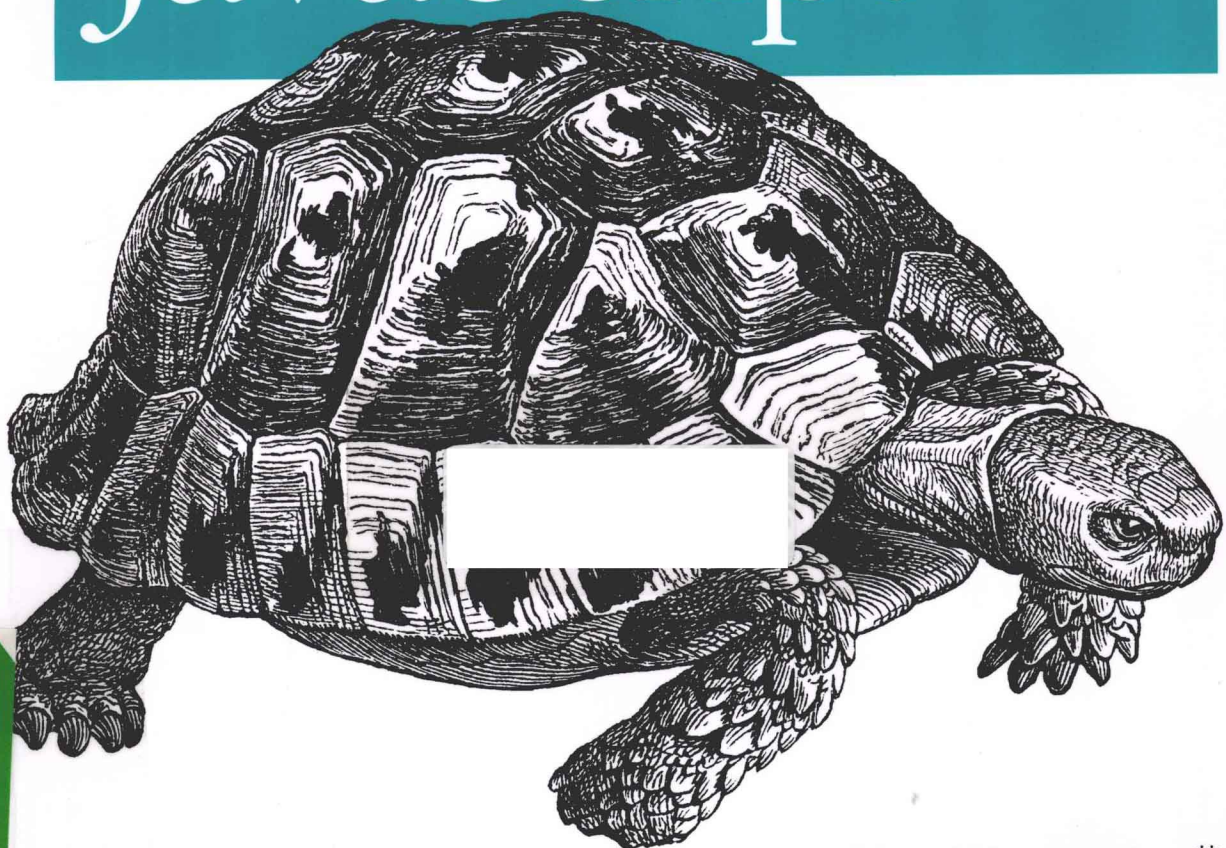
Maintainable JavaScript

编写可维护的 JavaScript



[美] *Nicholas C. Zakas* 著
李晶 郭凯 张散集 译

O'REILLY®
人民邮电出版社
POSTS & TELECOM PRESS

O'REILLY®

编写可维护的 JavaScript

[美] Nicholas C. Zakas 著

李 晶 郭 凯 张散集 译

人 民 邮 电 出 版 社

北 京

图书在版编目（CIP）数据

编写可维护的JavaScript / (美) 扎卡斯著
(Zakas, N. C.) 著 ; 李晶, 郭凯, 张散集译. -- 北京 :
人民邮电出版社, 2013.4
ISBN 978-7-115-31008-8

Ⅰ. ①编… Ⅱ. ①扎… ②李… ③郭… ④张… Ⅲ.
①JAVA语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2013)第025073号

编写可维护的 JavaScript

◆ 著 [美] Nicholas C. Zakas
译 李 晶 郭 凯 张散集
责任编辑 陈冀康

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京隆昌伟业印刷有限公司印刷

◆ 开本：787×1092 1/16
印张：15.5
字数：281 千字 2013 年 4 月第 1 版
印数：1－3 500 册 2013 年 4 月北京第 1 次印刷

