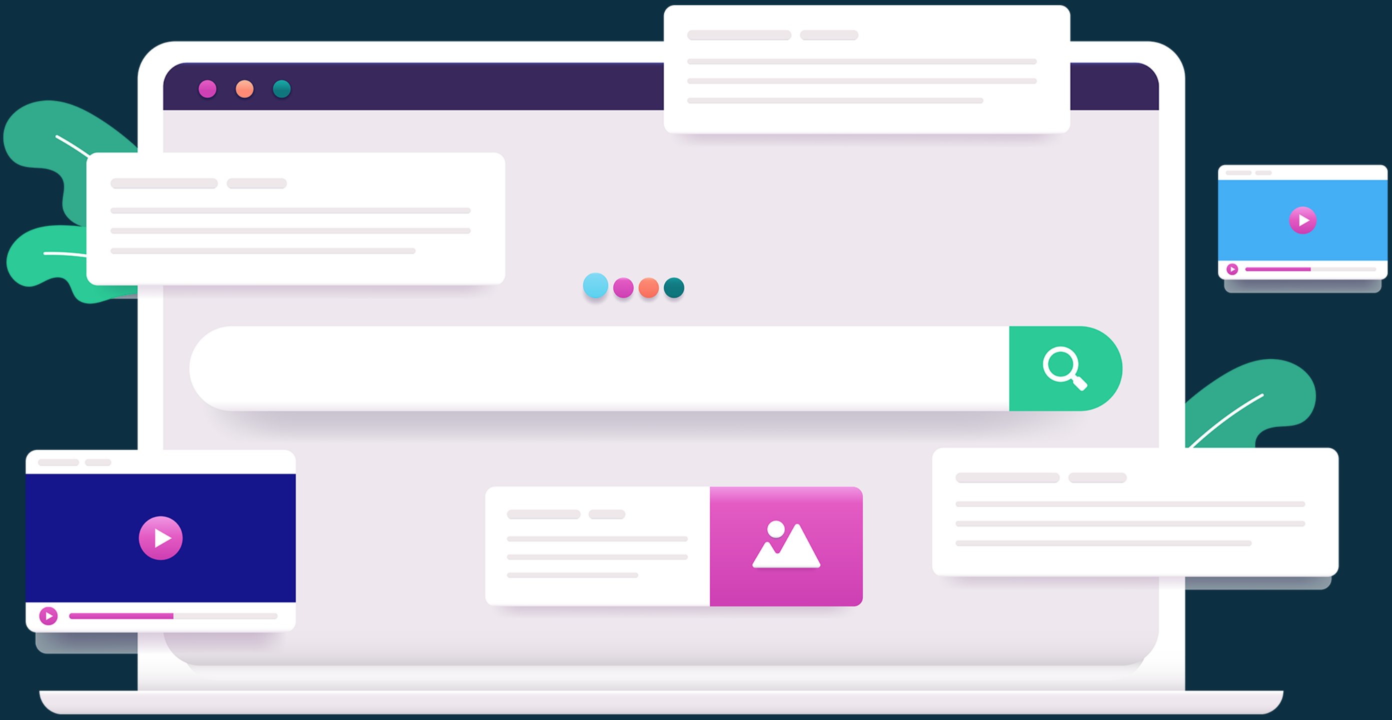
Electron Security



nodeIntegrationInSubFrames

NOP Team

nodeIntegrationInSubFrames

0x01 简介

大家好，今天和大家讨论 `nodeIntegrationInSubFrames` ，这个选项看起来和 `nodeIntegration` 很像，不过后面跟了 `InSubFrames` ，说明是在 `SubFrames` 中开启 `Node.js`

这是一个实验性质的选项，决定是否允许在子页面(`iframe`)或子窗口(`child window`)中集成 `Node.js` ； 预先加载的脚本会被注入到每一个 `iframe` ，你可以用 `process.isMainFrame` 来判断当前是否处于主框架（ `main frame` ）中

> https://www.electronjs.org/zh/docs/latest/api/structures/browser-window-options

0x02 SubFrames

官方文档中 `SubFrames` 是指 `iframe` 和子窗口，那 `iframe` 和子窗口到底是用来干嘛的呢?

其实都是为了在一个页面中嵌入其他页面，例如我想在搜狐的网站中嵌入一段人民日报的新闻页面

这种行为在 `Electron` 官方文档中叫做 `Web` 嵌入，关于 `web` 嵌入，后续我们还会出单独的文章进行讨论

> https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/iframe
>
> https://www.electronjs.org/zh/docs/latest/tutorial/web-embeds#iframes

