Web开发经典丛书

Electron in Action

Electron 跨平台开发实战

[美] 史蒂文•金尼(Steve Kinney) 著
涂曙光 译

MANNING

清华大学出版社

Web 开发经典丛书

Electron 跨平台开发实战

[美] 史蒂文·金尼(Steve Kinney) 著

涂曙光 译

清华大学出版社

北 京

Steve Kinney
Electron in Action
EISBN: 978-1617294143

图书在版编目(CIP)数据

Electron 跨平台开发实战 / (美)史蒂文·金尼(Steve Kinney) 著；涂曙光 译. —北京：清华大学出版社，2019
(Web 开发经典丛书)
书名原文：Electron in Action
ISBN 978-7-302-53489-1

Ⅰ. ①E… Ⅱ. ①史… ②涂… Ⅲ. ①JAVA 语言－程序设计 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字(2019)第 179546 号

责任编辑：王　军
装帧设计：孔祥峰
责任校对：成凤进
责任印制：宋　林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市宏图印务有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张**：25.75　　**字　　数**：518 千字
版　　次：2019 年 9 月第 1 版　　**印　　次**：2019 年 9 月第 1 次印刷
定　　价：98.00 元

产品编号：082776-01

译 者 序

第一次认识到 Electron 的强大潜力，是在安装并体验了 Visual Studio Code 之后。我惊诧于通过 Electron 框架竟然可以编写出如此复杂且强大的应用，更重要的是，这种应用还可以运行在三大主流操作系统(Windows、macOS 和 Linux)之上。它所展现出的这种强大潜力吸引我进一步去了解和学习 Electron，并在这个过程中不断地被它的简洁、强大和无比活跃的社区所折服。

《Electron 跨平台开发实战》是一本注重实战的书。只要读者具备基本的 JavaScript 编码能力，且对 Node.js 了解一二，就一定能够毫不费力地阅读本书，轻松掌握使用 Electron 创建跨平台应用程序的技能。虽然 Electron 官网上有中文文档，但是文档仅仅专注于 Electron 框架的各个技术点，并没有循序渐进地介绍如何从头到尾创建一个 Electron 应用，也没有涵盖在创建 Electron 应用的过程中必然涉及的其他一些需求(比如，应用采用哪种最合适的数据本地存储方案？如何在 Electron 应用中使用 React 框架作为 UI 界面框架？)，官方中文文档并未尝试成为一份 Electron 学习教程，而这恰恰是本书的主要撰写目的。

在本书的翻译过程中，我得到了申贝贝和刘勤的许多协助。他们对初稿进行了技术审阅，并毫不吝啬地指出并修正了诸多文法与用词上的不当之处。在此对他们的辛勤付出表达最衷心的感谢！

希望读者能够享受阅读本书的过程，并体会学习 Electron 的乐趣。

涂曙光

2019 年 4 月于深圳

序　言

Electron 是现今最令我兴奋的技术之一，希望你阅读本书时也能感受到那种兴奋之情。正如我在本书中所述，Electron 既可以让 Web 开发者使用 Web 开发技能突破浏览器的限制，创建功能更强大的桌面应用，也可以让他们为命令行工具创建图形用户界面(Graphical User Interfaces，GUI)，从而将充满创意的产品推广给更多对命令行不熟悉的普通用户。Electron 为 Web 技术搭建了一座更强大、更丰富应用程序的桥梁。

Electron 很罕见地在开源社区掀起了一股旋风。它既足够底层，让你能够快速掌握其基础知识，又足够强大，让你可以构建出超级复杂的应用。它封装了创建桌面应用所必须做的一些繁琐复杂工作，但又不会让你觉得它是一个过于神秘的黑盒子。Electron 有一个热情的社区提供支持，这个社区向开发者提供各种有用的工具库，帮助他们轻松实现各种功能。

你也许已经通过 Atom、Slack、Visual Studio Code 或其他某个品牌程序，知道了 Electron 的存在。但是，本书主要写给想要创建一些新奇原创作品的爱好者和独立开发者。Electron 在大型团队中很受欢迎，但对于想要创建自娱自乐应用的个人开发者，或者想要创建内部工具的小型团队而言，Electron 同样是很好的选择。

一旦熟悉了 Electron，就将打开一扇通往各种可能性的大门。你在熟练掌握了 Electron 之后，会觉得它看起来很酷，但是一时之间也很难立马想出能够用它做点儿什么。没有关系，这时只需要让它在你的脑海中停留一阵子，很快，你的脑子里就会冒出各种应用的新点子。

这种事也曾在我身上发生过，我已经向许多人传授过 Electron，我猜测大部分人或多或少都会有类似的体验。《Electron 跨平台开发实战》成书源于我曾在美国和哥伦比亚旅行，举办的各种使用 Electron 创建跨平台桌面应用的讲座的经历。Manning 出版社邀请我撰写一本 Electron 题材的书，我抓住了这个机会，将本书写了出来。本书也被用于我的讲座中，而讲座也为我提供了全新的视野，帮助我对本书进行改进。

书稿如果一直留在我的计算机中，我会不停地对它进行改进。它帮助我不断厘清

创建 Electron 应用的思路，在处理项目中新的挑战和实现功能时，它还变身为一份工作日志。撰写图书是一个不断重复的良性循环，虽然我很想念这个过程，但还是很高兴在本书出版之日，终于不必再将自己的每个夜晚和周末都花费在它身上。

作 者 简 介

Steve Kinney 是 SendGrid 的首席工程师，也是一名国际演讲者和 DinosaurJS 会议(在位于科罗拉多州丹佛市举办的 JavaScript 技术会议)的组织者。此前，他是软件与设计图灵学院前端工程项目的创始主任，并曾经在纽约市从事过七年教学工作。

致　谢

撰写一本书最让我感到惊讶的事情是它竟然包含如此之多的工作，它的完成牵涉如此之多的人。首先，我需要感谢我的妻子 Logan，她容忍我把夜晚和周末的时间都花费在了这本书上，而这些时间原本应该属于我的家人。她的宽容和支持对本书的诞生至关重要。

感谢 Helen Stergius，是她给了我一个宽松的交稿期限，让我可以平衡自己的家庭、日常工作和本书的写作。无论我的时间多么紧张，Helen 总能保持积极的心态。感谢 Nickie Buckner 在我撰写本书时运行了其中所有的代码，给我鼓励，并纠正了拼写错误。感谢 Doug Warren 在本书接近完成时对它做了最后的技术审校。感谢 Brian Sawyer 在最开始出面邀请我撰写本书，以及 Marjan Bace 为本书开了绿灯。

感谢 Frontend Masters 的 Marc Grabanski，让我有机会在全球受众面前讲授本书的内容，并向我提供富有洞察力的反馈。感谢 Jeff Casimir 向我提供了一个平台，去教育无数的软件开发新手。感谢 Meeka Gayhart、Louisa Barrett、Jhun de Andreas、Brenna Martenson 和 Brittany Storoz，我在他们需要我时掉了链子，但他们不但容忍了我，还待我如常。

感谢以下评论者，他们在本书撰写过程中阅读了本书，并在论坛中留下了自己的反馈：Aiden Mark Humphreys、Alan Bogusiewicz、Alexey Galiulin、Anto Aravinth、Ashwin Raj、Buu Nguyen、Daniel Posey、Frederic Flayol、Harald Kuhn、Hari Khalsa、Iain Shigeoka、Jay Kelkar、Jim McGinn、Jimmy Qiu、Jon Riddle、Matteo Gildone、MladenĐurić、PhilippeCharrière、Raq Khan 和 William Wheeler。你们帮助改进了本书的内容，找出了其中的错误。

感谢 Cheng Zhao 和所有维护 Electron 的人。没有你们完成的那些难以置信的工作，本书也就不复存在了。此外，你们对细节的关注和用户优先的心态，使我可以很容易地讲解如何实现在其他平台上很难实现的功能。你在阅读《Electron 跨平台开发实战》的过程中，会无数次发现只需要使用 Electron 内置的某个 API，就能解决一个棘手问题。感谢技术社区做出的惊人贡献，他们提供了一个第三方生态系统，帮助开发者创建 Electron 应用。有些时候，Electron 没有对某个问题提供内置的方案，但总有一个第

三方库可以解决这个问题。对于一个开发平台来说，只有社区好，才是真的好。

最后，感谢丹佛市的 Novo Coffee，感谢他们向我提供了冷饮和写作座位，以及必不可少的咖啡因。

前　言

《Electron 跨平台开发实战》的主要目标是帮助你快速着手创建 Electron 应用。通过将各种基础概念体现于实践代码之中，我们可以更好地学习这些概念。这本书不仅介绍 Electron 的基础知识，还会提供创建新应用的灵感和点子。

