JavaScript著名专家撰写

指导读者进入JavaScript框架设计的魔法指南

JavaScript 框架设计

司徒正美 ◎ 编著

人民邮电出版社
POSTS & TELECOM PRESS

JavaScript 框架设计

司徒正美　编著

人 民 邮 电 出 版 社

北　京

内 容 提 要

本书是一本全面讲解 JavaScript 框架设计的图书，详细地讲解了设计框架需要具备的知识，主要包括的内容为：框架与库、JavaScript 框架分类、JavaScript 框架的主要功能、种子模块、模块加载系统、语言模块、浏览器嗅探与特征侦测、样式的支持侦测、类工厂、JavaScript 对类的支撑、选择器引擎、浏览器内置的寻找元素的方法、节点模块、一些有趣的元素节点、数据缓存系统、样式模块、个别样式的特殊处理、属性模块、jQuery 的属性系统、事件系统、异步处理、JavaScript 异步处理的前景、数据交互模块、一个完整的 Ajax 实现、动画引擎、API 的设计、插件化、当前主流 MVVM 框架介绍、监控数组与子模板等。

本书适合前端设计人员、JavaScript 开发者、移动 UI 设计者、程序员和项目经理阅读，也可作为大中专院校相关专业的师生学习用书和培训学校的教材。

◆ 编　　著　司徒正美
责任编辑　张　涛
责任印制　程彦红

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鑫丰华彩印有限公司印刷

◆ 开本：800×1000　1/16
印张：
字数：　千字　　2014 年 3 月第 1 版
印数：　册　　2014 年 3 月北京第 1 次印刷

定价：　元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号

前　言

首先说明一下，本书虽是讲解框架设计，但写个框架不是很深奥的事情，重点是建立更为完整的前端知识树。只有自己尝试写个框架，才有机会接触像原型、作用域、事件代理、缓存系统、定时器等深层知识，也才有机会了解 applyElement、swapNode、importNode、removeNode、replaceNode、insertAdjacentHTML、createContextualFragment、runtimeStyle 等偏门 API，也才会知晓像 getElementById、getElementsByTagName、setAttribute、innerHTML 存在大量的 Bug，当然你也可以尝试最近几年浏览器带来的新 API（包括 ECMA262v5、v6、HTML5 或大量误认为是 HTML5 的新模块），如 Object.defineProperty、CSS.supports、WebKitShadowRoot、getDefaultComputedStyle……

虽然这难免落入“造轮子”的怪圈中，但“造轮子”在这世界却是出奇普遍。任何创造性的活动，一开始都是临摹他人的作品。就算不“造轮子”，也要收集一大堆轮子，作家有他的素材集，设计师有大量 icon 与笔刷，普通的“码农”也有个 commonjs 存放着一些常用的函数。以前的程序员们，经常会为了做一个数据处理程序而自己开发一门编程语言。如 Charls Moore，他在美国国家天文台做射电望远镜数据提取程序时开发了 Forth；高德纳为了让自己写的书排版漂亮些，写了 TeX；DHH 为了做网站写了 Rails……如果连写一个控件都要百度或 Google 查找答案，那水平不容易提高。

当前很少有技术书教你写框架的，即便是众多的 Java 类图书，大多数也是教你如何深入了解 SHH 的运作机理。

如果你是这两三年才接触 JavaScript，那恭喜你了。现在 JavaScript 的应用洪荒时代已经过去，Portotype.js 的幕府“统治”也已结果，且已迎来非常强势的 jQuery 纪元，有大量现成的插件可用，许多公司都用 jQuery，意味着我们的技术有了用武之地。

但事实上还是要通过调试程序获得经验，只从 JavaScript 书上学习的那些知识点没法明白 jQuery 的源代码。

许多大公司的架构师根据技术发展的情况，他们都有自己一套或几套 JavaScript 底层库，各个部门视情况还发展针对于自己业务情况的 UI 库。而企业开发中，UI 库就像流水线那么重要。而底层库只是一个好用的“锤子”或“胶钳”。要想迅速上手这么多公司框架，基础知识无疑是非常重要的。假若之前自己写过框架，那就有了经验了。道理是一样的，框架设计的一些“套路”肯定存在的。本书就是把这些“潜规则”公开出来，迅速让自己成长为技术达人。

1. 框架与库

下面稍微说一下框架与库的区别。

库是解决某个问题而拼凑出来的一大堆函数与类的集合。例如，盖一个房子，需要有测量的方法、砌砖的方法、安装门窗的方法等。每个方法之间都没什么关联。至于怎么盖房子都由自己决定。

框架则是一个半成品的应用，直接给出一个骨架，还例如盖房子，照着图纸砌砖、铺地板与涂漆就行了。在后端 Java 的三大框架中，程序员基本上就是与 XML 打交道，用配置就可以处理 80%的编程问题。

从上面描述来看，框架带来的便利性无疑比库好许多。但前端 JavaScript 由于存在一个远程加载的问题，对 JavaScript 文件的体积限制很大，因此，框架在几年前都不怎么吃香。现在网速快多了，设计师在网页制造的地位（UED）也不比昔日，因此，集成程度更高的 MVC、MVVM 框架也相继面世。

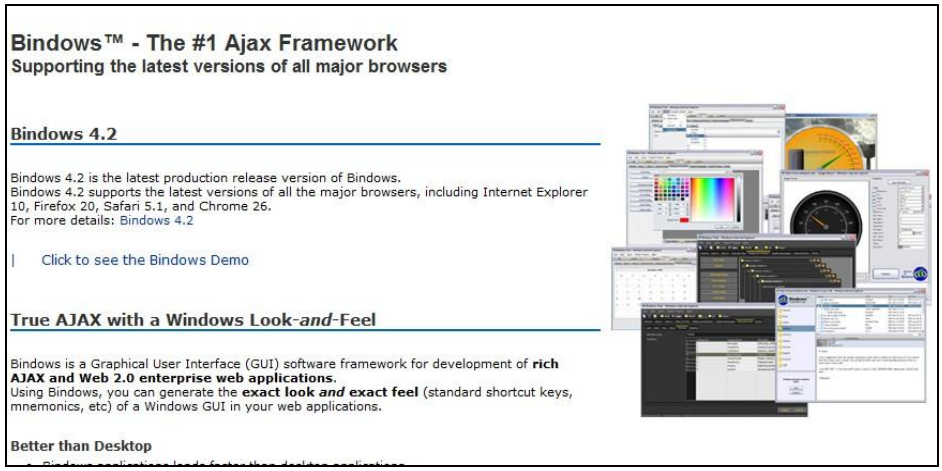
不过，无论是框架还是库，只要在浏览器中运行，就要与 DOM 打交道。如果不用 jQuery，就要发明一个半成品 jQuery 或一个半成品 Prototype。对想提升自己的能力的人来说，答案其实很明显，写框架还能提升自己的架构能力。

2. JavaScript 年发展历程

第一个年代，洪荒时代。从 1995 年到 2005 年，就是从 JavaScript 发明到 Ajax 概念[1]的提出。其间打了第一场浏览器战争，IE VS Netscape。这两者的 DOM API 出入很大，前端开发人员被迫改进技术，为了不想兼容某一个浏览器，发明 UA（navigator.userAgent）嗅探技术。