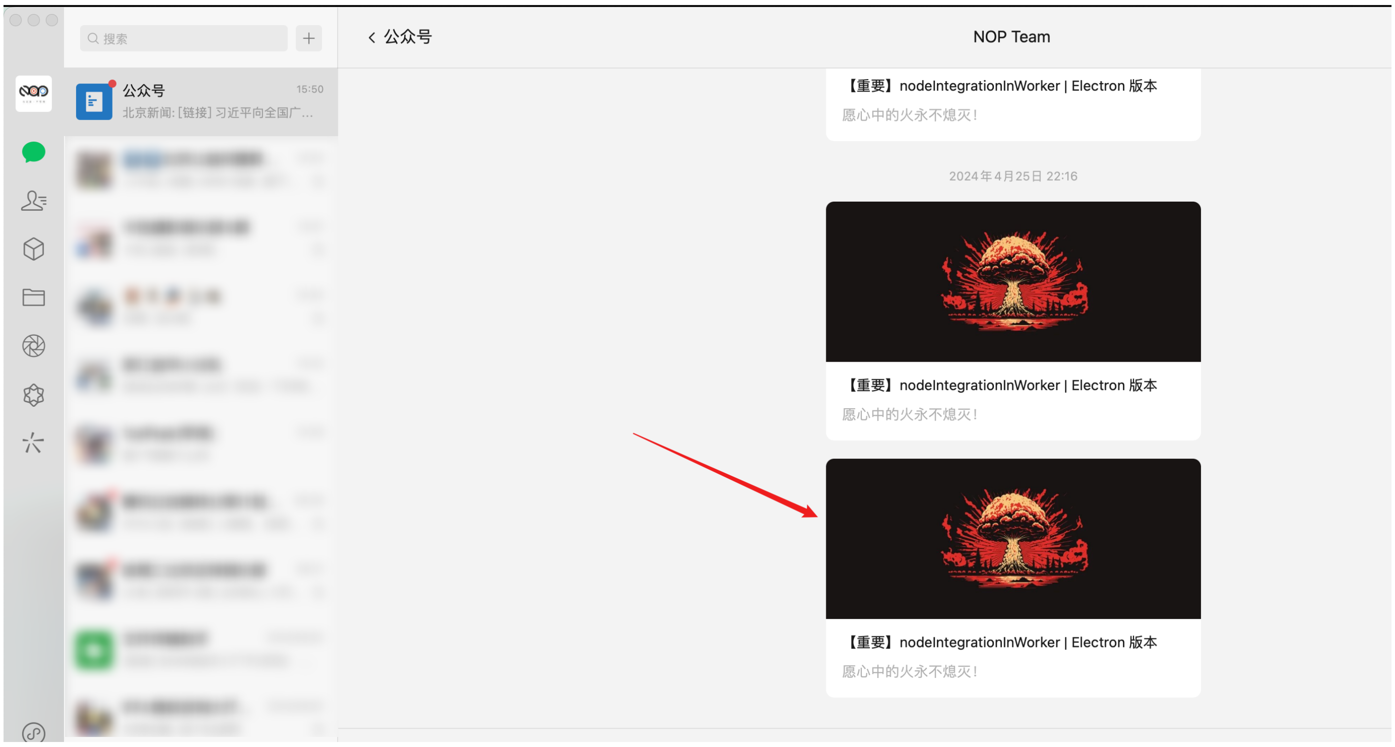
`iframe` 在之前已经参与了很多测试了，在 `web` 技术中也包含，大家了解得可能已经比较透彻了

通过 `iframe` 的内容本身有自己独立的上下文(`context`)，而嵌入它的网页被称为父级浏览上下文，当然这是可以嵌套的，就像物理机里装虚拟机，在虚拟机里又装了虚拟机一个道理，而最终的物理机被称为顶级浏览上下文

