第16章 DOM和BOM(2-4课时)

使用外部JS


```
25 <script type="text/javascript" src="http://static.womai.com/zhongliang/shan2014/js/base.js?t=20150415"></script>
```

本章学习目标

主要内容：

- 学会使用JavaScript内置对象的常用属性及方法来解决具体的问题；
- 理解文档对象模型的节点树的构建及节点类型的划分；
- 学会使用document对象常用的方法来设计动态的网页效果
- 理解浏览器对象模型的各对象的层次关系；
- 学会使用window对象的定时器及对话框方法；
- 了解navigator、screen、history、location等对象的属性和方法。

16.1 常用对象

JavaScript的对象类型（分为4类）：本地对象、内建对象、宿主对象、自定义对象。

(1)本地对象(native object)，ECMA-262定义为“独立于宿主环境的ECMAScript实现提供的对象”。简单来说，本地对象就是ECMA-262定义的类（引用类型）。

它们包括：Object、Function、Array、String、Boolean、Number、Date、RegExp、Error、EvalError、RangeError、ReferenceError、SyntaxErro、TypeError、URIError等。这些对象独立于宿主环境，先定义对象，实例化后再通过对象名来使用。

16.1 常用对象-续

(2)内置对象(built-in object)。由ECMAScript实现提供的、不依赖与宿主环境的对象，在ECMAScript运行之前就已经创建好的对象就叫做内置对象。

这意味着开发者不必明确实例化内置对象，它已被实例化了。ECMA-262只定义了两个内置对象，即Global和Math。Global是全局对象，全局对象只是一个对象，而不是类。既没有构造函数，也无法实例化一个新的全局对象。

例如isNam(),isFinite(),parseInt()和parseFloat()等，都是Global对象的方法。Math对象直接使用，如Math.Random()、Math.round(20.5)等。

16.1 常用对象-续

(3)宿主对象(host object)。ECMAScript实现的宿主环境提供的对象。所有BOM和DOM对象都是宿主对象。通过它可以与文档和浏览器环境进行交互，如document、window和frames等。

(4)自定义对象。根据程序设计需要，由编程人员自行定义的对象。例如定义一个person对象，它有4个属性分别是firstName、lastName、age、eyeColor，同时给属性赋值。定义格式如下：

```
var person=new Object();  /* 这是一种方法*/
    person.firstname="Bill";
    person.lastname="Gates";
    person.age=56;
    person.eyecolor="blue";
var person={firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};/*另一方法*/
```

16.1.1 Array

Array对象用于在单个的变量中存储多个相同类型的值，其值可以是字符串、数值型、布尔型等，但由于JavaScript是弱类型的脚本语言主，所以数组元素也可以不一致。通过声明一个数组，将相关的数据存入数组，使用循环等结构对数组中的每个元素进行操作。

1.创建Array对象

1)基本语法

```
var stu1=new array();
var stu2=new Array(size);
var stu3=new Array(element0, element1, ..., elementn);
var str4=[“C++程序设计”,“HTML开发基础”,“数据库原理””];
```

2)参数说明

- **参数size定义数组元素的个数。返回的数组的长度stu2.length等于size。**

16.1.1 Array

- 参数element0、…、elementn是参数列表。当使用这些参数来调用构造函数Array()时，新创建的数组的元素就会被初始化为这些值。

2.数组的返回值

数组变量stu1、stu2、stu3返回新创建并被初始化了的数组。如果调用构造函数Array()时没有使用参数，那么返回的数组为空，数组的length为0。

3.数组元素初始化与修改指定数组元素

```
var stu = new Array();          /*先定义数组*/
stu[0] = "计算机组成原理";       /*给数组元素赋值*/
 stu[1] = "Java程序设计";        /*给数组元素赋值*/
 var len=stu.length;           /*len的值为2*/
 stu[1] = "创新创业教育";       /* 修改数组中第2个元素 */
```

16.1.1 Array

4.数组对象的属性和方法

- 数组的下标从0开始，到数组的长度-1为止。
- 如访问第一个元素的代码可以这样写：var cn = stu[0];
- 数组下标可以用变量替代，但变更的取值范围必须符合数组下标的边界，否则返回*undefined*。例如：var i = 3;var cn = stu[i];
- 可以用再赋值的方式来修改数组对应位置的元素，如： var stu1[2] = “王建伟”;

16.1.1 Array

使用数组对象的属性和方法

length：数组对象长度；

join(separator)：把数组各个项用某个字符(串)连接起来，但并不修改原来的数组，默认用逗号分隔。

例如：

```
var cn=stu1.join('-'); //短横线作为分隔符
```

Cn的值为“张有为-蒋丽娟-王一新-李大为”。

pop()：删除并返回数组的最后一个元素。

例如：`var cn=stu.pop();`

则变量cn获得的值是“李大为”。

16.1.1 Array

push(newelement1,.,newelementX):可向数组的末尾添加一个或多个元素，并返回新的长度。例如：

var length=stu.push(“张丽丽”，“付天舒");

则变量length获得的值6。

shift()：用于把数组的第一个元素从其中删除，并返回第一个元素的值。

var ss=stu.shift();

unshift(newelement1,newelement2,....,newelementX)： 向数组的开头添加一个或更多元素，并返回新的长度。

var s= stu.unshift(“储国王”)；s=5

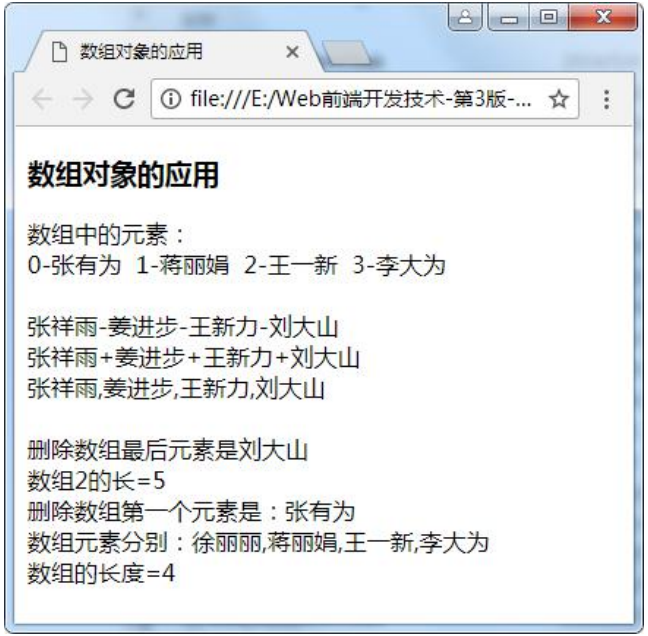
其它方法:sort()、reverse()、toString()等。

16.1.1 Array-案例

```
<!--  edu_16_1_1.html -->
<html>
   <head>
      <title>数组对象举例</title>
   </head>
  <body>
      <h3>对象的类型应用案例</h3>
  <script type="text/javascript">
      var stu1 = new Array("张有为","蒋丽娟","王一新","李大为");
      var stu2 =["张祥雨","姜进步","王新力","刘大山"];
      document.write("数组中的元素：<br>");
      //访问数组中的元素
      for (var i=0;i<=stu1.length-1;i++ )
      {
        document.write(i+"-"+stu1[i]+"  ");
      }
      document.write("<br><br>");
      //join方法的使用
      document.write(stu2.join("-")+"<br>");//"-"分隔
      document.write(stu2.join("+")+"<br>");//"+"分隔
      document.write(stu2.join()+"<br>");  //默认
```

16.1.1 Array-案例

```
    //pop,push方法的使用
    document.write("<br>删除数组最后元素是
"+stu2.pop());
    var s=stu2.push("沈通达","高学衡");
    document.write("<br>数组2的长="+s);
    var stu1 = new Array ("张有为","蒋丽娟","王一新","李
大为");
    //shift,unshift方法的使用
    var ss=stu1.shift();
    document.write("<br>删除数组第一个元素是："+ss);
    //在数组开始处插入新元素
    var s=stu1.unshift("徐丽丽");//在IE中显示undefined
    document.write("<br>数组元素分别："+stu1+"<br>
数组的长度="+s); //在IE中用stu1.length代替
</script>
 </body>
</html>
```



16.1.2 Date

JavaScript脚本内置对象Date用于处理日期和时间。Date对象有很多方法，可以提取时间和日期。

1.创建日期对象

```
var today=new Date();
var today=new Date(毫秒数);
var today=new Date(标准时间格式字符串);
var today=new Date(年,月,日,时,分,秒,毫秒);
```

■**应用举例**

```
var today=new Date();          //自动使用当前的日期和时间
var today=new Date(3000);   //1970年1月1日，0时0分3秒
var today=new Date(“Apr 15,2016 15:20:00”);// 2016年4月15日15:20:0
var today=new Date(2016,3,25,14,42,50,50);// 2016年4月25日14:42:50
```

16.1.2 Date-提取日期字段方法

方法名	说明
getDate()	从Date对象返回一个月中的某一天(1～31)。
getDay()	从Date对象返回一周中的某一天(0～6)。
getMonth()	从Date对象返回月份(0～11)。
getFullYear()	从Date对象以四位数字返回年份。
getHours()	返回Date对象的小时(0～23)。
getMinutes()	返回Date对象的分钟(0～ 59)。
getSeconds()	返回Date对象的秒数(0～ 59)。
getMilliseconds()	返回Date对象的毫秒(0～999)。
getTime()	返回至今的毫秒数。

16.1.2 Date-日期转换与调整

将日期转化为字符串较

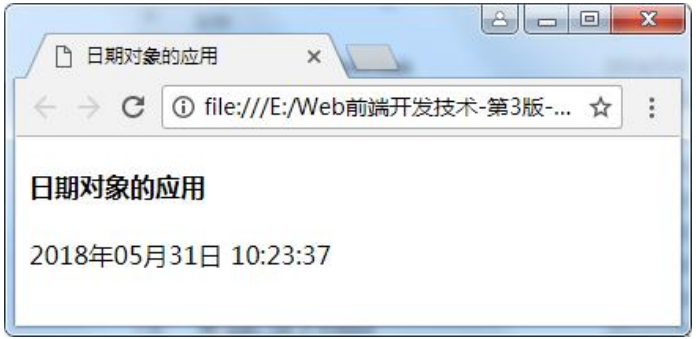
```
today.toString(); //把Date对象转换为字符串
today.toLocaleString(); //转换为本地时间串
today.toDateString(); //日期部分转为字符串
today.toTimeString();//时间部分转为字符串
```

调整日期对象的日期和时间

```
var today = new Date();
today.setDate(today.getDate()+5); //将日期调整到5天以后，如果碰到跨年月，自动调整
today.setFullYear(2009,11,11); //调整today对象到2009-11-11，月和日期参数可以省略
```

16.1.2 Date-案例

```
<!-- edu_16_1_2.html -->
<html >
<head>
<title>日期对象举例</title>
</head>
<body >
<script type="text/javascript">
var now = new Date();
var y = now.getFullYear();
var m = now.getMonth()+1;
var d = now.getDate();
var h = now.getHours();
var mi = now.getMinutes();
var s = now.getSeconds();
if(m<10){m= “0” +m} //处理成两位表示
if(d<10){d="0"+d}
if(h<10){h="0"+h}
if(mi<10){mi="0"+mi;}
if(s<10){s="0"+s;}
var str = y+"年"+m+"月"+d+"日
   "+h+":"+mi+":"+s;
document.write(str);
</script>
</body>
</html>
```



16.1.3 Math

Math 对象提供多种算术常量和函数，执行普通的算术任务。可以直接通过“Math”名来使用它提供的属性和方法。

```
var area=Math.PI*radius*radius ;//计算圆的面积
var r= Math.random();  //生成介于 0.0 - 1.0之随机数
var s3=sqrt(10); //10的平方根，值小于0，则返回 NaN。其他函数:
Math.max(x,y):返回 x 和 y 中的最高值。
var max=Math.max(100,200,300);//max=300
Math.min(x,y):返回 x 和 y 中的最低值。
var min=Math.min(1,45,67);//min=1
Math.pow(x,y):返回 x 的 y 次幂(xy)。
```

16.1.3 Math

ceil()：对数进行上舍入。 `var x=Math.ceil(10.5);//x=11`

floor()：对数进行下舍入。 `var x=Math.floor(10.5);//x=10`

round()：把数四舍五入为最接近的整数。

```
var x=Math.round(10.5); //x=10,
var y=Math.round(-10.5);//y=-10
```

exp()：返回 e 的指数。

```
var x=Math.exp(1);//x=2.718
```

log()：返回数的自然对数（底为e）。

pow(x,y)：返回 x 的 y 次幂。

产生某一区间数据方法：[m,n]

```
var num=Math.floor(Math.random()*(n-m+1)+m)
var num=Math.round(Math.random()*(n-m)+m)
var num=Math.ceil(Math.random()*(n-m)+m)
```

