《编写可维护的JavaScript》

图书基本信息

书名：《编写可维护的JavaScript》

13位ISBN编号：9787115310088

10位ISBN编号：7115310084

出版时间：2013-4

出版社：人民邮电出版社

作者：扎卡斯

页数：226

译者：李晶,郭凯,张散集

《编写可维护的JavaScript》

内容概要

《编写可维护的JavaScript》向开发人员阐述了如何在团队开发中编写具备高可维护性的JavaScript代码，书中详细说明了作为团队一分子，应该怎么写JavaScript。《编写可维护的JavaScript》内容涵盖了编码风格、编程技巧、自动化、测试等几方面，既包括具体风格和原则的介绍，也包括示例和技巧说明，最后还介绍了如何通过自动化的工具和方法来实现一致的编程风格。
《编写可维护的JavaScript》适合前端开发工程师、JavaScript程序员和学习JavaScript编程的读者阅读，也适合开发团队负责人、项目负责人阅读。

《编写可维护的JavaScript》

作者简介

Nicholas C. Zakas是一名前端开发顾问、作者和演讲家。在Yahoo！供职超过5年时间，在这期间他曾是Yahoo！首页首席前端工程师和YUI库代码贡献者。他著有《JavaScript高级程序设计》、《Ajax高级程序设计》和《高性能JavaScript》等书籍。Zakas倡导了很多最佳实践，包括渐进增强、可访问性、性能、扩展性和可维护性等。他的博客地址是：http://www.nczonline.net/。他的Twitter是：@slicknet。
李晶，花名拔赤，淘宝前端工程师，具有多年前端开发经验，在团队协作、组件开发、移动Web App等方面有深入研究，曾经参与淘宝首页、KISSY等项目开发。他翻译过《JavaScript Web富应用开发》、《JavaScript权威指南（第六版）》、《What is Node?》等书籍，热爱分享，喜欢折腾。微博http://weibo.com/jayli。
郭凯，花名流火，淘宝前端工程师，喜欢登山，也喜欢夜深人静时一人静静地写代码，折腾过许多小站均未能持久，有In、Juicer等开源项目。钟爱JavaScript，也喜欢Python，自嘲所学杂而不精。博客http://benben.cc。
张散集，花名一舟，淘宝前端工程师。他主要从事技术管理，负责淘宝网（北京）的新业务技术和前端团队，热爱前端新技术的推广与应用。翻译作品有《JavaScript Web富应用开发》和《JavaScript权威指南（第六版）》。

《编写可维护的JavaScript》

书籍目录

章节摘录

版权页： 插图： 方法封装了从DOM中删除一个元素的操作，屏蔽了开发者要访问该元素父节点的需求。 从JavaScript的可维护性而言，门面是非常合适的方式，自己可以完全控制这些接口。你可以允许访问任何底层对象的属性或方法，反之亦然，也就是有效地过滤对该对象的访问。你也可以对已有的方法进行改造，使其更加简单易用（上段示例代码就是一个案例）。底层的对象无论如何改变，只要修改门面，应用程序就能继续正常工作。 门面实现一个特定接口，让一个对象看上去像另一个对象，就称作一个适配器。门面和适配器唯一的不同是前者创建新接口，后者实现已存在的接口。 11.4关于Polyfill的注解 随着 ECMAScript 5和HTML 5的特性开始在各种浏览器中的实现，JavaScript polyfills（也称为 shims）变得流行起来了。一个polyfill是指一种功能的模拟，这些功能在新版本的浏览器中已经有完备定义并原生实现了。例如，ECMAScript 5为数组增加了forEach（）函数。该方法可以在ECMAScript 3中模拟，以便在老版本的浏览器中如同新版本一样使用。polyfills的关键是它们模拟的原生功能要以完全兼容的方式来实现。因此在有些浏览器中存在了这些功能，所以有必要检测不同情况下它们的处理是否符合标准的方式。为了达到目的，polyfills经常会给非自己拥有的对象新增一些方法。我不是polyfills的粉丝，不过对于别人使用它们，我表示理解。相比其他的对象修改而言，polyfills是有界限的，是相对安全的。因为原生实现中是存在这些方法并能工作的，有且仅当原生方法不存在时，polyfills才新增这些方法，并且它们和原生版本方法的行为是完全一致的。 polyfills的优点是，当只支持浏览器的原生功能时，它们非常容易删除。如果你选择使用某个polyfill，你自己做好严格审查。要保证它的功能和原生的版本尽可能的近似，多检查一下这种库代码有单元测试并严格验证了这些功能。polyfills的缺点是，它们可能没有精确地实现它们（原生浏览器环境）所缺失的功能，从而给你带来的麻烦比缺失的功能要多得多。

《编写可维护的JavaScript》

编辑推荐

《编写可维护的JavaScript》中强调：任何语言都需要强调编码风格的一致性。只要是团队开发，每个人都以相同方式编写代码就是至关重要的。这样大家才能方便地互相看懂和维护对方的代码。运用《编写可维护的JavaScript》中讲述的技巧和技术，可以使JavaScript团队编程从侠义的个人偏好的阴霾走出来，走向真正的高可维护性、高效能和高水准。它是一本构建编码风格手册，帮助开发团队从“游击队”走向“正规军”。

