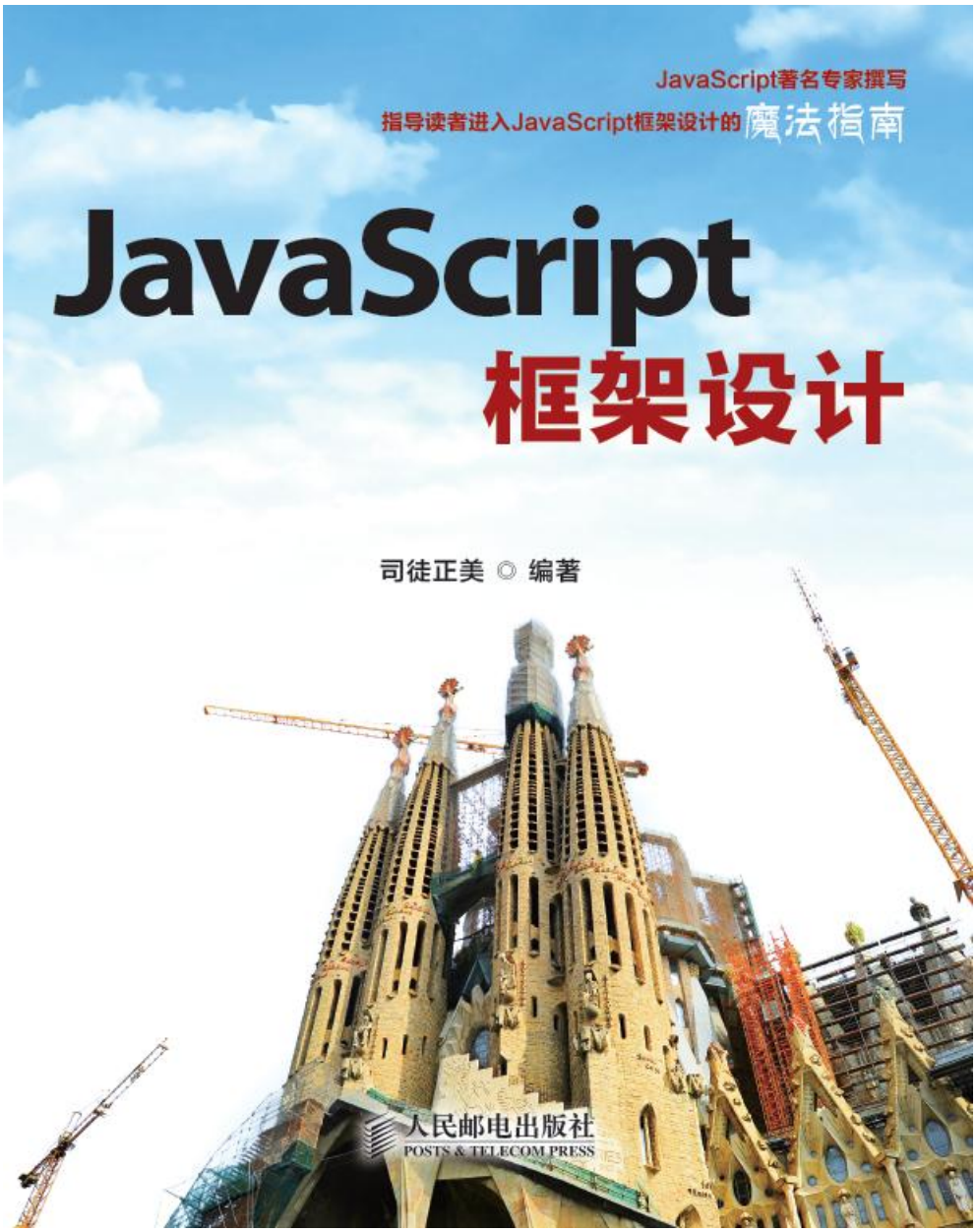

JavaScript著名专家撰写
指导读者进入JavaScript框架设计的魔法指南
JavaScript
框架设计
司徒正美 ◎ 编著
人民邮电出版社
POSTS & TELECOM PRESS

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　杨林杰

◆ 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：28.75

字数：699 千字　　2014 年 4 月第 1 版

印数：1 – 3 500 册　　2014 年 4 月北京第 2 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

自序

——做一个不可替代的架构师

历时两年多，《JavaScript 框架设计》终于付梓出版了。应各方面的要求，特写一篇序，隆重介绍一下此书对各位程序员的钱途有什么帮助及阅读顺序等疑问。作为国内第一本讲述前端框架构的书，它里面充斥着许多大家前所未闻的知识，这些知识有 50%只见于 github 的 issue，讲述各种隐秘的浏览器兼容性问题及各种神奇的修复方案，或者是某些危险但美丽的黑魔法，另外 50%我深夜梦游般在外国某些大牛（不局限于英语，有日语、俄语、韩语等，不同的语言的人，受制了他们的语法结构，他们的思考回路是与我们不同的，给出的答案有时真的是拍案叫绝）的博客或网站寻觅的神奇东东。在我通过编写 dom Framework、mass Framework 这两大框架（dom Framework 是老式的金字塔式的基于类的大框架，mass Framework 是拥有 AMD 加载器的开放式框架，再到后来的 avalon，是鬼怪式的分层构架的 MVVM 框架，黑魔法满满的），建立完整的知识树后，狂热的心情如征服六国后的秦王，热衷于收集各种奇珍异宝于我的博客——你们看的部分，只是我未公开的十分之一而已，本书将额外开恩公布另外的十分之二。

其实这世界很奇妙，当你水平上去后，就算你不想继续，这世界也推动着你前进。就像 jQuery1.3 通过 Sizzle 大获成功后，成为世界的明星，就算 John Resig 想撒手，但这么多 pull request，它也只能从更好变成更加好！三年前，当我写了三百多篇的博文时，出版社已经找上门来了约稿了。在那半年内，总共有三四家来找我，让我看到前端的希望。于是我的重心由 ruby 慢慢转向 JavaScript。待到我加入盛大创新院后，我已经确保我能 hold 住《javascript 框架设计》这个大题目，于是签约写书。不过，在最初我提交给出版社的目录里，我有着更为恢宏的目标，包含拖放组件、路由系统，及各式 UI 组件，但最后由于篇幅的问题，只好说声抱歉了。

