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 7 章 交互式网页与 JAVASCRIPT

关键知识点：

- 前端编程原理
- 事件驱动
- 前端编程基本模式
- JavaScript 基本语法
- null 和 undefined
- 变量作用域
- 字符串、数组、函数

现代 Web 应用中，常常把浏览器端称为前端，把 Web 服务器端，包括 Web 服务器和 Web 应用称为后端。早期网页的动态性，也就是响应用户操作，显示不同内容的能力，完全来自于后端。后端根据用户的请求和应用状态、应用逻辑生成网页代码给回浏览器。浏览器只是简单的解析和显示，网页在浏览器中所有内容都是静态的。用户对网页的所有操作，浏览器都会向后端提交请求，而后由后端再生成新的网页代码，给回浏览器解析、渲染、展示。这样每一次用户和网页或者 Web 应用的交互，都表现为网页的一次刷新。

虽然 Web 应用中不少用户操作确实需要后端来计算。例如，填写注册用户时，只有后端才能够真正判断用户给出的用户名是否已经被注册使用。但是，并不是所有操作都是这样，有些操作的处理完全可以在前端完成。例如，注册用户时，提示用户所填写的电子邮件地址格式是否正确；又如，响应用户点击，将页面某部分内容暂时收起来（隐藏）。

如果能够在前端提供网页动态性，让在浏览器端中无需向后端发送请求，也无需刷新页面，直接响应用户的部分操作；那么，既减轻了 Web 服务器负载，又改善了用户体验，显然十分有益。

事实上所有现代浏览器都提供这种前端动态性，前端动态性已经是现代 Web 应用最重要的技术构成之一。现代 Web 应用普遍使用 JavaScript 语言在浏览器端操纵网页对象，动态生成、改变、更新网页内容、结构和外观，及时响应用户操作。这种具备前端动态性的网页，称为动态 HTML（Dynamic HTML）。

注意，动态 HTML 和动态网页（Dynamic Web page，参见第 5 章）是不同的概念。前者是前端技术，指网页包括了 JavaScript 代码，在浏览器中显示时，可以响应用户操作，发生动态改变，其动态性在前端。后者是后端技术，指生成网页的程序，Web 服务器根据浏览器请求，运行动态网页程序，生成并返回不同的网页代码给浏览器，其动态性在后端。

从本章开始，本书将讲解现代 Web 程序设计中的前端编程，也就是浏览器编程，即如何编写动态 HTML。前端编程的主要编程范式是事件驱动（Event Driven），其核心是使用 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。本章着重介绍浏览器端程序，也就是脚本语言的运行原理，JavaScript 语言的基本语法、构造和特点。本章需要读者对计算机程序的运行原理、计算机程序设计和面向对象的程序设计有初步的了解。

本节所有例程，请使用调试工具 Firebug，或者 IE 调试工具运行（参考 7.3.3）。

7.1 案例：登山俱乐部—好用的前端动态性

上一章的登山俱乐部 Web 应用——登录和订餐，让俱乐部成员感觉不错。如同所有的好的 Web 应用一样，很快用户就提出了新的需求。有成员说，目前俱乐部首页公告栏里面内容太多了，能不能不要一次都显示出来呢？可不可以将部分内容先收起来，当读者感兴趣点击的时候再显示？还有成员说，订餐页面是否可以提供个计算器，方便大家对订餐金额做个验算呢？又有成员建议，现在俱乐部有好多名山靓照，可不可以给出一个在线画册，展示精选的照片呢？

这些需求有技术上的共同特点，即并不需要后端处理，用前端技术在浏览器中动态改变现有网页的内容、结构和外观就能够实现。本章和下一章部分将由浅入深地介绍前端技术，逐步介绍浏览器端编程原理、编程语言——JavaScript 和编程对象——DOM。在本章末案例研究中将首先实现收放新闻内容的功能和计算器，参考图 7-1。在线画册则留到下一章实现。

图 7-1 点击标题收起/放开新闻内容功能示意图

7.2 前端编程原理

进行前端编程，必须清楚了解浏览器的运行原理，知道浏览器获取、解析、加载、渲染网页过程，明白脚本程序如何与浏览器交互，如何响应用户操作，改变网页的内容、结构和外观。

7.2.1 浏览器工作原理

浏览器是 Web 用户电脑或其它设备上的运行的一个应用程序，一个进程。网页和 Web 应用通过浏览器最终展示给用户，并通过浏览器和用户交互。具体运行当中，浏览器按照获取、解析、加载、渲染的步骤将网页呈现在浏览器窗口内。

首先，浏览器负责与 Web 服务器通信，通过 HTTP(s) 协议向后端发送请求，获取响应，也就是网页代码。得到网页代码后，浏览器将对获得的网页代码进行解析，并将解析后的 HTML 加载为自己进程内存空间中的对象。此时，网页已经不再是简单的文本，而是存活在自己进程内存空间中的可操作对象。这些对象是进行前端编程的关键，现代浏览器基本服从 W3C DOM 标准[1]，按照它来设计自己的网页对象。

完成解析之后，浏览器将根据网页代码中 CSS 样式，计算网页外观，而后使用渲染引擎将网页最后渲染成为展示在浏览器窗口内的样子。整个过程如图 7-2 所示。

图 7-2 浏览器网页获取、解析、加载和渲染示意图

[1] http://www.w3.org/DOM/。事实上，浏览器对标准的兼容程度有差异，参考浏览器兼容性问题（前向引用）。

渲染引擎

渲染引擎（render engine），也称为 Web 浏览器引擎（Web browser engine）或布局引擎（layout engine），是浏览器最复杂和最重要的组件（component），它负责在浏览器窗口“绘制”网页，将网页展示为用户最终看到的模样。

目前世界上有着上千种众多浏览器，但是却只有不超过 10 种渲染引擎，包括：Amaya、Trident、Tasman、Gecko、WebKit、KHTML、Presto iCab 等等，各种浏览器都使用这些渲染引擎开发，例如：IE 和各种称为 IE 内核的浏览器，包括 360 浏览器、QQ 浏览器、Maxthon、迅雷浏览器等等，都是使用了 Trident 渲染引擎；而 Safari、Google Chrome 等大量浏览器使用了 WebKit 渲染引擎；本课程推荐使用的 Firefox 和其他 Mozilla 基金会的浏览器使用了 Gecko 渲染引擎。

更多情况，请参考：
http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(HTML)

7.2.2 前端脚本编程原理

前端动态性就是浏览器在运行时，根据用户操作，动态改变网页的内容、结构和外观。要动态改变网页的内容、结构和外观，从 7.2.1 介绍的浏览器工作原理来看，实际上就是要动态改变浏览器进程内存空间里的网页对象，脚本语言就是用来完成这一任务的绝佳工具（参考本节末侧边栏）。现代 Web 应用的前端动态性，就是通过使用 JavaScript 脚本语言，操作浏览器内存空间里的网页对象来实现的。

也就是说，这样具备前端动态性，能及时响应用户操作的网页，除了网页内容的 HTML 代码、网页外观的 CSS 代码之外，还包括了可以在浏览器内动态执行的 JavaScript 代码。

浏览器从 Web 服务器获取所有代码后，解析 HTML，加载为网页内存对象，而后应用 CSS 样式，渲染成为浏览器窗口中的样子。对于 JavaScript 代码，则部分由浏览器在加载时执行，例如，给网页元素注册相应事件处理器（Event Handler）的 JavaScript 代码；其余大部分 JavaScript 代码，用于响应用户操作，在用户操作网页时执行。

浏览器内置的 JavaScript 引擎负责解释执行 JavaScript 代码，动态改变内存中的网页对象，随后渲染引擎重新渲染网页，浏览器窗口中的网页就显示出了相应的变化，用户的操作也就得到了响应。参考图 7-3。

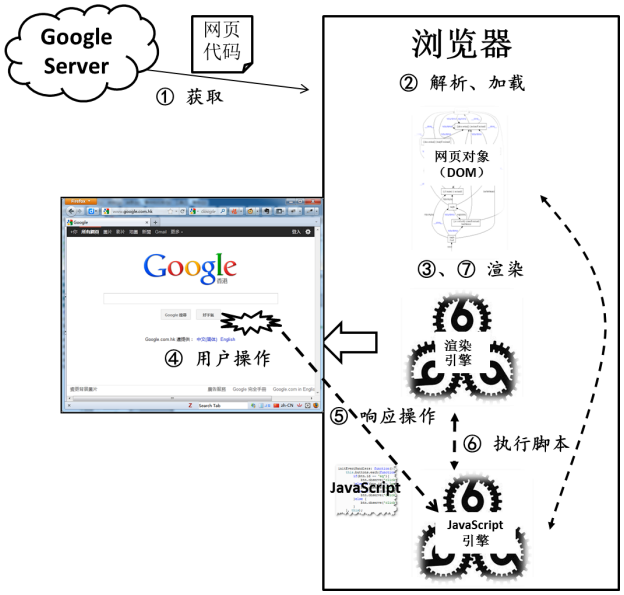

图 7-3 网页前端动态性示意图

脚本语言

脚本语言是一种特殊用途的解释性语言，它的解释引擎通常内置在某个系统程序或者应用程序之内，这样的系统程序或者应用程序称之为脚本语言的宿主。宿主程序运行时，解释引擎运行在宿主进程空间内，因而脚本语言能够通过解释引擎操纵宿主程序进程空间内的对象，从而改变宿主行为，为宿主程序提供运行时的动态性。

