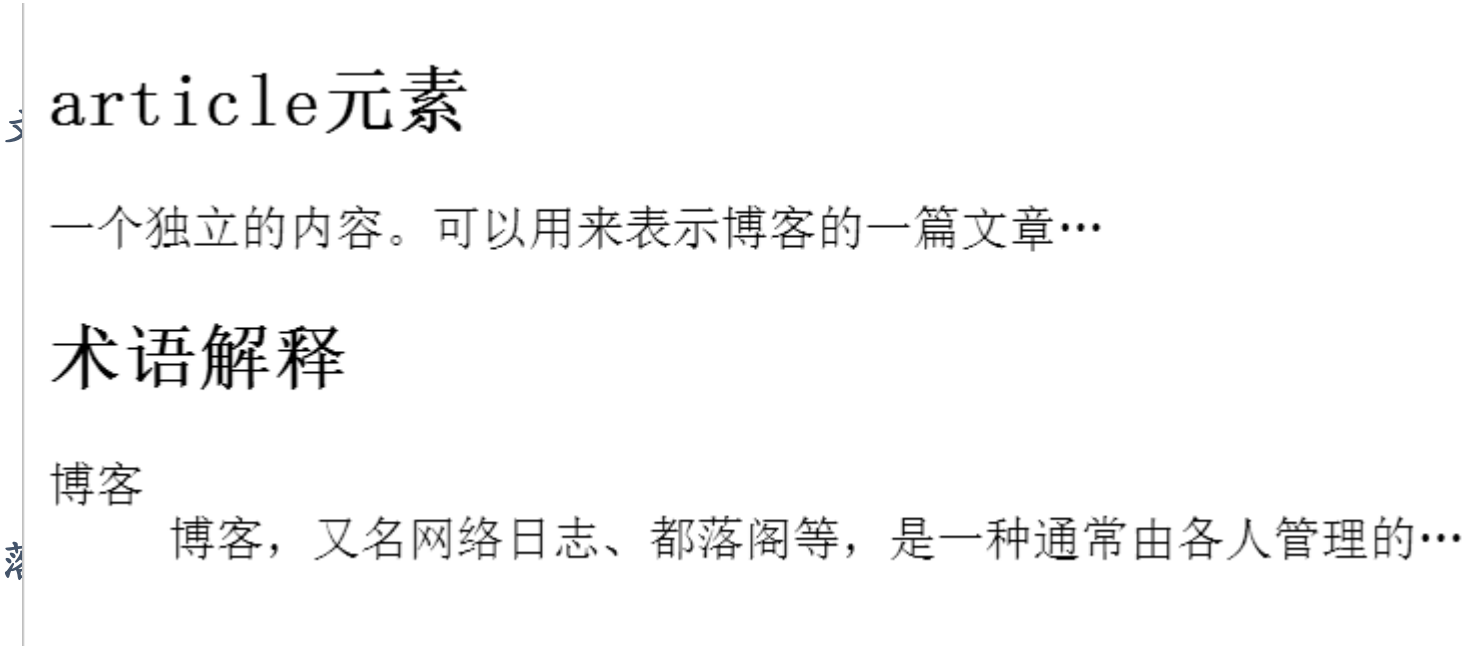
dl列表页可以用来表示一些页面或article元素中内容的辅助信息，例如作者、类别等。

增强的页面元素

改良的dl列表

4-29.html

```
<h3>用于表示术语的dl列表示列</h3>
<article>
    <h1>article元素</h1>
    <p>一个独立的内容。可以用来表示博客的一篇文
    <aside>
    <h2>术语解释</h2>
        <dl>
            <dt>博客</dt>
            <dd>博客，又名网络日志、都落
        </dl>
    </aside>
</article>
```



增强的页面元素

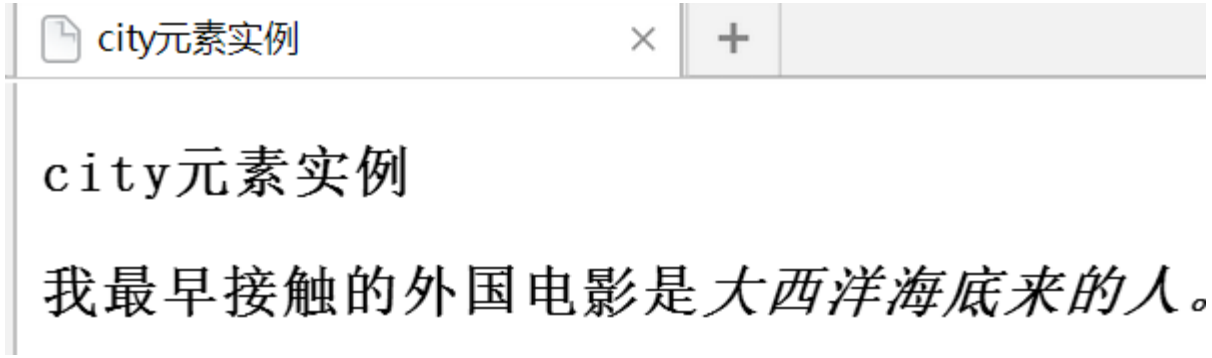
加以严格限制的cite元素

cite元素表示作品（例如一本书、一篇文章、一首歌曲等）的标题。该作品可以在页面中被详细引用，也可以只在页面中提一下。

在HTML4中，cite元素可以用来表示作者，但是在HTML5中明确规定了不能用cite元素表示包括作者在内的任何人名，因为人的名字不是标题（当然除非标题就是一个人的名字），但是为了与HTML4或之前版本的网页兼容，并没有把它当作错误，所以这只是一个规定而已。

4-30.html

```
<h3>city元素实例<h3>

<p>我最早接触的外国电影是<cite>大西洋海底来的
```



增强的页面元素

重新定义的small元素

Small元素已经完全重新定义了，从仅仅是一个通用的表现性元素，变成了使得文本显示得较小，实际上表示“附属细则”，它通常用来免责、警告、提出法律限制或版权。附属细则有时候也用于表明权限，用于满足许可性需求。同时不允许被应用在页面主内容中，只允许被当做辅助信息用inline方式内嵌在页面上使用。同时，small元素也不意味着元素中内容字体会变小，如果需要将字体变小，需要配合着CSS样式表来使用。

HTML5开发教程

- HTML 5的新特性
- HTML 5与HTML 4的区别
- HTML 5的结构
- HTML 5中的表单
- HTML 5中的文件与拖放
- 多媒体播放
- 绘制图形
- 数据存储
- 获取地理位置信息
- 练习

目录

HTML 5中的文件与拖放

- 选择文件
- 使用FileReader接口读取文件
- 拖放API 290

目录

选择文件

本节需要掌握的知识

1、通过file对象选择文件
2、通过file对象选择文件
3、通过类型过滤选择的文件

选择文件

通过file对象选择文件

在HTML4中，file控件内只允许放置一个文件，但是到了HTML5中，通过添加multiple属性，在file控件内允许一次放置多个文件。控件内的每一个用户选择的文件都是一个file对象，而FileList对象则为这些file对象的列表，代表用户选择的所有文件。

File对象有两个属性，name属性表示文件名，不包括路径，lastModifiedDate属性表示文件的最后修改日期。

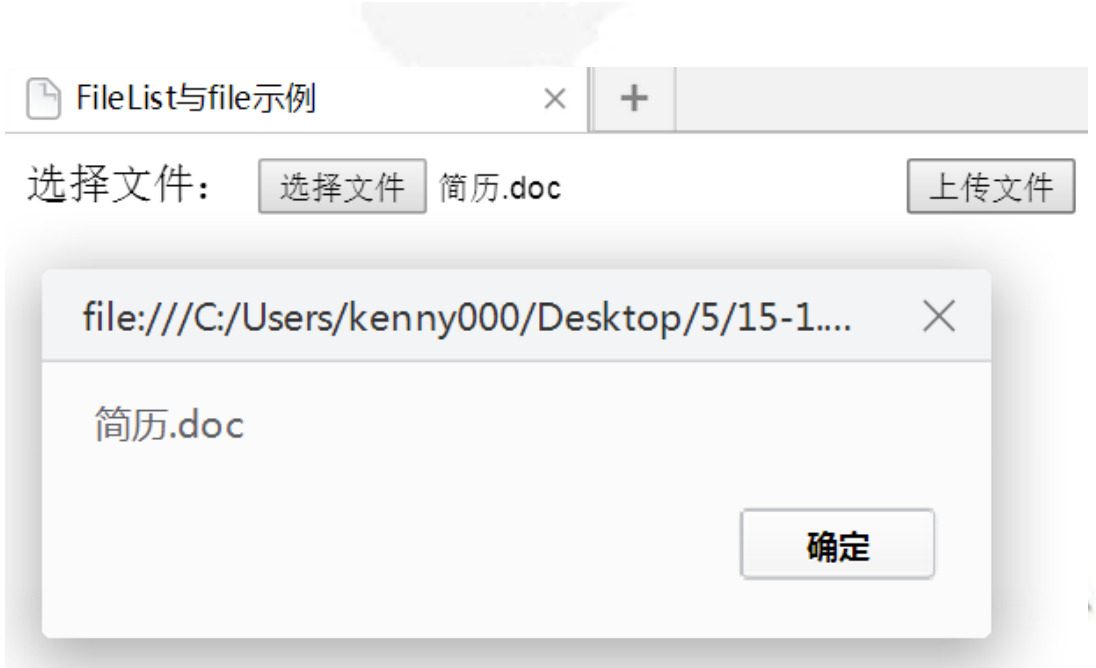
例：在本例中通过单击“浏览”按钮，选择要上传得文件，然后单击“上传文件”按钮，将会弹出一个对话框，在这个对话框中将显示上传文件的名称。

选择文件

通过file对象选择文件

5-1.html

```
<!DOCTYPE html><head>
<meta charset="UTF-8">
<title>FileList与file示例</title>
</head>
<script language=javascript>
function ShowName()
{
   var file;
   //返回FileList文件列表对象
   for(var
     i=0;i<document.getElementById("file").files.length;i++)
     {
     //file对象为用户选择的单个文件
      file = document.getElementById("file").files[i];
      //弹出文件名
      alert(file.name);
}
</script>
选择文件:
<input type="file" id="file"size="50"/>
<input type="button" onclick="ShowName();" value="上传文件"/>
```



选择文件

使用Blob接口获取文件的类型与大小

Blob表示二进制原始数据，它提供一个slice方法，可以通过该方法访问到字节内部的原始数据块。

Blob对象有两个属性，size属性表示一个blob对象的字节长度，type属性表示blob的MIME类型，如果是未知类型，则返回一个空字符串。

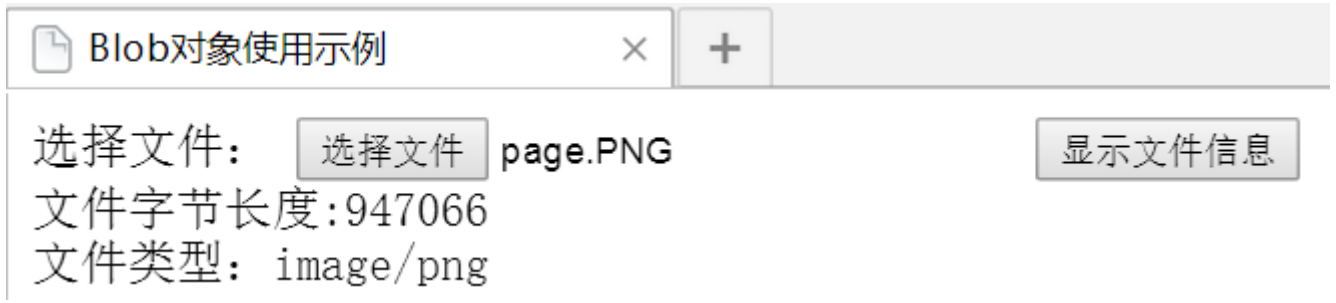
例:下面通过一个实例来对blob对象的两个属性做一些解释。在本例中，首先通过单击“浏览”按钮选择文件，然后单击“显示文件信息”按钮，在页面中将显示浏览文件的文件长度与文件类型。

选择文件

使用Blob接口获取文件的类型与大小

5-2.html

```
<!DOCTYPE html><head>
<meta charset="UTF-8">
<title>Blob对象使用示例</title>
<script language=javascript>
function ShowFileType()
{
   var file;
   //得到用户选择的第一个文件
   file = document.getElementById("file").files[0];
   var size=document.getElementById("size");
   //显示文件字节长度
   size.innerHTML=file.size;
   var type=document.getElementById("type");
   //显示文件类型
   type.innerHTML=file.type;
}
</script>
选择文件:
<input type="file" id="file" />
<input type="button" value="显示文件信息"
onclick="ShowFileType();"/><br/>
文件字节长度:<span id="size"></span><br/>
文件类型： <span id="type"></span>
```



选择文件

通过类型过滤选择的文件

在上面的实例中，对于图像类型的文件，blob对象的type属性都是以“image/”开头的，后面紧跟这图像的类型，利用此特性我们可以在JavaScript中判断用户选择的文件是否为图像文件，如果在批量上传时，只允许上传图像文件，可以利用该属性，如果用户选择的多个文件中有不是图像的文件时，可以弹出错误提示信息，并停止后面的文件上传，或者跳过这个文件，不将该文件上传。

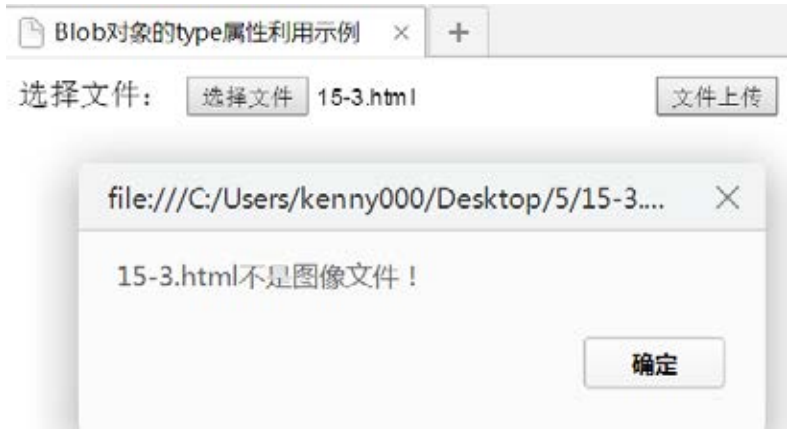
例：下面是对图像类型的判断的实例，在该实例中首先对上传的文件进行判断，如果上传的文件不是图像文件将弹出对话框给出提示，如果是图像文件则显示文件可以上传。

选择文件

通过类型过滤选择的文件

5-3.html

```
<!DOCTYPE html><head>
<meta charset="UTF-8">
<title>Blob对象的type属性利用示例</title>
<script language=javascript>
function FileUpload()
{
   var file;
   for(var i=0;i<document.getElementById("file").files.length;i++)
   {
      file = document.getElementById("file").files[i];
      if(!/image\/\w+/.test(file.type))
      {
         alert(file.name+"不是图像文件！ ");
         break;
      }
      else
      {

         alert(file.name+"文件可以上传");
      }
   }
}
</script>
选择文件:
<input type="file" id="file" multiple/>
<input type="button" value="文件上传"
onclick="FileUpload();"/>
```



HTML 5中的文件与拖放

- 选择文件
- 使用FileReader接口读取文件
- 拖放API 290

目录

使用FileReader接口读取文件

本节需要掌握的知识

1、检测浏览器对FileReader接口
2、FileReader 接口的方法
3、使用 readAsDataURL方法预览图片
4、使用 readAsText方法读取文本文件
5、FileReader接口中的事件

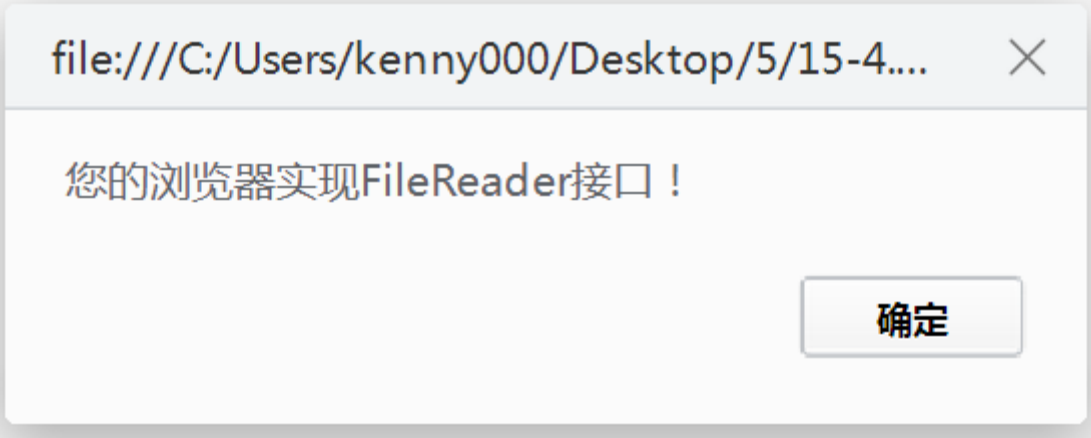
使用FileReader接口读取文件

检测浏览器对FileReader接口

有一种方法可以检查您的浏览器是否对FileReader接口提供支持，如下所示。

5-4.html

```
if ( typeof FileReader === 'undefined' )
{
        alert( " 您的浏览器未实现 FileReader 接口 " );
}
else
{
        var reader = new FileReader();        // 正常使用浏览器
}
```



使用FileReader接口读取文件

FileReader 接口的方法

FileReader 的接口拥有 4 个方法，其中 3 个用以读取文件，另一个用来中断读取。下面的表格列出了这些方法以及它们的参数和功能，需要注意的是，**无论读取成功或失败，方法并不会返回读取结果，这一结果存储在 result属性中**。

方法名	参数	描述
abort	(none)	中断读取
readAsBinaryString	file	将文件读取为二进制码
readAsDataURL	file	将文件读取为 DataURL
readAsText	file, [encoding]	将文件读取为文本