愿望总是被现实所掣肘，亚历山大想证明世界，可他的 HP 也是这么短，经不起长途跋涉。实体书与博客是不同的，它必须要形成一个体系，文体也有要求，不能太口语化，什么喜乐哀痛必须收起来，板着脸正正经经地对大家宣讲。错别字也要收殓一下，虽然我找了许多高手审稿。可惜个个都开写轮眼，自动过滤掉错别字，最终还是让出版社的张涛编辑帮忙处理了大部分错别字与病句，太专业的东西他也无能为力，但愿不影响阅读。幸好各位大侠贡献了不少冷癖有用的知识点，让本书充实了不少，因此才一改再改，三番四次，导致两年多才交稿。在盛大创新院时，不断有人（同事或群友）问我的书什么时候出来，人家半年就搞出一本了。我今天终于不为这问题烦恼了。

在继续这篇软文时，我脑海真的是闪现许多词汇，什么大教堂与集市、造轮子与 DRY、公司利益与个人成长……一个个来吧！

大教堂与集市说的是如何构建一个软件工程，是大教堂式的专制主义还是开放式的以众包方式让大家贡献源码。前者，缓和一点地说，是英雄主义，jQuery 之于 John Resig，Node.JS 之于 Ryan Dahl，他们单枪匹马开创了一个新天地。后者，最杰出的代表是 Linux。但在这个互相浸透学习的世界，绝对的东西是不存在的。jQuery 与 Node.JS 现在也是在众多的项献者的努力下前进，原作者向新目的地进发了。在国内，你懂的，每个人都自命不凡，一开始只能是你一个踽踽独行。只有你真正成为明灯式的人物后，才有人追随。

造轮子与 DRY 这问题在新浪微博上也吵过许多次了。有个软件设计原则叫 DRY，防止组员们随意克隆代码，或在不知情的情况下重复发明相似的功能模块。公司出于利益的考虑，也不愿给出更多时间造轮子，上网找一个 jQuery 插件了事。因此，中小公司的页面非常恐怖，允斥着

大量第三方插件，而相对而言，大阿狸能用的基本自己做，这正是游击队与正规军的区别。从国内看，最重视 JavaSript 的公司也恰恰是大阿狸，他们拥有国内最庞大优秀的前端团队（700 多人），小公司还是一个前端对 20 个后端的节奏。招这么多高手干嘛呢？造轮子！当然，这不是一个轮子所代表的，这涉及一整套的工具链，目的是实现前端自动化集成布署。写框架与 UI 组件是其中一个很少环节，这也是一般人能理解的东西，更多高大上的东西，大公司也不会公开出来。但你起码拥有创造 UI 组件这样的能力，你才能有使用更底层的工具的能力。

公司利益与个人成长，这个更不用说。只有目光短浅的公司，才会用杂牌的组件写程序。大公司早已为你准备一整套东西了。而你的任务就是成长到具有写 UI 组件的地步，进入架构师，为公司的未来挑战做好更多准备（工具）。HTML5 对于一般人而言，好像是非常遥远的事，但大公司早已有一帮人用它做出许多好东西，为公司产品的用户体验添砖加瓦。为了积聚这实力，你必须自己暗暗发力，偷偷自己写一套东西。之前人家写过的弹出层、富文本编辑器、语法高亮插件……你一套也不能少，这样你才能接触到之前碰不到的原生 API 与知识点。如弹出层有关垂直居中的 CSS 知识点、select 穿透问题、富文本编辑器用到的 iframe 知识点、Range 与 Selection 对象的知识点，语法高亮则是你正则的大检验！如果写业务代码，你写十年，水平还是那样。因此有句话说——“用一年的经验混十年”。

最近在微博看到一件可怕的事：

【我所了解的一个精神失常的程序员】不久前我们公司有个程序员精神失常。他走进经理办公室开始大喊大叫，说着一些奇怪的事情。如果不是了解他，就会以为他磕了药。但是事实上他简直就是精神失常了！

他是我在编程行业见过的最勤奋员工。他经常在下班后加班，周末的时候，当管理人员需要人手去处理紧急工作时，他总是随叫随到。在这个阶段公司并不赚钱，老板需要尽可能快的完成项目，所以任何被客户急催的软件开发都会自动分配给他。他很乐意地全心投入把工作做完整是老板喜欢的地方。

“我能力强，我效率高，我应该是公司的关键人物”，其实那是错的，不可替代性才是最重要的。如果靠“卖力”增加不可替代性，作用是微乎其微的，还是得靠“高门槛”。我认识的工程师里，越是技术好的工程师越会意识到这个问题，然后去做一些“深度”的发展，这也算是工程师的自我保护吧。说什么做 IT 没前途，30 岁要转行，这只是无能者的藉口。

古人说——“人无远虑，必有近忧”。你平时有这么多空闲时间，为何不努力提升一下自己的水平呢。不去认真阅读一下大师们的框架，不自己写一个框架。记得当初我在博客宣传我的框架，被某个嫉妒的人骂个狗血喷头，两年过去，他消声匿迹，而我，从一个公司的核心前端变为另一个公司的核心前端，现在是去哪儿网的前端架构师。因此要相信自己！不要怕这怕那，有努力就有回报！

再回来说本书，前端的知识点是非常庞杂的。但知识只有串起来，形成知识树才是你自己的。现在市面上的书，还是依照老旧的方法教人，一开始总是历史回顾，然后是各种数据类型介绍，然后是语法（条件分支、循环分支）什么的，最后再来几个“真实案例”。这对于 90%半路出家的前端来说，未免太闷了。而且前端不单单是 JavaScript，JavaScript 只是水泥，或者说储如化学分子这东西，而我们工作是为了构建一整座大厦！只有肉眼看到的物理级别的东西才是主角。它

们就是本书的重点，DOM 与 BOM。JavaScript 通过特性侦测或传参等，进入不同的分支，来解决前端兼容性问题。本书介绍了大量这样的黑魔法，如何知道当前浏览器是支持这个事件呢？为何在这里要劫持 this 呢？怎么样让选择器引擎跑得更快。于是这一个个疑问，便化解成本书所介绍的知识点，什么 AMD 加载器、选择器引擎、批量生成一堆元素节点……