《编写可维护的JavaScript》

名人推荐

“ 本书是一本教你写出具有前瞻性的JavaScirpt代码的完全手册，在团队作战中特别有用。 ”——Ryan Grove，Yahoo! YUI工程师“ 作者将他十多年工作经验的精华部分浓缩至这本通俗易读的书中。我建议每个开发工程师和在校学生尽早阅读本书。不管你有多少经验，本书中的每一页内容都会让你变得更加优秀且倍受大家欢迎。 ”——Lea Verou，Web设计师兼Web开发人员“ 这是一本前端工程师的指南，指明了在编码过程需要注意的方方面面。提高可维护性是一个非常大的话题，而这本书，是一个非常不错的起点。 ”——王保平（玉伯），支付宝Web前端工程师

《编写可维护的JavaScript》

精彩短评

1、值得学习的一本书，然后你写的代码会比原先漂亮。
2、买这本书之前，关于翻译质量不佳之类的说法让我犹豫了一段时间。买了后我觉得确实有些地方出现词不达意和错别字的现象，但是这种错误其实并不多，完全在可以接受的范围内。而且译者根据国内的环境，以及国内外文化差异造成的一些进行了特别的解释跟说明，方便理解。再说说书的内容，因为本人一直在web开发，尤其是前端开发方面没有团队，都是通过自己学习、摸索进步的，所以很多设计、处理的方式并不规范，用的都是野路子（hack）。这对日后的调试、二次开发造成了很多不便，也对系统添加了很多隐患。本书非常系统的说明了各种开发中可能碰见的这些问题，并且非常到位的给出了解决思路以及方式方法。这一点对我来说意义非凡，使我提升巨大。所以我推荐所有有js开发基础经验并且希望进一步提高、规范自己开发的朋友入手。如果你是“单打独斗”，会对你日后开发有极大帮助，如果你在一个团队中，或者刚加入一个正规的团队，这本书也可以让你在很短的时间内融入其中
3、适合初级向中级进阶。书中前两部分都非常不错，值得每一个业余JavaScript程序员学习规范自己的开发。第三部分略有过时，读读也不错。目前用的比较多的应该是Grunt、gulp和webpack这些吧。不过类似的前端工具发展太快了。
4、自动化部分有点旧了，前两个部分还是有收获的。
5、前十三章对我帮助很大，之后的章节也不是说帮助不大，最主要是缺少有规范的项目的磨练，光看有点不知所云，mark，等以后回来再翻多几遍吧（唯一觉得不好的事，译者作为业内人士，翻译的质量真的是惨不忍睹，错字多，而且有些地方还翻译错误，毁了一本好书）
6、除了第二部分编程实践有一些点之前确实没有注意到，我认为第一部分编程风格是开发者最基本的素质并应长期坚持，第三部分自动化因为前端日新月异的工具(坑)层出不穷，其实已经挺old-fashioned了。但书中提到的整个最佳实践和开发流程应被一直遵守。
7、颇有收获
8、又是Zakas的书，规范书，看看就好
9、打基础好书
10、更多地注重JS的编写风格、命名规范等，收获了诸如nameSpace函数，和一些优秀的编程习惯，比如说从处理程序抽离应用逻辑，以及分离代码的配置数据等等。后面几章讲了已经过时的基于Java是Ant配置工具，现在依然从Grunt变换到Gulp，也提到了CI和Phantom，很有前瞻性呀。总的来说，还是不错，看得很快，适合当做日常读物的补充，没有精读。
11、很简洁的规范书，看完还是有些启发的，以前都没太多注意，规范起来！！！
12、不错的书，工作之余看看既不费脑也不费时，对规范化代码风格很有帮助！
13、好书，值得一读
14、不错的一本总结js编程规范的书，有一定实践经验后看还是挺有收获的，翻译不至于像某些人评论的那么差，反正我看下来是没有什么理解上的问题的，书里讲自动化的ant工具我直接跳过了，感觉用grunt或者gulp才是主流
15、适合初学者看
16、本身作者就是前端的业界大牛，尼古拉斯·扎卡斯。原来就写过一本非常不错的《JavaScript高级程序设计》。基本上搞前端重构的同学都会去读那本书。这本书也是从作者工作的经验和角度谈自己在团队中的一些JavaScript编码规范，非常具备指导意义。
17、终于看完了，断断续续读了一年多的书，早点读完就好了，书里基本上把最近才知道的或用过的思想和工具介绍了个遍，是学习JavaScript最佳实践的至少要完整读一遍的好书
18、很多实用的编程实践，书中很多内容以前也有些接触，感觉工作后领悟更深了
19、里面讲了一些关于如何编写一个规范的代码，例如，在编写注释的时候都需要空一行，在具有逻辑性代码的地方，如何缩进等等，但里面有一些章节是基本不会用到的
20、更多的编程理念和方法，需要一定的实践之后做提升用
21、能让你节省更多的代码阅读时间，多陪陪家人
22、好书，推荐，在火车上一夜看完的
23、东西不错，是正品，对学习帮助比较大