本书读者对象

本书适用于这样一些受众：想要创建功能远远超出浏览器功能的桌面应用的人；想要为自己创建一款满足个人需要的桌面应用，但又不希望重新学习一门新的编程语言或框架的人；想要突破自身能力，使用同一份代码创建运行于多个操作系统之上应用的小型团队。而在笔者的内心深处，本书既适用于想要为命令行应用创建 GUI 界面的人，也适用于希望无须在计算机上安装 Node.js 即可运行应用程序或工具的人。

本书假设读者已经熟练掌握了 JavaScript，但是由于读者的技术背景各不相同，有些读者可能只对某个领域比较有经验，所以如果书中讲到了读者可能不熟悉的 Web 或 Node.js 知识点，我会对它们进行讲解。

本书的组织结构：路线图

本书分为 16 章。各章的内容都前后衔接、循序渐进。如果只想了解如何在你的应用中实现某个特定功能，也可以直接跳到相关章节进行阅读。

第 1 章将讲述 Electron 是什么。我们可以看到一些 Electron 能够做到，而浏览器和 Node.js 无法独自做到的事情。

第 2 章将开始创建一个简单的 Electron 应用。该章的目标是通过亲自动手实践，让读者了解使用 Electron 创建一个应用是多么容易。

第 3 章将介绍本书中的一个主要应用——Fire Sale，这是一个让用户可以打开文件系统中的 Markdown 文件并对它们进行编辑的应用。

第 4 章将介绍如何使用原生系统对话框和警示框，用户可以使用对话框从文件系

统中选择一个文件并在 Fire Sale 中编辑它。Fire Sale 应用会模糊 DOM 与 Node 标准库之间的界线，协调两者以实现对话框功能。

第 5 章将向应用添加多窗口支持，多窗口会引出一系列单浏览器窗口应用和无窗口 Node.js 应用不会遇到的问题。

第 6 章将引入更多的原生操作系统集成功能。我们将把 Fire Sale 中打开的文档添加到操作系统的最近打开文件列表中，创建一个检查当前打开的文件是否已被其他应用修改的监听器，并根据当前文件是否有未保存的更改，更新窗口标题栏中显示的文本。

第 7 章将探讨如何创建原生应用菜单和上下文菜单，前者被一个应用的所有窗口共享，后者会在用户右击鼠标时弹出。

第 8 章将讲解如何基于应用的状态更新应用菜单，包括启用或禁用指定的菜单项。

第 9 章将创建一种驻留在 macOS 的菜单栏中或 Windows 的系统托盘区域的新型应用。Web 应用是无法将自己驻留在这些区域的。你在该章会创建 Clipmaster 应用，这是一个精简的剪贴板管理器，可以从系统剪贴板读取和写入内容，响应全局快捷键，并在完成操作时向用户显示通知。

第 10 章将使用第三方库，创建一个像 Fire Sale 那样有 UI 界面的新版 Clipmaster 应用。这个新版本的名称是 Clipmaster 9000，它可以访问 GitHub 的 Gist API，并且可以通过快捷键把保存的剪贴项发布到另一个 Web 系统。

之前我们一直在使用标准 JavaScript 语言实现 Electron 应用的功能。在第 11 章，将展示如何在应用中使用 Babel、TypeScript 和 CoffeeScript 等可转译为 JavaScript 的语言，以及 Sass 和 Less 等可转译为 CSS 的语言。你在该章会使用 React 创建一个名为 Jetsetter 的旅行打包列表应用。

第 12 章将介绍如何将数据保存到数据库，将演示在客户端代码中，如何使用 SQLite 数据库实现数据读写操作。除了 SQLite 之外，还会演示使用基于浏览器的 IndexedDB 数据库。

第 13 章将介绍 Spectron 这个测试工具，它让开发者可以为 Electron 应用编写 Selenium 测试代码。我们将为前面章节创建的 Clipmaster 9000 应用编写一组测试。

第 14 章将介绍对应用进行打包的工具。使用工具打包应用之后，可以将它们分发给不想仅通过命令行启动应用的用户。其实不光是用户，很多开发者也不太喜欢使用命令行启动应用。

第 15 章涵盖如何为 macOS 应用签名，如何创建 Windows 安装程序，以及如何创建用于收集错误日志和崩溃报告的简单服务器。

第 16 章将讲解如何把应用发布到 Mac App Store。如果你更喜欢自己分发应用，就不必把应用发布到 Mac App Store。但是，如果你确实想那样做但又对流程不熟悉，那么该章对你会非常有帮助。

关于代码

本书包含许多源代码示例，一些以单独的代码清单形式出现，另一些则直接插入正文。有时，代码会以粗体标出，以表示它们相对于当前章节的前面部分已经发生变化，这通常是由于向原有代码添加新功能导致的。

在许多场景中，原始代码都被重新格式化，我们做了折行处理，并重新规划了缩进，以满足排版需求。在极少的情况下，即使重新格式化代码也仍然不够，所以会用到行连续符号(➥)，表示书页上的两行实际上是连续的单行代码。此外，使用文本对代码进行说明时，通常会从代码清单中将注释删除。许多代码清单都带有额外的文字标注，以突出代码里面重要的概念。

本书所有代码都可以在出版社网站(www.manning.com/books/electron-in-action)和GitHub(https://github.com/electron-in-action)上找到，也可通过扫描封底的二维码下载。大部分情况下，每一章都会有一个对应的分支。本书后半部分的一些章会使用前面某章创建的应用，因此会在那一章开始的时候提供一个起始分支，在那一章结束的时候再提供一个结束分支。如果某章结束时的最终代码比较短，则将它直接列在那一章的结尾。如果某章结束时的最终代码比较长，则将完整代码放在附录中。2018 年 5 月，GitHub 发布了一个 Web 服务和一个 npm 包，通过 GitHub 发布的开源 Electron 应用可以利用它们轻松实现自动升级(https://electronjs.org/blog/autoupdating-electron-apps)。如果你的应用符合那些条件，可以考虑使用 GitHub 提供的 update-electron-app 库。如果你的应用并不符合条件，或者你不想使用 update-electron-app 库，第 15 章将讲述如何自己实现自动升级功能。

在本书撰写过程中，最让笔者心惊肉跳的事是 Electron、Node.js 或 Chromium 发布了新版本(即使只是小的版本)，导致书中的某个代码示例无法工作。这种事已经发生过不止一次。

我承诺将不断更新本书代码，保持它们有效，并在每章分支的 README.md 文件中提供勘误。如果书中的代码不能正常工作，请务必查看本书的 GitHub 代码库或图书论坛。

图书论坛

读者可以免费访问由 Manning Publications 运营的私人网络论坛，在论坛上发表评论、询问技术问题，并得到来自作者与其他用户的帮助。论坛位于 https://forums.manning.com/forums/electron-in-action。有关 Manning 论坛和行为规则的更多信息，请访问 https://forums.manning.com/forums/about。

目　录

第Ⅰ部分　Electron 入门知识

第 1 章　介绍 Electron ····· 3
1.1　什么是 Electron ····· 3
1.1.1　什么是 Chromium Content Module ····· 5
1.1.2　什么是 Node.js ····· 6
1.2　哪些人在使用 Electron ····· 6
1.3　阅读本书之前，需要知道些什么 ····· 8
1.4　为何要使用 Electron ····· 8
1.4.1　重用现有技能 ····· 10
1.4.2　访问原生操作系统 API ····· 10
1.4.3　更高运行权限，更少功能限制 ····· 11
1.4.4　在浏览器环境中使用 Node 的功能 ····· 13
1.4.5　离线优先 ····· 14
1.5　Electron 的工作原理 ····· 14
1.5.1　主进程 ····· 15
1.5.2　渲染器进程 ····· 15
1.6　对比 Electron 与 NW.js ····· 16
1.7　本章小结 ····· 17

第 2 章　创建第一个 Electron 应用 ····· 19
2.1　创建一个书签列表应用 ····· 20
2.1.1　组织 Electron 应用的结构 ····· 21
2.1.2　package.json ····· 22
2.1.3　在项目中下载并安装 Electron ····· 23
2.2　使用主进程 ····· 25
2.3　创建一个渲染器进程 ····· 26
2.3.1　从渲染器进程加载代码 ····· 30
2.3.2　在渲染器进程中引入文件 ····· 31
2.3.3　在渲染器进程中添加样式 ····· 32
2.4　实现 UI 界面 ····· 33
2.4.1　在 Electron 中发出跨域请求 ····· 35
2.4.2　解析响应文本 ····· 36
2.4.3　使用 Web Storage API 存储从响应中得到的信息 ····· 38
2.4.4　显示请求的结果 ····· 39
2.4.5　预防错误 ····· 43
2.4.6　一个不期而至的 bug ····· 45
2.5　本章小结 ····· 49

