概要目录

第一部分 Web 程序设计基础

第 1 章 互联网与 Web

第 2 章 HTML 基础知识

第 3 章 层叠样式表 CSS 入门

第 4 章 网页布局技术

第二部分 服务器简单编程

第 5 章 PHP 语言入门

第 6 章 浏览器服务器交互初步

第三部分 浏览器编程基础

第 7 章 交互式网页与 JAVASCRIPT

第 8 章 DOM 概述

第 9 章 高级 WEB 客户端编程技术

第四部分 现代 Web 编程技术

第 10 章 AJAX 技术

第 11 章 WEB SERVICES 与 MASHUP

第 12 章 富客户端技术

第 13 章 WEB 安全与 WEB 工程

第 14 章 WEB 技术发展趋势

第 15 章 WEB 创业与运营

第 8 章 DOM 概述

关键知识点：

■

前端编程的核心是使用 JavaScript 语言，响应用户操作，通过网页在浏览器进程内存中的标准模型和接口 DOM，操纵浏览器中的网页对象，为网页增加丰富的动态性。其中，DOM，也就是 Document Object Model 发挥着至关重要的作用。

本章简述 DOM 的目的、其历史与发展，讲解 DOM 树、DOM 元素、与 DOM 紧密相关的浏览器对象模型 BOM、DOM 事件，解析这些可编程操作的要素的原理、设计和具体用法。

8.1 案例：登山俱乐部—在线画册和重构

上一章提出了登山山俱乐部 Web 应用的一些需求，包括：收放（隐藏/显示）新闻内容、计算器和在线画册。这些需求都可以使用前端技术解决，基于基本的 JavaScript 语法和最简单的 document.getElementById DOM 方法，上一章已经初步实现了收放（隐藏/显示）新闻内容和计算器两项需求，参见第 7 章案例研究。本章将使用更多 DOM 方法，重构改进现有的代码实现，使其更加简洁、高效，可扩展性、可维护性更好。同时，本章还将实现在线画册，展示登山俱乐部的名山靓照。

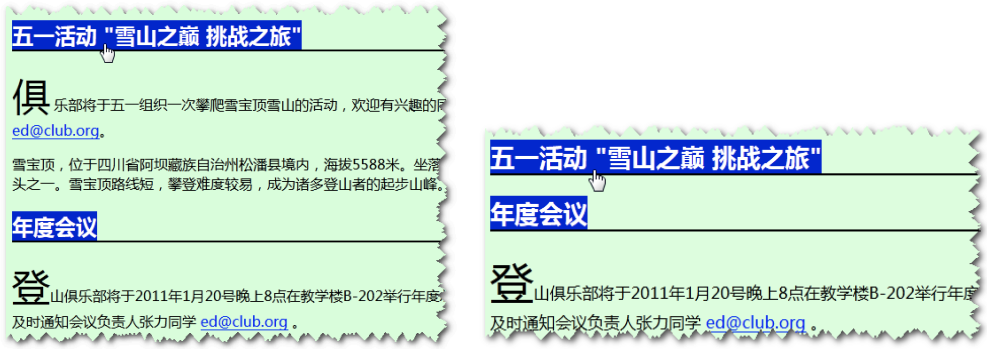

图 8-1 点击标题收起/放开新闻内容功能示意图

8.2 DOM 概述

浏览器从 Web 服务器获得网页代码，包括 HTML、CSS 和 JavaScript 代码，然后处理显示为浏览器窗口中的网页。在浏览器的软件架构和设计上，虽然还可以使用其它类型的模块分工与设计，但是迄今为止，大都将其工作分为获取、解析、加载、渲染几个阶段[1]。在这几个阶段中，浏览器根据解析出的 HTML 元素，在进程空间里创建对象，并把它们关联在一起的过程，称为加载。加载完成之后，网页不再是代码，而是存活在其浏览器进程内存空间里的一组对象，浏览器会根据这组对象渲染网页，也就是根据这组对象的类型、属性和互相之间关系，确定它们在浏览器窗口

[1] 现在绝大多数的浏览器，历史上都是同源的。这些浏览器代码大都可以追溯到第一款图形界面浏览器——Mosaic，参考 sidebar “浏览器战争”。

中出现的位置、大小、外形，并将它们绘制出来。

图 8-2 浏览器网页获取、解析、加载和渲染示意图

显然，网页在浏览器进程空间里的对象，其设计对浏览器至关重要。早期，各浏览器厂商，独立设计网页对象模型，供自己的浏览器使用（参考本节 sidebar：浏览器战争）。随着 Web 的普及、动态 HTML（Dynamic HTML，参考第 7 章）、前端动态性的重要意义日益彰显，各种浏览器纷纷通过脚本语言，开放了自己的网页对象模型接口，便于编写动态 HTML，实现前端动态性。

然而，不同厂商的网页对象模型不同，这给网页作者和 Web 应用开发者带来了极大的麻烦和困扰[2]。因此，W3C 开始了网页对象模型的标准化工作，逐步制定了 DOM（Document Object Model）标准[3]，规范和指导浏览器的设计与实现，提供统一的 API，为动态 HTML 和 Web 应用开发解决浏览器兼容性问题。

根据 W3C 的定义[4]：“DOM 是与**平台无关**、与**语言无关**的**接口**，它允许程序

[2] 参见 Wikipedia DOM 条目，Legacy DOM 小节：http://en.wikipedia.org/wiki/Document_Object_Model#Legacy_DOM

[3] http://www.w3.org/DOM/

[4] W3C 原文：The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page.

和脚本动态访问和更改文档（网页）的内容、结构和外观，更改后的结果将和当前的页面（网页）其它内容一同呈现。”

DOM 到目前为止有 3 个层次，由一系列标准推荐案（recommendations）[5]定义了程序和脚本动态访问和更改网页的一系列 API。DOM Level 2，常简称 DOM2、或者 DOM，是目前主流浏览器、动态 HTML 和 Web 应用共同遵从的标准[6]。

浏览器战争

Web 20 多年的快速发展历史中，浏览器的市场竞争是一个重要的现象，这些竞争激烈且影响巨大，被人们称为浏览器战争。迄今为止，有 3 次浏览器战争。

Mosaic 战争，90 年代初期，多家软件公司从国家超级计算机应用中心（NCSA）获得了第一款图形界面浏览器——Mosaic 的使用版权，在此基础上开发自己的浏览器产品，争夺新兴的 Web 浏览器市场，最后网景公司的 Netscape Navigator 浏览器胜出。

第一次战争，1995 年 12 月 7 日，微软公司在珍珠港事件 54 周年这一天，在西雅图总部会议中心宣示要打垮网景。当时，Netscape Navigator 浏览器的市场份额高达 90%以上，比尔盖兹将微软形容成遭日本袭击的美国舰队。随后，微软发布了 Internet Explorer（IE）通过操作系统捆绑等市场手段，实现了自己的目标。1998 年底，网景被微软击败，被美国在线（AOL）收购。IE 浏览器的市场份额一度高达 96%（2002）。这场浏览器战争的一个重要后果就是大家意识到，Web 标准对整个 Web 的至关重要，一系列标准由此开始制定，DOM 就是这一历史事件的结果之一。

第二次战争，第一次战争中失败的网景公司并没有真正认输，在其被 AOL 收购前夕，网景公司宣布开放了网景浏览器源代码，从此开始了另一种形式的浏览器战争。基于网景浏览器源代码，Mozilla 基金会的 Firefox 迅速成长，特别在 IE 浏览器安全问题不断的情况下，吸引了大量用户。随后，Opera、Safari、Google Chrome 等也都纷纷问世，互联网第二次浏览器战争正在进行，远未结束……

更多情况，请参考：http://en.wikipedia.org/wiki/Browser_wars

8.2.1 DOM 基本情况

简单地说，网页的 DOM 是一棵树，DOM 树，参考图 8-3。

[5] http://www.w3.org/DOM/DOMTR

[6] W3C 浏览器 DOM 兼容性测试：http://www.w3.org/2003/02/06-dom-support.html，检查浏览器对各层次 DOM 的支持。

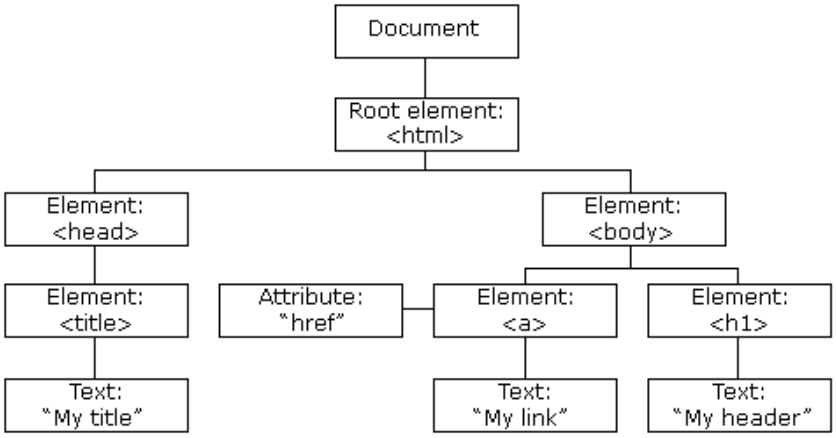

图 8-3 DOM 树示意图

从图 8-3 看到，HTML DOM 树由一系列节点（node）构成，这些节点是浏览器解析加载网页代码得到的内存对象的接口，节点类型有文档（document）、元素（Element）、文本（Text）、属性（Attribute）和注释（Comment）。DOM 文档对象 document 是 DOM 树的根，是向程序和脚本开放的全局对象。DOM 通过它提供了对所有节点的访问和丰富的功能。程序和脚本可以遍历 DOM 树，访问、获取 DOM 树上的各种类型的节点（元素、属性），读取、更改这些节点的值，还可以动态删除或者插入节点、子树。程序和脚本对 DOM 树的更改会被浏览器及时渲染，更新到浏览器窗口中。

同时，DOM 的事件（event）机制十分重要，它将操作系统传递给浏览器的用户事件，翻译成对应的 DOM 事件后，沿着 DOM 树传播到相应的节点（元素）。为 DOM 元素添加特定 DOM 事件的事件处理器（event handler），也称为事件监听器（event listener），这些元素就有了响应用户操作的能力，能够在用户做出该操作时，执行相应事件处理器，响应操作，更新网页。