应用程序通过内置脚本语言解释引擎，并开放恰当的对象，可以产生很灵活的运行时动态性，也就是通过脚本语言编程控制的应用程序行为。目前，除了浏览器，还有很多应用程序这样做，例如：Microsoft Office 中的各种应用程序，都可以使用 VBScript 进行编程控制；各种 PDF 阅读器，也都内置了 JavaScript（ECMAScript）引擎，可以执行附加在 PDF 文档中的 JavaScript（ActionScript）脚本。

7.2.3 事件驱动程序设计

JavaScript 在浏览器中的用途是增加网页前端动态性，以便及时响应用户的操作。这使得绝大部分情况下前端编程符合事件驱动（Event Driven）的程序设计范式。事件驱动式程序的执行不同于顺序式程序，顺序式程序从入口（通常是 main 函数），逐句执行到结束；而事件驱动的程序代码何时执行，取决于运行时事件，通常是用户的操作——用户采取某个操作，随之产生某个事件，此时这个事件的事件处理器

（Event Handler）代码就会被执行。如果该事件不发生，那么这个事件处理器就不会被执行；如果该事件再次发生，那么这个事件处理器就会再次被执行。

例如，图 7-4 的示例中，用户点击网页中的按钮，产生了点击（click）事件，此时点击事件的事件处理器——onclick 方法就会被执行，也就弹出了对话框。用户不点击按钮，对应 onclick 方法就不会执行；用户多次点击按钮，则 onclick 方法就多次执行，每点击一次，执行一次。

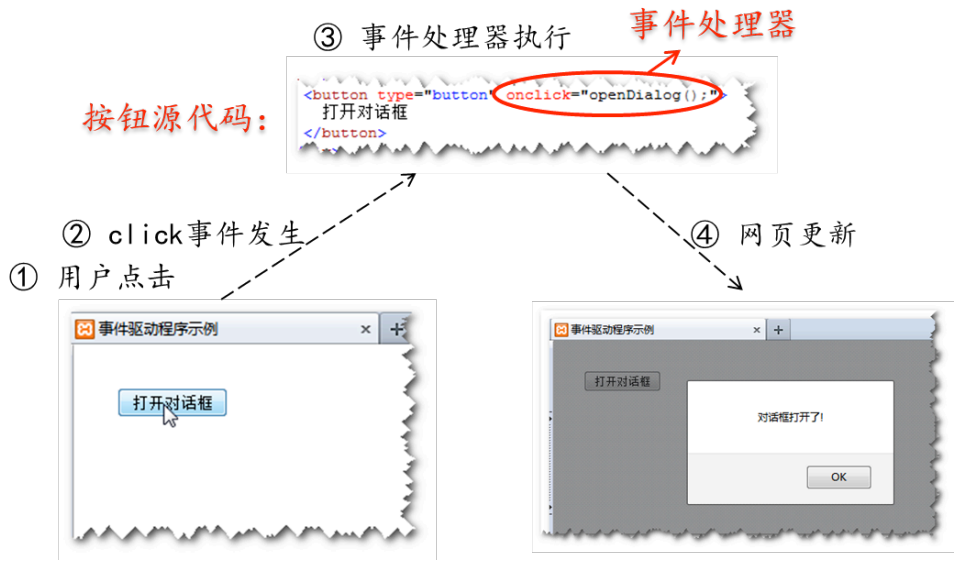

图 7-4 事件驱动程序运行示意图

至此，可以总结出前端编程的基本特点：

前端编程基本特点：使用 **JavaScript** 语言编写各类**事件处理器**，在事件处理器内通过**标准接口**操作**网页内存对象**，更新网页的内容、结构和外观。

通过这个特点可以看出，前端编程的学习重点有 3 个：1）JavaScript 语言；2）网页内存对象的标准接口，也就是 DOM；3）浏览器事件机制，主要是 DOM 事件。

7.3 JavaScript 语言简介

JavaScript 语言在编程语言分类上和之前学习的 PHP 语言一样，是一种弱类型、动态类型的脚本语言（参考）。JavaScript 的用途不同于 PHP，PHP 通常主要用于后端动态网页，而 JavaScript 主要用于为网页增加前端动态性。除此之外，JavaScript 还被广泛用于多种环境，为各种应用程序增加动态性，包括：PDF 阅读器，Windows

和 Mac OS 系统的桌面小应用（Widget），等等。

JavaScript 和 Java 语言之间并无直接的渊源关系，它为 Web 前端动态性而生，最初由网景（Netscape）公司为其网景（Netscape）浏览器开发，后来逐渐成长成为所有浏览器都支持的标准语言。目前，JavaScript 的语言标准由"欧洲计算机制造商联合会"（European Computer Manufactures Association, ECMA）标准化组织维护[2]。

近 20 年来 Web 的蓬勃发展，特别是 Web 2.0 时代大量 Web 应用的产生，极大地促进了 JavaScirpt 语言的发展。目前，JavaScript 语言已经从一门简单的脚本语言，发展成了能够同时支持结构化（procedure）、面向对象（object-oriented）和函数式（functional）等多种编程范式的现代程序设计语言，拥有庞大的程序员队伍和众多支持厂商，是现今世界上使用范围最广、最有前途的程序语言之一。

学习 JavaScript 的注意事项：JavaScript 的快速发展使得语言本身充满了各种特性，有的好、有的坏。程序员既可以用 JavaScript 写出优雅、高效的程序，也有可能写出糟糕的程序。并且，因为其简单易学，许多非专业人士也写出了大量糟糕的 JavaScript 代码。更糟糕的是，Web 上这样劣质代码比比皆是，有的还以教程、例程的形式存在，很可能误导初学者。因此，对于初学者有以下建议：1）请从一本好的专业书籍开始学习 JavaScript，而不是网上教程，例如本书或者阅读材料中的。2）所有程序必须吃透，知道其运行过程，明白其质量高低；切不可只要其能运行，就不论其它。对于代码，程序员要知其然，还必须要知其所以然。3）coding，coding，and coding，编写代码是学习任何编程语言的最佳方式。

[2] ECMA 制定的标准语言称之为 ECMAScript，严格说来 JavaScript 语言是它的一个方言，也就是一个子集。此外的方言还有 Microsoft 公司的 Jscript，以及 Adobe 公司的 ActionScript。有时，在不严格的情况下，人们会把这几种语言都称作 JavaScript。

JavaScript 引擎竞赛带来的 JavaScript 技术创新

JavaScript 解释器也称 JavaScript 引擎，它负责解释执行 JavaScript 程序，是现代浏览器当中的重要组件。早期的 JavaScript 语言因为运行效率差，被诟病为玩具语言（toy language），其原因就是 JavaScript 引擎效率低。

随着 Web 应用的日益流行，浏览器 JavaScript 程序的执行效率日渐重要，各大浏览器厂商纷纷开发、更新自己的 JavaScript 引擎，以期提高浏览器性能，改善用户响应时间，进而吸引用户，占据更大市场。

自 2008 年起，浏览器市场竞争进入白热化，JavaScript 引擎更是竞争的焦点，从 Google Chrome 声称其 JavaScript 引擎比其它浏览器高效开始，各大浏览器，如： Firefox、IE、Opera、Safari 均奋起反击，纷纷表示自己新版本的 JavaScript 引擎后来居上。例如：Webkit 的 Squirrelfish 和 Firefox 的 TraceMonkey，性能都优于 Google Chrome 的 JavaScript 引擎。

Google 迅速发起反击，随 Chrome 2 发布了划时代的 JavaScript 引擎 V8，V8 将 JavaScript 的执行效率提高了至少一个数量级，其运行效率已经接近编译执行的二进制代码。

V8 成就了一代传奇，它的高效率使其不仅仅用于普通脚本语言使用的环境。2009 年开始大热的 Node.js 利用 V8 的出色性能和事件驱动式 I/O，用 JavaScript 语言构造了 Web 服务器，并获得了很大成功。令人吃惊的是，Node.js 构造的 Web 服务器，其并发服务能力和响应时间，竟然优与传统的、使用编译型语言开发的 Web 服务器，如 Apache、Nginx、IIS 等等；特别在高并发的情况下，Node.js 更是远远胜出，表现极为出色。

这样的结果，颠覆了长期以来人们对脚本语言执行效率低的印象，在学术界和产业界都引起了轰动，并促生了一系列技术创新，如 CoffeScript、Sass、Less、Haml，……，这些技术方兴未艾，大有开启 Web 前端技术新时代的趋势（本书 14 章将对这些内容做简单介绍）。

然而，JavaScript 引擎竞赛远未就此终结，苹果(Safari)的 Nitro，Mozilla (Firefox) 的 JägerMonkey 等等 JavaScript 引擎都正在赶上甚至超越 Google V8，这不仅仅使得 Web 应用的前端技术有了更加坚实的基础，也使得基于 JavaScript 的技术前景更加明朗，有可能走向更为广阔的天地。

7.3.1 JavaScript 语言特点

JavaScript 是一种弱类型、动态类型脚本语言。它是一种瑞士军刀、万能式的语言，可用它进行不同范式的编程，包括：结构化编程（Procedure Programming）、面向对象编程（Object-Oriented Programming）和函数式编程（Functional Programming）等。它的多用途和快速发展的历史，使得其语言、语法要素中优劣相间、好坏参杂。

因此，本书介绍当中，根据业界实践总结，摒弃了容易造成编程错误的语法和语言要素，并省略了不常用的部分，仅仅介绍编写整洁、高效的 Web 前端代码所必需的语法和语言要素。更多 JavaScript 语言内容，请参考阅读材料 definitive javascript。