第Ⅱ部分　使用 Electron 创建跨平台应用

第 3 章　创建一个笔记应用 ····· 53
3.1　定义应用的目标 ····· 54
3.2　打下基础 ····· 55
3.3　初始化并启动应用 ····· 56
3.3.1　实现 UI 界面 ····· 57
3.3.2　优雅地显示浏览器窗口 ····· 63
3.4　实现基本功能 ····· 64
3.5　调试 Electron 应用 ····· 67
3.5.1　调试渲染器进程 ····· 67

3.5.2 调试主进程 …… 68
3.5.3 使用 Visual Studio Code 调试主进程 …… 69
3.6 本章小结 …… 72

第 4 章 使用原生文件对话框与实现跨进程通信 …… 73
4.1 触发原生文件对话框 …… 74
4.2 使用 Node 读文件内容 …… 77
4.2.1 限定允许打开的文件类型 …… 78
4.2.2 在 macOS 系统中实现工作表对话框 …… 81
4.3 实现跨进程通信 …… 82
4.4 使用跨进程通信调用文件打开功能 …… 87
4.4.1 理解 CommonJS 模块系统 …… 88
4.4.2 引入其他进程的功能 …… 88
4.5 从主进程向渲染器进程发送内容 …… 90
4.6 本章小结 …… 95

第 5 章 创建多窗口应用 …… 97
5.1 创建和管理多个窗口 …… 98
5.1.1 主进程与多个窗口之间的通信 …… 99
5.1.2 将指向当前窗口的引用传给主进程 …… 101
5.2 改进新建窗口的用户体验 …… 103
5.3 与 macOS 集成 …… 105
5.4 本章小结 …… 108

第 6 章 操作文件 …… 111
6.1 跟踪当前打开的文件 …… 113
6.1.1 使用当前文件的名称更新窗口标题 …… 114
6.1.2 检测是否修改过当前文件 …… 116
6.1.3 启用 UI 界面上的 Save File 和 Revert 按钮 …… 118
6.1.4 更新 macOS 系统的展示文件 …… 119
6.2 跟踪最近打开的文件 …… 120
6.3 保存文件 …… 122
6.3.1 导出渲染的 HTML 内容 …… 123
6.3.2 常用路径 …… 124
6.3.3 从渲染器进程保存文件 …… 125
6.3.4 保存当前文件 …… 125
6.3.5 回滚文件 …… 127
6.4 通过拖曳打开文件 …… 127
6.4.1 忽略无关区域的拖曳操作 …… 127
6.4.2 提供可视化反馈 …… 128
6.4.3 打开拖曳过来的文件 …… 131
6.5 监控文件的变动 …… 131
6.6 丢弃未保存的修改前提示用户 …… 134
6.7 本章小结 …… 138

第 7 章 创建应用菜单和上下文菜单 …… 141
7.1 替换并复制默认菜单 …… 143
7.1.1 macOS 系统的 Edit 菜单消失之谜 …… 144
7.1.2 替换 Electron 默认菜单的隐形代价 …… 146

7.1.3 实现 Edit 和 Window 菜单……147
7.1.4 定义菜单项的 role 属性和键盘快捷键……148
7.1.5 恢复 macOS 系统的应用菜单……149
7.1.6 添加 Help 菜单……153
7.2 在菜单中添加应用特有的功能……155
7.3 创建上下文菜单……160
7.4 本章小结……162

第 8 章 与操作系统更深入地集成以及动态启用菜单项……163
8.1 在渲染器进程(UI 界面)中使用 shell 模块……164
8.2 在应用菜单中使用 shell 模块……167
8.3 在上下文菜单中使用 shell 模块……169
8.3.1 决定将功能放在菜单中还是 UI 界面上……170
8.3.2 决定将功能放在应用菜单还是上下文菜单中……171
8.4 在适当的时候禁用菜单项……171
8.4.1 动态启用和禁用上下文菜单中的菜单项……172
8.4.2 动态启用和禁用应用菜单中的菜单项……175
8.5 本章小结……181

第 9 章 介绍 tray 模块……183
9.1 开始创建 Clipmaster……184
9.2 使用 tray 模块创建一个应用……185
9.2.1 为 macOS 和 Windows 选择不同的图标……187
9.2.2 支持 macOS 的深色模式……188
9.2.3 从剪贴板读取内容并保存剪贴项……189
9.3 读写剪贴板的内容……191
9.3.1 写入剪贴板……193
9.3.2 处理极端场景……195
9.4 注册全局快捷键……198
9.5 显示通知……201
9.6 在 macOS 系统中单击菜单栏图标时，切换显示的图标……204
9.7 完整的代码……205
9.8 本章小结……208

第 10 章 在应用中使用 menubar 库……209
10.1 使用 menubar 开始创建应用……210
10.2 向 UI 界面添加剪贴项……214
10.3 在应用中操作剪贴项……216
10.3.1 使用事件代理避免内存泄漏……216
10.3.2 删除一个剪贴项……217
10.3.3 将数据写入剪贴板……219
10.4 发布剪贴项……220
10.5 显示通知和注册全局快捷键……223
10.5.1 注册全局快捷键……224
10.5.2 处理从未打开过窗口的极端场景……226
10.6 添加第二个菜单……227

10.7 本章小结……229

第 11 章 使用转译器和框架……231
11.1 介绍 electron-compile ……233
11.2 打造应用的基础……234
11.3 在 React 中创建 UI 界面 ……241
11.3.1 Application 组件……241
11.3.2 显示物品列表 ……244
11.4 添加新的物品……248
11.5 实时重新加载与模块热加载……252
11.5.1 启用实时重新加载……252
11.5.2 实现模块热加载 ……253
11.6 本章小结……256

第 12 章 保存用户数据以及使用 Node.js 原生模块……259
12.1 在 SQLite 数据库中存储数据……260
12.1.1 使用 electron-rebuild 确保编译出正确的版本……261
12.1.2 使用 SQLite 和 Knex.js ……262
12.1.3 将数据库功能挂载到 React 应用 ……263
12.1.4 从数据库获取所有数据项 ……265
12.1.5 向数据库中添加数据项 ……266
12.1.6 更新数据库中的数据项 ……268
12.1.7 删除数据项 ……270
12.1.8 将数据库存储在正确的地方……275
12.2 IndexedDB ……276
12.2.1 在 IndexedDB 中创建仓库 ……277
12.2.2 从 IndexedDB 获取数据 ……278
12.2.3 向 IndexedDB 写入数据 ……279
12.2.4 将数据库操作连接到 UI 界面 ……282
12.3 本章小结……284

第 13 章 使用 Spectron 测试应用……285
13.1 介绍 Spectron ……287
13.2 使用 Spectron 和 WebdriverIO ……289
13.3 设置 Spectron 和测试运行器 ……290
13.4 使用 Spectron 编写异步测试……292
13.4.1 等待窗口加载……293
13.4.2 测试 Electron Browser Window API……294
13.4.3 使用 Spectron 遍历和测试 DOM……294
13.4.4 使用 Spectron 控制 Electron API……297
13.5 本章小结……298

第III部分 部署 Electron 应用

第 14 章 构建并部署应用 ……301
14.1 介绍 Electron Packager……302
14.1.1 设置 Electron Packager…302
14.1.2 配置输出目录……304
14.1.3 配置应用的名称和版本……304

14.1.4 更新应用图标……305
14.1.5 针对多个操作系统进行构建……306
14.2 使用 asar 档案文件……307
14.3 Electron Forge……310
14.3.1 将一个 Electron 应用导入 Electron Forge……311
14.3.2 使用 Electron Forge 构建应用……312
14.4 本章小结……313

第 15 章 发布和更新应用……315
15.1 收集崩溃报告……315
15.1.1 设置崩溃报告……316
15.1.2 设置一台接收崩溃报告的服务器……319
15.1.3 报告未捕获异常……322
15.2 应用签名……324
15.2.1 签署 macOS 应用……325
15.2.2 在 Windows 系统中创建安装程序和进行代码签名……328
15.3 自动更新应用……331
15.3.1 在 Electron 中设置自动升级……332
15.3.2 搭建一台自动更新服务器……334
15.4 本章小结……337

第 16 章 通过 Mac App Store 分发应用……339
16.1 将应用提交到 Mac App Store……339
16.1.1 签署应用……340
16.1.2 向 Mac App Store 注册应用……345
16.1.3 将应用添加到 iTunes Connect……346
16.1.4 为 Mac App Store 打包应用……348
16.1.5 配置应用类别……352
16.1.6 注册应用以打开一种文件类型……352
16.2 验证和上传应用……354
16.3 收尾事项……355
16.4 本章小结……355

