
Electron
101
2020-06-30 智慧检务事业部
周炼赤

目录

1 基础介绍

2 简单实战

3 与 Web 开发的不同


1
基础介绍



现在由 OpenJS 基金会维护的一个开源框架



其中Chromium为其提供强大的UI能力，可以在不考虑兼容性的情况下利用强大的web生态来开发界面；
Nodejs让Electron有了底层的操作能力，比如像文件读写、集成C++等，还能使用大量的npm包；
Native api解决了跨平台问题，首先它提供了统一的原生界面，比如窗口、托盘，其次是系统能力，比如Notification，最后是应用的基础能力，比如软件更新、崩溃监控等。



Electron 的历史和演变，要从浏览器说起，然后是 nodejs，最后才是 electron 本身。

1989 年，英国科学家伯纳斯李发明了万维网。

1990 年他写出了世界上第一个浏览器，叫 Nexus，但是它有个很大的缺陷，就是不能显示图片。

1993 年，NSCA（美国国家超级电脑应用中心）研发了第一款能够看到图片的浏览器，Mosaic，马赛克。

1994 年，Mosaic 的核心成员 Marc Andreessen 和吉姆克拉克一起创建了网景公司，他们的愿景是希望打破 Mosaic 的垄断地位，期初他们起了一个名字，叫 Mozilla，后来因为市场的原因还是用了 Netscape，但 useragent 还是用的 mozilla。随着网景浏览器的飞速发展，奠定了其新霸主的地位。

1995 年微软发现浏览器的巨大市场之后，推出了 IE1，也就是这一年，网景公司招来了天才程序员布兰登艾奇，网景高层在发布网景2.0的十天之前才决定要加入一个交互语言，后来为了赶需求，布兰登艾奇用了十天时间创造了 JavaScript。但是微软毕竟是巨头，通过操作系统的垄断和捆绑销售，逐步拿下了市场份额。

1998 年网景被击败。受 Linux 开源影响，网景决定开放源码回击 IE，转型为开源软件，名为 Mozilla，成立了 Mozilla 非营利组织。

2002 年在网景源码基础上派生了火狐，它在安全、插件扩展、开发调试工具上都有着巨大的贡献。

2008 年，Chrome 诞生，这是具有颠覆性意义的，重新设计后的极致简洁，然后引入了多进程带来的稳定，还有带来 10 倍提速的 V8 引擎。对应 Chrome 也有一份开源项目，叫 Chromium。

2009 年，Ryan Dahl 基于 V8 将非阻塞 IO 和 JS 整合，让 JS 延伸到了服务器端。那有了 Chromium、V8、Nodejs 开源之后，终于有人开始尝试 JS 来做桌面端了。

No. 1 基础介绍

什么时候用 Electron ?

- 快速试错
- 开发特定领域的软件
- 同时开发 Web 版 + 桌面端

生产率	256	商业	26
开发者工具	194	教育	20
实用工具	138	图形与设计	17
音乐	57	娱乐	7
社交网络	48	图书	6
照片和视频	46	生活方式	5
游戏类	34	新闻	5
金融	30	医疗类	2

No. 1 基础介绍

Electron 谁在用？



No. 1 基础介绍

学习 Electron 的好处？



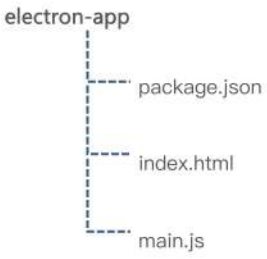
造工具



技术广度

No. 1 基础介绍

Electron 最小组成



package.json (包体描述)

```
{
  "name": "electron-app",
  "version": "1.0.0",
  "main": "main.js"
}
```

index.html (渲染进程)

```
<html>
      <body>hello world!</body>
</html>
```

main.js (主进程)

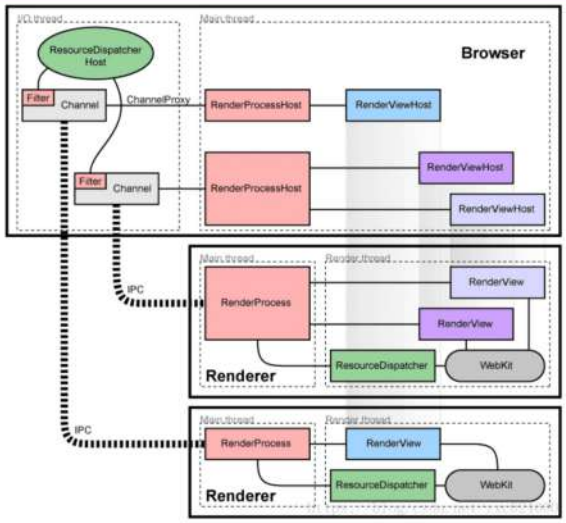
```
const { app, BrowserWindow } = require('electron')
let win
app.on('ready', () => {
  win = new BrowserWindow()
  win.loadFile('index.html')
})
```