在 `Electron` 之前的测试中，我们只用到了一个窗口，我们一直称之为主窗口，但从逻辑角度来说，没有子窗口的存在，也就没有什么主窗口之说

大家有些时候在使用应用程序的时候，点击某个功能会跳出来一个新的窗口，这个就叫做子窗口

举个例子，我们在电脑版微信中查看公众号文章时，点击文章，会出现一个新的窗口来显示文章内容，而不是在原本的窗口呢，这样原本的窗口可以继续聊天等

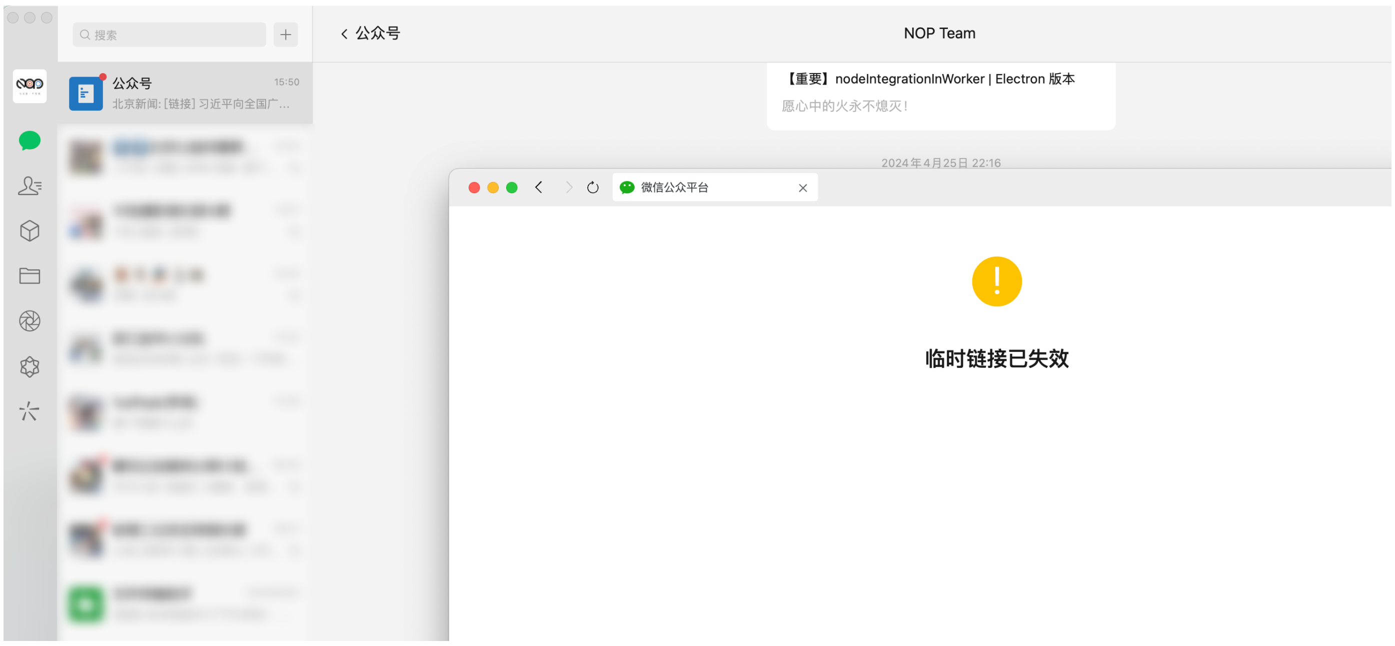


【重要】nodeIntegrationInWorker | Electron 版本
愿心中的火永不熄灭！

2024年4月25日 22:16

【重要】nodeIntegrationInWorker | Electron 版本
愿心中的火永不熄灭！

【重要】nodeIntegrationInWorker | Electron 版本
愿心中的火永不熄灭！



【重要】nodeIntegrationInWorker | Electron 版本
愿心中的火永不熄灭！

2024年4月25日 22:16

临时链接已失效

> https://www.electronjs.org/zh/docs/latest/api/browser-window#%E7%88%B6%E5%AD%90%E7%AA%97%E5%8F%A3

创建子窗口的方式也比较简单

```
const { BrowserWindow } = require('electron')

const top = new BrowserWindow()
const child = new BrowserWindow({ parent: top })
child.show()
top.show()
```

在配置参数中添加 `parent: xxx` 指定父窗口即可

问题来了，为什么要设置父子窗口呢?

在之前的一些版本中，似乎子窗口会继承父窗口的一些配置，但后来主要是为了生命周期等，简单来说，我把父窗口关了，子窗口也会被关闭或其他设置

该参数要在父窗口初始化是配置，而不是子窗口

0x03 测试 iframe

1. 搭建 iframe 服务器

`192.168.31.216`

`1.html`

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <div>
    <h1>iframe 页面 - 1.html</h1>
    <script src="iframe_1.js"></script>
  </div>
</body>
</html>
```

其中 `iframe_1.js`

```js
require('child_process').exec('deepin-music');
```

同时，我们再搭建一个 `iframe` + `window.open` 的 `2.html`

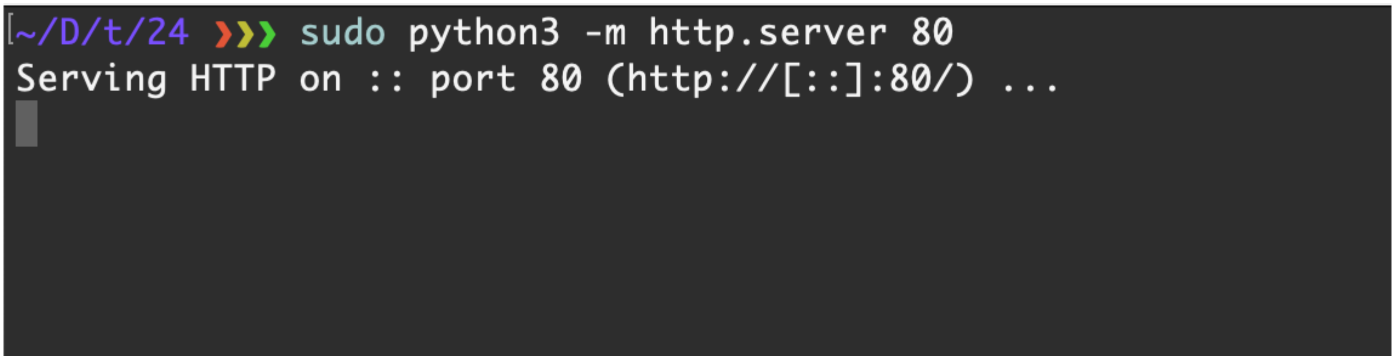
```html
<!DOCTYPE html>
<html>
<head>
</head>
<body>
  <div>
    <h1>iframe 2.html</h1>
    <script>window.open("http://192.168.31.216/3.html")</script>
  </div>
</body>
</html>
```

其中 `3.html` 执行 `iframe_2.js` ，打开相册

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <div>
    <h1>iframe 页面 - 3.html</h1>
    <script src="iframe_2.js"></script>
  </div>
</body>
</html>
```

`iframe_2.js`

```js
require('child_process').exec('deepin-album');
```



```
~/D/t/24 ❯❯❯ sudo python3 -m http.server 80
Serving HTTP on :: port 80 (http://[::]:80/) ...
```

2. 搭建测试环境

关闭 `CSP` ，关闭 `sandbox` ，在 `index.html` 中嵌入 `iframe`

`main.js`

```js
// Modules to control application life and create native browser window
const { app, BrowserWindow } = require('electron')
const path = require('node:path')
```

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      sandbox: false,
      nodeIntegrationInSubFrames: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  // mainWindow.webContents.openDevTools()
}

app.whenReady().then(() => {
  createWindow()

  app.on('activate', function () {
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

app.on('window-all-closed', function () {
  if (process.platform !== 'darwin') app.quit()
})
```

`index.html`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
```