所有前端框架面对的问题都是一样，不同的是解决手段的高下程度。于是 Prototype 死了，jQuery 火了。angular 爆发了，jQuery 沉寂了！本书的章节就是按照编写一个多文件框架的顺序来写。最开始肯定是种子模块，定义框架的名字与版本号，与一些最核心的方法，还有加载器。然后通过加载器，添加一些常用的工具模块，对 JavaScript 语言进行扩张与修复。之后是数据缓存什么的，再之后是主菜，各种 DOM 问题，节点啊，样式啊，事件啊，动画啊……最后是 MVVM，当前最强大的前端解决方案。通过引入双向绑定与分层架构，完全脱离 DOM 进行前端开发。

你或者有过激情，你或者有过梦想，但当你的 KIP 考核点是 PM 那些荒唐的改来改去的功能点，多炽热的火焰也会被浇灭。因此，你必须要搞出一点东西出来，努力爬上去。是废命于加班，天天写业务，还是专注于底层框架的研发，为某个难题而苦恼，完全在于你一念之差。“是金子总会发光的”，或“是石头到哪都不会发光的”，也完全在于你一念之差。本书将为你提供了一个可能性及一大堆技能点，打开了一个美丽的新世界，提供了一个 X 年不遇的机会，准备了一个迅速上升的渠道。

可能有些人会嫌它贵，也有些人怕自己看不明白这么“高大上”的东西。我说一个故事吧。

一个特别喜爱昆虫的人做了这样一个实验 :他将跳蚤放进敞口的瓶子里 ,它立刻便跳了出来。当把瓶子盖上时 , 跳蚤还是会竭力跳出瓶子 , 它不停地撞击着瓶盖的内侧。一个多小时后 , 他还在那样跳着。差不多三个小时后 , 它依然在跳 , 只是它不再撞着瓶盖了 , 此时它跳的高度离瓶盖大约 1 厘米左右 , 而且每一次都是如此。这时 , 瓶盖被拿掉了 , 但是跳蚤并没有跳出瓶口 , 它依然保持着有瓶盖时的高度 , 再也跳不出瓶口。

同理，如果你总嫌这个贵那个贵，你又不努力改变现状，过了一段时间，你就会习惯了，就会安于贫困了。就像瓶子的跳蚤那样安于天命，永远困死在瓶子里。

怕自己理解不了这东西，这虽然是一个理智的考量，但只要是人就会遇到瓶颈，但问题是如何突破瓶颈。瓶盖又不是总是盖上的，有机遇你得抓住！

我也曾毕业找不到工作，潦倒到当了一年保安。但我相信“是金子总会发光的”，我终会一鸣惊人，我现在只是一只受伤的野狼，我不会被命运所屈服驯化。因此，跟我咆哮吧……

无名的生命之花 已惨遭摧残践踏
一度坠地的飞鸟 正焦急以待风起
一味埋头祈祷 却不会有任何改变
若想有所改变 就请起而奋战吧
踏过尸体前行的我们
嘲笑这进击意志的猪猡啊
家畜般的安宁 那虚伪的繁荣
请赐予誓死之饿狼以自由！
……

——进击的巨人 OP《红莲の弓矢》

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 getElement ById、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。一般创造性的活动，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的 UI 库。而企业开发中，UI 库就像流水线那样重要。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。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

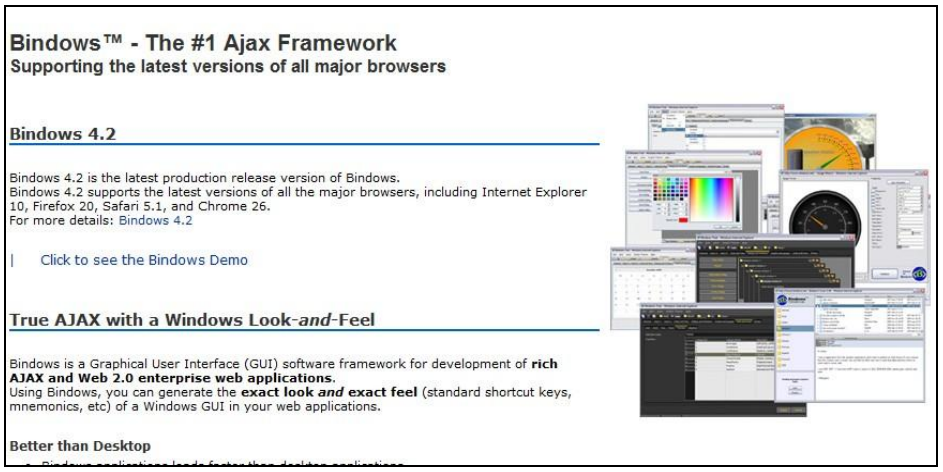
快多了，设计师在网页制造的地位（UED）也不比昔日，因此，集成程度更高的 MVC、MVVM 框架也相继面世。

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 发展历程

第 1 时期，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。



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 3 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。

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

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 Firefox 3 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（JavaScript 程序员）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]。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 时期，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。

第 1 种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。

第 2 种出现的是以类工厂为导向的框架，如著名的 Prototype，还有 mootools、Base2、Ten。

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

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，用户只需专注于业务代码。

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 中的源代码，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 4 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，源程序和答疑网址：https://github.com/RubyLouvre/jsbook/issues。编辑联系邮箱：zhangtao@ptpress.com.cn。

目　　录

第 3 章　语言模块

1995 年，Brendan Eich 读完了所有在程序语言设计中曾经出现过的错误，自己又发明了一些更多的错误，然后用它们创造出了 LiveScript。之后，为了紧跟 Java 语言的时髦潮流，它被重新命名为 JavaScript。再然后，为了追随一种皮肤病的时髦名字，这语言又被命名为 ECMAScript。

上面一段话出自博文《编程语言伪简史》，JavaScript 受到最辛辣的嘲讽，可见，它在当时是多少不受欢迎的。抛开偏见，JavaScript 的确有许多不足之外，由于互联网的传播性及浏览器大战，JavaScript 之父失去对此门语言的掌控权，即便他想修复这些 Bug 或推出某些新特，也要所有浏览器大厂都点头才行。IE6 的市场独占性，打破了他的奢望。这个待到 Chrome 诞生，才有所改善。