7.3.2 给网页添加 JavaScript

用 JavaScript 脚本给网页增加前端动态性，首先需要给相应的网页增加 JavaScript 代码，告诉浏览器这些代码用于该网页。源代码 7-1 和源代码 7-2 就是图 7-4 中响应用户点击弹出对话框 HTML 代码和 JavaScript 代码。

源代码 7-1 图 7-4 HTML 代码

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript" src="event_driven.js"></script>
……
</head>
<body>
……
    <button type="button" onclick="openDialog();">
        打开对话框
    </button>
……
```

源代码 7-2 图 7-4 JavaScript 代码（event_driven.js）

```
function openDialog(){
    alert("对话框打开了!");
}
```

HTML script 元素（标签）将 JavaScript 脚本代码绑定到当前网页。script 元素的 src 属性给出了脚本代码的 URL 地址，该 URL 可以是相对地址，如源代码 7-1，也可以是在当前 Web 应用中的绝对地址，或者是包括服务器域名在内的完整 URL。

例如，当源代码 7-1 的 HTML 位于"/examples/event_driven.html"时，源代码 7-1 中 src 的 URL 也可以写作"/examples/event_driven.js"。此外，网页上使用的 JavaScript 脚本并不一定与网页来自同一个 Web 服务器[3]，完全有可能在网页上加载来自其它 Web 服务器的 JavaScript 脚本，此时需要使用完整的 URL，例如："http://my.ss.sysu.edu.cn/MWP/examples/event_driven.js"。

非侵入式 JavaScript：虽然也可以使用中的方式，直接在 HTML 代码当中嵌入 JavaScript，如源代码 7-3。但是，这样做违反了软件工程的关注点分离（Separation of Concerns）和模块化（modulization）的原则，不推荐使用。正确的做法应该是使用源代码 7-1 的方式用 script 元素的 src 属性，将独立的 JavaScript 脚本文件绑定到当前网页上。这种将 JavaScript 代码与 HTML 代码分离的风格，称为非侵入式 JavaScript（Unobtrasive JavaScript），参考 3.1.1 非侵

[3] 这是 Web Mashup 中的 Dynamic Script 技术的关键，后面的章节将详细介绍。

入式 CSS,后面章节还将就此话题做进一步介绍。

源代码 7-3 在 HTML 代码中直接嵌入 JavaScript 代码（不推荐）

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
……
<script type="text/javascript">
    function openDialog(){
        alert("对话框打开了!");
    }
</script>
</head>
<body>
……
    <button type="button" onclick="openDialog();">
        打开对话框
    </button>
……
```

7.3.3 JavaScript 调试工具

跟踪调试程序，发现其中的问题，是程序员的基本功之一。4.3.2 节中的介绍的 Firefox 浏览器 Firebug 插件和 IE 浏览器自带的调试工具（IE 7 以上版本）都可以用来调试 JavaScript 脚本。它们都能够捕获当前网页 JavaScript 代码执行中的错误，也可以对当前网页的 JavaScript 进行调试——设置断点，逐句、逐过程（方法）、逐断点步进，查看当前各个变量的值，和方法的调用堆栈。图 7-5、图 7-6 分别是使用 Firebug 和 IE 调试工具的情况。

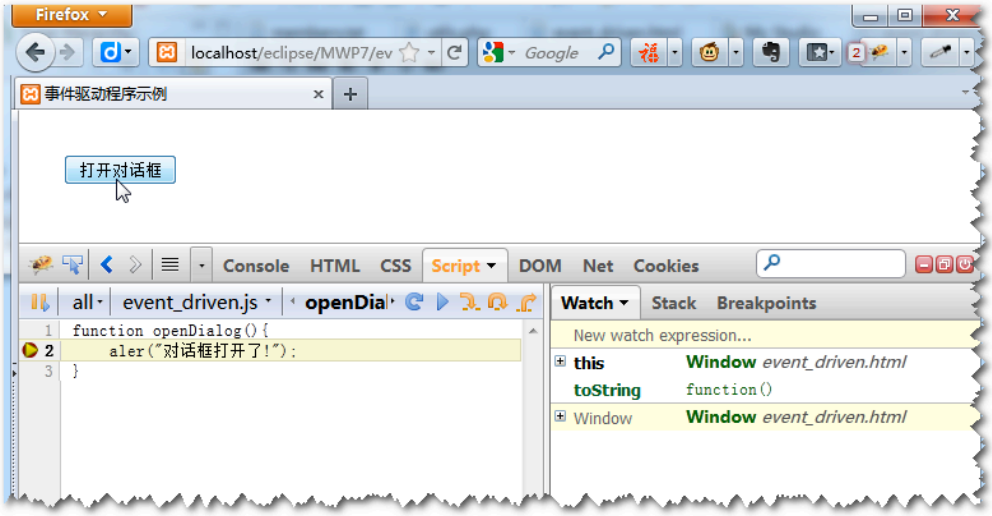

图 7-5 使用 Firebug 调试工具调试 JavaScript

（下栏是 Firebug，其左边显示了被调试脚本源代码，和程序员设置的断点。图中程序正执行到断点处，程序员可以选择源代码上方的各种按钮，步进控制程序执行。Firebug 右边显示了当前可见各个变量的值。）

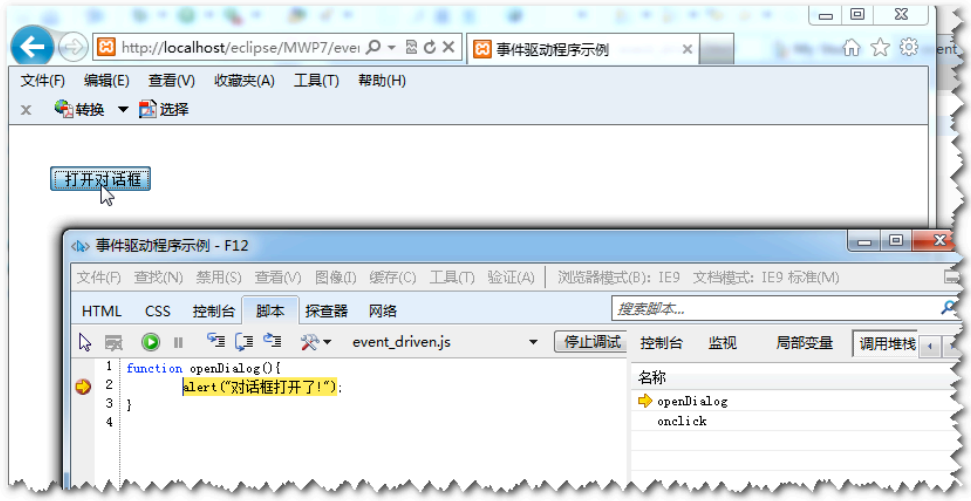

图 7-6 使用 IE 调试工具调试 JavaScript

（IE 调试工具与 Firebug 及其类似，其各部分功能几乎完全相同，参考图 7-5）

巧用控制台(console):Firebug 和 IE 调试工具都包括了控制台(console，见图 7-5、图 7-6 源代码上方标签栏)。控制台中不仅能够看到当前网页 JavaScript 代码出现的错误和其它调试信息，还可以实时输入 JavaScript 脚本，在当前网页的上下文（进程空间）中执行。这也就是说，程序员可以在这里输入 JavaScript 脚本，实时访问、操作当前网页上所有可见的 JavaScript 变量、对象、函数等等。这对于调试 JavaScript 代码十分有用，**对于初学者编写尝试性程序，更是最佳练习方式**。

图 7-7 展示了在控制台上输入并执行脚本，激发网页中按钮的点击（click）事件，就好像用户点击了按钮一样。这里只是展示控制台的用法，代码的具体含义和执行机制请暂时忽略。

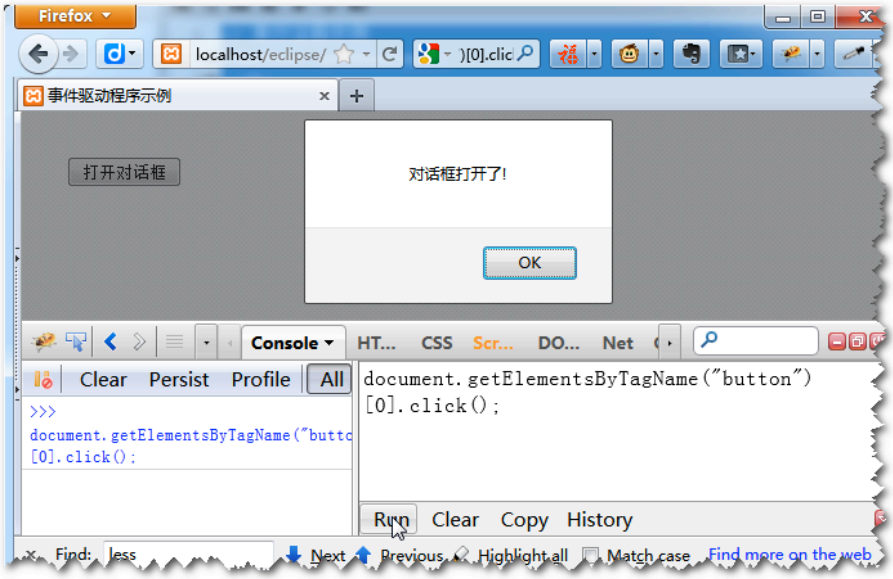

图 7-7 使用控制台实时执行 JavaScript 脚本

（在 Firebug 右边区域输入代码后，点击“run”，代码就会在网页进程的当前上下文中执行。这里就激发了点击按钮事件，导致对话框打开了。Firebug 左边区域右边显示了代码执行的各种信息。