上一章的众多例子，包括：事件驱动、加法器、收放新闻、计算器等等，已经使用了 DOM。这些 JavaScript 源代码中，document 是 DOM 全局对象，其 getElementById 方法能够获取网页上相应 Id 的元素，因而被广泛使用。onclick 是许多 DOM 元素的属性，指向 DOM 事件 click 的事件处理器。此外，style、textContent 等也是常用的 DOM 元素属性。

本章将进一步以 DOM Level 2 为准，全面介绍 DOM 树、DOM 元素、与 DOM 紧密相关的浏览器对象模型 BOM、DOM 事件等 API。

自测题

1. DOM 是什么，它对于前端编程有什么作用？
2. 目前主流浏览器支持的 DOM 标准是什么 Level，此外还有哪些 Level 的 DOM？
3. 从数据结构的角度来看，DOM 是什么，它的结构如何？

8.3 DOM 元素

DOM 是网页在浏览器进程空间中对象的开放接口。网页上每一个 HTML 元素经过浏览器解析，加载后都成为了浏览器进程空间中的对象，也就是 DOM 元素。HTML 元素的属性，解析加载之后成为了 DOM 元素的属性（attribute）节点，可以通过 DOM 元素的 getAttribute 和 setAttribute 方法来使用。DOM 元素还有其它丰富的属性（property）和方法，便于程序和脚本对它的操作。图 8-4 是 HTML 元素加载为 DOM 元素的示意图。

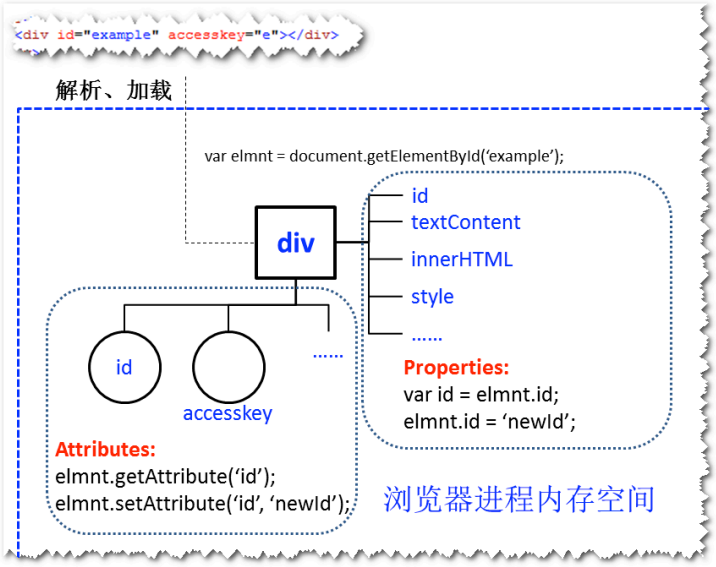

图 8-4 HTML 元素加载为 DOM 元素示意图

Attributes 和 Properties：DOM 元素有两种不同的属性 attribute 和 property。这两个英文单词通常都翻译为属性，但是对于 DOM 元素来说，二者大不相同。

从图 8-4 中可以看出，写在 HTML 内，也就是 HTML 标签内声明的是 attribute。它们被加载为 DOM 中的属性（attribute）类型节点，可以通过 DOM 元素的 getAttribute 和 setAttribute 方法进行访问和更改。另一面，property 是 DOM 固有的属性，由 DOM 标准定义，很多并不是 HTML 标签内声明的内容，而是浏览器在解析和加载网页时，计算生成的。例如：textContent，存储了当前元素标签之间的文本内容。

值得注意的是，部分 property 和 attribute 实际上是相同的，如图 8-4 所示 id。**在本节后面将直接使用英文单词 attribute 或者 property，以避免混淆。**

JavaScript 使用 DOM 元素时，很多时候都是访问和更改其 attributes 或者 properties。其中，最为常见的操作就是操纵 DOM 元素的文本，调整其外观，从而改变它在浏览器中显示的内容和模样。

8.3.1 操纵文本

最终显示在网页上的内容，大部分是文本，因此，读取、操作和更新这些文本内容是前端程序的一大任务。DOM 元素提供了两个 properites，innerHTML 和 textContent[7]，用于操作 HTML 元素内的文本。innerHTML 返回的是包括了子元素 HTML 标签在内的所有文本，即 HTML 源代码中在元素开始和结束标签之间的所有文本。textContent 返回的则是去除了 HTML 标签后的文本，参考源代码 8-1，注意代码注释。

源代码 8-1 innerHTML 与 textContent 对比

```
------------------------- HTML -----------------------------
……
<div id="example" accesskey="e">
    <p>文本内容1</p><p>文本内容2</p>
</div>
<br/>
<button onclick="changTextContent();"> 改变textContent</button>
<br />
<button onclick="changeInnerHTML();">改变innerHTML</button>
……
---------------------- JavaScript --------------------------
window.onload = function(){
  var div_example = document.getElementById("example");
  alert(div_example.textContent); //文本内容1文本内容2
  alert(div_example.innerHTML); //<p>文本内容1</p><p>文本内容2</p>

  changTextContent = function(){
```

[7] IE7 之前，不支持 textContent，而使用类似的 innerText。

```
    div_example.textContent = "<p>新文本内容</p>"
  }

  changeInnerHTML = function(){
    div_example.innerHTML = "<p>新文本内容</p>"
  }
}
```

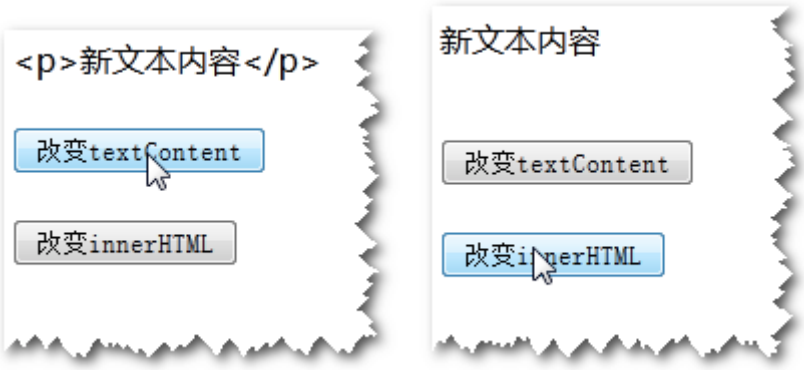

图 8-5 源代码 8-1 运行效果示意图（左：改变 textContent，右：改变 innerHTML）

对 DOM 元素的 textConent、或者 innerHTML property 进行赋值，会即刻反映到浏览器窗口内显示的网页上。对 textContent 的赋值，被浏览器解读为纯文本，如图 8-5 左，其中的 HTML 标签被视为的网页上的文字，经过转义后输出，在网页上显示。对 innerHTML 的赋值，被浏览器作为 HTML 代码加载渲染，其中的 HTML 标签被视为 HTML 元素，如图 8-5 右，浏览器动态解析和加载这些 HTML 元素，生成相应的 DOM 元素附着在当前 DOM 元素之下。这里实际上插入了一个新的 DOM 元素，由 HTML 的 p 元素生成，其 textContent 为“新文本内容”，参见源代码 8-1。

在第 7 章的案例中，计算器使用弹出框提醒用户输入的表达式有错误，这样的方式有些突兀，用户体验不好。源代码 8-2 做了改进，将警告信息直接在计算器输入框上方显示。

源代码 8-2 改进计算器错误信息显示方式

```
------------------------- HTML -----------------------------
……
<div id = "calculator">
    <div id = "error"></div>
    <div id="output_field">
……
------------------------- CSS ------------------------------
……
div#error{color:red;}
……
---------------------- JavaScript --------------------------
```

```
……
function showError(message){
    document.getElementById("error").textContent = message;
}
……
```

图 8-6 源代码 8-2 运行效果示意图

value 属性与网页内容：对于一些由 value 属性定义其显示内容的 HTML 元素，例如：**文本框，各种按钮等，不能使用 textContent 和 innerHTML**，而应该通过其 DOM 元素的 value property 来获取/改变其显示内容。例如，计算器例程中的表达式和计算结果的显示，就是通过给 output_field 文本框的 value 属性赋值得以实现的，参见源代码 8-3。

源代码 8-3 文本框使用 value property 示例

```
------------------------- HTML ------------------------------
……
<div id = "calculator">
    <div id = "error"></div>
    <div id="output_field">
……
----------------------- JavaScript --------------------------
……
function updateOutput(data){
  document.getElementById("output").value = data;
}
```

……

8.3.2 调整外观

除了更改网页文字内容，调整改变网页元素的外观，是响应用户操作的另一种重要方式。通过 DOM 元素的 style property 可以获取/更新其外观。style property 的值不是简单的字符串，而是一个对象。此对象拥有丰富的 property，分别对应网页元素的 CSS 属性，参见表 8-1。

表 8-1 部分 DOM style property 名与 CSS 属性名对照表

DOM style property 名	CSS 属性名	值示例
backgroundColor	background-color	“#ff00dd”
border	border	“1px solid red”
color	color	“red”
cssFloat	float	“left”
fontWeight	font-weight	“bold”
fontSize	font-size	“12pt”
zIndex	z-index	“456”

DOM style property 名与 CSS 属性名不同： DOM style property 的名称使用驼峰式大小写（CamelCase），而 CSS 属性名使用短横线分隔多个英文单词。此外，float 是 JavaScript 的保留字，因此 CSS 的 float 属性，在 DOM style property 里面用 cssFloat 表示。

对 DOM 元素 style property 的更新，相当于为元素设定了相应的 CSS 属性，进而立即影响到网页元素在浏览器窗口里的外观。图 8-7 的当中，用户点击“加下划线”按钮后，按钮相应的事件处理器代码（参见源代码 8-4），使用 DOM style 的 textDecoration property 将网页文字加上了下划线。

图 8-7 源代码 8-4 运行效果示意图（左后：点击前，右前：点击后）

源代码 8-4 使用 DOM style property 改变网页外观示例

```
------------------------- HTML -----------------------------
……
<div id="poem">
    <h1>将进酒</h1>
    <h2>李白</h2>
    <p>
        君不见黄河之水天上来，奔流到海不复还。 <br/>
        君不见高堂明镜悲白发，朝如青丝暮成雪。 <br/>
        ......<br/>
        与尔同消万古愁。
    </p>
    <button id="changeButton">加下划线</button>
</div>
……
---------------------- JavaScript --------------------------
……
document.getElementById("changeButton").onclick = function(){
    document.getElementById("poem").style.textDecoration =
'underline';
}
……
```

