

目 录

第 1 章 JavaScript 简介	1	3.4.2 Undefined 类型	24
1.1 JavaScript 简史	1	3.4.3 Null 类型	25
1.2 JavaScript 实现	2	3.4.4 Boolean 类型	26
1.2.1 ECMAScript	3	3.4.5 Number 类型	27
1.2.2 文档对象模型 (DOM)	5	3.4.6 String 类型	32
1.2.3 浏览器对象模型 (BOM)	8	3.4.7 Object 类型	35
1.3 JavaScript 版本	8	3.5 操作符	36
1.4 小结	9	3.5.1 一元操作符	36
第 2 章 在 HTML 中使用 JavaScript	10	3.5.2 位操作符	39
2.1 <script>元素	10	3.5.3 布尔操作符	44
2.1.1 标签的位置	12	3.5.4 乘性操作符	47
2.1.2 延迟脚本	13	3.5.5 加性操作符	48
2.1.3 异步脚本	13	3.5.6 关系操作符	50
2.1.4 在 XHTML 中的用法	14	3.5.7 相等操作符	51
2.1.5 不推荐使用的语法	16	3.5.8 条件操作符	53
2.2 嵌入代码与外部文件	16	3.5.9 赋值操作符	53
2.3 文档模式	16	3.5.10 逗号操作符	54
2.4 <noscript>元素	18	3.6 语句	54
2.5 小结	18	3.6.1 if 语句	54
第 3 章 基本概念	19	3.6.2 do-while 语句	55
3.1 语法	19	3.6.3 while 语句	55
3.1.1 区分大小写	19	3.6.4 for 语句	56
3.1.2 标识符	19	3.6.5 for-in 语句	57
3.1.3 注释	20	3.6.6 label 语句	58
3.1.4 严格模式	20	3.6.7 break 和 continue 语句	58
3.1.5 语句	20	3.6.8 with 语句	60
3.2 关键字和保留字	21	3.6.9 switch 语句	60
3.3 变量	22	3.7 函数	62
3.4 数据类型	23	3.7.1 理解参数	64
3.4.1 typeof 操作符	23	3.7.2 没有重载	66
		3.8 小结	67

第 4 章 变量、作用域和内存问题	68	5.5.3 作为值的函数	112
4.1 基本类型和引用类型的值	68	5.5.4 函数内部属性	113
4.1.1 动态的属性	68	5.5.5 函数属性和方法	116
4.1.2 复制变量值	69	5.6 基本包装类型	118
4.1.3 传递参数	70	5.6.1 Boolean 类型	120
4.1.4 检测类型	72	5.6.2 Number 类型	120
4.2 执行环境及作用域	73	5.6.3 String 类型	122
4.2.1 延长作用域链	75	5.7 单体内置对象	130
4.2.2 没有块级作用域	76	5.7.1 Global 对象	131
4.3 垃圾收集	78	5.7.2 Math 对象	134
4.3.1 标记清除	78	5.8 小结	137
4.3.2 引用计数	79	第 6 章 面向对象的程序设计	138
4.3.3 性能问题	80	6.1 理解对象	138
4.3.4 管理内存	81	6.1.1 属性类型	139
4.4 小结	81	6.1.2 定义多个属性	142
第 5 章 引用类型	83	6.1.3 读取属性的特性	143
5.1 Object 类型	83	6.2 创建对象	144
5.2 Array 类型	86	6.2.1 工厂模式	144
5.2.1 检测数组	88	6.2.2 构造函数模式	144
5.2.2 转换方法	89	6.2.3 原型模式	147
5.2.3 栈方法	90	6.2.4 组合使用构造函数模式和原型模式	159
5.2.4 队列方法	91	6.2.5 动态原型模式	159
5.2.5 重排序方法	92	6.2.6 寄生构造函数模式	160
5.2.6 操作方法	94	6.2.7 稳妥构造函数模式	161
5.2.7 位置方法	95	6.3 继承	162
5.2.8 迭代方法	96	6.3.1 原型链	162
5.2.9 归并方法	97	6.3.2 借用构造函数	167
5.3 Date 类型	98	6.3.3 组合继承	168
5.3.1 继承的方法	100	6.3.4 原型式继承	169
5.3.2 日期格式化方法	101	6.3.5 寄生式继承	171
5.3.3 日期/时间组件方法	102	6.3.6 寄生组合式继承	172
5.4 RegExp 类型	103	6.4 小结	174
5.4.1 RegExp 实例属性	105	第 7 章 函数表达式	175
5.4.2 RegExp 实例方法	106	7.1 递归	177
5.4.3 RegExp 构造函数属性	107	7.2 闭包	178
5.4.4 模式的局限性	109	7.2.1 闭包与变量	181
5.5 Function 类型	110	7.2.2 关于 this 对象	182
5.5.1 没有重载 (深入理解)	111	7.2.3 内存泄漏	183
5.5.2 函数声明与函数表达式	111		