- App 生命周期
- 系统事件处理
- 原生 GUI

No. 1 基础介绍

Electron 架构原理

- Chromium 架构



Chromium 本质是 chrome 的开源版，也是一个浏览器，它需要去创建窗口、右键菜单、管理 tab 页面、扩展程序等，而处理这些事项的进程，我们称之为主进程。也即上面的 Browser，而对应每个具体页面的进程，我们称之为渲染进程，即下方的两个 renderer。

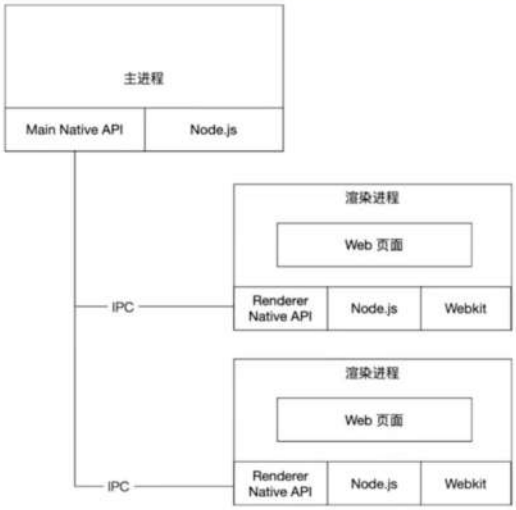
一个浏览器里有一个 Browser 和多个 Renderer，这两类进程需要通信交互才能运转，即通过 IPC 跨进程通信。

主进程中的 RenderProcessHost 和渲染进程中的 RenderProcess 就是专门用来处理 IPC 事件的。而在渲染进程中，我们最熟悉的页面就是在 Renderview 中基于 webkit 的排版展示出来的。最后的 ResourceDIspatcher，是用来处理资源请求的，当我们的页面需要请求资源时，会通过 ResourceDispatcher 创建一个请求 ID，转发到 IPC，在 Browser 进程里处理，然后返回。
这张图带来的信息主要有三点：1、Chromium 是多进程架构，包括一个 Browser 进程和多个 Renderer 进程；2、进程间需要 IPC 通信；3、我们 Web 关注的只是很小的部分。

No. 1 基础介绍

Electron 架构原理

- Electron 架构



由于 electron 使用了 Chromium 来展示 web 页面，所以其多进程架构也会被用到 electron 中，但是和 chromium 不一样的有两点：1、它在各个进程中暴露了一些 native api；2、引入了 nodejs。
于是我们在 electron 中可以使用 Chromium 和 node，比如可以通过 node 去管理窗口，在页面中使用 node 库。
这其实很不容易，因为在主线程中同一个时间下只能运行一个事件循环。



这就是 electron 原理的重点，就是如何整合事件循环。主要的思路有两种，第一种在实现起来，尤其是针对不同系统 GUI 的时候有很多坑，所以后来选择了第二种思路，也就是将 nodejs 集成到 Chromium 里。

No. 1 基础介绍

桌面端技术的选型 —— 为什么要开发桌面端？

- 更快捷的入口
- 离线可用
- 调用系统能力（通知、硬件.....）
- 安全需求
- .....

No. 1 基础介绍

桌面端技术的选型 —— Native（C++/C#/Objective-C）

- 高性能
- 原生体验
- 包体积小
- 门槛高
- 迭代速度慢

No. 1 基础介绍

桌面端技术的选型 —— QT

- 基于 C++
- 跨平台（Mac、Windows、Linux、iOS、Android、嵌入式）
- 高性能
- 媲美原生的体验
- 门槛高
- 迭代速度一般





No. 1 基础介绍

桌面端技术的选型 —— Flutter

- 跨端（iOS、Android、Mac、Windows、Linux、Web）
- PC 端在发展中（Mac -> Linux、Windows）
- 基建少

No. 1 基础介绍

桌面端技术的选型 —— NW.js

- 跨平台（Mac、Windows、Linux），V0.14.7 支持 XP
- 迭代快，Web 技术构建
- 源码加密、支持 Chrome 扩展
- 不错的社区
- 包体积很大
- 性能一般