24、写出容易维护的的代码对于自己和整个项目来说都收益良多，前三分之二内容比较好，提醒到一些平时注意不到的细节，后面关于ant的介绍感觉用不到~
25、一本巨好的书。可惜最开始没看到，走了好多好多好多弯路。
26、干货不是很多，没有期望中的那么好，也可能是自己经验不够，没能体悟出里面的精髓
27、强烈建议leader读一读，越来越觉得大多数时候，代码最重要的是可读性。这本书里面的某些东西，还是很有价值的。
28、读过 但是基本没什么映像了
29、初一看原以为书比较薄, 价格略贵. 认真看了一遍之后收获颇丰.
30、厕所必备读物啊。。。
31、对于这种标准性差的弱类型语言，标准化规范化是多么的重要，这本书所讲的无疑是最好的编码典范！
32、很多内容过时了。另外我是支持不加分号的。
33、13章及以后是项目实践部分的内容，暂时还没接触过。13章之前的内容，也基本上接触过。
34、这本书前半部分的内容和《JavaScript : The Good Parts》有点重复，都是讲如何正确使用js的特性，而且最好是先读一下《JavaScript : The Good Parts》再读这本书，因为书中有些地方直接引用和提到了前者，可能因为两人都是雅虎的同事吧。后半部分可以很好的帮助自己从无到有建立起自己的js build环境。
35、编码习惯养成很重要，推荐
36、交互也存在与程序员之间，他们或许并未谋面。这类交互是用户交互的必然延续。于是作者开篇引言“ 程序是写给人读得，只是偶尔让计算机执行一下 ”
37、前半段挺实用的，后半段很多工具根本不认识，以后用到再翻翻了。
38、终于读完辣
39、翻译略有瑕疵。确实一本好书，读着读着就想起之前读的红皮书。
40、很好的一本书，让自己写代码的思路更加清晰了
41、中规中矩的书，之前在网络上看到的许多规范着书中都能找到。还行，并不是非常惊艳的一本书
42、通俗易懂，注意到很多细节，值得一看
43、前面两部分很棒，第三部分过时了。排版方面有不少瑕疵，感觉很不用心。不过从技能提高的角度来讲，还是强烈推荐阅读。
44、还没看.扫了一下目录 内容应该不错.应该可以学到东西.
45、可以了解一下，一些可能被忽略的问题
46、比较有用的只有第二部分. 第一部分和第三部分直接跳过了. 对于像我这样刚接触比较大的项目的新手, 对写代码还是挺有帮助的.
47、一直想找不错的js规范书籍，书里介绍的还是挺不错的
48、适合新手阅读，很多重要的点都点到了！但是并没有给出很好的解决方案。
49、绝对好书，编码规范很重要
50、讲的挺系统的，但是有些内容有点过时了，第二部分有亮点
51、还不错。养成良好的编码规范
52、野生的前端走向规范，后半部分介绍的工具大多都过时，了解即可
53、看了以后很有收获，发现了平时在编写代码是自己以为很好，实际上未必的错误编写习惯。
54、学习js应该都知道这书没的说，确实好，
55、很多规范看起来简单，真的写的时候要遵守起来还是很难的，得花时间捉摸一下
56、强推JavaScript高级程序设计一书，作者都是Zakas，这本书更像是高级程序设计的一个小的子集，主要内容是编码规范等，收获不大。
57、对提升代码质量很有帮助
58、手边工具书，推荐
59、好书 中文翻译得很好

60、前两部分还可以，第三部分太偏工程细节了。
另外勘误较多。。。

61、还不错，适合初学者规范自己的风格。有些专题没有深入讲解原因和解决方案。
62、讲得很详细，适合Team Leader看，以便推广
63、内容偏旧了点，有些说法已经过时了。
64、介绍了编码规范、自动化等，浏览下就可以了。
65、还没读完，基本上是zakas那本红宝书的子集，重点是javascript代码风格、规范以及最佳实践。
66、有助！