但在 IE6 时期，浏览器提供的原生 API 的数量是极其贫乏的，因此各个框架都创造了许多方法以弥补这缺陷。视框架作者原来的语言背景不同，这些方法也是林林总总的。其中最杰出的代表是王者 Prototype.js，把 ruby 语言的那一套方式或范式搬过来，从底层促进了 JavaScript 的发展。ecma262v6 添加那一堆字符串，数组方法，差不多就是改个名字而已。

即便是浏览器的 API 也不能尽信，尤其是 IE6、IE7、IE8？到处是 BUG，因此这也列入框架的工作范围。

本章主要是围绕着 mass Framework 的 lang 与 lang_fix 模块展开，可以到这里下载。

https://github.com/RubyLouvre/mass-Framework/blob/1.4/lang.js

https://github.com/RubyLouvre/mass-Framework/blob/1.4/lang_fix.js

3.1　字符串的扩展与修复

我发现脚本语言都对字符串特别关注，有关它的方法特别多。我把这些方法分为三大类。

第一类，与标签无关的实现： charAt、charCodeAt、concat、indexOf、lastIndexOf、localeCompare、match、replace,search、slice、split、substr、substring、toLocaleLowerCase、toLocaleUpperCase、toLowerCase、toUpperCase 及从 Object 继承回来的方法，如 toString、valueOf。

第二类，与标签有关的实现，都是对原字符串添加一对标签：anchor、big、blink、bold、fixed、fontcolor、italics、link、small、strike、sub、sup。

第三类是后来添加或未标准化的浏览器方法：trim、quote、toSource、trimLeft、trimRight。其中 trim 已经标准化，后四个是 Firefox 的私有实现。

我们再看 ecma262v6（2012.6.15）打算要添加的方法：repeat、startsWith、endsWidth、contains。

再看伟大的 Prototype.js 添加的扩展：gsub、sub、scan、truncate、strip、stripTags、stripScripts、extractScripts、evalScripts、escapeHTML、unescapeHTML、parseQuery、toArray、succ、times、camelize、capitalize、underscore、dasherize、inspect、unfilterJSON、isJSON、evalJSON、include、startsWith、endsWith、empty、blank、interpolate。

其中 gsub、sub、scan 与正则相关，直接取自 ruby 的命名。

truncate 是字符串截取，非常有用的方法，许多框架都有它的“微创新”。

strip 即 trim，已标准化。

stripTags 去掉字符串中的标签对，非常有用。

stripScripts 作为 stripTags 的补充，因为单单把 script 标签去掉，里面不该显示出来的 script.text 就暴露出来了。

extractScripts 与 evalScripts 是抽取与执行字符串中的脚本，IE 的 innerHTML 在某种情况下可以这样做，但其他浏览器不行，框架有责任屏蔽此差异性。

escapeHTML 与 unescapeHTML 是对用户的输入输出操作进行转义，非常有用。

parseQuery 基本上用于对 URL 的 search 部分进行操作，转换成对象，非常有用。

toArray 原本也是 ecma262v6 打算要添加的方法，用于转换成数组，不过这个用户也易实现，因此被抛弃了。

succ 是用于 ObjectRange 内部使用的。

times 即 ecma262v6 的 repeat 方法。

Camelize、capitalize、underscore、dasherize 这四个用于转换命名风格，非常有用。

inspect 就是在两端加双引号，用于构建 JSON，相当于 Firefox 的私有实现 quote。

UnfilterJSON、isJSON、evalJSON 与 JSON 相关。

include 就是 contains，与 startsWith、endsWith 成为 ecma262v6 的标准方法。

empty、blank 是对空白进行判定，很简单的方法。

interpolate 用于模板，其他框架通常称之为 format 或 substitute。

Prototype.js 这些有用的扩展会被其他框架抄去，我们查看哪些经常被抄，就知道哪些方法最有价值了。

Rirght.js 的字符串扩展：include、blank、camelize、capitalize、dasherize、empty、endsWith、evalScripts、extractScripts、includes、on、startsWith、stripScripts、stripTags、toFloat、toInt、trim、underscored。

Mootools 的字符串扩展（只取原型扩展）：test、contains、trim、clean、camelCase、hyphenate、capitalize、escapeRegExp、toInt、toFloat、hexToRgb、rgbToHex、substitute、stripScripts。

dojo 的字符串扩展：rep、pad、substitute、trim。rep 就是 repeat 方法。

EXT 的字符串扩展：capitalize、ellipsis、escape、escapeRegex、format、htmlDecode、htmlEncode、leftPad、parseQueryString、trim、urlAppend。

qooxdoo 的字符串扩展：format、hyphenate、pad、repeat、startsWidth、stripScripts、stripTags、toArray、trim、trimLeft、trimRight。

Tangram 的字符串扩展：decodeHTML、encodeHTML、escapeReg、filterFormat、format、formatColor、stripTags、toCamelCase、toHalfWidth、trim、wbr。

通过以上竞争对手分析，我在 mass Framework 为字符串添加如下扩展，各位写框架的朋友可以视自己的情况进行增减：contains、startsWith、endsWith、repeat、camelize、underscored、capitalize、stripTags、stripScripts、escapeHTML、unescapeHTML、escapeRegExp、truncate，wbr、pad。其中前四个 ecma262v6 的标准方法，接着九个发端于 Prototype.js 广受欢迎的工具方法，wbr 是来自 Tangram，用于软换行，这出于汉语排版的需要。pad 也是一个很常用的操作，被收纳。

下面是各种具体实现。

contains 方法：判定一个字符串是否包含另一个字符串。常规思维，使用正则，但每次都要用 new RegExp 来构造，性能太差，转而使用原生字符串方法，如，indexOf、lastIndexOf、 search。

```
function contains(target, it) {
    return target.indexOf(it) != -1; //indexOf 改成 search , lastIndexOf 也行得通
}
```

在 mootools 的版本中，我看到它支持更多参数，估计目的是判定一个元素的 className 是

否包含某个特定的 class。众所周知，元素可以添加多个 class，中间以空格隔开，使用 mootools 的 contains 就很方便检测包含关系了。

```
function contains(target, str, separator) {
   return separator ?
          (separator + target + separator).indexOf(separator + str + separator) > -1 :
          target.indexOf(str) > -1;
}
```