微信开发者工具

No. 1 基础介绍

桌面端技术的选型 —— Electron

- 跨平台（Mac、Windows、Linux，不支持 XP）
- Web 技术构建
- 活跃的社区（8w+ star）
- 大型应用案例
- 包体积大
- 性能一般

No. 1 基础介绍

桌面端技术的选型 —— 其他

- Garlo（Google 出品，复用安装好的 Chrome，目前并不成熟）
- WPF（基于 .NET，只做 Windows 平台）
- Chromium Embedded Framework（CEF，嵌入式框架）
- PWA（桌面入口、通知能力、离线可用）

No. 1 基础介绍

桌面端技术的选型 —— 各框架对比

	Electron	Native	QT	NW
性能	✓	✓✓✓	✓✓	✓
安装包大小	✓	✓✓✓	✓	✓
原生体验	✓	✓✓✓	✓✓	✓
跨平台	✓✓✓	×	✓✓✓	✓✓✓
开发效率	✓✓✓	✓	✓✓	✓✓✓
人才储备	✓✓✓	✓✓	✓✓	✓✓✓
社区	✓✓✓	✓✓	✓	✓✓
适用场景	• 跨平台应用 • quick delivery • 前端技术栈	• 专业应用 • best performance	• 跨平台应用 • better performance	• 跨平台（包括 XP） • quick delivery • 前端技术栈


2
简单实战

No. 2 简单实战

开发前准备

- 安装 node（推荐 nvm 方式）
- Electron 安装
  - npm install electron —save-dev
  - npm install —arch=ia32 —platform=win32 electron
  - npx electron —v
- 加速技巧
  - ELECTRON_MIRROR=https://cdn.npm.taobao.org/dist/electron/ npm install electron —save-dev

No. 2 简单实战

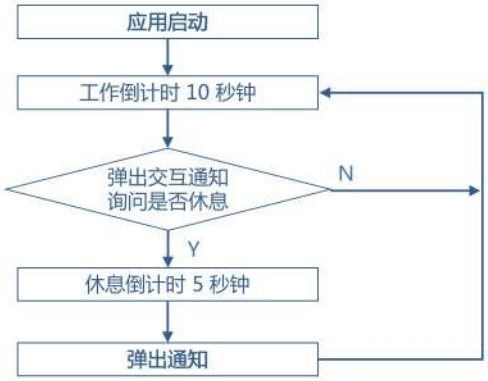
开发一个简单版的番茄计时器

- 番茄工作法是一种时间管理法；
- 使用定时器分割出一个一般为「25 分钟工作时间和 5 分钟休息时间」的时间段。

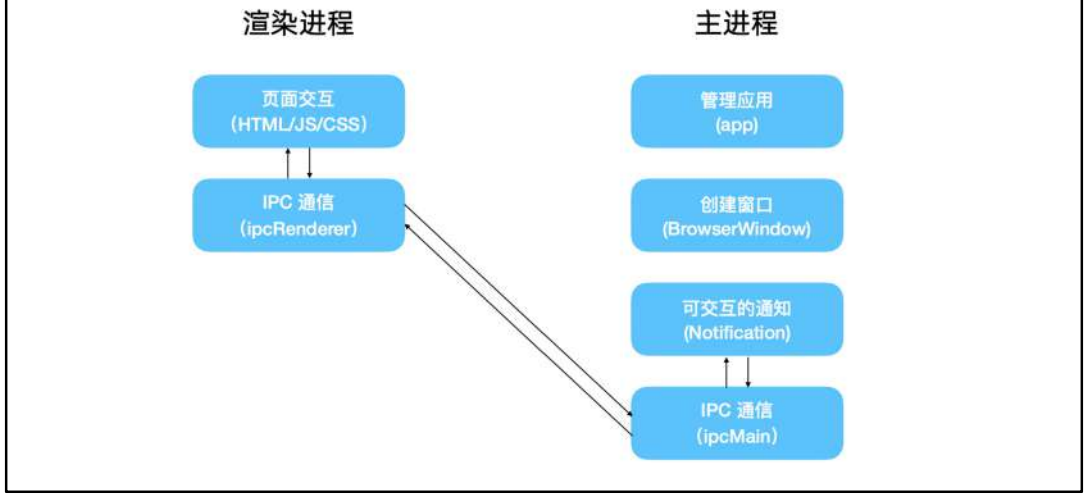


No. 2 简单实战

开发一个简单版的番茄计时器



No. 2 简单实战


渲染进程
主进程
页面交互
(HTML/JS/CSS)
IPC 通信
(ipcRenderer)
管理应用
(app)
创建窗口
(BrowserWindow)
可交互的通知
(Notification)
IPC 通信
(ipcMain)