7.3 模仿块级作用域	184	10.1.1 Node 类型	248
7.4 私有变量	186	10.1.2 Document 类型	253
7.4.1 静态私有变量	188	10.1.3 Element 类型	261
7.4.2 模块模式	189	10.1.4 Text 类型	270
7.4.3 增强的模块模式	191	10.1.5 Comment 类型	273
7.5 小结	192	10.1.6 CDATASection 类型	274
第 8 章 BOM	193	10.1.7 DocumentType 类型	274
8.1 window 对象	193	10.1.8 DocumentFragment 类型	275
8.1.1 全局作用域	193	10.1.9 Attr 类型	276
8.1.2 窗口关系及框架	194	10.2 DOM 操作技术	277
8.1.3 窗口位置	197	10.2.1 动态脚本	277
8.1.4 窗口大小	198	10.2.2 动态样式	279
8.1.5 导航和打开窗口	199	10.2.3 操作表格	281
8.1.6 间歇调用和超时调用	203	10.2.4 使用 NodeList	283
8.1.7 系统对话框	205	10.3 小结	284
8.2 location 对象	207	第 11 章 DOM 扩展	286
8.2.1 查询字符串参数	207	11.1 选择符 API	286
8.2.2 位置操作	208	11.1.1 querySelector() 方法	286
8.3 navigator 对象	210	11.1.2 querySelectorAll() 方法	287
8.3.1 检测插件	211	11.1.3 matchesSelector() 方法	288
8.3.2 注册处理程序	213	11.2 元素遍历	288
8.4 screen 对象	214	11.3 HTML5	289
8.5 history 对象	215	11.3.1 与类相关的扩充	289
8.6 小结	216	11.3.2 焦点管理	291
第 9 章 客户端检测	217	11.3.3 HTMLDocument 的变化	292
9.1 能力检测	217	11.3.4 字符集属性	293
9.1.1 更可靠的能力检测	218	11.3.5 自定义数据属性	293
9.1.2 能力检测, 不是浏览器检测	220	11.3.6 插入标记	294
9.2 怪癖检测	220	11.3.7 scrollIntoView() 方法	298
9.3 用户代理检测	221	11.4 专有扩展	298
9.3.1 用户代理字符串的历史	222	11.4.1 文档模式	298
9.3.2 用户代理字符串检测技术	228	11.4.2 children 属性	299
9.3.3 完整的代码	242	11.4.3 contains() 方法	300
9.3.4 使用方法	245	11.4.4 插入文本	301
9.4 小结	246	11.4.5 滚动	303
第 10 章 DOM	247	11.5 小结	304
10.1 节点层次	247		

