

Electron Security

sandbox



NOP Team



sandbox | Electron 安全

0x01 简介

大家好，今天跟大家讨论的内容是 `sandbox`，`sandbox` 是一项 `Chromium` 的功能，它使用操作系统来显著地限制渲染器进程可以访问的内容，在 `Electron` 中，限制的方面还要包括 `Node.js` 能力

这篇文章内容很重要，因为它修正了我们之前

`nodeIntegration`、`contextIsolation`、`Preload` 等内容中的错误，所以请大家至少把总结章节看完

<https://www.electronjs.org/zh/docs/latest/tutorial/security#4-%E5%90%AF%E7%94%A8%E8%BF%9B%E7%A8%8B%E6%B2%99%E7%9B%92%E5%8C%96>

<https://chromium.googlesource.com/chromium/src/+/HEAD/docs/design/sandbox.md>

<https://www.electronjs.org/zh/docs/latest/tutorial/sandbox>

0x02 Chromium 沙盒

<https://chromium.googlesource.com/chromium/src/+/HEAD/docs/design/sandbox.md>

https://chromium.googlesource.com/chromium/src/+/HEAD/docs/design/sandbox_faq.md

在上面的两篇文章中，`Chromium` 官方详细介绍了 `Chromium` 沙盒及其使用的具体技术、策略、遇到的困难和解决办法，这是一个比较复杂的工程

`Chromium` 的 `sandbox` 并不仅仅给 `Chromium` 等浏览器使用，它可以给任意 `C/C++` 应用程序使用，它作为一个 `C++` 库，可以在调用后开始保护应用程序，可以创建沙盒进程，这是一种在非常限制的环境中执行的进程。沙盒进程可以自由使用的唯一资源是 `CPU` 周期和内存。例如，沙盒进程无法写入磁盘或显示自己的窗口。他们究竟能做什么是由一个明确的策略控制的。

`Chromium` 渲染器是沙盒进程。

Chromium 沙盒设计原则如下

- 不重复造轮子
- 最小特权原则
- 假设沙盒中的代码是恶意的
- 仿真不是安全性

仿真和虚拟机解决方案本身并不提供安全性。沙盒不应依赖代码仿真、代码转换或修补来提供安全性。

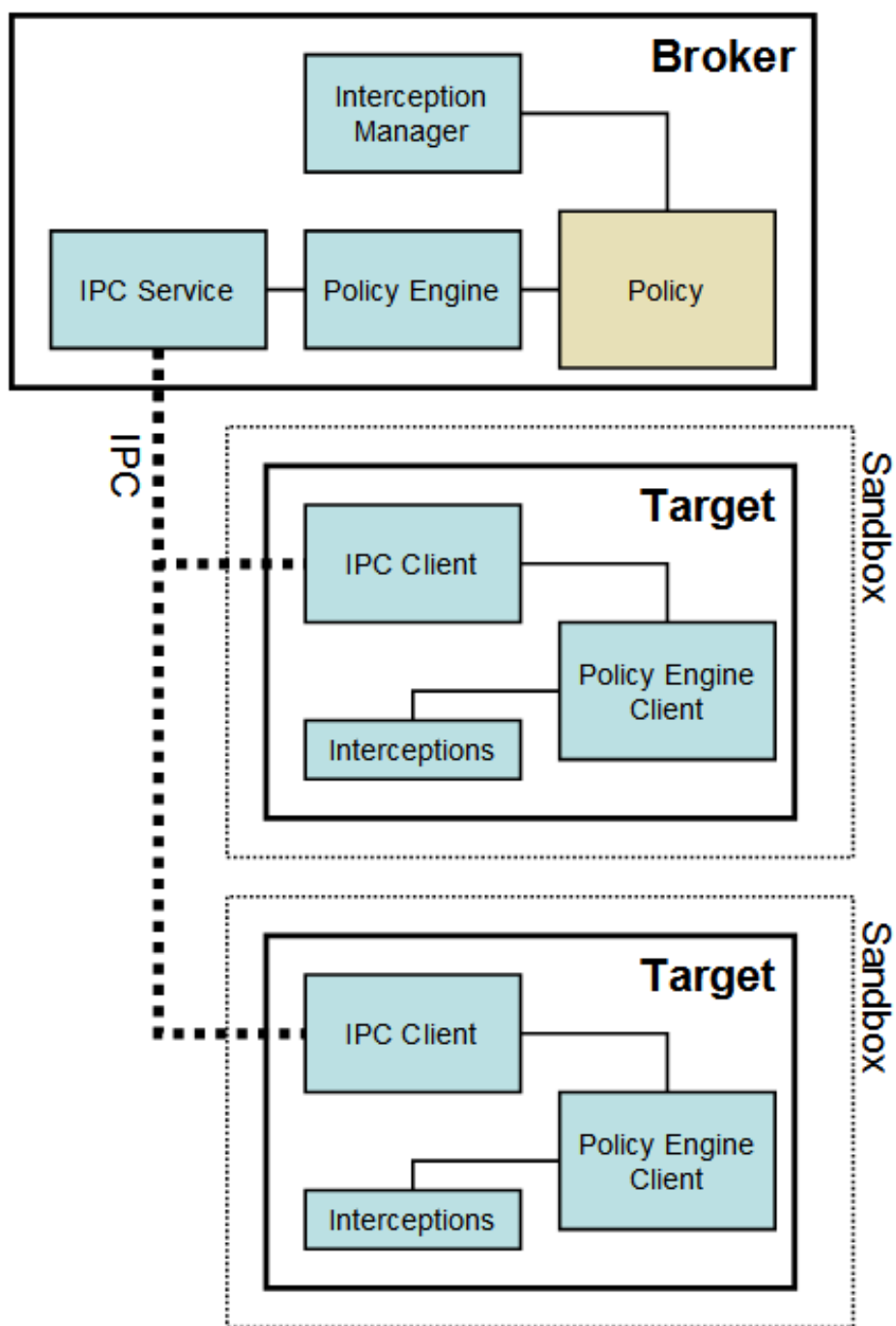
在不同平台上，沙盒都有自己的架构，关于 Windows、Linux、Mac 上具体策略如下