16.1.3 Math-案例

```
<!--  edu_16_1_4.html -->
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>随机产生[m,n]区间内10个整数</title>
<script  type="text/javascript">
function $(id){return document.getElementById(id);}
function createInt(){
var m=parseFloat($("minN").value);//解析为实数
var n=parseFloat($("maxN").value);//解析为实数
var array_int=new Array();
if(m>=n)    //合法性检验
{  alert("数组上、下限不能相同！主重新输入");
    $("minN").focus();//让文本框自动获取焦点
} else   {
for(var i=0;i<10;i++)
{ //产生 m-n 之间的随机数
    array_int[i]=Math.round((Math.random()*(n-m)+m)); }
}
$("array_num").value=array_int.join(",");
```

16.1.3 Math-案例

```
}</script>
</head><body>
<h3>随机产生[m,n]区间内10个整数</h3>
<form name="Form1">
  下限: <input type="text" name="minN" id="minN" size="20" value=10>
  上限: <input type="text" name="maxN" id="maxN" size="20" value=90><br><br>
  产生数组：<input type="text" name="" id="array_num" size="40" readonly><br><br>
  <input type="button" value="产生10个随机整数" onclick="createInt();">
  <input type="reset">
</form></body></html>
```

16.1.4 Number

使用强制类型转换函数Number(value)可以把给定的值转换成数字（可以是整数或浮点数）。

```
var ss=Number(false)          //返回值为0
var ss=Number(true)           //返回值为1
var ss=Number(null)           //返回值为0
var ss=Number(100)            //返回值为100
var ss=Number("5.5 ")         //返回值为5.5
var ss=Number("56 ")          //返回值为56
var ss=Number(undefined)      //返回值为NaN
var ss=Number("5.6.7 ")   //返回值为NaN，与parseFloat(“5.6.7”)不同
var ss=Number(new Object())   //返回值为NaN
```

16.1.5 String

String对象属于JavaScript核心对象之一。主要提供诸多方法实现字符串检查、抽取子串、字符串连接、字符串分割等字符串相关操作，可以通过如下方式生成String对象。创建String对象:

```
var  s1 = "Welcome to you!";
var  s2 = new String("Welcome to you!");
```

◆强制类型转换String(value)

```
var s1=String("100")        //返回值为字符串100
var s1=String("acdd")       //返回值为字符串acdd
var s1=String("false")      //返回值为字符串false
var s1=String(true)         //返回值为字符串true
var s1=String(null)         //返回值为字符串null
var s1=new Array("111","222","333");String(s1)   //返回值为111,222,333
var s1=String(new Object())       //返回值为字符串[object,Object]
```

16.1.5 String

1.获取String对象长度属性

String对象常用的属性有length，返回目标字符串中字符数目。

```
var s1 = "hello,world";
var len = s1.length; //s1.length返回11，s1所指向的字符串有11个字符
```

2.连接两个字符串

String对象的concat( )方法能将作为参数传入的字符串加入到调用该方法的字符串的末尾，并将结果返回给新的字符串。

```
var targetString=new String("Welcome to ");
var strToBeAdded=new String("the world!");
var finalString=targetString.concat(strToBeAdded);
```

3.把字符串分割为字符串数组

split()方法可以把字符串分割成字符串数组。

16.1.5 String

```
<script type=text/javascript>
    var str1 = " How are you doing today?"
    var subarray =str1.split(" "); //subarray是一个数组
    for(var i=0;i<subarray.length;i++)
    {    document.write(subarray [i]);
         document.write("<br>"); }
 </script>
```

- Split()方法的返回值是字符串数组。可用Array对象的方法访问字符串数组中的元素。Split()方法分割方法还有很多。

```
    var sub1 = str1.split(""); //把字符串按字符分割，返回数组["H","o","w",...]
    var sub2 = str1.split("o"); //把字符串按字符o分割，返回数组["H","w are y","u d","ing t","day?"]
```

16.1.5 String

4.String对象的显示风格方法

big()：变大些 ；　　small()：变小些；
fontsize()：字体大小；　fontcolor()：字体颜色；
bold()：变粗些　　；italics()：斜体；
sub()：下标　；　　up()：下标

5.字符串的大小写转换

toLowerCase()：把字符串转换为小写
toUpperCase()：把字符串转换为大写

16.1.5 String-案例

```
<!-- edu_16_1_5.html -->
<!doctype html>
<html lang="en">
<body>
<h4>字符串显示风格方法的应用</h4>
<script type="text/javascript">
var MyString=new String("How Are You?");
document.write("原始字符串："+MyString+"<br><hr>");
document.write("big()方法："+MyString.big()+"<br>");
document.write("smal ()方法："+MyString.small()+"<br>");
document.write("bold()方法："+MyString.bold()+"<br>");
document.write("fontcolor('ff0000')方法：
"+MyString.fontcolor('ff0000')+"<br>");
document.write("fontsize(5)方法：
"+MyString.fontsize(5)+"<br>");
document.write("italics()方法："+MyString.italics()+"<br>");
document.write("strike()方法："+MyString.strike()+"<br>");
document.write("sub()方法："+MyString.sub()+"<br>");
document.write("sup()方法："+MyString.sup()+"<br>");
</script>
</body>
</html>
```

16.1.6 Boolean

Boolean对象

```
var bo=new Boolean(value);
var bo1=Boolean(value);
```

- **参数为下列情况时返回true**

  1、true、”true”、”false”、”dfaf a”

- **参数为下列情况时返回flase**

0、null、false、NaN、“”、空

16.2 HTML DOM

- document对象是客户端JavaScript最为常用的对象之一，在浏览器对象模型中，它位于window对象的下一层级。
- document对象包含一些简单的属性，提供了有关浏览器中显示文档的相关信息，例如：该文档的URL、字体颜色，修改日期等。
- document对象还包含一些引用数组的属性，这些属性可以代表文档中的表单、图象、链接、锚以及applet。
- 同其他对象一样，document对象还定义了一系列的方法，通过这些方法，可以使JavaScript在解析文档时动态地将HTML文本添加到文档中。

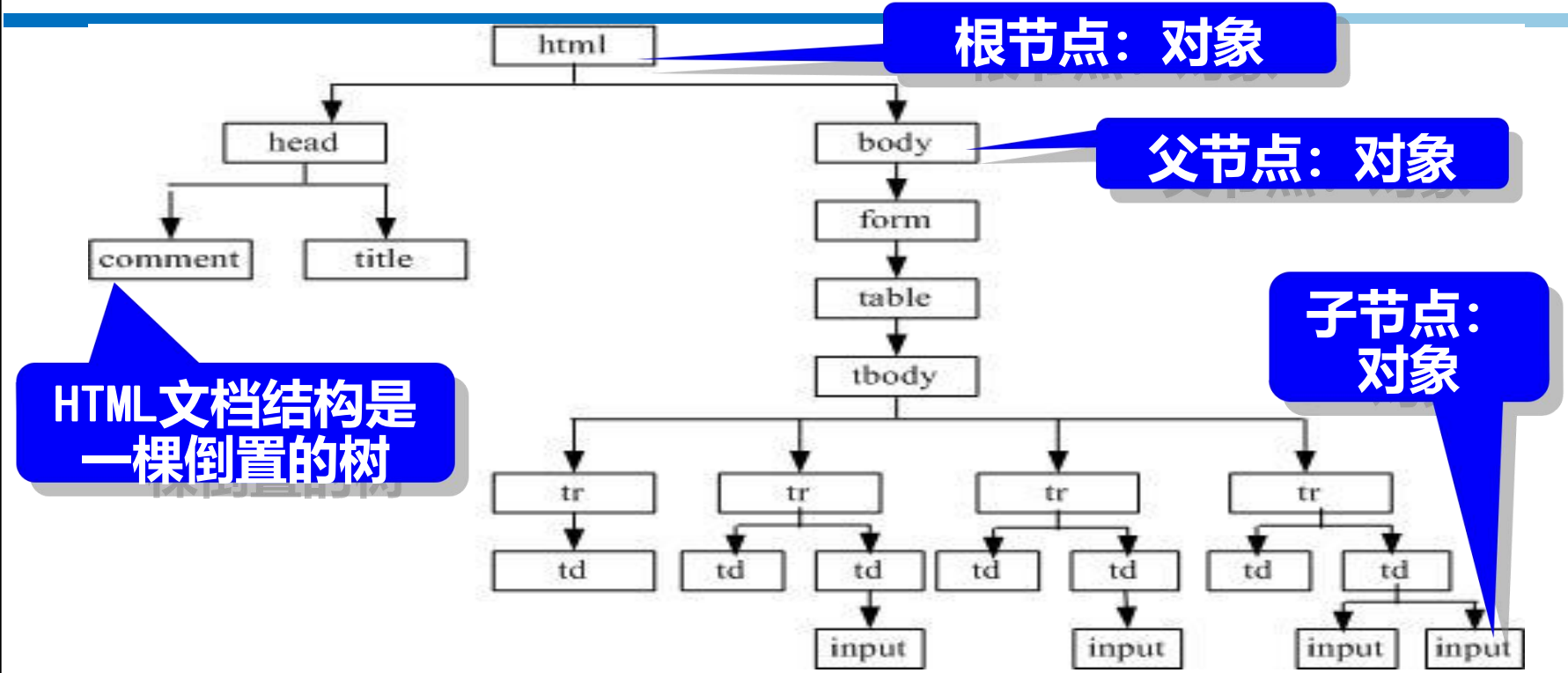
16.2.1 DOM简介

DOM（Document Object Model）模型

HTML DOM定义了访问和操作HTML文档的标准方法。

DOM将HTML文档表达为树结构。HTML文档结构好像倒垂的一棵树一样，其中<html>标记就是树的根节点，<head>、<body>是树的两个子节点。这种描述页面标记关系的树型结构称为DOM节点树（文档树）。

16.2.2 DOM节点树



注：由DOM节点树可以写出相应的HTML代码。

16.2.3 DOM节点

根据HTML DOM规范，HTML文档中的每个成分都是一个节点。具体的规定如下：

- 整个文档是一个文档节点；
- 每个 HTML 标签是一个元素节点；
- 包含在 HTML 元素中的文本是文本节点；
- 每一个 HTML 属性是一个属性节点；
- 注释属于注释节点。

16.2.3 DOM节点

1.根节点：

```
var root=document.documentElement;
```

2.子节点：

```
var aNodeList=root.childNodes; //对象数组
```

节名（如根节点root）.firstChild、节名（如根节点root）.lastChild可获取一个节点的第一和最后一个子节点。节点有三个属性分别是nodeName、nodeType、nodeValue。

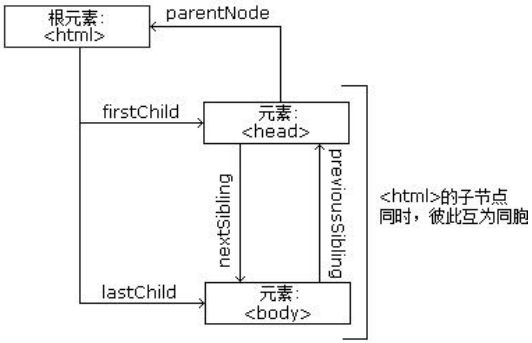
3.父节点：

```
var parentNode=aNode.parentNode;
```

4.兄弟节点：

```
var preNode=aNode.previousSibling;//返回前个节点引用
var nextNode=aNode.nextSibling;//返回后一个节点引用
```

16.2.3 DOM节点-关系



HTML文档来主要节点：

- 元素节点 (Element Node), <html>,<head>、<body>、<h1>、<p>和<ul>等标签都是元素节点。
- 文本节点(Text Node),文本节点包含在元素节点内，如h1、p、li等节点就可以包含一些文本节点。
- 属性节点(Attribute Node),属性节点总是被包含在元素节点当中，可以通过元素节点对象调用getAttribute( )方法来获取属性节点。

16.2.4 DOM节点访问

访问节点的方式很多种，可以通过document对象的方法来访问节点，也可以通过元素节点的属性来访问节点。

- 通过getElementById( )方法访问节点

```
var s=document.getElementById(id);
```

- 通过getElementsByName ( )方法访问节点

```
var s=document.getElementByName(name);
```

- 通过getElementsByTagName ( )方法访问节点

```
var s=document.getElementByTagName(tagname);
```

- 通过form元素访问节点

```
var loginform = document.myform;//myform为form对象的名称
var username1=loginform.elements[0];//通过elements属性来访问
var username2=loginform.username;//通过name属性来访问
```

16.2.4 DOM节点访问-方法1

1）通过ID访问页面元素

语法： document.getElementById(id)

参数： id必选项，为字符串(String)

返回值：对象或null(无符合条件的对象)

例如：var userName=document.getElementById("userName").value;

注意：

- 最好为需要交互的元素设定一个唯一的id，以便查找；
- getElementById()返回的是一个页面元素的引用.
- 元素ID引用时，必须加引号，如“userName”，否则易引起语法错误，缺少对象。

```
function $(id){return document.getElementById(id);}//调用时参数加双引号
```

16.2.4 DOM节点访问-方法2

2) 通过Name访问页面元素

语法： document.getElementsByName(name);

参数： name :必选项为字符串(String)

返回值：数组对象; 如果无符合条件的对象，则返回空数组

例如：
```
var userNameInput=document.getElementsByName("userName");
var userName=userNameInput[0].value;
```

使用该方法需要注意：

- 该方法返回一个数组，引用元素必须通过下标进行；
- 如果返回一个长度为0的数组，没有元素；可以通过对象的length属性来判断。

```
function $name(name){return document.getElementsByName(name);}
```