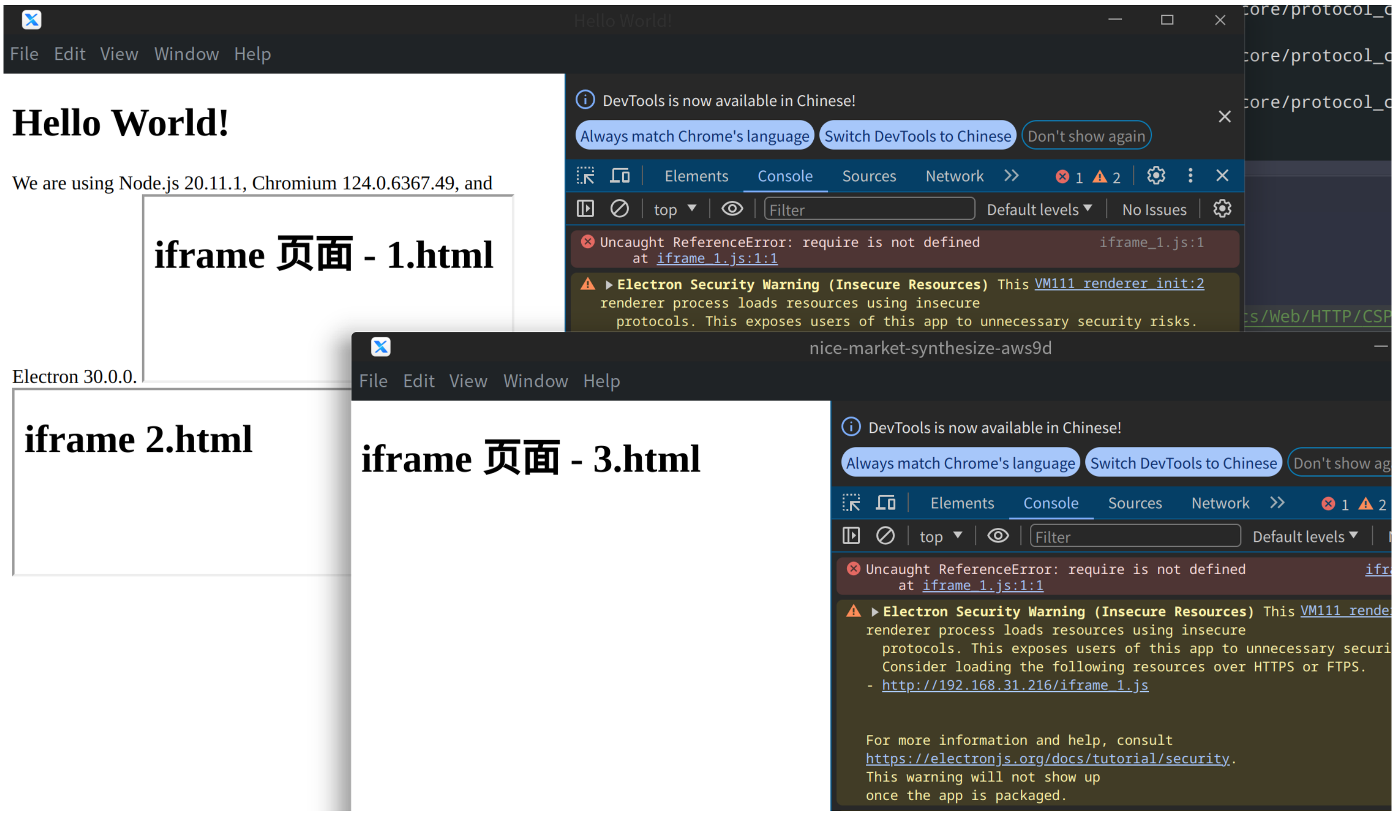
```html
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src
'self'; script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <script src="./renderer.js"></script>
    <iframe src="http://192.168.31.216/1.html"></iframe>
    <iframe src="http://192.168.31.216/2.html"></iframe>
  </body>
</html>
```

以上安全配置总结为:

- `nodeIntegration: false`
- `contextIsolation: true`
- `sandbox: false`
- `nodeIntegrationInSubFrames: true`

3. 测试执行 Node.js

执行测试


Hello World!
File Edit View Window Help
DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to Chinese
Don't show again
Elements
Console
Sources
Network
top
Filter
Default levels
No Issues
nice-market-synthesize-aws9d

Hello World!