使用FileReader接口读取文件

FileReader 接口的实例

- 使用 readAsDataURL方法预览图

例:在本例中通过单击“浏览”按钮，选择要预览的图片，然后单击“读取图像”按钮。预览的图片将在页面中显示。

使用FileReader接口读取文件

FileReader 接口的实例

5-5.html

```
<!DOCTYPE html><head>
<meta charset="UTF-8">
<title>fileReader方法示例</title>
</head>
<p>
   <label>请选择一个文件：</label>
   <input type="file" id="file" />
   <input type="button" value="读取图像" onclick="readFile ()"/>
</p>
<div name="result" id="result">
    <!-- 这里用来显示读取结果 -->
</div>
<script language=javascript>
var result=document.getElementById("result");
var file=document.getElementById("file");
```

使用FileReader接口读取文件

FileReader 接口的实例

```
if (typeof FileReader == 'undefined' )
{
  result.innerHTML = "<p>抱歉，你的浏览器不支持 FileRea
  file.setAttribute( 'disabled','disabled' );
}
//将文件以Data URL形式进行读入页面
function readFile ()
{
   //检查是否为图像文件
   var file = document.getElementById("file").files[0];
   if(!/image\/\w+/.test(file.type))
   {
      alert("请确保文件为图像类型");
      return false;
   }
```


```
   }
}

</script>
```

使用FileReader接口读取文件

FileReader 接口的实例

- 使用 readAsText方法读取文本文件

例 在本例中通过单击“浏览”按钮，选择要浏览的文本文件，然后单击“读取文本文件”按钮，文本文件的内容将在页面中显示。

使用FileReader接口读取文件

FileReader 接口的实例

5-6.html

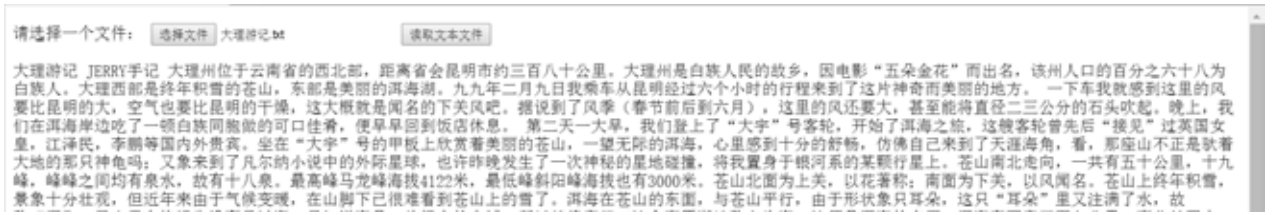
```
<!DOCTYPE html><head>
<meta charset="UTF-8">
<title>fileReader方法示例</title>
</head>
<p>
 <label>请选择一个文件：</label>
  <input type="file" id="file" size="40"  />
  <input type="button" value="读取文本文件" onclick="readAsText()"/>
</p>
<div name="result" id="result">
   <!-- 这里用来显示读取结果 -->
</div>
<script language=javascript>
var result=document.getElementById("result");
var file=document.getElementById("file");
```

使用FileReader接口读取文件

FileReader 接口的实例

```
if (typeof FileReader == 'undefined' )
{
  result.innerHTML = "<p>抱歉，你的浏览器不支持 FileReader</p>";
  file.setAttribute( 'disabled','disabled' );}
function readAsText(){
    var file = document.getElementById("file").files[0];
    var reader = new FileReader();
    //将文件以文本形式进行读入页面
    reader.readAsText(file);
    reader.onload = function(f) {
        var result=document.getElementById("result");
        //在页面上显示读入文本
        result.innerHTML=this.result;    }
}
</script>
```

TEXT文件也还要存储为UTF-8格式



使用FileReader接口读取文件

FileReader接口中的事件

FileReader 包含了一套完整的事件模型，用于捕获读取文件时的状态，下面的表格中归纳了一些事件。

事件	描述
onabor	中断时触发
onerror	出错时触发
onload	文件读取成功完成时触发
onloadend	读取完成触发，无论成功或失败
onloadstart	读取开始时触发
onprogress	读取中

使用FileReader接口读取文件

FileReader接口中的事件实例

当fileReader对象读取文件时，会伴随着一系列事件，它们表示读取文件时不同的读取状态。

例：下面通过一个图片上传的实例，来看一下这些读取状态的先后顺序。

使用FileReader接口读取文件

FileReader接口中的事件实例

5-7.html

```
<!DOCTYPE html><head>
<meta charset="UTF-8">
<title>fileReader对象的事件先后顺序</title>
</head>
<script language=javascript>
var result=document.getElementById("result");
var input=document.getElementById("input");
if(typeof FileReader=='undefined')
{
   result.innerHTML = "<p class='warn'>抱歉，你的浏览器不支持 FileReader</p>";
   input.setAttribute( 'disabled','disabled' );
}
```

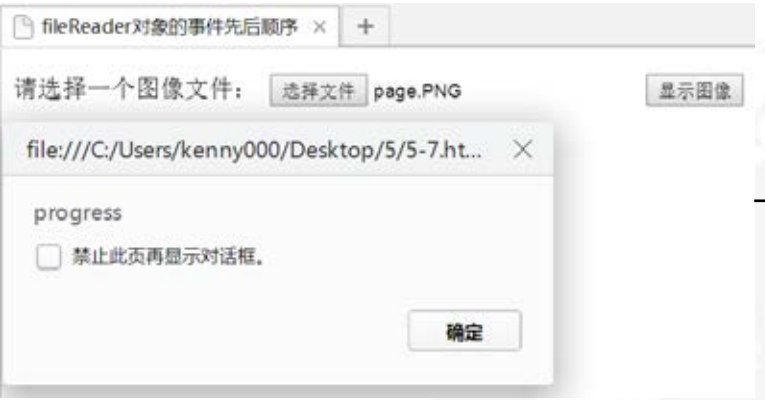
使用FileReader接口读取文件

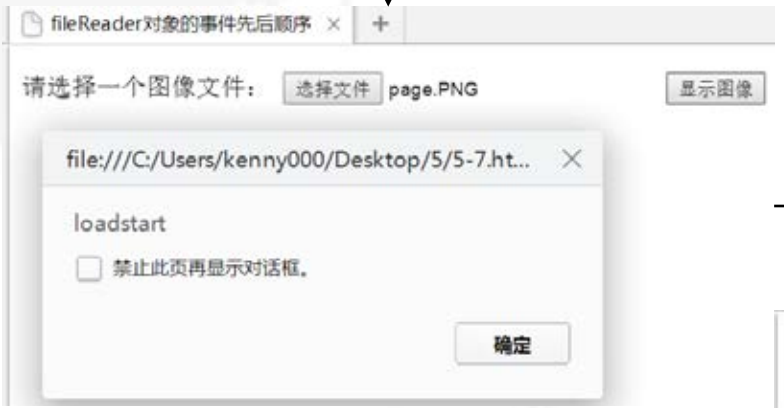
FileReader接口中的事件实例

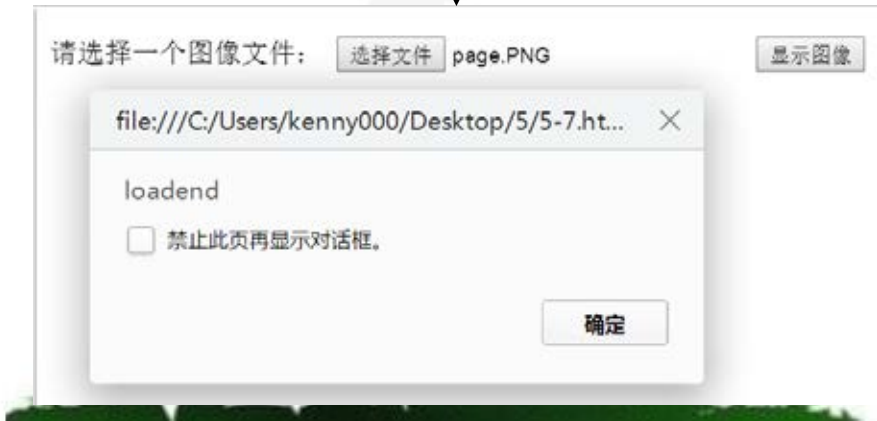
```
function readFile()
{
   var file = document.getElementById("file").files[0];
   var reader = new FileReader();
   reader.onload = function(e)   {
      result.innerHTML = '<img src="'+this.result+'" alt=""/>'
      alert("load");   }
   reader.onprogress = function(e)   {
      alert("progress");   }
   reader.onabort = function(e)   {
      alert("abort");   }
   reader.onerror = function(e)   {
      alert("error");   }
   reader.onloadstart = function(e)   {
      alert("loadstart");   }
   reader.onloadend = function(e)   {
      alert("loadend");   }
   reader.readAsDataURL(file);
}
</script>

<p>
<label>请选择一个图像文件：</label>
<input type="file" id="file" />
<input type="button" value="显示图像" onclick="readFile()"
/>
</p>
<div name="result" id="result">
<!-- 这里用来显示读取结果 -->
</div>
```

使用FileReader接口读取文件

FileReader接口中的事件实例







HTML 5中的文件与拖放

- 选择文件
- 使用FileReader接口读取文件
- 拖放API 290

目录

拖放API

本节需要掌握的知识

1、实现拖放的步骤
2、通过拖放显示欢迎信息

拖放API

拖放API

在HTML5中，提供了直接支持拖放操作的API。虽然HTML5之前已经可以使用mousedown, mousemove, mouseup来实现拖放操作，但是只支持在浏览器内部的拖放，而在HTML5中，已经支持在浏览器与其他应用程序之间的数据的互相拖动，同时也大大简化了有关于拖放方面的代码。

拖放API

实现拖放的步骤

在HTML5中要想实现拖放操作，至少要经过如下两个步骤：

- 1、将想要拖放的对象元素的draggable属性设为true (draggable="true")。这样才能将该元素进行拖放。另外，img元素与a元素（必须指定href)，默认允许拖放。
- 2、编写与拖放有关的事件处理代码。关于拖放存在如下表所示的几个事件：

事件	产生事件的元素	描述
dragstart	被拖放的元素	开始拖放操作
drag	被拖放的元素	拖放过程中
dragenter	拖放过程中鼠标经过的元素	被拖放的元素开始进入本元素的范围内
dragover	拖放过程中鼠标经过的元素	被拖放的元素正在本元素范围内移动
dragleave	拖放过程中鼠标经过的元素	被拖放的元素离开本元素的范围
drop	拖放的目标元素	有其他元素被拖放到了本元素中
dragend	拖放的对象元素	拖放操作结束

拖放API

拖放实例

例 下面我们将按照上面的步骤实现一个拖放实例。在该实例中，有一个显示“拖放”文字的div元素，可以把它拖放到位于它下部的div元素中，每次被拖放时，在下部的div元素中会追加一次“mr欢迎你”文字。

拖放API

拖放实例

5-8.html

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>拖放示例</title>
<script type="text/javascript">
function init()
{
   var source = document.getElementById("dragme");
   var dest = document.getElementById("text");
   // (1) 拖放开始
   source.addEventListener("dragstart", function(ev)
   {
// 向dataTransfer对象追加数据
      var dt = ev.dataTransfer;
      dt.effectAllowed = 'all';
     //(2) 拖动元素为dt.setData("text/plain", this.id);
      dt.setData("text/plain", "上海欢迎你");
   }, false);
   // (3) dragend： 拖放结束
   dest.addEventListener("dragend", function(ev)
   {
      //不执行默认处理（拒绝被拖放）
      ev.preventDefault();
   }, false);
   // (4) drop:被拖放
```

拖放API

拖放实例

```
dest.addEventListener("drop", function(ev)
  {
    // 从DataTransfer对象那里取得数据
    var dt = ev.dataTransfer;
    var text = dt.getData("text/plain");
    dest.textContent += text;
    //(5) 不执行默认处理（拒绝被拖放）
    ev.preventDefault();
    //停止事件传播
    ev.stopPropagation();
  }, false);
}
//(6) 设置页面属性，不执行默认处理（拒绝被拖放）
```

```
document.ondragover = function(e){e.preventDefault();};
document.ondr                                          );};
</script>
</head>
<body onload="
<h1>拖放欢迎语
<!-- (7) 把dragga
<div id="dragme                                        200px; border:
  1px solid gray
  请拖放
</div>
<br>
<div id="text" st                                      border: 1px
  solid gray;"><, ...
</body>
```

拖放欢迎语

请拖放

上海欢迎你上海欢迎你上海欢迎你上海欢迎你

HTML5开发教程

- HTML 5的新特性
- HTML 5与HTML 4的区别
- HTML 5的结构
- HTML 5中的表单
- HTML 5中的文件与拖放
- 多媒体播放
- 绘制图形
- 数据存储
- 获取地理位置信息
- 练习

目录

多媒体播放

- HTML 5多媒体的简述
- 媒体元素基本属性
- 多媒体元素常用方法
- 多媒体元素重要事件

目录

HTML5多媒体的简述

本节需要掌握的知识

1、目前浏览器对Video和Audio的支持
2、基本语法

HTML 5多媒体的简述

目前浏览器对Video和Audio的支持

浏览器	支持情况
Chrome	3.0及以上的版本
Firefox	3.5及以上的版本
Opera	10.5及以上的版本
Safari	3.2及以上的版本

HTML 5多媒体的简述

基本语法

```
<audio src=1.mp3>
该浏览器不支持audio元素
</audio>

<video src=1.mp4>
该浏览器不支持video元素
</audio>
```

自动选择播放的格式

```
<video width="640" heigh="320">
<source src="demo/sample.ogv"  type="ogv/ogg" code="theora, virbis"/>
<source src="demo/sample.mov" type="ogv/quicktime"/>
</video>
```

多媒体播放

- HTML 5多媒体的简述
- 媒体元素基本属性
- 多媒体元素常用方法
- 多媒体元素重要事件

目录

多媒体元素基本属性

本节需要掌握的知识

1、video元素属性
2、audio元素属性

多媒体元素基本属性

video元素与audio元素相同的属性

- **src属性和autoplay属性**

➢ src属性用于指定媒体数据的URL地址。

➢ autoplay属性用于指定媒体是否在页面加载后自动播放，使用方法如下：

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

- **perload属性**

该属性用于指定**视频或音频数据是否预加载**。如果使用预加载，则浏览器会预先将视频或音频数据进行缓冲，这样可以加快播放速度，因为播放时数据已经预先缓冲完毕。该属性有三个可选值，分别是“**none**”、“**metadata**”和“**auto**”，**其默认值为“auto”**。

➢ none值表示不进行预加载；

➢ metadata表示只预加载媒体的元数据（媒体字节数、第一帧、播放列表、持续时间等）。

➢ auto表示预加载全部视频或音频。

perload属性的使用方法如下所示。

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

多媒体元素基本属性

video元素与audio元素相同的属性

- **poster（video元素独有属性）**

  当视频不可用时，可以使用该元素向用户展示一幅替代用的图片。当视频不可用时，最好使用poster属性，以免展示视频的区域中出现一片空白。该属性的使用方法如下所示：

```
<video src="sample.mov" psoter="cannotuse.jpg"></video>
```

- **loop属性**

  用于指定是否循环播放视频或音频，其使用方法如下：

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

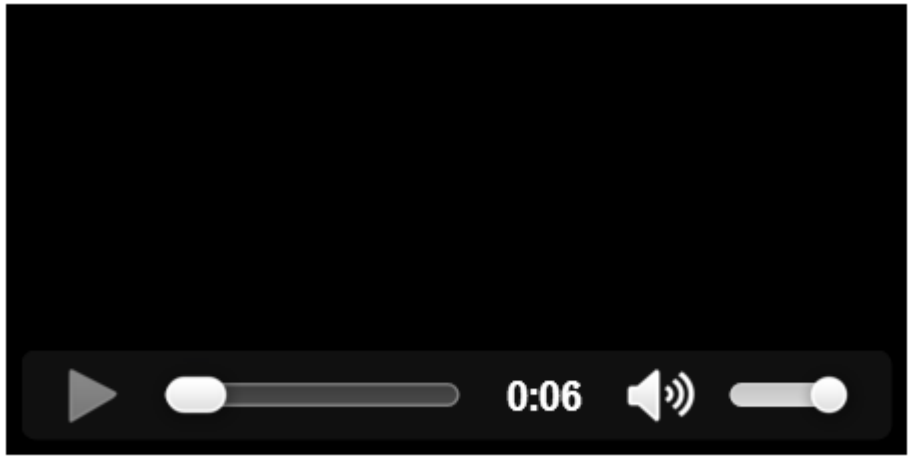
多媒体元素基本属性

video元素与audio元素相同的属性

6-1.html

- **controls属性**

指定是否为视频或音频添加浏览器自带的播放用的控制条。控制条中具有播放、暂停等按钮。其使用方法如下：




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

- **width属性与height属性（video元素独有属性）**

用于指定视频的宽度与高度（以像素为单位），使用方法如下：

```
<video src="sample.mov" width="500" height="500"></video>
```

多媒体元素基本属性

video元素与audio元素相同的属性

- **error属性**

在读取、使用媒体数据的过程中，在正常情况下，该属性为null，但是任何时候只要出现错误，该属性将返回一个MediaError对象，该对象的code属性返回对应的错误状态码，其可能的值包括：

- MEDIA_ERR_ABORTED（数值1）：媒体数据的下载过程由于用户的操作原因而被终止。
- MEDIA_ERR_NETWORK（数值2）：确认媒体资源可用，但是在下载时出现网络错误，媒体数据的下载过程被终止。
- MEDIA_ERR_DECODE(数值3）：确认媒体资源可用，但是解码时发生错误。
- MEDIA_ERR_SRC_NOT_SUPPORTED（数值4）：媒体资源不可用媒体格式不被支持。

注意：error属性为只读属性。

多媒体元素基本属性

video元素与audio元素相同的属性

```
<video id="videoElement" src="sample.mov" controls="controls"></video>
<script>
var video=document.getElementById("videoElement");
video.addEventListener(“error“,function()
        var error=video.error;
        switch(error.code)
        {
                case 1: {alert(“视频下载过程被终止.。”); break;}
                case 2: {alert(“网络发生故障”); break;}
                case 3: {alert(“解码失败”); break;}
                case 4: {alert(“不支持播放的格式”); break;}
                default: alert(“发生未知的错误”);
        }
},false);
</script>
```

多媒体元素基本属性

video元素与audio元素相同的属性

- networkState属性

该属性在媒体数据加载过程中读取当前网络的状态，其值包括：

- NETWORK_EMPTY（数值0）：元素处于初始状态。
- NETWORK_IDLE（数值1）：浏览器已选择好用什么编码格式来播放媒体，但尚未建立网络连接。
- NETWORK_LOADING（数值2）：媒体数据加载中。
- NETWORK_NO_SOURCE（数值3）：没有支持的编码格式，不执行加载。

注意：networkState属性为只读属性

多媒体元素基本属性

video元素与audio元素相同的属性

