前言

在互联网发展的早期，JavaScript 就已经成为了支撑网页内容交互体验的基础技术。那时 JavaScript 的作用可能仅仅是生成一些闪烁的鼠标轨迹或者烦人的弹出窗口，但是经过了大约 20 年的发展，JavaScript 的技术和能力都发生了天翻地覆的变化，现在的 JavaScript 毫无疑问已经成为了世界上使用范围最广的软件平台——互联网——的核心技术。

但是作为一个语言来说，它总是成为大家批评的对象，部分原因是它有很多历史遗留问题，但主要原因是它的设计哲学有问题。就像 Brendan Eich 曾经说过的，JavaScript 甚至连名字都给人一种“蠢弟弟”的感觉，就像是它更成熟的大哥 Java 的不完整版本。不过名字只不过是营销策略上的一个意外，这两个语言有许多本质上的区别。JavaScript 和 Java 的关系，就像 Carnival（嘉年华）和 Car（汽车）的关系一样，八竿子打不着。

JavaScript 借鉴了许多语言的概念和语法，比如 C 风格的过程式编程以及不太明显的 Scheme/List 风格的函数式编程，因此吸引了许多开发者，甚至是那些不会编程的新手。用 JavaScript 来编写“Hello World”是非常简单的，因此这门语言很有吸引力并且很好上手。

虽然 JavaScript 可能是最早出现的语言之一，但是由于其本身的特殊性，相比其他语言，能真正掌握 JavaScript 的人比较少。如果想用 C、C++ 这样的语言编写功能全面的程序，那需要对语言有很深的了解。但是对于 JavaScript 来说，编写功能全面的程序仅仅是冰山一角。

JavaScript 语言本质上有许多复杂的概念，但是却用一种*看起来*比较简单的方式体现出来，比如回调函数，因此 JavaScript 开发者通常只是简单地使用这些特性，并不会关心语言内部的实现原理。

JavaScript 既是一门充满吸引力、简单易用的语言，又是一门具有许多复杂微妙技术的语言，即使是经验丰富的 JavaScript 开发者，如果没有认真学习的话也无法真正理解它们。

这就是 JavaScript 的矛盾之处，也是这门语言的阿喀琉斯之踵[1]。由于 JavaScript 不必理解就可以使用，因此通常来说很难真正理解语言本身，这就是我们面临的挑战。

使命

如果每次遇到 JavaScript 中出乎意料的行为时，你的反应就是把它加入黑名单（很多人都是这么做的），那用不了多久你就会把 JavaScript 语言真正的多样性全部排除。

剩下的部分就是非常著名的“好的部分”（Good Parts），但是亲爱的读者们，我恳请你们把它称作“简单的部分”、“安全的部分”甚至“不完整的部分”。

“你不知道的 JavaScript”系列丛书要做的事恰好相反：学习并且深入理解整个 JavaScript，尤其是那些“难的部分”。

在本书中，我们要直面当前 JavaScript 开发者不求甚解的大趋势，他们往往不会深入理解语言内部的机制，遇到困难就会退缩。我们要做的恰好相反，不是退缩，而是继续前进。

你们应当像我一样，不满足于只是让代码正常工作，而是想要弄清楚“为什么”。我希望你能勇于挑战这条崎岖颠簸的“少有人走的路”，拥抱整个 JavaScript。掌握了这些知识之后，无论什么技术、框架和流行词语你都能轻松理解。

这个系列中的每本书专注于语言中一个最容易被误解或者最难理解的核心部分，进行深入、详尽的介绍。在阅读本书时，你应当审视自己对于 JavaScript 的理解，仔细思考书中讲解的理论和那些“你需要知道”的东西。

现在你所理解的 JavaScript 很可能是从别人那里学来的不完整版。这样的 JavaScript 只是真正的 JavaScript 的影子。学完这个系列之后，你就会掌握真正的 JavaScript。读下去吧，我的朋友，JavaScript 恭候你的光临。

小结

JavaScript 非常特殊，只学一部分的话非常简单，但是想要完整地学习会很难（就算学到够用也不容易）。当开发者感到迷惑时，他们通常会责怪语言本身，而不是怪自己对语言缺乏了解。这个系列就是为了解决这个问题，让你打心眼儿里欣赏这门语言。