《编写可维护的JavaScript》

精彩书评

1、无论你是在校学生还是前端工程师都不可不阅之书，无论是初学者还是很有经验开发工程师它既能让其基础打得更牢，也能让你更上一层楼！一部构建编码风格手册的经典著作，一部帮助开发团队从“游击队”走向“正规军”的伟大之作，一部错过为之后悔的最新力作！
2、翻译的挺不错的，我是自己买纸质书的，虽然用了很少时间看完了，但是这本书对于刚刚入门js的童鞋还是不错的，对于规范代码和编写高质量的代码还是特别有建议。但是在生产实践中我还是喜欢用模块的的方式来管理代码，不过这本书里没有提及。是一个遗憾哟。
3、本书的第二部分是精华。倒没太多新东西，之前的《高性能JS》《JS高级程序设计》以及《JS模式》大部分都提到过，这里算是个总结。让人很不爽的一点是：原书中的“提示建议”，是以“爪印”图标和不同的样式跟正文区分开的；中文版中图标和样式都被干掉了，跟正文混在一起。奇怪的是：前言中“本书约定”里这些图标倒是还在。之前翻过英文版，因此没怎么仔细读。大致发现了一些翻译和排版的问题：p1 第一部分“程序是写给人读的，只是偶尔让计算机执行一下。”原文：“Programs are meant to be read by humans and only incidentally for computers to execute.”应该是：“程序是给人读的，顺带让计算机执行。”————————————————————————————p1同上 有些人可能来自某个“皮包公司”，身兼数职，在公司里什么都做；原文：“Some may come from one-man shops where they could do whatever they wanted;”“one-man shops”翻译成“皮包公司”并不合适也算不上幽默；“dowhatever they wanted”应该是指想怎么编码都可以，“身兼数职，什么都做”这个引申的过头了。————————————————————————————p2“毫无疑问，全球性的大公司都对外对内发布过编程风格文档。”原文：“It’s no wonder that large companies around the world have published style guidelines either internally or publicly.”“no wonder that”应该是“无怪乎，难怪”的意思。————————————————————————————p28开头那几行是注释，排版错误给弄成正文了。————————————————————————————p103最后一段第一句有个错字“恬当”，这个应该用是五笔打出来的。————————————————————————————————p135“我相当乐意花一整天的时间通过编程把一个任务实现自动化，除非这个任务手动只需要10秒钟就能完成”原文：“I . . . am rarely happier than when spending an entire day programming my computer to perform automatically a task that would otherwise take me a good ten seconds to do by hand.”这个应该是搞反了，原文意思是说——手动花10秒我都愿意去自动化，而不是——手动只要10秒我就不去自动化了。
4、麻雀虽小，却五脏俱全，这本书涵盖：编程风格（基本格式化、注释、语句和表达式、变量），编程实践（UI层松耦合、避免使用全局变量等等）第三部分自动化。第一部分比较基础，接触也比较多，经常培训新员工也会讲解代码规范的内容，代码规范虽然简单，但却十分重要。这部分由于熟悉也看得比较快。第二部分编程实践，属于很多知道，却不明白原理，看完这本书之后，仔细想想原理便对它“日久也弥新”了。比如UI松耦合为什么要抽离css，js，而且这里面也牵涉了一个概念“js模板”（将HTML从js中抽离），以前用过js模板，但没怎么发现它的好处，听了这本书的一些讲解觉得还是有道理的，道理就是方便调试，（ps这个道理看起来真站不住脚啊，哈哈哈）方法有三：从服务器加载、简单客户端模板（自己写个函数来实现页面标签的组装）、复杂客户端模板（使用js模板，如Handlebar等）。第一个问题：var name = "Nicholas"; alert(name.toUpperCase()); 尽管name是一个字符串，是原始类型不是对象，但你仍然可以使用诸如toUpperCase()之类的方法，即将字符串当做对象来对待。这种做法之所以行得通，是因为在这条语句背后js引擎创建了string类型的新实例，紧跟着就被销毁了，当再次需要时就会又创建另外一个对象。var name = new String("Nicholas"); name.author = true; alert(name.author);//truevar name = "Nicholas"; name.author = true; alert(name.author);//undefined在避免使用全局变量中，抛出概念：模块，其中又包含YUI模块和AMD模块。模块是一种通用的功能片段，它并没有创建新的全局变量或命名空间，相反，所有这些代码都存放于一个表示执行一个任务和发布一个接口的单函数中。可以用一个名称来表示这个模块，同样这个模块可以依赖其他模块。还有一种类型是零全局变量，jquery这些插件就是使用的这个立即执行的函数来实现零全局变量的。事件处理：业务逻辑和事件的处理分开比较：typeof 用来检测原始值：string number boolean undefined 最后一个原始值null 一般不用于检测。运行type null返回"object",这是一种抵消的判断null方法。如果需要检测null，则直接使用恒等运算符（===

）或者非恒等（!==）,检测引用值 Object、Array、Date、Error，typeof运算符显得力不从心，因为都返回“ object ”，使用instanceof 检测函数 typeof myFunc === “ function ” 检测数组Object.prototype.toString.call(value) === "[object Array]"检测属性if("count" in object)如果是只想检查实例对象的某个属性是否存在，则使用hasOwnProperty （）方法。将配置数据从代码中分离出来Props2Js工具读取Java属性文件，并给出三种格式的输出。抛出自定义错误的好处是可以区分出是浏览器本身的还是自己写的验证错误（感觉try catch错误自己用得很少啊 囧）浏览器嗅探避免浏览器推断，而应该用特性检测第三部分自动化文件合并加工、精简压缩、文档化、自动化测试等等最后决定咱们项目组使用CI持续集成哈哈哈，终于写完，看完这本书收益良多，对于如何改进js，提供了很多可以扩展的方向！

5、前几天同事刚推荐的这本“ 乌龟书 ”说是牛逼人物写的，大致翻了一遍，如中国足球一样粗糙的翻译是这本书中文版最大的败笔，翻译的太随便了，很多词不达意或不到位的地方，也不知道是译者英文水平的问题还是。。。