<https://chromium.googlesource.com/chromium/src/+/HEAD/docs/design/sandbox.md#sandbox-windows-architecture>

<https://chromium.googlesource.com/chromium/src/+/HEAD/docs/linux/sandboxing.md>

<http://dev.chromium.org/developers/design-documents/sandbox/osx-sandboxing-design>

在 Windows 平台架构如下



沙盒整体分为两部分: **Broker** 和 **Target**

Broker 是负责控制, 相当于管理人员, **Target** 可以有很多个, 是实际去干活的打工人(客户端), 它们之间有不同的职责, 通过 **IPC** 进行通信

在 Chromium 中, **Broker** 始终是浏览器进程。从广义上讲, 代理是沙盒进程活动的特权控制者/监督者。 **Broker** 的职责是

- 为每个 **Target** 进程指定策略

- 生成 `Target` 进程
- 托管沙盒策略引擎服务
- 托管沙盒拦截管理器
- 托管沙盒 `IPC` 服务（到 `Target` 进程）
- 代表 `Target` 进程执行策略允许的操作

`Target` 的职责是

- 沙盒化所有代码
- 沙盒 `IPC` 客户端
- 沙盒策略引擎客户端
- 沙盒拦截

`Broker` 与 `Target` 之间的 `IPC` 通信是一种低级机制（与 `Chromium` 的 `IPC` 不同），用于透明地将某些 `Windows API` 调用从 `Target` 转发到 `Broker`：这些调用根据策略进行评估。然后，`Broker` 执行策略允许的调用，并通过相同的 `IPC` 将结果返回给目标进程。

具体 `Chromium` 的沙盒技术细节可以参考上面提到的文章，内容较为详细

0x03 Electron 沙盒

在 `Electron` 中沙盒进程大部分地表现都与 `Chromium` 差不多，但因为介面是 `Node.js` 的关系 `Electron` 有一些额外的概念需要考虑

对于渲染进程来说，如果设置了沙盒化，则它的行为和常规 `Chromium` 渲染器是一致的，它不可以执行 `Node.js`

对于 `Preload` 脚本来说，它属于是渲染进程的一部分，但沙盒化后仍然可以使用部分 `Node.js` 的 `API`，毕竟它要负责渲染器进程和主进程之间的通信，`Electron` 官方给 `Preload` 脚本提供了一个 `require` 方法，这个方法名字和 `Node.js` 中的 `require` 一样，但提供形式是 `Polyfilled`，也就是说 `Electron` 自己定制实现并提供的，具体可以

使用哪些 API 可以参照之前预加载脚本那篇文章

为单个渲染进程设置沙盒化也比较简单，只需要设置 `sandbox: true`

```
app.whenReady().then(() => {
  const win = new BrowserWindow({
    webPreferences: {
      sandbox: false
    }
  })
  win.loadURL('https://google.com')
})
```

当然也可以全局沙盒化

```
app.enableSandbox()
app.whenReady().then(() => {
  // 因为调用了app.enableSandbox(), 所以任何sandbox:false的调用都会被覆盖。
  const win = new BrowserWindow()
  win.loadURL('https://google.com')
})
```