We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

iframe 页面 - 1.html

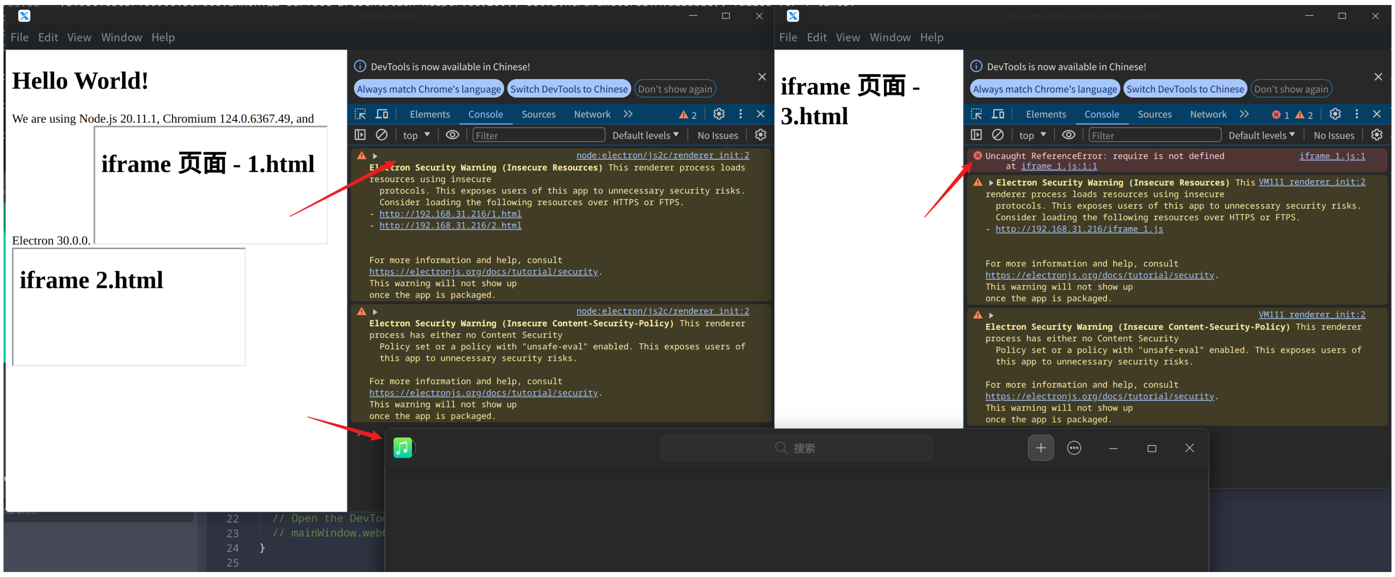
iframe 2.html

iframe 页面 - 3.html

```
Uncaught ReferenceError: require is not defined          iframe_1.js:1
    at iframe_1.js:1:1
▶ Electron Security Warning (Insecure Resources) This VM111 renderer_init:2
renderer process loads resources using insecure
  protocols. This exposes users of this app to unnecessary security risks.
  Consider loading the following resources over HTTPS or FTPS.
- http://192.168.31.216/iframe_1.js


For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
```

并没有成功执行，我们尝试将安全限制都放开，看看能不能执行

- `nodeIntegration: true`
- `contextIsolation: false`
- `sandbox: false`
- `nodeIntegrationInSubFrames: true`


Hello World!
File Edit View Window Help
DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to Chinese
Don't show again
Elements
Console
Sources
Network
top
Filter
Default levels
No Issues
搜索

Hello World!

We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

iframe 页面 - 1.html

iframe 2.html

```
▶                                        node:electron/js2c/renderer_init:2
Electron Security Warning (Insecure Resources) This renderer process loads
resources using insecure
  protocols. This exposes users of this app to unnecessary security risks.
  Consider loading the following resources over HTTPS or FTPS.
- http://192.168.31.216/1.html
- http://192.168.31.216/2.html


For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.

▶                                        node:electron/js2c/renderer_init:2
Electron Security Warning (Insecure Content-Security-Policy) This renderer
process has either no Content Security
  Policy set or a policy with "unsafe-eval" enabled. This exposes users of
  this app to unnecessary security risks.

For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
```

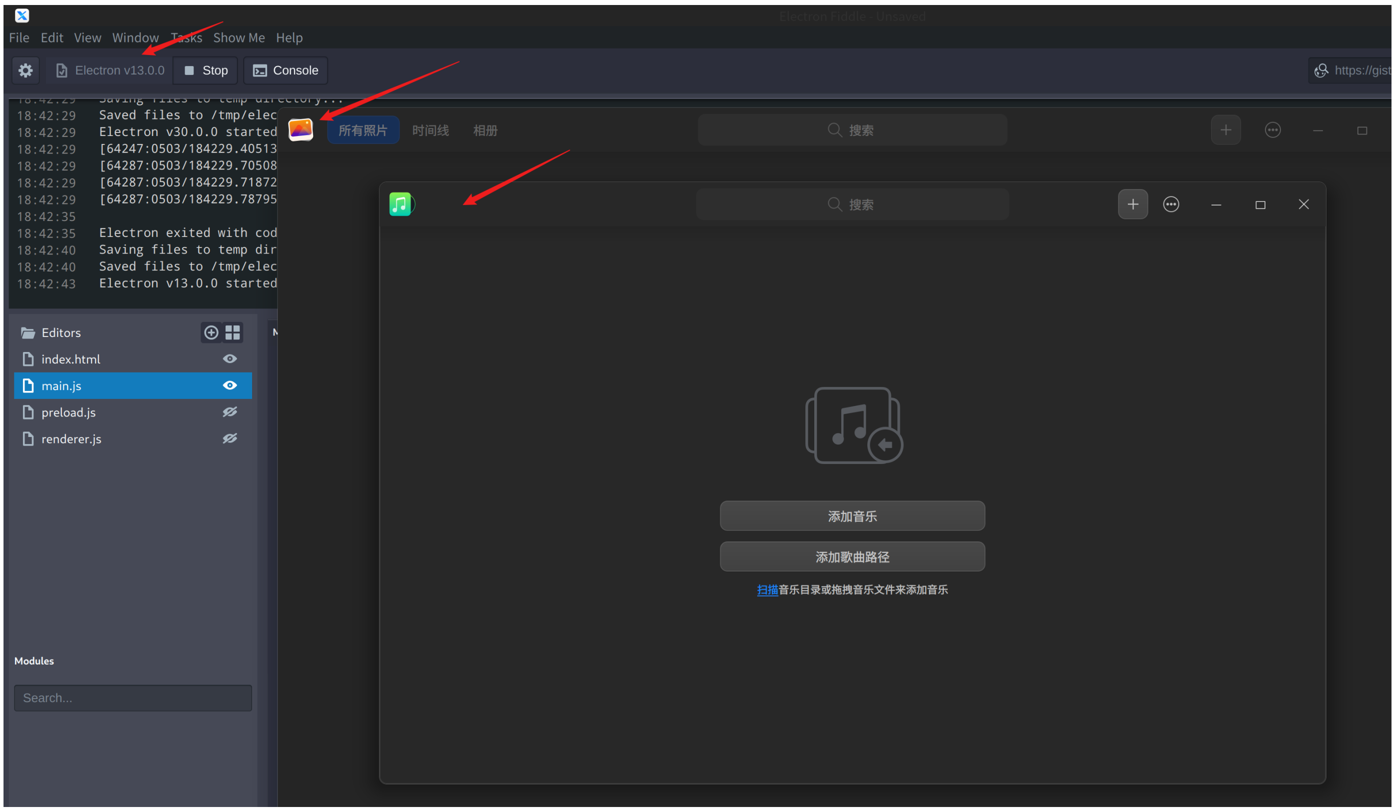
```
// Open the DevTo
// mainWindow.web
}
```