限于篇幅，更多调试功能使用和 JavaScript 程序调试技巧，请阅读本章阅读材料，或自行网上搜索。

JavaScript 编程开发工具

JSLint 是一个静态代码分析工具，能够检查 JavaScript 代码中的语法错误和编码不规范。

Aptana 是一个基于 Eclipse 优秀的 Web 工程集成开发环境，可以支持多种前、后端编程语言，包括 HTML、CSS、JavaScript 和 PHP、Java、Ruby 等等。

自测题

1. 从用户点击网页上的链接开始，到新的网页在浏览器里显示为止，浏览器都进行了哪些工作？
2. 网页上的 JavaScript 程序在哪里执行，由谁执行，如何执行？
3. 什么是事件驱动的程序，它和普通的过程式程序有和区别，一般在哪里使用？

4. Web 前端编程的基本特点是什么？
5. 从程序语言的分类来说，JavaScript 属于哪种语言，和 PHP 是同一类吗？
6. 什么是脚本语言，脚本语言一般用在什么场合？
7. 如何给一个网页增加 JavaScript 代码，都有什么方式，应该使用哪种方式，为什么？
8. 如何调试网页上的 JavaScript 代码，请说明一种常用调试工具的用法。

7.4 JavaScript 基础

从上面源代码 7-2 和图 7-7 中输入到控制台的代码来看，JavaScript 的总体语法风格与 C、C++、Java 类似，以分号作为行分隔符，使用大括号来定义代码块。本节开始具体学习 JavaScript 的基本语法和语言要素。

行末分号：JavaScript 的行末分号是可选的，语句之间可以通过换行分隔，而无需行末分号。一行中写多条语句时，必须用分号分隔。本书遵从 JavaScript 社区绝大部分程序员的习惯，语句末尾一律使用分号，而不论是否有换行作为分隔。

7.4.1 注释

JavaScript 的注释和 C、C++、Java 等语言一样，有行末（单行）注释和多行注释。行末跟在"//"之后的内容，为行末（单行）注释，而包含在"/*"和"*/"之间的内容是多行注释，可跨越多行，参见源代码 7-4。

源代码 7-4 JavaScript 注释示例

```
// 单行注释示例
function openDialog(){
    /*
     * 多行
     * 注释示例
     */
    alert("对话框打开了!"); //行末注释示例
}
```

7.4.2 输出

JavaScript 语言运行在浏览器内，为网页增加前端动态性，其语言目的和运行环

境，使得它没有很多编程语言具有的 print 之类的输出语句。不过，JavaScript 可以通过浏览器弹出框输出信息。这样的弹出框有 3 种，简单弹出框、确认弹出框和输入对话框，分别由 JavaScript 系统函数 alert、confirm 和 prompt 产生，源代码 7-2 是 alert 生成简单弹出框（图 7-4 右）的代码示例，confirm 和 prompt 与之类似，略有差异，将在 7.6.4 讲述。

此外，在 JavaScript 程序中可以使用 console 对象来输出调试信息，这些信息会被输出到调试工具，如 Firebug 和 IE 调试工具的控制台上，在调试程序时非常有用。参考源代码 7-5 和其运行结果图 7-8。

源代码 7-5 使用 console 输出调试信息[4]

```
function openDialog(){
    console.log("openDialog函数被调用了");
    alert("对话框打开了!");
}
```

图 7-8 源代码 7-5 运行结果

除了 console.log 方法，还可以通过 console 对象的 info、warn 和 error 方法生成不同级别（消息、警告、错误）的调试信息。这些级别在控制台中会有不同的显示，并可以过滤分类显示，对于程序的调试维护，特别有用。

[4] 注意，在使用 console 对象时，通常需要先判断其是否存在，以免造成错误。参考 http://stackoverflow.com/questions/398111/javascript-that-detects-firebug。

图 7-9 调试信息级别（左上至右下，info 信息、warn 警告、error 错误）

7.4.3 类型

JavaScript 有 8 种内置类型，参见表 7-1，本节将讲述除数组（array）、对象（object）和函数（function）之外的几类，数组和函数在 7.6 节讲述，对象在第 9 章讲述。

表 7-1 JavaScript 内置数据类型

类型	描述	示例
number	数字，整数或者浮点数	42、-17、0、3.14、2.4e-6、NaN
boolean	布尔值	true、false
string	文本、字符串	"hello world"、'你好'
array	数组，一组可通过自然数下标索引访问的数据	[12, 17, '你好', 0]
object	对象，包含属性和行为的实体	{name: "张三", age: 24}

function	函数，一组可以执行的语句	function openDialog(){
		  alert("对话框打开了");
		}

null	空	null
undefined	未定义	undefined

typeof 函数——确定值的类型：JavaScript 提供了系统函数 typeof，返回值的类型。例如：typeof("hello world")的返回"string"，typeof(1.0)返回"number"。

7.4.4 数字与数学运算

JavaScript 没有真正的整形数：JavaScript 使用 Number 类型同时表示整数和浮点数。实际上，JavaScript 中并没有真正意义上的整型数，所有的数字都表示为 64 位浮点数。只是在输出时，当浮点数没有小数部分时，省略了小数点和零。

JavaScript 的 Number 类型还包括了几个特殊的常量用来表示一些特殊的数值，参见表 7-2。

表 7-2 Number 类型常数

常数	描述
number.MAX_VALUE	系统可表示的最大正有理数
number.MIN_VALUE	系统可表示的最小小数
number.NaN （或者 NaN）	不是数字
number.NEGTIVE_INFINITY	表示数值已经超出系统能表示的最大负数
number.POSITIVE_INFINITY	表示数值已经超出系统能表示的最大正数

这些常数里面，NaN 是一个非常特殊的值，它是 Number 类型，但是却代表了”不是数字”的含义。**任何数学运算，如果有操作数无法转换为 Number，那么其结果就是 NaN**。例如：3 / “2”和 3 / “2.0”的结果都是 1.5，而 3 / “2x”的结果就是 NaN。

JavaScript 中算术运算和 C、C++、Java 等程序设计语言，以及 PHP 语言一样，算术运算符如下表所示。

表 7-3 JavaScript 算术运算符

运算符	解释
+	加法
-	减法
*	乘法
/	除法
%	取余

JavaScript 通过 Math 对象提供了常用的数学常数和函数，参见表 7-4 和表 7-5。

表 7-4 JavaScript 数学常数

常量	值
Math.PI	3.14159……，圆周率 π

Math.E	2.71828……，自然对数底 e
Math.LN2	0.69314……，2 的自然对数

表 7-5 JavaScript 数学函数

函数	解释
Math.abs(x)	绝对值
Math.ceil(x)、Math.floor(x)	天花板（向上取整）、地板（向下取整）
Math.cos(x) 、 Math.sin(x) 、Math.tan(x)	相应的三角函数
Math.log(x)	自然对数（也可给出底，计算任意底的对数）、10 为 3 的对数
Math.min(x, y, …)、Math.max(x, y, …)	最小值、最大值
Math.pow(base, exponent)	幂函数
Math.random()	随机数
Math.round(x)	四舍五入
Math.sqrt(x)	平方根

7.4.5 变量

JavaScript 的变量和其它高级编程语言一样，都是一块命名的内存空间，用来存放值。JavaScript 的变量名由英文字母、数字和下划线组成，第一个字符必须是字母或者下划线。变量名大小写敏感。

作为动态类型语言，JavaScript 和 PHP 一样，变量无需类型声明，即可直接赋值使用。 JavaScript 变量可使用 var 关键字声明和赋值使用，也可以直接赋值使用。直接赋值使用也称为隐式变量声明（implicit variable declaration）。源代码 7-6 展示了两种变量声明与赋值的方式，其差异在于变量的作用域不同，参考 7.6.1 节。

源代码 7-6 变量示例

```
var temp = 1;
alert(temp);          // 弹出框 1
global_temp = 0;      // 隐式声明
alert(globel_temp);   // 弹出框 0
```

与 PHP 一样，作为动态类型语言，JavaScript 的变量前后可以赋予不同类型的值，参考源代码 7-7。

源代码 7-7 变量示例

```
var answer = 2010;
alert(typeof answer);    // number
```

```
answer = "今年";
alert(typeof answer);    // string
```

和很多高级编程语言一样，JavaScript 也提供了算术运算和赋值表达的简写形式，运算赋值操作符，参见表 7-6。

表 7-6 JavaScript 运算赋值操作符

运算符	示例	等价表达式
++	x++	x = x + 1
--	x--	x = x – 1
+=	x += y	x = x + y
-=	x -= y	x = x – y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

7.4.6 字符串

JavaScript 中字符串操作非常重要，因为不论是用户在网页上的输入，还是网页元素的各种属性，都是字符串类型。因此，JavaScript 语言同 PHP 语言一样提供了强大、丰富的字符串处理机制，PHP 字符串参考 5.3.8 节。

a) 字符串表示和下标索引

JavaScript 字符串类型（string）用来表示一串字符。和 PHP 一样，JavaScript 没有字符类型，单个字符就是长度为 1 的字符串。和 PHP 一样，JavaScript 的字符串既可以使用英文双引号”“括起来，也可以使用英文单引号”括起来。不同于 PHP 的是，JavaScript 没有翻译式字符串（Interpreted String）。