虽然沙盒限制比较强大，但是 Electron 官方还是强调了，尽量不要在沙盒中渲染不受信任的内容

0x04 sandbox 历史

- Electron 3.0 允许在沙盒化的渲染进程中使用 `webview`
- Electron 6.0 混合沙盒默认启用

此时开始，`sandbox` 显式地设置为 `true` 后，`Preload` 不可以执行危险的 `Node.js` API

- Electron 20.0 默认情况下会对渲染器进行沙盒化

此时开始，默认情况下 `Preload` 不可以执行危险的 `Node.js` API

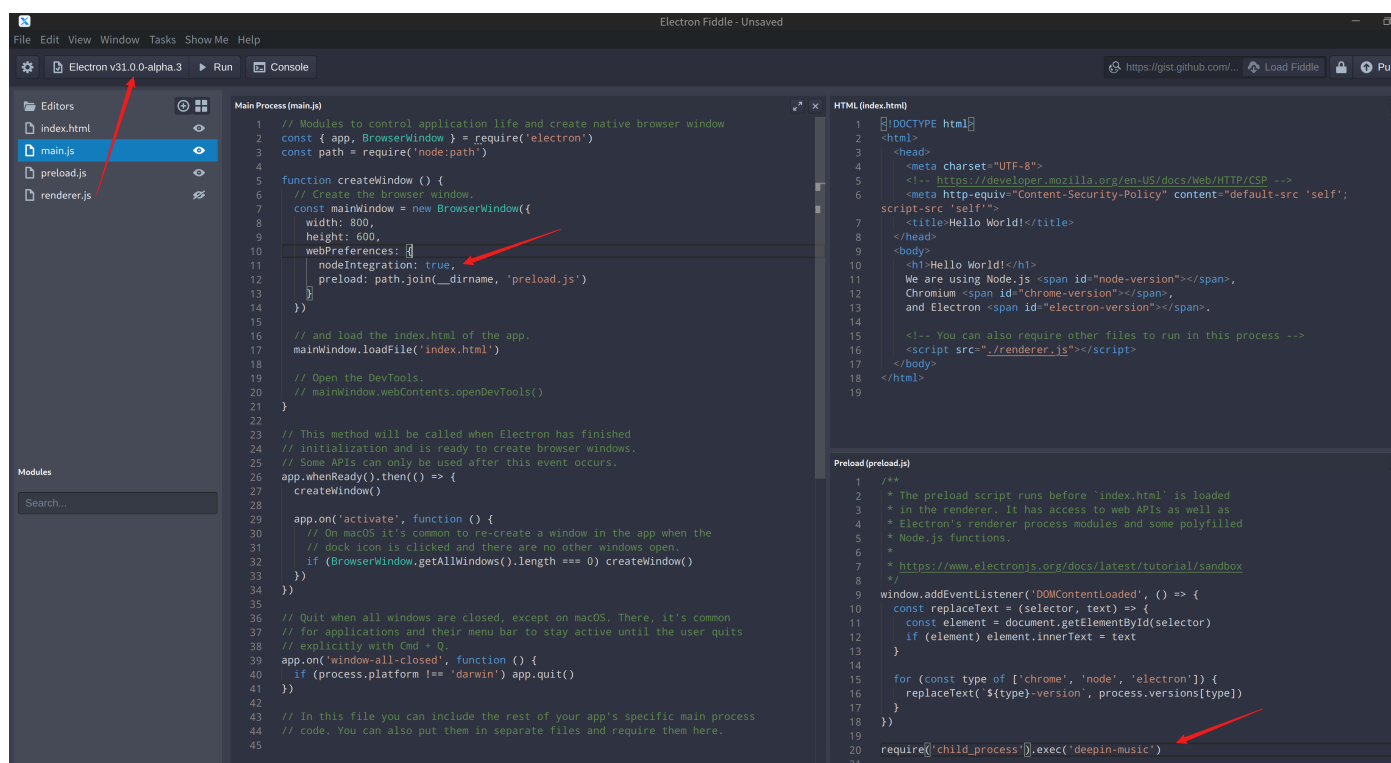
这里大家(尤其是看了我们之前的文章的朋友们)一定要注意一个问题

在 **Electron 20.0** 版本后, 虽然默认对渲染器进行沙盒化, 但这并不等于从 **20.0** 版本开始默认 **sandbox: true**