著作权合同登记号 图字：01-2012-8552 号
ISBN 978-7-115-31008-8

定价：55.00 元

读者服务热线：(010)67132692 印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第 0021 号

目录

第一部分

编程风格

“程序是写给人读的，只是偶尔让计算机执行一下。”

——Donald Knuth。[①]

当你刚刚组建一个团队时，团队中的每个人都各自有一套编程习惯。毕竟，每个成员都有着不同的背景。有些人可能来自某个“皮包公司”（one-man shop），身兼数职，在公司里什么事都做；还有些人会来自不同的团队，对某种特定的做事风格情有独钟（或恨之入骨）。每个人都觉得代码应当按照自己的想法来写，这些通常被归纳为个人编程嗜好。在这个过程中[②]应当尽早将确定统一的编程风格纳入议题。

我们会经常碰到这两个术语：“编程风格”（style guideline）和“编码规范”（code convention）。编程风格是编码规范的一种，用来规约单文件中代码的规划。编码规范还包含编程最佳实践、文件和目录的规划以及注释等方面。本书集中讨论 JavaScript 的编码规范。

为什么要讨论编程风格

提炼编程风格是一道工序，花再多的时间也不为过。毕竟每个人都有自己的想法，

① 译注：高德纳（Donald Ervin Knuth）是世界顶级计算机科学家之一，被公认为现代计算机科学的鼻祖，著有《计算机程序设计艺术》（The Art of Computer Programming）等经典著作，在不多的业余时间里，Knuth 不仅写小说，还是一位音乐家、作曲家、管风琴设计师。

② 译注：意指组建团队的过程。

如果一天当中你有 8 小时是在写代码，那么你自然希望用一种舒服的方式来写代码。刚开始，团队成员对新的编程风格有点不适应，全靠强势的项目组长强制推行才得以持续。一旦风格确立后，这套编程风格就会促成团队成员高水准的协作，因为所有代码（的风格）看起来极为类似。

在团队开发中，所有的代码看起来风格一致是极其重要的，原因有以下几点。

- 任何开发者都不会在乎某个文件的作者是谁，也没有必要花费额外精力去理解代码逻辑并重新排版，因为所有代码排版格式看起来非常一致。我们打开一个文件时所干的第一件事，常常不是立即开始工作而是首先修复代码的缩进，当项目很庞大时，你会体会到统一的编程风格的确大幅度节省了时间成本。
- 我能很容易地识别出问题代码并发现错误。如果所有代码看起来很像，当你看到一段与众不同的代码时，很可能错误就产生在这段代码中。

毫无疑问，全球性的大公司都对外或对内发布过编程风格文档。

编程风格是个人的事情，只有放到团队开发中才能发挥作用。本书的这部分给出了 JavaScript 编码规范中值得关注（推荐）的方面。在某些场景中，很难说哪种编程风格好，哪种编程风格不好，因为有些编程风格只是某些人的偏好。本章不是向你灌输我个人的风格偏好，而是提炼出了编程风格应当遵循的重要的通用准则。本书附录 A 中给出了我个人的 JavaScript 编程风格。

有用的工具

开发编码指南是一件非常困难的事情——执行是另外一回事。在团队中通过讨论达成一致和进行代码评审（code review）时，每个人都很关注编码风格，但在平时大家却常常将这些抛在脑后。工具可以对每个人实时跟踪。这里有两个用来检查编程风格的工具，这两个工具非常有用：JSLint 和 JSHint。

JSLint 是由 Douglas Crockford 创建的。这是一个通用的 JavaScript 代码质量检查工具。最开始，JSLint 只是一个简单的查找不符合 JavaScript 模式的、错误的小工具。经过数年的进化，JSLint 已经成为一个有用的工具，不仅仅可以找出代码中潜在的错误，而且能针对你的代码给出编码风格上的警告。

Crockford 将他对 JavaScript 风格的观点分成了三个不同的部分。

- “JavaScript 风格的组成部分（第一部分）”（http://javascript.crockford.com/style1.html），包含基本的模式和语法。
- “JavaScript 风格的组成部分（第二部分）”（http://javascript.crockford.com/style2.html），包含一般性的 JavaScript 惯用法。
- “JavaScript 编程语言的编码规范”（http://javascript.crockford.com/code.html），这个规范更加全面，从前两部分中提炼出了编程风格的精华部分，同时增补了少量的编程风格指引。

JSLint 直接吸纳了很多 Crockford 所提炼的编程风格，而且很多时候我们无法关闭 JSLint 中检查编程风格的功能。所以 JSLint 是一个非常棒的工具，当然前提是你认可 Crockford 关于编程风格的观点。