本书中的许多例子都需要运行在即将到来的现代 JavaScript 引擎环境中，比如 ES6。部分代码在旧（ES6 之前的）引擎上可能无法正常运行。

注 1：指某人或某事物的最大或者唯一弱点，即罩门关键所在。——译者注

本书排版约定

本书中使用以下排版约定。

- 楷体
  表示新的术语。
- 等宽字体
  表示代码段以及段落中的程序元素，比如变量、函数名、数据库、数据类型、环境变量、语句以及关键字。
- 等宽粗体
  表示命令中不可改动的部分。
- 等宽斜体
  表示将由用户提供的值（或由上下文确定的值）替换的文本。

这个图标表示提示或建议。

这个图标表示重要说明。

这个图标表示警告或提醒。

使用代码示例

可以在这里下载本书第一部分“作用域和闭包”随附的资料（代码示例、练习题等）：http://bit.ly/1c8HEWF。

可以在这里下载本书第二部分“`this` 和对象原型”随附的资料（代码示例、练习题等）：http://bit.ly/ydkjs-this-code

让本书助你一臂之力。也许你需要在自己的程序或文档中用到本书中的代码。除非大段大

段地使用，否则不必与我们联系取得授权。例如，无需请求许可，就可以用本书中的几段代码写成一个程序。但是销售或者发布 O'Reilly 图书中代码的光盘则必须事先获得授权。引用书中的代码来回答问题也无需授权。将大段的示例代码整合到你自己的产品文档中则必须经过许可。

使用我们的代码时，希望你能标明它的出处，但不强求。出处信息一般包括书名、作者、出版商和书号，例如：*Scope and Closures*，Kyle Simpson 著（O'Reilly，2014）。版权所有，978-1-491-33558-8。

如果还有关于使用代码的未尽事宜，可以随时与我们联系：permissions@oreilly.com。

Safari® Books Online

Safari Books Online（http://www.safaribooksonline.com）是应需而变的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。

Safari Books Online 是技术专家、软件开发人员、Web 设计师、商务人士和创意人士开展调研、解决问题、学习和认证培训的第一手资料。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，我们网上见。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

O’Reilly 的每一本书都有专属网页，你可以在那儿找到本书的相关信息，包括勘误表、示例代码以及其他信息。本书第一部分“作用域和闭包”的网址是 http://oreil.ly/JS_scope_closures。本书第二部分“`this` 和对象原型”的网址是 http://bit.ly/ydk-js-this-object-prototypes。

对于本书的评论和技术性问题，请发送电子邮件到：

bookquestions@oreilly.com

要了解更多 O’Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址如下：http://facebook.com/oreilly

请关注我们的 Twitter 动态：http://twitter.com/oreillymedia

我们的 YouTube 视频地址如下：http://www.youtube.com/oreillymedia

要查看“你不知道的 JavaScript”系列丛书中的全部图书，请访问：

http://YouDontKnowJS.com

第一部分

作用域和闭包

[美] Kyle Simpson 著
赵望野 译

序

小时候，我特别喜欢把东西拆成零部件，然后再重新装回去——旧的移动电话、立体声音响等我能拿到的一切物件都没能幸免。对于年幼的我来说，使用这些东西还为时过早，但是一旦它们坏了，我就立刻想弄清楚它们的工作原理。

记得有一次，我看见一个老式收音机的电路板，其中有一个缠满铜线的奇怪长管。我不知道这个长管的用途，所以立刻开始研究它。它有什么用？为什么出现在收音机里？为什么它看起来和电路板的其他部分不一样？为什么会有铜线缠绕着它？如果我把铜线拆下来会发生什么？现在我知道了，这是一个在晶体管收音机中很常见的、由缠绕着铜线的铁氧体棒制成的环形天线。

你是否也曾对解答各种各样的为什么很上瘾？大多数孩子都会。事实上，这可能是孩子身上我最喜欢的地方——求知欲很强。

很遗憾，现在我从事着一份专业性的工作，并以制作一些东西来度日。而我儿时的梦想是有一天能够制作那些被我拆开过的东西。当然，现在我所制作的大部分东西都是用 JavaScript 做成的，而不是铁氧体棒……但它们很相似！尽管我曾经一度非常热爱制作东西，但是现在却更渴望了解事物的运行原理。我经常寻找解决问题或修复 bug 的最佳方法，却很少花时间来研究我所使用的工具。