和其它语言一样，字符串内特殊字符用反斜杠”\”来转义表示，例如：”\n”表示换行，”\\”表示反斜杠”\”，”\’”表示单引号，”\”“表示双引号。

源代码 7-8 JavaScript 字符串示例

```
var str1 = “Hello world”;
var str2 = “Hello\nworld”;
alert(str1);     // 弹出”Hello world”
alert(str2);     // 弹出”Hello” 换行 “world”
alert(str1[0]);  // 弹出”H”
```

源代码 7-8 中 str2 的下标索引与对应值的情况如表 7-7 所示。

表 7-7 str2 下标索引与对应值

索引	0	1	2	3	4	5	6	7	8	9	10
值	H	e	l	l	o	\n	w	o	r	l	d

不同于 PHP，**JavaScript 的多行字符串，一定要用转义符"\n"，而不能直接在代码里表示**（对比源代码 5-15 和源代码 7-8）。这是因为 PHP 的语句一定要使用分号作为结束，而 JavaScript 语句结尾的分号可选，源程序中的换行会被 JavaScript 解释器认为是下一条语句。

b) 字符串运算

JavaScript 和其他大部分高级程序语言一样，使用加号"+"作为字符串拼接操作符。当加号操作符的两个操作数有一个为字符串类型时，加号表示字符串拼接操作，非字符串操作数会隐式转换为字符串。

表 7-8 JavaScript 字符串运算示例

表达式	值
1 + "2"	"12"
1 + "3 个和尚"	"13 个和尚"
(1+3) + "5"	"45"

解析数字：初学者常犯的错误是在进行算术运算时，直接使用了用户在界面的输入，或者是 HTML 元素的属性。这些值其类型都是字符串，**在进行算术运算前必须要解析为数字！**

因为 JavaScript 语言只有 Number 一种数字类型，所以没有使用强制类型转换的语法，类似(Number)"2.8"，来将字符串转换为数字；而是提供了 parseInt 和 parseFloat 两个系统函数来将字符串解析为数字

表 7-9 JavaScript 解析数字示例

表达式	值
parseInt("2")	2
parseInt("2.8")	2
parseInt("2.8 个和尚")	2
parseInt("　　2.8 个和尚")	2
parseInt("")	NaN
parseInt("有 2.8 个和尚")	NaN
parseFloat("2.8")	2.8
parseFloat("2.8 个和尚")	2.8

parseFloat(“2.8p3 和尚”)	2.8
parseFloat(“2.00000”)	2

c) 字符串操作

不同于 PHP 以函数的形式提供了大部分对字符串的操作（参考 5.3.8 c) ），JavaScript 字符串操作采用了面向对象的形式。JavaScript 中的字符串都是对象（object），拥有自己的属性和方法，我们通过这些属性和方法来操纵字符串。

字符串的一个重要属性是 length，表示了字符串的长度。

源代码 7-9 字符串长度示例

```
var str3 = "你好world";
alert(str1.lenth);        // 11
alert(str3.length);       // 7
```

字符串常用的方法如表 7-10 所示，表中假设通过字符串对象 str 使用这些方法。

表 7-10 JavaScript 字符串常用方法

方法	用途
charAt(index)	在 index 处的字符，等于 str[index]
charCodeAt(index)	给出 index 处字符的编码（数字）
String.fromCharCode(code)	静态方法，将 code（数字）转换为对应的字符
indexOf(searchStr) indexOf(searchStr, fromIndex)	在 str 中从 formIndex（缺省为 0）开始查找 searchStr，找到则返回其第一次出现的位置；未找到返回-1
split(delimeter) split(delimeter, howMany)	将 str 以 delimeter 为分隔符，截断为多个字符串，将这些字符串前 howMany 个组成一个数组返回；无 howMany 参数，则返回所有
substring(start) substring(start, stop)	返回 str 从 start 到 stop 的部分，stop 缺省为 str 结束
toLowerCase()	将 str 转换为全小写字符
toUpperCase()	将 str 转换为全大写字符

源代码 7-10 给出了使用这些方法的示例。

源代码 7-10 字符串方法使用示例

```
var str = "你好World Wide Web!";
alert(str.charAt(0));           // "你"
alert(str.charCodeAt(0));       // 20320 (运行环境为UTF-8编码时)
```

```
alert(String.fromCharCode(20320)); // "你"
alert(str.indexOf("我"));           // -1
alert(str.indexOf("W"));        // 2
alert(str.split("W", 3));       // ["你好", "orld ", "ide "]
alert(str.substring(2));        // "World Wide Web!"
alert(str.toUpperCase());       // "你好WORLD WIDE WEB!"
alert(str);                     // "你好World Wide Web!"
```

JavaScript 字符串不可改变！：字符串在 JavaScript 和许多高级语言一样，例如：Java、C#等等，是不可改变的（immutable）。也就是说，一经产生，字符串本身的值就再也不会发生改变。变量赋值为字符串后，除非重新赋值，其值不变，参考源代码 7-11。PHP、C、C++等语言则不同，字符串本身的值是可以改变的，参考源代码 7-12。

源代码 7-11 JavaScript 字符串 immutable 示例

```
var str = "Hello";
str[0] = "W";
alert(str);   // "Hello"
```

源代码 7-12 PHP 字符串 mutable 示例

```
<?php
$str = "Hello";
$str[0] = "W";
echo $str;    // "Wello"
?>
```

程序语言学习的触类旁通

各种程序语言，在许多地方都有类似之处。例如，字符串操作，大部分语言都提供了分割/拼接字符串、查询子字符串，大小写转换等方法，方法的名称和参数也都大同小异。很显然，这是因为不论哪种语言，在进行字符串操作时，都需要这些方法。因此，学习一门新语言时，有关这些操作的知识显然是可以借鉴、迁移的。

同时，通过对各种语言设计细节上差异的比较，能够更加深入了解语言背后的设计思想，有助于更好掌握和使用语言。例如，大部分面向对象的语言中，如：JavaScript、Java、C#等，字符串都是 immutable 的；而大部分结构化编程语言，字符串都是 mutble 的，例如：C、C++、PHP 等*。

这是因为，面向对象的程序，其逻辑抽象是一组互相交互的对象，编写程序的重点在于构造正确的对象和对象间的关系，将字符串设计为 immutable，有助于安全地在对象间进行分享，而不会因为某个对象对字符串的更改，给其它对象带来意外的变化。而结构化程序，其逻辑抽象是数据流和一组对数据的加工，将字符串设计为 mutable，使得加工写起来更加方便、简洁。

*注：JavaScript、C++、PHP 等语言既可用于结构化编程，也可用于面向对象。C++、PHP 字符串 mutable 的设计，是出于对结构化编程的考虑。

7.4.7 null 和 undefined

JavaScript 同时有两个用来表示“没有”的类型——null 类型和 undefined 类型。这两个类型各自都只有一个值，和其类名称一样，分别是 null 和 undefined。

null 表示变量的值“没有”，为“空”，或者说这个变量没有值。undefine 表示的是这个变量没有定义，不存在。undefinded 是变量根本不存在，null 是变量存在但是没有值。

C、C++、Java 等静态类型语言的变量必须声明类型后，才能够使用，显然不可能有 undefined 的情况。这些语言的未初始化赋值的变量值为 null。动态类型的语言，变量可以不用声明，直接使用，有可能出现 undefined 和 null 两种情况。不过，绝大部分的动态语言并没有区分这种差异，同时有 null 和 undefined 是 JavaScript 的一大特点。

自测题

1. JavaScript 中有 print 语句吗，为什么？JavaScript 常用什么输出消息？
2. JavaScript 中哪个数据类型用以表示 C 语言中的 int，哪个用以表示 float？
3. 请给出 JavaScript 语句，声明并初始化变量 gpa 的值为 2.8
4. JavaScript 中如果一个变量未经声明就直接赋值，会是什么结果？

5. JavaScript 中如果一个变量未经声明就输出它的值，会是什么结果？
6. 给出下列 JavaScript 表达式的值：
   a) 12 - 2 -3
   b) 12 - (2 - 3)
   c) 22 + 4 * 2
   d) 4 * 3 / 2 + 5.5 * 2
   e) 26 % 10 % 4 * 3
   f) “2 + 2 “ + 3 + 4
   g) 3 + 4 + “2 + 2”
   h) 10.0 / 2 / 4
7. 给出下列 JavaScript 表达式的值：
   a) Math.abs(-1.6)
   b) Math.pow(2, 3)
   c) Math.ceil(9.1)
   d) Math.max(4, 3, 7, 9)
   e) 7 - 2 + Math.log(Math.pow(Math.E, 5))
8. 如果变量 str 的声明语句如 var str = “长城=Great Wall”，请给出下面表达式的值：
   a) str.length
   b) str.toLowerCase()
   c) str.substring(3, 7)
   d) str.indexOf(“Great”)
   e) str.charAt(3)
   f) str.substring(str.indexOf(“a”))

   完成上面计算之后，str 的值是什么，为什么？
9. JavaScript 的 null 和 undefined 值都是什么类型，各自表达什么含义，有什么区别？

7.5 JavaScript 逻辑与控制

逻辑控制是表达程序执行逻辑的关键，本节讲述 JavaScript 逻辑运算符、逻辑表达式，条件分支语句和循环语句。

7.5.1 布尔值

和其他编程语言一样，JavaScript 语言的 boolean 类型有两个值，true 和 false。JavaScript 源程序中对 true 和 false 的大小写敏感，只有 true 和 false 才是合法的 boolean 值，这一点和 PHP 不同（参考 5.3.9）。