JSHint 是 JSLint 的一个分支项目，由 Anton Kovalyov 创建并维护。JSHint 的目标是提供更加个性化的 JavaScript 代码质量和编程风格检查的工具。比如，当出现语法错误的时候，JSHint 几乎可以关掉所有编程风格检查，这样你可以完全自定义消息提示。Kovalyov 非常鼓励大家通过 GitHub（http://github.com）上的源代码库参与 JSHint 项目并为之贡献代码。

你可以将这些工具中的一种集成到打包过程中，通过这种方式推行编码规范是一个不错的方法。这种方法同时可以监控你的 JavaScript 代码中潜在的错误。

第 1 章

基本的格式化

编程风格指南的核心是基本的格式化规则（formatting rule）。这些规则直接决定了如何编写高水准的代码。与在学校学习写字时所用的方格纸类似，基本的格式化规则将指引开发者以特定的风格编写代码。这些规则通常包含一些你不太在意的有关语法的信息，但对于编写清晰连贯的代码段来说，每一条信息都是非常重要的。

1.1 缩进层级

关于 JavaScript 编码风格，我们首先要讨论的是（几乎所有的语言都是如此）如何处理缩进。对这个话题是可以争论上好几个小时的，缩进甚至关系到软件工程师的价值观。在确定编程风格之初应当首先确定缩进格式，这非常重要，以免工程师后续会陷入那个老生常谈的打开文件时二话不说先重排代码缩进的问题之中。来看一下这段代码（为了演示，这里故意修改了示例代码的缩进）。

```
if (wl && wl.length) {
            for (i = 0, l = wl.length; i < l; ++i) {
        p = wl[i];
        type = Y.Lang.type(r[p]);
        if (s.hasOwnProperty(p)) { if (merge && type == 'object') {

    Y.mix(r[p], s[p]);
} else if (ov || !(p in r)) {
                    r[p] = s[p];
            }
      }
```

```
    }
}
```

快速读懂这段代码并不容易。这里的缩进并不统一，一眼看去 else 是对应到第 1 行的 if 语句。但实际上这个 else 和代码第 5 行的 if 语句相对应。罪魁祸首是多位开发人员在同一段代码里应用了不同的缩进风格。这恰恰说明了统一缩进风格的重要性。如果有适当的缩进，这段代码将变得更加易读。

```
if (wl && wl.length) {
    for (i = 0, l = wl.length; i < l; ++i) {
        p = wl[i];
        type = Y.Lang.type(r[p]);
        if (s.hasOwnProperty(p)) {
            if (merge && type == 'object') {
                Y.mix(r[p], s[p]);
            } else if (ov || !(p in r)) {
                r[p] = s[p];
            }
        }
    }
}
```

坚持使用适度的缩进是万里长征的第一步——本章在下面将提到这种做法可以带来其他可维护性方面的提升。

对于大多数编程风格来说，代码到底应该如何缩进并没有统一的共识。有两种主张。

使用制表符进行缩进

每一个缩进层级都用单独的制表符（tab character）表示。所以一个缩进层级是一个制表符，两个缩进层级为两个制表符，以此类推。这种方法有两个主要的好处。第一，制表符和缩进层级之间是一对一的关系，这是符合逻辑的。第二，文本编辑器可以配置制表符的展现长度[①]，因此那些想修改缩进尺寸的开发者可以通过配置文本编辑器来实现，想长即长，想短可短。使用制表符作缩进的主要缺点是，系统对制表符的解释不一致。你会发觉在某个系统中用一款编辑器打开文件时看到的缩进，和在另外一个系统中用相同的编辑器打开文件时看到的不一样。对于那些追求（代码展现）一致性的开发者来说，这会带来一些困惑。这些差异、争论会导致不同的开发者对同一段代码有不同的看法的，而这些正是团队开发需要规避的。

① 译注：通常一个制表符长度相当于 4 个字符。

使用空格符进行缩进

每个缩进层级由多个空格字符组成。在这种观点中有三种具体的做法：2 个空格表示一个缩进，2 个空格表示一个缩进，以及 8 个空格表示一个缩进。这三种做法在其他很多编程语言中都能找到渊源。实际上，很多团队选择 4 个空格的缩进，对于那些习惯用 2 个空格缩进和用 8 个空格缩进的人来说，4 个空格缩进是一种折中的选择。使用空格作缩进的好处是，在所有的系统和编辑器中，文件的展现格式不会有任何差异。可以在文本编辑器中配置敲击 Tab 键时插入几个空格。也就是说所有开发者都可以看到一模一样的代码呈现。使用空格缩进的缺点是，对于单个开发者来说，使用一个没有配置好的文本编辑器创建格式化的代码的方式非常原始。