注，本章的所有工具函数都是以静态方法，将它们变成原型方法，我在结束这章时给一个函数变换方法。

startsWith 方法：判定目标字符串是否位于原字符串的开始之处，可以说是 contains 方法的变种。

```
//最后一参数是忽略大小写
function startsWith(target, str, ignorecase) {
   var start_str = target.substr(0, str.length);
   return ignorecase ? start_str.toLowerCase() === str.toLowerCase() :
          start_str === str;
}
```

endsWith 方法：与 startsWith 相反。

```
//最后一参数是忽略大小写
function endsWith(target, str, ignorecase) {
   var end_str = target.substring(target.length - str.length);
   return ignorecase ? end_str.toLowerCase() === str.toLowerCase() :
          end_str === str;
}
```

repeat 方法：将一个字符串重复自身 N 次，如 repeat（"ruby", 2）得到 rubyruby。

版本 1：利用空数组的 join 方法。

```
function repeat(target, n) {
   return (new Array(n + 1)).join(target);
}
```

版本 2：版本 1 的改良版，创建一个对象，拥有 length 属性，然后利用 call 方法去调用数组原型的 join 方法，省去创建数组这一步，性能大为提高。重复次数越多，两者对比越明显。另，之所以要创建一个带 length 属性的对象，是因为要调用数组的原型方法，需要指定 call 的第一个参数为类数组对象。而类数组对象的必要条件是其 length 属性的值为非负整数。

```
function repeat(target, n) {
   return Array.prototype.join.call({
      length: n + 1
   }, target);
}
```

版本 3：版本 2 的改良版，利用闭包将类数组对象与数组原型的 join 方法缓存起来，省得每次都重复创建与寻找方法。

```
var repeat = (function() {
   var join = Array.prototype.join, obj = {};
   return function(target, n) {
      obj.length = n + 1;
      return join.call(obj, target);
   }
})();
```

版本 4：从算法上着手，使用二分法，比如我们将 ruby 重复 5 次，其实我们在第二次已得 rubyruby，那么 3 次直接用 rubyruby 进行操作，而不是用 ruby。

```
function repeat(target, n) {
   var s = target, total = [];
   while (n > 0) {
      if (n % 2 == 1)
         total[total.length] = s;//如果是奇数
      if (n == 1)
         break;
      s += s;
      n = n >> 1;//相当于将 n 除以 2 取其商,或说开 2 二次方
   }
   return total.join('');
}
```

版本 5：版本 4 的变种，免去创建数组与使用 jion 方法。它的悲剧之处在于它在循环中创建的字符串比要求的还长，需要回减一下。

```
function repeat(target, n) {
   var s = target, c = s.length * n
   do {
      s += s;
   } while (n = n >> 1);
   s = s.substring(0, c);
   return s;
}
```

版本 6：版本 4 的改良版。

```
function repeat(target, n) {
   var s = target, total = "";
   while (n > 0) {
      if (n % 2 == 1)
         total += s;
      if (n == 1)
         break;
      s += s;
      n = n >> 1;
   }
   return total;
}
```

版本 7：与版本 6 相近，不过递归在浏览器下好像都做了优化（包括 IE6），与其他版本相比，属于上乘方案之一。

```
function repeat(target, n) {
   if (n == 1) {
      return target;
   }
   var s = repeat(target, Math.floor(n / 2));
   s += s;
   if (n % 2) {
      s += target;
   }
   return s;
}
```

版本 8：可以说是一个反例，很慢，不过实际上它还是可行的，因为没有人将 n 设成上百成千。

```
function repeat(target, n) {
   return (n <= 0) ? "" : target.concat(repeat(target, --n));
}
```

经测试，版本 6 在各浏览器的得分是最高的。

byteLen 方法：取得一个字符串所有字节的长度。这是一个后端过来的方法，如果将一个英文字符插入数据库 char、varchar、text 类型的字段时占用一个字节，而一个中文字符插入时占

用两个字节，为了避免插入溢出，就需要事先判断字符串的字节长度。在前端，如果我们要用户填空的文本，需要字节上的长短限制，比如发短信，也要用到此方法。随着浏览器普及对二进制的操作，这方法也越来越常用。

版本 1：假设字符串每个字符的 Unicode 编码均小于等于 255，byteLength 为字符串长度；再遍历字符串，遇到 Unicode 编码大于 255 时，为 byteLength 补加 1。

```
function byteLen(target) {
    var byteLength = target.length, i = 0;
    for (; i < target.length; i++) {
        if (target.charCodeAt(i) > 255) {
            byteLength++;
        }
    }
    return byteLength;
}
```

版本 2：使用正则，并支持制定汉字的存储字节数。比如 mysql 存储汉字时，是用 3 个字节数的。

```
function byteLen(target, fix) {
    fix = fix ? fix : 2;
    var str = new Array(fix + 1).join("-")
    return target.replace(/[^\x00-\xff]/g, str).length;
}
```

版本 3：来自腾讯的解决方案。腾讯通过多子域名+postMessage+manifest 离线 proxy 页面的方式扩大 localStorage 的存储空间，在这过程中需要知道用户已经存了多少东西，因此，我们就必须编写一个严谨的 byteLen 方法。

```
/**
 *
http://www.alloyteam.com/2013/12/js-calculate-the-number-of-bytes-occupied-by-a-str
ing/
 * 计算字符串所占的内存字节数，默认使用 UTF-8 的编码方式计算，也可制定为 UTF-16
 * UTF-8 是一种可变长度的 Unicode 编码格式，使用 1 至 4 个字节为每个字符编码
 *
 * 000000 - 00007F(128 个代码)      0zzzzzzz(00-7F)                                一个字节
 * 000080 - 0007FF(1920 个代码)     110yyyyy(C0-DF) 10zzzzzz(80-BF)                  两个字节
 * 000800 - 00D7FF
   00E000 - 00FFFF(61440 个代码)    1110xxxx(E0-EF) 10yyyyyy 10zzzzzz                3 个字节
 * 010000 - 10FFFF(1048576 个代码)  11110www(F0-F7) 10xxxxxx 10yyyyyy 10zzzzzz      4 个字节
 *
 * 注：Unicode 在范围 D800-DFFF 中不存在任何字符
 *                                      {@link                                      <a
onclick="javascript:pageTracker._trackPageview('/outgoing/zh.wikipedia.org/wiki/UTF
-8');"
 * href="http://zh.wikipedia.org/wiki/UTF-8">http://zh.wikipedia.org/wiki/UTF-8</a>}
 *
 * UTF-16 大部分使用两个字节编码，编码超出 65535 的使用 4 个字节
 * 000000 - 00FFFF  两个字节
 * 010000 - 10FFFF  4 个字节
 *
 *                                      {@link                                      <a
onclick="javascript:pageTracker._trackPageview('/outgoing/zh.wikipedia.org/wiki/UTF
-16');"
 *
href="http://zh.wikipedia.org/wiki/UTF-16">http://zh.wikipedia.org/wiki/UTF-16</a>}
 * @param  {String} str
 * @param  {String} charset utf-8, utf-16
 * @return {Number}
 */
```

```
function byteLen(str, charset){
   var total = 0,
       charCode,
       i,
       len;
   charset = charset ? charset.toLowerCase() : '';
   if(charset === 'utf-16' || charset === 'utf16'){
       for(i = 0, len = str.length; i < len; i++){
           charCode = str.charCodeAt(i);
           if(charCode <= 0xffff){
               total += 2;
           }else{
               total += 4;
           }
       }
   }else{
       for(i = 0, len = str.length; i < len; i++){
           charCode = str.charCodeAt(i);
           if(charCode <= 0x007f) {
               total += 1;
           }else if(charCode <= 0x07ff){
               total += 2;
           }else if(charCode <= 0xffff){
               total += 3;
           }else{
               total += 4;
           }
       }
   }
   return total;
}
```