附录 Fire Sale 和 Clipmaster 9000 的源代码……357

第 I 部分

Electron 入门知识

最近你用过 Slack[1]吗？如果没有，或许你曾使用 Atom 或 Visual Studio Code 编写过一些代码，或使用 WhatsApp[2]桌面应用给朋友发送过消息。如果被我说中，就表明你已用过 Electron 应用了。那么，到底什么是 Electron？对于这个问题，可以用一句话给出简短的回答：Electron 是一个使用 Web 技术创建桌面应用的平台，用它创建的桌面应用可以运行在 macOS、Windows、Linux 等多种操作系统之上。Electron 整合了 Node.js 和 Chromium 这两项技术，后者是 Google Chrome 浏览器的开源内核版本。对于“什么是 Electron”这个问题，更长、更详细的答案将在第 1 章给出，而本书的全部内容实际上也围绕着这个问题展开。

如果你身处一支小型开发团队中，团队的任务是为多个操作系统平台创建桌面应用，Electron 将是用来实现构想的不错选择。选择 Electron，既不必同时管理两三个不同的代码库版本，也不用在两三个不同平台上调试排错，更无须把同一个功能在不同平台上重复地编写两三次。如果你是一个 Node.js 开发者，想要将命令行程序介绍给更广泛的受众，Electron 可以帮助你轻松地为应用创建图形用户界面(GUI)，而不必为此专门学习一门全新的技术。如果你是一个习惯用自己的方案去解决问题的 Web 开发者，Electron 能让 JavaScript 代码突破仅允许运行于浏览器沙盒之中的限制，获得访问各个计算机部件的能力。

根据我的经验，学习 Electron 是一个效果立竿见影，同时又路途漫长的过程。只要刚入门，你就可以迅速创建一个简单的 Electron 程序，启动它，然后高兴地在 macOS 系统的 Dock[3]中或 Windows 系统的任务栏上看到它启动后的图标，多加一点代码，立

1 译者注：Slack 是一个知名的企业聊天群组工具，集合了聊天、文档整合、搜索等多种功能，主要为企业团队成员之间提供协作服务。

2 译者注：WhatsApp 是一个在欧美非常流行的聊天社交工具。

3 译者注：Dock 指的是 macOS 操作系统桌面下方类似 Windows 任务栏的一块区域，其中会显示正在运行的应用。

刻又能看到使用 JavaScript 代码打开的原生操作系统文件对话框。随着对 Electron 日渐熟悉，你的头脑中会冒出不少应用的好点子，这些点子单凭浏览器或 Node.js 无法实现，这将是一种全新的、通过其他方式都无法创建的新型应用。我希望本书中的应用示例不仅仅提供技术上的指导，还能在你成为一名桌面应用开发者的道路上提供些许灵感。

在第 I 部分，我们将弄清楚什么是 Electron，并认识一些正在使用 Electron 的重量级玩家。在第 1 章我会详加说明 Electron 与浏览器应用的不同之处。在第 2 章，我会创建一个简单的 Electron 应用，让你意识到使用 Electron 创建桌面应用是何等简单而有趣。

第1章

介绍 Electron

本章内容：

- 什么是 Electron
- Electron 基于哪些技术构建而成
- Electron 与传统 Web 应用有何区别
- Electron 应用的结构
- 在生产环境中使用 Electron 创建真实的应用

如今 Web 已经无所不在。它是一个无与伦比的平台，人们在这个平台上创建出了各种协作式应用，供运行着各式各样操作系统的设备访问。但是，许多类型的应用都无法在浏览器环境中创建出来。Web 应用不能访问文件系统。它们无法执行非 JavaScript 代码。它们不能调用许多操作系统级别的 API，但调用这些 API 对于桌面应用而言轻而易举。网络不稳定时，大部分 Web 应用将不可用。

长久以来，Web 开发者如果要创建桌面应用，需要去学习一套完全不同的开发技术。大部分人都没有太多的时间和精力去学习那些新的语言和框架。有了 Electron，Web 开发者就可以使用已经掌握的技能去开发一个桌面应用，让它具备许多只有原生桌面应用才具备的功能。

1.1 什么是 Electron

Electron 是一个可以让你使用 HTML5、CSS 和 JavaScript 创建桌面应用的运行时(runtime)，是由名叫 Cheng Zhao 的 GitHub 工程师创建的一个开源项目。以前这个项

目被称为 Atom Shell，Atom 是 GitHub 使用 Web 技术创建的一个跨平台文本编辑器，而 Electron 则是 Atom 的基石。

你也许听说或曾经使用过 Apache Cordova 或 Adobe PhoneGap，它们能将 Web 应用封装到原生 shell 里面，然后作为手机 App 部署到 iOS、Android 和 Windows Phone。你可以将 Electron 想象成类似的工具，只不过 Electron 是将 Web 应用封装成桌面应用。

Electron 可以让你使用已经掌握的 Web 技术创建桌面应用。在本书中，你将学习如何创建可以调用 Windows/macOS/Linux 操作系统 API 的桌面应用。

Electron 将 Chromium Content Module 和 Node.js 运行时整合在了一起。开发者利用 Electron，可以通过 Web 页面创建 GUI 界面，同时通过与具体操作系统无关的 API，调用 Windows/macOS/Linux 原生操作系统的功能。

Chromium 和 Node 这两个应用平台凭借各自的能力，在各自领域都拥有广泛的使用者，也各自诞生了许多应用。Electron 将这两个平台合二为一，让开发者可以使用 JavaScript 语言创建出一种全新的应用程序。在所有浏览器中能实现的功能，在 Electron 中就可以实现；所有在 Node 中能实现的功能，在 Electron 中也同样可以实现。

使用 Electron 让人兴奋之处在于，你可以同时使用两个平台的技术。原本在任何一个平台上都不可能单独创建出来的应用，现在集两个平台之力，终于变得可能。本书始终致力于向你展示这些无穷的可能性。Electron 不但可以用来创建类似原生桌面应用模样的 Web 应用，还可以用来为原本只有命令行界面的 Node 应用创建 GUI 界面，如图 1.1 所示。

举个例子，假设你想要创建一个查看和编辑计算机文件夹中图片文件的应用。传统的浏览器应用无法访问文件系统，它们既无法访问图片文件所在的目录，也无法加载目录中的图片，更无法将你在应用中对图片所做的修改再保存回文件。你可以使用 Node 来实现所有这些功能，但是 Node 应用没有 GUI 界面，这就让普通用户使用这个应用的难度大大提高。通过使用 Electron，将浏览器环境和 Node 整合到一起，你就可以创建一个带有 GUI 界面的应用，并在界面上实现上述所有功能，如图 1.2 所示。

Electron 并不复杂，它实际上相当简单。就如同可以在命令行中使用 Node，你同样可以使用 Electron 命令行工具运行 Electron 应用。要创建一个 Electron 应用，无须先去学习一大堆的东西，而是可以用任何想要的方式来组织应用。当然，最好还是遵循本书提供的技巧与最佳实践。

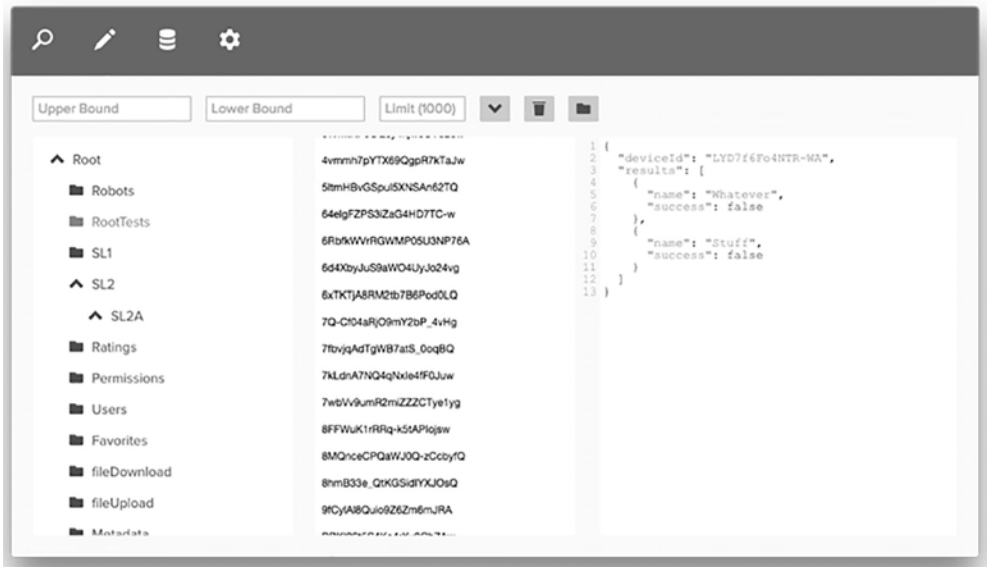

图 1.1　LevelUI 是一个使用 Electron 创建的应用，用来管理 Node 的 LevelUp 数据库。在传统的浏览器中，是无法创建出这个应用的，因为浏览器无法访问位于用户计算机上的本地数据库。由于 LevelUI 是一个已编译的 C++模块，因此浏览器中的代码也无法使用它，只有运行在 Node 中的代码才能调用这个 C++模块