```
<video id="videoElement" src="sample.mov" controls="controls"></video>
<script>
var video=document.getElementById("videoElement");
video.addEventListener(“process“,function(e)
{
        var networkStateDisplay=document.getElementById(“networkState");
        if (vedio. networkState==2){ networkStateDisplay.InnerHTML=“加载中…”;
        }else if (vedio. networkState==3{networkStateDisplay.InnerHTML=“加载失败”;
        }
},false);
</script>
```

多媒体元素基本属性

video元素与audio元素相同的属性

- currentSrc属性、buffered属性

- 可以用currentSrc属性来读取播放中的媒体数据的URL地址，该属性为只读属性。
- buffered属性返回一个实现TimeRanges接口的对象，以确认浏览器是否已缓存媒体数据。

TimeRanges对象的作用：TimeRanges对象表示一段时间范围，在大多数情况下，该对象表示的时间范围是一个单一的以“0”开始的范围，但是如果浏览器发出Range Rquest请求，这时TimeRanges对象表示的时间范围是多个时间范围。

TimeRanges对象的属性：该对象具有一个length属性，表示有多少个时间范围，多数情况下存在时间范围时，该值为“1”；不存在时间范围时，该值为”0”，

TimeRanges对象有两个方法：start(index)和end(index)，多数情况下将index设置为“0”就可以了。当用element.buffered语句来实现TimeRanges接口时，start(0)表示当前缓存区内从媒体数据的什么时间开始进行缓存，end(0)表示当前缓存区内的结束时间。

注意： buffered属性为只读属性

多媒体元素基本属性

video元素与audio元素相同的属性

- **readyState属性**

该属性返回媒体当前播放位置的**就绪状态**，其值包括：

- HAVE_NOTHING（数值0）：**没有获取到媒体的任何信息**，当前播放位置没有可播放数据。
- HAVE_METADATA（数值1）：已经获取到了**足够的媒体数据**，但是**当前播放位置没有有效的媒体数据**（也就是说，获取到的媒体数据无效，不能播放）。
- HAVE_CURRENT_DATA（数值2）：当前播放位置已经有数据可以播放，**但没有获取到可以让播放器前进的数据**。当媒体为视频时，意思是**当前帧的数据已获得，但还没有获取到下一帧的数据**，或者**当前帧已经是播放的最后一帧**。
- HAVE_FUTURE_DATA（数值3）：当前播放位置已经有数据可以播放，而且也获取到了可以让播放器前进的数据。当媒体为视频时，意思是当前帧的数据已获取，而且也获取到了下一帧的数据，**当前帧是播放的最后一帧readyState属性不可能为HAVE_FUTURE_DATA。**
- HAVE_ENOUGH_DATA（数值4）：当前播放位置已经有数据可以播放，同时也获取到了可以让播放器前进的数据，而且浏览器确认媒体数据以某一种速度进行加载，可以保证有足够的后续数据进行播放。

注意：readyState属性为只读属性。

多媒体元素基本属性

video元素与audio元素相同的属性

●seeking属性和seekable属性

- seeking属性返回一个布尔值，表示浏览器是否正在请求某一特定播放位置的数据，true表示浏览器正在请求数据，false表示浏览器已停止请求。
- seekable属性返回一个TimeRanges对象，该对象表示请求到的数据的时间范围。当媒体为视频时，开始时间为请求到视频数据第一帧的时间，结束时间为请求到视频数据最后一帧的时间。

注意：这两个属性均为只读属性。

●currentTime属性、startTime属性和duration属性

- currentTime属性用于读取媒体的当前播放位置，也可以通过修改currentTime属性来修改当前播放位置。如果修改的位置上没有可用的媒体数据时，将抛出INVALID_STATE_ERR异常；如果修改的位置超出了浏览器在一次请求中可以请求的数据范围，将抛出INDEX_SIZE_ERR异常。
- startTime属性用来读取媒体播放的开始时间，通常为“0”。
- duration属性来读取媒体文件总的播放时间。

多媒体元素基本属性

video元素与audio元素相同的属性

- **played属性、paused属性和ended属性**

- played属性返回一个TimeRanges对象，从该对象中可以读取媒体文件的已播放部分的时间段。开始时间为已播放部分的开始时间，结束时间为已播放部分的结束时间。
- paused属性返回一个布尔值，表示是否暂停播放，true表示媒体暂停播放，false表示媒体正在播放。
- ended属性返回一个布尔值，表示是否播放完毕，true表示媒体播放完毕，false表示还没有播放完毕。

注意：三者均为只读属性。

多媒体元素基本属性

video元素与audio元素相同的属性

- **defaultPlaybackRate属性和playbackRate属性**
  - defaultPlaybackRate属性用来读取或修改媒体默认的播放速率。
  - playbackRate属性用于读取或修改媒体当前的播放速率。
- **volume属性和muted属性**
  - volume属性用于读取或修改媒体的播放音量，范围为“0”到“1”，“0”为静音，“1”为最大音量。
  - muted属性用于读取或修改媒体的静音状态，该值为布尔值，true表示处于静音状态，false表示处于非静音状态。

多媒体播放

- HTML 5多媒体的简述
- 多媒体元素基本属性
- 多媒体元素常用方法
- 多媒体元素重要事件

目录

多媒体元素常用方法

本节需要掌握的知识

1、媒体播放时的方法
2、canPlayType(type)方法

多媒体元素常用方法

媒体播放时的方法

- 使用media.play()播放视频，并会将media.paused的值强行设为false。
- 使用media.pause()暂停视频，并会将media.paused的值强行设为ture。
- 使用media.load()重新载入视频，并会将media.playbackRate的值强行设为media.defaultPlaybackRate的值，且强行将media.error的值设为Null.

例 下面来看一个媒体播放的示例。在本例中通过video元素加载一段视频文件，为了展示视频播放时所应用的方法，在控制视频的放时，并没有应用浏览器自带的控制条来控制视频的播放而是通过添加“播放”与“暂停”按钮来控制视频文件的播放与暂停。

多媒体元素常用方法

媒体播放时的方法

6-2.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"></meta>
<title>媒体播放示例</title>
<script>
var video;
function init()
{
   video = document.getElementById("video1");
   //监听视频播放结束事件
   video.addEventListener("ended", function()
   {
      alert("播放结束。");
   }, true);
}
function play()
{
   // 播放视频
   video.play();
}
function pause()
{
   //暂停播放
   video.pause();
}
</script>
</head>
<body onload="init()">
<!—可以添加controls属性来显示浏览器自带的播放用的控制条。 -->
  <video id="video1"  src="2.ogv" >
  </video><br/>
  <button onclick="play()">播放</button>
```

多媒体元素常用方法

媒体播放时的方法

使用canPlayType(type)方法测试浏览器是否支持指定的媒介类型，该方法的定义如下所示。。

```
var support=videoElement.canPlayType(type);
```

语法解释：videoElement表示页面上的video元素或audio元素。

参数说明：type参数，指定方法与source元素的type参数的指定方法相同，都用播放文件的MIME类型来指定，可以在指定的字符串中加上表示媒体编码格式的codes参数。

该方法返回3个可能值（均为浏览器判断的结果）：

- 空字符串：浏览器不支持此种媒体类型；
- maybe：浏览器可能支持此种媒体类型；
- probably：浏览器确定支持此种媒体类型.

多媒体播放

- HTML 5多媒体的简述
- 多媒体元素基本属性
- 多媒体元素常用方法
- 多媒体元素重要事件

目录

多媒体元素重要事件

本节需要掌握的知识

1、事件处理方式
2、事件介绍
3、事件示例

多媒体元素重要事件

事件处理方式

在利用video元素或audio元素读取或播放媒体数据的时候，会触发一系列的事件，如果用JavaScript脚本来捕捉这些事件，就可以对这些事件进行处理了。对于这些事件的捕捉及其处理，可以按两种方式来进行。

- 一种是监听的方式：addEventListener(“事件名”,处理函数,处理方式)方法来对事件的发生进行监听，该方法的定义如下所示。

```
videoElement.addEventListener(type,listener,useCapture);
```

语法说明：videoElement表示页面上的video元素或audio元素。type为事件名称，listener表示绑定的函数，useCapture是一个布尔值，表示该事件的响应顺序，该值如果为true，则浏览器采用Capture响应方式，如果为false，浏览器采用bubbing响应方式，一般采用false，默认情况下也为false。

- 另一种是直接赋值的方式。事件处理方式为JavaScript脚本中常见的获取事件句柄的方式。

多媒体元素重要事件

事件介绍

我们将介绍一下浏览器在请求媒体数据、下载媒体数据、播放媒体数据一直到播放结束这一系列过程中，到底会触发哪些事件。

- **loadstart事件：** 浏览器开始请求媒介；
- **progress事件：** 浏览器正在获取媒介；
- **suspend事件：** 浏览器非主动获取媒介数据，但没有加载完整个媒介资源；
- **abort事件：** 浏览器在完全加载前中止获取媒介数据，但是并不是由错误引起的；
- **error事件：** 获取媒介数据出错；
- **emptied事件：** 媒介元素的网络状态突然变为未初始化；可能引起的原因有两个：1、载入媒体过程中突然发生一个致命错误；2、在浏览器正在选择支持的播放格式时，又调用了load方法重新载入媒体。
- **stalled事件：** 浏览器获取媒介数据异常；
- **play事件：** 即将开始播放，当执行了play方法时触发，或数据下载后元素被设为autoplay（自动播放）属性。

多媒体元素重要事件

事件介绍

- **pause事件：** 暂停播放，当执行了pause方法时触发。
- **loadedmetadata事件：** 浏览器获取完媒介资源的时长和字节
- **loadeddata事件：** 浏览器已加载当前播放位置的媒介数据；
- **waiting事件：** 播放由于下一帧无效（例如未加载）而已停止（但浏览器确认下一帧会马上有效）；
- **playing事件：** 已经开始播放
- **canplay事件：** 浏览器能够开始媒介播放，但估计以当前速率播放不能直接将媒介播放完（播放期间需要缓冲）；
- **canplaythrough事件：** 浏览器估计以当前速率直接播放可以直接播放完整个媒介资源（期间不需要缓冲）；

多媒体元素重要事件

事件介绍

- **seeking事件：** 浏览器正在请求数据（seeking属性值为true）；
- **seeked事件：** 浏览器停止请求数据（seeking属性值为false）；
- **timeupdate事件：** 当前播放位置（currentTime属性）改变，可能是播放过程中的自然改变，也可能是被人为地改变，或由于播放不能连续而发生的跳变；
- **ended事件：** 播放由于媒介结束而停止；
- **ratechange事件：** 默认播放速率（defaultPlaybackRate属性）改变或播放速率（playbackRate属性）改变；
- **durationchange事件：** 媒介时长（duration属性）改变
- **volumechange事件：** 音量（volume属性）改变或静音（muted属性）。

多媒体元素重要事件

事件示例

例 本节中，将通过一个实例来讲解一下多媒体元素事件的用法，在本例中将在页面中显示要播放的多媒体文件，同时显示多媒体文件的总时间，当单击“播放”按钮时，将显示当前播放的时间。多媒体文件的总时间与当前时间将以（时：分：秒）的形式显示。

多媒体元素重要事件

事件示例

6-3.html

```
<!DOCTYPE html>
<html lang="en"><head>
<meta charset=utf-8 />
<title>事件捕捉实例</title>
</head>
<body>
  <video >
    <source  src="1.mp4" type="video/ogg" />
   </video>
  <div class="videochrome paused">
   <div class="controls">
    <div class="scrub">
     <table width="150" border="0" cellpadding="0" cellspacing="0">
```

多媒体元素重要事件

事件示例

```
            <tr>
                    <td width="50" scope="row"><button class="play" title="play">播放</button></td>
            <td width="50" align="center"><div class="duration">总时长：  0:00</div></td>
            <td width="50" align="center"><div class="loaded"><div class="buffer"><div class="playhead"><span>总时长:
  0:00</span></div></div></div></td>
            </tr>
     </table>
    </div>
   </div>
  </div>
<script>
var video = document.getElementsByTagName('video')[0],
  //通过querySelector方法获标签的值并赋给变量
```

多媒体元素重要事件

事件示例

```
wrapper = document.querySelector('.videochrome'),
buffer = document.querySelector('.videochrome .controls .buffer'),
playhead = buffer.querySelector('.playhead'),
play = wrapper.querySelector('.play'),
duration = wrapper.querySelector('.duration'),
currentTime = playhead.querySelector('span');
video.addEventListener('loadeddata', canplay, false);    //使用事件监听准备播放
video.addEventListener('play', playEvent, false);        //使用事件播放
video.addEventListener('pause', pausedEvent, false);     //播放暂停
video.addEventListener('ended', function () {  //播放结束后停止播放
 this.pause();       //显示暂停播放
}, false);
video.addEventListener('durationchange', updateSeekable, false);   //播放的时长被改变
```

多媒体元素重要事件

事件示例

```
video.addEventListener('timeupdate', updatePlayhead, false);        //使用事件监听方式捕捉事件
function canplay() {              //调用canplay函数初始化媒体
  initControls();
}
function playEvent() {
  play.innerHTML = '暂停';
}
function pausedEvent() {
  play.innerHTML = '播放';
}
function asTime(t) {
  t = Math.round(t);
  var s = t % 60;
```

多媒体元素重要事件

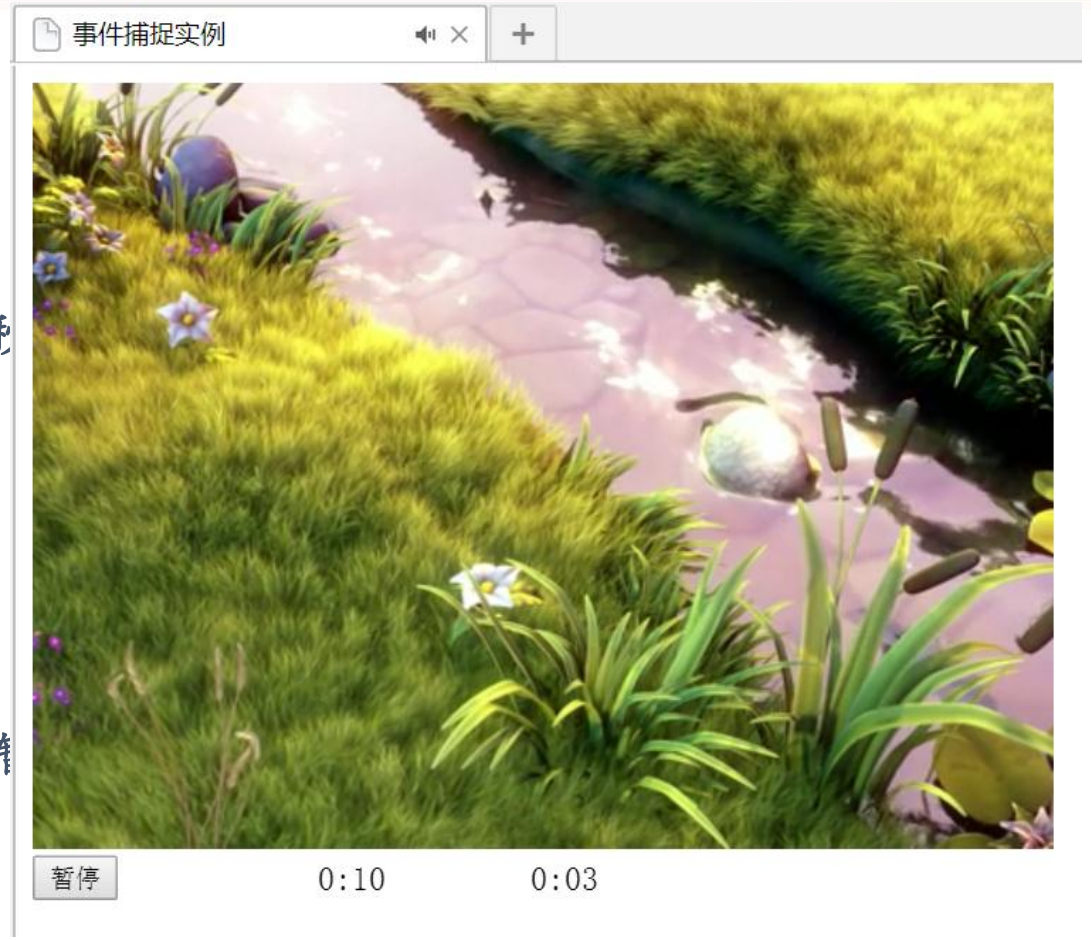
事件示例

```
var m = ~~(t / 60);
  return m + ':' + two(s);
}
function two(s) {
  s += "";
  if (s.length < 2) s = "0" + s;
  return s;
}
function updateSeekable() {
  duration.innerHTML = asTime(video.duration); //媒体文件的总播放时间
}
function updatePlayhead() {
  currentTime.innerHTML = asTime(video.currentTime);  //媒体的当前播放时间
```

多媒体元素重要事件

事件示例

```
}
function initControls() {
  duration.innerHTML = asTime(video.duration);  //将播放时间以分和秒
  play.onclick = function () {
    if (video.ended) {  //如果媒体播放结束，播放时间从0开始
      video.currentTime = 0;
    }
    video[video.paused ? 'play' : 'pause']();  //通过三元运算执行播放和暂
  };
}
</script>
</body>
</html>
```



HTML5开发教程

- HTML 5的新特性
- HTML 5与HTML 4的区别
- HTML 5的结构
- HTML 5中的表单
- HTML 5中的文件与拖放
- 多媒体播放
- 绘制图形
- 数据存储
- 获取地理位置信息
- 练习