16.2.4 DOM节点访问-其它方法

3）通过标记名访问页面元素

语法： document.getElementsByTagName(tagname);

参数： tagname:必选项为字符串(String)

返回值：数组对象; 如果无符合条件的对象，则返回空数组

```
function $tag(tagname){
return document.getElementsByTagName(tagname);} //调用时加双引号
```

4）获得当前页面所有的Form对象

语法： document.forms

参数： 无；返回值：数组对象; 如果无符合条件的对象(Form对象)，则返回空数

```
var loginform=document.forms[0]; varusername=loginform.elements[0];
var password=loginform.elements[1];
```

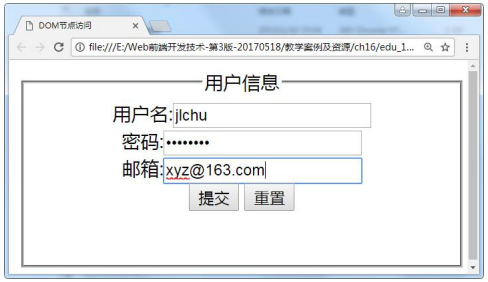
16.2.5 DOM节点操作

1.创建和修改节点

方法名	说明
createElement(tagname)	创建标记名为tagname的节点。
createTextNode(text)	创建包含文本text的文本节点。
createDocumentFragment()	创建文档碎片。
createAttribute()	创建属性节点。
createComment(text)	创建注释节点。
removeChild(node)	删除一个名为node的子节点。
appendChild(node)	添加一个名为node的子节点。
insertBefore(nodeB,nodeA)	在名为nodeA节点前插入一个名为nodeB的节点
replaceChild(nodeB,nodeA)	用一个名为nodeB节点替换另一个名为nodeA节点
cloneNode(boolean)	克隆一个节点，它接收一个boolean参数，为true时表示该节点带文字；false表示该节点不带文字

16.2.5 DOM节点操作-案例

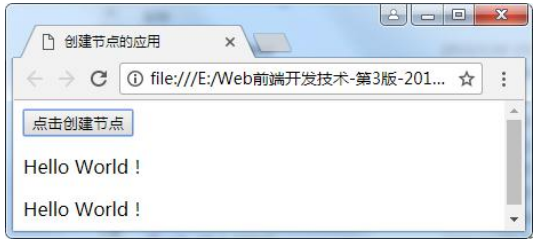
```
<!--  edu_16_2_2.html -->
<!doctype html>
<html lang="en">
   <head><meta charset="UTF-8">
   <title>DOM节点访问</title>
   <script type="text/javascript">
       function validate(){ //此处为用户登录时的校验处理代码 }
   </script></head>
   <body>
   <form method="post" action="" name="myform">
   <fieldset style="width:350px;height:150px;text-align:center;">
   <legend align="center">用户信息</legend>
   用户名:<input type="text" name="username" id="username">br>
   密码:<input type="password" name="password"
   id="password"><br>
   邮箱:<input type="text" name="email" id="email"><br>
   <input type="button" value="提交" onclick="validate();">
   <input type="reset">
   </fieldset>
   </form></body>
</html>
```



16.2.5 DOM节点操作案例1

```
<!--  edu_16_2_3.html -->
<html>
<head>
<title>创建节点举例</title>
<script  type="text/javascript">


function createP(){
   var op =document.createElement("p");
   var otext = document.createTextNode("Hello World！ ");
   op.appendChild(otext);
   document.forms[0].appendChild(op); }
</script>
</head>
<body>
    <form name="form1">
      <input type="button" value="点击创建节点" onClick="createP()">
    </form>
    </body>
   </html>
```



14.2.5 DOM节点操作案例2

```
<!--  edu_16_2_4.html -->
<html>
<head>
<title>删除、插入、替换节点举例</title>
<script language="javascript" type="text/javascript">
function operateNode(){
   //将页面上的<p>元素删除
   var p = document.getElementsByTagName("p")[0];
   document.form1.removeChild(p);
   //将页面中的<h2>元素更换为<h5>元素并重新设置文本节点内容
   var h5 =document.createElement("h5");
   var otext = document.createTextNode("web前端开发技术！");
   h5.appendChild(otext);
   var h2 = document.getElementsByTagName("h2")[0];
   document.form1.replaceChild(h5,h2);
   //在hdb元素前插入一个<p>元素
```

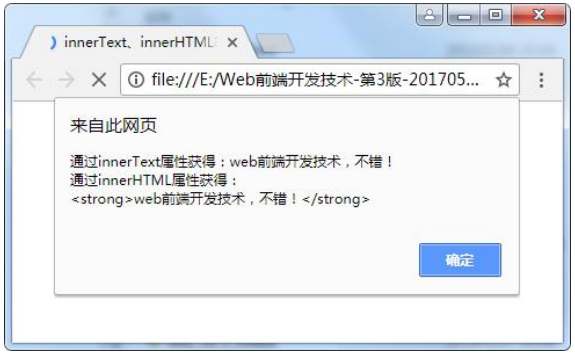
16.2.5 DOM节点操作案例2

```
var op =document.createElement("p");
var otext = document.createTextNode("中国的是世界的！");
op.appendChild(otext);
var hdb = document.getElementsByTagName("hdb")[0];
document.form1.insertBefore(op,hdb);
}
</script>
</head>
<body>
<form name="form1">
  <h2>javaScript程序设计</h2>
  <p>hello world!</p>
  <hdb>世界的也是中国的！</hdb><br>
  <input type="button" value="点击修改节点"
onClick="operateNode()">
</form>
</body>
</html>
```

16.2.5 DOM节点操作案例3

```
<!--  edu_16_2_5.html -->
<html><head>
<title>innerText、innerHTML举例</title>
<script type="text/javascript">
function textGet() {
   var oDiv = document.getElementById("oDiv");
   var msg = "通过innerText属性获得：";
   msg+=oDiv.innerText;
   msg+="\n通过innerHTML属性获得： "
   msg+=oDiv.innerHTML;
   alert(msg);
}
</script>
</head>
<body onload="textGet()">
   <div id="oDiv" >
   <strong>web前端开发技术，不错！</strong>
   </div>
</body>
</html>
```

2.节点的innerText和innerHTML属性



16.2.5 DOM节点操作案例4

3.获取并设置指定元素属性

```
<!--  edu_16_2_5.html -->
<!doctype html>
<html lang="en">
<head>
<title>获得、设置节点属性</title>
<style type="text/css">
td{text-align:center;}
</style>
<script type="text/javascript">
var table, color   //全局变量
function $(id){return document.getElementById(id);}
function randomInteger(){//随机产生0-255之间的整数
var int=Math.floor(Math.random()*256);
return int;}
function changeColor(){
table=$("myTable");     //全局变量
color=table.getAttribute("bgcolor");  //保存原始值
var rc=randomInteger().toString(16);//转换十六进制
```

16.2.5 DOM节点操作案例5

```
var gc=randomInteger().toString(16);//转换十六进制
var bc=randomInteger().toString(16);//转换十六进制
var color1="#"+rc+gc+bc;   //形成6位十六进制数
table.setAttribute("bgColor",color1);
}
function restoreColor(){
table.setAttribute("bgColor",color);
}
</script>
</head>
<body>
<form method="post" action="">
<table align="center" border="1" bgColor="#99cccc" width="500px" id="myTable">
<tr>
    <caption>专业学生花名册</caption>
    <td>序号</td><td>姓名</td><td>学号</td><td>专业</td>
</tr>
<tr>
<td>1</td><td>储致衡</td><td>1209520112</td><td>计算机科学与技术</td>
</tr>
```

16.2.5 DOM应用实例-getAttribute( )、setAttribute( )

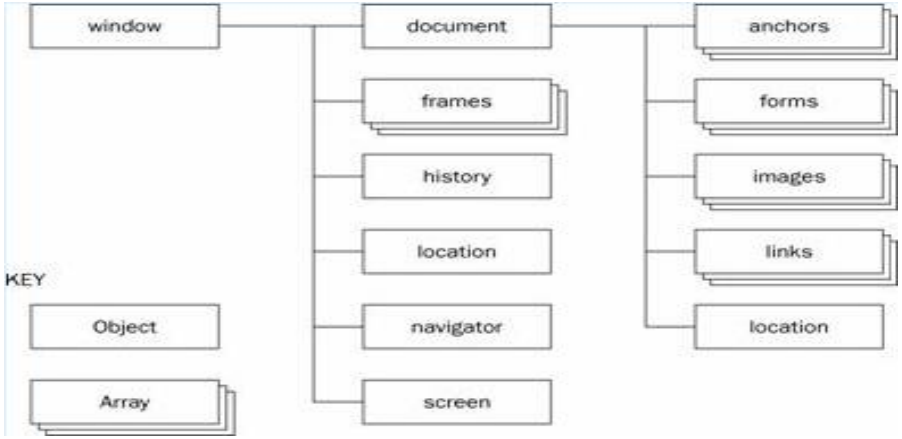
```
<tr>
<td>2</td><td>李大磊</td><td>1303020122</td><td>软件工程</td>
</tr>
<tr>
<td colspan="4"><input type="button" value="更改颜色" onclick="changeColor()"><input
type="button" value="还原颜色" onclick="restoreColor()"></td>
</tr>
</table>
</form>
</body>
</html>
```

16.3 BOM

客户端浏览器这些预定义的对象统称为浏览器对象，它们按照某种层次组织起来的模型统称为浏览器对象模型（BOM-Browser Object Model）。浏览器对象模型 (BOM) 使 JavaScript 有能力与浏览器“对话”。

BOM提供了独立于内容而与浏览器窗口进行交互的对象。BOM由一系列相关的对象构成，并且每个对象都提供了很多方法与属性。

浏览器对象的模型图如右边所示。



16.3.1 window对象

window对象位于浏览器对象模型的顶层，是document、frame、location等其他对象的父类。window对象的常用方法如下表。

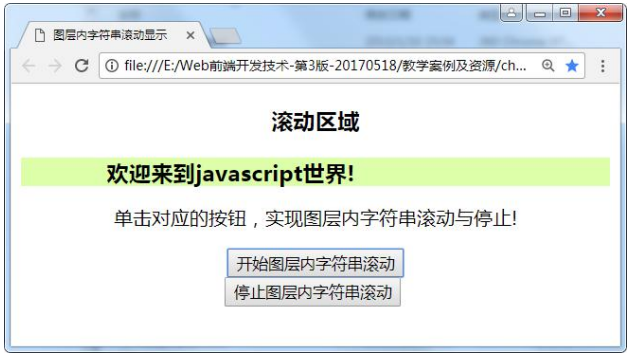
方法名	说明
open(url,name,features,replace)	打开新的浏览器窗口或查找一个已命名的窗口。
prompt(“提示信息”，默认值)	显示可提示用户输入的对话框。
blur()	把键盘焦点从顶层窗口移开。
close()	关闭浏览器窗口。
focus()	把键盘焦点给予一个窗口。
setInterval(code,interval)	按照指定的周期（以毫秒计）来调用函数或计算表达式。
setTimeout(code,delay)	在指定的毫秒数后调用函数或计算表达式。
clearInterval(intervalID)	取消由setInterval()设置的timeout。
clearTimeout(timeoutID)	取消由setTimeout()方法设置的timeout。

16.3.1 window对象-案例

```
<!--  edu_16_3_1.html -->
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>图层内字符串滚动显示</title>
<style type="text/css  ">
#myDiv{width:100%;height:24px;background:#DDFFAA;}
</style>
<script type="text/javascript">
var TimerID;
var loop=1;//设置启动次数，防止多次启动
var dir=1;//设置方向变量初值
var str_num=0;                    //用于动态显示的目标字符串
var str="欢迎来到javascript世界!";
function $(id){return document.getElementById(id);}
function startMove(){
//设定图层内动态显示的字符串信息
  var str_space="";
  str_num=str_num+1*dir;//动态改变运动步长
if(str_num>50 || str_num<0){
              dir=-1*dir;     }     //改变运动方向
for(var i=0;i<str_num;i++){
    str_space+=" ";}
$("myDiv").innerHTML="<h3>"+str_space+str+"</h3>";//动态赋值
}
function MyStart(){
  //图层内字符串滚动开始
  if (loop==1)
{TimerID=setInterval("startMove();",100);}
loop++;
}
function MyStop(){
//图层内字符串滚动结束，并更新图层内字符串
clearInterval(TimerID);
```

16.3.1 window对象-案例

```
loop=1;//恢复初始值
$("myDiv").innerHTML="<h3>图层内字符串滚动结束
      !</h3>";
}
</script>
</head>
<body >
<h3 align="center">滚动区域</h3>
<div name="" id="myDiv">欢迎来到JavaScript世界
      !</div>
<div style="text-align:center;">
<p>单击对应的按钮，实现图层内字符串滚动与停止!</p>
<form name="MyForm">
<input type="button" value="开始图层内字符串滚动"
      onclick="MyStart()"><br>
<input type="button" value="停止图层内字符串滚动"
      onclick="MyStop()"> <br>
</form>
</div>
</body>
</html>
```