即 **Electron 20.0** $\neq$ **sandbox:true**

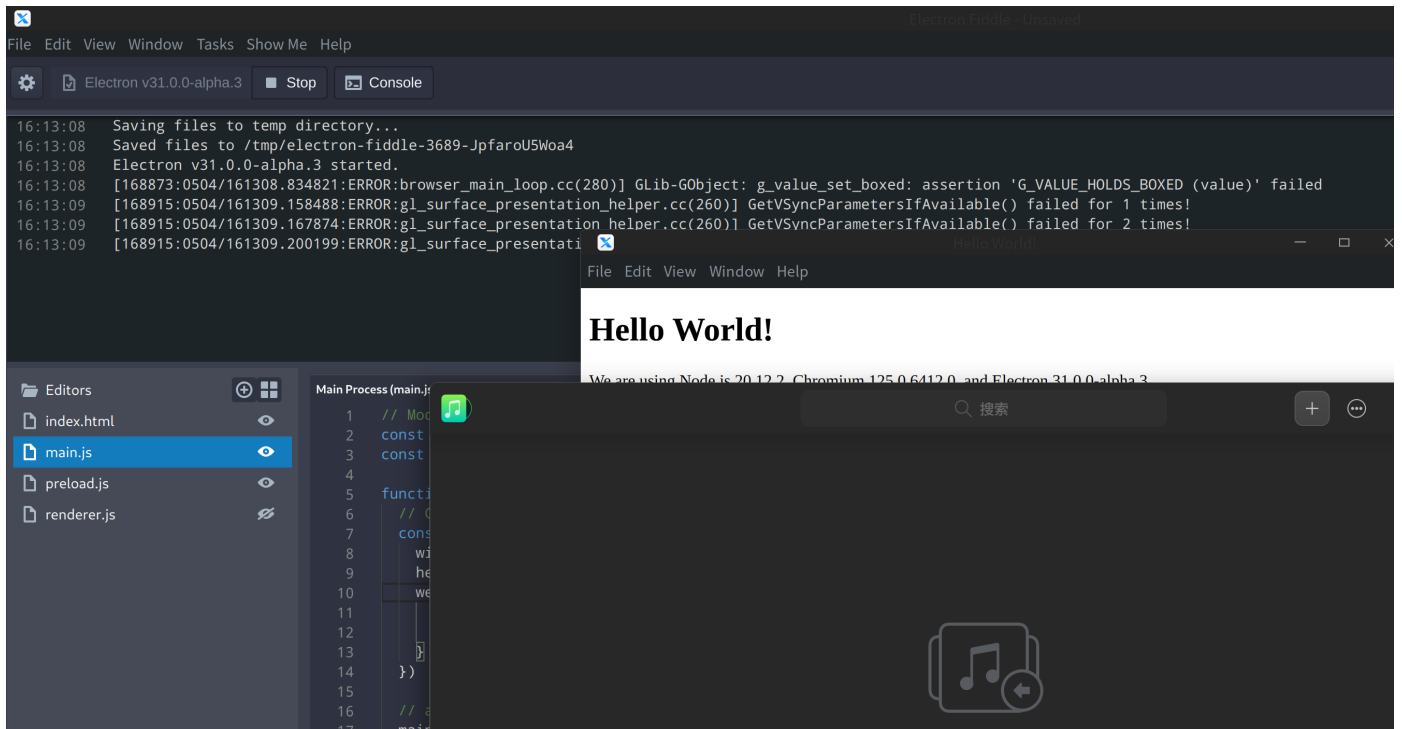
因为当 **nodeIntegration**

、**nodeIntegrationInSubFrames**、**nodeIntegrationInWorker** 被设置为 **true** 时, **sandbox** 对于 **Node.js** 的保护效果会失效