目录

绘制图形

canvas的基础知识

在画布中使用路径

运用样式与颜色

绘制渐变图形

绘制变形图形

组合多个图形

给图形绘制阴影

应用图像

绘制文字

保存与恢复状态

文件的保存

对画布绘制实现动画

综合实例——桌面时钟

目录

Canvas的基础知识

本节需要掌握的知识

1、canvas是什么
2、在页面中放置canvas元素
3、绘制带边框矩形

Canvas的基础知识

canvas是什么

概述：Canvas元素是HTML5中新增的一个重要元素，专门用来绘制图形。在页面上放置一个canvas元素，就相当于在页面上放置了一块“画布”，可以在其中进行图形的描绘。

语法格式：<canvas></canvas>

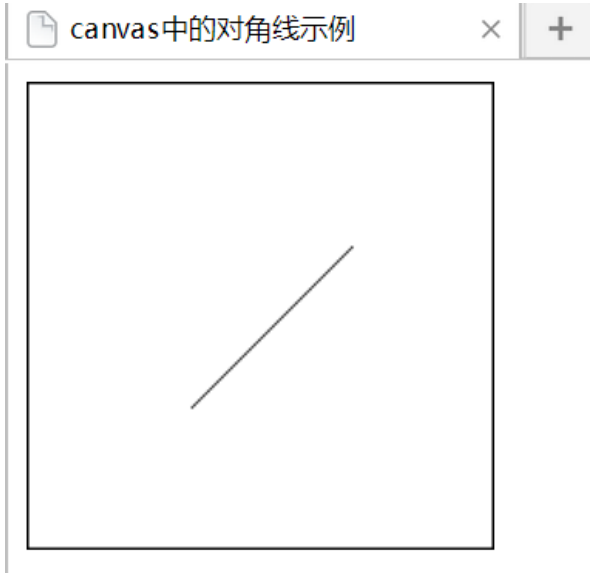
注意，在canvas元素里进行绘画，并不是指拿鼠标来作画。在网页上使用canvas元素时，它会创建一块矩形区域。默认情况下该矩形区域**宽为300像素，高为150像素**，用户可以自定义具体的大小或者设置canvas元素的其他特性。在页面中加入了canvas元素后，我们便可以通过JavaScript来自由地控制它。可以在其中添加图片、线条以及文字，也可以在里面绘图，甚至还可以加入高级动画。

Canvas的基础知识

在页面中放置canvas元素

7-1.html

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>canvas中的对角线示例</title>
</head>
<body>
<canvas id="djx" style="border: 1px solid;"  width="200" height="200"> </canvas>
<script>
    function drawDiagonal() {
        // 取得canvas元素及其绘图的上下文
        var canvas = document.getElementById('djx');
        var context = canvas.getContext('2d');
        // 用绝对坐标来创建一条路径
        context.beginPath();
        context.moveTo(70, 140); //起点和终点
        context.lineTo(140, 70); //画线
        // 将这条线绘制到canvas上
        context.stroke();
    }
    window.addEventListener("load", drawDiagonal, true);
</script>
</body>
</html>
```



Canvas的基础知识

绘制带边框矩形

本节中将详细介绍如何在canvas画布中绘制一个矩形。在本例中调用了脚本文件中的draw函数进行图形描绘。该函数放置在body的属性中，使用onload=“draw(‘canvas’);”语句。调用脚本文件中的draw函数进行图像描画。在本例中draw函数的功能是把canvas画布的背景用浅蓝色涂满，然后画出一个绿色正方形，边框为红色.

Canvas的基础知识

绘制带边框矩形

7-2.html

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>canvas元素示例</title>
<script >
function draw(id) {
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    context.fillStyle = “green”; //填充样式
    context.strokeStyle = “red”; //边框样式
    context.lineWidth=1; //宽度
    context.fillRect(50,50,100,100); //填充矩形
    context.strokeRect(50,50,100,100); //填充边框
}
</script>
</head>
<body onload="draw('canvas');">
<h1>canvas中画矩形</h1>
<canvas id="canvas" width="400" height="300" />
</body>
</html>
```

绘制图形

目录

在画布中使用路径

本节需要掌握的知识

1、使用arc方法绘制圆形
2、使用moveTo与lineTo路径绘制火柴人
3、贝塞尔和二次方曲线

在画布中使用路径

使用arc方法绘制圆形

要想绘制其他图形，需要使用路径。同绘制矩形一样，绘制开始时还是要取得图形上下文，然后需要执行如下步骤。

✓ 开始创建路径

使用图形上下文对象的**beginPath()**方法，该方法的定义如下所示**context.beginPath()**该方法不使用参数。通过调用该方法，开始路径的创建

✓ 创建图像的路径

使用图形上下文对象的act方法。该方法的定义如下所示。

```
context.arc(x,y,radius, startAngle, endAngle,anticlockwise)
```

该方法使用六个参数，**x**为绘制圆形的起点横坐标，**y**为绘制圆形的起点纵坐标，**radius**为圆形半径，**startAngle**为开始角度，**endAngle**为结束角度，**anticlockwise**为是否按顺时针方向进行绘制。在canvas API中，绘制半径与弧时指定的参数为开始弧度与结束弧度，如果习惯使用角度，请使用如下所示的方法将角度转换为弧度。

```
var radians =degrees*math.PI/180
```

其中**math.PI**表示角度为**180**度，**math.PI*2**表示角度为**360**度。

arc方法不仅可以用来绘制圆形，也可以用来绘制圆弧。因此，使用时必须要指定开始角度与结束角度。因为这两个角度决定了弧度。**Anticlockwise**参数为一个布尔值的参数，参数值为**true**时，按顺时针绘制；参数值为**false**时，按逆时针方向绘制

在画布中使用路径

使用arc方法绘制圆形

✓ 关闭路径

路径创建完成后，使用图形上下文对象的closePath方法将路径关闭。该方法定义如下所示。

```
context.closePath();
```

将路径关闭后，路径的创建工作就完成了，但是需要注意的是，这时只是路径创建完毕而已，还没有真正绘制图形。

✓ 进行圆形绘制，并设定绘制样式。实现的代码如下所示。

```
context.fillStyle = 'rgba(255, 0, 0, 0.25)';
context.fill();
```

在画布中使用路径

使用arc方法绘制圆形

7-3.html

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>canvas元素示例</title>
<script >
function draw(id)
{
   var canvas = document.getElementById(id);
   if (canvas == null)
      return false;
   var context = canvas.getContext('2d');
   context.fillStyle = "#EEEEFF";
   context.fillRect(0, 0, 400, 300);

      context.beginPath();
      context.arc( 100, 100, 75, 0, Math.PI * 2, true);//x=100,y=100, radius=75, startAngle=0, endAngle=360 ,anticlockwise=true
      context.closePath();
```

在画布中使用路径

使用arc方法绘制圆形

```
context.fillStyle = 'rgba(255, 0, 0, 0.25)';
context.fill();

}
</script>
</head>
<body onload="draw('canvas');">
<h1>canvas中画圆形</h1>
<canvas id="canvas" width="400" height="300" />
</body>
</html>
```

在画布中使用路径

使用moveTo与lineTo路径绘制火柴人

- **moveTo(x,y)**：不绘制，只是将当前位置移动到新的目标坐标(x，y)。
- **lineTo(x,y)**：不仅将当前位置移动到新的目标坐标(x，y)，而且在两个坐标之间画一条直线。

简而言之，上面两个函数的区别在于：moveTo就像是提起画笔，移动到新位置，而lineTo告诉canvas用画笔从纸上的旧坐标画条直线到新坐标。需要提醒大家注意的是，不管调用它们哪一个，都不会真正画出图形，因为我们还没有调用stroke或者fill函数。目前，我们只是在定义路径的位置，以便后面绘制时使用。

我们再来看一个特殊的路径函数叫做closePath。这个函数的行为和lineTo很像，唯一的差别在于closePath会将路径的起始坐标自动作为目标坐标。closePath还会通知canvas当前绘制的图形已经闭合或者形成了完全封闭的区域，这对将来的填充和描边都非常有用。

在画布中使用路径

使用moveTo与lineTo路径绘制 火柴人

7-4.html

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>canvas元素示例</title>
<script >
function draw(id)
{
    var canvas = document.getElementById(id);
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 300, 300
}
```

```
context.beginPath();
context.strokeStyle = '#c00';
context.lineWidth = 3;
context.arc(100, 50, 30, 0, Math.PI*2, true);
context.fill();
context.stroke();

context.beginPath();
context.strokeStyle = '#c00';
context.lineWidth = 3;
context.arc(100, 50, 20, 0, Math.PI, false);
context.fill();
context.stroke();

context.beginPath();
context.fillStyle = '#c00';
context.arc(90, 45, 3, 0, Math.PI*2, true);
context.fill();
context.stroke();
context.moveTo(113, 45);
```

在画布中使用路径

使用moveTo与lineTo路径绘制 火柴人

```
        context.arc(110, 45, 3, 0, Math.PI*2, true);
        context.fill();
        context.stroke();

        context.beginPath();
        context.moveTo(100, 80); // move to neck
        context.lineTo(100, 180); // body
        context.lineTo(75, 250); // 绘制左腿
        context.moveTo(100, 180); // move to hips
        context.lineTo(125, 250); // 绘制右腿
        context.moveTo(100, 90); // move to shoulders
        context.lineTo(75, 140); // 绘制左胳膊
        context.moveTo(100, 90); // back to shoulders
        context.lineTo(125, 140); // 绘制右胳膊
        context.stroke();
}
   context.closePath();
```

```
</script>
</head>
<body onload="draw('canvas');">
<h1>canvas中绘制 火柴人</h1>
<canvas id="canvas" width="400" height="300" />
</body>
</html>
```

在画布中使用路径

贝塞尔和二次方曲线

绘制贝塞尔曲线主要使用**bezierCurveTo**方法。该方法可以说是lineTo的**曲线版**，将从当前坐标点到指定坐标点中间的贝塞尔曲线追加到路径中。该方法的定义如下所示。

bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)

该方法使用六个参数。绘制贝塞尔曲线的时候，需要两个控制点，**cp1x**为第一个控制点的横坐标，**cp1y**为第一个控制点的纵坐标；**cp2x**为第二个控制点的横坐标，**cp2y**为第二个控制点的纵坐标；**x**为贝塞尔曲线的**终点横坐标**，**y**为贝塞尔曲线的**终点纵坐标**。

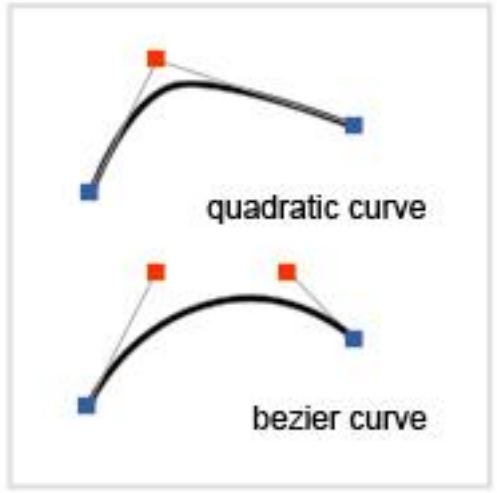
在画布中使用路径

绘制二次样条曲线

绘制二次样条曲线，使用的方法是quadraticCurveTo。该方法的定义如下所示。

quadraticCurveTo(cp1x, cp1y, x, y)

两种方法的区别如下图所示。它们都是一个起点,一个终点（图中的蓝点），但二次方贝塞尔曲线只有一个（红色）控制点）而三次方贝塞尔曲线有两个。参数 x 和 y 是终点坐标，cp1x 和 cp1y 是第一个控制点的坐标，cp2x 和 cp2y 是第二个的。



在画布中使用路径

应用实例

例 下面我们先来看一下使用bezierCurveTo方法的实例。本例中我们使用bezierCurveTo方法绘制一个红色实心的的红心。

7-5.html

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>canvas元素示例</title>
<script >
function draw(id)
{
   var canvas = document.getElementById(id);
   var context = canvas.getContext('2d');
   context.fillStyle = "#EEEEFF";
   context.fillRect(0, 0, 150, 150);
            context.beginPath();
            context.fillStyle = '#c00';
            context.strokeStyle = '#c00';
            context.moveTo(75,40);
            context.bezierCurveTo(75,37,70,25,50,25);
            context.bezierCurveTo(20,25,20,62.5,20,62.5);
            context.bezierCurveTo(20,80,40,102,75,120);
            context.bezierCurveTo(110,102,130,80,130,62.5);
            context.bezierCurveTo(130,62.5,130,25,100,25);
            context.bezierCurveTo(85,25,75,37,75,40);
            context.fill();
            context.stroke();
}
   context.closePath();
</script>
</head>
<body onload="draw('canvas');"
<h1>canvas中绘制红心</h1>
<canvas id="canvas" width="400
</body>
</html>
```

在画布中使用路径

应用实例

例 下面再来看一下使用quadraticCurveTo方法绘制二次方线的实例。本例主要是绘制了一个用于解释说明的对话框。

7-6.html

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>canvas元素示例</title>
<script >
function draw(id)
{
   var canvas = document.getElementById(id);
   var context = canvas.getContext('2d');
   context.fillStyle = "#EEEEFF";
   context.fillRect(0, 0, 150, 150);
            context.beginPath();
            context.moveTo(75,25);
            context.strokeStyle = '#c00';
            context.quadraticCurveTo(25,25,25,62.5);
            context.quadraticCurveTo(25,100,50,100);
            context.quadraticCurveTo(50,120,30,125);
            context.quadraticCurveTo(60,120,65,100);
            context.quadraticCurveTo(125,100,125,62.5);
            context.quadraticCurveTo(125,25,75,25);
            context.st
            context.fil
}
   context.closePath
</script>
</head>
<body onload="dra
<h1>canvas中绘制
<canvas id="canvas
</body>
</html>
```

canvas中绘制对话框

绘制图形

canvas的基础知识

在画布中使用路径

运用样式与颜色

绘制渐变图形

绘制变形图形

组合多个图形

给图形绘制阴影

应用图像

绘制文字

保存与恢复状态

文件的保存

对画布绘制实现动画

综合实例——桌面时钟

目录

运用样式与颜色

本节需要掌握的知识

1、fillStyle 和 strokeStyle属性
2、透明度 globalAlpha
3、线型 Line styles

运用样式与颜色

fillStyle 和 strokeStyle属性

如果想要给图形上色，有两个重要的属性可以做到：**fillStyle** 和 **strokeStyle**。这两个属性的定义方法如下所示。

```
fillStyle = color
strokeStyle = color
```

strokeStyle 是用于设置图形轮廓的颜色，而 fillStyle用于设置填充颜色。color 可以是表示 CSS 颜色值的字符串、渐变对象或者图案对象。默认情况下，线条和填充颜色都是黑色(CSS 颜色值 #000000)。这里需要注意的是如果自定义颜色则应该保证输入符合 CSS颜色值标准的有效字符串。下面的代码都是符合标准的颜色表示方式，都表示同一种颜色（橙色）。

```
context.fillStyle = "orange";
context.fillStyle = "#FFA500";
context.fillStyle = "rgb(255,165,0)";
context.fillStyle = "rgba(255,165,0,1)";
```

运用样式与颜色

fillStyle 和 strokeStyle属性

```
function draw(id)
{
    var canvas = document.getElementById(id);
    var context = canvas.getContext('2d');
    for (var i=0;i<6;i++){
    for (var j=0;j<6;j++){
    context.fillStyle = 'rgb(' + Math.floor(255-42.5*i) + ',' + Math.floor(255-42.5*j) + ',0)';
     context.fillRect(j*25,i*25,25,25);
}
  }
}
</script>
```

7-7.html

运用样式与颜色

fillStyle 和 strokeStyle属性

7-8.html

```
function draw(id) {
   var context = document.getElementById('canvas').getContext('2d');
   for (var i=0;i<6;i++){
    for (var j=0;j<6;j++){
      context.strokeStyle = 'rgb(0,' + Math.floor(255-42.5*i) + ',' +
                  Math.floor(255-42.5*j) + ')';
      context.beginPath();
      context.arc(12.5+j*25,12.5+i*25,10,0,Math.PI*2,true);
      context.stroke();
    }
   }
 }
</script>
```

运用样式与颜色

透明度 globalAlpha

通过设置 globalAlpha属性或者使用一个半透明颜色作为轮廓或填充的样式来绘制透明或半透明的图形。globalAlpha属性定义代码如下所示

```
globalAlpha = transparency value
```

这个属性影响到 canvas 里所有图形的透明度，其有效的值范围是0.0 （完全透明）到 1.0（完全不透明），默认是 1.0。 globalAlpha 属性在需要绘制大量拥有相同透明度的图形时候相当高效

例 下面 通过一个示例来了解一下globalAlpha属性的应用。本例中用四色格作为背景，设置 globalAlpha 为 0.3后，在上面画一系列半径递增的半透明圆。最终结果是一个径向渐变效果。圆叠加得越更多，原先所画的圆的透明度会越低。通过增加循环次数，画更多的圆，背景图的中心部分会完全消失。

运用样式与颜色

透明度 globalAlpha

7-9.html

```
<script >
function draw(id) {
    var context = document.getElementById('canvas').getContext('2d');
    context.fillStyle = '#FD0';
    context.fillRect(0,0,75,75);
    context.fillStyle = '#6C0';
    context.fillRect(75,0,75,75);
    context.fillStyle = '#09F';
    context.fillRect(0,75,75,75);
    context.fillStyle = '#F30';
    context.fillRect(75,75,75,75);
    context.fillStyle = '#FFF';
    context.globalAlpha = 0.3;
    for (var i=0;i<7;i++){
        context.beginPath();
        context.arc(75,75,10+10*i,0,Math.PI*2,true);
        context.fill();
    }
}
</script>
```

globalAlpha 属性实例

运用样式与颜色

线型 Line styles

- 线型包括如下属性：

lineWidth = value

lineCap = type

lineJoin = type

miterLimit = value

通过这些属性来设置线的样式。下面将结合实例来讲解一下各属性的应用及应用后的效果.