truncate 方法：用于对字符串进行截断处理，当超过限定长度，默认添加三个点号或其他什么的。

```
function truncate(target, length, truncation) {
   length = length || 30;
   truncation = truncation === void(0) ? '...' : truncation;
   return target.length > length ?
          target.slice(0, length - truncation.length) + truncation : String(target);
}
```

camelize 方法：转换为驼峰风格。

```
function camelize(target) {
   if (target.indexOf('-') < 0 && target.indexOf('_') < 0) {
       return target;//提前判断，提高 getStyle 等的效率
   }
   return target.replace(/[-_][^-_]/g, function(match) {
       return match.charAt(1).toUpperCase();
   });
}
```

underscored 方法：转换为下划线风格。

```
function underscored(target) {
   return target.replace(/([a-z\d])([A-Z])/g, '$1_$2').
          replace(/\-/g, '_').toLowerCase();
}
```

dasherize 方法：转换为连字符风格，亦即 CSS 变量的风格。

```
function dasherize(target) {
   return underscored(target).replace(/_/g, '-');
}
```

capitalize 方法：首字母大写。

```
function capitalize(target) {
   return target.charAt(0).toUpperCase() + target.substring(1).toLowerCase();
}
```

stripTags 方法：移除字符串中的 html 标签，但这方法有缺陷，如里面有 script 标签，会把这些不该显示出来的脚本也显示出来。在 Prototype.js 中，它与 strip、stripScripts 是一组方法。

```
function stripTags(target) {
   return String(target || "").replace(/<[^>]+>/g, '');
}
```

stripScripts 方法：移除字符串中所有的 script 标签。弥补 stripTags 方法的缺陷。此方法应在 stripTags 之前调用。

```
function stripScripts(target) {
   return String(target || "").replace(/<script[^>]*>([\S\s]*?)<\/script>/img, '')
}
```

escapeHTML 方法：将字符串经过 html 转义得到适合在页面中显示的内容，如将<替换为<。

```
function escapeHTML(target) {
   return target.replace(/&/g, '&amp;')
          .replace(/</g, '&lt;')
          .replace(/>/g, '&gt;')
          .replace(/"/g, "&quot;")
          .replace(/'/g, "&#39;");
}
```

unescapeHTML：将字符串中的 html 实体字符还原为对应字符。

```
function escapeHTML(target) {
   return target.replace(/&/g, '&amp;')
          .replace(/</g, '&lt;')
          .replace(/>/g, '&gt;')
          .replace(/"/g, "&quot;")
          .replace(/'/g, "&#39;");//IE 下不支持&apos;(单引号)转义
}
```

escapeRegExp 方法：将字符串安全格式化为正则表达式的源码。

```
function escapeRegExp(target) {
   return target.replace(/([-.*+?^${}()|[\]\/\\])/g, '\\$1');
}
```

pad 方法：与 trim 相反，pad 可以为字符串的某一端添加字符串。常见的用法如日历在月份前补零，因此也被称之为 fillZero。我在博客上收集许多版本的实现，在这里转换静态方法一并放出。

版本 1：数组法，创建数组来放置填充物，然后再在右边起截取。

```
function pad(target, n) {
   var zero = new Array(n).join('0');
   var str = zero + target;
   var result = str.substr(-n);
   return result;
}
```

版本 2：版本 1 的变种。

```
function pad(target, n) {
   return Array((n + 1) - target.toString().split('').length).join('0') + target;
}
```

版本 3：二进制法。前半部分是创建一个含有 *n* 个零的大数，如（1<<5）.toString（2），生成 100000，（1<<8）.toString（2）生成 100000000，然后再截短。

```
function pad(target, n) {
   return (Math.pow(10, n) + "" + target).slice(-n);
}
```

版本 4：Math.pow 法，思路同版本 3。

```
function pad(target, n) {
   return ((1 << n).toString(2) + target).slice(-n);
}
```

版本 5：toFixed 法，思路与版本 3 差不多，创建一个拥有 *n* 个零的小数，然后再截短。

```
function pad(target, n) {
   return (0..toFixed(n) + target).slice(-n);
}
```

版本 6：创建一个超大数，在常规情况下是截不完的。

```
function pad(target, n) {
   return (1e20 + "" + target).slice(-n);
}
```

版本 7：质朴长存法，就是先求得长度，然后一个个地往左边补零，加到长度为 n 为止。

```
function pad(target, n) {
   var len = target.toString().length;
   while (len < n) {
       target = "0" + target;
       len++;
   }
   return target;
}
```