DOM style property 是字符串类型：DOM style property 中，类似 fontSize、borderWidth、padding 等属性都是有单位（em、pt、px）的字符串，而不是数值。误将数值直接赋值给这些 properties，是使用中常见的错误，参考源代码 8-5。

源代码 8-5 style property 是字符串，错误和正确的赋值方式示例

```
------------------------- 错误做法 ----------------------------
document.getElementById("poem").style.fontSize = 24;

document.getElementById("poem").style.fontSize += 5;

------------------------- 正确做法 ----------------------------
document.getElementById("poem").style.fontSize = "24px";

var poem = document.getElementById("poem");
var fontSize = parseInt(poem.style.fontSize);
poem.style = (fontSize + 5) + "px"; // 假定使用px作为单位
```

程序设计与 API

进行程序设计离不开使用各种类型的 API，程序员是否需要牢记 API 中具体的方法、属性、参数和用法呢？

现代各种语言、框架、类库的 API 越来越庞大，内容庞杂，并且更新迅速。C 语言不到 40 个内置函数，而 PHP 则有超过 700 个，DOM 也有几百个方法和属性。显然，记忆这些内容，对程序员是极大的负担，并且越来越不现实。那么，现代程序员应该如何应对呢？

笔者浅见，主要的策略有：

明白语言、框架、类库的意图、原理与设计，知道它能够做什么，不能做什么。

善用搜索工具，在需要时，搜索 API 和其用法。

善用现代集成开发环境（IDE），正确配置好即时代码助手（content assistant），给出 API 提示。

8.3.3 非侵入式 CSS

3.1.1 节提到作为非侵入式 CSS（Unobtrasive CSS） 要求将 HTML 与 CSS 分离。事实上，非侵入式 CSS 还要求将 CSS 与 JavaScript 分离，不要在 JavaScript 的代码中直接描述元素的外观样式，也就是说，不能在 JavaScript 中，通过直接更改 DOM 元素的 style property，来更改网页外观。

源代码 8-4 和源代码 8-5 的做法都不符合非侵入式 CSS 要求。正确的做法是，将网页外观的定义，全部写入 CSS 当中，并为需要应用的外观给定恰当的 class 名称，然后通过 JavaScript 操纵 DOM 元素，将此 class 名称加入到 DOM 元素的 className，从而应用外观。DOM 元素的 className property，其值和其 class attribute 一样，也就是 HTML 的 class 属性的值。按照非侵入式 CSS 的要求，源代码 8-4 应该改为源代码 8-6 的方式。

源代码 8-6 非侵入式 CSS 与使用 DOM 元素的示例 1

```
------------------------- CSS  -----------------------------
.changed { text-decoration: underline;}
---------------------- JavaScript  --------------------------
……
document.getElementById("changeButton").onclick = function(){
    document.getElementById('poem').className = 'changed';
}……
```

但是，一个元素可能已经有了某些 class 名称[8]。源代码 8-6 直接对 className 的赋值，很有可能会覆盖掉已有的 class，导致元素某些外观样式的丢失。因此，更好的做法应该是将新的 class 名称附加到 className，并保证其没有重复出现，如源代码 8-7。

源代码 8-7 非侵入式 CSS 与使用 DOM 元素的示例 2

```
------------------------- CSS  -----------------------------
.changed { text-decoration: underline;}
---------------------- JavaScript  --------------------------
……
document.getElementById("changeButton").onclick = function(){
    var poem = document.getElementById('poem');
    var className = poem.className;
    if(className.indexOf('changed') < 0) poem.className += '
changed';
}……
```

源代码 8-6 的写法不会覆盖掉原有的 class 名称，安全许多，但是较为繁琐，第 9 章当中将介绍更加精炼的方式。

8.4 DOM 树

在响应用户操作，调整改变网页的过程中，除了对网页内容、网页外观的调整，调整网页的结构，动态删除或者插入一些 DOM 元素，显然也十分重要。本节讲述 DOM 树的基本结构，如何遍历、导航，获取 DOM 树上需要操作的节点（元素），以及如何添加/删除节点（元素）。

8.4.1 DOM 树与 HTML 文档

DOM 树是浏览器在解析、加载网页代码之后，在浏览器进程内存空间里面形成的数据结构。网页代码中，HTML 文档表达了网页的结构，其逻辑组织是一颗树。DOM 树则是这棵树的运行时表示。浏览器在加载网页时，按照 HTML 文档中的元素嵌套关系和出现先后次序，生成 DOM 树。因此，在结构上二者完全相同，每个 HTML 文档当中的元素都成为了 DOM 树上的节点（node），也就是 DOM 元素（element）。例如：源代码 8-8 中的 HTML 文档，加载之后成为如图 8-8 所示 DOM 树。

源代码 8-8 HTML 文档运行时加载成为 DOM 树

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
```

[8] 参考 3.2.2 节，HTML 元素的 class 属性是一个字符串，字符串可以包括由空格分隔的多个 class 名称。此时，HTML 元素同时从属于这些 class。

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>DOM树</title>
</head>
<body>
    <h1>DOM树</h1>
    <p>DOM树是网页内存对象模型，参考<a
href="http://www.w3.org/DOM/">W3C DOM</a>。</p>
    <ul>
        <li>DOM节点</li>
        <li>导航与遍历</li>
        <li>选择元素</li>
    </ul>
</body>
</html>
```

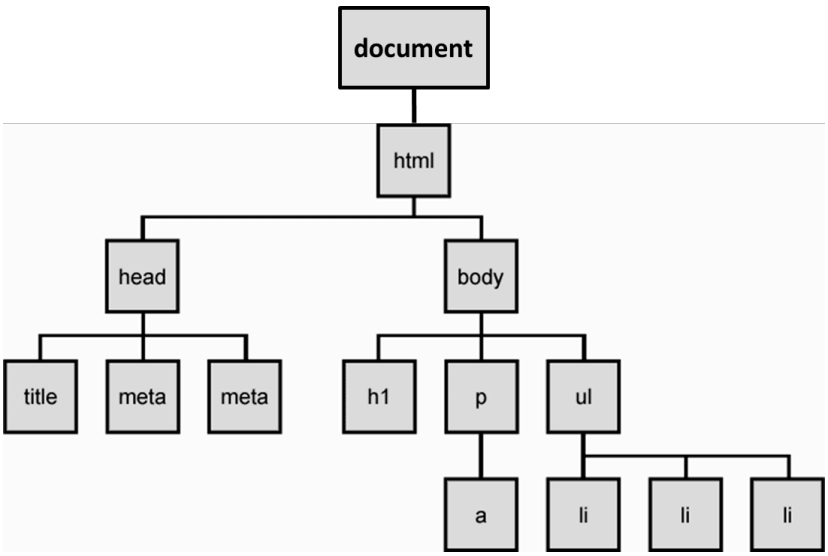

图 8-8 HTML 文档运行时加载成为 DOM 树

整个 DOM 树的根是其访问的入口，document 对象，该对象被浏览器直接公开提供给 JavaScript 程序使用，它有丰富的方法和 properties 用于对 DOM 树的操作，详细的 document 对象介绍，将在 8.5.2 节进行。

8.4.2 DOM 节点

如同其它树形结构一样，DOM 树上的每一个数据（对象）称为 DOM 节点（DOM node），一棵树只有一个根节点（root），除了根节点，每个节点有且只有一个父节点（parent node）。除了最底层的叶子节点（leaf node），每个节点都有至少一个或者多个子节点（child nodes）。从根节点出发到任意一个节点，有且只有一条路径（path）。

a) DOM 节点类型

浏览器解析、加载网页时，不仅仅是 HTML 元素，而是所有 HTML 文档内的内容，都被加载成为不同类型的 DOM 节点：

- 整个文档被被加载为 DOM 树的根节点，document
- 每个 HTML 元素都被加载为 DOM 元素，如源代码 8-8 加载成为图 8-8 所示 DOM
- 所有文本被加载成为 DOM 的文本节点，图 8-8 省略未显示
- 所有 HTML 元素的属性被加载成为 DOM 的属性节点，图 8-8 省略未显示，参考图 8-4
- HTML 注释被加载成为 DOM 的注释节点*

*注释节点因为其不显示，也不是为 Web 用户所作，通常不在前端编程考虑的范畴之内。

为了简洁，图 8-8 省略了源代码 8-8 加载形成的 DOM 树上的文本、属性节点。事实上对于源代码 8-8 中的文本和属性，加载形成 DOM 树上一定有相应的节点。例如：源代码 8-9 是源代码 8-8 的部分，其加载后的所有 DOM 节点如图 8-9 所示。其中直角矩形代表 DOM 元素节点，圆角矩形代表 DOM 文本节点，圆圈代表 DOM 属性节点。

源代码 8-9 HTML 文档运行时加载成为 DOM 树（包括属性、文本节点）

```
……
<p>DOM树是网页内存对象模型，参考<a href="http://www.w3.org/DOM/">W3C DOM</a>。</p>
……
```

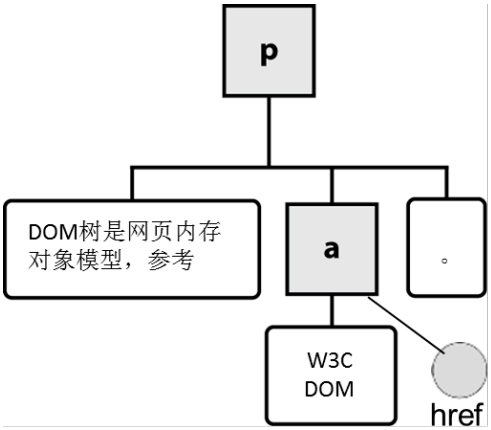

图 8-9 HTML 文档运行时加载成为 DOM 树（包括属性、文本节点）

b) DOM 节点的特点

DOM 节点有以下特点：

- 元素节点表示 HTML 元素，可以有子节点，子节点可以是元素、文本和属性节点
- 文本节点表示 HTML 标签之间的文本，只能够是 DOM 树的叶子节点，不可以有子节点
- 属性节点表示 HTML 标签内的属性定义，同样只能够是 DOM 树的叶子节点，不可以有子节点

请注意所有 HTML 标签之间的文本都会被解析为文本节点，例如：源代码 8-9 中，链接之后的句号“。”，加载成为了一个文本节点如图 8-9。

HTML 文档中的空白也会被加载到 DOM 文本节点：在 HTML 标签之间有空白时，这些空白也会被解析为文本节点！虽然最后浏览器会在渲染时塌缩空白，但是在加载的 DOM 中所有的空白都会原封不动的保存在 DOM 的文本节点上。例如，源代码 8-10 中链接结束标签和段落结束标签</p>之间只有换行空白，并没有任何其它字符；即便如此，生成的 DOM 树上，链接元素 a 之后，还是会有一个文本节点，其值为换行符。这种空白文本节点，在对 DOM 树进行遍历与导航时，常常会被忽视，是初学者常见的错误。