第 12 章 DOM2 和 DOM3	305	13.4.9 触摸与手势事件	399
12.1 DOM 变化	305	13.5 内存和性能	402
12.1.1 针对 XML 命名空间的变化	306	13.5.1 事件委托	402
12.1.2 其他方面的变化	309	13.5.2 移除事件处理程序	404
12.2 样式	312	13.6 模拟事件	405
12.2.1 访问元素的样式	313	13.6.1 DOM 中的事件模拟	405
12.2.2 操作样式表	317	13.6.2 IE 中的事件模拟	410
12.2.3 元素大小	320	13.7 小结	411
12.3 遍历	326	第 14 章 表单脚本	412
12.3.1 NodeIterator	328	14.1 表单的基础知识	412
12.3.2 TreeWalker	330	14.1.1 提交表单	413
12.4 范围	332	14.1.2 重置表单	414
12.4.1 DOM 中的范围	332	14.1.3 表单字段	414
12.4.2 IE8 及更早版本中的范围	340	14.2 文本框脚本	419
12.5 小结	343	14.2.1 选择文本	420
第 13 章 事件	345	14.2.2 过滤输入	423
13.1 事件流	345	14.2.3 自动切换焦点	426
13.1.1 事件冒泡	346	14.2.4 HTML5 约束验证 API	427
13.1.2 事件捕获	346	14.3 选择框脚本	431
13.1.3 DOM 事件流	347	14.3.1 选择选项	432
13.2 事件处理程序	348	14.3.2 添加选项	434
13.2.1 HTML 事件处理程序	348	14.3.3 移除选项	435
13.2.2 DOM0 级事件处理程序	350	14.3.4 移动和重排选项	435
13.2.3 DOM2 级事件处理程序	351	14.4 表单序列化	436
13.2.4 IE 事件处理程序	352	14.5 富文本编辑	438
13.2.5 跨浏览器的事件处理程序	353	14.5.1 使用 contenteditable 属性	438
13.3 事件对象	355	14.5.2 操作富文本	439
13.3.1 DOM 中的事件对象	355	14.5.3 富文本选区	441
13.3.2 IE 中的事件对象	358	14.5.4 表单与富文本	443
13.3.3 跨浏览器的事件对象	360	14.6 小结	443
13.4 事件类型	362	第 15 章 使用 Canvas 绘图	445
13.4.1 UI 事件	362	15.1 基本用法	445
13.4.2 焦点事件	367	15.2 2D 上下文	446
13.4.3 鼠标与滚轮事件	368	15.2.1 填充和描边	446
13.4.4 键盘与文本事件	379	15.2.2 绘制矩形	447
13.4.5 复合事件	384	15.2.3 绘制路径	449
13.4.6 变动事件	385	15.2.4 绘制文本	451
13.4.7 HTML5 事件	388	15.2.5 变换	453
13.4.8 设备事件	395		

15.2.6 绘制图像	456	17.2.3 错误 (error) 事件	505
15.2.7 阴影	457	17.2.4 处理错误的策略	506
15.2.8 渐变	458	17.2.5 常见的错误类型	507
15.2.9 模式	460	17.2.6 区分致命错误和非致命 错误	510
15.2.10 使用图像数据	460	17.2.7 把错误记录到服务器	511
15.2.11 合成	462	17.3 调试技术	512
15.3 WebGL	463	17.3.1 将消息记录到控制台	512
15.3.1 类型化数组	463	17.3.2 将消息记录到当前页面	515
15.3.2 WebGL 上下文	468	17.3.3 抛出错误	515
15.3.3 支持	478	17.4 常见的 IE 错误	516
15.4 小结	478	17.4.1 操作终止	516
第 16 章 HTML5 脚本编程	480	17.4.2 无效字符	518
16.1 跨文档消息传递	480	17.4.3 未找到成员	518
16.2 原生拖放	481	17.4.4 未知运行时错误	519
16.2.1 拖放事件	482	17.4.5 语法错误	519
16.2.2 自定义放置目标	482	17.4.6 系统无法找到指定资源	519
16.2.3 dataTransfer 对象	483	17.5 小结	520
16.2.4 dropEffect 与 effectAllowed	484	第 18 章 JavaScript 与 XML	521
16.2.5 可拖动	485	18.1 浏览器对 XML DOM 的支持	521
16.2.6 其他成员	485	18.1.1 DOM2 级核心	521
16.3 媒体元素	486	18.1.2 DOMParser 类型	522
16.3.1 属性	487	18.1.3 XMLSerializer 类型	523
16.3.2 事件	488	18.1.4 IE8 及之前版本中的 XML	523
16.3.3 自定义媒体播放器	488	18.1.5 跨浏览器处理 XML	527
16.3.4 检测编解码器的支持情况	489	18.2 浏览器对 XPath 的支持	529
16.3.5 Audio 类型	490	18.2.1 DOM3 级 XPath	529
16.4 历史状态管理	491	18.2.2 IE 中的 XPath	534
16.5 小结	492	18.2.3 跨浏览器使用 XPath	535
第 17 章 错误处理与调试	493	18.3 浏览器对 XSLT 的支持	537
17.1 浏览器报告的错误	493	18.3.1 IE 中的 XSLT	537
17.1.1 IE	493	18.3.2 XSLTProcessor 类型	541
17.1.2 Firefox	494	18.3.3 跨浏览器使用 XSLT	543
17.1.3 Safari	496	18.4 小结	544
17.1.4 Opera	497	第 19 章 E4X	546
17.1.5 Chrome	498	19.1 E4X 的类型	546
17.2 错误处理	499	19.1.1 XML 类型	546
17.2.1 try-catch 语句	500	19.1.2 XMLList 类型	547
17.2.2 抛出错误	503	19.1.3 Namespace 类型	548