6、Nicholas的每一本书都对得起深入浅出四个字，前面的语法部分可说是一气呵成，例释并举，让人印象深刻。后面的工具部分，稍微有点拖沓和杂乱，不过也指明了解决问题的实用方案。从页数和价格上看，有点小贵，不过还是值了。附录A是干货，可保留电子版随时备查。

7、我是一个野生JavaScript程序员，我相信大多数JavaScript程序员都是野蛮生长，靠的是自身的好奇和勤勉。什么是野生JavaScript程序员？我是这样定义的：基本靠自学；通过不停的`console.log`和对比来写出凑合能运行的页面；页面上充斥着全局变量；命名风格随心所欲，受到自学的其他语言的影响；调试异常困难；只能写出耦合度低的小型页面；配置数据和代码耦合紧密，修改数据困难；随处定义变量；随意注释或者不注释；有大量从stackoverflow拷贝的代码……对于野生JavaScript程序员，也许可以很开心地做自己的小项目，但当需要多人合作的时候，就会让团队陷入噩梦。因为“ 程序是给人读的，只是偶尔让计算机执行一下 ”，所以勉强能运行的程序不是我们的目标，我们的目标是写出可维护的JavaScript。之前也读过《代码整洁之道》，是针对所有语言的通用手册，比较泛泛而谈，但本书针对性很强，没有散弹枪，每一颗子弹都狙击到目标上：糟糕的JavaScript代码。本书适合跟jslint配合使用，因为jslint能检查出很多人可能漏掉的细节：比如没有在`function`顶部就声明所有的变量，比如缩进，比如没有声明“ strict mode ”。下面说说印象比较深的几章：第6章 避免使用全局变量我之前知道JavaScript变量如果没有用`var`来声明的话（而且全局没有同名变量），会隐式地创建全局变量，所以我就知道随处使用`var`，本身告诉我，“ 所有的var语句都会提前到包含这段逻辑的函数的顶部执行 ”，所以应该在函数的顶部定义所有的局部变量。而且，由于JavaScript没有块作用域，所以在`for`循环和`if`中创建变量是没有意义的，都应该提前到函数的顶部去定义。那么如何避免全局变量呢？本章提出了几种解决方案：- 避免意外的全局变量——声明“ strict mode ”- 单全局变量方式——把所有的功能都封装到一个对象中- 零全局变量——创建一个即时执行的匿名函数第11章 不是你的对象不要动说实话，我犯过这样的错误，我在创建一个iOS项目的时候，其中的webview的JavaScript代码需要在xcode的控制台输出信息，我就重写了`console.log`这个函数，当时觉得这个小聪明很方便，后来却给我带来了无尽的麻烦，总之我后来又创建了一个新的对象ios来专门做这件事。第9章 将配置数据从代码中分离出来也是血泪教训，数据（比如颜色，宽度等）耦合到代码中，修改的时候异常困难，所以本章建议单独创建一个配置对象来修改数据，并且还提供了一些工具来把java配置文件转化成json数据。本书值得经常翻一翻，而且篇幅短小，是非常不错的床头技术书。

8、“ 作者将他十多年工作经验的精华部分浓缩至这本通俗易读的书中。我建议每个开发工程师和在校学生尽早阅读本书。不管你有多少经验，本书中的每一页内容都会让你变得更加优秀且倍受大家欢迎。 ”——Lea Verou，Web设计师兼Web开发人员

9、书分三部分。第一部分，制定了编码规范（附录一把这部分做了压缩）。第二部分，告诉你一些在团队中应该遵守的基本的编写代码的规则（或者说禁忌吧～）。第三部分，使用ant脚本来自动化生产，验证，压缩等环节（自己测试过发现作者写的代码有问题，譬如jshint参数应该放置到最后）。总结来说，是一本在上下班路上看的书，对于不太重视js或者根本没有js独立团队的公司，本书的作用不明显。

10、zakas推荐给大家的一个javascript开发规范文档。从代码风格到一些开发技巧，如何避免一些糟糕的情况，到目录管理，还有持续集成部署。一本全面的规范类书籍！当然zakas也有讲到一些深层的知识，还有ECMAScript5的一些新api，要搞html5开发的朋友更需要看这本书。你能从里面学到一些自己