版本 8：也就是现在 mass Framework 使用的版本，支持更多参数，允许从左或从右填充，以及使用什么内容进行填充。

```
function pad(target, n, filling, right, radix) {
   var num = target.toString(radix || 10);
   filling = filling || "0";
   while (num.length < n) {
       if (!right) {
           num = filling + num;
       } else {
           num += filling;
       }
   }
   return num;
}
```

wbr 方法：为目标字符串添加 wbr 软换行。不过需要注意的是，它并不是在每个字符之后都插入<wbr>字样，而是相当于在组成文本节点的部分中的每个字符后插入<wbr>字样。如 aabbcc，返回 a<wbr>a<wbr>b<wbr>b<wbr>c<wbr>c<wbr>。另外，在 Opera 下，浏览器默认 css 不会为 wbr 加上样式，导致没有换行效果，可以在 css 中加上 wbr:after { content: "\00200B" } 解决此问题。

```
function wbr(target) {
   return String(target)
          .replace(/(?:<[^>]+>)|(?:&#?[0-9a-z]{2,6};)|(.{1})/gi, '$&<wbr>')
          .replace(/><wbr>/g, '>');
}
```

format 方法：在 C 语言中，有一个叫 printf 的方法，我们可以在后面添加不同的类型的参数嵌入到将要输出的字符串中。这是非常有用的方法，因为在 JavaScript 涉及大量这样的字符串拼接工作。如果涉及逻辑，我们可以用模板，如果轻量点，我们可以用这个方法。它在不同框架名

字是不同的，Prototype.js 叫 interpolate, Base2 叫 format，mootools 叫 substitute。

```
function format(str, object) {
   var array = Array.prototype.slice.call(arguments, 1);
   return str.replace(/\\?\#{([^{}]+)\}/gm, function(match, name) {
      if (match.charAt(0) == '\\')
         return match.slice(1);
      var index = Number(name)
      if (index >= 0)
         return array[index];
      if (object && object[name] !== void 0)
         return  object[name];
      return  '';
   });
}
```

它支持两种传参方法，如果字符串的占位符为 0、1、2 这样的非零整数形式，要求传入两个或两个以上的参数，否则就传入一个对象，键名为占位符。

```
var a = format("Result is #{0},#{1}", 22, 33);
alert(a);//"Result is 22,33"
var b = format("#{name} is a #{sex}", {
   name: "Jhon",
   sex: "man"
});
alert(b);//"Jhon is a man"
```

quote 方法：在字符串两端添加双引号，然后内部需要转义的地方都要转义，用于接装 JSON 的键名或模析系统中。

版本一：

```
//http://code.google.com/p/jquery-json/
var escapeable = /["\\\x00-\x1f\x7f-\x9f]/g,
       meta = {
   '\b': '\\b',
   '\t': '\\t',
   '\n': '\\n',
   '\f': '\\f',
   '\r': '\\r',
   '"': '\\"',
   '\\': '\\\\'
};
function quote(target) {
   if (target.match(escapeable)) {
      return '"' + target.replace(escapeable, function(a) {
         var c = meta[a];
         if (typeof c === 'string') {
            return c;
         }
         return '\\u' + ('0000' + c.charCodeAt(0).toString(16)).slice(-4)
      }) + '"';
   }
   return '"' + target + '"';
}
```

版本二，来自百度的 etpl 模板库：

```
//https://github.com/ecomfe/etpl/blob/2.1.0/src/main.js#L207
function stringLiteralize(source) {
   return '"'
          + source
          .replace(/\x5C/g, '\\\\')
          .replace(/"/g, '\\"')
          .replace(/\x0A/g, '\\n')
          .replace(/\x09/g, '\\t')
          .replace(/\x0D/g, '\\r')
```

```
            + '"';
}
```

当然，如果浏览器已经支持原生 JSON，我们直接用 JSON.stringify 就行了，另，FF 在 JSON 发明之前，就支持 String.prototype.quote 与 String.quote 方法，我们在使用 quote 之前判定浏览器是否内置这些方法。

字符串好像没有大的浏览器兼容问题，有的话是 IE6、IE7 不支持用数组中括号取它的每一个字符，需要用 charAt 来取；IE6、IE7、IE8 不支持垂直分表符,因此有如下 hack。

```
var isIE678= !+"\v1" ;
```

好了，我们来修复旧版本 IE 中的 trim 函数。这是一个很常用的操作，通常用于表单验证，我们需要把两端的空白去掉，清除“杂质”后，或转换数值进行范围验证，或进行空白验证，或字数验证……由于太常用，相应的实现也非常多。我们可以一起看看，顺便学习一下正则。

版本 1：看起来不怎么样，动用了两次正则替换，实际速度非常惊人，主要得益于浏览器的内部优化。base2 类库使用这种实现。在 Chrome 刚出来的年代，这实现是异常快的，但 Chrome 对字符串方法的疯狂优化，引起了其他浏览器的跟风。于是正则的实现再也比不了字符串方法了。一个著名的例子字符串拼接，直接相加比用 Array 做成的 StringBuffer 还快，而 StringBuffer 技术在早些年备受推崇！

```
function trim(str) {
   return str.replace(/^\s\s*/, '').replace(/\s\s*$/, '');
}
```

版本 2：和版本 1 很相似，但稍慢一点，主要原因是它最先是假设至少存在一个空白符。Prototype.js 使用这种实现，不过其名字为 strip，因为 Prototype 的方法都是力求与 Ruby 同名。

```
function trim(str) {
   return str.replace(/^\s+/, '').replace(/\s+$/, '');
}
```

版本 3：截取方式取得空白部分（当然允许中间存在空白符），总共调用了四个原生方法。设计得非常巧妙，substring 以两个数字作为参数。Math.max 以两个数字作参数，search 则返回一个数字。速度比上面两个慢一点，但基本比 10 之前的版本快！

```
function trim(str) {
   return str.substring(Math.max(str.search(/\S/), 0),
          str.search(/\S\s*$/) + 1);
}
```

版本 4：这个可以称得上版本 2 的简化版，就是利用候选操作符连接两个正则。但这样做就失去了浏览器优化的机会，比不上版本三。由于看来很优雅，许多类库都使用它，如 jQuery 与 mootools。

```
function trim (str) {
   return str.replace(/^\s+|\s+$/g, '');
}
```

版本 5：match 如果能匹配到东西会返回一个类数组对象，原字符匹配部分与分组将成为它的元素。为了防止字符串中间的空白符被排除，我们需要动用到非捕获性分组（?:exp）。由于数组可能为空，我们在后面还要做进一步的判定。好像浏览器在处理分组上比较无力，一个字慢。所以不要迷信正则，虽然它基本上是万能的。

```
function trim(str) {
   str = str.match(/\S+(?:\s+\S+)*/);
   return str ? str[0] : '';
```

```
}
```