16.3.2 navigator对象

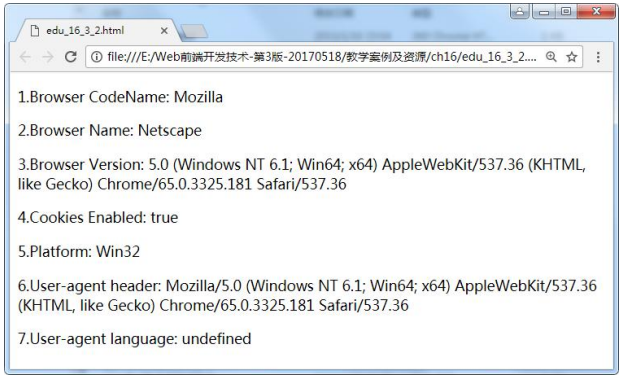
navigator对象用于获取用户浏览器的相关信息。

- navigator对象的属性

属性名	说明
appName	返回浏览器的名称。
appVersion	返回浏览器的平台和版本信息。
platform	返回运行浏览器的操作系统平台。
systemLanguage	返回操作系统使用的默认语言。
userAgent	返回由客户机发送服务器的user-agent头部的值。
appCodeName	返回浏览器的代码名。
appName	返回浏览器的名称。
appVersion	返回浏览器的平台和版本信息。

16.3.2 navigator对象-案例

```
<!-- edu_16_3_2.html -->
<html>
<head>
<title>navigator对象实例</title>
<script type="text/javascript">
function getInfo(){
  document.writeln("<h3>navigator信息</h3><br>");
  document.writeln("Web浏览器名称:
"+navigator.appName+"<br>");
  document.writeln("Web浏览器版本:
"+navigator.appVersion+"<br>");
  document.writeln("运行浏览器的平台:
"+navigator.platform+"<br>");
  document.writeln("浏览器执行的语言版本:
"+navigator.language+"<br>");
  document.writeln("用户代理:
"+navigator.userAgent+"<br>");
}
</script></head>
<body onload="getInfo()"></body>
</html>
```



16.3.3 screen对象

screen对象用于获取用户屏幕设置的相关信息，具有height、width、availHeight、availWidth等属性。

```
<!-- edu_16_3_3.html -->
<html>
<head><title>screen对象实例</title>
<script type="text/javascript">
function getScreenInfo(){
document.write("<h3>screen对象的信息</h3><br>");
document.write("屏幕的总高度："+screen.height+"<br>");
document.write("屏幕的可用高度：
"+screen.availHeight+"<br>");
document.write("屏幕的总宽度："+screen.width+"<br>");
document.write("屏幕的可用宽度：
"+screen.availWidth+"<br>"); }
</script></head>
<body onload="getScreenInfo()"></body>
</html>
```

16.3.4 history对象

history对象表示窗口的浏览历史，并由window对象的history属性引用该窗口的history对象。

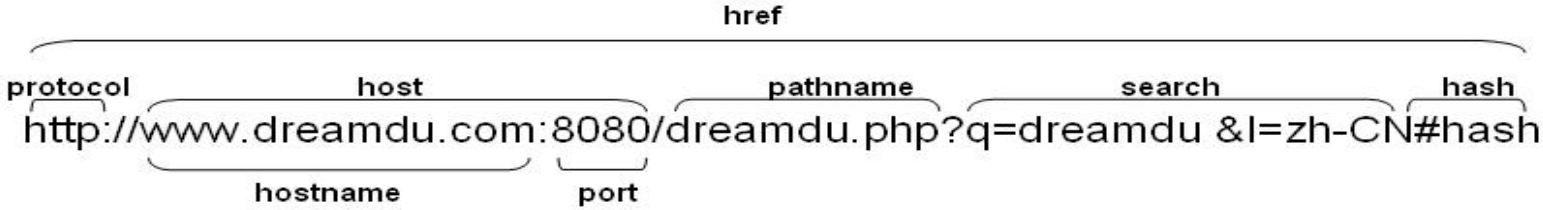
方法名	说明
forward()	加载history列表中的下一个 URL。
back()	加载history列表中的前一个 URL。
go(number|URL)	加载history列表中的某个具体页面。URL参数指定要访问的URL，number参数指定要访问的URL在history的URL列表中的位置。

- 在实际开发中，如下使用history方法：

```
history.back()   //与单击浏览器后退按钮执行的操作一样
history.go(-2)   //与单击两次浏览器后退按钮执行的操作一样
history.forward()  //等价于点击浏览器前进按钮或调用history.go(1)。
```

16.3.5 location对象

location对象用来表示浏览器窗口中加载的当前文档的URL，该对象的属性说明了URL中的各个部分，



16.3.5 location对象

location对象的常用属性

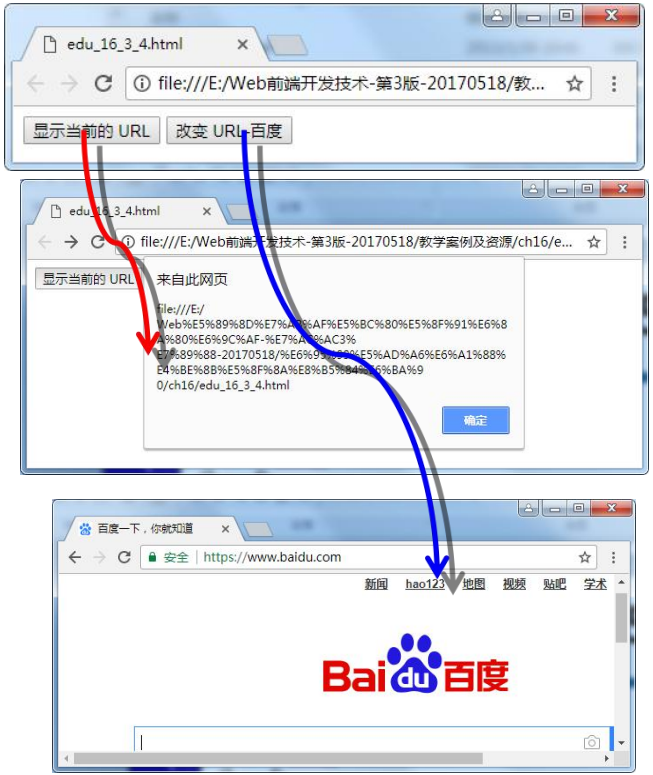
属性名	说明
hash	设置或返回从井号(#)开始的URL(锚)
href	设置或返回完整的URL
hostname	设置或返回URL中的主机名
protocol	设置或返回当前URL的协议
port	设置或返回当前URL的端口号
Pathname	设置或返回当前URL的路径部分
Host	设置或返回URL中的主机名和端口号的组合
Search	设置或返回从问号(?)开始的 URL(查询部分)

location对象的属性及方法

方法名	说明
reload()	重新加载当前文档。
assign()	加载新的文档。
replace()	用新的文档替换当前文档。

16.3.5 location对象-案例

```
<!-- edu_16_3_4.html -->
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <script type="text/javascript">
  function currLocation(){alert(window.location)}
  function  newLocation(){
    window.location= " http://www.baidu.com "
  }
 </script>
</head>
<body>
 <input type="button" onclick="currLocation()"
value="显示当前的 URL">
 <input type="button" onclick="newLocation()"
value="改变 URL-百度">
</body>
</html>
```



16.4 综合实例

以“Web前端开发技术”课程网站开发为例，设计一个含有二级水平导航菜单、图像切换、下拉列表导航等功能的网站。

16.4 综合实例

1.页面布局设计

对照课程网站效果图进行DIV分区设计，页面布局如下。

2.网站中实现的主要技术

- ✓ DIV+CSS+JavaScript实现的二级导航菜单。
- ✓ JavaScript实现图像自动定时切换。
- ✓ Window对象open方法和select对象的options、seleccted Index属性实现下拉列表导航功能。

16.4 综合实例-二级菜单

1)JS二级导航菜单

二级水平导航菜单实现技术分析：一级菜单、二级菜单在不同区域中单独显示；一级导航菜单采用无序列表实现，1个li标记表示1个主导航栏目，两个导航栏目之间插入1个分隔线(空li标记,设置背景图像)；在一级导航菜单上设置onmouseover事件句柄属性，绑定事件处理函数qiehuan(num)；

二级导航菜单显示规则：默认显示第1个一级导航栏目对应的二级导航菜单，其余二级菜单默认是不显示，只有当鼠标悬停(盘旋)在相应的一级导航菜单上时才能调用qiehuan(num)函数，将id为“qh_con”+num的DIV的display属性值改为block，显示其对应的二级导航菜单；所有的二级导航菜单分别定义在不同的div中。

16.4 综合实例-二级导航

二级导航菜单统一放在1个id为“mmenu_con”的父DIV中，每1个二级子菜单单独放在1个独立的子DIV中。CSS样式定义参见style2menu.css，二级导航菜单结构如下：

```
<div id="menu_con">
    <div id="qh_con0" style="display: block">  <!-- 第一个子菜单 -->
      <ul>
          <li><a href="#"><span>课程发展</span></a></li>
          <li class="menu_line2"></li>          <!-- 子菜单分隔线 -->
          <li><a href="#"><span>课程特色</span></a></li>
          <li class="menu_line2"></li>          <!-- 子菜单分隔线 -->
          <li><a href="#"><span>教学成果</span></a></li>
      </ul>
    </div>
    <div id="qh_con1" style="display: block"> <!-- 第二个子菜单 -->
    ...
    </div>
    ...          <!-- 第n个子菜单 -->
</div>
```

16.4 综合实例-其它技术

2)图像自动定时切换

图像自动定时切换实现技术：在DIV中插入一个图像的超链接，定义图像img标记的id，通过id获取img对象，动态修改img的src属性，实现图像切换使用Windos对象的setInterval(code,interval)、clearInterval(intervalID)两个方法来实现间隔时间执行代码和取消执行代码。在switchpic.js中定义初始化init()、切换switchPic()、重新鼠标移出reStart()、鼠标悬停pause()共4个JavaScript函数。

3)下拉列表框导航

下拉列表框导航实现技术：设置select标记的onchange属性，并绑定事件代码，用windos对象的open(url,name,features,replace)实现在单击下拉列表框中任一选项时，能够打开相关的超链接。

16.4 综合实例-主体部分代码

◆ 下拉列表超链接设置

```
<select size="1" name="d1 "
onchange="window.open(this.options[this.selectedindex].value)">
    <option>网络课程资源链接</option>
    <option value="http://www.icourses.cn/home/">中国mooc大学
</option>
</select
```

3.主要实现的代码

1)参见edu_16_4_1.html。

2)二级导航菜单切换显示参见qiehuan.js代码

3)图像切换参见switchpic.js代码

4)页面CSS样式参见style2menu.css代码

本章小结

本章介绍对象的概念及Array、Date、Math、Number、String、Boolean等常用的核心对象。

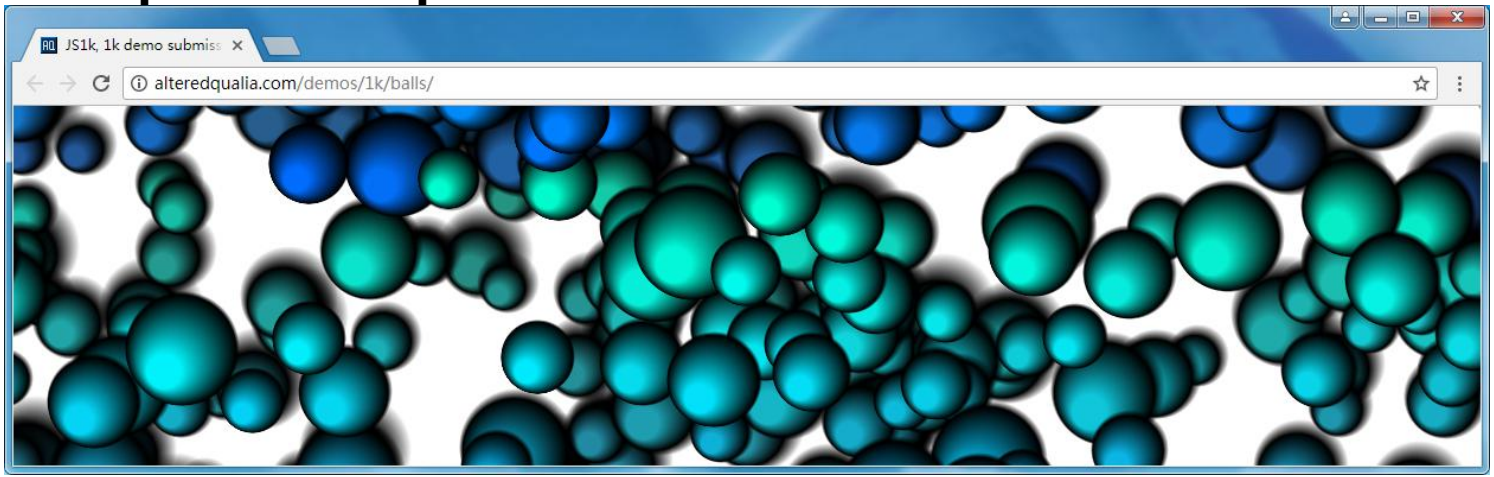
HTML文档中的每个标记都是一个节点，这些标记之间存在着一定的关系，这种描述页面标记关系的树型结构称为DOM节点树。对于DOM节点的访问除了通过form对象的elements属性或该节点的name属性来访问外，还可以通过document对象的getElementById()、getElementsByName()、getElementsByTagName()等方法来访问；document对象应用非常广泛，除了访问节点外，还可以调用该对象的方法和属性来动态地创建和修改节点、设置节点的属性。