尽管有人争辩说应当优先考虑使用一种缩进约定，但说到底这只是一个团队偏好的问题。这里我们给出一些各式各样的缩进风格作为参考。

- jQuery 核心风格指南（jQuery Core Style Guide）明确规定使用制表符缩进。
- Dauglas Crockford 的 JavaScript 代码规范（Douglas Crockford's Code Conventions for the JavaScript Programming Language）规定使用 4 个空格字符的缩进。
- SproutCore 风格指南（SproutCore Style Guide）规定使用 2 个空格的缩进。
- Goolge 的 JavaScript 风格指南（Google JavaScript Style Guide）规定使用 2 个空格的缩进。
- Dojo 编程风格指南（Dojo Style Guide）规定使用制表符缩进。

我推荐使用 4 个空格字符为一个缩进层级。很多文本编辑器都默认将缩进设置为 4 个空格，你可以在编辑器中配置敲入 Tab 键时插入 4 个空格。使用 2 个空格的缩进时，代码的视觉展现并不是最优的，至少看起来是这样。

尽管是选择制表符还是选择空格做缩进只是一种个人偏好，但绝对不要将两者混用，这非常重要。这么做会导致文件的格式很糟糕，而且需要做不少清理工作，就像本节的第一段示例代码显示的那样。

1.2 语句结尾

有一件很有意思且很容易让人困惑的事情，那就是 JavaScript 的语句要么独占一行，要么以分号结尾。类似 C 的编程语言，诸如 C++和 Java，都采用这种行结束写法，即结尾使用分号。下面这两段示例代码都是合法的 JavaScript。

```
// 合法的代码
var name = "Nicholas";
function sayName() {
    alert(name);
}
// 合法的代码，但不推荐这样写
var name = "Nicholas"
function sayName() {
    alert(name)
}
```

有赖于分析器的自动分号插入（Automatic Semicolon Insertion，ASI）机制，JavaScript 代码省略分号也是可以正常工作的。ASI 会自动寻找代码中应当使用分号但实际没有分号的位置，并插入分号。大多数场景下 ASI 都会正确插入分号，不会产生错误。但 ASI 的分号插入规则非常复杂且很难记住，因此我推荐不要省略分号。看一下这段代码。

```
// 原始代码
function getData() {
    return
    {
        title: "Maintainable JavaScript",
        author: "Nicholas C. Zakas"
    }
}
// 分析器会将它理解成
function getData() {
    return;
    {
        title: "Maintainable JavaScript",
        author: "Nicholas C. Zakas"
    };
}
```

在这段代码中，函数 getData()的本意是返回一个包含一些数据的对象。然而，return 之后新起了一行，导致 return 后被插入了一个分号，这会导致函数返回值是 undefined。

可以通过将左花括号移至与 return 同一行的位置来修复这个问题。

```
// 这段代码工作正常，尽管没有用分号
function getData() {
    return {
        title: "Maintainable JavaScript",
        author: "Nicholas C. Zakas"
    }
}
```

ASI 在某些场景下是很管用的，特别是，有时候 ASI 可以帮助减少代码错误。当某个场景我们认为不需要插入分号而 ASI 认为需要插入时，常常会产生错误。我发现很多开发人员，尤其是新手们，更倾向于使用分号而不是省略它们。

Douglas Crockford 针对 JavaScript 提炼出的编程规范（下文统称为 Crockford 的编程规范）推荐总是使用分号，同样，jQuery 核心风格指南、Google 的 JavaScript 风格指南以及 Dojo 编程风格指南都推荐不要省略分号。如果省略了分号，JSLint 和 JSHint 默认都会有警告。

1.3 行的长度

和缩进话题息息相关的是行的长度。如果一行代码太长，编辑窗口出现了横向滚动条，会让开发人员感觉很别扭。即便是在当今的宽屏显示器中，保持合适的代码行长度也会极大地提高工程师的生产力。很多语言的编程规范都提到一行代码最长不应当超过 80 个字符。这个数值来源于很久之前文本编辑器的单行最多字符限制，即编辑器中单行最多只能显示 80 个字符，超过 80 个字符的行要么折行，要么被隐藏起来，这些都是我们所不希望的。相比 20 年前的编辑器，现在的文本编辑器更加精巧，但仍然有很多编辑器保留了单行 80 个字符的限制。此外关于行长度，还有一些常见的建议。

1. Java 语言编程规范中规定源码里单行长度不超过 80 个字符，文档中代码单行长度不超过 70 个字符。

2. Android 开发者编码风格指南规定单行代码长度不超过 100 个字符。

3. 非官方的 Ruby 编程规范中规定单行代码长度不超过 80 个字符。