➢ **lineWidth 属性**

该属性设置当前绘线的粗细，属性值必须为正数。默认值是1.0。线宽是指给定路径的中心到两边的粗细。换句话说就是在路径的两边各绘制线宽的一半。因为画布的坐标并不和像素直接对应，当需要获得精确的水平或垂直线的时候要特别注意。

运用样式与颜色

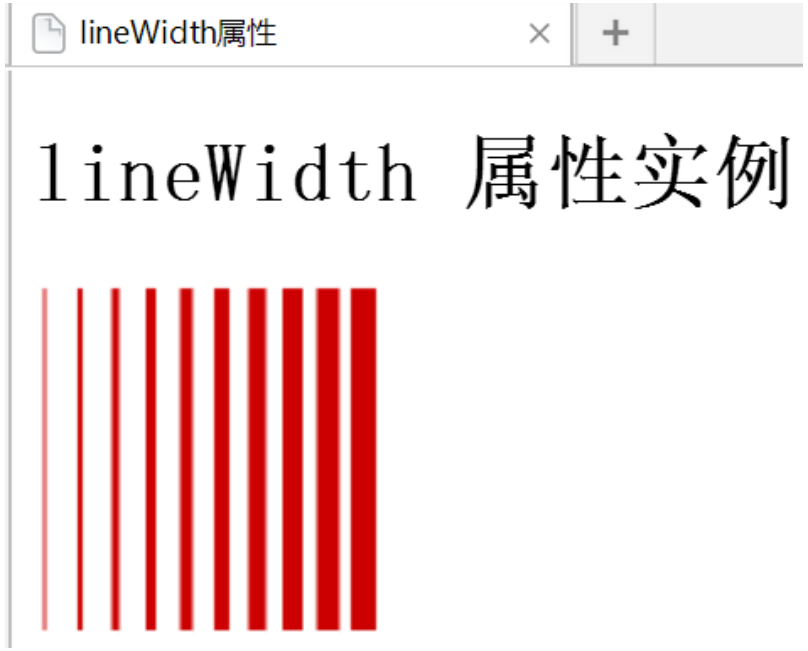
线型 Line styles

➢ lineWidth 属性

该属性设置当前绘线的粗细，属性值必须为正数。默认值是1.0。线宽是指给定路径的中心到两边的粗细。换句话说就是在路径的两边各绘制线宽的一半。因为画布的坐标并不和像素直接对应，当需要获得精确的水平或垂直线的时候要特别注意。

7-10.html

```
function draw(id) {
  var context = document.getElementById('canvas').getContext('2d');
  for (var i = 0; i < 10; i++){
  context.lineWidth = 1+i;
  context.beginPath();
    context.strokeStyle = '#c00';
  context.moveTo(5+i*14,5);
  context.lineTo(5+i*14,140);
  context.stroke();
}}
```



运用样式与颜色

线型 Line styles

➢lineCap 属性

该属性决定了线段端点显示的样子。它可以为下面的三种值之一：butt、round 和 square，默认是 butt。

7-11.html

```
function draw(id) {
  var context =
document.getElementById('canvas').getContext('2d');
  var lineCap = ['butt','round','square'];

      context.strokeStyle = '#09f';
      context.beginPath();
      context.moveTo(10,10);
      context.lineTo(140,10);
      context.moveTo(10,140);
      context.lineTo(140,140);
      context.stroke();
```

```
context.strokeStyl
 for (var i=0;i<line
  context.lineWid
  context.lineCap
  context.beginPa
  context.moveTo
  context.lineTo(2
  context.stroke()
 }
}
```

运用样式与颜色

线型 Line styles

➢lineJoin 属性

该属性值决定了图形中两线段连接处所显示的样子。它可以是以下三种值之一：round, bevel 和 miter。默认是 miter。

7-12.html

```
function draw(id) {
    var context =
document.getElementById('canvas').getContext('2d');
    var lineJoin = ['round','bevel','miter'];
                        context.strokeStyle = '#09f';
                        context.lineWidth = 10;
  for (var i=0;i<lineJoin.length;i++){
              context.lineJoin = lineJoin[i];
              context.beginPath();
              context.moveTo(-5,5+i*40);
              context.lineTo(35,45+i*40);
              context.lineTo(75,5+i*40);
              context.lineTo(115,45+i*40);
              context.lineTo(155,5+i*40);
context.stroke();
  }
}
```

绘制图形

目录

绘制渐变图形

本节需要掌握的知识

1、绘制线性渐变
2、绘制径向渐变

绘制渐变图形

绘制线性渐变

渐变的概念：渐变是指在填充时从一种颜色慢慢过渡到另外一种的颜色。

绘制线性渐变时，需要使用到LinearGradient对象。使用图像上下文对象的createLinearGradient方法创建该对象。该方法的定义如下所示。

```
context.createLinearGradient(xStart,yStart,xEnd,yEnd)
```

该方法使用四个参数，**xStart**为渐变起始地点的横坐标，**yStart**为渐变起始地点的纵坐标，**xEnd**为渐变结束地点的横坐标，**yEnd**为渐变结束地点的纵坐标。

绘制渐变图形

使用addColorStop方法绘制颜色

创建了一个使用两个坐标点的LinearGradient对象之后，使用addColorStop方法进行绘制颜色，该方法的定义如下所示。

context. addColorStop(offset,color)

该方法使用两个参数——offset和color。Offset为所设定的颜色离开渐变起始点的偏移量。该参数的值是一个范围在0到1之间的浮点值，渐变起始点的偏移量为0，渐变结束点的偏移量为1。

```
function draw(id) {                                    7-13.html
    var context =
document.getElementById('canvas').getContext('2d');
    var lingrad = context.createLinearGradient(0,0,0,150);
                        lingrad.addColorStop(0, 'black');
                        lingrad.addColorStop(1, 'white');
                        context.fillStyle = lingrad;
                        context.fillRect(10,10,130,130);
}
```

绘制渐变图形

绘制径向渐变

径向渐变是指沿着圆形的半径方向向外进行扩散的渐变方式。譬如在绘制太阳时，沿着太阳的半径方向向外扩散出去的光晕，就是一种径向渐变使用图形上下文对象的createLinearGradient方法绘制径向渐变，该方法的定义如下所示。

```
context.createRadialGradient(xStart,yStart,radiusStart,xEnd,yEnd,radiusEnd)
```

该方法使用六个参数，xStart为渐变开始圆的圆心横坐标，yStart为渐变开始圆的圆心纵坐标，radiusStart为开始圆的半径，xEnd为渐变结束圆的圆心横坐标, yEnd 为渐变结束圆纵的坐标，radiusEnd为结束圆的半径。

在这个方法中，分别指定了两个圆的大小与位置。从第一个圆的圆心处向外进行扩散渐变，一直扩散到第二个圆的外轮廓处。在设定颜色时，与线性渐变相同，使用的是addColorStop方法进行设定。同样是需要设定0到1之间的浮点数来作为渐变转折点的偏移量。

绘制渐变图形

绘制径向渐变

7-14.html

```
function draw(id) {
    var context = document.getElementById('canvas').getContext('2d');
    var radgrad = context.createRadialGradient(45,45,10,52,50,30);
        radgrad.addColorStop(0, '#A7D30C');
        radgrad.addColorStop(0.9, '#019F62');
        radgrad.addColorStop(1, 'rgba(1,159,98,0)');
  var radgrad2 = context.createRadialGradient(105,105,20,112,120,50);
        radgrad2.addColorStop(0, '#FF5F98');
        radgrad2.addColorStop(0.75, '#FF0188');
        radgrad2.addColorStop(1, 'rgba(255,1,136,0)');
    var radgrad3 = context.createRadialGradient(95,15,15,102,20,40);
        radgrad3.addColorStop(0, '#00C9FF');
        radgrad3.addColorStop(0.8, '#00B5E2');
        radgrad3.addColorStop(1, 'rgba(0,201,255,0)');
    var radgrad4 = context.createRadialGradient(0,150,50,0,140,90);
        radgrad4.addColorStop(0, '#F4F201');
```

绘制渐变图形

绘制径向渐变

```
        radgrad4.addColorStop(0.8, '#E4C700');
        radgrad4.addColorStop(1, 'rgba(228,199,0,0)');

    context.fillStyle = radgrad4;
    context.fillRect(0,0,150,150);
    context.fillStyle = radgrad3;
    context.fillRect(0,0,150,150);
    context.fillStyle = radgrad2;
    context.fillRect(0,0,150,150);
    context.fillStyle = radgrad;
    context.fillRect(0,0,150,150);
}
```

绘制图形

canvas的基础知识

在画布中使用路径

运用样式与颜色

绘制渐变图形

绘制变形图形

组合多个图形

给图形绘制阴影

应用图像

绘制文字

保存与恢复状态

文件的保存

对画布绘制实现动画

综合实例——桌面时钟

目录

绘制变形图形

本节需要掌握的知识

1、坐标的变换
2、矩阵变换

绘制变形图形

坐标的变换

对坐标的变换处理，有如下三种方式。

- 平移

移动图形的绘制主要是通过translate 方法来实现的，该方法定义如下所示。

```
context. translate(x, y);
```

translate 方法使用两个参数——x表示将坐标轴原点向左移动多少个单位，默认情况下为像素；y表示将坐标轴原点向下移动多少个单位。

- 缩放

使用图形上下文对象的scale方法将图形缩放。该方法的定义如下所示。

```
context.scale(x,y);
```

绘制变形图形

坐标的变换

scale方法使用两个参数，x是水平方向的放大倍数，y是垂直方向的放大倍数。将图形缩小的时候，将这两个参数设置为0到1之间的小数就可以了，例如0.5是指将图形缩小一半。

- 旋转

使用图形上下文对象的rotate方法将图形进行旋转。该方法的定义如下所示。

```
context.rotate(angle);
```

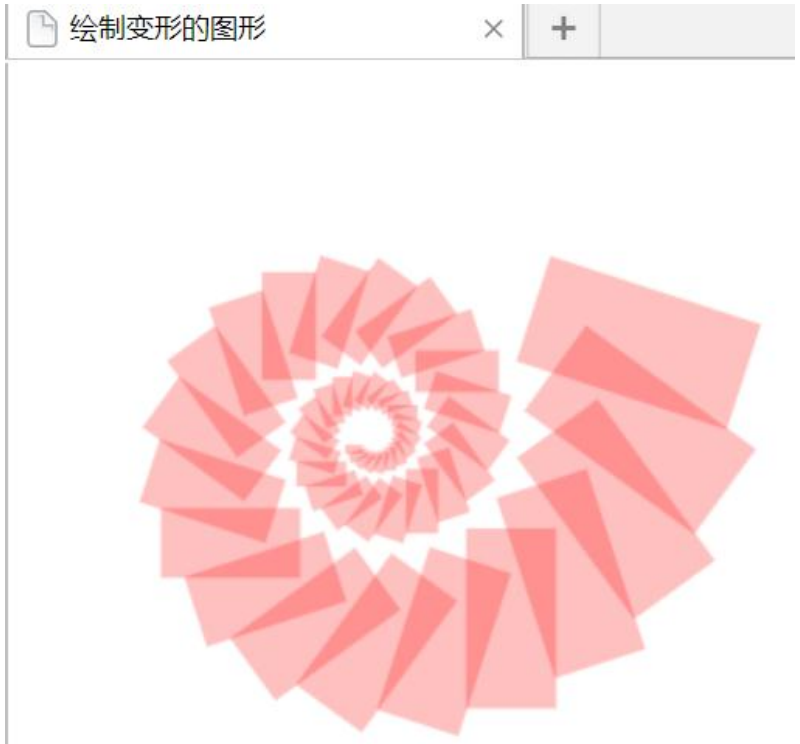
rotate方法接受一个参数angle，angle是指旋转的角度，旋转的中心点是坐标轴的原点。旋转是以顺时针方向进行的，要想逆时针旋转时，将angle设定为负数就可以了。

绘制变形图形

坐标的变换

7-15.html

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#FFF"; //设置背景色为白色
    context.fillRect(0, 0, 400, 300); //创建一个画布
    // 图形绘制
    context.translate(200,50);
    context.fillStyle = 'rgba(255,0,0,0.25)';
    for(var i = 0;i < 50;i++)
      {
        context.translate(25,25); //图形向左，向下各移动25
        context.scale(0.95,0.95); //图形缩放
        context.rotate(Math.PI / 10); //图形旋转
        context.fillRect(0,0,100,50);
      }
}
```



绘制变形图形

矩阵变换

变换矩阵，这个矩阵是专门用来实现图形变形的，它与坐标一起配合使用，以达到变形的目的。当图形上下文被创建完毕时，事实上也创建了一个默认的变换矩阵，如果不对这个变换矩阵进行修改，那么接下来绘制的图形将以画布的最左上角的坐标原点绘制图形，绘制出来的图形也经过缩放、变形的处理，但是如果对这个变换矩阵进行修改，那么情况将会是不一样的。

使用图形上下文对象的transform方法修改变换矩阵，该方法的定义如下所示。

transform(m11, m12, m21, m22, dx, dy)

该方法使用一个新的变换矩阵与当前变换矩阵进行乘法运算，该变换矩阵的形式如下所示。

m11	m21	dx
m12	m22	dy
0	0	1

其中**m11，m21，m12，m22**四个参数用来修改使用这个方法之后绘制图形时的计算方法，以达到变形目的，**dx与dy**参数移动坐标原点，dx表示将坐标原点在x轴上向右移动x个单位，dy表示将坐标原点在y轴上向下移动y个单位。默认情况下以像素为单位。

绘制变形图形

矩形变换实例

translate(x,y)->context.translate(1,0,0,1,x,y)或context.translate(0,1,1,0,x,y)

scale(x,y)->context.translate(x,0,0,y,0,0)或context.translate(0,y,x,0,0,0)

rotate(angle)->

```
context.translate(Math.cos(angle*Math.PI/180), Math.sin(angle*Math.PI/180), -
    Math.sin(angle*Math.PI/180), Math.cos(angle*Math.PI/180),0,0);
```

或

```
context.translate(Math.sin(angle*Math.PI/180), Math.cos(angle*Math.PI/180),
    Math.cos(angle*Math.PI/180), Math.sin(angle*Math.PI/180),0,0);
```

绘制变形图形

矩形变换实例

例 下面通过实例来看一下transform方法的工作原理。在该实例中，用循环的方法绘制了几个圆弧，圆弧的大小与位置均不变，只是使用了transform方法让坐标原点每次向下移动10个像素，使得绘制出来的圆弧相互重叠，然后对圆弧设置七彩颜色，使这些圆弧的外观达到彩虹的效果。

绘制变形图形

矩形变换实例

7-16.html

```
function draw(id)
{
   var canvas = document.getElementById(id);
   var context = canvas.getContext('2d');
   /* 定义颜色 */
   var colors = ["red", "orange", "yellow", "green", "blue", "navy", "purple"];
   /* 定义线宽*/
   context.lineWidth = 10;
   context.transform(1, 0, 0, 1, 100,0)
   /*循环绘制圆弧*/
   for( var i=0; i<colors.length; i++ )
   {
      /* 定义每次向下移动10个像素的变换矩阵 */
      context.transform(1, 0, 0, 1, 0, 10);
      /* 设定颜色 */
      context.strokeStyle = colors[i];
      /* 绘制圆弧 */
      context.beginPath();
      context.arc(50, 100, 100, 0, Math.PI, true);
      context.stroke();
   }
}
```

绘制变形图形

矩形变换实例

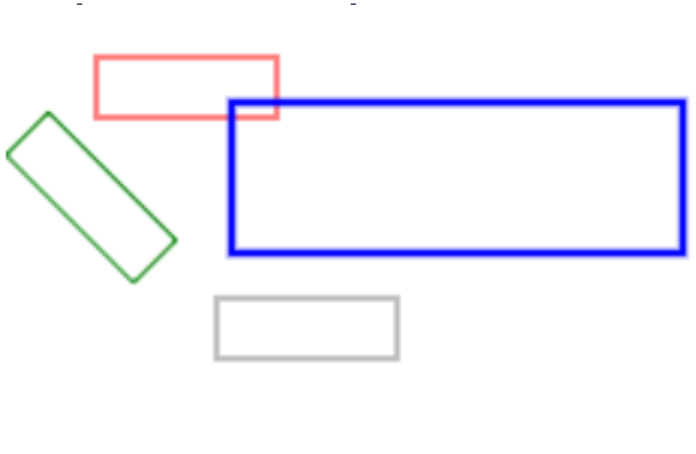
例下面通过实例来了解一下setTransform的具体的使用方法。在该实例中首先创建一个红色边框的长方形，然后将该长方形顺时针旋转45度，绘制出一个新的长方形，并且绘制其边框为绿色，然后将红色长方形扩大2.5倍绘制新的长方形，边框为蓝色，最后在红色长方形右下方绘制同样大小的长方形，边框为灰色。

绘制变形图形

矩形变换实例

7-17.html