19.1.4	QName 类型	549	21.4.3	Preflighted Reqeusts	584
19.2	一般用法	550	21.4.4	带凭据的请求	585
19.2.1	访问特性	551	21.4.5	跨浏览器的 CORS	585
19.2.2	其他节点类型	552	21.5	其他跨域技术	586
19.2.3	查询	553	21.5.1	图像 Ping	586
19.2.4	构建和操作 XML	555	21.5.2	JSONP	587
19.2.5	解析和序列化	557	21.5.3	Comet	588
19.2.6	命名空间	558	21.5.4	服务器发送事件	590
19.3	其他变化	559	21.5.5	Web Sockets	591
19.4	全面启用 E4X	560	21.5.6	SSE 与 Web Sockets	593
19.5	小结	561	21.6	安全	593
第 20 章	JSON	562	21.7	小结	594
20.1	语法	562	第 22 章	高级技巧	596
20.1.1	简单值	562	22.1	高级函数	596
20.1.2	对象	563	22.1.1	安全的类型检测	596
20.1.3	数组	564	22.1.2	作用域安全的构造函数	597
20.2	解析与序列化	565	22.1.3	惰性载入函数	600
20.2.1	JSON 对象	565	22.1.4	函数绑定	602
20.2.2	序列化选项	566	22.1.5	函数柯里化	604
20.2.3	解析选项	569	22.2	防篡改对象	606
20.3	小结	570	22.2.1	不可扩展对象	606
第 21 章	Ajax 与 Comet	571	22.2.2	密封的对象	607
21.1	XMLHttpRequest 对象	571	22.2.3	冻结的对象	608
21.1.1	XHR 的用法	573	22.3	高级定时器	609
21.1.2	HTTP 头部信息	575	22.3.1	重复的定时器	610
21.1.3	GET 请求	576	22.3.2	Yielding Processes	612
21.1.4	POST 请求	577	22.3.3	函数节流	614
21.2	XMLHttpRequest 2 级	578	22.4	自定义事件	616
21.2.1	FormData	578	22.5	拖放	618
21.2.2	超时设定	579	22.5.1	修缮拖动功能	620
21.2.3	overrideMimeType() 方法	580	22.5.2	添加自定义事件	622
21.3	进度事件	580	22.6	小结	624
21.3.1	load 事件	580	第 23 章	离线应用与客户端存储	626
21.3.2	progress 事件	581	23.1	离线检测	626
21.4	跨源资源共享	582	23.2	应用缓存	627
21.4.1	IE 对 CORS 的实现	582	23.3	数据存储	628
21.4.2	其他浏览器对 CORS 的 实现	584	23.3.1	Cookie	629
			23.3.2	IE 用户数据	637