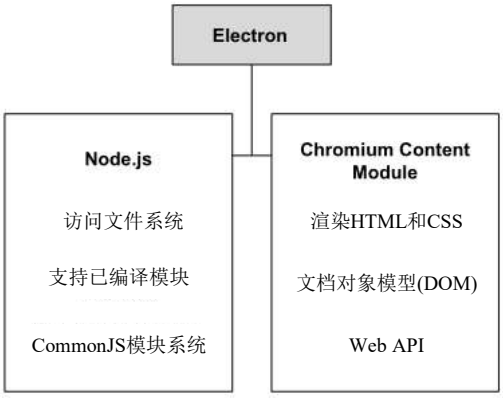

图 1.2　Electron 整合了 Chromium 的核心 Web 浏览组件和 Node 的底层系统访问组件

1.1.1　什么是 Chromium Content Module

Chromium 是 Google Chrome 浏览器的开源版本。它与 Chrome 共用大部分源代码，所以除了少部分功能以及许可协议不同之外，它与 Chrome 几乎完全相同。

Chromium 的核心组件是 Content Module，它除了为 Chromium 提供在单独进程中渲染 Web 页面和使用 GPU 加速渲染的功能之外，还包含了 Blink 渲染引擎和 V8 JavaScript 引擎。Content Module 所实现的是一个浏览器必须具备的核心功能。它从一台 Web 服务器上获取 HTML 内容，在浏览器中渲染这些 HTML 元素，加载 HTML 所引用的 CSS 和 JavaScript，根据 CSS 样式正确地装饰页面元素，并执行 JavaScript 代码。

理解 Content Module 的最简单方式，是看看那些不由它负责实现的功能：不负责支持 Chrome 插件；不负责将书签和访问历史记录同步到 Google 的云服务；不负责加密并保存浏览网页时输入的密码，也不负责在用户下次访问同一个页面时将这些密码自动填入页面；不负责检测页面是否包含另外一种语言的文字，也不负责调用 Google 翻译服务，将这些文字翻译成用户能够理解的语言。Content Module 只负责处理那些与渲染 HTML/CSS/JavaScript 有关的核心工作。

1.1.2 什么是 Node.js

JavaScript 在诞生的头十五年中，都只是一门仅能用在 Web 浏览器中的语言。虽然历史上曾经有人尝试想办法在服务器上运行 JavaScript，但是那些项目从未掀起过任何波澜。Node.js 项目于 2009 年发布，它是一个开源项目，提供了一个支持使用 JavaScript 开发服务器端应用的跨平台运行时。它使用开源的 Google V8 引擎执行 JavaScript 代码，同时添加了诸多 API 接口，让 JavaScript 代码可以访问文件系统、创建服务器或从外部模块中载入代码。

在过去短短几年，Node 获得了巨大的关注，并被广泛地用于各种目的，从编写 Web 服务器到控制机器人，再到创建桌面应用(你也许猜到了)。Node 包含一个名为 npm 的程序包管理器，有了 npm，开发者才可以自由自在地引用 registry 中多达 25 万个的第三方程序包。

1.2 哪些人在使用 Electron

诸多商业公司，无论大小，都在使用 Electron 创建桌面应用。上文中提到过，Electron 被开发出来，原本用作 GitHub Atom 文本编辑器的底层框架。Atom 既然是文本编辑器，就需要能够访问文件系统。其他一些编写文本编辑程序的公司，也在选用 Electron 作为他们产品的底层框架。Nuclide(一个由 Facebook 发布的程序包)就是基于 Atom 创建的，它把 Atom 从一个普通的文本编辑器转换为功能完整、对 React Native/Hack/Flow

项目具有最好支持的集成开发环境(IDE)。微软也使用 Electron 开发跨平台的 Visual Studio Code 编辑器，使之能运行于 macOS/Windows/Linux 之上。

除了文本编辑器之外，Electron 当然还可以用来创建其他类型的桌面应用。Slack(一个流行的通信应用)也使用 Electron 打造它的 Windows 和 Linux 版本。Nylas 正是因为使用了 Electron，才使得 N1 电子邮件客户端程序在所有主流平台上都让人赏心悦目。Nylas 甚至让它的电子邮件客户端能支持 JavaScript 插件，第三方开发者可以通过开发插件，为客户端添加新功能，扩展原有的用户界面。

Particle 这家公司专门提供创建定制硬件的开发套件，它使用 Electron 创建 IDE。用户可以在 IDE 中编写代码，然后通过移动网络或 Wi-Fi 网络将代码部署到硬件设备上。Mapbox Studio 是一款用来设计定制地图的软件，地图设计师可以在软件中导入存储在本地计算机上的数据，并直接在本地计算机上处理数据，无须再将数据通过互联网上传到 Mapbox 的服务器，由此带来的是更快的处理速度、更好的用户体验。

Dat 是一款用来对分布式数据进行共享、同步和版本化管理的开源工具。项目的开发团队由三名 Web 开发者组成，是外来资金赞助的项目。尽管人员不多，但 Dat 团队使用 Electron 成功为项目发布了一个桌面应用。2015 年，一家软件咨询公司 Wiredcraft，使用 Electron 创建了一个可以离线工作的 Windows 程序，在缅甸收集和更正投票者登记信息。这个数据收集程序需要支持离线存储收集到的数据，然后在网络处于在线状态时，将数据发布出去。为避免在一个陌生的领域重新学习一门新技术，Wiredcraft 抛弃了 C++，选择了 Electron，让公司可以使用熟悉的 HTML/CSS/JavsScript 实现自己的目标。

基于 Electron 创建的 Brave 是一种专注于速度和安全性的全新浏览器，由 JavaScript 语言的发明者 Brendan Eich 亲手构建，如图 1.3 所示。没错，你甚至可以使用 Web 技术来构建 Web 浏览器。

每天都有基于 Electron 的新项目发布，无论是公司还是个人开发者，都看到了 Electron 的独特价值：能够在利用 Web 天然平台无关性的同时，创建桌面应用类型的产品。完成本书的学习之后，你也能利用已掌握的 Web 开发技能，创建出无法在传统浏览器环境中实现的新型应用。

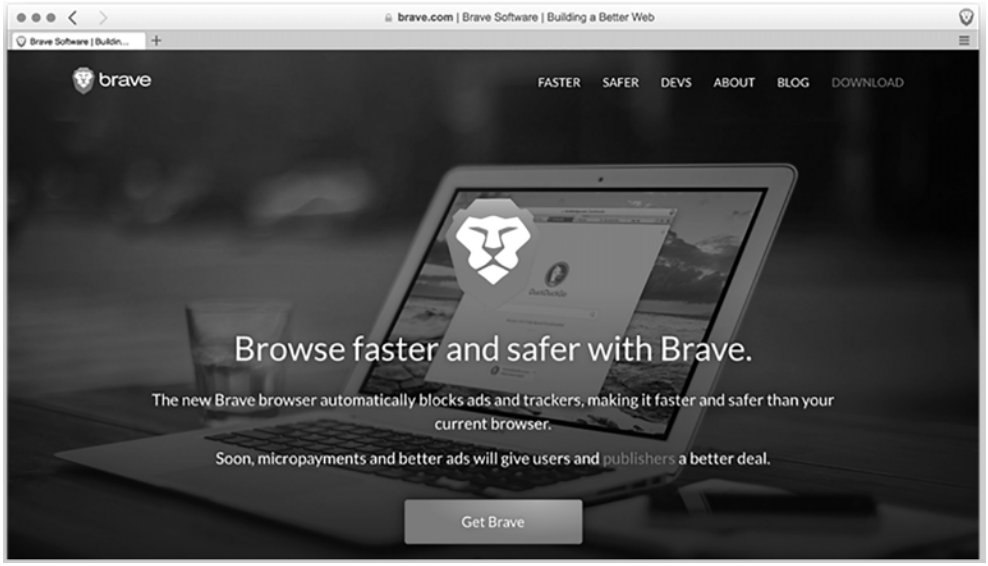

图 1.3 Brave 是一种基于 Electron 构建的全新 Web 浏览器

1.3 阅读本书之前，需要知道些什么

本书所面向的读者，是希望将现有技能用于创建桌面应用的 Web 开发者。阅读本书之前，不需要具备任何创建桌面应用的经验。

你不一定要对编写 JavaScript/HTML/CSS 达到非常精通的程度，但至少应相当熟练。我不会在本书中讲述 JavaScript 语言的变量或条件表达式之类的基础知识，如果你已经熟悉 JavaScript 的基本语法特性，就大致已经具备了阅读本书的条件。如果你已经熟悉 Node.js 的一些常见用法与模式，例如 Node.js 模块系统的工作原理，对阅读本书也大有帮助。我们在本书中用到这些概念时，会再对它们详加讨论。