忽略掉的东西，或者从里面学到一些新的技巧，虽然该书比较薄，但是最重要的是能学到真正有用的东西。假如团队里的开发人员都有看过这本书就好了……
……
……

11、只要是团队开发，每个人以相同的方式编写代码就是至关重要的。书中陈述了前端开发中编写javascript脚本一些常见的注意事项，对web开发人员和团队都有很大的参考价值。前端技术发展太快，各种新技术层出不穷，工具更新换代很快。我们需要理解程序开发本质原理，注意代码规范。最后引用本书开篇高纳德的一句话：“ 程序是写给人读的，只是偶尔让计算机之行以下 ”。
12、“ 这是一本前端工程师的指南，指明了在编码过程需要注意的方方面面。提高可维护性是一个非常大的话题，而这本书，是一个非常不错的起点。 ”——王保平（玉伯），支付宝Web前端工程师“ 本书是一本教你写出具有前瞻性的JavaScirpt代码的完全手册，在团队作战中特别有用。 ”——Ryan Grove，Yahoo! YUI工程师
13、写程序最重要的是思路清晰。HTML+CSS+Javascript跟其他面向对象开发的语言区别很大，灵活所以容易 凌乱本书第二部分讲了很多基本原则，实现解耦。看完本书，再看一些模块化开发的资料，就开始具备JS的内功。
14、如此之薄的一本书，第一版次整本读下来，如果很细心的话你会发现有n多个错误，责任编辑，难道你是边打飞机边工作的么？我只能说这尼玛太没有职业道德了！这实在太符合国产的现状了！关于翻译，有兴趣的可以看看《编码的奥秘》，呵呵俩字已经代表我的态度！挣钱——>不择手段地挣钱——>移民——>摆脱这个没道德的国度——>哇嘎嘎！

《编写可维护的JavaScript》

章节试读

1、《编写可维护的JavaScript》的笔记-《编写可维护的JavaScript》勘误表 2013年5月第二次印刷更正

序号	页号	位置	原文	应改为	提交者
1	6	第3行	2个空格表示一个缩进，2个空格表示一个缩进	4个空格表示一个缩进	豆瓣网友谷神出幕，冉海俊
2	28	第1-3行	前三行内容字体	（注释字体）	@土匪上树
3	39	倒数第2行	hasPwnProperty()	hasOwnProperty()	@土匪上树
4	50	第14行	我们推荐不要使用==和!==	我们推荐不要使用==和!=	@土匪上树，冉海俊，August
5	51	倒数第4行	console.log(count); //15	console.log(number); //15	豆瓣网友谷神出幕
6	60	第1行	接收	接受	豆瓣网友谷神出幕，冉海俊
7	66	倒数第5行	list.appendChild(div.firstChild)	mylist.appendChild(div.firstChild)	@土匪上树，lastnorman第68页16行代码做同样修改）
8	87	倒数第4行	MyAPplication.handleClick()	MyApplication.handleClick()	@土匪上树
9	98	第1行	在判断实例对象是否存在时…	在判断实例对象的属性是否存在时…	@土匪上树
10	109	第4行	生产环节	生产环境	@土匪上树
11	114	第18行	这种‘覆盖加可靠退化’的模式至少和原生方法一样不好	这种‘覆盖加可靠退化’的模至少和覆盖原生方法一样不好	@土匪上树
12	116	第5行	document.getElementsByClassname()	document.getElementsByClassName()	@土匪上树
13	131	第5行	开发人员开始将这种代码：	代码被修改成这样：	@土匪上树
14	133	第5行	其反命题…那么它不是Internet Explorer	那么它是Internet Explorer	@土匪上树
15	164	第6行	从可以下载YUI Compressor	从http://yuilibrary.com/download/yuicompressor/可以下载YUI Compressor	@土匪上树
16	194	第14行	话费	花费	@土匪上树
17	208	倒数第5行	一个双引号	一个冒号	@土匪上树

2、《编写可维护的JavaScript》的笔记-第20页

Google的javascript风格指南和Crokford的编程规范，推荐使用直接量代替Objec构造函数；而对于数组，推荐声明的时候使用中括号将数组元素括起来创建新的数组。

3、《编写可维护的JavaScript》的笔记-第65页

书中说：注释是和元素及文本一样的DOM节点，因此可以通过javascript将其提取出来。比如：

```
&lt;ul id="mylist"&gt;
 &lt;!--&lt;li id="item%s"&gt;&lt;a href="%s"&gt;%s&lt;/a&gt;&lt;/li&gt;--&gt;
 &lt;li&gt;&lt;a href="#"&gt;First item&lt;/a&gt;&lt;/li&gt;
 &lt;li&gt;&lt;a href="#"&gt;Second item&lt;/a&gt;&lt;/li&gt;
 &lt;li&gt;&lt;a href="#"&gt;Third item&lt;/a&gt;&lt;/li&gt;
&lt;/ul&gt;
```

js的提取：

```
var mylist = document.getElementById("mylist"),
    templateText = mylist.firstChild.nodeValue;
```

firstChild.nodeValue获取到的值为null，js有这个方法吗？表示怀疑。

之后我用jquery的children().first().html()获取到的是
First item
，并不是注释的<li>节点。

以上。

4、《编写可维护的JavaScript》的笔记-第40页

for-in不要用来遍历数组，而是用来遍历对象。

5、《编写可维护的JavaScript》的笔记-第54页

构建软件设计的方法有两种：一种是把软件做得很简单以至于明显找不到缺陷；另一种是把它做得很复杂以至于找不到明显的缺陷。

6、《编写可维护的JavaScript》的笔记-第42页

所有的var语句都提前到包含这段逻辑的函数的顶部执行。其实，这个在《javascript高级程序设计》和《javascript权威指南》中都有讲到。在声明一个函数的时候，我们最好把函数中用到的局部变量，都放在函数顶部声明，保证代码的清晰。