```
function draw(id)
{
   var canvas = document.getElementById(id);
   var context = canvas.getContext('2d');
   /* ------------绘制红色长方形-------- */
   context.strokeStyle = "red";
   context.strokeRect(30, 10, 60, 20);
   /* -----绘制顺时针旋转45°后的绿色长方形------ */
   var rad = 45 * Math.PI / 180;          //绘制45度圆弧
  context.setTransform(Math.cos(rad), Math.sin(rad), -Math.sin(rad),
Math.cos(rad), 0, 0 );          //定义顺时针旋转45°的变换矩阵
   /* ----------绘制图形---- */
   context.strokeStyle = "green";
   context.strokeRect(30, 10, 60, 20);
   /* ------绘制放大2.5倍后的蓝色长方形-------- */
   context.setTransform(2.5, 0, 0, 2.5, 0, 0);   //定义放大2.5倍
的变换矩阵
   /* 绘制图形 */
   context.strokeStyle = "blue";
   context.strokeRect(30, 10, 60, 20);
   /* 将坐标原点向右移动40像素，向下移动80像素后绘制
灰色长方形*/
   context.setTransform(1, 0, 0, 1, 40, 80); //定义将坐标原点
向右移动40像素，向下移动80像素的矩阵
   /* 绘制图形 */
   context.strokeStyle = "gray";
   context.stroke
}
```



绘制图形

canvas的基础知识

在画布中使用路径

运用样式与颜色

绘制渐变图形

绘制变形图形

组合多个图形

给图形绘制阴影

应用图像

绘制文字

保存与恢复状态

文件的保存

对画布绘制实现动画

综合实例——桌面时钟

目录

组合多个图形

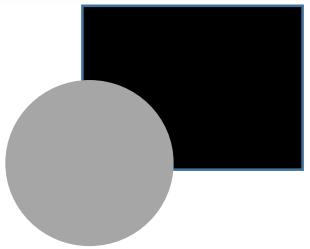
本节需要掌握的知识

1、组合多个图形

组合多个图形

Context.globalCompositeOperation=type

- source-over
- destination
- source-in
- destination-in
- source-out
- destination-out
- source-astop
- destination-astop
- lighter
- darker
- xor
- copy



source-over

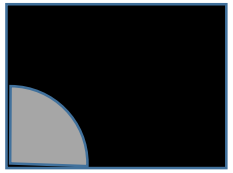
destination

source-in

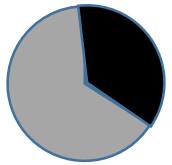
destination-in

source-out

destination-out



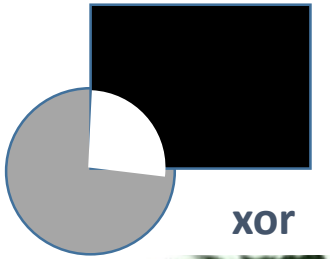
source-astop



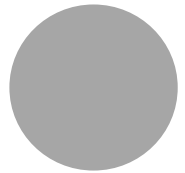
destination-astop

lighter

darker



xor



copy

绘制图形

目录

给图形绘制阴影

本节需要掌握的知识

1、给图形绘制阴影

给图形绘制阴影

在HTML5中，使用canvas元素可以给图形添加阴影效果。添加阴影效果时，只需利用图形上下文对象的几个关于阴影绘制的属性就可以了，如下所示：

- **shadowOffsetX** ——阴影的横向位移量
- **shadowOffsetY**——阴影的纵向位移量
- **shadowBlur** ——阴影的模糊范围
- **shadowColor** ——阴影的颜色

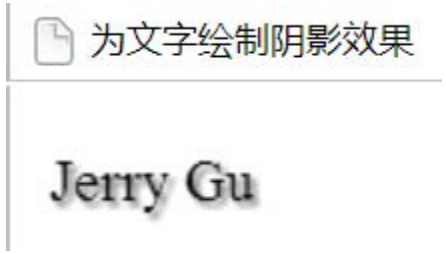
shadowOffsetX 和 **shadowOffsetY** 用来设定阴影在 X 和 Y 轴的延伸距离，它们是不受变换矩阵所影响的。负值表示阴影会往上或左延伸，正值则表示会往下或右延伸，他们默认都是 0。

shadowBlur 用于设定阴影的模糊程度，它表示图形阴影边缘的模糊范围。如果不希望阴影的边缘太清晰，需要将阴影的边缘模糊化时可以使用该属性。设定该属性值时必须要设定为比0大的数字，否则将被忽略。一般设定在0至10之间，开发时可以根据情况调整这个数值，以达到满意效果。

shadowColor 用于设定阴影效果的延伸，值可以是标准的 CSS 颜色值，默认是全透明的黑色。

给图形绘制阴影

```
function draw(id) {                    7-18.html
    var context = document.getElementById('canvas').getContext('2d');
            context.shadowOffsetX = 2;
            context.shadowOffsetY = 2;
            context.shadowBlur = 2;
            context.shadowColor = "rgba(0, 0, 0, 0.5)";

            context.font = "20px Times New Roman";
            context.fillStyle = "Black";
            context.fillText("Jerry Gu", 5, 30);
}
```



绘制图形

目录

应用图像

本节需要掌握的知识

1、绘制图像
2、图像的局部放大
3、图像平铺
4、图像裁剪
5、像素的处理

应用图像

绘制图像

绘制图像时，需要使用drawImage方法，该方法的定义如下所示。

- drawImage(image, x, y)
- drawImage(image, x, y, width, height)
- drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)

应用图像

绘制图像

- 第一种方法只使用三个参数，第一个参数可以是一个**img元素**、**一个video元素**，或者**一个JavaScript中的image对象**，使用该参数代表的实际对象来装载图像文件。x与y为绘制时该图像在画布中的**起始坐标**。

- 第二种方法中前三个参数的使用方法与第一种方法中的使用方法一样，**width、 height是指绘制时的图像的宽度与高度**。使用第一种方法绘制出来的图像与原图大小相同，而使用第二种方法可以用来进行图像缩放。

应用图像

绘制图像

- 第三种方法可以用来将画布中已绘制的图像的全部或者局部区域复制到画布中的另一个位置上。该方法使用九个参数，image仍然代表被复制的图像文件，sx与sy分别表示源图像的被复制区域在画布中的起始横坐标与起始纵坐标，sWidth与sHeight表示被复制区域的宽度与高度，dx与dy表示复制后的目标图像在画布中的起始横坐标与起始纵坐标，dWidth与dHeight表示复制后的目标图像的宽度与高度。该方法可以只复制图像的局部，只要将sx与sy设为局部区域的起始点坐标，将sWidth与 sHeight设为局部区域的宽度与高度就可以了。该方法也可以用来将源图像进行缩放，只要将dWidth与dHeight设为缩放后的宽度与高度就可以了。

应用图像

绘制图像的步骤

首先使用不带参数的new方法创建image对象，然后设定该image对象的src属性为需要绘制的图像文件的路径，具体代码如下所示。

```
image=new Image();
image.src="image.jpg"; //设置图像路径
```

然后就可以使用drawImage方法绘制该图像文件了。

事实上，即使设定好Image对象的src属性后，也不一定立刻就能把图像绘制完毕，譬如有时该图像文件是一个来源于网络的比较大的图像文件，这时用户就要有足够的耐心等待图像全部装载完毕才能看见该图像了。这种情况下，只要使用如下所示的方法，就可以解决这个问题了。

```
image.onload=function(){绘制图像的函数}
```

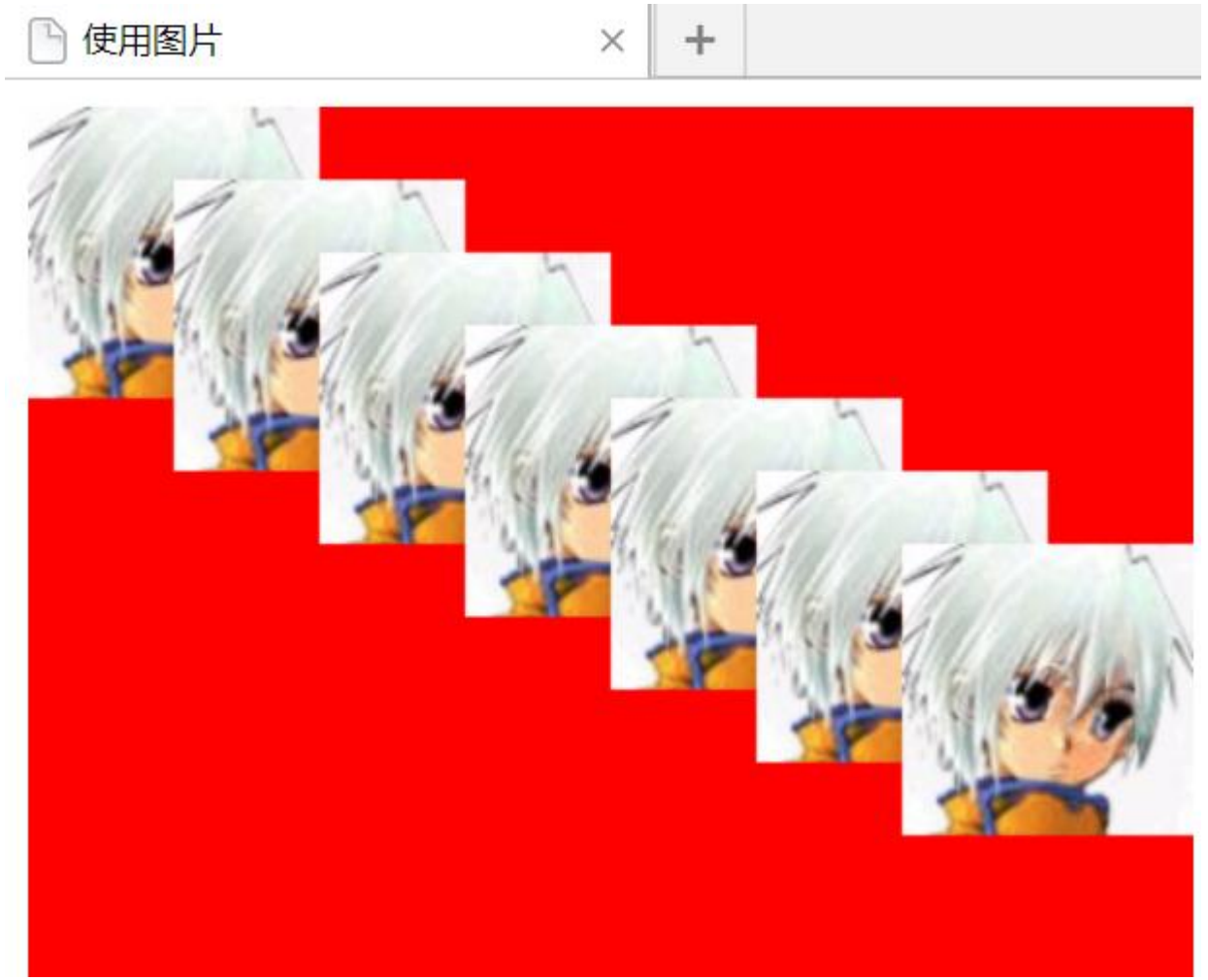
在image对象的onload事件中同步执行绘制图像的函数，就可以一边装载一边绘制了。

应用图像

绘制图像的步骤

7-19.html

```
function draw(id)
{
   var canvas = document.getElementById(id);
   var context = canvas.getContext('2d');
   context.fillStyle = "red";
   context.fillRect(0, 0, 400, 300);
   image = new Image();
   image.src = "imagemr.jpg";
   image.onload = function()
   {
      drawImg(context,image);
   };
}
function drawImg(context,image)
{
   for(var i = 0;i < 7;i++)
      context.drawImage(image,0 + i * 50,0 + i * 25,100,100);
}
```



应用图像

图像的局部放大

图片的局部放大，可以使用如下方法实现。

drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)

例下面通过一个实例来具体讲解该方法是如何实现图像的局部放大的。本例中将卡通人物的头部放大。

应用图像

图像的局部放大

```
function draw(id)                    7-20.html
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "red";
    context.fillRect(0, 0, 400, 300);
    image = new Image();
    image.src = "imagemr.jpg";
    image.onload = function()
    {
        drawImg(context,image);
    };
}
function drawImg(context,image)
{
    var i=0;
    //首先调用该方法绘制原始图像
    context.drawImage(image,0,0,100,100);
    //绘制将局部区域进行放大后的图像
    context.drawImage(image,23,5,57,90,110,0,100,100);
}
```

应用图像

图像平铺

所谓图像平铺就是用按一定比例缩小后的图像将画布填满，有两种方法可以实现该技术，一种是使用前面所介绍drawImage方法，另一种是使用图形上下文对象的createPattern方法，该方法定义如下所示。

```
context.createPattern(image,type);
```

该方法使用两个参数，image参数为要平铺的图像，type参数的值必须是下面的字符串之一：

- **no-repeat**：不平铺
- **repeat-x**：横方向平铺
- **repeat-y**：纵方向平铺
- **repeat**：全方向平铺

应用图像

图像平铺

```
function draw(id)                    7-21.html
{
   var image = new Image();
   var canvas = document.getElementById(id);
   var context = canvas.getContext('2d');
   image.src = "imagemr.jpg";
   image.onload = function()
   {
      drawImg(canvas,context,image);
   };
}
```

```
function drawImg(canvas,context,image)
{
   //平铺比例
   var scale=1
   //缩小后图像宽度
   var n1=image.width/scale;
  //缩小后图像高度
   var n2=image.height/scale;
  //平铺横向个数
   var n3=canvas.width/n1;
  //平铺纵向个数
   var n4=canvas.height/n2;
   for(var i=0;i<n3;i++)
      for(var j=0;j<n4;j++)
       context.drawImage(image,i*n1,j*n2,n1,n2);
}
```

应用图像

图像平铺

7-21.html

```
function draw(id){
    var image = new Image();
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    image.src = "imagemr.jpg";
    image.onload = function()   {
    var ptrn = context.createPattern(image,'repeat'); //创建填充样式，全方向平铺
    context.fillStyle = ptrn; //指定填充样式//填充画布
    context.fillRect(0,0,400,300);
    };
}
```

应用图像

图像裁剪

canvas API的图像裁剪功能是指，在画布内使用路径，只绘制该路径所包括区域内的图像，不绘制路径外部的图像。

使用图形上下文对象的**不带参数的clip方法**来实现canvas 元素的图像裁剪功能。该方法**使用路径来对canvas 画布设置一个裁剪区域**。因此，必须先创建好路径。路径创建完成后，调用clip方法设置裁剪区域。

应用图像

图像裁剪

7-22.html

```
function draw(id)
{
    var canvas = document.getElementById(id);
    var context = canvas.getContext('2d');
    var gr = context.createLinearGradient(0,200,150,0);
    gr.addColorStop(0,'rgb(255,255,0)');
    gr.addColorStop(1,'rgb(0,255,255)');
    context.fillStyle = gr;
    context.fillRect(0, 0, 200, 200);
    image = new Image();
    image.onload = function()
    {
        drawImg(context,image);
    };
    image.src = "imagemr.jpg";
}
function drawImg(context,image)
{
    createyuanClip(context);
    context.drawImage(image,-50,-80,300,300);
}
function createyuanClip(context)
{
     context.beginPath();
       context.arc( 100, 100, 75, 0, Math.PI * 2, true);
       context.closePath();
      context.clip();
}
```

应用图像

像素的处理

使用图像上下文对象的getImageData方法来获取图像中的像素，该方法的定义如下所示。

```
var imagedata = context.getImageData(sx,sy,sw,sh);
```

该方法使用四个参数，sx、sy分别表示所获取区域的起点横坐标、起点纵坐标，sw、sh分别表示所获取区域的宽度和高度。

Imagedata变量是一个CanvasPixelArray对象，具有height，width，data等属性。data属性是一个保存像素数据的数组，内容类似于“[r1,g1,b1,a1, r2,g2,b2,a2, r3,g3,b3,a3,...]”,其中，r1,g1,b1,a1为第一个像素的红色值，绿色值，蓝色值，透明值；r2,g2,b2,a2分别为第二个像素的红色值，绿色值，蓝色值，透明值，依次类推。data.length为所取得像素的数量。

应用图像

图片反显使用的函数

图像上下文对象的**putImageData**方法,该方法的定义如下所示:

```
context.putImageData(imagedata,dx,dy[,dirtyX,dirtyY,dirtyWidth, dirtyHeight]);
```

该方法使用七个参数，**imagedata**为前面所述的像素数组，**dx**、**dy**分别表示重绘图像的起点横坐标，起点纵坐标，后面**dirtyX**、**dirtyY**、**dirtyWidth**、**dirtyHeight**这四个参数为可选参数，给出一个矩形的起点横坐标、起点纵坐标、宽度与高度，如果加上这四个参数，则只绘制像素数组中这个矩形范围内的图像。

应用图像

图片反显使用的函数

```
function draw(id)
                                    7-23.html
{
   var canvas = document.getElementById(id);
   var context = canvas.getContext('2d');      }
   var  image = new Image();
   image.src = "imagemr.jpg";
   image.onload = function()
   {
   context.drawImage(image,0,0);
   var imagedata = context.getImageData(0,0,image.width,image.height);
   for(var i =0, n=imagedata.data.length;i<n;i+=4)//使用i+=4确保确保是像素
   {
    imagedata.data[i+0]=255-imagedata.data[i+0];          //得到反向的颜色红色
    imagedata.data[i+1]=255-imagedata.data[i+2];  //得到反向的颜色绿色
     imagedata.data[i+2]=255-imagedata.data[i+1]; //得到反向的颜色蓝色
   }
   context.putImageData(imagedata,0,0);
   } ;
```

绘制图形

目录

绘制文字

本节需要掌握的知识

1、绘制文字

绘制文字

绘制文字时可以使用**fillText**方法或**strokeTex**t方法。

- **fillText**方法用**填充方式绘制字符串**，该方法的定义如下所示。

```
void fillText(text,x,y,[maxWidth]);
```

该方法接受四个参数，第一个参数**text**表示要绘制的文字，第二个参数**x**表示绘制文字的起点横坐标，第三个参数**y**表示绘制文字的起点纵坐标，第四个参数**maxWidth**为可选参数，表示显示文字时的最大宽度，可以防止文字溢出。

- **strokeText**方法用**轮廓方式绘制字符串**，该方法的定义如下所示。

```
void stroke text(text,x,y,[maxWidth]);
```

该方法参数功能与**fillText**方法相同

绘制文字

7-24.html