但是，CSS 当中的+，所指的下一个 sibling，不包括文本节点。

源代码 8-10 HTML 文档运行时加载成为 DOM 树（空白内容保留在文本节点内）

```
……
<p>DOM树是网页内存对象模型，参考<a href="http://www.w3.org/DOM/">W3C
DOM</a>
</p>
……
```

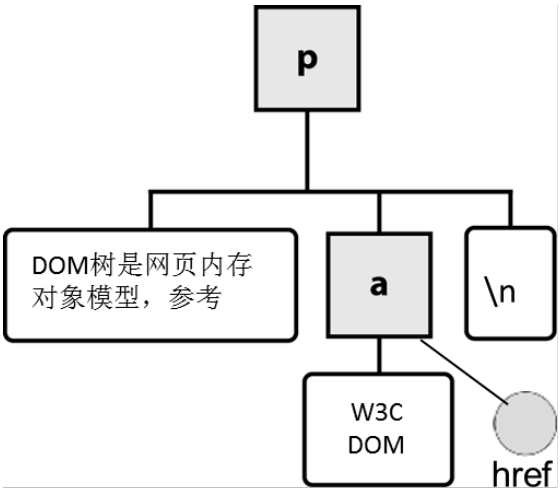

图 8-10 HTML 文档运行时加载成为 DOM 树（空白内容保留在文本节点内）

c) DOM 节点 properties

所有 DOM 节点，不论其类型都有以下 properties。

表 8-2 DOM 节点 properties

property 名	描述
nodeName	节点名（只读），元素节点名为 HTML 标签名（全大写），属性节点名为 HTML 属性名，文本节点名始终为”#text”，文档节点名为”#document”
nodeValue	节点值，元素节点值为 null，属性节点值为 HTML 属性值，文本节点值为文本内容
nodeType	节点类型（只读），元素节点为 1，属性节点为 2，文本节点为 3，文档节点为 9，注释节点为 8
parentNode	父节点
previousSibling	前一个兄弟节点
nextSibling	后一个兄弟节点
firstChild	第一个子节点

lastChild	最后一个子节点
childNodes	所有子节点的数组

通过 nodeName、nodeValue、nodeType，可以访问 DOM 节点的节点名、节点值和节点类型，参考源代码 8-11。

源代码 8-11 DOM 节点 properties 示例

```
------------------------- HTML -----------------------------
<li id="dom-node">DOM节点</li>
......
---------------------- JavaScript --------------------------
var node = document.getElementById("dom-node");
alert(node.nodeName);    // LI
alert(node.nodeType);    // 1
alert(node.nodeValue);   // undefined

var child = node.firstChild;
alert(child.nodeName);   // #text
alert(child.nodeType);   // 3
alert(child.nodeValue);  // DOM节点
```

8.4.3 遍历与导航

对于数这种数据结构，常用的操作就是遍历。DOM 树也不例外，在前端编程中常常会从一个 DOM 节点出发，遍历其子树，或者导航到其它节点。DOM 节点提供了一系列 properties 用于遍历和导航，表 8-2 的后六行展示了这些 properties，它们分别指向与当前 DOM 节点的对应的 DOM 节点。例如，源代码 8-9 运行时生成的 DOM 树(如图 8-9)，其遍历和导航关系，如图 8-11 所示。

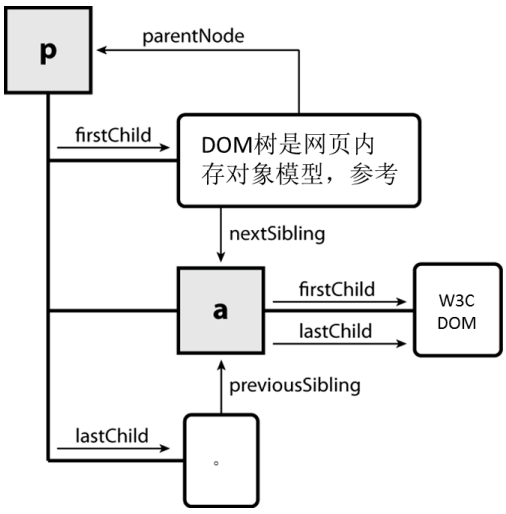

图 8-11 源代码 8-9 DOM 树导航关系

JavaScript 程序可以使用 DOM 节点的遍历与导航 properties，得到需要的节点，从 DOM 的 p 元素节点出发，用两种不同的方式，得到了链接中的文本节点。

源代码 8-12 使用 DOM 节点 properties 进行 DOM 树遍历与导航示例

```
// 得到DOM p节点，参考8.4.4节
var p = document.getElementsByTagName('p')[0];

var textNode1 = p.firstChild.nextSibling.firstChild;
var textNode2 = p.childNodes[1].childNodes[0];
alert(textNode1.nodeValue);         // W3C DOM
alert(textNode2.nodeValue);         // W3C DOM
alert(textNode1 === textNode2);     // true

alert(textNode1.firstChild);            // null
alert(textNode1.lastChild);             // null
alert(textNode1.nextSibling);           // null
alert(textNode1.previousSibling);       // null
alert(textNode1.childNodes.length);     // 0
```

在使用 DOM 节点的遍历与导航 properties 的时候，如果某个 property 指向的节点不存在，则其值为 null。例如源代码 8-12 中 textNode1 指向了内容为“W3C DOM”的文本节点，此节点为叶子节点，且没有兄弟。所以无论是其 firstChild、lastChild，还是 nextSilbling、previousSibling，都是 null，而其 childNodes 则是空数组，参考源代码 8-12 后半部分。

attribute 节点不参与遍历与导航：DOM 节点的遍历与导航 properties 指向的节点不包括 attribute 节点。例如，源代码 8-9 生成的 DOM 树如图 8-9，DOM a 元素，有 attribute href 节点。但是，其 childNodes 指向的节点中并不包括这个 attribute 节点，参见图 8-11。

attribute 节点本身不会有子节点（参见 8.4.2b)节）。因此，将其排除到导航节点之外，不会影响对 DOM 树所有元素按图索骥地遍历、导航。

注意 HTML 空白形成的文本节点：使用 DOM 节点的遍历与导航 properties 时，容易犯的错误是忽略了 HTML 文档中换行和缩进造成的空白文本节点，参考 8.4.2b)中的注意。例如：源代码 8-13 中的 HTML 文档片段，形成的 DOM div 元素（节点），其子节点不是 1 个，而是 3 个，包括 div 开始标签和 p 开始标签间空白形成的文本节点，p 元素节点，和 p 结束标签与 div 结束标签之间空白形成的文本节点，参见源代码 8-13 中的 JavaScript 代码。

源代码 8-13 HTML 空白形成的文本节点示例

```
------------------------- HTML -----------------------------
……
<div id="example">
    <p>DOM树是网页内存对象模型，参考W3C DOM</p>
</div>
……
---------------------- JavaScript --------------------------
var childNodes = document.getElementById("example").childNodes;
alert(childNodes.length);       // 3
alert(childNodes[0].nodeName);  // #text
alert(childNodes[1].nodeName);  // P
alert(childNodes[2].nodeName);  // #text
```

8.4.4 选择节点

使用 DOM 节点的遍历与导航 properties，从 DOM 树暴露给 JavaScript 的公开对象——DOM 树的根 document 对象出发，原则上可以访问到 DOM 树上所有的节点。然而，这样的方法显然比较麻烦，不利于程序员理解，易于出错。参考源代码 8-12 中不论是获取 textNode1 的方式，还是获取 textNode2 的方式，程序员不对照 HTML 仔细查看，不画出一个类似图 8-11 的导航关系图，可能很难正确理解。

因此，DOM 为 document 对象设计了几个从 DOM 树上选取相应节点（元素）的方法，方便程序员获取需要操纵的 DOM 元素。

表 8-3 document 对象的 DOM 元素选择方法

方法名	描述
getElementById	返回 id property/attribute 为给定值的 DOM 元素
getElementsByTagName	返回给定标签名对应的所有 DOM 元素
getElmentsByName	返回具有给定 name attribute 的所有 DOM 元素

表 8-3 中的方法比 DOM 节点的遍历与导航 properties，显然更易于理解和使用。其中的 document.getElementById 已经多次在前面的例程里出现，更几乎是所有前端程序员学习 DOM，第一个需要掌握的方法。

其余两个方法的使用，也能够给前端编程带来很大便利。例如，在图 8-12 所示的 HTML 文档渲染而成的网页上，希望为增加网页行为，响应用户操作，即当用户选择某个单选按钮时，就将诗歌内容改为相应颜色。此时，使用 document DOM 元素选择方法（源代码 8-14 中“使用 document 方法”部分），显然比使用 DOM 节点的遍历与导航 properties（源代码 8-14 中“使用 properties”部分），代码更加简洁易懂。

图 8-12 源代码 8-14 运行效果

源代码 8-14 使用 document 对象的方法选择 DOM 元素示例

```
------------------------- HTML -----------------------------
……
<body>
<div>
    <h1>将进酒</h1>
    <h2>李白</h2>
    <p>
        君不见黄河之水天上来，奔流到海不复还。 <br/>
        ......
    </p>
    <form action="#">
        请选择文本颜色：
        <label><input type="radio" name="color" value="red"/>红
</label>
        <label><input type="radio" name="color" value="blue"/>蓝
</label>
        <label><input type="radio" name="color" value="black"/>
黑</label><br />
    </form>
……
---------------------- JavaScript --------------------------
window.onload = function(){
  var radios = getAllRadios();
  for (var i = radios.length - 1; i >= 0; i--) {
    radios[i].onclick = changeColor;
  }
}
---------------------- 使用properties --------------------------
function getAllRadios(){
```

```
    // document.body就是HTML body元素，参考图 8-4和8.5.2a)节
    // 难以理解childNodes[1].childNodes[7]到底是什么
    var form = document.body.childNodes[1].childNodes[7];
    var radios = [];
    // 繁复的筛选
    var nodes = form.childNodes;
    for(var i = 0; i < nodes.length; i++)  {
        if(nodes[i].nodeName != 'LABEL') continue;
        radions.push(nodes[i].firstChild);
    }
    return radios;
}

function changeColor(event){
    // 难以理解childNodes[1].childNodes[5]到底是什么
    var paragraph = document.body.childNodes[1].childNodes[5];
    // event.target指向用户点击的元素，参考8.6.1节
    paragraph.style.color = event.target.getAttribute('value');
}

---------------------- 使用document方法 -------------------------
function getAllRadios(){
    return document.getElementsByName("color");
}

function changeColor(event){
    var paragraph = document.getElementsByTagName('p')[0];
    paragraph.style.color = event.target.getAttribute('value');
}
```