这个时期的杰出代表是 Bindows[2]，2003 年发布，它提供了一个完整的 Windows 桌面系统，支持能在 EXT 看到的各种控件，如菜单、树、表格、滑动条、切换卡、弹出层、测量仪表（使用 VML 实现，现在又支持 SVG）。现在版本号是 4.x，如下图所示。

1 http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications
2 http://www.bindows.net/



其他比较著名的还有，Dojo（2004 年）、Sarissa（2003 年）、JavaScript Remote Scripting（2000 年）。

Dojo 有 IBM 做后台，有庞大的开发团队在做，质量有保证，被广泛整合到各大 Java 框架内（ struct2、Tapestry、Eclipse ATF、MyFaces）。特点是功能无所不包，主要分为 Core、Dijit、DojoX 三大块。Core 提供 Ajax、events、packaging、CSS-based querying、animations、JSON 等相关操作 API。Dijit 是一个可更换皮肤，基于模板的 Web UI 控件库。DojoX 包括一些新颖的代码和控件，如 DateGrid、charts、离线应用和跨浏览器矢量绘图等，如下图所示。

JavaScript Remote Scripting 是较经典的远程脚本访问组件，支持将客户端数据通过服务器做代理进行远程的数据/操作交互。

Sarissa 封装了在浏览器端独立调用 XML 的功能。

第 2 时期，Prototype“王朝”，2005 年～2008 年。其间打了第 2 次浏览器“战争”，交战双方是 IE6、IE7、IE8 VS Firefox 1、Firefox 2、Firefox 3，最后 Firefox3 大胜。浏览器“战争”中，Prototype 积极进行升级，加入诸多 DOM 功能，因此，JSer 比之前好过多了。加之，有 rails、

script.aculo.us（一流的特效库）、Rico 等助阵，迅速占领了市场。

Prototype 时期，面向对象技术发展到极致，许多组件成套推出。DOM 特征发掘也有序进行，再也不依靠浏览器嗅探去刻意屏蔽某一个浏览器了。无侵入式 JavaScript 开发得到推崇，所有 JavaScript 代码都抽离到 JavaScript 文件，不在标签内“兴风作浪”了。

Prototype 的发展无可限量，直到 1.5 版本对 DOM 进行处理，这是一个错误[3]。比如它一个很好用的 API-getElementsByClassName，由于 W3C 的标准化，Prototype 升级慢了，它对 DOM 的扩展成为了它的“地雷”。

第 3 时代，jQuery 纪元，2008 年到现在（如下图所示）。

jQuery 则以包裹方式来处理 DOM，而且配合它的选择器引擎，若一下子选到 N 个元素，那么就处理 N 个元素，是集化操作，与主流的方式完全不一样。此外，它的方法名都起得很特别，人们一时很难接受。

> 2007 年 7 月 1 日，jQuery 发布了 1.1.3 版本，它的宣传是。
> （1）速度改良：Dom 的遍历比 1.1.2 版本快了大概 **800**%。
> （2）重写了事件系统：对键盘事件用更优雅的方法进行了处理。
> （3）重写了 effects 系统：提高了处理速度。

停滞不前的 Prototype 已经跟不上时代的节奏，jQuery 在 1.3x 版本时更换 Sizzle，更纯净的 CSS 选择器引擎，易用性与性能大大提高，被程序员一致看好的 mouseenter、mouseleave 及事件代理，也被整合进去，jQuery 就占据了市场。

3．JavaScript 框架分类

如果是从内部架构与理念划分，目前 JavaScript 框架可以划分为 5 类。

首先出现的是以命名空间为导向的类库或框架，如创建一个数组用 new Array()，生成一个

[3] 详见 Prototype 核心成员的反思：http://perfectionkills.com/whats-wrong-with-extending-the-dom/

对象用 new Object()，完全的 Java 风格，因此我们就可以以某一对象为根，不断为它添加对象属性或二级对象属性来组织代码，金字塔般地垒叠起来。代表作如早期的 YUI 与 EXT。

接着出现的是以类工厂为导向的框架，如著名的 Prototype，还有 mootools、Base2、Ten。它们基本上除了最基本的命名空间，其他模块都是一个由类工厂衍生出来的类对象。尤其是 mootoos 1.3 把所有类型都封装成 Type 类型。

第 3 种就是以 jQuery 为代表的以选择器为导向的框架，整个框架或库主体是一个特殊类数组对象，方便集化操作——因为选择器通常是一下子选择到 N 个元素节点，于是便一并处理了。JQuery 包含了几样了不起的东西：“无 new 实例化”技术，$(expr)就是返回一个实例，不需要显式地 new 出来；get first set all 访问规则； 数据缓存系统。这样就可以复制节点的事件了。此外，IIFE（Immediately-Invoked Function Expression）也被发掘出来。

第 4 种就是以加载器串联起来的框架，它们都有复数个 JavaScript 文件，每个 JavaScript 文件都以固定规则编写。其中最著名的莫过于 AMD。模块化是 JavaScript 走向工业化的标志。《Unix 编程艺术》列举的众多“金科玉律”的第一条就是模块，里面有言——“要编写复杂软件又不至于一败涂地的唯一方法，就是用定义清晰的接口把若干简单模块组合起来，如此一来，多数问题只会出现在局部，那么还有希望对局部进行改进或优化，而不至于牵动全身”。许多企业内部框架都基本采取这种架构，如 Dojo、YUI、kissy、qwrap 和 mass 等。

第 5 种就是具有明确分层构架的 MV*框架。首先是 JavaScript MVC（现在叫 CanJS），backbonejs 和 spinejs，然后更符合前端实际的 MVVM 框架，如 knockout、ember、angular、avalon、winjs。在 MVVM 框架中，原有 DOM 操作被声明式绑定取代了，由框架自行处理，用户只需专致于业务代码。

4. JavaScript 框架的主要功能

下面先看看主流框架有什么功能。这里面包含 jQuery 这个自称为库的东西，但它接近 9000 行，功能比 Prototype 还齐备。这些框架类库的模块划分主要依据它们在 github 中的源码，基本上都是一个模块一个 JavaScript 文件。

jQuery

jQuery 强在它专注于 DOM 操作的思路一开始就是对的，以后就是不断在兼容性，性能上进行改进。

jQuery 经过多年的发展，拥有庞大的插件与完善的 Bug 提交渠道，因此，可以通过社区的力量不断完善自身。

Prototype.js

早期的王者，它分为四大部分。

- 语言扩展。
- DOM 扩展。
- AJAX 部分。
- 废弃部分（新版本使用其他方法实现原有功能）。

Prototype.js 的语言扩展覆盖面非常广，包括所有基本数据类型及从语言借鉴过来的“类”。其中 Enumerable 只是一个普通的方法包，ObjectRange、PeriodicalExecuter，Templat 则是用 Class 类工厂生产出来的。Class 类工厂来自社区贡献。

mootools

它由于 API 设计得非常优雅，其官方网站上有许多优质插件，因此才没有在原型扩展的反对浪潮中没落。

RightJS

又一个在原型上进行扩展的框架。

MochiKit

一个 Python 风格的框架，以前能进世界前十名的。

Ten

日本著名博客社区 Hatena 的 JavaScript 框架，由 amachang 开发，受 Prototype.js 影响，是最早期以命名空间为导向的框架的典范。

mass Framework

它是一个模块化，以大模块开发为目标，jQuery 式的框架。

经过细节比较，我们很容易得出以下框架特征的结论。

- 对基本数据类型的操作是基础，如 jQuery 就提供了 trim、camelCase、each、map 等方法；Prototype.js 等侵入式框架则在原型上添加 camelize 等方法。
- 类型的判定必不可少，常见形式是 isXXX 系列。
- 选择器、domReady、Ajax 是现代框架的标配。
- DOM 操作是重中之重，节点的遍历、样式操作、属性操作也属于它的范畴，是否细分就看框架的规模了。
- brower sniff 已过时，feature detect 正被应用。不过特性侦测还是有局限性，如果针对