版本 6：把符合要求的部分提供出来，放到一个空字符串中。不过效率很差，尤其是在 IE6 中。

```
function trim(str) {
   return str.replace(/^\s*(\S*(\s+\S+)*)\s*$/, '$1');
}
```

版本 7：与版本 6 很相似，但用了非捕获分组进行了优点，性能效之有一点点提升。

```
function trim(str) {
   return str.replace(/^\s*(\S*(?:\s+\S+)*)\s*$/, '$1');
}
```

版本 8：沿着上面两个的思路进行改进，动用了非捕获分组与字符集合，用?顶替了*，效果非常惊人。尤其在 IE6 中，可以用疯狂来形容这次性能的提升，直接秒杀 FF3。

```
function trim(str) {
   return str.replace(/^\s*((?:[\S\s]*\S)?)\s*$/, '$1');
}
```

版本 9：这次是用懒惰匹配顶替非捕获分组，在火狐中得到改善，IE 没有上次那么疯狂。

```
function trim(str) {
   return str.replace(/^\s*([\S\s]*?)\s*$/, '$1');
}
```

版本 10：我只想说，搞出这个的人已经不是用厉害来形容，已是专家级别了。它先是把可能的空白符全部列出来，在第一次遍历中砍掉前面的空白，第二次砍掉后面的空白。全过程只用了 indexOf 与 substring 这个专门为处理字符串而生的原生方法，没有使用到正则。速度快得惊人，估计直逼内部的二进制实现，并且在 IE 与火狐（其他浏览器当然也毫无疑问）都有良好的表现。速度都是零毫秒级别的。PHP.js 就收纳了这个方法。

```
function trim(str) {
   var whitespace = ' \n\r\t\f\x0b\xa0\u2000\u2001\u2002\u2003\n\
  \u2004\u2005\u2006\u2007\u2008\u2009\u200a\u200b\u2028\u2029\u3000';
    for (var i = 0; i < str.length; i++) {
       if (whitespace.indexOf(str.charAt(i)) === -1) {
          str = str.substring(i);
          break;
       }
    }
    for (i = str.length - 1; i >= 0; i--) {
       if (whitespace.indexOf(str.charAt(i)) === -1) {
          str = str.substring(0, i + 1);
          break;
       }
    }
    return whitespace.indexOf(str.charAt(0)) === -1 ? str : '';
}
```

版本 11：实现 10 的字数压缩版，前面部分的空白由正则替换负责砍掉，后面用原生方法处理，效果不逊于原版，但速度都非常逆天。

```
function trim(str) {
   str = str.replace(/^\s+/, '');
   for (var i = str.length - 1; i >= 0; i--) {
      if (/\S/.test(str.charAt(i))) {
         str = str.substring(0, i + 1);
         break;
      }
   }
```

```
    return str;
}
```

版本 12：版本 10 更好的改进版，注意说的不是性能速度，而是易记与使用方面。

```
function trim(str) {
    var str = str.replace(/^\s\s*/, ""),
           ws = /\s/,
           i = str.length;
    while (ws.test(str.charAt(--i)))
        return str.slice(0, i + 1);
}
```

版本 13：原作者@ialeafs 称它为 trimChunge，通过字符的 charCodeAt 值来判定是否为空白，速度也非常逆天，它仅次于版本 10，快于版本 11、12，不过此版本能处理的空白很有限。

```
function trim(str) {
    var m = str.length;
    for (var i = -1; str.charCodeAt(++i) <= 32; )
    for (var j = m - 1; j > i && str.charCodeAt(j) <= 32; j--)
    return str.slice(i, j + 1);
}
```

但这还没有完。如果你经常翻看 jQuery 的实现，你就会发现 jQuery1.4 之后的 trim 实现，多出了一个对 xA0 的特别处理。这是 Prototype.js 的核心成员 • kangax 的发现，IE 或早期的标准浏览器在字符串的处理上都有 BUG，把许多本属于空白的字符没有列为\s。根据屈屈的博文[4]，浏览器会把 WhiteSpace 和 LineTerminator 都列入空白字符。ecma262 v5 文档规定的 WhiteSpace 总共有这么多东西：

Unicode 编码	说　明
U+0020	SPACE，<SP>
U+00A0	NO-BREAK SPACE，<NBSP>
U+1680	OGHAM SPACE MARK，欧甘空格
U+180E	Mongolian Vowel Separator，蒙古文元音分隔符
U+2000	EN QUAD
U+2001	EM QUAD
U+2002	EN SPACE，En 空格。与 en 同宽（em 的一半）

续表

Unicode 编码	说　明
U+2003	EM SPACE，Em 空格。与 em 同宽
U+2004	THREE-PER-EM SPACE，Em 三分之一空格
U+2005	FOUR-PER-EM SPACE，Em 四分之一空格
U+2006	SIX-PER-EM SPACE，Em 六分之一空格
U+2007	FIGURE SPACE，数字空格。与单一数字同宽
U+2008	PUNCTUATION SPACE，标点空格。与同字体窄标点同宽
U+2009	THIN SPACE，窄空格。em 六分之一或五分之一宽
U+200A	HAIR SPACE，更窄空格。比窄空格更窄
U+200B	Zero Width Space，<ZWSP>，零宽空格
U+200C	Zero Width Non Joiner，<ZWNJ>，零宽不连字空格
U+200D	Zero Width Joiner，<ZWJ>，零宽连字空格
U+202F	NARROW NO-BREAK SPACE，窄式不换行空格

4 https://www.imququ.com/post/bom-and-javascript-trim.html

Unicode 编码	说　　明
U+205F	MEDIUM MATHEMATICAL SPACE，中数学空格 用于数学方程式
U+2060	Word Joiner，同 U+200B，但该处不换行 Unicode3.2 新增，代替 U+FEFF
U+3000	IDEOGRAPHIC SPACE，表意文字空格。即全角空格
U+FEFF	Byte Order Mark，<BOM>，字节次序标记字符 不换行功能于 Unicode3.2 起废止

LineTerminator 的家族如下：

Unicode 编码	说　　明
U+000D	<CR>回车符
U+000A	<LF>换行符
U+1680	<LS>行分隔符
U+180E	<PS>段落分隔符

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 4 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。

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 4 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 3 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）第三个、第四个、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，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[5]。

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。

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

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[6]，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() {
      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 23	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();
      };
```

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

```
        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。