```
function draw(id)
{
   var canvas = document.getElementById(id);
   if (canvas == null)
      return false;
   var context=canvas.getContext('2d');
   context.fillStyle= '#00f';
   context.font= 'italic 30px sans-serif';
   context.textBaseline = 'top';
   //填充字符串
   context.fillText  ('顾翔欢迎你', 0, 0);
   context.font='bold  30px sans-serif';
   //轮廓字符串
   context.strokeText('顾翔欢迎你', 0, 50);
}
```

绘制图形

目录

保存与恢复状态

本节需要掌握的知识

1、保存与恢复状态

保存与恢复状态

Save() 和 restore()函数用于保存与恢复状态

用于：

- 图像或图形的变形
- 图像的剪裁
- 改变图形上下文的以下属性： storeStyle, fillStyle, globalAlpha, linewidth, lineCap, linJoin, meterLimit, sandowOffsetX, sandowOffseY, ShandowBlur, ShandowColor, ShandowCompositeOperation

绘制图形

目录

文件的保存

本节需要掌握的知识

1、文件的保存

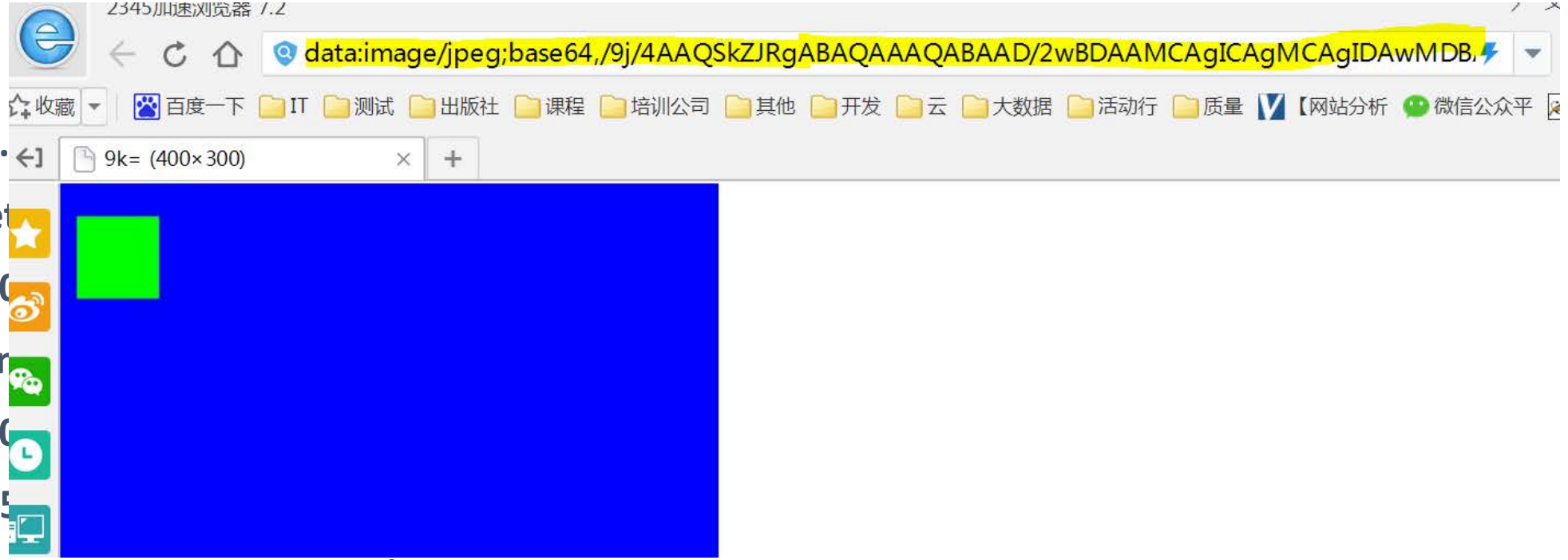
文件的保存

canvas.toDataURL(type);

type为MINE类型

7-25.html

```
function draw(id)
{
    var canvas = document.
    var context = canvas.ge
    context.fillStyle = "rgb(
    context.fillRect(0, 0, ca
    context.fillStyle = "rgb(
    context.fillRect(10, 20, 
    window.location =canvas.toDataURL("image/jpeg");
}
```



绘制图形

目录

对画布绘制实现动画

本节需要掌握的知识

1、对画布绘制实现动画

对画布绘制实现动画

由于是用脚本操控canvas对象的，这样要实现一些交互动画也是相当容易的。只不过，canvas 从来都不是专门为动画而设计的（不像 Flash），这样难免会有些限制。可能最大的限制就是图像一旦绘制出来，它就是一直保持那样了。如果需要移动它，我们不得不对所有东西（包括之前的）进行重绘了。

对画布绘制实现动画的步骤如下：

（1）预先编写好用来绘图的函数，在该函数中先用clearRect方法将画布整体或局部擦除。

（2）使用setInterval方法设置动画的间隔时间。 setInterval方法为HTML中固有的方法，该方法接受两个参数，第一个参数表示执行动画的函数，第二个参数为时间间隔，单位为毫秒

对画布绘制实现动画

```
var context;
var width,height;                    7-26.html
var i;
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    context = canvas.getContext('2d');
  width=canvas.width;
   height=canvas.height;
    i=0;
    setInterval(rotate,100);        //十分之一秒
}
function rotate()
{
    context.clearRect(0,0,width,height);
    context.fillStyle = "blue";
    context.fillRect(i, 0, 20, 20);
    i=i+20;
}
```

绘制图形

目录

综合实例——桌面时钟

本节需要掌握的知识

1、综合实例——桌面时钟

综合实例——桌面时钟

例 本节中将综合本章所讲的canvas API知识，制作一个桌面时钟。在制作桌面时钟时，应用了图形的路径、变形以及动画制作等。

综合实例——桌面时钟

7-27.html

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>桌面时钟</title>
<script >
function time(){
  clock();
  setInterval(clock,1000);//调用clock函数执行动画操作，千分之一秒
}
function clock(){
  var now = new Date();        //实例化对象
  var context = document.getElementById('canvas').getContext('2d');
  context.save();     //保存当前状态
  context.clearRect(0,0,150,150);         //擦除画布
```

综合实例——桌面时钟

```
context.translate(75,75);      //向左，向下平移75个单位
 context.scale(0.4,0.4);       //图形缩放0.4
 context.rotate(-Math.PI/2); //逆时针旋转90度
 context.strokeStyle = "black";          //设置图形边框的样式颜色为黑色
 context.fillStyle = "white";  //填充颜色为白色
 context.lineWidth = 8;        //设置线宽为8
 context.lineCap = "round";  //线段端为默认的圆形

 // Hour marks
 context.save();//保存当前状态
 for (var i=0;i<12;i++){       //通过for循环设置表盘的小时时间隔
  context.beginPath();         //创建设置小时的路径
  context.rotate(Math.PI/6);//顺时针旋转30度
  context.moveTo(100,0);     //将当前位置移动到指定的位置
```

综合实例——桌面时钟

```
context.lineTo(120,0);
  context.stroke(); //绘制时钟小时的时间隔
 }
 context.restore(); //调用restore从栈中取出之前保存的图形

 // Minute marks
 context.save();
 context.lineWidth = 5;
 for (i=0;i<60;i++){ //通过for循环设置表盘的分钟间隔
  if (i%5!=0) {       //通过if语句判断结果，如果相除结果不为0，则继续执行循环
    context.beginPath();       //创建设置分钟的路径
    context.moveTo(117,0);
    context.lineTo(120,0);
    context.stroke();
```

综合实例——桌面时钟

```
}
  context.rotate(Math.PI/30);          //顺时针旋转6度
 }
 context.restore();

 var sec = now.getSeconds();//设置秒钟时间变量
 var min = now.getMinutes();          //设置分钟时间变量
 var hr  = now.getHours();            //设置小时时间变量
 hr = hr>=12 ? hr-12 : hr;

 context.fillStyle = "black";

 // 绘制时针
 context.save();
```

综合实例——桌面时钟

```
context.rotate( hr*(Math.PI/6) + (Math.PI/360)*min + (Math.PI/21600)*sec )
  context.lineWidth = 14;
  context.beginPath();
  context.moveTo(-20,0);
  context.lineTo(80,0);
  context.stroke();
  context.restore();

  // write Minutes
  context.save();
  context.rotate( (Math.PI/30)*min + (Math.PI/1800)*sec )
  context.lineWidth = 10;
  context.beginPath();
  context.moveTo(-28,0);
```

综合实例——桌面时钟

```
context.lineTo(112,0);
  context.stroke();
  context.restore();

  // Write seconds
  context.save();
  context.rotate(sec * Math.PI/30);
  context.strokeStyle = "#D40000";
  context.fillStyle = "#D40000";
  context.lineWidth = 6;
  context.beginPath();
  context.moveTo(-30,0);
  context.lineTo(83,0);
  context.stroke();
```

综合实例——桌面时钟

```
context.restore();
    context.beginPath();
  context.lineWidth = 14;
  context.strokeStyle = '#325FA2';
  context.arc(0,0,142,0,Math.PI*2,true);
  context.stroke();
  context.restore();
}
</script>
</head>
<body onload="time('canvas');">
<canvas id="canvas" width="400" height="300" />
</body>
</html>
```

数据存储

目录

数据存储

- 初识Web Storage
- 本地数据库

目录

初识Web Storage

本节需要掌握的知识

1、Web Storage是什么
2、使用WebStorage中的API
3、sessionStorage和localStorage的实例——计数器
4、Web Storage综合实例——留言本
5、JSON对象的存数实例——用户信息卡

初识Web Storage

Web Storage是什么

Web Storage功能，顾名思义，就是在Web上存储数据的功能，而这里的存储，是针对客户端本地而言的。它包含两种不同的存储类型：Session Storage和Local Storage。 不管是Session Storage还是Local Storage，它们都能支持在同域下存储5MB数据.

- **sessionStorage**

将数据保存在session对象中。所谓session，是指用户在浏览某个网站时，从进入网站到浏览器关闭所经过的这段时间，也就是用户浏览这个网站所花费的时间。Session对象可以用来保存在这段时间内所要求保存在任何数据。

- **localStorage**

将数据保存在客户端本地的硬件设备中，即使浏览器被关闭了，该数据仍然存在，下次打开浏览器访问网站时仍然可以继续使用. 这两种不同的存储类型区别在于，sessionStorage为临时保存，而localStorage为永久保存

初识Web Storage

使用WebStorage中的API

下面我们讲解如何使用WebStorage的API。目前WebStorage的API有如下这些：

- **Length**：获得当前webstorage中的数目。
- **key(n)**：返回webstorage中的第N个存储条目。
- **getItem(key)**：返回指定key的存储内容，如果不存在则返回null。注意，返回的类型是String字符串类型。
- **setItem(key, value)**：设置指定key的内容的值为value。
- **removeItem(key)**：根据指定的key，删除键值为key的内容。
- **clear**：清空webstorate的内容。

初识Web Storage

数据的存储与获取

在localStorage中设置键值对数据可以应用setItem()，代码如下所示：

```
localStorage.setItem("key", "value);
```

获取数据可以应用getItem()，代码如下所示：

```
var val = localStorage.getItem("key");
```

当然也可以直接使用localStorage的key方法，而不使用setItem和getItem方法，如下：

```
localStorage.key = "value";
var val = localStorage.key;
```

HTML5存储是基于键值对（key/value）的形式存储的，每个键值对称为一个项（item）存储和检索数据都是通过指定的键名，键名的类型是字符串类型。值可以是包括字符串、布尔值、整数，或者浮点数在内的任意JavaScript支持的类型。但是，最终数据是以字符串类型存储的。

调用结果是将字符串value设置到sessionStorage中，这些数据随后可以通过键key获取。调用setItem()时，如果指定的键名已经存在，那么新传入的数据会覆盖原先的数据。调用getItem()时，如果传入的键名不存在，那么会返回null，而不会抛出异常。

初识Web Storage

数据的删除和清空

removeItem()用于从Storage列表删除数据代码如下：

```
var val = sessionStorage.removeItem(key);
```

也可以通过传入数据项的key从而删除对应的存储数据代码如下：

```
var val = sessionStorage.removeItem(1);
```

说明：数字1会被转换为string，因为key的类型就是字符串。

clear()方法用于清空整个列表的所有数据，代码如下：

```
sessionStorage.clear();
```

同时可以通过使用length属性获取Storage中存储的键值对的个数：

```
var val = sessionStorage.length;
```

注意：removeItem可以清除给定的key所对应的项，如果key不存在则“什么都不做”；clear会清除所有的项，如果列表本来就是空的就“什么都不做”。

初识Web Storage

sessionStorage和localStorage的实例——计数器

例 本例主要是通过sessionStorage和localStorage对页面的访问量进行计数。当在文本框内输入数据后，分别可以单击“session保存”按钮和“local保存”按钮对数据进行保存，还可以通过session读取”按钮和“local读取”按钮对数据进行读取。

但是两种方法对数据的处理方式不一样，使用sessionStorage方法时，如果关闭了浏览器，这个数据就丢失了下一次打开浏览器，点击读取数据按钮时，读取不到任何数据。使用localStorage方法时，即使浏览器关闭了，下次打开浏览器时仍然能够读取保存的数据。但是，数据保存是按不同的浏览器分别进行的，也就是说，如果打开别的浏览器，是读取不到在这个浏览器中保存的数据的。

初识Web Storage

sessionStorage和localStorage的实例——计数器

8-1.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>sessionStorage与localStorage区别</title>
</head>
<body>
<h1>计数器</h1>
<p class="msg" id="msg_1"> </p>
<p class="form_item"><label for="">要保存的数据：</label>
<input type=“text” name=“text-1” value=“” id=“text-1”/></p>  <!-- test1:输入数据-->
```

初识Web Storage

sessionStorage和localStorage的实例——计数器

```
<p class="form_item">
    <input type=“button” name=“btn-1” value=“session保存” id=“btn-1”/>  <!-- btn-1 :session保存-->
    <input type=“button” name=“btn-2” value=“session读取” id=“btn-2”/>  <!-- btn-2 :session读取-->
</p>
<p class="form_item">
    <input type="button" name="btn-3" value="local保存" id="btn-3"/> <!-- btn-3 :local保存-->
    <input type="button" name="btn-4" value="local读取" id="btn-4"/> <!-- btn-4:local保存-->
</p>
<p class="count_wrap">
session计数：<span class="count" id=''></span>  
local计数：<span class="count" id='local_count'><session_count/span></p>
<script>
```

初识Web Storage

sessionStorage和localStorage的实例——计数器

```
function getE(ele){   //自定义一个getE()函数
        return document.getElementById(ele);   //返回并调用document对象的getElementById方法输出变量
    }
    var text_1 = getE('text-1'),   //声明变量并为其赋值
        mag = getE('msg_1'),
        btn_1 = getE('btn-1'),
        btn_2 = getE('btn-2'),
        btn_3 = getE('btn-3'),
        btn_4 = getE('btn-4');
    btn_1.onclick = saveSessionStorage;
    btn_2.onclick = loadSessionStorage;
    btn_3.onclick = saveLocalStorage;
    btn_4.onclick = loadLocalStorage;
```

初识Web Storage

sessionStorage和localStorage的实例——计数器

```
function saveSessionStorage(){
    sessionStorage.setItem('msg',text_1.value + 'session');
}
function loadSessionStorage(){
    mag.innerHTML = sessionStorage.getItem('msg');
}
function saveLocalStorage(){
    localStorage.setItem('msg',text_1.value + 'local');
}
function loadLocalStorage(){
    mag.innerHTML = localStorage.getItem('msg');
}
```

初识Web Storage

sessionStorage和localStorage的实例——计数器

```
//记录页面次数
        var local_count = localStorage.getItem('a_count')?localStorage.getItem('a_
        getE('local_count').innerHTML = local_count;
        localStorage.setItem('a_count',+local_count+1);


        var session_count = sessionStorage.getItem('a_count')?sessionStorage.getI
        getE('session_count').innerHTML = session_count;
        sessionStorage.setItem('a_count',+session_count+1);


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

初识Web Storage

Web Storage综合实例——留言本

例 在本节中，来看一个简单Web留言本的示例。使用一个多行文本框输入数据，点击按钮时将文本框中的数据保存到localStorage中，在表单下部放置一个P元素来显示保存后的数据。如果只保存文本框中内容，并不能知道该内容是什么时候写好的，所以保存该内容的同时，也保存了当前日期和时间，并将该日期和时间一并显示在P元素中。利用Web Storage保存数据时，数据必须是“键名/键值”这样的格式，所以将文本框的内容作为键值，保存时的日期和时间作为键名来进行保存，计算机中对于日期和时间的值是以时间戳的形式进行管理的，所以保存时不可能存在重复的键名。

初识Web Storage

Web Storage综合实例——留言

8-2.html

```
function saveStorage(id)
{
   var data = document.getElementById(id).value;
   var time = new Date().getTime();
   localStorage.setItem(time,data);
   alert("数据已保存。");
   loadStorage('msg');
}
```


```
    {
      var key = localStorage.key(i);
      var value = localStorage.getItem(key);
      var date = new Date();
      date.setTime(key);
      var datestr = date.toGMTString();
      result += '<tr><td>' + value + '</td><td>' + datestr +
'</td></tr>';
    }
    result += '</table>';
    var target = document.getElementById(id);
    target.innerHTML = result;
}
```

初识Web Storage

Web Storage综合实例——留言本

```
function clearStorage()
{
   localStorage.clear();
   alert("全部数据被清除。");
   loadStorage('msg');
}
</script>
</head>
<body>
<h1>简单Web留言本</h1>
<textarea id="memo" cols="60" rows="10"></textarea><br>
<input type="button" value="添加" onclick="saveStorage('memo');">
<input type="button" value="全部清除" onclick="clearStorage('msg');">

<hr>
<p id="msg"></p>
</body>
</html>
```

初识Web Storage

JSON对象的存数实例——用户信息卡

JSON是一种将对象与字符串可以相互表示的数据转换标准。JSON一直是通过HTTP将对象从浏览器传送到服务器一种常用格式。现在，可以通过序列化复杂对象将JSON数据保存在Storage中，以实现复杂数据类型的持久化。

例 下面是一个用户信息卡的实例，在该实例中将用户的信息使用JSON格式进行保存。使用JSON的格式作为文本保存来保存对象，获取该对象时再通过JSON格式来获取，可以保存和读取具有复杂结构的数据了。

初识Web Storage

JSON对象的存数实例——用户信息卡

8-3.html

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>用户信息卡</title>
<script type="text/javascript" >
function saveStorage()
{
    var data = new Object;
    data.name = document.getElementById('name').value;
    data.email = document.getElementById('email').value;
    data.tel = document.getElementById('tel').value;
    data.memo = document.getElementById('memo').value;
    var str = JSON.stringify(data);
```

初识Web Storage

JSON对象的存数实例——用户信息卡

```
localStorage.setItem(data.name,str);
    alert("数据已保存。");
}
function findStorage(id)
{
    var find = document.getElementById('find').value;
    var str = localStorage.getItem(find);
    var data =  JSON.parse(str);
    var result = "姓名: " + data.name + '<br>';
    result += "EMAIL: " + data.email + '<br>';
    result += " 电话号码: " + data.tel  + '<br>';
    result += "备注: " + data.memo + '<br>';
    var target = document.getElementById(id);
```

初识Web Storage

JSON对象的存数实例——用户信息卡

```
target.innerHTML = result;
}
</script>
</head>
<body>
<h1>用户信息卡</h1>
<table>
   <tr><td align="right">姓名:</td><td><input type="text" id="name"></td></tr>
   <tr><td align="right">EMAIL:</td><td><input type="text" id="email"></td></tr>
   <tr><td align="right">电话号码:</td><td><input type="text" id="tel"></td></tr>
   <tr><td align="right">说明:</td><td><input type="text" id="memo"></td></tr>
   <tr>
      <td colspan="2" align="center"><input type="button" value="保存" onclick="saveStorage();"></td>
```

初识Web Storage

JSON对象的存数实例——用户信息卡