7.5.2 逻辑运算符

JavaScript 逻辑运算符和 PHP 完全一样（参考 5.3.9），见表 7-11。

表 7-11 JavaScript 逻辑运算符

操作符	解释
==	相等（忽略类型）
!=	不等（忽略类型）
===	相等（考虑类型）
!==	不等（考虑类型）
>	大于
<	小于
>=	大于等于
<=	小于等于
&&	与
||	或
!	非

JavaScript 在进行比较的时候，大部分操作符，如“==”、“!=”、“>”、“>=”等等，会进行类型转换。对于不同类型的操作数，先将它们转换到同一类型，再进行比较。当比较不同类型操作数，又无法类型转换时，则返回 false。运算符“===”、“!==” 不同，它们会先考虑比较对象的类型，类型不同则为 false，类型相同再来看值是否相等，参见表 7-12。

表 7-12 JavaScript 比较运算示例

操作符	结果
10 < “42”	true
10 < “42 人”	false
10 > “42 人”	false
10 == “42 人”	false
42 == “42”	true
42 == “42.0”	true
42 === 42.0	true
42 === “42”	false

和 PHP 一样，JavaScript 所有数据类型都可以转换为 Boolean 类型，可以方便地用在逻辑表达式中。但是，JavaScript 的转换规则和 PHP 却有所不同，JavaScript 中以下值被转换为 false，其余值均被转换为 true。

- 0
- NaN
- 空字符串“”

- null
- undefined

空数组、“0”：PHP 中空数组和字符串“0”、“0.0”都被视为 FALSE（参考 5.3.9），而在 JavaScript 中它们都是 true。

值得注意的是 null 和 undefined 都被认为是 false。因此，表达式 null == undefined 的值为 true；又因为 null 和 undefined 的类型不同，所以，表达式 null === undefined 的值为 false。

另一个值得注意的是 NaN。NaN 不仅仅本身被视为 false，**表达式 NaN == NaN 和 NaN=== NaN 的值都是 false**！因为 NaN 表达的不是一个真正意义上的值，而是指一个数学表达式中有非数字，无法计算出结果。表达式 2 + “3 人” 和 3+”5 天”的值都是 NaN，而显然 2 + “3 人” 不等于 3+”5 天” ，所以无论表达式 2 + “3 人” == 3+”5 天”，还是 2 + “3 人” === 3+”5 天”都应该是 false。

a）运算优先级

JavaScript 中各种运算的优先级如表 7-13，表中上面行的优先级较高，同一行内操作符优先级相同。

表 7-13 JavaScript 运算优先级（降序）

类别	操作符
成员、索引操作符	. []
方法（函数）调用、对象创建	() new
逻辑非、负号、自增、自减	! - ++ --
乘、除、取模	* / %
加、减	+ -
关系比较	< > <= >=
相等、不等比较	== != === !==
逻辑与	&&
逻辑非	||
赋值	= += -+ *= /= %=

7.5.3 控制语句

同 C、C++、Java 和 PHP 语言类似，JavaScript 提供 5 种基本控制语句：if/else、for、while、do/while、for…in、switch。本节将介绍其中前 4 个，for…in 将在对象一

节介绍。switch 使用较少，且其用法类同于 C、C++、Java 等语言，此处不再赘述，有关细节可查询 W3Schools 教程[5]。

a) if/else 语句

if/else 语句和其它语言类同，用于构造逻辑分支语句，允许程序在运行时依据判断条件的不同值，选择执行不同的语句分支。其使用示例如源代码 7-13。

源代码 7-13 if/else 语句示例

```
if(a == 5){
    alert("a equals 5.");
}else if(a == 6){
    alert("a equals 6.");
}else{
    alert("a is neither 5 nor 6.");
```

b) for 语句

for 语句也与其它语句类似，用来构造循环执行语句。for 语句头包含分号分隔的 3 个子句，分别初始化循环变量语句、循环条件表达式和每次循环后更新循环变量的语句。基本的模板如源代码 7-14，使用示例如源代码 7-15。

源代码 7-14 for 语句模板

```
for(初始化循环变量; 循环条件表达式; 更新循环变量){
    循环体语句……
}
```

源代码 7-15 for 语句示例

```
for(var i = 1; $i <=5; $i++){
    alert(i + " squared is " +  i * i + ".");
}
```

c) while 和 do/while 语句

while 和 do/while 语句是另外一种形式的循环执行语句。与 for 语句不同，其语句头只有测试循环条件的表达式，没有初始化和更新循环变量的子句。while 的语句模板如源代码 7-16，do/while 的语句模板如源代码 7-17。do/while 不同与 while 在于，不论测试条件一开始的真假，do/while 的循环体都会至少被执行一次；while 的

[5] http://www.w3schools.com/js/default.asp

循环体在测试条件一开始即为假的情况下，一次也不会被执行。

源代码 7-16 while 语句模板

```
while (循环条件表达式){
    循环体语句……
}
```

源代码 7-17 do/while 语句模板

```
do {
    循环体语句……
}while(循环条件表达式)
```

自测题

1. JavaScript 中为何同时有 == 和 === 操作符，它们有什么区别？
2. 下面这些值，哪些是 true，哪些是 false？
   a) 0.000
   b) “0”
   c) “” （空字符串）
   d) [ ] （空数组）
   e) “false”
   f) parseInt(0.97)
   g) 42 / “ten”

3. 请用更加简洁的表达式改写下面的代码

```
if(amount == null || amount == undefined){
    var warning = document.getElementById("warnngarea");
    warning.innerHTML = "请输入数值！";
}
```

7.6 JavaScritp 进阶

本节介绍更多 JavaScript 语法，简单介绍变量作用域、数组、函数和弹出框。关于作用域、数组和函数的深入介绍，参考第 9 章。

7.6.1 变量作用域

和许多程序语言一样，JavaScript 的变量分为全局（global）作用域和局部（local）作用域。在函数内部使用 var 关键字声明赋值的变量，为局部变量。局部变量仅仅

在其声明时所在函数中有效，函数之外，不可访问。在所有函数外声明的变量和在函数内直接声明赋值（隐式变量声明，不用 var 关键字）的变量，为全局变量，可以在任何地方被访问。不同于 PHP，在函数内使用全局变量时，无需先使用 global 语句声明。

源代码 7-18 变量作用域示例

```
var a = 2;          // 全局变量

function scopeExample(){
    alert(a);       // 弹出2
    a = 2 * 2
    b = 3;          // 隐式变量声明，b为全局变量！
    var c = 5;      // 局部变量
    // 这里a为4，b为3，c为5
}
scopeExample();
//  这里a为4，b为3，c为undefined
```

同 PHP 一样，JavaScript 的最小作用域也是函数作用域，函数内控制结构内声明的局部变量，在控制结构外，函数内可以被访问。这一点和大多数静态类型的语言不同（参考 5.5.2）。

源代码 7-19 函数作用域示例

```
function scopeExample2(){
    for(var i = 0; i < 10; i++){
        var a = i * i;
    }
    // 这里i和a依然能够被访问
    alert(i);
    alert(a);
}
scopeExample2();
// i 和 a 不能被访问，undefined
```

循环变量不要忘记 var：循环变量如源代码 7-19 中的变量 i，如果忘记使用关键字 var 声明，会成为全局变量！这是个常见错误。

慎用全局变量：绝大多数编程语言都认为要慎用全局变量，因为它很容易被某些代码“不注意地”改动。使用全局变量还违反低耦合的原则，使用全局变量的模块间会经由它产生相互依赖。网页前端 JavaScript 代码中更要尽量避免使用全局变量，因为一个网页可能同时使用若干个脚本，包括来源不同的脚本，而所有

这些脚本中的全局变量，都同时可以被其它脚本访问，这非常容易造成名称冲突和变量值被错误修改的问题。

实际上，JavaScript 的变量作用域与 PHP 的变量作用域并不完全相同，在处理嵌套函数时它们有很大不同，参考 9.2.2 节。

7.6.2 数组

JavaScript 数组与 PHP 的数组很类似（参考 5.5.4），其特点有：

- 容量可以自动调整
- 使用自然数作为下标（从 0 开始），可以不连续赋值
- 同一数组的多个元素的类型可以不同，可以是任何类型

源代码 7-20 数组示例

```
var a = [];             //空数组
a[0] = 23;              //23存储在数组下标为0处

var a2 = ["some", "strings", "in", "an", "array"];
a2.push("Ooh~");        //在数组末尾添加"Ooh~"
alert(a2.length);       //6
a2[100] = "Yeah!"       //将数组大小变为101，并给下标100处赋值"Yeah!"

alert(a2[99]);          //undefined
alert(a2.length);       //101
```

事实上，JavaScript 数组也是对象（object，参考第 9 章对象部分）。数组对象有 length 属性，值为数组大小。此外，数组对象还有一系列有用的方法，用来操纵数组元素（数据），参见表 7-14

表 7-14 JavaScript 数组方法

方法	用途
concat(array1, …, arrayN)	将多个数组拼接成为一个
join() join(separator)	将数组元素以 separator 为分隔符拼接成为一个字符串
pop()	弹出数组最后一个元素
push(value) push(value1, …, valueN)	将一个或者多个值压入数组
shift()	从头部取出一个元素，并依次向前移动其余元素
unshift(value) unshift(value1, …, valueN)	向头部添加一个元素，并依次向后移动其余元素
reverse()	改变当前数组，将其顺序翻转
sort()	改变当前数组，将其排序

slice(startIndex) slice(startIndex, endIndex)	返回当前数组的子数组，子数组从 startIndex 开始，到 endIndex 结束，endIndex 缺省为数组长度
splice(index, count, value1, …, valueN)	将当前数组从 index 起 count 个元素删除，并更换插入给定的多个值（value1, …, valueN）