于某个浏览器版本的渲染 Bug、安全策略或某些 BUG 的修正，还是要用到浏览器嗅探。但它应该独立成一个模块或插件，移出框架的核心。

- 现在主流的事件系统都支持事件代理。
- 数据的缓存与处理，目前浏览器也提供 data-*属性进行这面的工作，但不太好用，需要框架的进一步封装。
- 动画引擎，除非你的框架像 Prototype.js 那样拥有像 script.aculo.us 这样顶级的动画框架做后盾，最好也加上。
- 插件的易开发和扩展性。
- 提供诸如 Deferred 这样处理异步的解决方案。
- 即使不专门提供一个类工厂，也应该存在一个名为 extend 或 mixin 的方法对对象进行扩展。jQuery 虽然没有类工厂，但在 jQuery UI 中也不得不增加一个，可见其重要性。
- 自从 jQuery 出来一个名为 noConflict 的方法，新兴的框架都带此方法，以求狭缝中生存。
- 许多框架非常重视 Cookie 操作。

最后感谢一下业内一些朋友的帮忙，要不是他们，书不会这么顺利地写出来。以下排名不分先后：玉伯、汤姆大叔、弹窗教主、貘大、linxz 、正则帝 abcd。这些都是专家级人物，在业界早已闻名遐迩。由于本人水平有限，书中难免存有不妥之处，请读者批评指正。

目　　录

第 1 章　种子模块……1
1.1　命名空间　1
1.2　对象扩展　3
1.3　数组化　4
1.4　类型的判定　6
1.5　主流框架引入的机制——domReady　14
1.6　无冲突处理　16
第 2 章　模块加载系统……18
2.1　AMD 规范　18
2.2　加载器所在路径的探知　19
2.3　require 方法　21
2.4　define 方法　27
第 3 章　语言模块……31
3.1　字符串的扩展与修复　31
3.2　数组的扩展与修复　43
3.3　数值的扩展与修复　50
3.4　函数的扩展与修复　56
3.5　日期的扩展与修复　61
第 4 章　浏览器嗅探与特征侦测……64
4.1　判定浏览器　64
4.2　事件的支持侦测　67
4.3　样式的支持侦测　69
4.4　jQuery 一些常用特征的含义　70
第 5 章　类工厂……72
5.1　JavaScript 对类的支撑　72
5.2　各种类工厂的实现　77
5.2.1　相当精巧的库—P.js…77
5.2.2　JS.Class……80
5.2.3　simple-inheritance……82
5.2.4　体现 JavaScript 灵活性的库——def.js……84
5.3　es5 属性描述符对 OO 库的冲击　88
第 6 章　选择器引擎……100

6.1 浏览器内置的寻找元素的方法 100
6.2 getElementsBySelector 102
6.3 选择器引擎涉及的知识点 106
6.4 选择器引擎涉及的通用函数 114
6.4.1 isXML ····· 114
6.4.2 contains ····· 115
6.4.3 节点排序与去重 ····· 117
6.4.4 切割器 ····· 121
6.4.5 属性选择器对于空白字符的匹配策略 ····· 123
6.4.6 子元素过滤伪类的分解与匹配 ····· 125
6.5 Sizzle 引擎 127
第 7 章 节点模块 ····· 137
7.1 节点的创建 138
7.2 节点的插入 149
7.3 节点的复制 155
7.4 节点的移除 158
7.5 innerHTML、innerText 与 outerHTML 的处理 161
7.6 一些奇葩的元素节点 164
7.6.1 iframe 元素 ····· 164
7.6.2 object 元素 ····· 174
7.6.3 video 标签 ····· 179
第 8 章 数据缓存系统 ····· 185
8.1 jQuery 的第 1 代缓存系统 185
8.2 jQuery 的第 2 代缓存系统 190
8.3 mass Framework 的第 1 代数据缓存系统 193
8.4 mass Framework 的第 2 代数据缓存系统 196
8.5 mass Framework 的第 3 代数据缓存系统 198
8.6 总结 199
第 9 章 样式模块 ····· 200

9.1 主体结构 201
9.2 样式名的修正 205
9.3 个别样式的特殊处理 206
9.3.1 opacity……206
9.3.2 user-select……208
9.3.3 background-position·208
9.3.4 z-index……209
9.3.5 盒子模型……210
9.3.6 元素的尺寸……211
9.3.7 元素的显隐……218
9.3.8 元素的坐标……222
9.4 元素的滚动条的坐标 228
第 10 章 属性模块……229
10.1 如何区分固有属性与自定义属性 231
10.2 如何判定浏览器是否区分固有属性与自定义属性 233
10.3 IE 的属性系统的三次演变 234
10.4 className 的操作 235
10.5 Prototype.js 的属性系统 240
10.6 jQuery 的属性系统 246
10.7 mass Framework 的属性系统 249
10.8 value 的操作 253
第 11 章 事件系统……256
11.1 onXXX 绑定方式的缺陷 257
11.2 attachEvent 的缺陷 258
11.3 addEventListener 的缺陷 259
11.4 Dean Edward 的 addEvent.js 源码分析 260
11.5 jquery1.8.2 的事件模块概览 263
11.6 jQuery.event.add 的源码解读 266
11.7 jQuery.event.remove 的源码解读 269
11.8 jQuery.event.dispatch 的源码解读 271

11.9 jQuery.event.trigger 的源码解读 276
11.10 jQuery 对事件对象的修复 280
11.11 滚轮事件的修复 286
11.12 mouseenter 与 mouseleave 事件的修复 290
11.13 focusin 与 focusout 事件的修复 293
11.14 旧版本 IE 下 submit 的事件代理的实现 295
11.15 oninput 事件的兼容性处理 296
第 12 章 章异步处理 ····· 298
12.1 setTimeout 与 setInterval 299
12.2 Mochikit Deferred 301
12.3 JSDeferred 309
12.3.1 得到一个 Deferred 实例310
12.3.2 Deferred 链的实现·312
12.3.3 JSDeferred 的并归结果 ····· 316
12.3.4 JSDeferred 的性能提速 ····· 318
12.4 jQuery Deferred 321
12.5 Promise/A 与 mmDeferred 327
12.6 JavaScript 异步处理的前景 334
第 13 章 数据交互模块 ····· 339
13.1 Ajax 概览 339
13.2 优雅地取得 XMLHttpRequest 对象 339
13.3 XMLHttpRequest 对象的事件绑定与状态维护 342
13.4 发送请求与数据 344
13.5 接收数据 346
13.6 上传文件 349
13.7 一个完整的 Ajax 实现 351
第 14 章 动画引擎 ····· 363