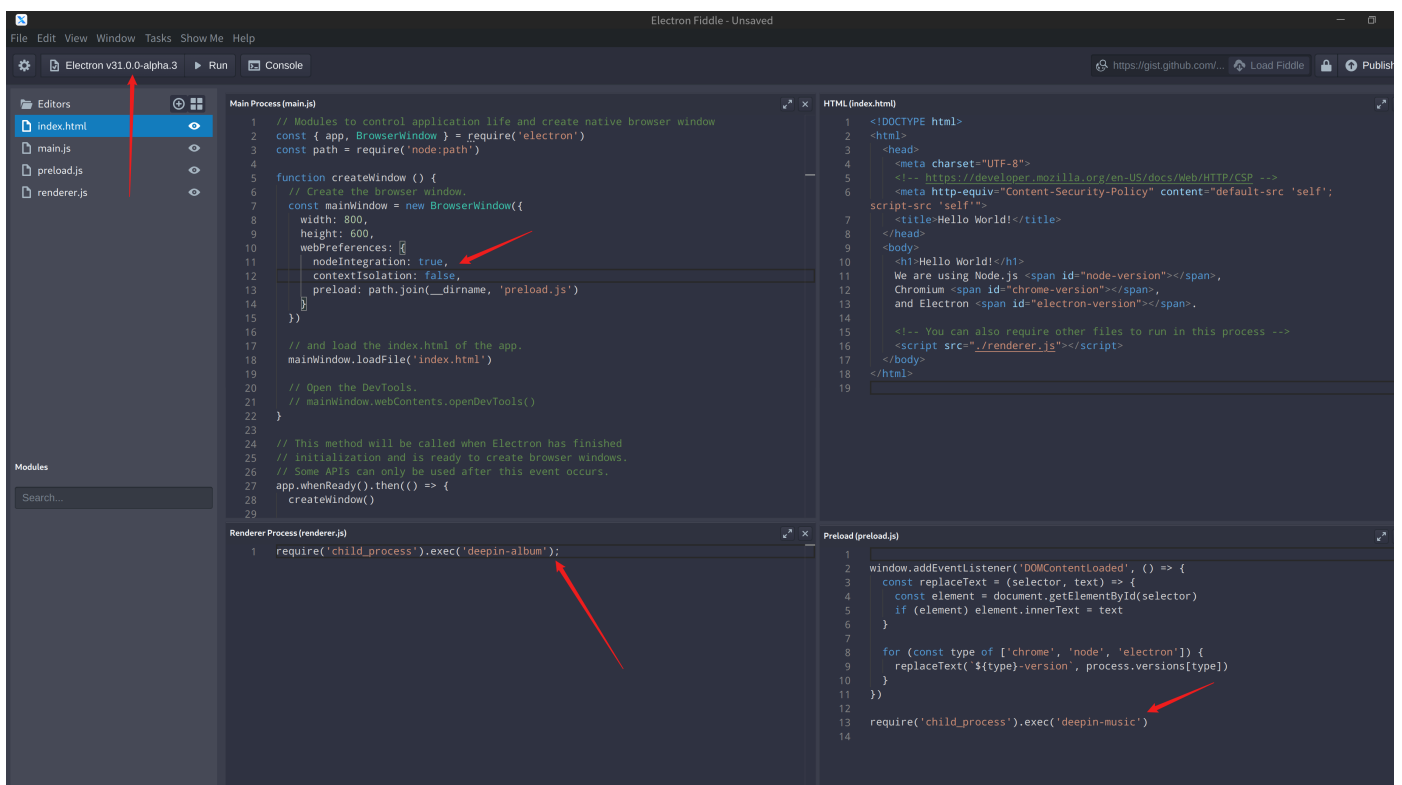


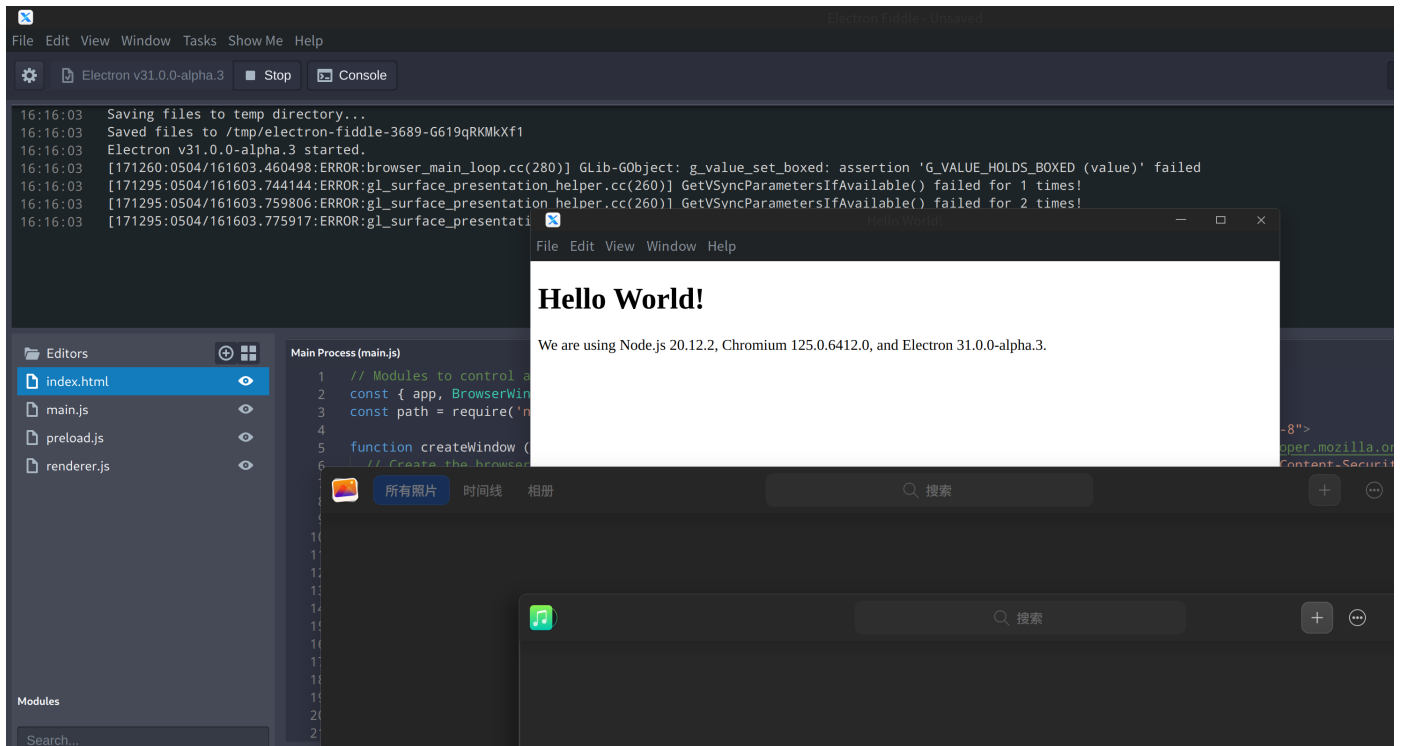
```
1 // Modules to control application life and create native browser window
2 const { app, BrowserWindow } = require('electron')
3 const path = require('node:path')
4
5 function createWindow() {
6   // Create the browser window.
7   const mainWindow = new BrowserWindow({
8     width: 800,
9     height: 600,
10    webPreferences: {
11      nodeIntegration: true,
12      preload: path.join(__dirname, 'preload.js')
13    }
14  })
15
16  // and load the index.html of the app.
17  mainWindow.loadFile('index.html')
18
19  // Open the DevTools.
20  mainWindow.webContents.openDevTools()
21 }
22
23 // This method will be called when Electron has finished