No. 2 简单实战

Electron 主进程模块讲解

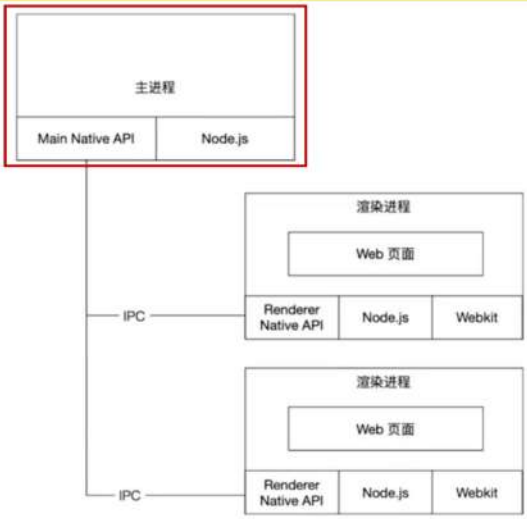
- app，用于控制应用生命周期。本例用到 app.on('ready', callback)，用于在应用就绪后开始业务
- BrowserWindow，用于创建和控制窗口
  - let win = new BrowserWindow({width, height, ...}) // 创建窗口，并设置宽高
  - win.loadURL(url)、win.loadFile(path) // 加载页面
- Notification，创建通知
  - let notification = new Notification({title, body, actions:[{text, type}]})
  - notification.show()
- ipcMain.handle(channel, handler)，处理传染进程的 channel 请求，在 handler 中返回结果


3
与 Web 开发的不同

No. 3 与 Web 开发的不同

主进程

- package.json 的 main 脚本的进程被称为主进程
- 每个应用只有一个主进程
- 管理原生 GUI，典型的窗口（BrowserWindow、Tray、Dock、Menu）
- 创建渲染进程
- 控制应用生命周期（app）

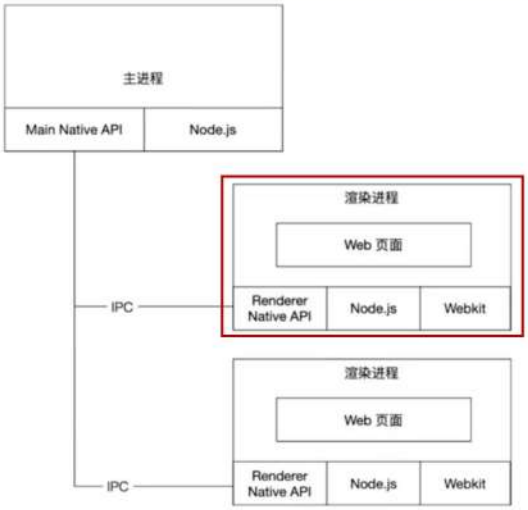


要从web页面跳出来，关注到外面，需要关注到渲染进程和主进程，要学会使用他们的模块，然后学会使用nodejs来帮助我们完成开发。

No. 3 与 Web 开发的不同

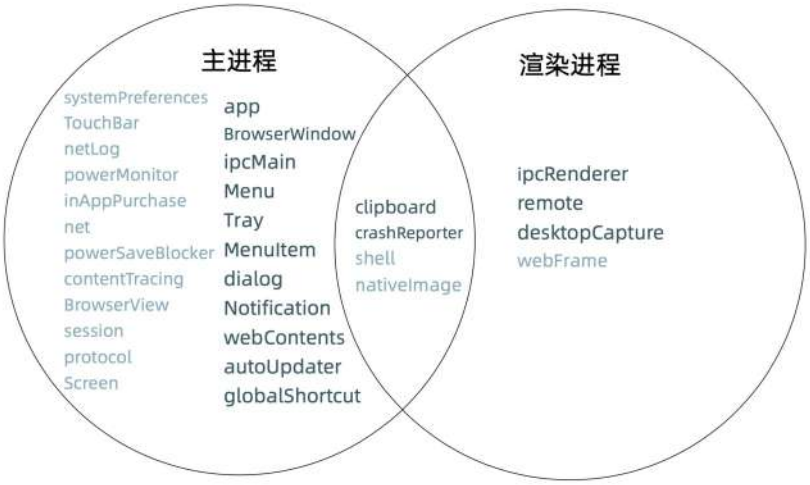
渲染进程

- 展示 Web 页面的进程称为渲染进程
- 通过 Node.js、Electron 提供的 API 可以跟系统底层打交道
- 一个 Electron 应用可以有多个渲染进程