7、《编写可维护的JavaScript》的笔记-第45页

```
if(condition) {
 function a() {
 }
} else {
 function a() {
 }
}
```

在大多浏览器下，都会自动使用第二个声明。而Firefox会根据condition的计算结果选用合适的函数声明。还记得在项目中，有同学使用过这样的代码，得检查一下咯。

8、《编写可维护的JavaScript》的笔记-第39页

Crockford的编程规范要求所有的for-in循环都必须使用hasOwnProperty()

9、《编写可维护的JavaScript》的笔记-第六章，避免使用全局变量

如章名所说的，这篇章都在说为什么要避免使用全局变量，如何避免。

为什么要避免使用全局变量？
怕命名的冲突，而且不利于维护。

如何避免？
把js代码置于严格模式中，“ use strict ”;使用命名空间，也就是把对象方法都定义到同一个命名空间之下。
模块化开发知道，了解过seajs和requirejs的模块化编程，知道CMD和AMD还有commonJS规范。

6.4的零全局变量，不知道它的用法，书上说最常用是创建一个书签，但还是不理解。

10、《编写可维护的JavaScript》的笔记-第18页

理解null的最好方式就是将它当做对象的占位符。书中不推荐的做法，使用null来检测是否传入了某个参数；用null来检测一个未初始化的变量。这个我现在还经常这么干。呵呵！

11、《编写可维护的JavaScript》的笔记-第59页

javascript不应当直接操作样式，如果需要操作样式，最佳方法是操作css的className，以保持css的松耦合！受用！

12、《编写可维护的JavaScript》的笔记-第53页

译注: JavaScript 权威指南 (第六版) 3.6 “ 包装对象 ” 详细解释了包装对象和包装类型的细节，需要尤为注意的一点是，原始值本身不具有对象特性，比如 1.toString() 是报错的，必须这样做:
var a=1; a.toString()
实际上，
1..toString()
就可以了。关键问题不是原始值是否具有对象特性，是数字后面的一个点，会被 JavaScript 引擎当作小数点而被解析，而不是表示对象的属性

而多一个点，则由于小数点已经存在了，它表示的一定是访问对象的属性
所以自己 JavaScript 不好不要乱评论人家 (是这样吧)

13、《编写可维护的JavaScript》的笔记-第1页

https://www.evernote.com/shard/s350/sh/04872a79-c2bb-42e3-bf08-ce6729fc253c/56b1b281878ac675

14、《编写可维护的JavaScript》的笔记-第11页

计算机科学只存在两个难题：缓存失效和命名。----------Phil Karlton

15、《编写可维护的JavaScript》的笔记-第1页

看完了第一章。

看到提的一些规范，还是觉得是这样没错。
每次写js的时候，都会纠结于变量和函数的命名，书中的准则挺好的，可以区分普通函数和构造函数，虽然我之前一直没注意这个问题。

普通函数用小驼峰，前缀加上is、can、has、get动词。
构造函数用大驼峰，这样就知道是要new的了。

用C#的时候，换行我一直是很随意的。现在我知道了要在运算符之后换行。

16、《编写可维护的JavaScript》的笔记-第107页

平时在写代码的时候，不会去特地抛出错误，其实有必要的，尤其是你不希望某些事情发生的时候可以抛出错误。

17、《编写可维护的JavaScript》的笔记-第97页

1、检测原始值
javascript中有5中原始类型：string、number、boolean、null和undefined。
如果检测string、number、boolean和undefined，最佳选择是使用typeof运算符typeof variable 或者typeof(variable)对于null的判断，应该用恒等(===)或者非恒等(!==)
2、检测引用值
1）在javascript检测自定义类型时，最好的做法是使用instanceof运算符，这也是唯一的方法value instanceof constructor内置的javascript类型也应用这种方式
对于内置的javascript对象还可以使用下面这种方式

```
Object.prototype.toString.call(value) === '[object xxxx]'
```

2）检测function也用typeof
typeof myFunc === ‘ function ’
唯一的问题是IE8之前的版本，使用typeof来检测DOM节点中的函数会返回“ object ”而不是“ function ”，因为这些早期版本没有将DOM实现为内置的javascript方法，导致内置typeof运算符把这些函数识别为对象。
在IE8之前的版本检测DOM方法是否存在最安全的做法是用in运算符来检测在document是否存在

```
if ('querySelectorAll' in document) {

}
```

3）检测数组

```
function isArray(value) {
  if (typeof Array.isArray === 'function') {
    return Array.isArray(value);
  } else {
    return Object.prototype.toString.call(value) === '[object Array]';
  }
}
```

IE9+、firefox4+、safari5+、opera10.5+和chrome都实现了Array.isArray()方法
3、检测属性
判断属性是否存在最好的办法是使用in运算符。
如果只想检查实例对象的某个属性是否存在（不是prototype的属性和继承属性），则使用hasOwnProperty

```
if (object.hasOwnProperty('xxxx')) {
}
```