23.3.3 Web 存储机制	638	25.1.4 webkitRequestAnimationFrame 与 msRequestAnimationFrame	685
23.3.4 IndexedDB	643	25.2 Page Visibility API	686
23.4 小结	654	25.3 Geolocation API	687
第 24 章 最佳实践	656	25.4 File API	689
24.1 可维护性	656	25.4.1 FileReader 类型	690
24.1.1 什么是可维护的代码	656	25.4.2 读取部分内容	692
24.1.2 代码约定	657	25.4.3 对象 URL	693
24.1.3 松散耦合	659	25.4.4 读取拖放的文件	694
24.1.4 编程实践	662	25.4.5 使用 XHR 上传文件	695
24.2 性能	666	25.5 Web 计时	696
24.2.1 注意作用域	666	25.6 Web Workers	697
24.2.2 选择正确方法	667	25.6.1 使用 Worker	697
24.2.3 最小化语句数	672	25.6.2 Worker 全局作用域	698
24.2.4 优化 DOM 交互	673	25.6.3 包含其他脚本	699
24.3 部署	676	25.6.4 Web Workers 的未来	700
24.3.1 构建过程	676	25.7 小结	700
24.3.2 验证	677	附录 A ECMAScript Harmony	701
24.3.3 压缩	679	附录 B 严格模式	717
24.4 小结	681	附录 C JavaScript 库	723
第 25 章 新兴的 API	682	附录 D JavaScript 工具	727
25.1 requestAnimationFrame()	682		
25.1.1 早期动画循环	682		
25.1.2 循环间隔的问题	683		
25.1.3 mozRequestAnimationFrame	683		

第 1 章

JavaScript 简介

本章内容

- JavaScript 历史回顾
- JavaScript 是什么
- JavaScript 与 ECMAScript 的关系
- JavaScript 的不同版本

JavaScript 诞生于 1995 年。当时，它的主要目的是处理以前由服务器端语言（如 Perl）负责的一些输入验证操作。在 JavaScript 问世之前，必须把表单数据发送到服务器端才能确定用户是否没有填写某个必填域，是否输入了无效的值。Netscape Navigator 希望通过 JavaScript 来解决这个问题。在人们普遍使用电话拨号上网的年代，能够在客户端完成一些基本的验证任务绝对是令人兴奋的。毕竟，拨号上网的速度之慢，导致了与服务器的每一次数据交换事实上都成了对人们耐心的一次考验。

自此以后，JavaScript 逐渐成为市面上常见浏览器必备的一项特色功能。如今，JavaScript 的用途早已不再局限于简单的数据验证，而是具备了与浏览器窗口及其内容等几乎所有方面交互的能力。今天的 JavaScript 已经成为一门功能全面的编程语言，能够处理复杂的计算和交互，拥有了闭包、匿名（lambda，拉姆达）函数，甚至元编程等特性。作为 Web 的一个重要组成部分，JavaScript 的重要性是不言而喻的，就连手机浏览器，甚至那些专为残障人士设计的浏览器等非常规浏览器都支持它。当然，微软的例子更为典型。虽然有自己的客户端脚本语言 VBScript，但微软仍然在 Internet Explorer 的早期版本中加入了自己的 JavaScript 实现^①。

JavaScript 从一个简单的输入验证器发展成为一门强大的编程语言，完全出乎人们的意料。应该说，它既是一门非常简单的语言，又是一门非常复杂的语言。说它简单，是因为学会使用它只需片刻功夫；而说它复杂，是因为要真正掌握它则需要数年时间。要想全面理解和掌握 JavaScript，关键在于弄清楚它的本质、历史和局限性。

1.1 JavaScript 简史

在 Web 日益流行的同时，人们对客户端脚本语言的需求也越来越强烈。那个时候，绝大多数因特网用户都使用速度仅为 28.8kbit/s 的“猫”（调制解调器）上网，但网页的大小和复杂性却不断增加。为完成简单的表单验证而频繁地与服务器交换数据只会加重用户的负担。想象一下：用户填写完一个表单，单击“提交”按钮，然后等待 30 秒钟，最终服务器返回消息说有一个必填字段没有