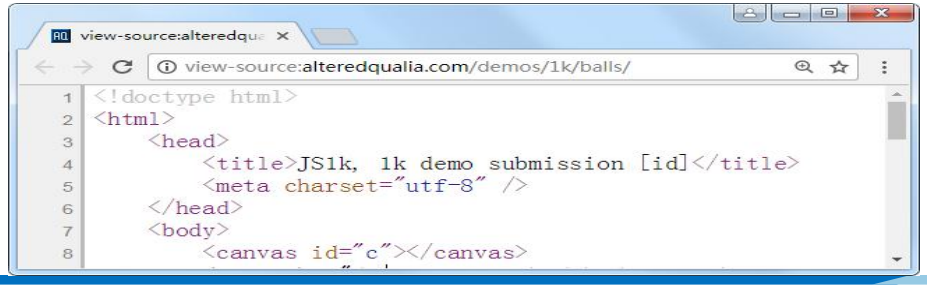
本章小结

BOM定义了浏览器对象（window、history、document、location、screen、navigator、frame等对象）的组成和相互关系，描述了浏览器对象的层次结构。在BOM中，每个对象都含有若干属性和方法，使用这些属性和方法可以操作Web浏览器窗口中的不同对象，控制和访问HTML页面中的不同内容。

第17章 HTML5高级应用

CANVAS应用:http://alteredqualia.com/demos/1k/balls/





本章学习目标

主要内容：

- 学会使用Web 本地存储对象解决客户端数据存储问题。
- 掌握Canvas 基本语法和学会绘制各种图形、文字及图像。
- 学会使用Web 拖放技术解决简单的实际应用问题。
- 理解Web Worker 多线程工作原理，学会使用多线程解决简单的实际应用问题。

17.1 HTML5 Web Storage

HTML5 提供了两种在客户端存储数据的新方法，分别是持久化的数据存储localStorage、会话式的数据存储sessionStorage。

在HTML5 中，数据不需要由每个服务器进行请求传递，只需在有请求时使用数据，这样就不会影响网站的性能，而且能够存储大量数据。数据通常以“键值对(key-value pair)”形式存在，Web网页的数据只允许该网页访问使用。

HTML5 之前客户端数据存储是由cookie完成的，由于cookie不能存储大量数据，需要通过服务器的请求来传递，往往造成Cookie响应速度慢、效率低。

17.1.1 localStorage 对象

localStorage 对象存储的数据没有时间限制，所以称为持久化存储，数据存储长期可用。

使用此类对象之前，最好先检查一下浏览器是否支持。检查代码如下：

```
if(typeof(Storage)!="undefined") {
        //是的! 支持 localStorage sessionStorage对象! //一些代码...}
else { //抱歉! 不支持web存储。 }
```

localStorage 对象和sessionStorage 对象具有同样的方法，仅仅是对象名称不同而已。

17.1.1 localStorage 对象(续)

localStorage 对象的常用方法：

- localStorage.setItem(key,value)：保存数据。
- localStorage.getItem(key)：读取数据。
- localStorage.removeItem(key)：删除单个数据。
- localStorage.clear()：删除所有数据。
- localStorage.key(index)：得到某个索引的key。

17.1.2 sessionStorage 对象

sessionStorage对象针对一个session进行数据存储。数据存储周期短，当用户关闭浏览器窗口后，数据会被删除。该对象的方法与localStorage 对象方法相同。

localStorage 对象案例1

【例17-1-1】localStorage 对象的应用


```
<!-- edu_17_1_1.html -->
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>localStorage对象的应用</title>
<style type="text/css">
div{
    text-align:center;
    padding:20px;
    border:10px ridge #005A9C;
    width:350px;
    height:250px;
    margin:0 auto;
}
</style>
</head>
```

localStorage 对象案例1

```
<body>
<div>
<h3>最可爱的人评选</h3>
<img src="45567.jpg" width="80" height="80" border="0" alt=""><br>
<p id="result"></p>
<p>刷新页面票数会增长。</p>
<p>请关闭浏览器后重试仍会增长</p>
</div>
<script type="text/javascript">
   var tickets=0;
   localStorage.setItem(tickets,0);
   if (localStorage.tickets){
       localStorage.tickets=parseInt(localStorage.tickets) +1;    }
   else{
       localStorage.tickets=1;
   }
   document.getElementById("result").innerHTML="已投: "+localStorage.tickets + "票";
</script>
</body>
</html>
```

localStorage 对象案例2

【例17-1-2】localStorage 对象实现的简易通讯录。


```
<script>
// 载入所有存储在localStorage的通讯录信息
loadAllInfo();
//保存一条通讯记录数据，同时显示在div中
function $(id){return
document.getElementById(id);}
function saveInfo(){
 var name1=$("username").value;//取姓名
 var phone1=$("userphone").value;    //取电话
 if (name1!="" && phone1!="") {//不为空处理
  localStorage.setItem(name1,phone1);
  loadAllInfo();
  alert("添加成功");
 }else{//姓名或电话为空，告警并获得焦点
  alert("请输入姓名和电话！");
  $("username").focus();
 }
}
```

localStorage 对象案例2

```
//以姓名查找通讯录信息
function findForName(){
var searchname =
$("search_name").value;
var searchphone =
localStorage.getItem(searchname); //取电话
$("userphone1").value=searchphone; //填充电话
}
//从localStorage中取出所有通讯录中信息，并展现到界面上
function loadAllInfo(){
// localStorage.clear();
var result = "";
if(localStorage.length>0){
result += "姓名
    电话
</br><hr>";
for(var i=0;i<localStorage.length;i++){
var name= localStorage.key(i);
var phone = localStorage.getItem(name);
result += name+"   ---   "+phone+"</br>";
}
$("displayallinfo").innerHTML = result;
}else{
$("displayallinfo").innerHTML = "数据为空
……";
}
}
//删除某一条通讯信息
function deleteName(){
localStorage.removeItem($("search_name")
.value); //取电话
$("search_name").value=""; //填充电话
loadAllInfo();
}
</script>
```

localStorage 对象案例2

```
<body>
<fieldset style="float:left;width:300px;text-align:center;">
        <legend>通讯录添加</legend><label for="name">姓名(key)：</label>
        <input type="text" id="username" name="username" required/><br/>
        <label for="telphone">电话(value)：</label>
        <input type="text" id="userphone" name="userphone" required/><br/>
        <br><input type="button" onclick="saveInfo()" value="添加通讯录"/>
        <input type="reset">
        <div id="displayallinfo" name="displayallinfo"></div>
</fieldset>
<fieldset style="float:left;width:300px;height:100px;text-align:center;">
        <legend>通讯录查询与删除</legend>
        <label for="search_phone">输入姓名：</label>
        <input type="text" id="search_name" name="search_name" required/><br>
        <label>电话：</label><input type="text" name="" id="userphone1" readonly>
        <br><input type="button" onclick="findForName()" value="查找通讯录"/>
        <input type="button" onclick="deleteName()" value="删除通讯录"/>

</fieldset>
</body>
```

17.1.3 浏览器端数据库IndexedDB

WebStorage可以用来存储键值对，然而它无法提供按顺序检索、高性能地按值查询或存储重复的键的功能。

IndexedDB 是一种轻量级NoSQL（Not Only SQL，泛指非关系型）数据库，用来持久化大量（250MB）客户端数据。它可以让Web 应用程序具有非常强大的查询能力，并且可以离线工作。IndexedDB 的数据操作直接使用JS 脚本，不依赖SQL 语句（最初的Web SQL数据库已被废弃），操作返回均采用异步。

localStorage 和sessionStorage 对象是采用同步技术实现少量（2.5～10MB）客户端数据（字符串）存储。

一个网站可能有一个或多个IndexedDB 数据库，每个数据库必须具有唯一的名称。

17.1.3 浏览器端数据库IndexedDB

使用IndexedDB 的基本步骤如下：

- 打开数据库并且开始一个事务。
- 创建一个对象仓库(Object Store)。
- 构建一个请求来执行一些数据库操作，例如增加或提取数据等。
- 通过监听正确类型的DOM 事件以等待操作完成。
- 在操作结果上进行一些操作（可以在request 对象中找到）。

1．浏览器支持IndexedDB 数据库情况判断

```
var indexedDB=window.indexedDB || window.mozIndexedDB ||
    window.webkitIndexedDB; //获得IndexedDB对象
if(window.indexedDB){
    alert("您的浏览器支持IndexedDB数据库。");
}else{alert("您的浏览器不支持IndexedDB数据库。");}
```

17.1.3 浏览器端数据库IndexedDB

2．数据库创建与打开

```
var request=window.indexedDB.open(DBName, DBVersion);
                                    //数据库存在打开它；否则创建
```

说明：若**DBName** 数据库创建之前并不存在，则会调用**onupgradeneeded** 接口，在这个函数中可以进行数据库初始化和创建索引。

在连接到数据库后，request 会监听三种状态。

- success：打开或创建数据库成功后绑定指定函数。
- error：打开或创建数据库失败后绑定指定函数。
- upgradeneeded：更新数据库后绑定指定函数。

【例17-1-3】创建名为myBooks 的数据库，并创建名为books的数据仓库，为数据仓库添加两个对象数据。

数据库创建与打开案例

```
<!--edu_17_1_3.html -->
<!DOCTYPE html>
<html><head><meta charset="UTF-8">
   <title>创建数据库和对象仓库</title>
   </head>
   <body><script>
//数据库存在，则打开它；否则创建。
   var request=window.indexedDB.open("myBooks",1);
   //捕获连接失败事件，并处理
request.onerror = function(event) {
   alert( “数据库连接失败： “+event.target.errorCode);//提示错
   误
}
request.onupgradeneeded = function(event) {
  // 当此数据库创建前不存在时，进行初始化
  var db = request.result;
  var store = db.createObjectStore("books", {keyPath: "isbn"});
  var titleIndex = store.createIndex("by_title", "title", {unique:
   true});//标题索引
  var authorIndex = store.createIndex("by_author", "author");//
   作者索引
// 填入初始值，添加2本书信息
  store.put({title:"计算机组成
原理(修订版)",author:"张功萱",
isbn:"9787302433637"});
  store.put({title:"Java 2实用
教程(第5版)", author: "耿祥义
", isbn:"9787302464259 "});
}
request.onsuccess =
function(event) {//捕获连接成
功事件，并处理
db = event.target.result; //
连接成功时，获取数据库对象(也
可用request.result)
  alert("数据库连接成功");
}
</script>
</body>
</html>
```

17.1.3 浏览器端数据库IndexedDB

upgradeneeded状态是在indexedDB 创建新的数据库时和indexedDB.open（DBName,DBVersion）DBVersion（数据库版本号）发生变化时才能监听到此状态。当版本号不发生变化时，不会触发此状态。数据库的ObjectStore 的创建、删除等都是在这个监听事件下执行的。

需要注意的有两点：

(1)当数据库连接时，open()方法返回一个IDBOpenDBRequest 对象，调用函数定义在这个对象上。

(2)在连接建立成功时，会触发onsuccess 事件，调用函数接收一个事件对象event 作为参数，其target.result 属性指向打开的IndexedDB 数据库。也可以使用监听器来捕获3 个事件，分别为success、error、upgradeneeded。

17.1.3 浏览器端数据库IndexedDB

可以通过下列方法为页面元素（对象）添加事件监听器。

element.addEventListener(event, function, useCapture);

- 其中useCapture参数可选，布尔值，指定事件是否在捕获或冒泡阶段执行。其值为true 表示事件句柄在捕获阶段执行；默认值为false表示事件句柄在冒泡阶段执行。

- indexedDB 对象的Open()方法需要监听的事件代码如下：

```
request.addEventListener('success', function(event){ //打开或创建数据库成功
}, false); //第3个参数为false:表示在冒泡阶段执行
request.addEventListener('error', function(event){ //打开或创建数据库失败
}, false); //第3个参数为false:表示在冒泡阶段执行
request.addEventListener('upgradeneeded', function(event){
//更新数据库时执行
}, false); //第3个参数为false:表示在冒泡阶段执行
```

17.1.3 浏览器端数据库IndexedDB

3．创建与删除ObjectStore

- 1）createObjectStore()方法创建对象仓库

```
var request=window.indexedDB.open("myBooks",1);
var db = request.result;
var store=db.createObjectStore(storeName,{keyPath: primaryKey,
autoIncrement: true|false});  //keyPath称为键路径，作为搜索关键字
```

- 2）deleteObjectStore()方法删除对象仓库

```
db.deleteObjectStore(objectStoreName); //基本语法
db.deleteObjectStore("books"); //举例-删除books对象仓库
```

- 3）createIndex()方法为对象仓库创建索引

```
var indexName=store.createIndex(index_name, index_key, {unique:
true|false});
```

17.1.3 浏览器端数据库IndexedDB

- 4）objectStoreNames 属性检查对象仓库是否存在

```
if(!db.objectStoreNames.contains("books")) { //判断某个对象仓库是否存在
    db.createObjectStore("books"); //不存在则创建该对象仓库
}
```