1.4 为何要使用 Electron

当你创建 Web 应用时，技术的选型必须偏保守，编写代码时也要小心谨慎。因为与创建服务器端应用不同，你编写的代码会运行在每个用户的计算机上。

Web 应用的用户既有可能使用 Chrome 或 Firefox 这些现代浏览器的最新版本，也有可能还在使用 Internet Explorer 的老版本。由于根本说不上来代码到底会被哪种浏览器执行，因此你的代码必须为各种可能出现的运行环境做好准备。

通常来说，最好让代码使用尽可能少的语法特性和浏览器功能，以确保在现今使

用的所有浏览器版本上，它们都能得到最广泛的支持。就算针对某个具体问题有一个更好、更高效，或者整体上更优雅的方案，说不定你也不能使用它。当你决定在代码中使用一个新的浏览器功能时，一般也需要实现一个能在旧版本浏览器上运行的后备方案，这个后备方案要包括添加浏览器检测代码、渐进式优化等大量工作，这些工作都会在项目开发流程中增添数不清的障碍。

当使用 Electron 创建应用时，会把一个特定版本的 Chromium 和 Node.js 打包进应用，所以，你可以放心大胆地在代码中使用该特定版本的 Chromium 和 Node.js 提供的功能。你完全不需要担心代码是不是使用了不被其他浏览器支持的功能。举例来说，如果打包到应用的 Chromium 版本支持 Service Worker API，你就一定可以在程序代码中使用这个 API，如图 1.4 所示。

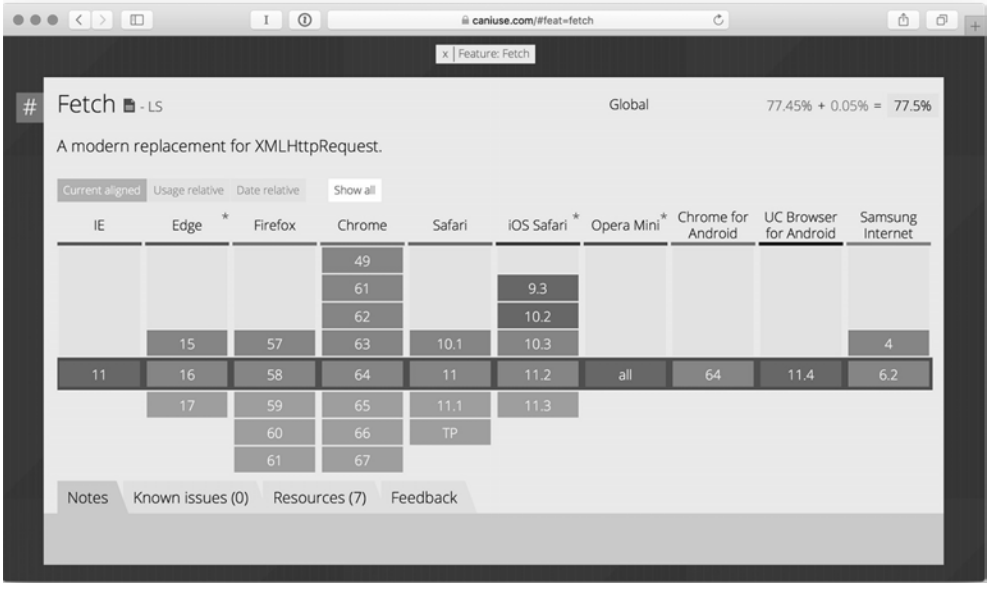

图 1.4　在一个基于浏览器运行的 Web 应用中，考虑到诸多浏览器对 Fetch API 的支持程度各不相同，在代码中使用这个 API 并非明智之举。但是在 Electron 应用中，由于打包了能够完整支持 Fetch API 的 Chromium 最新稳定版本，因此对它的使用也就不再是一个问题了

由于 Electron 包含了一个相对来说版本很新的 Chromium，因此你在开发过程中可以使用最新的 Web 平台技术。通常来说，Chromium 每 6 周发布一个新的稳定版本，Electron 中包含的 Chromium 版本只比最新稳定版本延后 1 到 2 周。Electron 也通常会在 Node.js 的最新版本发布 1 个月之后，将其包含进来，以确保能够使用最新版本的 V8 引擎。由于 Electron 通过包含 Chromium 已经引入了 V8 的最新版本，因此在 Node 最新版本发布后可以稍微多等一段时间，以便 Node 新版本带来的一些 bug 被 Node

团队修复。

1.4.1 重用现有技能

如果你有和我一样的技术背景，那就意味着相对于创建桌面应用而言，你也对创建 Web 应用具备更多的经验。你非常想要将创建桌面应用的能力加入个人“技能库”，但是又没有那么多时间和精力，去掌握一门新的编程语言和一个新框架。

学习新的编程语言和框架，是一项不容忽视的投资。作为一名 Web 开发者，虽然意味着需要处理各种不同浏览器、各种不同屏幕尺寸的兼容性问题，但是你已经习惯了让创建的 Web 应用为所有用户提供一致的功能与体验。但是当你打算进入传统桌面应用领域时，要学习的可就不止一门语言和一个框架了。如果桌面应用需要支持 Windows/macOS/Linux 这三种不同的操作系统，那么你至少需要学习三门语言和三个框架。

个人开发者和小型团队也许无法承担为每个桌面操作系统雇用一名单独开发者的成本，在这种情况下，选择使用 Electron 创建桌面应用，就可以将一个不可能完成的任务变为可能。Electron 让开发者能够使用已经具备的技能，在所有主流桌面平台上发布应用。有了 Electron，为了支持多个桌面操作系统所花费的精力，甚至比开发者在开发 Web 应用时处理浏览器兼容性问题所花费的精力更少。

1.4.2 访问原生操作系统 API

Electron 应用和其他桌面应用类似。与其他原生的应用一样，Electron 应用也被部署到文件系统中，在运行时，它们也会出现在 macOS 的 Dock 区域，或在 Windows 和 Linux 的任务栏中。Electron 应用可以调用打开文件和保存文件原生对话框。这些对话框支持使用参数对它们进行配置，以限定用户在同一时间是否只允许操作系统选择具备某特定扩展名的单个文件，还是允许选择整个目录或多个文件。你可将文件拖放到 Electron 应用中，触发不同的操作。

同时，Electron 应用可以像其他任何应用那样，对应用菜单进行定制，如图 1.5 所示。它们可以创建自定义的上下文菜单，在用户单击鼠标右键时将菜单显示出来。使用 Chromium 的通知 API，它们可以触发系统级别的通知。它们还可以从系统剪贴板读取数据，也可以将文本、图片和其他媒体数据写入系统剪贴板。

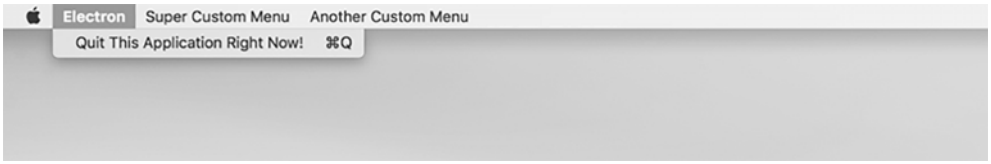

图 1.5　Electron 允许开发者定制应用菜单

与传统的 Web 应用不同，Electron 应用并不受限于浏览器的功能。你可以创建出能够留存于菜单栏(menu bar)或系统托盘(system tray)中的应用，如图 1.6 所示。你甚至能够注册全局快捷键，让用户在操作系统中只按下特定的键，就能启动这些 Electron 应用，或触发应用中的某个功能。