8.4.5 创建、添加和删除节点

documet.createElement 方法用来创建元素节点，document.createTextNode 用来创建文本节点，而属性节点则通过 DOM 元素的 setAttribute 方法（参考 8.3 节）来创建并添加到 DOM 元素，参考源代码 8-15。

源代码 8-15 创建新 DOM 节点

```
var newParagraph = document.createElement("p");
alert(newParagraph.nodeName);       // P
var newText = document.createTextNode("君不见高堂明镜悲白发");
alert(newText.nodeName);            // #text
alert(newText.nodeValue);           // 君不见高堂明镜悲白发
newParagraph.setAttribute("title", "新段落");
alert(newParagraph.getAttribute("title")); // 新段落
```

新创建的 DOM 节点独立存在，与当前 DOM 树并无关联，也就无法在浏览器窗口中得以显示。要将新创建的 DOM 节点展现给用户，必须将其附加到当前 document 对象上。document 对象提供了附加、插入、删除、替换 DOM 节点的方法，参见表 8-4。

表 8-4 document 对象的 DOM 元素附加、插入、删除、替换方法

方法名	描述
appendChild(node)	将 node 附加成为当前节点的子节点，成为其 lastChild
insertBefore(new, old)	将 new 插入成为当前节点的子节点，为 old 节点的 previousSilbling
removeChild(node)	从当前节点的子节点当中删除 node
replaceChild(new, old)	将当前节点的 old 子节点，替换为 new

使用表 8-4 方法，将新创建的 DOM 节点附加或者插入到当前 DOM，它们就得以在浏览器中显示。同样，将 DOM 树上的节点删除，其对应的内容也就会从浏览器中消失。图 8-13 中用户点击“添加/移除段落”，一段新的诗句就会添加或被移除。其代码如源代码 8-16 所示。

图 8-13 源代码 8-16 运行效果

源代码 8-16 使用 document 对象的方法添加/删除 DOM 节点示例

```
------------------------- HTML -----------------------------
.....
<div>
    <h1>将进酒</h1>
    <h2>李白</h2>
    <p>
        君不见黄河之水天上来，奔流到海不复还。 <br/>
        ......
    </p>
        <button>添加/移除段落</button>
```

```
</div>
……
------------------------- CSS -------------------------------
.newParagraph{border: 1px solid black;}

------------------------ JavaScript -------------------------
window.onload = function(){
  newParagraph = document.createElement("p"); // 新诗句段落，用全局
变量存储
  var newText = document.createTextNode("君不见高堂明镜悲白发，朝如
青丝暮成雪");
  newParagraph.className = "newParagraph";
  newParagraph.appendChild(newText);
  document.getElementsByTagName('button')[0].onclick =
toggleNewParagraph;
}

function toggleNewParagraph(){
  var poem = document.getElementsByTagName('div')[0];
    var paragraphs = document.getElementsByTagName('p');
    if(paragraphs.length > 1){
        poem.removeChild(newParagraph);
    }else{
      var oldParagraph = document.getElementsByTagName('p')[0];
      poem.insertBefore(newParagraph, oldParagraph.nextSibling);
    }
}
```

8.5 DOM 和 BOM 全局对象

至今为止，上一章和本章的示例当中多次出现了 document 对象，许多程序中它都是使用 DOM 的入口。document 对象是 DOM 树的根，是浏览器进程按照 DOM 标准暴露给 JavaScript 程序使用的全局对象（接口）。JavaScript 程序通过使用它，可以遍历、导航、获取 DOM 树上的节点，然后进一步操作。

DOM 标准定义了浏览器应当如何将当前加载网页的对象暴露给脚本和程序使用。但是，有些时候，除了当前网页，脚本程序还需要使用其它和浏览器相关的数据，例如：当前浏览器的类型、厂商、版本，用户的浏览器访问历史，浏览器窗口当前在整个用户屏幕上的位置，等等。浏览器通过 **BOM（Browser Object Model）提供了几个全局对象（接口），向脚本和程序开放这些数据**，参考表 8-5。

表 8-5 BOM 对象一览表

对象名	描述
window	浏览器用于显示网页的窗口
document	浏览器窗口内当前的网页，DOM 树的根（即是

	BOM 成员，又是 DOM 成员）
location	当前网页的 URL
navigator	浏览器本身
screen	浏览器当前占据的屏幕区域
history	浏览器用户访问历史

BOM 是浏览器运行时向脚本和程序开放的对象（接口）模型，它包括了通过 document 对象开放的 DOM。BOM 和 DOM 的关系，如图 8-14 所示。

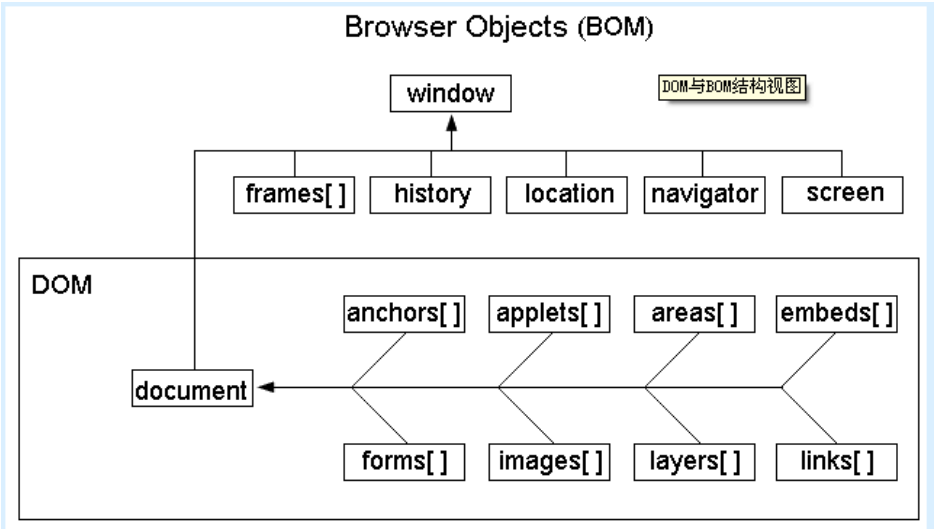

图 8-14 BOM 与 DOM 关系图[9]

BOM 无标准、有共识：不同于 DOM，迄今为止还没有关于 BOM 的统一标准。不过，由于不同浏览器之间的渊源关系和业界的共识，绝大部分的浏览器都开放了表 8-5 中这些对象，并且普遍有相同（类似）的方法。下面介绍绝大部分的浏览器均支持 BOM 对象方法。

8.5.1 window

window 是浏览器当前窗口对象，它是浏览器进程对外开放的最顶层对象，也是整个网页上 JavaScript 脚本的最顶层作用域，所有的全局对象、全局变量实际上都是它的属性或者说成员变量，参考第 9 章作用域一节。

同样，第 7 章讲述的 JavaScript 输出方法，alert、confirm、prompt，实际上都是 window 对象的方法。也就是说，当 JavaScript 脚本使用 alert、confirm、prompt

[9] 摘自 http://blog.csdn.net/yjsuge/article/details/6538631。

等方法时，实际上是通过 window 对象，要求浏览器进程生成需要的对话框。此时，浏览器进程会使用操作系统接口，使用操作系统的视窗子系统接口，在屏幕上绘制出需要的对话框。

window 对象还提供其它丰富、有用的方法，参考表 8-6。

表 8-6 window 对象常用方法列表

对象名	描述
alert、confirm、prompt	弹出对话框（参考相应章节）
setInterval 、 setTimeout 、 clearInterval、clearTimeout	定时器（参考定时器一节）
open(url, name, options) 、close()	打开新的浏览器窗口，关闭当前浏览器窗口[10]
print()	打印当前网页
focus()、blur()	使当前浏览器窗口获得焦点、使当前浏览器失去焦点[11]
scrollBy(dx, dy)、scorllTo(x, y)	将浏览器窗口内页面纵向滚动 dx（负值为向上），横向滚动 dy（负值向左）；将浏览器窗口内页面滚动到(x, y)座标

源代码 8-17 展示了使用 winow 对象 open 方法打开浏览器新窗口（标签页）的例子，运行截图如图 8-15。

源代码 8-17 使用 winow 对象 open 方法打开浏览器新窗口（标签页）示例

```
------------------------- HTML -----------------------------
......
        <button>打开新窗口</button>
......
------------------------ JavaScript -------------------------
window.onload = function(){
  var button = document.getElementsByTagName('button')[0];
  button.onclick = function(){
    window.open(docuemt.URL, "新窗口"); //参考表 8-7
  }
}
```

[10] 根据用户配置不同，多标签页浏览器，或为打开新标签页，关闭当前标签页。

[11] 多任务操作系统中，获得焦点的窗口和用户交互，接受用户的操作，其余窗口在后台运行。

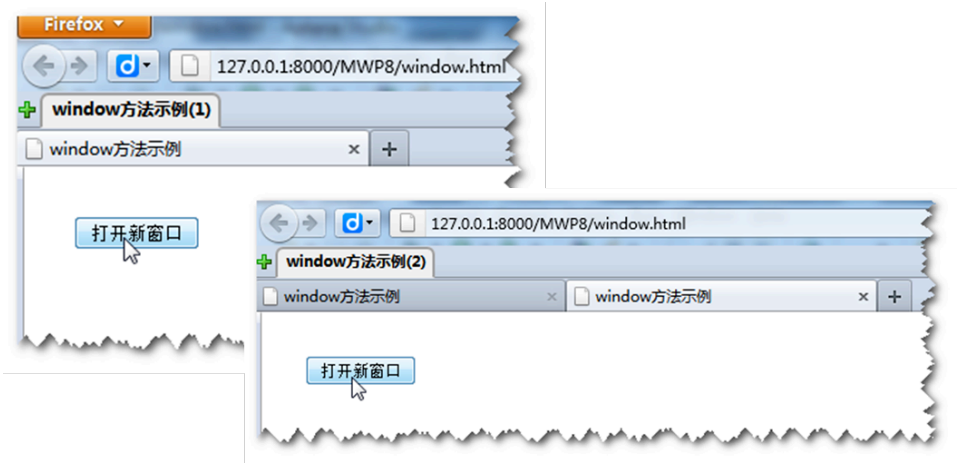

图 8-15 源代码 8-17 运行效果（左上：点击按钮前，右下：点击按钮后）

window 对象调节窗口位置和大小方法不再适用：早期 window 对象提供了 moveBy、moveTo，resizeBy、resizeTo 等方法调节窗口位置和大小。现代浏览器大多都不再支持这些方法，不再让网页代码控制浏览器窗口的大小和在屏幕上的位置，而让浏览器用户自行决定。