14.1 动画的原理 363
14.2 缓动公式 365
14.3 API 的设计 368
14.4 mass Framework 基于 JavaScript 的动画引擎 369
14.5 requestAnimationFrame 377
14.6 CSS3 transition 383
14.7 CSS3 animation 388
14.8 mass Framework 基于 CSS 的动画引擎 390
第 15 章 插件化 398
15.1 jQuery 的插件的一般写法 398
15.2 jQuery UI 对内部类的操作 401
15.3 jQuery easy UI 的智能加载与个别化制定 403
15.4 更直接地操作 UI 实例 406
第 16 章 MVVM 409
16.1 当前主流 MVVM 框架介绍 410
16.2 属性变化的监听 416
16.3 ViewModel 418
16.4 绑定 429
16.5 监控数组与子模板 437

第 10 章　属性模块

通常我们把对象的非函数成员叫做属性。对于元素节点来说，其属性大体分成两大类，固有属性与自定义属性（特性）。固有属性一般遵循驼峰命名风格，拥有默认值，并且无法删除。自定义属性是用户随意添加的键值对，由于元素节点也是一个普通的 JavaScript 对象，没有什么严格的访问操作，因此命名风格林林总总，值的类型也乱七八糟。但是随意添加属性显然不够安全，比如引起循环引用什么的，因此，浏览器提供了一组 API 来供人们操作自定义属性，即 setAttribute、getAttribute、removeAttribute。当然还有其他 API，不过这是标准套装，只有在 IE6、IE7 那样糟糕的环境下，我们才求助于其他 API，一般情况下这三个足矣。我们通称它们为 DOM 属性系统。DOM 属性系统对属性名会进行小写化处理，属性值会统一转字符串。

```
var el =  document.createElement("div")
el.setAttribute("xxx", "1")
el.setAttribute("XxX", "2")
el.setAttribute("XXx", "3")
console.log(el.getAttribute("xxx"))
console.log(el.getAttribute("XxX"))
```

IE6、IE7 会返回"1"，其他浏览器返回“3”。在前端的世界，我们真是走到哪都能碰到兼容性问题。这只是冰山一角，IE6、IE7 在处理固有属性时要求进行名字映射，比如 class 变成 className，for 变成 htmlFor。对于布尔属性（一些只返回布尔的属性），浏览器间的差异更大，具体如表 10.1 所示。

```
<input type="radio"  id="aaa">
<input type="radio" checked id="bbb">
<input type="radio" checked="checked" id="ccc">
<input type="radio" checked="true" id="ddd">
<input type="radio" checked="xxx" id="eee">

"aaa,bbb,ccc,ddd,eee".replace(/\w+/g,function( id ){
   var elem =  document.getElementById( id )
   console.log(elem.getAttribute("checked"));
})
```

表 10.1

	#aaa	#bbb	#ccc	#ddd	#eee
IE7	false	true	true	true	true

IE8	""	"checked"	"checked"	"checked"	"checked"

续表

IE9	null	""	"checked"	"true"	"xxx"
FF15	null	""	"checked"	"true"	"xxx"
Chrome23	null	""	"checked"	"true"	"xxx"

因此框架很有必要提供一些 API 来屏蔽这些差异性。但在 IE6 统治时期，这个需求并不明显，因为 IE6、IE7 不区分固有属性与自定属性，setAttribute 与 getAttribute 在当时的人看来只是一个语法糖，用 el.setAttribute("innerHTML","xxxx")与用 el.innerHTML = "xxx"效果一样，而且后者更方便。即使早期应用那么广泛的 Prototype.js，提供属性操作 API 也非常贫乏，只有 identify、readAttribute、writeAttribute、hasAttribute、classNames、hasClassName、addClassName、removeClassName、toggleClassName 与$F 方法。Prototype.js 也察觉到固有属性与自定义属性在 DOM 属性系统的差异，在它的内部，搞了个 Element._attributeTranslations。然而，Prototype.js 这个属性系统内部还是优先使用 el[name] 方式来操作属性，而不是 set/getAttribute（接下来的章节，我会分析它的实现）。

jQuery 早期的 attr 方法，其行为与 Prototype 一模一样。只不过 jQuery1.6 之前，是使用 attr 方法同时实现了读、写、删掉这三种操作。从易用性来说，不区分固有属性与自定义属性，由框架自动内部处理应该比 attr、prop 分家更容易接受。那么是什么逼迫 jQuery 这样做的呢？是选择器引擎。

jQuery 是最早以选择器为向导的类库。它最开始的选择器引擎是 Xpath 式，后来换成 Sizzle，以 CSS 表达式风格来选取元素。在 CSS2.1 引入了属性选择器[aaa=bbb]，IE7 也开始残缺支持。Sizzle 当然毫不含糊地实现了这语法。属性选择器是最早突破类名与 ID 的限制求取元素的。为了显摆它的强大，设计者让它拥有多种形态，满足人们各种匹配需要。比如，它可以只写属性名，[checked]，那么上例中的后四个元素都选中。[checked=true]选中第四个元素。[checked=xxx]选中第四个元素。true 与 xxx 都是用户在标签的预设值。而一字不差地取回这个预设值的工作也只有 setAttribute 才能做到。根据标准，setAttribute 是应该返回用户设置的那个字符串。而 el[xxx]这样的取法就不一定了，比如 el.checked 就返回布尔值，表示两种状态。这种通过状态取元素的方式就不归属性选择器管了，CSS3 又新设了个状态伪类满足人们的需求。

此外，属性还能以[name^=value]、[name*=value]、[name$=value]等更精致的方式来甄选元素，而这一切都建立在获取用户预设值的基础上。因此 jQuery 下了很大决心，把 prop 从 attr 切割出来。虽然为了满足用户的向前兼容需求，又偷偷地让 attr 做了 prop 的事，但以此为契机，jQuery 团队挖掘出更多兼容性问题与相应解决方案。元素内部撑起整个属性系统的 attributes 类数组属性也从幕后走到前台，为世人所知。

浏览器经过这么多年的发展，谁也说不清某个元素节点拥有多少个属性。for..in 循环也不行，因为它对不可遍历的属性无能为力。在 IE6、IE7 中，attributes 会包含上百个特性节点，不管你是用 setAttribute 定义的属性，还是以 el[xxx]=yyy 的定义的属性，还是没有定义的属性。

可惜到 IE8 与其他浏览器中，你只看到寥寥可数的几个特性节点。称为显式属性（specified attribute）。

显式属性就是被显式设置的属性，分两种情况，一种是写在标签内的 HTML 属性，一种是通过 setAttribute 动态设置的属性，这些属性不分固有还是自定义，只要设置了，就出现在 attributes 中。在 IE6、IE7 中，我们也可以通过特性节点的 specified 属性判定它是否被显式设置。在 IE8 或其他浏览器，我们想判定一个属性是否为显式属性，直接用 hasAttribute API 判定。

```
var  isSpecified = !"1"[0] ? function(el, attr){
     return el.hasAttribute(attr)
} : function(el, attr){
     var val = el.attributes[attr]
     return !!val && val.specified
}
```