```
</tr>
</table>
<hr>
<p>查询:
<input type="text" id="find">
<input type="button" value="按姓名查询" onclick="findStorage('msg');">
</p>
<p id="msg"></p>
</body>
```

数据存储

- 初识Web Storage
- 本地数据库

目录

本地数据库

本节需要掌握的知识

1、Web SQL数据库简介
2、使用Web SQL Database API
3、本地数据库实例——用户登录

本地数据库

Web SQL数据库简介

Web SQL数据库是存储和访问数据的另一种方式。从其名称可以看出，这是一个真正的数据库，可以查询和加入结果。在HTML5中，大大丰富了客户端本地可以存储的内容，添加了很多功能来将原本必须要保存在服务器上的数据转为保存在客户端本地，从而大大提高了Web应用程序的性能，减轻了服务器端的负担.

在这其中，一项非常重要的功能就是数据库的本地存储功能。在HTML5中内置了一个可以通过SQL语言来访问数据库。在HTML4中，数据库只能放在服务器端，只能通过服务器来访问数据库，但是在HTML5中，可以就像访问本地文件那样轻松地对内置数据库进行直接访问了。

本地数据库

使用Web SQL Database API

- **1、打开和创建数据库**

打开和创建数据库必须使用openDatabase方法来创建一个访问数据库的对象。该方法的使用方法如下所示。

```
var db=openDatabase( 'db', '1.0' , 'first database',2*1024*1024);
```

该方法使用四个参数，第一个参数为数据库名，第二个参数为版本号，第三个参数为数据库的描述，第四个参数为数据库的大小。该方法返回创建后的数据库访问对象，如果该数据库不存在，则创建该数据库.

测试代码:

```
var db;
if(window.openDatabase){
    db = openDatabase('mydb', '1.0' , 'My first database',2*1024*1024);
}
```

本地数据库

使用Web SQL Database API

- 2、创建数据表

实际访问数据库的时候，还需要使用transaction方法，用来执行事务处理。transaction方法的使用方法如下所示。

```
db.transaction(function(tx)){
   tx.executeSql('CREATE TABLE tweets(id,date,tweet)');
});
```

transaction方法使用一个回调函数为参数。在这个函数中，执行访问数据库的语句。

本地数据库

使用Web SQL Database API

要创建数据表（以及数据库上的任何其他事务），必须启动一个数据库“事务”，并且在回调中创建该表。事务回调接受一个参数，其中包含了事务对象，这就是允许运行SQL语句并且运executeSql方法（在下面的例子中，就是tx）的内容。这通过使用从openDatabase返回的数据库对象来完成，并且像下面这种调用事物的方法如下所示。

```
var db;
if(window.openDatabase){
  db = openDatabase('mydb', '1.0' , 'My first database',2*1024*1024);
  db.transaction(function(tx)){
  tx.executeSql('CREATE TABLE tweets(id,date,tweet)');
  });
}
```

本地数据库

使用Web SQL Database API

- 3、插入和查询数据

executeSql方法的完整定义如下所示。

```
transaction.executeSql(sqlquery,[],dataHandler,errorHandler);
```

第一个参数为需要执行的SQL语句。

第二个参数为SQL语句中所有使用到的参数的数组。在executeSql方法中，将SQL语句中所要使用到的参数先用“？”代替，然后依次将这些参数组成数组放在第二个参数中，如下所示。

```
transaction.executeSql("UPDATE user set age=? where name=?;",[age,name]);
```

本地数据库

使用Web SQL Database API

第三个参数为执行sql语句成功时调用的回调函数。该回调函数的传递方法如下所示。

```
function dataHandler(transaction,results){//执行SQL语句成功时的处理}
```

该回调函数使用两个参数，第一个参数为transaction对象，第二个参数为执行查询操作时返回的查询到的结果数据集对象。

第四个参数为执行SQL语句出错时调用的回调函数。该回调函数的传递方法如下所示。

```
function errorHandler(transaction,errmsg){//执行SQL语句出错时的处理
};
```

该回调函数使用两个参数，第一个参数为transaction对象，第二个参数为执行发生错误时的错误信息文字。

本地数据库

本地数据库实例——用户登录(仅供学习，案例不好)

例 在本节中，我们用户登录界面作为实例，来看一下具体如何对本地数据库进行简单操作的。在页面中输入用户名和密码单击“登录”按钮，登录成功后，用户名、密码以及登录时间将显示在页面上，单击“注销”按钮，将清除已经登录的用户名、密码以及登录时间。

本地数据库

本地数据库实例——用户登录

8-4.html

```
<form action="#" method="get" accept-charset="utf-8">
<p class="form_item">
用户名： <input type="text" name="" value="" id="name" required/>
</p>
<p class="form_item">
密&amp;nbsp;&amp;nbsp;码： <input type="password" name="" value="" id="msg" required></textarea>
</p>
<p class="form_item">
<input type="submit" id="save" value="登录"/>
<input type="submit" id="clear" value="注销"/>
</p>
<hr>
</form>
```

本地数据库

本地数据库实例——用户登录

```
<script type="text/javascript" charset="utf-8">
    (function(){
        var datalist = getE('datalist');
        if(!datalist){
            datalist = document.createElement('dl');
            datalist.className = 'datalist';
            datalist.id = 'datalist';
            document.body.appendChild(datalist);
        }
        var result = getE('result');
        var db = openDatabase('myData','1.0','test database',1024*1024);
        showAllData();
```

本地数据库

本地数据库实例——用户登录

```
db.transaction(function(tx){
    tx.executeSql('CREATE TABLE IF NOT EXISTS MsgData(name TEXT,msg TEXT,time INTEGER)',[]);


})
getE('clear').onclick = function(){
    db.transaction(function(tx){
        tx.executeSql('DROP TABLE MsgData',[]);
    })
    showAllData()
}
getE('save').onclick = function(){
    saveData();
    return false;
```

本地数据库

本地数据库实例——用户登录

```
}
    function getE(ele){
        return document.getElementById(ele);
    }
  function showData(row){
        var dt = document.createElement('dt');
                    dt.innerHTML = row.name;
                    var dd = document.createElement('dd');
        dd.innerHTML = row.msg;
                    var tt = document.createElement('tt');
                    var t = new Date();
                    t.setTime(row.time);
                    tt.innerHTML =t.toLocaleDateString()+" "+ t.toLocaleTimeString();
```

本地数据库

本地数据库实例——用户登录

```
datalist.appendChild(dt);
        datalist.appendChild(dd);
                datalist.appendChild(tt);
    }
    function showAllData()
        {

        db.transaction(function(tx)
                {
            tx.executeSql('CREATE TABLE IF NOT EXISTS MsgData(name TEXT,msg TEXT,time INTEGER)',[]);
            tx.executeSql('SELECT * FROM MsgData',[],function(tx,result){
                removeAllData();
                for(var i=0; i < result.rows.length; i++){
```

本地数据库

本地数据库实例——用户登录

```
showData(result.rows.item(i));
            }
        });
    });
}
function addData(name,msg,time)
        {
    db.transaction(function(tx)
                {
        tx.executeSql('INSERT INTO MsgData VALUES(?,?,?)',[name,msg,time],function(tx,result)
                        {
                        alert("登录成功");
    },
```

本地数据库

本地数据库实例——用户登录

```
                                function(tx,error){
            alert(error.source + ':' + error.message);
        });             });         }
    function saveData()       {
        var name =getE('name').value;
        var msg = getE('msg').value;
        var time = new Date().getTime();
        addData(name,msg,time);
        showAllData();
    }
})();
 </script>
```

HTML5开发教程

- HTML 5的新特性
- HTML 5与HTML 4的区别
- HTML 5的结构
- HTML 5中的表单
- HTML 5中的文件与拖放
- 多媒体播放
- 绘制图形
- 数据存储
- 获取地理位置信息
- 练习

目录

离线应用程序

- Geolocation API的概述
- position对象
- 案例

目录

Geolocation API的概述

本节需要掌握的知识

1、使用getCurrentPosition获取当前地理位置
2、持续监视当前地理位置的信息
2、停止获取当前用户的地理位置信息

Geolocation API的概述

使用getCurrentPosition获取当前地理位置

使用getCurrentPosition方法来取得用户当前的地理位置信息，该方法的定义如下所示。

```
void getCurrentPosition(onSuccess,onError,options);
```

- 第一个参数为获取**当前地理位置信息成功**时所执行的回调函数；
- 第二个参数为获取**当前地理位置信息失败时**所执行的回调函数；
- 第三个参数为**一些可选属性**的列表。

注意：其中第二、三个参数为可选属性。

Geolocation API的概述

getCurrentPosition方法中的第一个参数

getCurrentPosition方法中的第一个参数为获取当前地理位置信息成功时所执行的回调函数。该参数的使用方法如下所示。

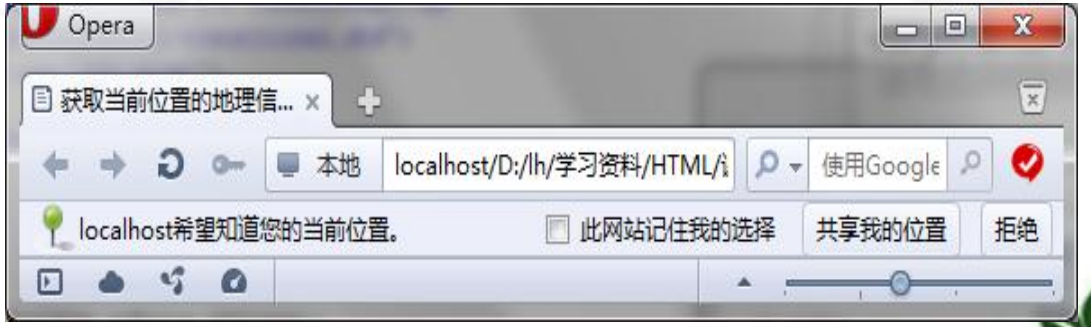
```
navigator.geolocation.getCurrentPosition(function(position)){
//获取成功时的处理}
```

在获取地理位置信息成功时执行的回调函数中，用到了一个参数position，它代表的是一个position对象

Geolocation API的概述

getCurrentPosition方法中的第二个参数

getCurrentPosition方法中的第二个参数为获取当前地理位置信息失败时所执行的回调函数。如果获取地理位置信息失败，你可以通过该回调函数把错误信息提示给用户。当在浏览器中打开使用了Geolocation API来获得用户当前位置信息的页面时，浏览器会询问用户是否共享位置信息。



Geolocation API的概述

getCurrentPosition方法中的第二个参数

如果在该画面中拒绝共享的话，也会引起错误的发生。该回调函数使用一个error对象作为参数，该对象具有以下两个

属性：

- **Code属性**

Code属性为以下三个值其中之一：

- ✓用户拒绝了位置服务（属性值为1）
- ✓获取不到位置信息（属性值为2）
- ✓获取信息超时错误（属性值为3）

- **Message属性**

message属性为一个字符串，在该字符串中包含了错误信息，这个错误信息在开发和调试时将很有用。但是需要注意的是有些浏览器是不支持message属性的，譬如Firefox3.6以上。

Geolocation API的概述

getCurrentPosition方法中的第二个参数

在getCurrentPosition方法中使用第二个参数来捕获错误信息的具体使用方法如下所示。

```
navigator.geolocation.getCurrentPosition(
  function(position)){
        var coords = position.coords;
        showMap(coords.latitude,coords.longitude,coords.accuracy);
  },
  //捕获错误信息
  function(error){
  var errorTypes = {
      1:'位置服务被拒绝',
      2:'获取不到位置信息',
      3:'获取信息超时'
  };
  alert(errorTypes[error.code]+":,不能确定你的当前地理位置");
  }
```

Geolocation API的概述

getCurrentPosition方法中的第三个参数

getCurrentPosition方法中的第三个参数可以省略，它是一些可选属性的列表，这些可选属性如下。

✓**enableHighAccuracy（布尔型，默认为false）**

是否要求高精度的地理位置信息，这个参数在很多设备上设置了都没用，因为使用在设备上时需要结合设备电量、具体地理情况来结合考虑。因此，多数情况下把该属性设为默认，由设备自身来调整。

✓**timeout（单位为毫秒，默认值为infinity/0）**

对地理位置信息的获取操作做一个超时限制（单位为毫秒）。如果在该时间内未获取到地理位置信息，则返回错误。

Geolocation API的概述

getCurrentPosition方法中的第三个参数

✓**maximumAge（单位为毫秒，默认值为0）**

对地理位置信息进行缓存的有效时间（单位为毫秒）。例如maximumAge：120000（1分钟是60000）。如果11点整得时候获取过一次地理位置信息，11:01的时候，再次调用navigator.geolocation.getCurrentPosition重新获取地理位置信息，则返回的依然为11:00时的数据（因为设置的缓存有效时间为2分钟）。超过这个时间后缓存的地理位置信息被废弃,尝试重新获取地理位置信息。如果该值被指定为0，则无条件重新获取新的地理位置信息。

Geolocation API的概述

getCurrentPosition方法中的第三个参数

```
navigator.geolocation.getCurrentPosition(
   function(position)){
          //获取地理位置信息成功时所做处理
   },
   function(error){
   //获取地理位置信息失败时所做处理
   },
   //以下为可选属性
   {
     //设置缓存有效时间为2分钟
     maximumAge:60*1000*2,
     //5秒钟内获取到地理位置信息则返回错误
     timeout:5000
    }
};
```

Geolocation API的概述

持续监视当前地理位置的信息

使用watchPosition方法来持续获取用户的当前地理位置信息，它会定期地自动获取，该方法定义如下所示。

```
int watchCurrentPosition(onSuccess,onError,options);
```

该方法三个参数的说明与使用方法与getCurrentPosition方法的参数说明与使用方法相同。该方法返回一个数字，这个数字的使用方法与Javascript脚本中setInterval方法的返回参数的使用方法类似，可以被clearWatch方法使用，停止对当前地理位置信息的监视。

Geolocation API的概述

停止获取当前用户的地理位置信息

使用该方法可以停止对当前用户的地理位置信息的监视。该方法定义如下所示。

```
void clearWatch(watchId);
```

该方法的参数为调用watchCurrentPosition方法监视地理位置信息时的返回参数。

离线应用程序

- Geolocation API的概述
- position对象
- 案例

目录

position对象

本节需要掌握的知识

1、 position对象

position对象

如果获取地理位置信息成功，则可以在获取成功后的回调函数中通过访问position对象的属性来得到这些地理位置信息。Position对象具有如下这些属性

- **latitude**

当前地理位置的纬度。

- **longitude**

当前地理位置的经度

- **altitude**

当前地理位置的海拔高度（不能获取时为null）

- **accuracy**

获取到的纬度或经度的精度（以米为单位）

- **altitudeAccurancy**

获取到的海拔高度的精度（以米为单位）

position对象

✓ **heading**

设备的前进方向。用面朝正北方向的顺时针旋转角度来表示（不能获取时为null）

✓ **speed**

设备的前进速度（以米/秒为单位，不能获取时为null）。

✓ **timestamp**

获取地理位置信息时的时间。

例 在本例中使用getCurrentPosition方法获取当前位置的地理信息，并且在页面中显示Position对象中当前地理位置的纬度和经度.

position对象

✓ **heading**

设备的前进方向。用面朝正北方向的顺时针旋转角度来表示（不能获取时为null）

✓ **speed**

设备的前进速度（以米/秒为单位，不能获取时为null）。

✓ **timestamp**

获取地理位置信息时的时间。

例 在本例中使用getCurrentPosition方法获取当前位置的地理信息，并且在页面中显示Position对象中当前地理位置的纬度和经度.

离线应用程序

- Geolocation API的概述
- position对象
- 案例

目录

案例

9-1.html

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">点击按钮获取您当前坐标（可能需要比较长的时间获取）：</p>
<button onclick="getLocation()">点我</button>
<script>
var x=document.getElementById("demo");
function getLocation()
 {
 if (navigator.geolocation)
   {
   navigator.geolocation.getCurrentPosition(showPosition,showError);
   }
 else{x.innerHTML="该浏览器不支持获取地理位置。";}
 }
```

案例

```
function showPosition(position)
 {
 x.innerHTML="纬度: " + position.coords.latitude +
 "<br>经度: " + position.coords.longitude;
 }
 function showError(error)
 {
 switch(error.code)
  {
  case error.PERMISSION_DENIED:
    x.innerHTML="用户拒绝对获取地理位置的请求。"
    break;
  case error.POSITION_UNAVAILABLE:
    x.innerHTML="位置信息是不可用的。"
    break;
  case error.TIMEOUT:
    x.innerHTML="请求用户地理位置超时。"
```

案例

```
break;
    case error.UNKNOWN_ERROR:
      x.innerHTML="未知错误。"
      break;
    }
  }

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

HTML5开发教程

目录

练习

上海旅游信息网

图片

用户名

密码

登录　　注册

用户信息注册

用户名:

密码:

确认密码:

Email:

手机:

注册

全为必填项

练习

上海欢迎您

图片

文字介绍

文字介绍

首页 景点 酒店 我

景点介绍

图片 外滩

图片 陆家嘴

图片 七宝

图片 顾村公园

图片 新世界

可往下翻滚

首页 景点 酒店 我

练习

百年外滩

图片

文字介绍

文字介绍

首页 景点 酒店 我

酒店介绍

地区 ▼ 价格 ▼ 类型 ▼ 查询

图片	锦江饭店 ￥1500-￥3500
图片	锦江饭店 ￥1500-￥3500
图片	锦江饭店 ￥1500-￥3500
图片	锦江饭店 ￥1500-￥3500
图片	锦江饭店 ￥1500-￥3500

可往下翻滚

首页 景点 酒店 我

练习

锦江饭店

图片

文字介绍

标准间：2500元
单人间：3000元
大床房：2500元
豪华间：3500元

预定

预定不做

首页 景点 酒店 我

用户账号：Jerry

修改密码

修改用户信息

修改用户信息页面同注册页面，用户名，密码不得修改

退出登录

首页 景点 酒店 我