24 // initialization and is ready to create browser windows.
25 // Some APIs can only be used after this event occurs.
26 app.whenReady().then(() => {
27   createWindow()
28
29   app.on('activate', function () {
30     // On macOS it's common to re-create a window in the app when the
31     // dock icon is clicked and there are no other windows open.
32     if (BrowserWindow.getAllWindows().length === 0) createWindow()
33   })
34 })
35
36 // Quit when all windows are closed, except on macOS. There, it's common
37 // for applications and their menu bar to stay active until the user quits
38 // explicitly with Cmd + Q.
39 app.on('window-all-closed', function () {
40   if (process.platform !== 'darwin') app.quit()
41 })
42
43 // In this file you can include the rest of your app's specific main process
44 // code. You can also put them in separate files and require them here.
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Electron 版本为 **31.0.0-alpha.3** , 设置了 **nodeIntegration: true** , 在 **Preload** 脚本中打开 **Music** 程序, 执行测试

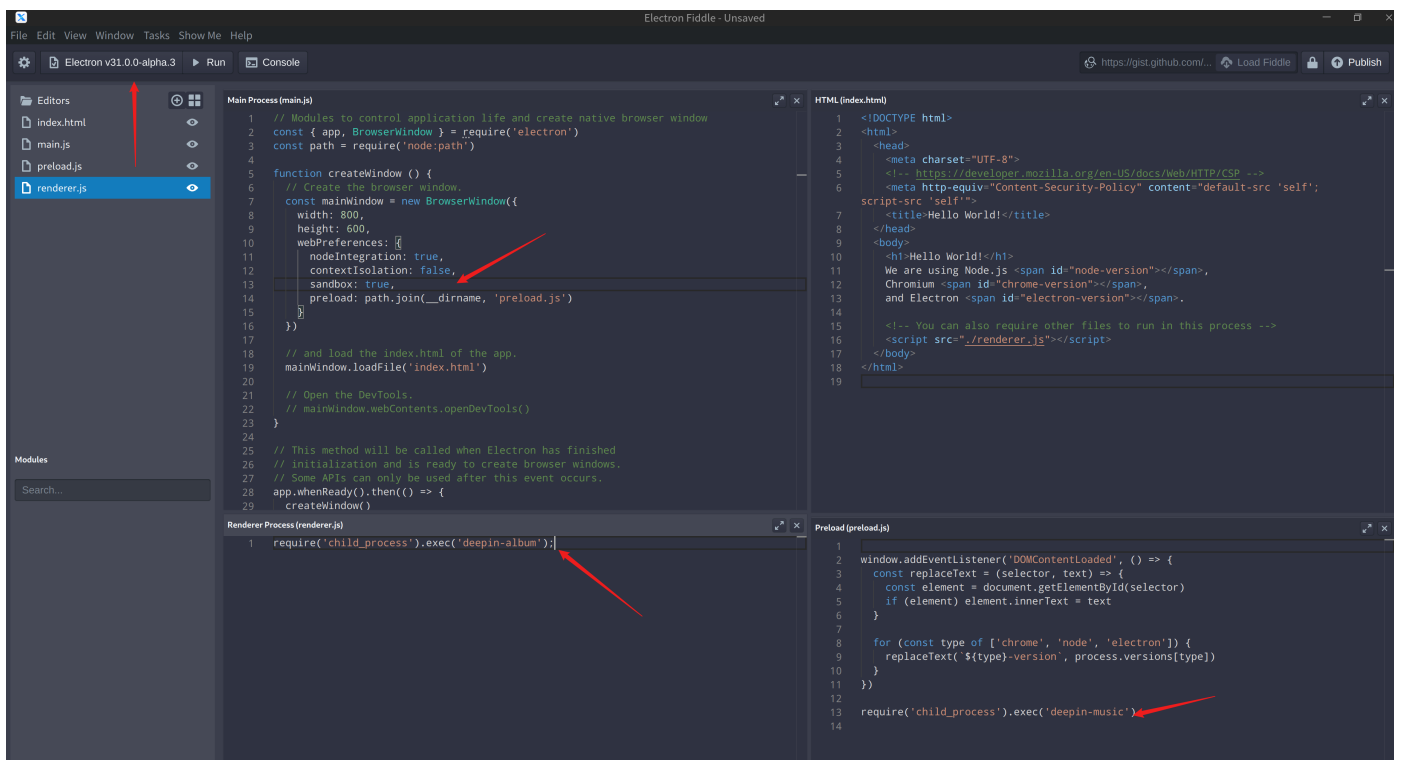


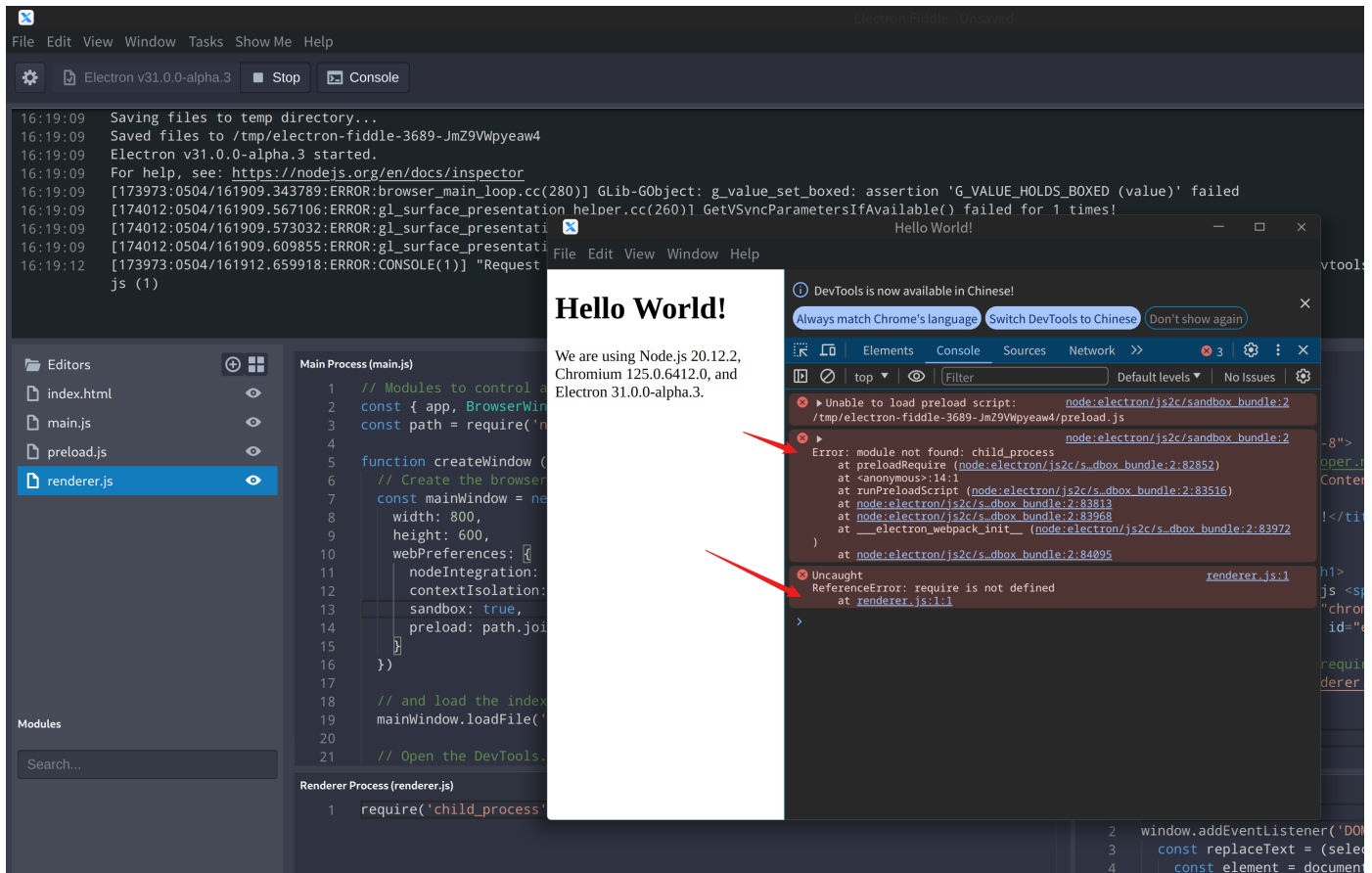
成功打开 `Music`，在渲染页面的 `renderer.js` 中通过 `Node.js` 打开相册，设置 `contextIsolation: false` 测试





成功执行，显式设置 `sandbox: true` 后再次测试





此时执行失败，所以 `Electron 20.0` $\neq$ `sandbox:true`

经过测试，`sandbox` 对于上下文隔离的保护与 `Node.js` 的效果一致，在 `Electron 20.0` 之后，默认配置下，当 `contextIsolation: false` 时，`sandbox` 对于上下文隔离的保护失效

0x05 总结

关于 `sandbox` 的设计架构及功能，通过之前的文章以及今天的介绍，尤其是介绍中的 `Chromium` 链接中已经说得比较详细了，今天这篇文章的重点在于 `Electron 20.0` $\neq$ `sandbox:true` 这件事