渲染进程和我们开发web页面差不多，只是在普通浏览器中，web的页面是运行在沙盒环境中，无法访问操作系统的原生资源；而electron可以让我们使用nodejs来访问我们的底层系统。

No. 3 与 Web 开发的不同



No. 3 与 Web 开发的不同

进程间通信

- Electron 进程间通信的目的：通知事件、数据传输、数据共享
- Electron 提供了 IPC 通信模块，主进程的 ipcMain 和渲染进程的 ipcRenderer
- ipcMain 和 ipcRenderer 都是 EventEmitter 对象

No. 3 与 Web 开发的不同

进程间通信 —— 从渲染进程到主进程

- Callback 写法：
  - ipcRenderer.send(channel, ...args)
  - ipcMain.on(channel, handler)
- Promise 写法（Electron 7.0 之后，处理请求+响应模式）：
  - ipcRenderer.invoke(channel, ...args)
  - ipcMain.handle(channel, handler)



因为是一个主进程、多个渲染进程，所以不能用 ipcMain.send()，因为不知道发给谁。因此需要在 webContents 具体窗口中发送。



这种情况在实际项目中的需求比较少，但有时候的确需要在两个页面间有一些联动，第一种就是事件，比如在大项目中，在聊天页面去点击一个文件预览，我们会进行一个窗口的打开，当窗口关闭的时候，我们需要将预览结束的事件返回给聊天的渲染进程，用来做数据统计，在electron 5之后，会比较方便。第二种就是数据共享，remote 方法会将我们的数据挂在一个全局的过程中，但不建议使用，因为如果用得不好会导致程序卡顿。

No. 3 与 Web 开发的不同

Native 能力及原生 GUI

- 使用 Electron API 创建原生 GUI
  - BrowserWindow 应用窗口
  - Tray 托盘
  - app 设置 dock.badge
  - Menu 菜单
  - dialog 原生弹框
  - TouchBar 苹果触控栏
  - .....

No. 3 与 Web 开发的不同

Native 能力及原生 GUI

- 使用 Electron API 获得底层能力
  - clipboard 剪切板
  - globalShortcut 全局快捷键
  - desktopCapture 捕获桌面
  - shell 打开文件、URL
  - ……

No. 3 与 Web 开发的不同

Native 能力及原生 GUI

- 使用 Node.js 获得底层能力
  - Electron 同时在主进程和渲染进程中对 Node.js 暴露了所有的接口
    - fs 进行文件读写
    - crypto 进行加解密
  - 通过 npm 安装即可引入社区上所有的 Node.js 库



我们还可以通过 nodejs 的插件机制去集成 c++，如果业务里有一些用 c++ 实现好的动态库，可以用 node-ffi 库去集成那些动态库，比如打印功能。

No. 3 与 Web 开发的不同

Native 能力及原生 GUI

- 调用 OS 能力
  - winRT (https://github.com/NodeRT/NodeRT)
  - AppleScript (https://github.com/TooTallNate/node-applescript)
  - Shell (node.js child_process)

No. 3 与 Web 开发的不同

Native 能力及原生 GUI

Node.js	原生 API	文件读写、操作系统、加解密
	Node.js 扩展	集成三方 C++ 库
Electron Native API	客户端 GUI	右键菜单、窗口定制 、系统托盘、Dock..
	桌面环境集成	系统通知、剪切板、系统快捷键、文件拖放
	设备 API	电源监视、内存、CPU、屏幕
OS	Applescript(macOS)	系统原生应用调用
	Windows RT	USB、蓝牙、预览文件

No. 3 与 Web 开发的不同

释放想象力

- **无兼容问题**
  - 不用担心在 IE、Firefox 上的表现差异了
  - 大胆使用 Chrome 浏览器已经支持的 API
  - babel 中设置 targets 为 Electron 对应的 Chrome 版本
- **举个栗子**
  - 纯天然 LazyLoad（https://mathiasbynens.be/demo/img-loading-lazy）

No. 3 与 Web 开发的不同

释放想象力

- ES 6/7/8/9/10 高级语法
  - async await / Promise
  - String/Array/Object 等高级用法
  - BigInt

No. 3 与 Web 开发的不同

释放想象力

- **无跨域问题**
  - 使用 Node.js 发送请求
  - 使用 Electron net 发送请求

No. 3 与 Web 开发的不同

释放想象力

- More
  - 操作本地文件
  - 更好的本地 DB
  - 多线程、多进程并行

No. 3 与 Web 开发的不同

释放想象力

- 无浏览器兼容问题
- 最新 Chrome Feature
- No Polyfill
- ES 高级语法
- 无跨域问题
- Powered by Node.js




Thanks ·