8.5.2 document

document 对象即是 BOM 成员，又是 DOM 成员。除了已经介绍的方法之外，document 对象还有一系列丰富的 properties 和方法供脚本和程序使用。

a) properties

表 8-7 列出了 document 对象的 properties。

表 8-7 document 对象 properties 列表

property 名	描述
cookie	当前网页有效的所有 cookie，以名值对的形式返回
domain	提供当前网页的 Web 服务器之域名
referrer	前一个网页，用户从那里点击链接访问了当前网页
title	当前网页的 title
URL	当前网页的 URL（参考源代码 8-17）
anchors	当前网页上的所有链接（仅包括<a>元素）

links	当前网页上的所有链接和热点（包括<a>元素和<area>元素[12]）
forms	当前网页上的所有表单
images	当前网页上的所有图片

例如，源代码 8-18 中的程序使用 images properties 为网页上所有的图片增加点击（click）事件处理器，响应用户点击操作。用户点击时，将弹出对话框说明图片的文件名。

源代码 8-18 使用 document 对象 properties 示例

```
window.onload = function(){
  var images = document.images;
  for (var i = 0; i < images.length; i++) {
    images[i].onclick = showName;
  }
}

function showName(event){
    var name = getName(event.target.src);
    alert(name);
}

function getName(url){
    return url.substring(url.lastIndexOf("/") + 1, url.length);
}
```

b) 方法

8.4.4 节已经讲述了如何使用 document 的方法选取 DOM 元素。除此之外，document 还提供了一组方法，供程序和脚本在运行时动态的写入网页文档。

表 8-8 document 对象动态写入网页文档方法列表

方法	描述
open()	打开输出流，准备写入文档内容
write(content)	在输出流当中写入网页内容
writeln(content)	在输出流当中写入网页内容，每次一行
close()	关闭当前输出流，将其内容显示到浏览器窗口中

源代码 8-19 使用 document 对象方法前端动态写入网页文档示例（网页加载完成前）

```
-------------------------- HTML ------------------------------
......
```

[12] HTML area 元素用来在 HTML map 元素（图片导航）上设置热区。map 元素和 area 元素的用法，请参考：http://www.w3.org/wiki/HTML/Elements/map。

```
<body>
    <div>
        <script>document.writeln("<p>动态写入的内容
</p>");</script>
        <button>再次动态写入</button>
    </div>
</body>
……
```

图 8-16 源代码 8-19 运行效果（上：浏览器窗口的显示，下：浏览器源代码视图）

慎用 document 对象 open、close 方法[13]：网页解析加载过程中，不需要使用 open 方法打开文档流。此时，当前网页的文档流已经打开，浏览器正在解析加载，只需要用 write 或者 wirteln 方法写入内容即可。写入的内容会立刻被浏览器解析，与原有网页内容一起加载。

但是，在网页加载完成后，如果要动态写入网页文本，则必须先使用 open 方法打开文档流，然后用 write 或者 wirteln 方法写入内容，写完之后还必须使用 close 方法关闭文档流。并且，此时浏览器会重新加载渲染动态写入的网页文档，也就是说，**原有文档所有内容会丢失，被替换为新写入的内容！**参见源代码 8-20 和图 8-17。

源代码 8-20 使用 document 对象方法前端动态写入网页文档示例（网页加载完成后）

```
------------------------- HTML -------------------------------
……
```

[13] 严格说来，document 对象动态写入网页文档的方法，不符合 XHTML 标准，应该尽量少使用。因为这样的动态生成方式与 XML 文档的解析要求不符，详情参考：http://www.w3.org/MarkUp/2004/xhtml-faq#docwrite。

```
<body>
    <div>
        <button>动态写入内容</button>
    </div>
</body>
……
------------------------ JavaScript --------------------------
window.onload = function(){
  document.getElementsByTagName("button")[0].onclick =
writeNewHTML;
}

function writeNewHTML(){
  document.open();
  var txt="<html><head><style>*{color:red}</style></head><body>
前端动态生成的HTML</body></html>";
  document.write(txt);
  document.close();
)
```

图 8-17 源代码 8-20 运行效果
（左上：点击按钮前，右上：点击按钮后，下：点击按钮后浏览器源代码视图）

8.5.3 location

location 对象当前访问网页的地址（URLs），其 properties 如表 8-9 所示

表 8-9 location 对象 properties 列表（参考第一章 URL）

方法	描述
href	完整的 URL，location.href === document.URL 为永真式
protocol	协议名
host	域名（主机名），包括端口号
hostname	域名（主机名），不包括端口号
port	端口号

pathname	文件路径
hash	锚点
search	查询字符串

源代码 8-21 使用这些 properties，在网页上列出了当前网页 URL 的信息，参考图 8-18

源代码 8-21 使用 screen 对象方法获取用户屏幕信息示例

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>screen对象用法示例</title>
<script>
    function showLocationInfo(){
        document.writeln("protocol: " + location.protocol + " <br />");
        document.writeln("host: " + location.host + " <br />");
        document.writeln("hostname: " + location.hostname + " <br />");
        document.writeln("port: " + location.port + " <br />");
        document.writeln("pathname: " + location.pathname + " <br />");
        document.writeln("hash: " + location.hash + " <br />");
        document.writeln("search: " + location.search + " <br />");
    }
</script>
</head>
<body>
    <p>
        <script>showLocationInfo();</script>
    </p>
</body>
</html>
```

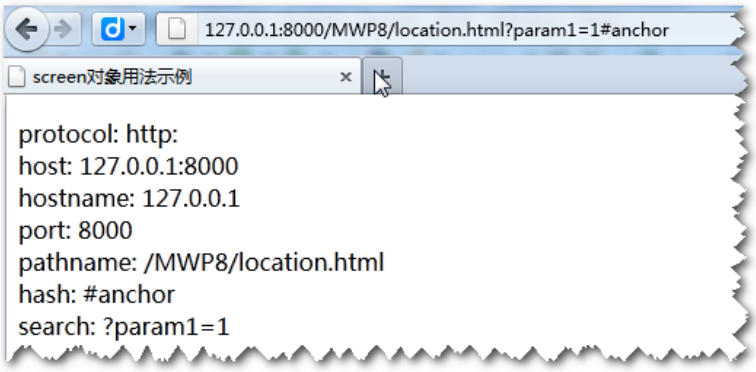

图 8-18 源代码 8-21 运行效果

location 对象还有 3 个有用的方法，可以命令浏览器重新向服务器请求网页，如表 8-10 所示。

表 8-10 location 对象方法列表

方法	描述
reload()	向服务器重新请求当前网页
assign(url)	在当前窗口（标签页）打开给定的网页（url）
replace(url)	与 assign 相同，在当前窗口（标签页）打开给定的网页（url）

8.5.4 navigator

navigator 对象代表了浏览器本身，其 properties 如表 8-11 所示

表 8-11 navigator 对象 properties 列表

property 名	描述
appCodeName	浏览器的代码名称
appName	浏览器名
appVersion	浏览器版本
cookieEnabled	是否支持 cookie
platform	浏览器是在什么平台下编译的
userAgent	浏览器在和 Web 服务器通信时，HTTP(s)报文头部 user-agent 字段的值

源代码 8-22 使用这些 properties，在网页上列出了浏览器的相关信息，参考图 8-19。

源代码 8-22 使用 navigator 对象方法获取浏览器信息示例

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>navigator对象用法示例</title>
<script>
  function showNavigatorInfo(){
    document.writeln("appCodeName: " + navigator.appCodeName + " <br />");
    document.writeln("appName: " + navigator.appName + " <br />");
    document.writeln("appVersion: " + navigator.appVersion + " <br />");
    document.writeln("cookieEnabled: " + navigator.cookieEnabled + " <br
/>");
    document.writeln("platform: " + navigator.platform + " <br />");
    document.writeln("userAgent: " + navigator.userAgent + " <br />");
  }
</script>
</head>
<body>
    <p>
        <script>showNavigatorInfo();</script>
    </p>
```

……

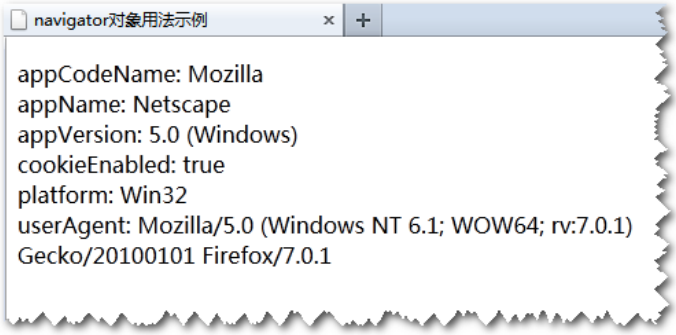

图 8-19 源代码 8-22 运行效果（作者 Firefox 7.0.1 中的运行结果）

8.5.5 screen

screen 对象代表了浏览器所在的用户屏幕，其 properties 如表 8-12 所示

表 8-12 screen 对象 properties 列表

property 名	描述
availHeight	用户屏幕的高度（不包括 Windows 的任务栏）
availWidth	用户屏幕的宽度（不包括 Windows 的任务栏）
height	用户屏幕的整个高度（包括 Windows 的任务栏）
width	用户屏幕的整个宽度（包括 Windows 的任务栏）
colorDepth	用户屏幕的颜色解析度，每个像素点使用多少位数字来表示

源代码 8-23 使用这些 properties，在网页上列出了用户屏幕的相关信息，参考图 8-20。

源代码 8-23 使用 screen 对象方法获取用户屏幕信息示例

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>navigator对象用法示例</title>
<script>
    function showScreenInfo(){
        document.writeln("availHeight: " + screen.availHeight + " <br
/>");
        document.writeln("availWidth: " + screen.availWidth + " <br />");
        document.writeln("height: " + screen.height + " <br />");
        document.writeln("width: " + screen.width + " <br />");
        document.writeln("colorDepth: " + screen.colorDepth + " <br />");
```