从开发者角度看，逻辑通顺，我想让渲染进程执行 `Node.js`，我就使用 `nodeIntegration*`，此时 `sandbox` 也好，其他什么安全配置也好，都应该为我让路，自动让路更好

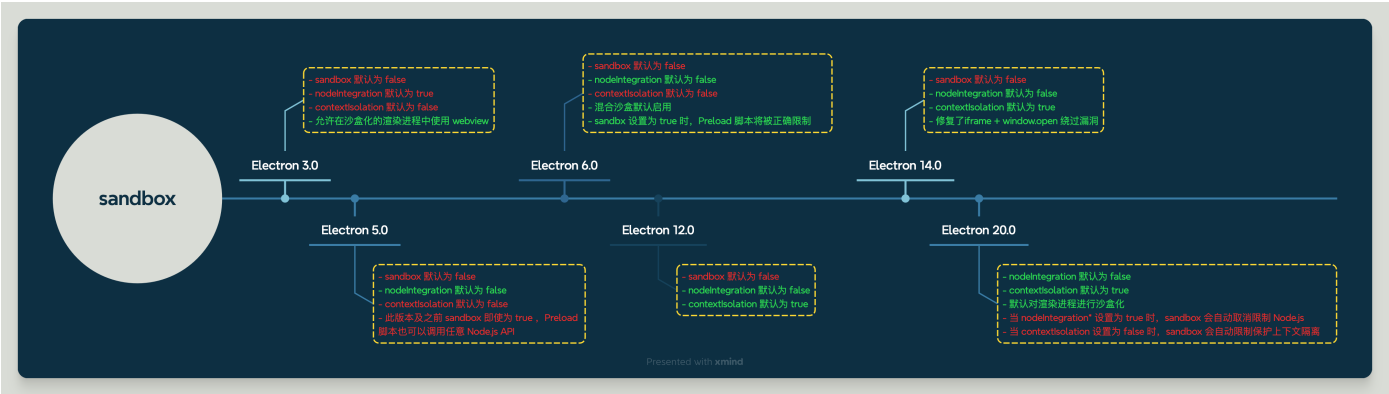
但是从安全人员角度看，在对程序进行审计的时候可能会造成一些疏忽，尤其是对 `Electron` 安全做过研究的安全研究员，可能会认为 `Electron 20.0` 版本以后，只要没有显式地设置 `sandbox: false` 就会被沙盒好好地保护着，其实并不是这么回事

当 `nodeIntegration` 、 `nodeIntegrationInSubFrames` 、 `nodeIntegrationInWorker` 被设置为 `true` 时， `sandbox` 对于 `Node.js` 的保护效果就会失效

当 `contextIsolation` 被设置为 `false` 时， `sandbox` 对于上下文隔离的保护效果就会失效

在之前 `nodeIntegration` 和 `contextIsolation` 文章中，我们测试过程中对于 `sandbox` 默认值只测试了预加载脚本的 `Node.js` 能力和显式设置 `sandbox: true/false` 时 `Node.js` 的表现，并没有对不设置 `sandbox` ，使用其默认值测试 `nodeIntegration` 和 `contextIsolation` 配置项，因此得出了在 `Electron 20.0` 以后默认情况下 `sandbox: true` 的错误结论

时间线如下







微信搜一搜

Q NOP Team