唯一需要注意的是在IE8以及更早的版本中，DOM对象并非继承Object，因此不包含hasOwnProperty方法，所以在调用DOM对象的hasOwnProperty方法之前先应该判断该方法是否存在

```
if ('hasOwnProperty' in object &amp;amp;&amp;amp; object.hasOwnProperty('xxxxx')) {
}
```

所以，在IE8之前的版本中判断属性是否存在，最好用in运算符

18、《编写可维护的JavaScript》的笔记-第39页

发现一个代码错误：
var prop =>应改为 var object
而且3.6节的前三段代码都有这个错误。
具体参见我的博客吧，懒得再写了。
http://zilong-thu.github.io/blog/2013/11/02/maintainable-javascript/

19、《编写可维护的JavaScript》的笔记-第73页

```
var YourGlobal = {
namespace: function(ns) {
var parts = ns.split("."),
object = this,
i, len;
for (i=0, len=parts.length; i &amp;lt; len; i++) {
if (!object[parts[i]]) {
object[parts[i]] = {};
}
object = object[parts[i]];
}
return object;
}
};
```

The variable YourGlobal can actually have any name. The important part is the name space() method, which nondestructively creates namespaces based on the string that is passed in and returns a reference to the namespace object. Basic usage:

```
/*
* Creates both YourGlobal.Books and YourGlobal.Books.MaintainableJavaScript.
* Neither exists before hand, so each is created from scratch.
*/
YourGlobal.namespace("Books.MaintainableJavaScript");
// you can now start using the namespace
YourGlobal.Books.MaintainableJavaScript.author = "Nicholas C. Zakas";
/*
* Leaves YourGlobal.Books alone and adds HighPerformanceJavaScript to it.
* This leaves YourGlobal.Books.MaintainableJavaScript intact.
*/
YourGlobal.namespace("Books.HighPerformanceJavaScript");
// still a valid reference
console.log(YourGlobal.Books.MaintainableJavaScript.author);
// You can also start adding new properties right off the method call
YourGlobal.namespace("Books").ANewBook = {};
```

Using a namespace() method on your one global allows developers the freedom to assume

that the namespace exists. That way, each file can call namespace() first to declare the namespace the developers are using, knowing that they won’t destroy the namespace if it already exists. This approach also frees developers from the tedious task of checking to see whether the namespace exists before using it.
As with other parts of your code, be sure to define some conventions around namespaces. Should they begin with uppercase letters as in YUI? Or be all lowercase as in Dojo? This is a question of preference, but defining these choices up front allows the team to use the one-global approach more effectively.这里不太懂，先标记一下。
Reference values are also known as objects. In JavaScript, any value that isn’t a primitive is definitely a reference.

```
// Internet Explorer 8 and earlier
console.log(typeof document.getElementById); // "object"
console.log(typeof document.createElement); // "object"
console.log(typeof document.getElementsByTagName); // "object"
```

This quirk arises due to how the browser implements the DOM. In short, these early versions of Internet Explorer didn’t implement the DOM as native JavaScript functions, which caused the native typeof operator to identify the functions as objects. Because the DOM is so well defined, developers typically test for DOM functionality using the in operator, understanding that the presence of the object member means that it’s a function, as in:

```
// detect DOM method
if ("querySelectorAll" in document) {
images = document.querySelectorAll("img");
}
```

This code checks to see whether querySelectorAll is defined in document, and if so, goes on to use that function. Though not ideal, this is the safest way to check for the presence of DOM methods if you need to support Internet Explorer 8 and earlier. In all other cases, the typeof operator is the best way to detect functions in JavaScript.
Detecting

All error types inherit from Error, so checking the type with instanceof Error doesn’t give you any useful information. By checking for the more specific error types, you get more robust error handling:

```
try {
// something that causes an error
} catch (ex) {
if (ex instanceof TypeError) {
// handle the error
} else if (ex instanceof ReferenceError) {
// handle the error
} else {
// handle all others
}
}var person = {
name: "Nicholas"
};
delete person.name;
```

console.log(person.name); // undefined
This example removes the name property from the person object. The delete operator works only on instance properties and methods. If delete is used on a prototype property or method, it has no effect. For example:
// No effect
delete document.getElementById;
console.log(document.getElementById("myelement")); // stil works
// Good
function getById (id) {
var element = null;
if (document.getElementById) { // DOM
element = document.getElementById(id);
} else if (document.all) { // IE
element = document.all[id];
} else if (document.layers) { // Netscape <= 4
element = document.layers[id];
}
return element;
}This is a good and appropriate use of feature detection, because the code tests for a feature and then, if it ’ s there, uses it. The test for document.getElementById() comes first because it is the standards-based solution. After that come the two browser-specific solutions. If none of these features is available, then the method simply returns null.
The best part about this function is that when Internet Explorer 5 and Netscape 6 were released with support for document.getElementById(), this code didn ’ t need to change.
The previous example illustrates several important parts of good feature detection:
1. Test for the standard solution
2. Test for browser-specific solutions
3. Provide a logical fallback if no solution is available

《编写可维护的JavaScript》

版权说明