统 API 做了封装，比如系统对话框、系统托盘、系统菜单、剪切板等。开发者基于 Electron 开发应用时，可以直接使用 JavaScript 访问这些 API。

其他 API，诸如网络访问控制、本地文件系统的访问控制等则由 Node.js 提供支持。

两个框架对于开发者来说差别并不是特别大，最主要的差别无过于 Electron 区分主进程和渲染进程。主进程负责创建、管理渲染进程以及控制整个应用的生命周期，渲染进程负责显示界面及控制与用户的交互逻辑。在 Electron 中主进程和渲染进程间通信需要经由 ipcMain 和 ipcRenderer 传递消息来实现。NW.js 则无须关注这些问题，它需要关注的是所有窗口共享同一个 Node.js 环境带来的问题。

NW.js 和 Electron 都是基于 Chromium 和 Node.js 实现的，Chromium 和 Node.js 的应用场景完全不同，它们的底层虽有一部分是相同的，但要想把它们两个整合起来并非易事。

显然 NW.js 和 Electron 都做到了，然而它们底层使用的整合技术却截然不同。NW.js 通过修改源码合并了 Node.js 和 Chromium 的事件循环机制，Electron 则是通过各操作系统的消息循环打通了 Node.js 和 Chromium 的事件循环机制（新版本的 Electron 是通过一个独立的线程完成这项工作的）。

NW.js 的实现方式更直接，但这也导致 Node.js 和 Chromium 耦合性更高。Electron 的实现方式虽然保证了 Node.js 和 Chromium 的松耦合，但也间接地创造出了主进程和渲染进程的概念，给开发人员实现应用带来了一定的困扰。

除此之外，其他方面的比较如表 1-1 所示。

表 1-1　Electron 与 NW.js 部分能力的对比表

能力	Electron	NW.js
崩溃报告	内置	无
自动更新	内置	无
社区活跃度	良好	一般
周边组件	较多，甚至很多是官方团队开发的	一般
开发难度	一般	较低
知名应用	较多	一般
维护人员	较多	一般

可以说 NW.js 和 Electron 各有优劣，我个人认为 Electron 更胜一筹，更值得推荐。