4．使用事务

1）IndexedDB 中的事务模式

- readonly：(默认)提供对某个对象存储的只读访问，在查询对象存储时使用。
- readwrite：提供对某个对象存储的读取和写入访问权。
- versionchange：提供读取和写入访问权来修改对象存储定义，或者创建一个新的对象存储。

2）创建事务的基本语法

```
var transaction = db.transaction(storeName, [transactionmode]);//语法
var objectStore =transaction.objectStore(storeName); //获取对象仓库
```

17.1.3 浏览器端数据库IndexedDB

3）transaction()方法的事件类型

该方法有三种事件，分别是中断、完成和错误。

- abort：事务中断。
- complete：事务完成。
- error：事务出错。

事件处理代码结构如下所示：

```
var transaction = db.transaction(["books"], "readonly");
 transaction.oncomplete = function(event) {
      console.log("数据添加成功！"); //alert("数据保存成功！");};
 transaction.onerror = function(event) {
      console.log("Error",e.target.error.name);
     //错误处理};
 transaction.onabort = function(event) {
      alert("数据保存失败！");};
```

17.1.3 浏览器端数据库IndexedDB

5．数据库的增删改查

1）存储数据准备

```
var booklists=[{title: “Web前端开发技术-Html、Css、JavaScript” ,author: “储久良",isbn: "9787302431695"},{title:"计算机组成原理(修订版)", author:"张功萱 “,isbn: "9787302433637"},{title:"Java 2实用教程(第5版)", author: "耿祥义 “,isbn:"9787302464259 "}];
```

2）数据库的增加、更新、删除

```
objectStore.add(objectName); //添加数据，当关键字存在时数据不会添加
objectStore.put(objectName);
//更新数据，当关键字存在时覆盖数据，不存在时会添加数据
objectStore.delete(value); //删除数据，删除指定的关键字对应的数据
objectStore.clear(); //清除objectStore
```

3）数据库的数据读取

17.1.3 浏览器端数据库IndexedDB

var request = objectStore.get(value); //查找数据，根据关键字查找指定的数据

常用方式。分配事件句柄，并绑定事件处理函数。

```
request.onsuccess=function(e){
    var books=e.target.result;
    console.log(books.title); //控制台输出图书的标题
};
request.onerror=function(e){
    console.log("数据读取失败！"); //控制台输出图书的标题
};
```

- 事件监听方式。分配事件句柄，并绑定事件处理函数。

```
request.addEventListener('success', function(event){ //增加事件监听器
  //异步查找后的调用函数，省略    }, false);
request.addEventListener('error', function(event){ //增加事件监听器
    //错误处理函数，省略    }, false);
```

6．遍历数据openCursor()方法

使用对象仓库的openCursor()方法可以实现遍历数据。该方法可以获取游标对象，然后利用游标移动来实现数据遍历。

openCursor()方法可以接受第二个参数，表示遍历方向，默认值为next，其他值为prev、nextunique 和prevunique。后两个值表示如果遇到重复值，会自动跳过。openCursor()方法是异步执行的，有两个事件分别是success（检索请求成功）和error（检索请求失败）

```
var cursor=objectStore.openCursor(); //打开游标，指派事件处理函数。
```

1）非索引查找

```
cursor.onsuccess=function(e){};
cursor.onerror=function(e){};
```

continue():将光标移到下一个数据对象,已到最后一个对象，则光标指向null

6．遍历数据openCursor()方法

console.dir()：可以显示一个对象所有的属性和方法。

console.log()：输出在控制台中，方便以后的调试。

2）IDBKeyRange 对象

通过索引可以读取指定范围内的数据。使用浏览器原生的IDBKeyRange 对象能够生成

- 指定范围的range 对象。生成方法有四种。

lowerBound()方法：指定范围的下限。
upperBound()方法：指定范围的上限。
bound()方法：指定范围的上下限。
only()方法：指定范围中只有一个值。

6．遍历数据openCursor()方法

3）按索引查找数据

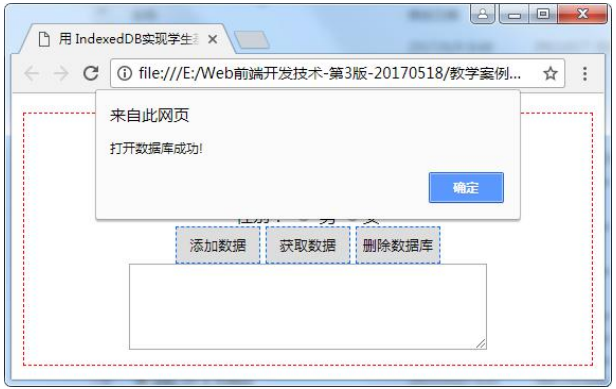
使用对象仓库的index()方法来实现检索。

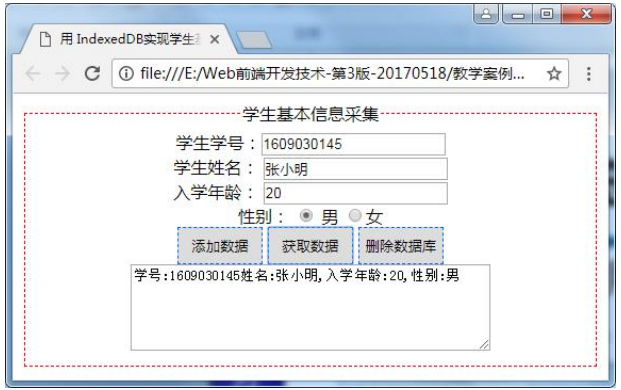
```
var index=objectStore.index(indexName); //indexName为已建立的索引名称
var cursor=index.openCursor(range); //用IDBKeyRange生成范围range
cursor.addEventListener('success', function(event){ //启动成功监听
    var result = event.target.result; //返回检索结果集
    if(result){
      console.log(result.value); //输出数据
      result.continue(); //迭代，游标下移
    }
}, false);
cursor.addEventListener('error', function(event){console.log("失败！ ");
},false);
```

浏览器端数据库IndexedDB案例

【例17-1-10】使用IndexedDB 实现学生基本信息采集系统。

创建userinfo数据库，创建user对象仓库。函数createDB(dbName) 的功能是根据指定参数创建数据库。函数deleteDB(dbName) 的功能是根据指定参数删除数据库。函数getObject() 的功能是获取满足条件的所有对象，并在多行文本域中分行显示。函数getOneObject(e) 的功能是读取某一个对象，并显示在多行文本域中。代码详见edu_17_1_10.html。





17.2 HTML5 Canvas 画布

HTML5 的canvas 标记用于图形的绘制，并通过JavaScript脚本来完成绘图。canvas标记本身并没有绘图能力，所有的绘制工作必须在JavaScript 内部完成。canvas 标记作为图形的容器，可以通过多种方法使用Canvas 绘制路径、盒、圆、字符以及添加图像。

17.2.1 canvas 标记

canvas 标记是双标记，必须设置宽度、高度及id。

1．基本语法

```
<canvas id="oneCanvas" width="" height=""></canvas>
```

默认情况下canvas 标记的width 为300px、高度height 为200px，页面上没有边框和内容。

常用的绘制颜色、样式和阴影的属性及说明如表17-1 所示。

17.2 HTML5 Canvas 画布

表17-1 颜色、样式和阴影

属性	描述
fillStyle	设置或返回用于填充绘画的颜色、渐变或模式。
strokeStyle	设置或返回用于笔触的颜色、渐变或模式。
shadowColor	设置或返回用于阴影的颜色。
shadowBlur	设置或返回用于阴影的模糊级别。
shadowOffsetX	设置或返回阴影与形状的水平距离。
shadowOffsetY	设置或返回阴影与形状的垂直距离。

2．绘制图形

利用canvas 标记绘制图形一般需要经过下列步骤：

(1)在body 标记中插入canvas 标记，并设置id、width、height。

```
<canvas id= “aCanvas” width= “200” height= “100” ></canvas>
```

(2)在body 标记中插入script 标记，并插入JavaScript 代码。

(3)通过id 获取页面上canvas 对象。

17.2 HTML5 Canvas 画布

var myCanvas=document.getElementById("aCanvas"); //获取Canvas对象

(4)创建具有绘图功能的环境对象context，参数为2d 或3d。

var conText=myCanvas.getContext("2d"); //获取绘图环境（也称上下文环境）

(5)在绘图环境对象内绘图。

- ✓ 填充。分为填充样式和填充图形。

```
conText.fillStyle="#FF0000";      //设置填充
conText.fillRect(10,10,150,75);  //填充矩形
```

- ✓ 绘制边框(外轮廓)。绘制样式和绘制图形及绘制线条的宽度（画笔粗细）。

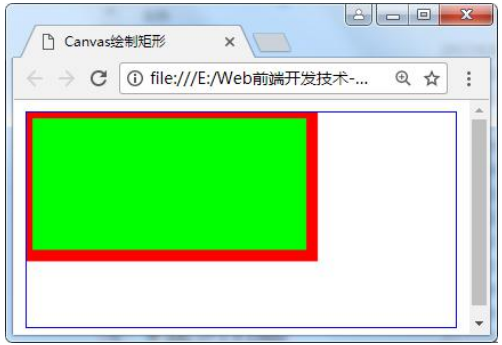
```
conText.strokeStyle="#FF0000"; //设置边框样式
conText.lineWidth=8;                  //图形边框宽度，不加单位px
conText.strokeRect(0,0,200,100); //绘制边框
```

- ✓ 清除矩形区域

```
conText.clearRect(x,y,width,height)
```

17.2 HTML5 Canvas 画布

【例17-2-1】用canvas 标记绘制矩形。




```
<!-- edu_17_2_1.html -->
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Canvas绘制矩形
</title>
</head>
<body>
<canvas id="oneCanvas" width="" height=""
style="border:1px solid blue"></canvas>
<script type="text/javascript">
    var myCanvas=document.getElementById("oneCanvas");
      //获取Canvas对象
   var conText=myCanvas.getContext("2d");
      //获取绘图环境（上下文环境）
   conText.fillStyle="#00FF00"; //设置填充样式
   conText.fillRect(0,0,200,100); //填充矩形
   conText.strokeStyle="#FF0000"; //设置边框样式
   conText.lineWidth=8; //图形边框宽度
   conText.strokeRect(0,0,200,100); //绘制边框
</script>
</body>
</html>
```

17.2.2 Canvas 坐标

Canvas 画布为二维网格，分X 轴和Y 轴，其中X 轴方向从左向右，Y 轴方向从上到下。

填充矩形方法：

fillRect (X,Y,width,height)；//X、Y为X 轴、Y 轴的坐标，其余两个参数分别表示矩形的宽度和高度。

17.2.3 Canvas 路径

在Canvas 上除了绘制矩形、正方形和直线外，需要使用路径来进行绘图。绘制前需要使用beginPath()方法开始路径，然后形成绘制路径，结束后需要使用closePath()方法关闭路径。最后才开始填充或绘制。

方法	描述
fill()	填充当前绘图（路径）。
stroke()	绘制已定义的路径（边框）。
beginPath()	起始一条路径，或重置当前路径。
moveTo()	把路径移动到画布中的指定点，不创建线条。
closePath()	创建从当前点回到起始点的路径。
lineTo()	添加一个新点，然后在画布中创建从该点到最后指定点的线条。
clip()	从原始画布剪切任意形状和尺寸的区域。
quadraticCurveTo()	创建二次贝塞尔曲线。
bezierCurveTo()	创建三次贝塞尔曲线。
arc()	创建弧/曲线（用于创建圆形或部分圆）。
arcTo()	创建两切线之间的弧/曲线。
isPointInPath()	如果指定的点位于当前路径中，则返回 true，否则返回 false。

Canvas 中绘图步骤。

（1）开始路径。方法为conText.beginPath()。

（2）绘制路径。方法如下：

```
conText.arc(150, 150, 100, 0, Math.PI * 2, true); //绘制路径
```

（3）关闭路径。方法为conText.closePath()。

（4）绘图。分为填充和绘制圆形边框，与绘制矩形类似。

```
conText.fillStyle = 'rgba(255,0,0,0.75)'; //填充样式
conText.fill();                           //填充绘图
conText.strokeStyle = 'rgba(0,255,0,0.50)'; //绘图样式
conText.lineWidth=20;                     //绘制边框宽度
conText.stroke();                         //绘图
```

rgba(red, green, blue, opacity)

opacity 属性表示透明度。允许的值为0～1 之间带有小数的数值。

【例17-2-2】Canvas绘制圆形案例

```
<!-- edu_17_2_2.html -->
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Canvas绘制圆形
</title>
</head>
```


```
<body>
<canvas id="oneCanvas" width="300" height="300"
style="background:#F0F0F0;"></canvas>
<script type="text/javascript">
    var myCanvas=document.getElementById("oneCanvas");//
获取Canvas对象
     var conText=myCanvas.getContext("2d"); //获取绘图环境
   conText.beginPath(); //开始路径
  conText.arc(150, 150, 100, 0, Math.PI * 2, true);//绘制路径
  conText.closePath();//关闭路径
  conText.fillStyle = 'rgba(255,0,0,0.75)';  //设置填充样式，第
4个参数表示透明度
  conText.fill();     //填充绘图
  conText.strokeStyle = 'rgba(0,255,0,0.50)';  //设置绘图样式
，第4个参数表示透明度
  conText.lineWidth=20; //绘制边框宽度
  conText.stroke();          //绘图
</script>
</body>
</html>
```