此外，HTML5 对属性进行了更多的分类，从外包到不同的对象。dataset 对象装载着所有以 data-开头的自定义属性。classList 装载着元素的所有类名，并且提供一套 API 来操作它们。formData 装载着所有要提供到后台的数据，以表单元素的 name 值与 value 值构成的不透明对象。尽管如此，还是有大量属性是没有编制的，它们代表着元素的各种状态以及其他元素的联结。正因为如此，它们的值才五花八门，如 uniqueNumber、tabIndex、colspan、rowspan 为整数，designMode、unselectable、autocomplete 的值不是 on 就是 off，iframe 通过 frameborder 的值是 0 还是 1 决定显示边框，通过 scrolling 的值是 yes 还是 no 决定显示滚动条，表单元素的 form 属性总是指向其外围的表单对象，表单元素的 checked、disabled、readOnly 等属性总是返回布尔值……

面对如此庞杂的属性，主流框架也纷纷建立了对应的模块来整治它。在 jQuery 中有 attributes 模块，YUI3 是 dom-class、dom-attrs 模块，dojo 是 dom-attr、dom-prop、dom-class 模块。从内容来看，类名都被单独提出来处理，在 jQuery 中，表单元素的 value 也被单独提出来处理。Prototype.js 虽然没有划分出来，但对付一般属性有 readAttribute 与 writeAttribute，ID 有 identify，表单元素的 value 有$F，类名更是对应多个方法，这个阵营与 jQuery 的属性模块一模一样。在 1.5 之前，Prototype.js 的属性模块一直是 jQuery 属性模块的蓝本。

10.1 如何区分固有属性与自定义属性

在 jQuery、mass Framework 中，提供两组方法来处理它们。但用户首先要知道他正在处理的东西是何物。虽然我们可以在以下链接查到每个元素拥有什么方法与属性，但显然不是每个人都那么勤奋。

http://msdn.microsoft.com/library/ms533029%28v=VS.85%29.aspx

我们需要做些实验找出其规律。

```
 <!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta content="IE=8" http-equiv="X-UA-Compatible"/>
    <title>如何区分属性与特性 by  司徒正美</title>
    <script type="text/javascript">
      var test = function(){
        var a = document.getElementById("test");
        a.setAttribute("title", "title");
        a.setAttribute("title2", "title2");
        alert(a.parentNode.innerHTML);
      }
    </script>
  </head>
  <body>
    <p><strong id="aaa">司徒正美</strong></p>
    <p><button type="button" onclick="test()">点我，进行测试</button></p>
  </body>
</html>
```

IE8 下打印出：

```
<STRONG id=test title=title title2="title2">司徒正美</STRONG>
```

Firefox15、Chrome23、IE9 下打印出：

```
<strong title2="title2" title="title" id="test">司徒正美</strong>
```

再将 a.setAttribute("title", "title"); a.setAttribute("title2", "title2") 这两行改成 a.title = "title";a.title2 = "title2"。

IE8 下打印出：

```
<STRONG id=test title=title title2="title2">司徒正美</STRONG>
```

Firefox15、Chrome23、IE9 下打印出：

```
<strong title2="title2" title="title" id="test">司徒正美</strong>
```

经过观察，旧版本 IE 下固有属性，如 title、id 是不带引号的，自定义属性是带引号的，但在标准浏览器下，我们无法区分，否决此方案。

在元素中有一个 attributes 属性，里面有许多特性节点，这些特性节点在 IE 下有一种名为 expando 的布尔属性，可以判定它是否为自定义属性。但在标准浏览器下没有此属性，否决此方案。

```
function isAttribute(attr, host){ //仅有 IE
  var attrs =  host.attributes;
  return attrs[attr] && attrs.expando == true
```

```
}
```

我们再换一个角度来看，如果是固有属性，以 el[xxx] = yyy 的形式赋值，再用 el.getAttribute() 来取值，肯定能取到东西，但自定义属性就不一样。

```
var a = document.getElementById("test");
a.title = 222
console.log(a.getAttribute("title"))       //"222"
console.log(typeof a.getAttribute("title")) //"string"

a.setAttribute("custom", "custom")
console.log(a.custom)         //undefined
console.log(typeof a.custom)   //"undefined"
```

不过要注意 IE6、IE7 下的特例：

```
a.setAttribute("innerHTML","xxx")
console.log(a.innerHTML)          //"xxx"
```

即使如此，我们也可以轻松绕过这个陷井，建一个干净的同类型元素作为测试样本就行了。

```
var a = document.createElement("div")
console.log(a.getAttribute("title"))
console.log(a.getAttribute("innerHTML"))
console.log(a.getAttribute("xxx"))
console.log(a.title)
console.log(a.innerHTML)
console.log(a.xxx)
```

IE6、IE7 下返回"","", null, "","",undefined。IE8、IE9、Chrome23、FF15，Opera12 下返回 null，null,null,"","",undefined。

因此我们可以推导出这样一个方法，回答我们这一节的标题。

```
function isAttribute(attr, host){
   //有些属性是特殊元素才有的，需要用到第二个参数
   host = host || document.createElement("div");
   return host.getAttribute(attr) === null && host[attr] === void 0
}
```

10.2 如何判定浏览器是否区分固有属性与自定义属性

经过社区的努力，现在大家都知道 IE6、IE7 不区分固有属性与自定义属性。这带来的结果是，对某个固有属性进行 setAttribute，我们不需要名字映射就能生效。但如果想通过浏览器嗅探法来识别 IE6、IE7，是远远不够的，因为用户可能使用旧版的标准浏览器上网。另外，我们也不得不考虑国内可恶的加壳 IE 浏览器。因此我们最好是通过特征侦测来判定浏览器是否支持此特性。

mootools 与 jQuery 各自使用了两种截然不同的方法来判定。mootools 以属性法设置一个

自定义属性，然后通 getAttribute 去取，看是不是等于预设值，是就证明它不区分固有属性与自定义属性。jQuer 则是先用 setAttribute 去设置 className，然后看它是当作固有属性还是自定义属性，如果是自定义属性，我们用 el.className 是取不到值的，具体如表 10.2 所示。

```
var el = document.createElement("div")
el.random = 'attribute';            //mootools
console.log(el.getAttribute("random") != 'attribute')

el.setAttribute("className", "t");     //jQuery
console.log(el.className !== "t")
```

表 10.2

	IE6	IE7	IE8	FF9	FF15	Chrome23
Mootools	false,	false	false	true	true	true
jQuery	false	false	true	true	true	true

要看哪个更精确，只需要 IE8 浏览器就行了。IE8 大肆重写内核，是区分固有属性与自定义属性的，因此 el.className 应该返回 undefined，导致结果为 true，因此 jQuery 获胜。jQuery 把这个特性称之为 getSetAttribute，意即 get/SetAttribute 没有 BUG； mass Framework 称之为 attrInnateName,意即不需要名字映射用原名就可以取值。它们都位于 supports 模块中。

10.3 IE 的属性系统的三次演变

微软在 IE4（1997 年）添加 setAttribute、getAttribute API，当时，DOM 标准（1998 年）还没有出来呢！而它的对手 NS6 到 2000 年才难产出来。