iframe 页面 - 3.html

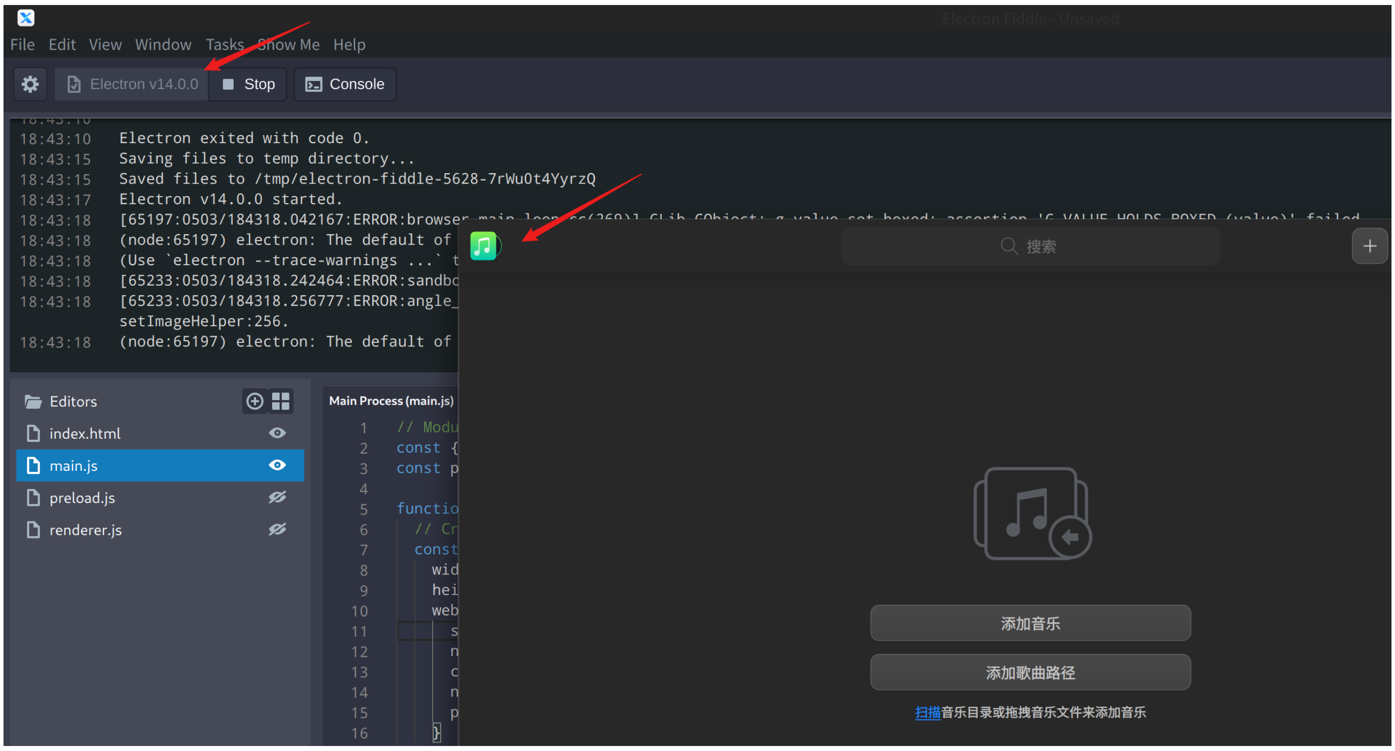
```
Uncaught ReferenceError: require is not defined          iframe_1.js:1
    at iframe_1.js:1:1
▶ Electron Security Warning (Insecure Resources) This VM111 renderer_init:2
renderer process loads resources using insecure
  protocols. This exposes users of this app to unnecessary security risks.
  Consider loading the following resources over HTTPS or FTPS.
- http://192.168.31.216/iframe_1.js


For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.

▶                                             VM111 renderer_init:2
Electron Security Warning (Insecure Content-Security-Policy) This renderer
process has either no Content Security
  Policy set or a policy with "unsafe-eval" enabled. This exposes users of
  this app to unnecessary security risks.

For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
```

这次的结果是 `iframe` 中的 `Node.js` 成功执行了，但是 `iframe + window.open` 打开的窗口执行的 `Node.js` 代码执行失败了

`iframe + window.open` 在 `Electron 14.0` 之前版本是可以执行的



```
18:42:29   Saved files to /tmp/elec
18:42:29   Electron v30.0.0 started
18:42:29   [64247:0503/184229.40513
18:42:29   [64287:0503/184229.70508
18:42:29   [64287:0503/184229.71872
18:42:29   [64287:0503/184229.78795
18:42:35
18:42:35   Electron exited with cod
18:42:40   Saving files to temp dir
18:42:40   Saved files to /tmp/elec
18:42:43   Electron v13.0.0 started
```



```
18:43:10   Electron exited with code 0.
18:43:15   Saving files to temp directory...
18:43:15   Saved files to /tmp/electron-fiddle-5628-7rWu0t4YyrzQ
18:43:17   Electron v14.0.0 started.
18:43:18   [65197:0503/184318.042167:ERROR:browser
18:43:18   (node:65197) electron: The default of
18:43:18   (Use `electron --trace-warnings ...` t
18:43:18   [65233:0503/184318.242464:ERROR:sandbo
18:43:18   [65233:0503/184318.256777:ERROR:angle_
           setImageHelper:256.
18:43:18   (node:65197) electron: The default of
```

因此想要在 `iframe` 中执行 `iframe` ，需要

- `sandbox: false`
- `nodeIntegration: true`
- `contextIsolation: false`

- `nodeIntegrationInSubFrames: true`

缺一不可

这里一定要注意各个安全配置的默认配置，在 `2024-04-25` 左右，我们 `NOP Team` 向 `Electron` 报告了一个安全问题，即虽然官网多个地方强调 `sandbox` 默认在 `Electron 20.0.0` 版本开始默认被设置为 `true` ，官网原话是