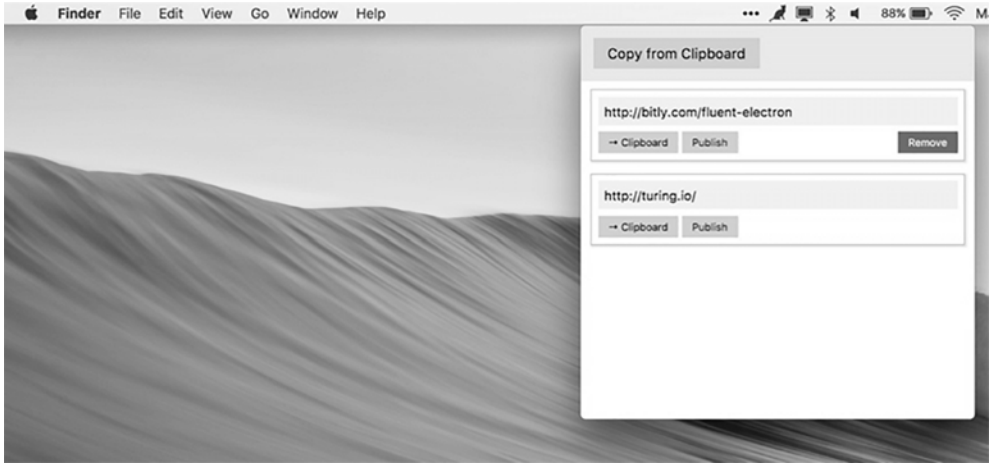

图 1.6　你可以创建出能够留存于菜单栏或系统托盘中的应用

Electron 应用可以访问系统级别的信息，例如当前计算机是否正在依赖电池运行，还是使用了外置电源。如有必要，它们还能让操作系统保持唤醒状态，避免系统进入节能模式。

1.4.3　更高运行权限，更少功能限制

Web 是史上最大的分布式应用平台。它的覆盖范围如此之广，以至于 Web 开发者无论为之付出多大的艰辛，看起来也是值得的。创建 Web 应用，必须精心设计服务器端应用与数以千万计的客户端应用实例之间的通信。客户端代码运行在用户的 Web 浏览器中，与服务器相隔万里。

除非将浏览器中的数据发送回服务器，否则每个浏览器会话中发生的任何操作，都只能作用于当前会话。在后续的浏览器会话中，你必须等待浏览器再将一个请求发送到服务器端，通过相同的用户身份标识，将保存在服务器上的更新数据获取回当前会话；如果客户端和服务器端都实现了 WebSockets 通信，用户数据更新的分发也可以通过 WebSockets 进行。

由于用户亲自下载、安装、运行桌面应用，因此对于桌面应用能做什么、不能做什么的限制会很少，你能够随心所欲地使用大量功能。然而，当用户浏览 Web 页面时，就不会对这个 Web 应用那么放心了。用户打开一个 Web 页面，并不意味着要在自己的计算机上“安装”这个页面，所以页面上运行的代码能做什么、不能做什么，都会受到许多限制。

当浏览器访问一个 Web 页面时，它会愉快地下载此 Web 页面上 HTML 代码所引用的资源，以及这些资源各自引用的其他依赖资源，然后开始执行代码。在过去的这些年里，浏览器厂商们为浏览器添加了各种限制，防止它们运行任何恶意代码，以保护用户和互联网上的其他网站。

我不是黑心大叔，但是为了列举下面这个例子，先假设我是。在我举的这个例子里，我运营着一家颇受欢迎的网站，售卖一些手工制品和工艺品。某天，一个竞争网站突然出现，它卖的东西和我差不多，但是价格要优惠不少。虽说现在我的网站暂时还没有受到这个竞争者的影响，但是它的存在已经足够让我晚上睡不着了。

我决定在我的网站页面上添加一些 JavaScript 代码，这些代码每隔几毫秒便会向竞争者的网站发送一个 AJAX 请求。当数以千计的访问者浏览我的网站时，这些代码会下载并运行在他们的浏览器上，迅速冲垮竞争者网站的服务器，让它无法处理用户正常的访问请求。虽说这些代码也会影响我的网站访问者的体验，但为了搞垮竞争者的网站，这是不得不付出的代价。

尽管我的计划异常卑劣，但事实上它并不会奏效。现代浏览器禁止客户端代码向第三方服务器发送请求，除非第三方服务器显式地声明一个策略，许可接收这些请求。

一般来说，绝大多数网站都不会提前声明类似这种策略。如果真的需要向一台第三方服务器发送请求，必须先向页面所属的服务器发送一个请求，让托管页面的服务器向第三方服务器发送请求，然后把请求的结果转发回页面。在上面这个例子中，如果我决定这样做，我的服务器会由于需要转发大量的网络请求而成为瓶颈，这使得这种攻击在技术上变得不可行。就算我还是决定这样去做，用自己的服务器作为攻击的代理，竞争者也很容易将我的服务器 IP 地址从所有访问请求列表中识别出来，然后直接屏蔽掉这个 IP 地址。

除了上面这个例子中讲述的限制以外，浏览器还对客户端代码可以访问哪些资源、可以做哪些操作，都进行了严格的限制。所有这些限制，都是为了给用户营造一个更安全、更可靠、具有更好体验的使用环境。这些限制的存在意义非凡，它们是整个互联网变得如此强大、如此实用的重要原因之一。

当然，所有这些有效而重要的安全限制，反过来也会严重限制使用 Web 技术创建的应用的功能。由于 Electron 应用和其他所有桌面原生应用一样，都是由用户亲自下载和安装，因此 Electron 应用可以自由地访问文件系统，就像其他原生桌面应用或服务器端 Node 程序一样。由于 Electron 应用可以获得与其他任何 Node 程序相同的运行权限和功能，因此 Electron 应用可以自由地向第三方 API 发送请求，而无须通过 Node 服务器中转，如图 1.7 所示。

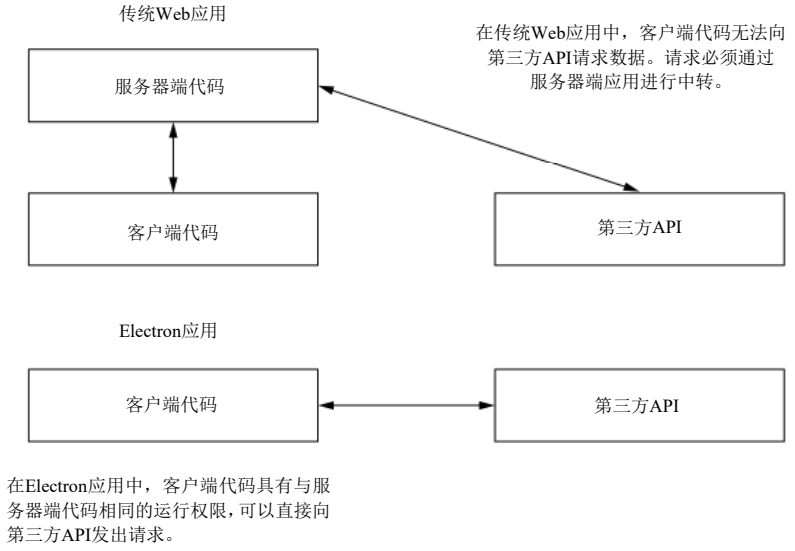

图 1.7　Electron 应用可以使用它们的 Node.js 运行时，向第三方 API 发送请求

1.4.4　在浏览器环境中使用 Node 的功能

除了具备访问文件系统、启动 Web 服务器的能力以外，Node.js 还使用了一套基于 CommonJS 模块规范的模块系统。Node.js 从诞生之初，就支持将代码分散到不同的模块中，并在每个模块文件中显式包含其依赖的其他模块。

将浏览器使用的 JavaScript 代码打包，从未像现在这么简单过。如果代码不多，可以将它们直接放到<script>和</script>标签的中间。如果代码比较多，可以将它们放到外部 JavaScript 文件中，然后使用<script>标签的 src 参数引用那个外部 JavaScript 文

件。你可以随心所欲地使用尽可能多的外部 JavaScript 文件，但是由于浏览器对每个单独外部文件都需要发出额外的请求，因此这会带来性能上的些许损失。

如果你喜欢，也可以使用 webpack 或 Browserify 等构建工具。但是由于在 Electron 应用中，浏览器代码可以通过 Node 的全局属性(例如 require、module 和 exports)直接使用 CommonJS 模块系统，因此也就无须再给 Electron 应用添加额外的构建操作了。

在 Electron 应用的浏览器环境中，可以访问所有的 Node API。通过使用 Node 模块系统，浏览器代码可以使用 Node 原生扩展提供的已编译模块，访问文件系统，并实现其他大量通常无法在普通浏览器环境中实现的功能。

1.4.5 离线优先

任何曾经带着电脑搭乘洲际航班的人都可以证明，在没有互联网连接的情况下，大部分基于浏览器的 Web 应用表现得都不太好。即使那些使用了任何流行客户端框架(比如 Ember/React/Angular)的高级 Web 应用，通常也需要连接到一台远端的服务器，以下载这些框架的各种资源。

Electron 应用在安装之时，就已经将自己下载到了用户的电脑上。在启动的时候，Electron 应用通常加载一个存储在本地的 HTML 文件。启动后，如果网络连接可用，它可以再从远端服务器上获取数据和资源。Electron 甚至内置了一个检测网络连接是否可用的 API。要使用 Electron 创建一个离线应用，简直轻而易举，毫不费力。实际上，Electron 应用启动后的默认状态就是离线，除非它的代码主动向互联网发起请求。只要创建的不是那种根本离不开网络的应用，比如聊天客户端，Electron 应用就能够像其他任何桌面应用一样，对离线场景应付自如。