17.2.4 Canvas 绘制线段

利用Canvas 标记可以绘制线段。常用的方法有moveTo(x,y)和lineTo(x,y)。

1．基本语法

```
context.moveto(x,y) //定义线段开始坐标，x为X轴坐标，y为Y轴坐标
context.lineto(x,y)  //定义线段结束坐标，x为X轴坐标，y为Y轴坐标
```

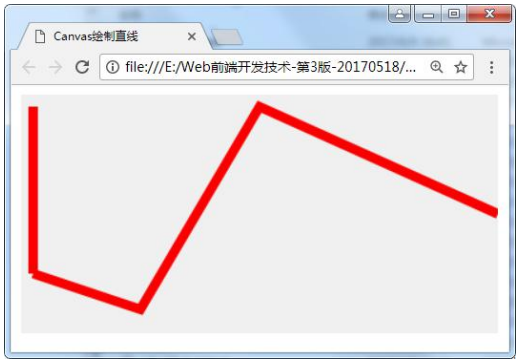
2．语法说明

两个方法的参数相同，x 为X 轴坐标，y 为Y 轴坐标。moveTo()表示设置线段的起点；lineTo()表示设置线段的终点。

【例17-2-3】用Canvas 标记绘制直线。

【例17-2-3】用Canvas 标记绘制直线

```
<!-- edu_17_2_3.html -->
<!doctype html>
<html lang="en">
<head><meta charset="UTF-8">
<title>Canvas绘制直线</title>
</head>
<body>
<canvas id="oneCanvas" width="400" height="200"
   style="background:#F0F0F0;"></canvas>
<script type="text/javascript">
var myCanvas=document.getElementById( “oneCanvas” );
var conText=myCanvas.getContext( “2d” ); //获取绘图环境
conText.strokeStyle = "rgb(250,0,0)";
conText.fillStyle = "rgb(250,0,0)"
conText.moveTo(10,10);//第1条线起点
conText.lineTo(10,150); //第1条线终点
conText.moveTo(10,150);//第2条线起点
conText.lineTo(100,180); //第2条线终点
conText.lineTo(200,10); //第3条以第2条的终点为起点-终点
conText.lineTo(400,100); //第4条以第3条的终点为起点-终点
conText.lineWidth=8;
conText.stroke();
</script>
</body>
</html>
```



17.2.5 Canvas 绘制文本

利用Canvas 除了可以绘制矩形、圆形等，还可以绘制文本。

1．基本语法

```
context.fillText(text,x,y); //在canvas上绘制实心的文本
context.strokeText(text,x,y); //在canvas上绘制空心的文本
context.font="font-style font-weight font-variant font-size/line-height
font-family “;
context.textAlign="start|end|left|right|center “; //水平对齐
ontext.textBaseline="alphabetic|top|hanging|middle|ideographic|bottom";
//垂直对齐
```

2．语法说明

context.fillText(text,x,y)：填充文本。

context.strokeText(text,x,y)：绘制文本轮廓。text表示要绘制的文本；x表示文本起点的X 坐标轴；y 表示文本起点的Y 坐标轴。

17.2.5 Canvas 绘制文本

context.font：设置字体样式。设置方法与CSS 的font 属性方法相同。

context.textAlign：设置或返回文本内容的当前对齐方式。其值可设置为start（文本在指定的位置开始）、end（文本在指定的位置结束）、left（文本左对齐）、center（文本的中心被放置在指定的位置）、right（文本右对齐）。

context.textBaseline：设置或返回在绘制文本时使用的当前文本基线（垂直对齐方式）。其值可设置为top（顶部）、hanging（悬挂，比top 略高些）、middle（中部）、alphabetic（默认，普通的字母基线）、ideographic（表意基线，与bottom 同效果）、bottom（底部）。textBaseline属性在不同的浏览器上效果不同，特别是使用“hanging”或“ideographic”时，在不同浏览器中效果不同。

```
context.textAlign="start"; //设置提示信息水平对齐方法
context.font="24px 黑体"; //设置提示信息字体
context.fillText("文本基线位置：",0,220); //设置提示信息
```

17.2.6 Canvas渐变

渐变可以填充在矩形、圆形、线条、文本等。各种形状可以自己定义不同的颜色。

1．基本语法

```
var grad=context.createLinearGradient(xstart,ystart,xend,yend);
//创建线条渐变
 grad.addColorStop(offset,color); //指定颜色停止,offset可以是0至1
 var grad=context.createRadialGradient(xstart,ystart,radiusstart,xend,
yend,radiusend);//圆径向渐变
 context.fillStyle=grad;//渐变对象变量
 context.fillRect(x,y,width,height);
```

2．语法说明

线条渐变createLinearGradient()中参数xstart 表示渐变开始点x 坐标；ystart 表示渐变开始点y 坐标；xEnd 表示渐变结束点x 坐标；yEnd 表示渐变结束点y 坐

17.2.6 Canvas渐变

标。addColorStop()中参数offset 表示设定的颜色离渐变结束点的偏移量(0～1)；color 表示绘制时要使用的颜色。

createRadialGradient()中参数有6 个，前3 个参数表示径向渐变开始圆心坐标和半径；后3 个参数表示径向渐变结束圆心坐标和半径。

- 当使用渐变对象时，必须使用两种或两种以上的停止颜色，设置制形状，如矩形、文本或一条线。fillStyle 或strokeStyle的值为渐变，然后绘

17.2.7 Canvas 绘制图像

把一幅图像放置到画布上，即在Canvas 上画出图像。

1．基本语法

```
context.drawImage(image,x,y); //在坐标(x,y)处开始绘制图像image
context.createPattern(image,type); //图像平铺
context.clip() ;//图像裁剪
var imagedata=context.getImageData(sx,sy,sw,sh); //像素处理
context.drawImage(image,x,y,width,height);//按指定宽度和高度绘图
context.drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh);
   //选取图像的部分矩形区域进行绘制
```

2．语法说明

drawImage(image,x,y)方法中参数说明如下：x 表示绘制图像的x 坐标；y 表示绘制图像的y 坐标；image 表示图像对象。

createPattern(image,type)方法中参数说明：type 取值有4 种

17.2.7 Canvas 绘制图像

，分别为no-repeat 表示不平铺、repeat-x 表示横方向平铺、repeat-y 表示纵方向平铺、repeat 表示全方向平铺。image为图像对象。

context.clip()方法只绘制封闭路径区域内的图像，不绘制路径外部图像。使用时先创建裁剪区域，再绘制图像。

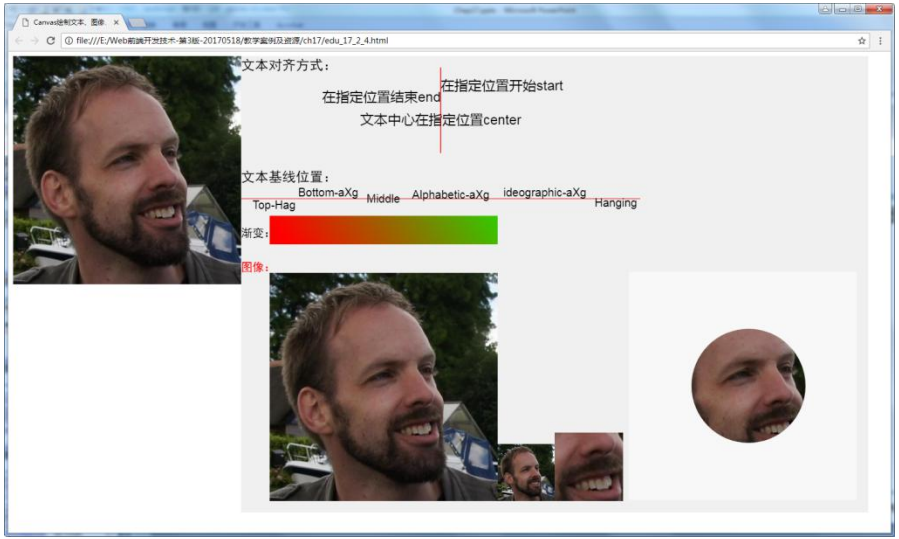
drawImage(image,x,y,width,height)方法中参数说明如下：x 表示绘制图像的x 坐标；y表示绘制图像的y 坐标；width 表示绘制图像的宽度；height 表示绘制图像的高度。

drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh)方法中参数说明如下：sx 表示图像上的x 坐标；sy 表示图像上的y 坐标；sw 表示矩形区域的宽度；sh 表示矩形区域的高度；dx 表示画在canvas 的x

17.2.7 Canvas 绘制图像

坐标；dy 表示画在canvas 的y 坐标；dw 表示画出来的宽度；dh 表示画出来的高度。

【例17-2-4】综合运用Canvas 绘制文本、图像、渐变。



代码详见：

edu_17_2_4.html

Canvas 绘制图像案例

```
<!-- edu_17_2_4.html -->
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Canvas绘制文本、图像、渐变</title>
<script type="text/javascript">
function showPage(){
var myCanvas=document.getElementById("oneCanvas");//获取Canvas对象
var conText=myCanvas.getContext("2d");  //获取绘图环境（上下文环境）对象
conText.strokeStyle = "rgb(250,0,0)";
conText.fillStyle = "rgb(0,0,0)"
// 在X轴150处绘制垂直红线
conText.textAlign="start"; //设置提示信息水平对齐方法
conText.font="24px 黑体";  //设置提示信息字体
conText.fillText("文本对齐方式：",0,24);//设置提示信息
conText.strokeStyle="red";
conText.moveTo(350,20);
conText.lineTo(350,170);
conText.stroke();
//绘制文本-textAlign属性应用
```

Canvas 绘制图像案例

```
conText.font="24px Arial";
conText.textAlign="start";
conText.fillText("在指定位置开始start",350,60);
conText.textAlign="end";
conText.fillText("在指定位置结束end",350,80);
conText.textAlign="center";
conText.fillText("文本中心在指定位置center",350,120);
conText.lineWidth=1;
conText.fill();
//在Y轴250处画一条水平红线
conText.textAlign="start";  //设置提示信息水平对齐方法
conText.font="24px 黑体";  //设置提示信息字体
conText.fillText("文本基线位置：",0,220); //设置提示信息
conText.strokeStyle="red";
conText.moveTo(0,250);
conText.lineTo(700,250);
conText.stroke();
//每个在y=250处设置不同的textbaseline值，显示单词的位置
conText.font="20px Arial";
conText.textBaseline="top";
conText.fillText("Top-Hag",20,250); //Hag表示字母组合
conText.textBaseline="bottom";
conText.fillText("Bottom-
aXg",100,250); //aXg表示字母组合
conText.textBaseline="middle";
conText.fillText("Middle",220,250);
conText.textBaseline="alphabetic";
conText.fillText("Alphabetic-
aXg",300,250); //aXg表示字母组合
conText.textBaseline="ideographic
";
conText.fillText("ideographic-
aXg",460,250); //aXg表示字母组合
conText.textBaseline="hanging";
conText.fillText("Hanging",620,250
);
```

Canvas 绘制图像案例

```
//绘制渐变
conText.font="20px 黑体";
conText.textBaseline="bottom";
conText.fillText("渐变：",0,320);
var grad=conText.createLinearGradient(50,280,400,50);//创建线条渐变
grad.addColorStop(0,"#FF0000");  //设置渐变停止颜色1
grad.addColorStop(1,"#00FF00");  //设置渐变停止颜色2
conText.fillStyle=grad;      //设置填充样式为渐变
conText.fillRect(50,280,400,50);   //填充矩形
/*绘制图像*/
var myCanvas=document.getElementById（“oneCanvas”）;//获取Canvas对象
var conText=myCanvas.getContext("2d");  //获取绘图环境（上下文环境）对象
conText.font="20px 黑体";
conText.textBaseline="bottom";
conText.fillText("图像：",0,380);
var img=new Image();
img.src="45567.jpg";
conText.drawImage(img,50,380); //在指定位置处开始绘图
conText.drawImage(img,450,680,100,100);//按指定宽度和高度绘图
/* 选取图像的部分矩形区域进行绘制 */
```

Canvas 绘制图像案例

```
conText.drawImage(img,200,200,100,100,550,660,120,120);
/* 图像圆形剪裁 */
conText.fillStyle="#F8F8F8";//填充样式
conText.fillRect(680,378,400,400);//填充
conText.beginPath();      //开始路径
conText.arc(890, 578, 100, 0, Math.PI * 2, true);  //形成圆形路径
conText.closePath();      //结束路径
conText.clip();                  //圆形剪裁
conText.drawImage(img,680,378);   //按圆形剪裁图像，其余部分不可见
}
</script>
</head>
<body onload="showPage();">
<div>
<img src="45567.jpg" id="myimg" style="float:left;"/>
<canvas id="oneCanvas" width="1100" height="800"
  style="background:#F0F0F0;"></canvas>
</div>
</body>
</html>
```