> 从 Electron 20 开始，渲染进程默认启用了沙盒，无需进一步配置。
>
> https://www.electronjs.org/zh/docs/latest/tutorial/sandbox

但是经过我们的测试，这个配置选项在目前最新版本 `Electron 30.0.2` 及之前的版本中默认并未设置为 `true`

目前我们已经等了 `Electron` 一周了，还没有在 `Github` 上给我们反馈，所以这篇文章也会在 `Electron` 确认并修复漏洞后发布

4. 测试预加载脚本

官网还提到一个功能，就是 `Preload` 会被注入到每一个 `iframe`

我们在 `Preload` 中创建一个 变量/常量，让 `iframe` 中的脚本 `alert` 弹窗显示出来

`preload.js`

```
window.iframe1 = "iframe_1 has got the value ";
window.iframe_open = "iframe_window_open has got the value";
```

修改 `iframe_1.js` 为

```
if (window.iframe1 !== undefined) {
    alert(window.iframe1);
} else {
    alert("iframe has got nothing");
}
```

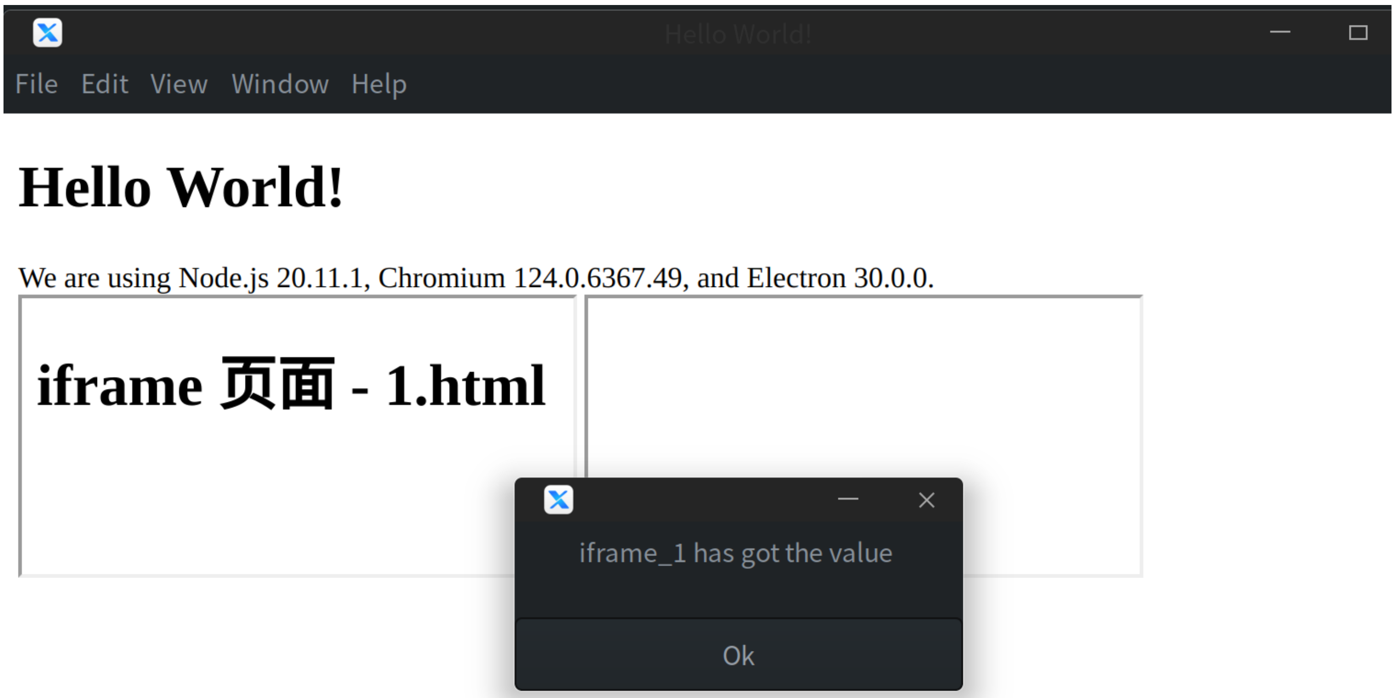
修改 `iframe_2.js` 为

```
if (window.iframe_open !== undefined) {
    alert(window.iframe_open);
} else {
    alert("iframe + window.open has got nothing");
}
```

关闭大部分安全配置

- `sandbox: false`
- `nodeIntegration: true`
- `contextIsolation: false`
- `nodeIntegrationInSubFrames: true`