```
    }
</script>
</head>
<body>
    <p>
        <script>showScreenInfo();</script>
    </p>
</body>
</html>
```

图 8-20 源代码 8-23 运行效果（作者 Firefox 7.0.1 中的运行结果）

8.5.6 history

history 对象代表了用户在当前窗口（标签页）访问过的网页（URLs），它只有一个 property，length，值为当前窗口访问过的网页的个数。history 对象还提供了 3 个方法，用来访问曾经访问过的网页，如表 8-13 所示

表 8-13 history 对象方法列表

方法	描述
back()	在当前网页的访问历史列表上，返回访问上一个网页
forward()	在当前网页的访问历史列表上，返回访问下一个网页
go(step)	在当前网页的访问历史列表上，返回访问从当前网页起第 step 个网页，step 为正数向前计算，为负数向后倒退

8.6 DOM 事件

前端编程就是使用 JavaScript 语言为 DOM（BOM）对象编写事件处理器（Event Handler）。在运行时，用户操作触发事件，事件处理器得以执行，它操纵 DOM 和 BOM 对象，改变网页在浏览器中的内容、结构和外观，表现出一定的行为，用户的操作也就得到了响应，实现了前端动态性。

8.6.1 event 对象

显然，DOM 事件（DOM event）在前端编程中地位非常重要。DOM event 是一个对象，其常用属性包括：

表 8-14 event 对象常用属性[14]

方法	描述
type	事件名称，例如点击事件为'click'
target	发生事件的 DOM（BOM）元素
currentTarget	其事件处理器正在执行的 DOM（BOM）元素
eventPhase	事件当前的阶段（capture、target、bubbling，参考 8.6.3 节）
altKey	事件发生时，键盘上的“Alt”键是否被按下
ctrlKey	事件发生时，键盘上的“Ctrl”键是否被按下
shiftKey	事件发生时，键盘上的“Shift”键是否被按下
button	事件发生时，用户鼠标哪个键被按下
clientX	事件发生时，鼠标在浏览器窗口的横坐标，浏览器左上角为原点
clientY	事件发生时，鼠标在浏览器窗口的纵坐标，浏览器左上角为原点
screenX	事件发生时，鼠标在屏幕上的横坐标，屏幕左上角为原点
screenY	事件发生时，鼠标在屏幕上的纵坐标，屏幕左上角为原点

所有的事件处理器都有一个参数 event，此参数就 DOM event 对象。事件处理器通过它获取事件的各种属性。源代码 8-24 中，按钮的 click 事件处理器将 event 的各种属性写在网页上，参见图 8-21。

源代码 8-24 event 对象属性示例

```
------------------------- HTML ------------------------------
……
```

[14] 不同浏览器在 event 属性上有所差异，特别是早期的浏览器，这里以 DOM 标准为准讲述。编码实践中，常常通过 JavaScript 库来帮助解决浏览器 JavaScript 和 DOM 兼容性问题，例如第 9 章介绍的 jQuery。

```
    <pre id="eventInfo"></pre>
    <button>查看event属性</button>
……
------------------------ JavaScript --------------------------
window.onload = function(){
    document.getElementsByTagName('button')[0].onclick =
showEventAttributes;
}

function showEventAttributes(event){
    var paragraph = document.getElementById("eventInfo");
    var eventInfo = createEventAttributesInfo(event);
    var textNode = document.createTextNode(eventInfo);
    paragraph.appendChild(textNode);
}

function createEventAttributesInfo(event){
    var message = "type: " + event.type + "\n";
    message += "target: " + event.target + "\n";
    message += "currentTarget: " + event.currentTarget + "\n";
    message += "altKey: " + event.altKey + "\n";
    message += "ctrlKey: " + event.ctrlKey + "\n";
    message += "shiftKey: " + event.shiftKey + "\n";
    message += "button: " + event.button + "\n";
    message += "clientX: " + event.clientX + "\n";
    message += "clientY: " + event.clientY + "\n";
    message += "screenX: " + event.screenX + "\n";
    message += "screenY: " + event.screenY;
    return message;
}
```

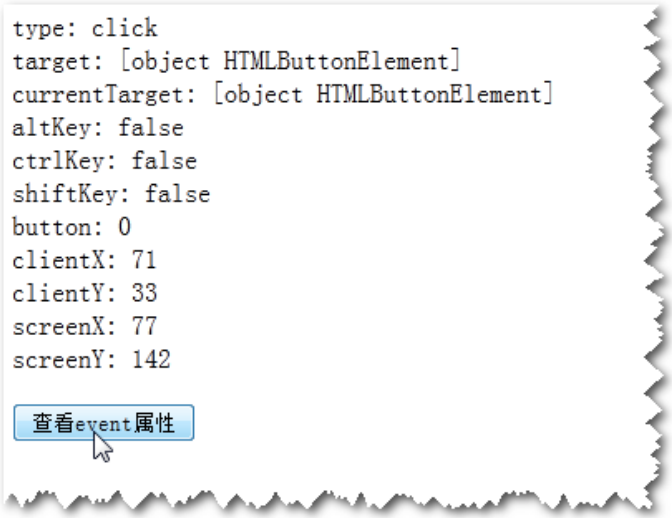

图 8-21 源代码 8-24 运行效果（鼠标左键点击“查看 event 属性”按钮后）

this == event.currentTarget：事件处理器代码中的 this：在事件处理器中直接使用 this 得到的是正在处理事件的 DOM 对象，也就是 event.currentTarget 指向的 DOM 对象，而如果调用其它函数再使用 this，得到的是 window 对象。事件发生时，事件处理器是作为正在处理事件 DOM 对象的相应“onXXX”属性被调用的，也就是执行类似“domElement.onXXX();”的语句。因此，事件处理器中函数中的 this 指向当前正在处理事件的 DOM 对象。而如果事件处理器调用了其它函数，则必须要小心，此时的 this 不再指向正在处理事件的 DOM 对象，参考源代码 8-25 和图 8-22 的示例。这样的问题和 JavaScript 对 this 关键字的解析方法有关，详细内容参考第 9 章 this 一节。

源代码 8-25 event.target vs. this 示例

```
------------------------- HTML -----------------------------
......
    <pre id="message"></pre>
    <button>target vs. this</button>
......
----------------------- JavaScript -------------------------
window.onload = function(){
    document.getElementsByTagName('button')[0].onclick =
showTargetAndThis;
}

function showTargetAndThis(event){
    var paragraph = document.getElementById("message");
    var eventInfo = createTargetAndThisInfo(event);
    var textNode = document.createTextNode(eventInfo);
    paragraph.appendChild(textNode);
}

function createTargetAndThisInfo(event){
    var message = "target: " + event.target + "\n";
    message += "this: " + this;
    return message;
}
```

图 8-22 源代码 8-25 运行效果（点击“target vs. this”按钮后）

8.6.2 添加事件处理器

给 DOM 元素添加事件处理器的方式有 3 种。例如，给网页上的按钮添加点击事件处理器

1. 给 HTML 元素增加事件处理器属性，事件处理器名称为“on 事件名”，参见源代码 8-26（方法 1）
2. 给 DOM 元素事件处理器 property 赋值，事件处理器 property 名称为“on 事件名”，参见源代码 8-26（方法 2）
3. **使用 DOM 元素的添加事件处理器方法 addEventListener**，方法第 1 个参数为事件名，第 2 个参数是事件处理器，参见源代码 8-26（方法 3）

源代码 8-26 不同方法添加事件处理器示例

```
---------------------- 方法1（HTML）---------------------------

<button onclick="alert('Hello world!')">……

---------------------- 方法2(JavaScript) ----------------------

button.onclick = function(){alert('Hello world!');};

---------------------- 方法3(JavaScript) ----------------------

button.addEventListener('click', function(){alert('Hello
world');});
```

第 1 种方法将 JavaScript 代码和 HTML 代码混杂在一起，把网页的行为和内容结构混为一谈，显然违反了最基本的原则（参考）。除了极为简单的网页外，不要这样做。

第 2 种方法在大部分的情况下能够很好地解决问题。不过，如果 DOM 元素本身已经有了该事件的处理器，这样的直接赋值，会将原有的处理器移除。也就是说，原来用来相应用户操作的行为不复存在了。有些时候，这并不是程序想要的结果，程序如果需要在保留原有处理行为的同时，添加新的行为，则要使用第 3 种方法。

对应于 addEventListener 方法，DOM 元素还有 removeEventListener 方法，将给定事件处理器移除。

源代码 8-27 中的按钮同时有两个事件处理器，并且使用 addEventListener 和 removeEventListener 方法，交替变化其中一个事件处理器，运行效果如图 8-23。

源代码 8-27 添加/移除事件处理器示例

```
------------------------- HTML -----------------------------
……
        <p></p>
        <button>变!</button>
……
------------------------ JavaScript -------------------------
window.onload = function(){
    var button = document.getElementsByTagName("button")[0];
    button.addEventListener('click',showClickTimes);
    button.addEventListener('click',handler1);
}

var count = 0;

function showClickTimes(event){
    var paragraph = document.getElementsByTagName("p")[0];
    count++;
    paragraph.textContent = "第" + count + "次点击";
}

function handler1(event){
    alert("去年今日此门中，人面桃花相映红");
    event.target.removeEventListener('click', handler1);
    event.target.addEventListener('click',handler2);
}

function handler2(event){
    alert("人面不知何处去，桃花依旧笑春风");
    event.target.removeEventListener('click', handler2);
    event.target.addEventListener('click', handler1);
}
```

图 8-23 源代码 8-24 运行效果（左：第 1 次点击，右：第 2 此点击）

8.6.3 DOM 事件传播机制

HTML 文档和 DOM 都是树型结构，因此当一个事件发生的时候，它既发生在当前元素上，也发生在当前元素的父元素和祖先元素上。例如，源代码 8-28 所示的网页，运行时如果用户点击了按钮，那么点击的事件也同样发生在 form、div 和 body 元素上。

源代码 8-28 DOM 树与事件