这也是为什么我一看到“你不知道的 JavaScript”系列图书就很激动，因为 JavaScript 的确有很多我不了解的地方。我每天从早到晚都在使用 JavaScript，并且已经持续了好几年，但我真的了解它了吗？答案是否定的。当然，我了解它的很多细节，并且经常阅读标准文档和邮件列表中的内容，但是了解的程度低于我内心那个六岁的孩子希望我达到的水平。

第一部分“作用域和闭包”是一个非常好的切入点。它对于如我一般的受众来说非常有用（希望对你也同样有用）。这本书并不会教你如何使用 JavaScript，但是它会让你意识到对于其内部的运行原理你可能了解得并不多。同时这本书出现的时机也非常巧：ES6 终于稳定下来了，并且各家浏览器的实现工作也正在逐步展开。如果你还没有学习其中的新功能（比如 `let` 和 `const`），这本书将起到很好的介绍作用。

所以我希望你能喜欢这本书，尤其希望 Kyle 对 JavaScript 工作原理每一个细节的批判性思考会渗透到你的思考过程和日常工作中。知其然，也要知其所以然。

Shane Hudson
www.shanehudson.net

第 1 章

作用域是什么

几乎所有编程语言最基本的功能之一，就是能够储存变量当中的值，并且能在之后对这个值进行访问或修改。事实上，正是这种储存和访问变量的值的能力将状态带给了程序。

若没有了状态这个概念，程序虽然也能够执行一些简单的任务，但它会受到高度限制，做不到非常有趣。

但是将变量引入程序会引起几个很有意思的问题，也正是我们将要讨论的：这些变量住在哪里？换句话说，它们储存在哪里？最重要的是，程序需要时如何找到它们？

这些问题说明需要一套设计良好的规则来存储变量，并且之后可以方便地找到这些变量。这套规则被称为作用域。

但是，究竟在哪里而且怎样设置这些作用域的规则呢？

1.1 编译原理

尽管通常将 JavaScript 归类为“动态”或“解释执行”语言，但事实上它是一门编译语言。这个事实对你来说可能显而易见，也可能你闻所未闻，取决于你接触过多少编程语言，具有多少经验。但与传统的编译语言不同，它不是提前编译的，编译结果也不能在分布式系统中进行移植。

尽管如此，JavaScript 引擎进行编译的步骤和传统的编译语言非常相似，在某些环节可能比预想的要复杂。

在传统编译语言的流程中，程序中的一段源代码在执行之前会经历三个步骤，统称为“编译”。

- *分词/词法分析*（Tokenizing/Lexing）
  这个过程会将由字符组成的字符串分解成（对编程语言来说）有意义的代码块，这些代码块被称为词法单元（token）。例如，考虑程序 `var a = 2;`。这段程序通常会被分解成为下面这些词法单元：`var`、`a`、`=`、`2` 、`;`。空格是否会被当作词法单元，取决于空格在这门语言中是否具有意义。

分词（tokenizing）和词法分析（Lexing）之间的区别是非常微妙、晦涩的，主要差异在于词法单元的识别是通过有状态还是无状态的方式进行的。简单来说，如果词法单元生成器在判断 `a` 是一个独立的词法单元还是其他词法单元的一部分时，调用的是有状态的解析规则，那么这个过程就被称为*词法分析*。

- *解析/语法分析*（Parsing）
  这个过程是将词法单元流（数组）转换成一个由元素逐级嵌套所组成的代表了程序语法结构的树。这个树被称为“抽象语法树”（Abstract Syntax Tree，AST）。
  `var a = 2;` 的抽象语法树中可能会有一个叫作 `VariableDeclaration` 的顶级节点，接下来是一个叫作 `Identifier`（它的值是 `a`）的子节点，以及一个叫作 `AssignmentExpression` 的子节点。`AssignmentExpression` 节点有一个叫作 `NumericLiteral`（它的值是 `2`）的子节点。

- *代码生成*
  将 AST 转换为可执行代码的过程称被称为代码生成。这个过程与语言、目标平台等息息相关。
  抛开具体细节，简单来说就是有某种方法可以将 `var a = 2;` 的 AST 转化为一组机器指令，用来*创建*一个叫作 `a` 的变量（包括分配内存等），并将一个值储存在 `a` 中。

关于引擎如何管理系统资源超出了我们的讨论范围，因此只需要简单地了解引擎可以根据需要创建并储存变量即可。

比起那些编译过程只有三个步骤的语言的编译器，JavaScript 引擎要复杂得多。例如，在语法分析和代码生成阶段有特定的步骤来对运行性能进行优化，包括对冗余元素进行优化等。