1.5 Electron 的工作原理

Electron 应用由两种进程组成：主进程以及零个或多个渲染器进程。每种进程都分别在应用中扮演着不同的角色。Electron 运行时内置了辅助应用开发的各种模块。有些模块可以用在这两种进程中，比如读写系统剪贴板的模块；而有些模块，比如访问操作系统 API 的模块，则仅限于在主进程中使用，如图 1.8 所示。

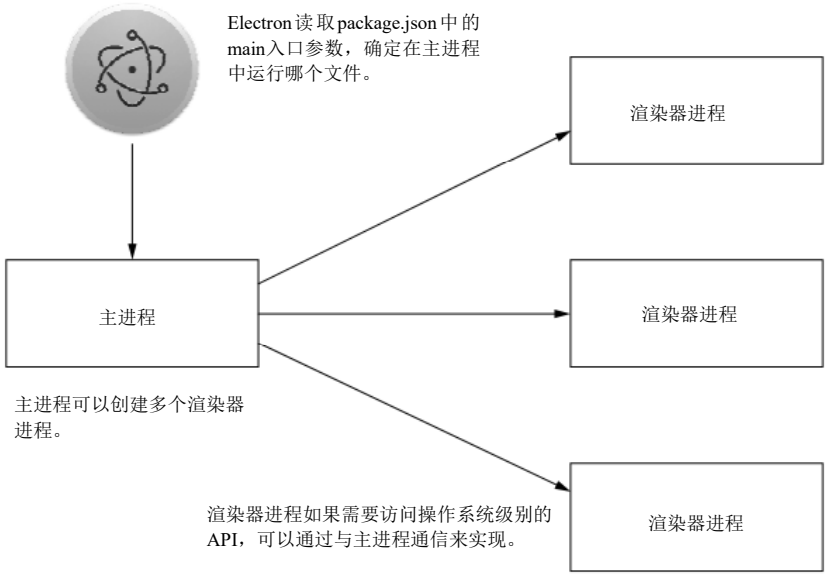

图 1.8　Electron 的多进程架构

当 Electron 启动时，它通过读取项目清单文件 package.json 中的 main 入口参数，确定 Electron 应用的启动文件。启动文件可以任意命名，只要名字正确地设置在 package.json 中即可。Electron 在主进程中运行这个文件。

1.5.1　主进程

主进程承担着几项重要的职能。它可以响应应用的生命周期事件，这些事件包括启动、退出、准备退出、正在切换到后台、正在切换回前台，等等。主进程还负责与原生操作系统 API 进行通信。如果想要显示一个打开或保存文件的对话框，就需要在主进程中实现。

1.5.2　渲染器进程

主进程可以使用 Electron 的 BrowserWindow 模块来创建和销毁渲染器进程。渲染器进程可以加载 Web 页面，向用户显示 GUI 界面。基于 Chromium 多进程架构的设计，每个渲染器进程都运行在各自的单个线程上。这些页面可以加载其他 JavaScript 文件，在各自的渲染器进程中执行代码。与普通 Web 页面不同，在渲染器进程中执行的代码可以访问 Node 的所有 API，利用这个特性，渲染器进程可以使用原生模块，实现与

底层系统的交互。

渲染器进程之间相互隔离，并且也不允许它们直接访问操作系统级别的 API。Electron 实现了一套进程间通信的机制，当渲染器进程需要使用文件打开或关闭对话框，或是需要访问任何操作系统级别的 API 时，它们可以与主进程通信，由主进程实现必要的功能。

1.6 对比 Electron 与 NW.js

Electron 与另外一个名为 NW.js 的项目(之前被称为 node-webkit)颇为类似。它们在很多方面有相同之处。实际上，Electron 的核心开发者在开始参与 Electron 项目之前，也是 NW.js 项目的重要贡献者之一。当然，两者在很多方面也有不同之处，如表 1.1 所示。

表 1.1 Electron 与 NW.js 的一些主要区别

	Electron	NW.js
平台	从近期版本开始被官方支持的 Chromium Content Module	Chromium 的分支版本
进程模型	多进程	共享的 Node 进程
崩溃报告	内置	不包含
自动更新	内置	不包含
支持的 Windows 版本	Windows 7 及后续版本	Windows XP 及后续版本

NW.js 使用 Chromium 的一个分支版本。Electron 则直接使用 Chromium 和 Node.js 的未修改原始版本。这种设计让 Electron 可以更快地跟上 Chromium 和 Node 最新版本的步伐。Electron 还内置了自动下载更新和崩溃报告模块，而 NW.js 则没有。

NW.js 应用在启动时会加载一个 HTML 页面。它的所有浏览器窗口都共享一个相同的 Node 进程，也就是说，如果在应用中打开的窗口超过一个，这些窗口都共享使用同一个 Node 进程。Electron 将 Node 进程与浏览器进程分隔开。在 Electron 应用中，主进程由 Node 启动。主进程可以打开浏览器窗口，每个窗口都拥有自己单独的进程，本书将这种进程称为渲染器进程。Electron 提供了一套进程间通信 API，用于主进程与渲染器进程之间的数据通信。

NW.js 支持 Windows XP 和 Windows Vista，所以在向后兼容性很重要的场景中，它可能是更好的选择。Electron 最低只支持 Windows 7 操作系统。由于 Chromium 的

FFmpeg 库是一个静态链接库，因此 Electron 默认支持更多的 codecs 编码格式，这个优势使得它对于着重多媒体特性的应用而言，通常会是更好的选择。使用 NW.js 的话，开发者需要手工链接 FFmpeg 库。

1.7　本章小结

- Electron 是一个使用 Web 技术创建桌面应用的运行时。
- Electron 项目始于 GitHub 为 Atom 文本编辑器创建的一个基础框架。
- Electron 整合了 Chromium Content Module 和 Node。前者是 Chrome 浏览器的“素颜”版本。
- Electron 通过对 Chromium Content Module 和 Node 的整合，让开发者能够创建出可以同时利用两者能力的应用。Electron 应用既能访问文件系统、使用已编译模块，又能渲染 UI 界面、使用 Web API。
- Electron 被用于创建各种大大小小的应用，比如 Atom、微软的 Visual Studio Code、Slack 等。
- 如果个人开发者或小型团队想要开发面向多个平台的应用，又不想学习好几门不同的语言和平台框架，Electron 是最佳选择。
- Electron 让 Web 开发者可以使用他们已经掌握的技能，创建在浏览器环境中不可能实现的桌面应用。
- Electron 包含 Chromium 和 Node 的最新版本，让开发者可以使用 Web 平台上最新、最好的特性。
- Electron 应用可以访问操作系统 API，如应用菜单和上下文菜单、打开和保存文件对话框、电池状态和电源设定，等。
- 相比基于浏览器的 Web 应用而言，Electron 应用具有更高的运行权限和更少的功能限制。
- Electron 应用由主进程以及零个或多个渲染器进程组成。
- 主进程负责处理与操作系统的交互、管理应用的生命周期和创建渲染器进程。
- 渲染器进程显示 UI 界面，响应用户的操作。
- Electron 和 NW.js 的不同之处在于，前者使用 Chromium 官方支持的 Content Module，而后者使用 Chromium 的一个分支版本。

第2章

创建第一个 Electron 应用

本章内容：

- 组织并创建第一个 Electron 应用
- 生成 Electron 应用所需的 package.json 文件
- 在项目中包含特定平台的 Electron 预编译版本
- 配置 package.json，启动主进程
- 在主进程中创建渲染器进程
- 利用 Electron 的沙盒机制，实现通常无法在浏览器中实现的功能
- 通过 Electron 内置模块规避一些常见问题

通过第 1 章，我们从高处俯瞰了 Electron 的全貌。不过，既然本书的书名叫做《Electron 跨平台开发实战》，那总得“实战”一下。在本章，我们通过从头到尾地创建一个管理书签列表的应用来学习怎么使用 Electron，这个新应用将具备一些只在最新版本浏览器中才能实现的功能。

在第 1 章的概览中，提到过 Electron 是一个类似 Node 的运行时。我并不想更正这个说法，相反，我要进一步强调它。Electron 不是一个框架，既没有为项目提供任何基本结构模板，也不会强制要求开发者必须以某种不容侵犯的规则去组织应用的结构或定义文件的名称。该怎么做，都由开发者自己决定。从积极的角度看，Electron 没有强制开发者遵循任何约定，在开始动手开发一个应用之前，我们无须了解太多概念性的模板文件。