```
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<div>
    <form action="#">
        <input type="button" value="点击"/>
    </form>
</div>
</body>
</html>
```

既然如此，那么需要定义事件在这些 DOM 元素之间发生的前后顺序和传递规律，这样用户行为发生、事件产生时，浏览器才有确定的行为，这样程序员才能够正确设计程序。DOM 规范定义了这一事件的传播机制，称为 DOM 事件传播（DOM event propogation），它规定了事件的产生、沿 DOM 传播的顺序和阶段、终止传播和停止执行浏览器默认事件处理器的方法等。

a) 捕获、目标与冒泡

按 DOM 2 标准，浏览器进程在捕获到来自操作系统的用户事件（User event）之后，将其转换成为 DOM 事件（DOM event），事件发生地点（发生时鼠标所在位置）对应的最内层 DOM 元素，被称为事件目标，DOM event 的 target 属性指向这一 DOM 元素。

事件（DOM event）分 3 个阶段在 DOM 树上传播。第 1 阶段称为捕获阶段（capture phase），事件从窗口（window，即图 8-24 中的 defaultView）开始，传递到 DOM 树的根 document，然后沿着从 DOM 树根到事件目标（event.target）的唯一的路径，逐次在 DOM 元素间传递。

第 2 阶段称为目标阶段（target phase）很短，特指事件传递到事件目标。其后，事件会反过来，沿着来时的路径，从事件目标再传递回 DOM 树的根 document，这个阶段称为冒泡阶段（bubbling phase），参见图 8-24。

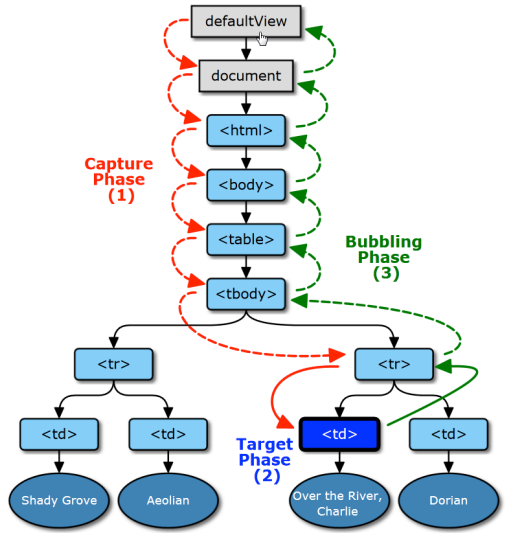

图 8-24 DOM event 传播阶段和路径[15]

在事件传播的每个阶段，可以注册相应的事件处理器。DOM 的 addEventListener 和 removeEventListener 方法都有第 3 个参数 isBubbling，该参数为 boolean 类型，用以说明事件处理器是否适用于冒泡阶段。其值默认为 false[16]，也就是说对捕获和目标阶段的事件进行监听和处理，而不理会冒泡阶段的事件。当设置其值为 true 时，对冒泡阶段的事件进行处理，不理会捕获和目标阶段的事件。

例如源代码 8-29 中的 HTML 文档，'outter' div 嵌套了'inner'div，'inner'div 嵌套了 button。JavaScript 程序分别为它们注册了 bubbling 和非 bubbling 的事件处理器，并将在事件处理器 showEvent 中显示当前事件的信息，包括：发生事件的 DOM 元素，执行事件监听器的 DOM 元素，事件传播的阶段，等等。图 8-25 展示了运行时事件发生和传播的顺序。在捕获阶段，事件从'outter' div，经过'inner'div，传递到 button，进入目标阶段。在冒泡阶段，传播顺序反过来，从'inner'div 传递到'outter' div。

源代码 8-29 DOM 事件传播示例

```
------------------------- HTML -------------------------------
<html xmlns="http://www.w3.org/1999/xhtml">
……
    <div id='outter'> Outter
        <div id='inner'> Inner<br />
            <button id='button'>触发事件</button>
```

[15] 摘自 http://www.w3.org/TR/DOM-Level-3-Events/images/eventflow.svg。早期的 Internet Explorer 事传播机制不符合标准，没有 capture 阶段，IE 9 已经符合标准：http://ie.microsoft.com/testdrive/HTML5/ComparingEventModels/Default.html。

[16] JavaScript 函数参数可以实现默认参数，请参考第 9 章函数部分。

```
……

--------------------------- CSS -----------------------------
div {padding:1em; border: 1px solid black;}

------------------------ JavaScript --------------------------
var sequenceNo = 1; // 用于标识事件发生顺序

window.onload = function(){
    registerListeners(false);
    registerListeners(true);
}

function registerListeners(isBubbling){
    clickShowEvent("outter", isBubbling);
    clickShowEvent("inner", isBubbling);
    clickShowEvent("button", isBubbling);
}

function clickShowEvent(elementId, isBubbling, sequenceNo){
    var element = document.getElementById(elementId);
    element.addEventListener('click', showEvent, isBubbling);
}

function showEvent(event){
    var message = '第 ' + sequenceNo + ' 次处理事件';
    sequenceNo++;
    message += '\n当前处理事件的DOM元素是 ' +
event.currentTarget.tagName + ', id: ' + event.currentTarget.id;
    message += '\n事件来源于DOM元素  ' + event.target.tagName + ',
id: ' + event.target.id;
    message += '\n事件处于 ' + getPhaseName(event.eventPhase) + '
阶段';
    alert(message);
}

function getPhaseName(eventPhase){
    if(eventPhase == 1) return 'capture';
    if(eventPhase == 2) return 'target';
    if(eventPhase == 3) return 'bubbling';
}
```

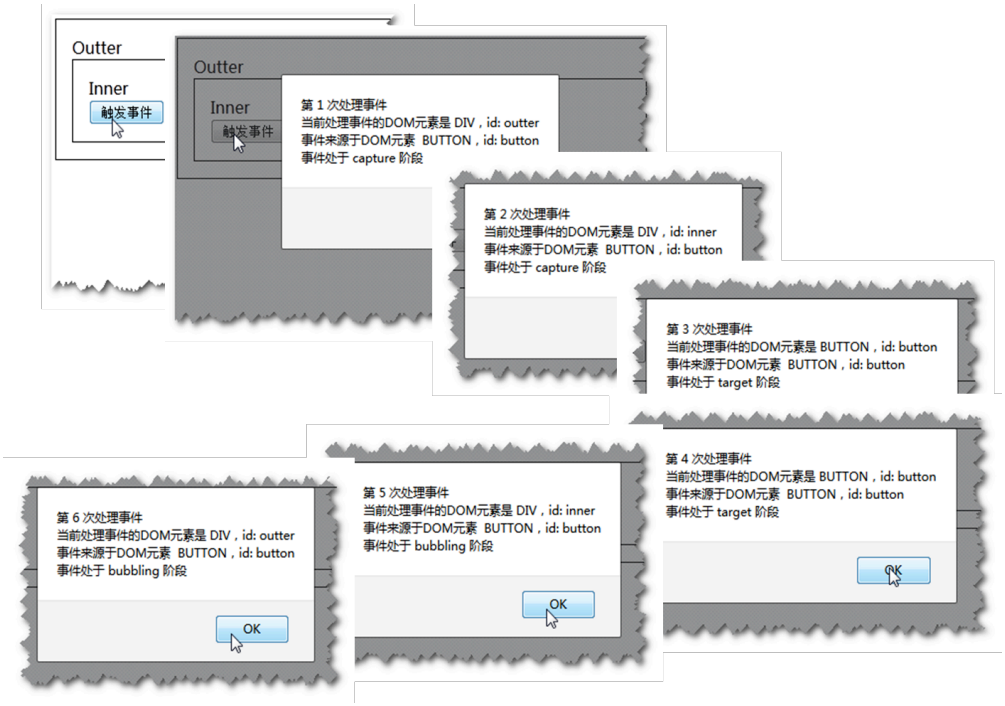

图 8-25 源代码 8-29 运行效果（截图按时间顺序从上到下排列）

区分捕获和目标：通过事件的 currentTarget 和 target 属性可以区分事件是出于捕获阶段还是目标阶段。二者不等则为捕获阶段，相等则是目标阶段。

8.6.4 常用事件

a) onload 与非侵入式 JavaScript

8.6.5 定时器

8.7 其它可编程对象

BOM

小结

- 浏览器获取、解析、加载、渲染网页，其核心是渲染引擎。
- Web 前端动态性是这样实现的，JavaScript 脚本响应用户对网页的操作，通过标准接口操纵网页在浏览器进程内存中的对象，改变其内容、结构、外观，反映在浏览器窗口中，就是网页发生了改变。
- 前端编程属于事件驱动的编程，最主要就是为各类浏览器和网页事件添加事件处理器。
- 浏览器的 JavaScript 引擎负责解释执行网页上的 JavaScript 代码。
- JavaScript 有 number、string、boolean、array、object、function、null 和 undefined 等 8 中内置类型。
- NaN 是特殊的 number，表示不是数字。
- 除了 false、0、NaN、空字符串""、null 和 undefined，其余值均为 true。特别是“0”、空数组[ ]，在 PHP 中为 false，在 JavaScript 中为 true
- JavaScript 在函数外或者函数内隐式声明的变量为全局变量。
- JavaScript 可以有没有函数名的函数，称为匿名函数。
- JavaScript 函数实际参数个数可以不同于形式参数，多则被忽略，少的被视为 undefined。
- document.getElementById 方法可以获得给定 id 的 HTML 元素在浏览器进程内存中的对象。

练习题

1. 改进本章案例：用颜色区分当前新闻是收起还是放开。对于放开展示的新闻，标题为蓝底白字；而收起的新闻，标题为灰底黑字。
2. 仿造**错误！未找到引用源。**案例研究，同样为日程表增加点击收放的功能。

进一步阅读

1. DOM 的目的、历史变迁与基本介绍：http://en.wikipedia.org/wiki/Document_Object_Model

2. JavaScript: The World's Most Misunderstood Programming Language: 为 JavaScript 语言正名的力作，澄清了不少关于 JavaScript 的错误说法: http://javascript.crockford.com/javascript.html
3. JavaScript 引擎介绍：http://en.wikipedia.org/wiki/JavaScript_engine
4. 使用 Firebug 调试 JavaScript：http://getfirebug.com/screencasts
5. 使用 IE 调试工具调试 JavaScript：http://msdn.microsoft.com/en-us/library/dd565627%28v=vs.85%29.aspx
6. ECMAScript Specification: http://www.ecma-international.org/publications/standards/Ecma-262.htm
7. Mozilla Developer Center – Core JavaScript 1.5 Guide: https://developer.mozilla.org/en/JavaScript/Guide
8. Mozilla Developer Center – Core JavaScript 1.5 Reference: https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference
9. W3Schools style DOM object 属性：http://www.w3schools.com/jsref/dom_obj_style.asp
10. W3Schools JavaScripts 教程：http://www.w3schools.com/js/default.asp
11. JavaScript: The Definitive Guide：http://books.google.com/books/about/JavaScript.html?id=2weL0iAfrEMC