这些方法中，join 方法是字符串 split 方法的逆运算（参考表 7-10）。slice 和 spice 方法则是非常好用的两个方法，能够很便捷地切分、更改数组。这两个方法的使用示例见源代码 7-21

源代码 7-21 JavaScript 数组方法使用示例

```
var userInfo = "张三|男|28|13860000660|zhangsan@mail.cn";
var userInfoArray = userInfo.split("|");
// [张三, 男, 28, 13860000660, zhangsan@mail.cn]

var nameAndGender = userInfoArray.slice(0,2).join(", ");
alert(nameAndGender); // 张三, 男

userInfoArray.splice(3, 1, "中山大学", "18509087532");
// [张三, 男, 28, 中山大学, 18509087532, zhangsan@mail.cn]

alert(userInfoArray.join("|"));
// 张三|男|28|中山大学|18509087532|zhangsan@mail.cn
```

7.6.3 函数

如同 C、C++、Java 和 PHP 一样，JavaScript 的函数由一组语句组成，封装了特定的程序逻辑，可以被多次调用（执行）。一个 JavaScript 函数由函数名、参数列表和函数体构成。JavaScript 函数名可选，没有函数名的函数称为匿名函数。可以通过函数名，或者指向函数的引用调用函数（还有其它方式，参考第 9 章），参数列表给出了调用函数时需要的参数，函数体是一系列语句，完成函数特定的程序逻辑，函数可以有一个返回值。

JavaScript 和 PHP 一样，声明函数时没有返回值类型，参数列表也无需给出参数类型。函数体也不必一定要显式 return 返回值，如果没有 return 返回值，则函数返回 undefined（PHP 返回 Null）。JavaScript 函数的声明和使用基本模板如源代码 7-22，源代码 7-23 是一个计算两点间直线斜率的函数示例。

源代码 7-22 函数模板

```
// 函数声明（定义）
function 函数名(形式参数1, 形式参数2, ……){
    函数体语句1;
    函数体语句2;
    ……
```

```
}
var 变量名 = function (形式参数1, 形式参数2, ……){ // 匿名函数
    函数体语句1;
    函数体语句2;
    ……
}
其它语句……
    …… 函数名(实际参数1, 实际参数2, ……)……; //函数调用
    …… 变量名(实际参数1, 实际参数2, ……)……; //通过函数引用，调用函数
```

源代码 7-23 计算两点间斜率的 JavaScript 函数示例

```
function slope(x1, y1, x2, y2){
        var slope = (y2 - y1) / (x2 - x1);
        return slope;
}

var slope1 = slope(0, 0, 1, 1);
alert(slope1);              // 1
var slope2 = slope(0, 0, 1, 1, 3); // 多余参数被忽略
alert(slope2);              // 1
var slope3 = slope(0, 0, 1); // 缺少参数，函数中为undefined
alert(slope3);              // NaN
```

源代码 7-23 中可以看到，当函数调用时提供的实际参数多于形式参数，多的参数会被忽略，而当实际参数不足，少于形式参数个数时，缺少的形式参数在函数执行中取 undefined 值。

7.6.4 弹出框

7.4.2 节讲述了简单弹出框 alert，JavaScript 还可以使用系统函数 confirm 和 prompt 分别生成确认弹出框和输入对话框。确认弹出框根据用户选择 OK 或者 Cancel 返回 true 和 false 供程序使用；输入对话框返回用户输入的值，当用户选择 Cancel 时，返回 null，参见图 7-10。

源代码 7-24 弹出框示例

```
var confirmResult = confirm("请确认：XXXXXXX");
alert("确认结果是：" + confirmResult);

var promptResult = prompt("请输入：");
alert("用户输入：" + promptResult);
```

图 7-10 源代码 7-24 运行结果（上：确认框，下：输入对话框）

7.6.5 异常处理（try/catch throw）

7.6.6 DOM 编程初步

DOM，Document Object Model，文档对象模型，是 W3C 标准[6]，定义了浏览器中网页的对象模型和其开放的编程接口。现代浏览器的实现大都遵循 DOM 标准[7]，因此理论上网页中的 JavaScript 代码能够在不同浏览器里有相同的执行结果[8]。

可以说，对于 Web 前端编程，JavaScript 只不过是语言工具，而 DOM 才是关键内核。通过 DOM，JavaScript 才得以操纵网页在内存中的对象，Web 才呈现出了前端动态性。DOM 将在下一章详细介绍，这里先通过简单的 DOM 编程示例，练习本章讲述的 JavaScript 语法和语言要素，看看 Web 应用的前端脚本大致是什么样子。

图 7-11 是一个简单的加法器，用户填写两个加数，然后点击“加”按钮，结果就出现在结果栏当中。这个加法器的 HTML 代码如源代码 7-25。JavaScript 代码见图 7-12。

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

[7] 早期部分浏览器，如 IE6 之前的版本，与 DOM 标准兼容性比较差。

[8] 事实上，因为浏览器兼容性（参见第 1 章），解决 JavaScript 浏览器兼容性始终是 Web 开发中的一大问题，第 9 章将进一步讲述此问题。

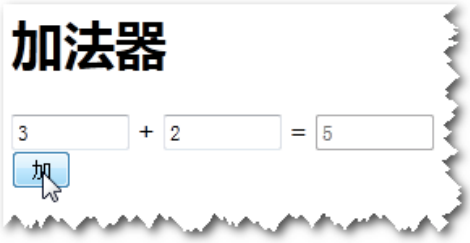

图 7-11 简单 JavaScript 加法器

源代码 7-25 简单加法器（图 7-9）HTML 代码

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
……
<script type="text/javascript" src="adder.js"></script>
<style>input{width:5em;}</style>
</head>
<body>
    <h1>加法器</h1>
<div>
    <input id="left-operand"/>  +  <input id="right-operand"/>
    = <input id="result" disabled="disabled"/><br />
    <button id="add-button">加</button>
</div>
……
```

```
window.onload = function(){
  document.getElementById("add-button").onclick = function(){
    var leftOperand = parseInt(document.getElementById("left-operand").value);
    var rightOperand = parseInt(document.getElementById("right-operand").value);
    var result = leftOperand + rightOperand;
    document.getElementById("result").value = result;
  }
}
```

图 7-12 简单加法器（图 7-11）JavaScript 代码

图 7-12 中的 JavaScript 程序逐行解释如下：

1. 在网页加载完成后，执行 function
2. 获取 id 为”add-button”的网页元素，为它的 click 事件添加事件处理器 function
3. 获得 id 为”left-operand”的网页元素（文本框），将它的值解析为 int
4. 获得 id 为”right-operand”的网页元素（文本框），将它的值解析为 int
5. 计算加法结果
6. 将 id 为”result”的网页元素（文本框），将其值设置为计算结果

网页加载完成后，“加”按钮就增加了 click 事件处理器。当“加”按钮被点击时，事件处理器代码执行，从两个操作数所在的文本框中取出用户的输入，而后计算结果，写入更新到结果文本框。

从图 7-12 程序可以看到，前端代码最主要的任务就是获取网页在内存里对象，为它们添加事件处理器，改变他们的属性。这一切都要通过网页内存对象的模型和其开放接口——DOM 来进行。

从图 7-11 代码中可以看到，可以使用 document.getElementById 这个方法来根据 id 获取 HTML 元素在内存中的对象。还可以进一步通过获得的对象读取、改写 HTML 元素相应的属性，而更新的结果会反映到浏览器窗口的当前网页上。有了这些知识，就可以完成中 7.1 中用户的需要，点击收起（隐藏）或者放出（显示）新闻内容。

7.7 案例研究：登山俱乐部网页—收放新闻与计算器

本节应用 JavaScript 基本知识和 7.6.6 节介绍的简单 DOM 方法 document.getElementById 来实现新闻内容的收起/放开功能和计算器。

7.7.1 收放新闻

回顾 Web 前端程序的基本特点——“使用 JavaScript 给 HTML 元素在浏览器中的内存对象增加事件处理器，该事件处理器响应用户操作，操纵网页元素的内存对象，更改其内容、结构、外观，从而改变浏览器窗口中的网页”。

据此，应该给每个新闻标题添加相应的事件处理器，响应用户点击，收放新闻。程序需要做的具体工作有：

1. 获取新闻标题 HTML 元素内存对象。
2. 给新闻标题 HTML 元素内存对象增加 click 事件处理器，响应用户点击操作。
3. 事件处理器中：
   a) 获得当前新闻标题对应的新闻内容 HTML 元素内存对象。
   b) 更改新闻内容 HTML 元素内存对象的外观属性，或从显示变为隐藏，或从隐藏变为显示。

a) 给需要操作的 HTML 元素加上 id 属性

使用 7.6.6 介绍的 document.getElementById 方法获取要操作的 HTML 元素的内存对象，包括新闻标题和内容段落，需要给这些 HTML 元素各自添加 id 属性，参见

源代码 7-26。

*注意，还有很多其它办法来得到 HTML 元素内存对象，将在下一章讲述。

源代码 7-26 修改 HTML 加入 id 属性

```
<div id="announcements">
    ……
    <h2><span id="ann1" class="announcement">五一活动  "雪山之巅 挑战之旅"</span></h2>
    <p id="ann1_para_1">
     ……
    </p>
    <p id="ann1_para_2">
     ……
    </p>
    <h2><span id="ann2" class="announcement">年度会议</span></h2>
    <p id="ann2_para_1">
      ……
    </p>
</div>
```

b) 给新闻标题增加点击事件处理器

有了 id，就可以通过 document.getElementById 函数获得 HTML 元素内存对象，为它们添加事件处理器，参见源代码 7-27。

源代码 7-27 给新闻标题添加点击事件处理器