① 对 IE 而言，当我们提到 JavaScript 时，实际上就是指 IE 对 JavaScript (ECMAScript) 的实现——JScript。最早的 JScript 基于 Netscape JavaScript 1.0 开发，于 1996 年 8 月随同 Internet Explorer 3.0 发布。

填好……当时走在技术革新最前沿的 Netscape 公司，决定着手开发一种客户端语言，用来处理这种简单的验证。

当时就职于 Netscape 公司的布兰登·艾奇（Brendan Eich），开始着手为计划于 1995 年 2 月发布的 Netscape Navigator 2 开发一种名为 LiveScript 的脚本语言——该语言将同时在浏览器和服务器的使用（它在服务器上的名字叫 LiveWire）。为了赶在发布日期前完成 LiveScript 的开发，Netscape 与 Sun 公司建立了一个开发联盟。在 Netscape Navigator 2 正式发布前夕，Netscape 为了搭上媒体热炒 Java 的顺风车，临时把 LiveScript 改名为 JavaScript。

由于 JavaScript 1.0 获得了巨大成功，Netscape 随即在 Netscape Navigator 3 中又发布了 JavaScript 1.1。Web 虽然羽翼未丰，但用户关注度却屡创新高。在这样的背景下，Netscape 把自己定位为市场领袖型公司。与此同时，微软决定向与 Navigator 竞争的自家产品 Internet Explorer 浏览器投入更多资源。Netscape Navigator 3 发布后不久，微软就在其 Internet Explorer 3 中加入了名为 JScript 的 JavaScript 实现（命名为 JScript 是为了避开与 Netscape 有关的授权问题）。以现在的眼光来看，微软 1996 年 8 月为进入 Web 浏览器领域而实施的这个重大举措，是导致 Netscape 日后蒙羞的一个标志性事件。然而，这个重大举措同时也标志着 JavaScript 作为一门语言，其开发向前迈进了一大步。

微软推出其 JavaScript 实现意味着有了两个不同的 JavaScript 版本：Netscape Navigator 中的 JavaScript、Internet Explorer 中的 JScript。与 C 及其他编程语言不同，当时还没有标准规定 JavaScript 的语法和特性，两个不同版本并存的局面已经完全暴露了这个问题。随着业界担心的日益加剧，JavaScript 的标准化问题被提上了议事日程。

1997 年，以 JavaScript 1.1 为蓝本的建议被提交给了欧洲计算机制造商协会（ECMA，European Computer Manufacturers Association）。该协会指定 39 号技术委员会（TC39，Technical Committee #39）负责“标准化一种通用、跨平台、供应商中立的脚本语言的语法和语义”（<http://www.ecma-international.org/memento/TC39.htm>）。TC39 由来自 Netscape、Sun、微软、Borland 及其他关注脚本语言发展的公司的程序员组成，他们经过数月的努力完成了 ECMA-262——定义一种名为 ECMAScript（发音为“ek-ma-script”）的新脚本语言的标准。

第二年，ISO/IEC（International Organization for Standardization and International Electrotechnical Commission，国际标准化组织和国际电工委员会）也采用了 ECMAScript 作为标准（即 ISO/IEC-16262）。自此以后，浏览器开发商就开始致力于将 ECMAScript 作为各自 JavaScript 实现的基础，也在不同程度上取得了成功。

1.2 JavaScript 实现

虽然 JavaScript 和 ECMAScript 通常都被人们用来表达相同的含义，但 JavaScript 的含义却比 ECMA-262 中规定的要多得多。没错，一个完整的 JavaScript 实现应该由下列三个不同的部分组成（见图 1-1）。

- ❑ 核心（ECMAScript）
- ❑ 文档对象模型（DOM）
- ❑ 浏览器对象模型（BOM）

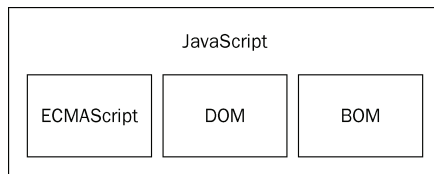


图 1-1