17.3 HTML5 拖放

拖放（Drag 和Drop）即抓取对象以后拖到另一个位置。任何元素都能够拖放，只要设置draggable 属性为true 即可。IE9、Firefox、Opera、Chrome 以及Safari6 等高版本的浏览器均支持拖放。

17.3.1 设置元素为可拖放

```
<标记 id= “” src= “” draggable= “true” ></标记>
```

17.3.2 拖放事件

表17-3 拖放过程发生的事件及说明

事件	事件属性	描述
dragstart	ondragstart	网页元素开始拖动时触发。
drag	ondrag	被拖动的元素在拖动过程中持续触发。
dragenter	ondragenter	被拖动的元素进入目标元素时触发，应在目标元素监听该事件。
dragleave	ondragleave	被拖动的元素离开目标元素时触发，应在目标元素监听该事件。
dragover	ondragover	被拖动元素停留在目标元素之中时持续触发，应在目标元素监听该事件。
drop	ondrop	拖动操作结束，放置元素时触发。监听器负责检索被拖动的数据以及在放置位置插入它。
dragend	ondragend	网页元素拖动结束时触发。

17.3.3 dataTransfer 对象

表17-4 dataTransfer 对象常用的属性及说明

属性	描述
dropEffect	拖放的操作类型，决定了浏览器如何显示鼠标形状，其值可为copy、move、link和none。
effectAllowed	指定所允许的操作，其值可为copy、move、link、copyLink、copyMove、linkMove、all、none和uninitialized（默认值，等同于all，即允许一切操作）。
files	包含一个FileList对象，表示拖放所涉及的文件，主要用于处理从文件系统拖入浏览器的文件。
types	储存在dataTransfer对象的数据的类型。

表17-5 dataTransfer 对象常用的方法及说明

属性	描述
setData(format, data)	在dataTransfer对象上储存数据。第一个参数format用来指定储存的数据类型，比如text、url、text/html等。
getData(format)	从dataTransfer对象取出数据。
clearData(format)	清除dataTransfer对象所储存的数据。如果指定了format参数，则只清除该格式的数据，否则清除所有数据。
setDragImage(imgElement, x, y)	指定拖动过程中显示的图像。默认情况下，许多浏览器显示一个被拖动元素的半透明版本。参数imgElement必须是一个图像元素，而不是指向图像的路径，参数x和y表示图像相对于鼠标的位置。

17.3.4 拖放操作实现步骤

拖放元素的过程可分为创建可拖放对象、设置放置对象两个步骤。

1．创建一个可拖放对象

```
<img src="45567.jpg" draggable="true" ondragstart="drag(event)"
id="drag1"/>
function drag(event) {event.dataTransfer.setData("Text",ev.target.id); }
```

说明：参数1 是数据类型，值为"Text"；参数2 是数据信息，值为可拖动元素的id ("drag1")。

2．设置放置对象

能够接受拖放元素的对象称为放置对象（或目标对象），放置对象至少要监听两个事件。

（1）dragover 事件。该事件对应的事件句柄ondragover，被拖动元素停留在放置对象之中时持续触发。默认方式下，无法将数据/元素放置到其他元素中，

17.3.4 拖放操作实现步骤

需要设置允许放置，必须阻止对元素的默认处理方式。通过给ondragover 事件属性绑定allowDrag(event)函数，在函数中使用event.preventDefault()方法来实现阻止默认处理方式。代码如下：

```
<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)">
  function allowDrop(event){
    event.preventDefault(); //阻止对元素的默认处理方式
  }
```

（2）drop 事件。该事件对应的事件句柄ondrop，允许执行真正的放置。ondrop 属性绑定drop(event)函数完成放置功能。当放置被拖数据时，会发生drop 事件。该事件将阻止对元素的默认处理方式、获得拖放元素的数据信息、添加被拖放的元素。代码如下：

17.3.4 拖放操作实现步骤

```
function drop(event){ //放置
   event.preventDefault(); //阻止对元素的默认处理方式
   var data=event.dataTransfer.getData("Text");//获取拖放数据（元素id）
   event.target.appendChild(document.getElementById(data));
   //被拖元素追加到放置元素中
}
```

以图层div 为例，把div 作为目标对象，设置图层的ondrop 和ondragover 事件属性，并绑定相关事件处理代码。代码如下：

```
<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)">
```

HTML5 拖放案例

【例17-3-1】HTML5 拖放图像应用。


```
<!-- edu_17_3_1.html -->
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>HTML5拖放图像</title>
<style type="text/css">
    #div1, #div2 {float:left; width:200px; height:200px;
          margin:15px;padding:15px;border:1px dashed #0066ff;}
     #drag1{width:200px;height:200px;}
</style>
<script type="text/javascript">
  function $(id){return document.getElementById(id);}//获取元素
  function allowDrop(ev){
    //阻止对元素的默认处理方式
     ev.preventDefault();
}
```

```
function drag(ev){
   //设置被拖数据的数据类型和值
   ev.dataTransfer.setData("Text",ev.target.id);}
function drop(ev){
    ev.preventDefault();//阻止对元素的默认处理方式
    var data=ev.dataTransfer.getData("Text");//获得被拖的数据
    ev.target.appendChild($(data));//添加拖拽元素
}
</script>
</head>
<body>
   <h3>图像在两个div中互拖放</h3><hr>
   <div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)">
      <img src="45567.jpg" draggable="true" ondragstart="drag(event)" id="drag1" />
    </div>
   <div id="div2" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
</body>
</html>
```

17.4 HTML5 Web Worker

Web Worker允许开发人员编写能够长时间运行而不被用户所中断的后台程序，去执行事务或者逻辑，并同时保证页面对用户的及时响应。

17.4.1 Web Worker 的工作原理

Web Worker 的**工作原理**是在包含JavaScript 脚本的Web 页面中，运行的JS 脚本称为主线程，并在主线程中使用Worker 类创建一个Worker，并向其传入一个参数，该参数是需要在另一个线程中运行的JavaScript 文件名称 (myWorker.js)，然后在这个实例上监听onmessage 事件。

17.4 HTML5 Web Worker

17.4.2 创建Web Worker 文件

利用JavaScript 创建一个外部Web Worker 文件myWorker.js。它是一个独立的JavaScript脚本文件。

17.4.3 创建Web Worker 对象

编辑完成Web Worker 文件后，需要利用Worker 类创建一个新的Worker 线程，并为其传入一个参数，该参数就是myWorker.js文件，从而实现调用。代码如下所示：

```
var worker = new Worker("myWorker.js"); //定义Worker，并传入参数
if (typeof(worker)=="undefined"){ //未定义，其类型为undefined
   worker=new Worker("myWorker.js "); //创建一个Worker
}
//通过postMessage() 方法将数据传递给主线程
worker.postMessage(data) //data可以是一个字符串或者JSON对象
```

17.4 HTML5 Web Worker

从Web Worker 发送和接收消息。为Web Worker 对象添加一个onmessage事件监听器来接收消息。

```
worker.onmessage=function(event){ //动态分配事件属性，绑定处理函数
    document.getElementById("result").innerHTML=event.data;
    //将接收的消息显示在指定的标记内
 }
```

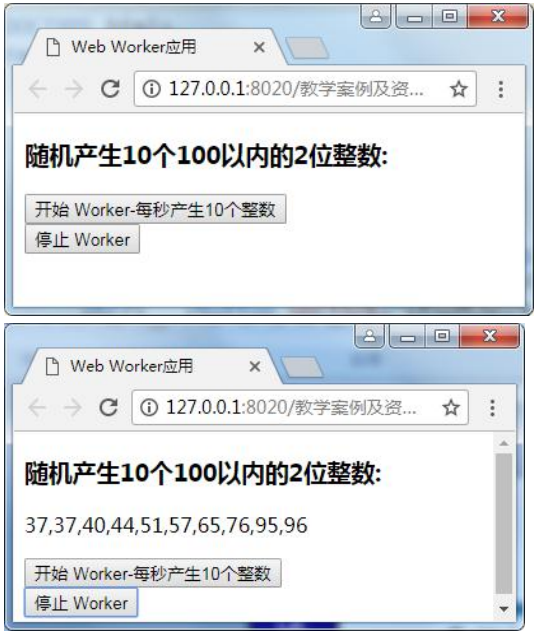
event.data 中存放来自新线程postMessage(data)方法回传的数据data。当然Worker 新线程也可以通过postMessage(data)方法来向主线程发送数据、绑定onmessage 方法来接收主线程发送过来的数据。

17.4.4 终止Web Worker

```
worker.terminate(); //终止新线程
```

HTML5 Web Worker案例

【例17-4-1】HTML5 Web Worker 多线程实现每隔1s 随机产生1 组10 个100 以内的两位整数（代码必须运行在服务器上）。




```
/*  myWorker.js
    每隔1秒随机产生10个10-99之间的整数
*/
var tenIntger=new Array();//定义保存随机2位整数
function createTenIntger(){//产生10个随机整数
  for (var j=0;j<10;j++)  //循环10次
  {  //利用数学函数随机产生10~99之间的整数，并存入数组中
     tenIntger[j]=Math.floor(Math.random()*90+10);
   }
  postMessage(tenIntger.sort());//数组元素排序后传递给主线程
  setTimeout("createTenIntger()",1000); //每隔1秒产生1次
}
createTenIntger();//调用方法,自动运行
```

HTML5 Web Worker案例代码

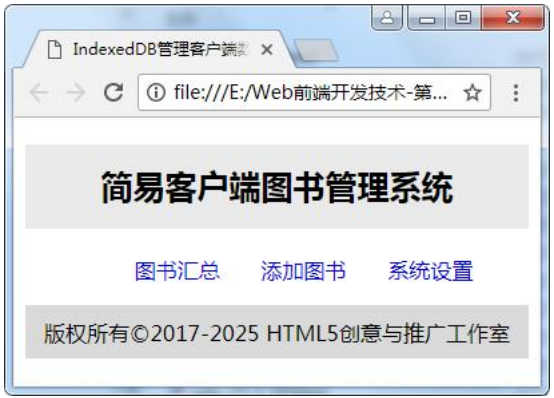
```
<!-- edu_17_4_1.html -->
<!DOCTYPE html>
<html lang="en">
<head>
<title>Web Worker应用</title>
<meta charset="UTF-8">
</head>
<body>
   <h3>随机产生10个100以内的2位整数: </h3>
    <p><output id="result"></output></p>
   <button onclick="startMyWorker()">开始 Worker-每秒产生10个整数</button>
   <br/><button onclick="stopMyWorker()">停止 Worker</button>
   <script>
     var worker;//定义全局变量
     function $(id){return document.getElementById(id);}//通过id获取对象
     function startMyWorker(){ //启动我的worker
       if(typeof(Worker)!=="undefined")//判断浏览器是否支持Web Worker
```

HTML5 Web Worker案例代码

```
{
    if(typeof(worker)=="undefined") //判断worker是否存在
    {
        worker=new Worker("myWorker.js");//不存在则创建Worker对象
    }
    worker.onmessage = function (event) {//捕获传递的消息
        $("result").innerHTML=event.data;//显示在指定的标记内
    }
}else{//浏览器不支持Web Worker
    $("result").innerHTML="对不起,您的浏览器不支持Web Worker...";
 }
}
function stopMyWorker(){
   worker.terminate();//终止线程
}
</script>
</body>
</html>
```

17.5 综合实例

利用IndexedDB 实现简易图书管理系统。页面设计功能要求如下。实现页面内容、页面表现与行为充分分离。页面采用HTML5 新增结构标记来设计，主要包括header、section、nav、footer 等标记，设计三个导航：图书汇总、添加图书、系统设置，采用三个section 分别设计三个不同的用户界面，页面效果如图所示。



(1)图书汇总页面。“检索图书”按钮功能有两个：不输入任何检索内容时，单击按钮能够实现检索对象仓库中的所有图书。

17.5 综合实例

(2)添加图书页面。“添加图书”按钮功能是将输入的图书标题、作者、ISBN等信息添加到对象仓库books 中。(3)系统设置页面。系统设置页面中设置三个命令按钮，分别是清除所有图书、清除数据库、数据库初始化。

具体代码参见edu_17_5_1.html、books.css和mybooks.js文件。

本章小结

本章介绍了HTML5 的一些需要借助于JavaScript 脚本来完成的功能，主要有客户端存储Web Storage、画布Canvas、拖放Drag & Drop、多线程Web Worker 等。通过大量的示范案例讲解了在实际开发中如何运用这些对象的方法和属性。

重点介绍了Web Storage 和IndexedDB 等客户端存储技术。其中localStorage、sessionStorage 对象可以存储少量客户端数据。而IndexedDB 数据库可以存储大量客户端数据。

HTML5 Canvas 通过JavaScript 脚本来完成绘图。canvas 标记本身并没有绘图能力，所有的绘制工作必须在JavaScript 内部完成。可以通过多种方法使用Canvas 绘制路径、盒、圆、字符以及添加图像。

本章小结

拖放（Drag 和Drop）是一种常见的特性，即抓取对象以后拖到另一个位置。任何元素都能够拖放，只要设置 draggable 属性为true 即可。

Web Workers 允许长时间运行脚本，而不阻塞脚本响应单击或者其他用户交互，它还允许执行长期任务而无须页面保持响应。