```
window.onload = function(){
    var ann1 = document.getElementById("ann1");
    var ann2 = document.getElementById("ann2");
    ann1.onclick = function(){
        ……
    }
    ann2.onclick = function(){
        ……
    }
};
```

源代码 7-27 整个是一个语句，意为给 window，即浏览器窗口的 load 事件，也就是网页加载完成事件，添加处理器。处理器为匿名函数，将在网页加载时被执行。

在 window 的 load 事件中添加网页元素的事件处理器，是 JavaScript 的惯例。网页加载后再执行代码，保障了网页元素已经被解析成内存里的对象，从而保障事件处理器能正确加到这些对象上。

为新闻标题 HTML 元素添加事件处理器部分，同样也使用了匿名函数的写法。有关网页各种事件的内容，参见第 8 章。

c) 显示/隐藏新闻内容

源代码 7-28 通过更改 HTML 元素内存对象 style 属性中 display 的值，改变元素在网页上的显示状态。display 为 none，元素不显示，display 为 block，元素显示为块元素。style 属性就是网页元素的 CSS 外观，包括了 CSS 的各种属性，关于 display 属性，参考第 3 章 CSS 部分。源代码 7-28 运行结果，参见图 7-1。

源代码 7-28 显示/隐藏新闻内容

```
ann1.onclick = function(){
    var ann1_para_1 = document.getElementById("ann1_para_1");
    var ann1_para_2 = document.getElementById("ann1_para_2");
    var style = ann1_para_1.style.display == "none" ? "block" :
"none";
    ann1_para_1.style.display = style;
    ann1_para_2.style.display = style;
}
```

至此，收起/放开新闻内容的功能就得到了实现。但是，对源代码 7-28 稍微观察就会发现，目前实现方式比较麻烦。增加新闻时，既要给出多个 id（标题 span、每个自然段 p），还要增加不少对应的 JavaScript 代码。并且，增加的代码和已有代码大量雷同，整个程序可扩展、可维护性很差。

这些问题将在下一章解决。下一章将详细讲述 DOM，利用 DOM 强大的接口，可以简洁高效地实现收放新闻的功能。

7.7.2 计算器

按照 SoC 的原则，HTML 定义网页的内容与结构，CSS 定义网页外观，JavaScript 负责网页行为的原则，参考。下面分别完成计算器的这几个部分。

a) 基本 HTML 和 CSS

首先，完成计算器的 HTML 和 CSS 代码，分别如：源代码 7-29 和源代码 7-30，计算器在网页上的外观如：图 7-13。

源代码 7-29 计算器 HTML 代码

```
……
<script type="text/javascript" src="calculator.js"></script>
```

```
<link rel="stylesheet" type="text/css" href="calculator.css" />
……
<div id = "main">
  <h1>简单计算器</h1>
    <div id = "calculator">
        <div id="output_field">
            <input type="text" id="output" name="output" />
        </div>
        <div id = "buttons">
            <button id="7">7</button>
            <button id="8">8</button>
            <button id="9">9</button>
            <button id="div">/</button>
            <button id="4">4</button>
            <button id="5">5</button>
            <button id="6">6</button>
            <button id="mul">*</button>
            <button id="1">1</button>
            <button id="2">2</button>
            <button id="3">3</button>
            <button id="sub">-</button>
            <button id="0">0</button>
            <button id="dot">.</button>
            <button id="back">&larr;</button>
            <button id="add">+</button>
            <button id="left_p">(</button>
            <button id="right_p">)</button>
            <button id="ce">CE</button>
            <button id="eq">=</button>
        </div>
    </div>
</div>
```

源代码 7-30 计算器 CSS 代码（calculator.css）

```
div#main {
    margin-left: auto; margin-right: auto;
    text-align: center; width: 240px;}

div#main > h1{font-size: 22px; color: green;}

div#calculator {border:solid 2px;  padding: 7px;   text-align:left;}

div#buttons {width: 240px;}

button {width: 50px;    margin-right: 2px;}

div#output_field {margin-bottom: 0.5em;}

input#output{border:groove; width: 214px;  text-align: right;}
```

图 7-13 源代码 7-29 和源代码 7-30 运行结果

b) eval 函数

作为解释型语言 JavaScript 可以动态执行源程序，包括运行时生成的源程序。JavaScript 提供了函数 eval，其参数为一个 JavaScript 表达式字符串。运行时，eval 函数将返回当前上下文这个表达式的值，例如：源代码 7-31。计算器程序中，使用 eval 函数，就可以方便的计算出用户输入的算术表达式的值。

源代码 7-31 eval 函数用法示例

```
eval("3 + 2 - 5 * 0");   // 5
var str = "Hello";
eval("str = str + ' World!'");
eval(alert(str)); // 弹出'Hello World!'
```

慎用 eval：eval 函数能够动态执行源代码，必须慎用。特别是当执行的源代码直接来自用户的收入，更要格外小心。如果不小心，就会和前面谈到的 HTML 代码注入一样，造成代码注入的安全问题。更多代码注入和 Web 安全问题，参考第 xx 节。这里的计算器程序，eval 函数计算的表达式，用户只能通过计算器按钮输入，所以没有注入问题。

c) 需求分析

计算器程序显然比之新闻收放程序逻辑上要复杂的多，按照人们的使用习惯，计算器通常的用例如下：

1. 响应用户对数字和算术操作符按钮的操作，记录并显示用户通过按钮

输入的算术表达式。

2. 响应用户功能按钮的操作。

a) 用户按下“←”按钮，删除当前算术表达式最后一个字符，并更新显示。

b) 用户按下“CE”按钮，清除当前算术表达式。

c) 用户按下“=”按钮，计算当前表达式的结果并显示。

i. 如果，算术表达式非法，弹出警告框提醒用户，并终止计算。

d) 程序设计

根据需求分析，类似收放新闻程序，遵照 Web 前端程序特点，计算器程序需要做的工作有：

1. 使用变量存储计算器的状态，也就是当前需要计算的算术表达式。
2. 获取所有用户可操作的 HTML 元素内存对象，也就是计算器的所有按钮，包括数字、算术操作、功能操作按钮。
3. 给所有按钮的点击（click）事件添加不同的事件处理器。
4. 事件处理器当中：
   a) 数字和算术操作按钮事件处理器：获得当前按钮的文本，附加到算术表达式，并更新结果框的显示。
   b) “←”按钮事件处理器：删除算术表达式最后一个字符，并更新结果框的显示。
   c) “CE”按钮事件处理器：清除算术表达式，并更新结果框的显示。
   d) “=”按钮事件处理器：计算算术表达式，并更新结果框的显示。然后，清除算术表达式。
      i. 如果算术表达式非法，弹出提示，并终止计算。

源代码 7-32 展示了整个 JavaScript 源程序。

源代码 7-32 计算器 JavaScript 代码(calculator.js)

```
var expression = ""; // 计算器状态，当前的算术表达式

window.onload = function(){ // 为所有按钮添加事件处理器
  addEventHandlersToNumberAndOperatorButtons();
  document.getElementById("eq").onclick = showResult;
  document.getElementById("ce").onclick = clearInput;
  document.getElementById("back").onclick = backDel;
}

// 为数字、算术操作按钮添加事件处理器
function addEventHandlersToNumberAndOperatorButtons(){
  var ids = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0'];
  ids = ids.concat(['add', 'sub', 'mul', 'div', 'dot', 'left_p',
'right_p']);
```

```
  for (var i = ids.length - 1; i >= 0; i--) {
    document.getElementById(ids[i]).onclick = appendExpression;
  }
}

// 数字、算术操作按钮的事件处理器
function appendExpression(event){
  expression += event.target.textContent; //获得按钮文字，参见第8章
  updateOutput(expression);
}

function updateOutput(data){
  document.getElementById("output").value = data;
}

// "="按钮的事件处理器
function showResult(){
  try {
    var result = eval(expression);
    updateOutput(result);
    expression = "";
  }
  catch (e) {
    alert("'" + expression + "' is invalid.");
  }
}

// "CE"按钮的事件处理器
function clearInput(){
  updateOutput("");
}

// "<-"按钮的事件处理器
function backDel(){
  expression = expression.substring(0, expression.length - 1);
  updateOutput(expression);
}
```

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 7.7 案例研究，同样为日程表增加点击收放的功能。
3. 改进计算器，增加平方、开方和三角函数计算功能。
4. 改进计算器，增加累加功能。

进一步阅读

1. 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
2. JavaScript 引擎介绍：http://en.wikipedia.org/wiki/JavaScript_engine
3. 使用 Firebug 调试 JavaScript：http://getfirebug.com/screencasts
4. 使用 IE 调试工具调试 JavaScript：http://msdn.microsoft.com/en-us/library/dd565627%28v=vs.85%29.aspx
5. ECMAScript Specification: http://www.ecma-international.org/publications/standards/Ecma-262.htm
6. Mozilla Developer Center – Core JavaScript 1.5 Guide: https://developer.mozilla.org/en/JavaScript/Guide
7. Mozilla Developer Center – Core JavaScript 1.5 Reference: https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference
8. W3Schools style DOM object 属性：http://www.w3schools.com/jsref/dom_obj_style.asp
9. W3Schools JavaScripts 教程：http://www.w3schools.com/js/default.asp
10. JavaScript: The Definitive Guide：

http://books.google.com/books/about/JavaScript.html?id=2weL0iAfrEMC