早期的 DOM API 于微软来说，只是它已有的一些方法的再包装，这些包装方法无法与它原来的那一套相媲美。document.getElementById("xxx")取元素节点，IE 下，这些带 ID 的元素节点自动就映射成一个个全局变量，直接 xxx 就拿到元素节点了，很便捷，或者使用 document.all[ID]来取，无论哪种都比标准的短；又如 getElementsByTagName，IE 下有 document.all.tags()方法，此方法直到 IE9 还有效。而 setAttribute("xxx","yyy")与 var ret = getAttribute("xxx")只不过是 el.xxx = "yyy"与 var ret = el.xxx 的另一种操作形式罢了。明白这一点我们就立即理解 IE 下这两个 API 的一些奇怪行为了。

el.setAttribute("className","aaa")是可行的，但 el.setAttribute("class","aaa")失败，因为我们可以用 className 修改类名，但不能用 class。

el.setAttribute("onclick", Function("alert(1)")) 是可行的，但 el.setAttribute("onClick", Function("alert(1)"))失败，因为我们是通过 el.onclick 绑定事件，而不是 el.onClick。

el.setAttribute("innerHTML"," test ")是可行的，因为我们可以用 innerHTML 添加内容。

element.setAttribute("style", "background-color: #fff; color: #000;")失败，因此 style 在 IE 下是个对象，"background-color: #fff; color: #000;"只能作为它的 cssText 属性的值。

DOM level 1 隔年就制定出来了。setAttribute/getAttribute 并没有微软想象得那么简单，它早期规定 getAttribute 必须也返回字符串，就算不存在也是空字符串。到后来，setAttribute/getAttribute 会对属性名进行小写化处理。用 getAttribute 去取没有显式设置的固有属性时，返回默认值（多数时候它为 null 或空字符串），对于没有显式设置的自定义属性，则返回 undefined。于是微软傻了眼，第一次改动就是匆匆忙忙支持小写化处理，并在 getAttribute 方法添加第二参数，以实现 DOM1 的效果。

getAttribute 的第二个参数有四个预设值：0 是默认，照顾 IE 早期的行为；1 属性名区分大小写；2 取出源代码中的原字符串值(注，IE5～IE7 对动态创建的节点没效，IE5～IE8 对布尔属性无效)；4 用于 href 属性，取得完整路径。

第二次演变是区分固有属性与自定义属性，取类名再也不用 className 了；布尔属性则遵循一个奇怪的规则，只要是显式设置了就返回与属性名同名的字符串，没有则返回""。我相信早期的标准浏览器也是这样做的。但标准浏览器很快就变脸了，统一返回用户的预设值。IE8 变得两边都不讨好，尽管它一心想与标准保持一致，标榜自己才是最标准的。比如它在当时还推出了 Object.defineProperty、querySelector、postMessage 等具有革命意义的新 API，但它们都与 W3C 争吵完的结果有出入。

第三次演变，不再对属性值进行干预，用户设什么返回什么，忠于用户的决定。对于这尘埃落定的方案，IE9 终于与标准吻合了。这也说明 IE 这种慢吞吞的大版本发布方式已经落伍了，虽然大家都对 FF 的版本狂飙有微词，但人家却保住了与 Chrome 叫板的地位。

综观 IE 的属性系统的悲剧，都因为微软总想抢占先机，而又与标准同步太慢所致。

第 11 章　事件系统

事件系统是一个框架非常重要的部分，用于响应用户的各种行为。

浏览器提供了三种层次的 API。

最原始的是写在元素标签内。

再次是在脚本中，以 el.onXXX= function 绑定的方式，通称为 DOM0 事件系统。

最后是多投事件系统，一个元素的同一类型事件可以绑定多个回调，通称为 DOM2 事件系统。由于浏览器大战，现存在两套 API。

- IE 与 Opera 方

绑定事件：el.attachEvent(“on”+type，callback)

卸载事件：el.detachEvent(“on”+type，callback)

创建事件：document.createEventObject()

派发事件：el.fireEvent(type, event)

- W3C 方

绑定事件：el.addEventListener(type, callback, [phase])

卸载事件：el.removeEventListener(type, callback, [phase])

创建事件：el.createEvent(types)

初始化事件：event.initEvent()

派发事件：el.dispatchEvent(event)

从 API 的数量与形式来看，W3C 提供的复杂很多，相对应也强大很多，下面我会逐一详述。

首先呈上我几个简单的实现。如果是简单的页面，就用简单的方式打发，没有必要动用框架。不过，事实上，整个事件系统就建立在它们的基础上。

```
function addEvent(el, type, callback, useCapture  ){
   if(el.dispatchEvent){//W3C 方式优先
      el.addEventListener( type, callback, !!useCapture  );
   }else {
      el.attachEvent( "on"+type, callback );
   }
   return callback;//返回 callback 方便卸载时用
```

```
}

function removeEvent(el, type, callback, useCapture  ){
   if(el.dispatchEvent){//W3C 方式优先
      el.removeEventListener( type, callback, !!useCapture  );
   }else {
      el.detachEvent( "on"+type, callback );
   }
}

function fireEvent(el, type, args, event){
   args = args || {}
   if(el.dispatchEvent){//W3C 方式优先
      event = document.createEvent("HTMLEvents");
      event.initEvent(type, true, true );
   }else {
      event = document.createEventObject();
   }
   for(var i in args)  if(args.hasOwnProperty(i)){
      event[i] = args[i]
   }
   if(el.dispatchEvent){
      el.dispatchEvent(event);
   }else{
      el.fireEvent('on'+type,event)
   }
}
```

11.1 onXXX 绑定方式的缺陷

onXXX 既可以写在 HTML 标签内，也可以独立出来，作为元素节点的一个特殊属性来处理。不过作为一种古老的绑定方式，它很难预测到后来人对这方面的扩展。

总结起来有以下不足。

（1）onXXX 对 DOM3 新增事件或 FF 某些私有实现无法支持，主要有以下事件：

DOMActivate
DOMAttrModified
DOMAttributeNameChanged
DOMCharacterDataModified
DOMContentLoaded
DOMElementNameChanged
DOMFocusIn
DOMFocusOut
DOMMouseScroll
DOMNodeInserted
DOMNodeInsertedIntoDocument
DOMNodeRemoved

DOMNodeRemovedFromDocument

DOMSubtreeModified

MozMousePixelScroll

不过稍安勿躁，上面的事件就算是框架，也只用到 DOMContentLoaded 与 DOMMouseScroll。DOMContentLoaded 用于检测 DomReady，DOMMouseScroll 用于在 FF 模拟其他浏览器的 mousewheel 事件。

（2）onXXX 只允许元素每次绑定一个回调，重复绑定在冲掉之前的绑定。

（3）onXXX 在 IE 下回调没有参数，在其他浏览器下回调的第一个参数是事件对象。

（4）onXXX 只能在冒泡阶段可用。

11.2 attachEvent 的缺陷

attachEvent 是微软在 IE5 添加的 API，Opera 也支持，也对于 onXXX 方式，它可以允许同一元素同种事件绑定多个回调，也就是所谓的多投事件机制。但它带来的麻烦只多不少，存在以下几点缺陷。

（1）IE 下只支持微软系的事件，DOM3 事件一概不能用。

（2）IE 下 attchEvent 回调中的 this 不是指向被绑定元素，而是 window!

（3）IE 下同种事件绑定多个回调时，回调并不是按照绑定时的顺序依次触发的！

（4）IE 下 event 事件对象与 W3C 的存在太多差异了，有的无法对上号，比如 currentTarget。

（5）IE 还是只支持冒泡阶段。

不过 IE 还是不断进步的。

```
<!DOCTYPE html>
<html>
   <head>
      <title></title>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <script>
         window.attachEvent("onload", function(e) {
            alert(e);
         });
      </script>
   </head>
   <body>
      <div>TODO write content</div>
   </body>
</html>
```

IE6、IE7（见图 11.1）

IE8（见图 11.2）

▲图 11.1

▲图 11.2

IE9（见图 11.3）

▲图 11.3

从一开始不知弹出什么东西，到现在明确这是一个事件对象。不过，W3C 的事件系统已经是大势所趋，微软也 hold 不住，从 IE9 起支持 W3C 那一套 API，这对我们实现事件代理非常有帮助！

11.3 addEventListener 的缺陷

W3C 这一套 API 也不是至善至美，毕竟标准总是滞后于实现，剩下的那四个标准浏览器各有自己的算盘，它们之间亦有不一致的地方。

（1）新事件非常不稳定，可能还没有普及开就被废弃。在早期的 Sizzle 选择器引擎中，有这么几句：

```
document.addEventListener(“DOMAttrModified”，invalidate, false);
document.addEventListener(“DOMNodeInserted”，invalidate, false);
document.addEventListener(“DOMNodeRemoved”，invalidate, false);
```

现在这三个事件被废弃了（准确来说，所有变动事件都完蛋了），FF14 与 Chrome18 开始用 MutationObserver 代替它。

（2）Firefox 既不支持 focusin、focus 事件，也不支持 DOMFocusIn、DOMFocusOut，直接现在也不愿意用 mousewheel 代替 DOMMouseScroll。Chrome 不支持 mouseenter 与 mouseleave。

因此不要以为标准浏览器就肯定会实现 W3C 钦定的标准事件，它们也有抗旨的时候，特征侦测必不可少。最恶心的是国内一些浏览器套用 webkit 内核，为了“超越”本版浏览器的

HTML5 跑分（HTML5test.com），竟然实现了一些无用的空接口来骗过特征侦测，因此必要时，我们还得使用非常麻烦的功能侦测来检测浏览器是否支持此事件。

（3）CSS3 给私有实现添加自定前缀标识的坏习惯也漫延到一些与样式息息相关的事件名上。比如 transitionend 事件名，这个后缀名与大小写混合成 5 种形态，相当棘手。

（4）第三、第四、第五个参数。

第三个参数 useCapture 只有非常新的浏览器中才是可选项，比如 FF6 或之前是必选的，为安全起见，请确保第三个参数为布尔。

第四个参数听说是 FF 专有实现，允许跨文档监听事件。

第五个参数？的确存在第五参数，不过它只存在于 Flash 语言的同名方法中。现在前端工程师还是要求助于 Flash，就作为一个知识点收下吧。有的面试官跨界考这种东西。在 Flash 下，addEventListener 的第四个参数用于设置该回调执行时的顺序，数字大的优先执行，第五个参数用来指定对侦听器函数的引用是弱引用还是正常引用。

（5）事件对象成员的不稳定。

W3C 那套是从浏览器商抄来的，人家都用了这么久，难免与标准不一致。

FF 下 event.timeStamp 返回 0 的问题，这个 BUG,2004 年就有人提交了，直到 2011 年才被修复。https://bugzilla.mozilla.org/show_bug.cgi?id=238041

Safari 下 event.target 可能是返回文本节点。

event.defaultPrevented，event.isTrusted 与 stopImmediatePropagation 的可用性很低，它们属于 DOM3 event 规范。

defaultPrevented 属性是用于确定事件对象有没有调用 preventDefault 方法。之前标准浏览器都统一用 getPreventDefault 方法来干这事，在 jQuery 源码中，你会发现它是用 isDefaultPrevented 方法来做。不过，isDefaultPrevented 的确曾列入 W3C 草稿，可参见这里：

http://www.w3.org/TR/2003/WD-DOM-Level-3-Events-20030331/ecma-script-binding.html

isTrusted 属性用于表示当前事件是否是由用户行为触发（比如说真实的鼠标点击触发一个 click 事件），还是由一个脚本生成的（使用事件构造方法，比如 event.initEvent）。

stopImmediatePropagation 用于阻止当前事件的冒泡行为并且阻止当前事件所在元素上的所有相同类型事件的事件处理函数的继续执行。

IE9+全部实现（但是，IE9、IE10 的 event.isTrusted 有 BUG。link.click()后返回的也是 true）。

Chrome5～Chrome17 部分实现（event.isTrusted 未支持）。

Safari5 才部分实现（event.isTrusted 未支持）。

Opera10、Opera11 部分实现（stopImmediatePropagation 以及 event.isTrusted 未实现，而仅仅实现了 defaultPrevented）。

Opera12 部分实现（stopImmediatePropagation 仍然未实现，但实现了 e.isTrusted）。

Firefox1.0～Firefox5，stopImmediatePropagation 和 defaultPrevented 未实现，仅仅实现了 event.isTrusted。isTrusted 在成为标准前，是 Firefox 的私有实现。

Firefox6～Firefox10，仅未实现 stopImmediatePropagation。Firefox11，终于实现了 stopImmediate Propagation。

（6）标准浏览器没有办法模拟像 IE6～IE8 的 propertychange 事件。

虽然标准浏览器有 input、DOMAttrModified、MutationObserver，但比起 propertychange 都弱爆了。propertychange 可以监听多种属性变化，而不单单是 value 值，另外它不区分 attribute 和 property，因此你无论是通过 el.xxx = yyy,还是 el.setAttribute(xxx, yyy)都接触过此事件。具体可参阅下面这篇博文。

http://www.cnblogs.com/rubylouvre/archive/2012/05/26/2519263.html

第 12 章　异步处理

浏览器环境与后端的 nodejs 存在着各种消耗巨大或堵塞线程的行为，对于 JavaScript 这样单线程的东西唯一的解耦方法就提供异步 API。异步 API 是怎么样的呢？简单来说，它是不会立即执行的方法。比方说，一个长度为 1000 的数组，在 for 循环内，可能不到几毫秒就执行完毕，若在后端的其他语言，则耗时更少。但有时候，我们不需要这么快的操作，我们想在页面上能用肉眼看到它执行的每一步，那就需要异步 API。还有些操作，比如加载资源，你想快也快不了，它不可能一下子提供给你，你必须等待，但你也不能一直干等下去什么也不干，得允许我们跳过这些加载资源的逻辑，执行下面的代码。于是浏览器首先搞出的两个异步 API，就是 setTimeout 与 setInterval。后面开始出现各种事件回调，它只有用户执行了某种操作后才触发。再之后，就更多，XMLHttpRequest、postMessage、WebWorkor、setImmediate、requestAnimationFrame 等。

这些东西都有一个共同的特点，就是拥有一个回调函数，描述一会儿要干什么。有的异步 API 还提供了对应的中断 API，比如 clearTimeout、clearInterval、clearImmediate、cancelAnimationFrame。

早些年，我们就是通过 setIimeout 或 setInterval 在网页上实现动画的。这种动画其实就是通过这些异步 API 不断反复调用同一个回调实现的，回调里面是对元素节点的某些样式进行很小范围的改动。

随着 iframe 的挖掘与 XMLHttpRequest 的出现，无缝刷新让用户驻留在同一个页面上的时间越来越长，许多功能都集成在同一个页面。为实现这些功能，我们就得从后端加载数据与模板，来拼装这些新区域。这些加载数据与模板的请求可能是并行的，可能是存在依赖的。只有在所有数据与模板都就绪时，我们才能顺利拼接出 HTML 子页面插入到正确的位置上。面对这些复杂的流程，人们不得不发明一些新模式来应对它们。最早被发明出来的是“回调地狱（callback hell）”，这应该是一个技能。事实上，几乎 javascript 中的所有异步函数都用到了回调，连续执行几个异步函数的结果就是层层嵌套的回调函数以及随之而来的复杂代码。因此有人说，回调就是程序员的 goto 语句[4]。

此外，并不是每一个工序都是一帆风顺的，如果有一个出错了呢，对于 JavaScript 这样单

[4] http://tirania.org/blog/archive/2013/Aug-15.html

线程的语言，往往是致命的，必须 try...catch，但 try...catch 语句只能捕捉当前抛出的异常，对后来执行的代码无效。

```
function throwError() {
    throw new Error('ERROR');
}
try {
    setTimeout(throwError, 3000);
} catch (e) {
    alert(e);//这里的异常无法捕获
}
```

这些就是本章所要处理的课题。不难理解，domReady、动画、Ajax 在骨子里都是同一样东西，假若能将它们抽象成一个东西，显然是非常有用的。

12.1 setTimeout 与 setInterval

首先我们得深入学习一下这两个 API。一般的书藉只是简单介绍它们的用法，没有对它们内在的一些隐秘知识进行描述。它们对我们创建更有用的异步模型非常有用。

（1）如果回调的执行时间大于间隔间隔，那么浏览器会继续执行它们，导致真正的间隔时间比原来的大一点。

（2）它们存在一个最小的时钟间隔，在 IE6～IE8 中为 15.6ms[5]，后来精准到 10ms，IE10 为 4ms，其他浏览器相仿。我们可以通过以下函数大致求得此值。

```
function test(count, ms) {
    var c = 1;
    var time = [new Date() * 1];
    var id = setTimeout(function() {
        time.push(new Date() * 1);
        c += 1;
        if (c <= count) {
            setTimeout(arguments.callee, ms);
        } else {
            clearTimeout(id);
            var tl = time.length;
            var av = 0;
            for (var i = 1; i < tl; i++) {
                var n = time[i] - time[i - 1]; //收集每次与上一次相差的时间数
                av += n;
            }
            alert(av / count); // 求取平均值
        }
    }, ms);
}
winod.onload = function() {
    var id = setTimeout(function() {
```

[5] http://ie.microsoft.com/testdrive/Performance/setImmediateSorting/Default.html

```
        test(100, 1);
        clearTimeout(id);
    }, 3000);
}
```

具体如表 12.1 所示。

表 12.1

Firefox 3.6.3	Firefox 18.1	Chrome 10.53	Chrome 10.53	Opera 12.41	Safari 5.01	IE 8	IE 10
15.59	3.98	3.92	3.6	4.01	4.12	15.91	3.91

但上面的数据很难与官方给出的数值一致，因为它太容易受外部因素影响，比如电池快没电了，同时打开的应用程序太多了，导致 CPU 忙碌，这些都会让它的数值偏高。

如果嫌旧版本 IE 的最短时钟间隔太大，我们或许有办法改造一下 setTimeout，利用 image 死链时立即执行 onerror 回调的情况进行改造。

```
var orig_setTimeout = window.setTimeout;
window.setTimeout = function (fun, wait) {
    if (wait < 15) {
        orig_setTimeout(fun, wait);
    } else {
        var img = new Image();
        img.onload = img.onerror = function () {
            fun();
        };
        img.src = "data:,foo";
    }
};
```

（3）有关零秒延迟，此回调将会放到一个能立即执行的时段进行触发。JavaScript 代码大体上是自顶向下执行，但中间穿插着有关 DOM 渲染、事件回应等异步代码，它们将组成一个队列，零秒延迟将会实现插队操作。

（4）不写第二参数，浏览器自动配时间，在 IE、Firefox 中，第一次配可能给个很大数字，100ms 上下，往后会缩小到最小时钟间隔，Safari、Chrome、Opera 则多为 10ms 上下。Firefox 中，setInterval 不写第二参数，会当作 setTimeout 处理，只执行一次。

```
window.onload = function() {
    var a = new Date - 0;
    setTimeout(function() {
        alert(new Date - a);
    });
    var flag = 0;
    var b = new Date,
        text = ""
    var id = setInterval(function() {
        flag++;
        if (flag > 4) {
            clearInterval(id)
```

```
            console.log(text)
        }
        text += (new Date - b + " ");
        b = new Date
    })
}
```

（5）标准浏览器与 IE10，都支持额外参数，从第三个参数起，作为回调的传参传入！

```
setTimeout(function() {
 alert([].slice.call(arguments));
}, 10, 1, 2, 4);
```

IE6～IE9 可以用以下代码模拟。

```
if (window.VBArray && !(document.documentMode > 9)) {
    (function(overrideFun) {
        window.setTimeout = overrideFun(window.setTimeout);
        window.setInterval = overrideFun(window.setInterval);
    })(function(originalFun) {
        return function(code, delay) {
            var args = [].slice.call(arguments, 2);
            return originalFun(function() {
                if (typeof code == 'string') {
                    eval(code);
                } else {
                    code.apply(this, args);
                }
            }, delay);
        }
    }
    );
}
```

（6）setTimeout 方法的时间参数若为极端值（如负数、0、或者极大的正数），则各浏览器的处理会出现较大差异，某些浏览器会立即执行。幸好最近所有最新的浏览器都立即执行了。

JavaScript 框架设计

乔布斯说过："真正的魔法，是用五千个点子打磨出一个产品。"我想我们很幸运，因为我们正要去做这样一件事：凝聚前人的智慧，实现一个真正的魔法。本书将引导大家了解近10年来大师们打造的优良框架以及其中令人称道的奇思妙想。让我们更深入、更彻底地认识JavaScript，领略jQuery等库的架构之美和设计之美，高屋建瓴地打造适合自己的前端框架！

作者简介

钟钦成 网名司徒正美，著名的JavaScript专家，立志做考古学家的日语系工程师，穿梭于二次元与二进制间的"魔法师"，做过陶艺，写过小说，涉猎Java、Ruby、 JavaScript，3年成就此书！

推荐媒介

博客园 cnblogs.com　51CTO.com 技术成就梦想　ITeye

封面图介绍

圣家族大教堂，西班牙巴塞罗那最负盛名的现代建筑和地标性建筑，也是世界上唯一一座尚未完成便被列入世界文化遗产的建筑。整体设计以大自然为灵感，是一个充满韵律动感的神奇建筑。

人民邮电出版社－信息技术分社
http://weibo.com/ptpitbooks

ISBN 978-7-115-34358-1

分类建议：计算机／程序设计／JavaScript
人民邮电出版社网址：www.ptpress.com.cn

封面设计：任文杰