执行测试

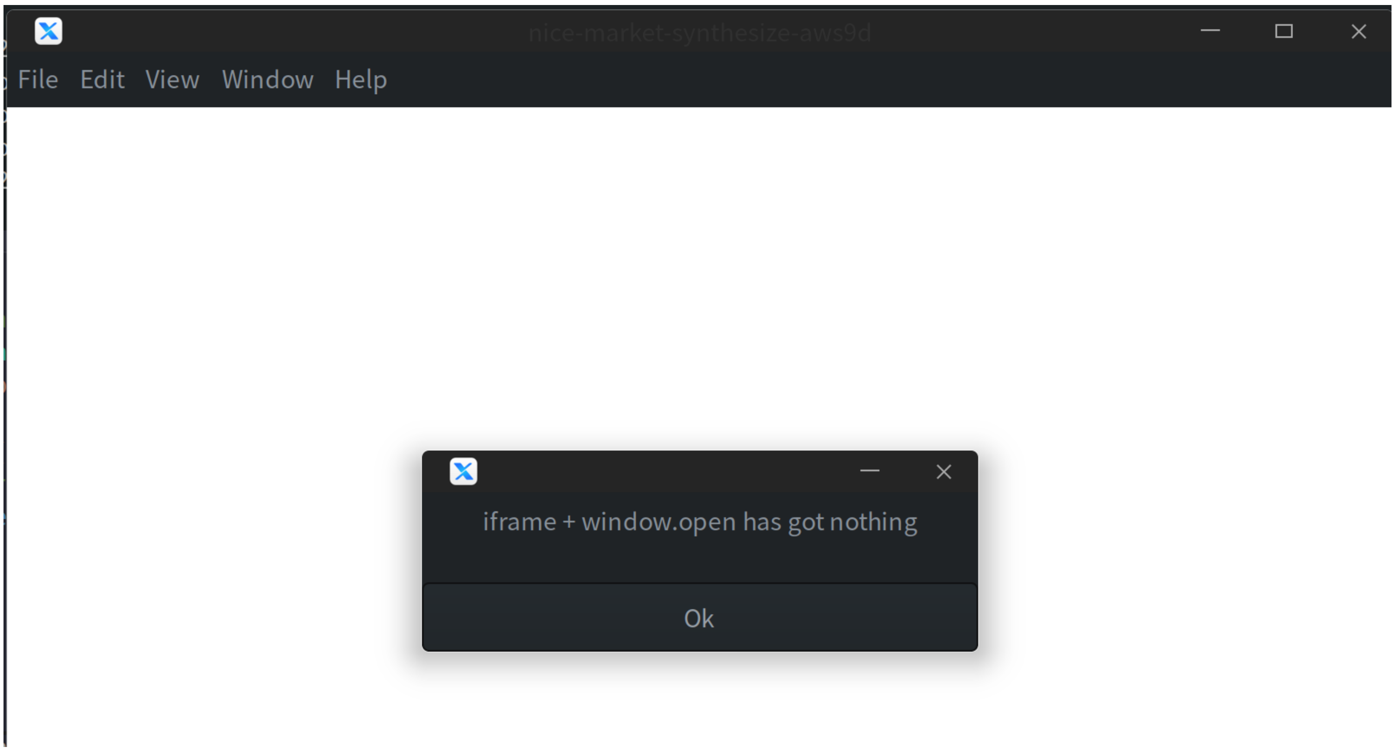


Hello World!

We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

iframe 页面 - 1.html

iframe_1 has got the value



iframe + window.open has got nothing

结果与上面一致， `iframe` 本身成功获取到了 `Preload` 中的内容， `iframe + window.open` 获取失败

`iframe + window.open` 在 `Electron 14.0` 之前版本是可以成功获取的

测试一下不同安全配置下， `iframe` 获取 `preload` 脚本中的内容的情况

经过测试，发现 `nodeIntegrationInSubFrames` 让 `iframe` 获取 `preload` 中暴露的方法和值只和 `nodeIntegrationInSubFrames` 本身有关，不受 `sandbox` 、 `nodeIntegration` 、 `contextIsolation` 影响，当然， `Preload` 暴露方法和值的方式受 `contextIsolation` 影响，当 `contextIsolation: true` 时需要通过 `contextBridge` 进行对外暴露

我这边也测试了一下， `contextIsolation: true` 时，开启 `nodeIntegrationInSubFrames` 后， `iframe` 也只是能获取到 `contextBridge.exposeInMainWorld` 暴露的方法和值，并不能获取到 `Preload` 中直接通过 `window.xxx` 这种形式设置的内容

5. 小结

`nodeIntegrationInSubFrames` 对于 `iframe` 有两个作用，第一个是赋予 `iframe` 执行 `Node.js` 的能力，但是条件比较苛刻，需要同时满足

- `sandbox: false`
- `nodeIntegration: true`
- `contextIsolation: false`
- `nodeIntegrationInSubFrames: true`

第二是让 `iframe` 获取到 `Preload` 中暴露的方法和值，这个功能只需要设置 `nodeIntegrationInSubFrames: true` 即可

经过测试， `nodeIntegrationInSubFrames` 不会让 `preload` 脚本获得额外执行 `Node.js` 的能力

所以这个配置项在一些社区在名字问题上争议比较大，默认人员认为这个名字不是很合理

0x04 测试子窗口

这个子窗口是让我比较疑惑的，我看创建子窗口的时候，子窗口可以有自己的安全配置呀，难道没有设置 `nodeIntegrationInSubFrames` 或设置 `nodeIntegrationInSubFrames: false` 后，即使子窗口设置了渲染进程可以执行 `Node.js` 也不会生效吗?

这听起来就很奇怪，我们测试一下就知道了

我们尝试创建子窗口，在主窗口中设置 `nodeIntegrationInSubFrames: false` ，并在子窗口设置渲染进程可以执行 `Node.js` ，咱们看看到底能不能执行

`main.js`

```
// Modules to control application life and create native browser window
const { app, BrowserWindow } = require('electron')
const path = require('path')

function createPatentWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegrationInSubFrames: false,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  // mainWindow.loadFile('index.html')
  return mainWindow
}

function createChildWindow (parentWindow) {
  const childWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      sandbox: false,
      nodeIntegration: true,
      contextIsolation: false,
      nodeIntegrationInSubFrames: true,
      parent: parentWindow
    }
  })
  return childWindow
}

app.whenReady().then(() => {
  const parentWindow = createPatentWindow()
  parentWindow.loadFile('index.html')

  const childWindow = createChildWindow(parentWindow)
  childWindow.loadFile('child.html')
  // createWindow()


  app.on('activate', function () {
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

app.on('window-all-closed', function () {
  if (process.platform !== 'darwin') app.quit()
})
```

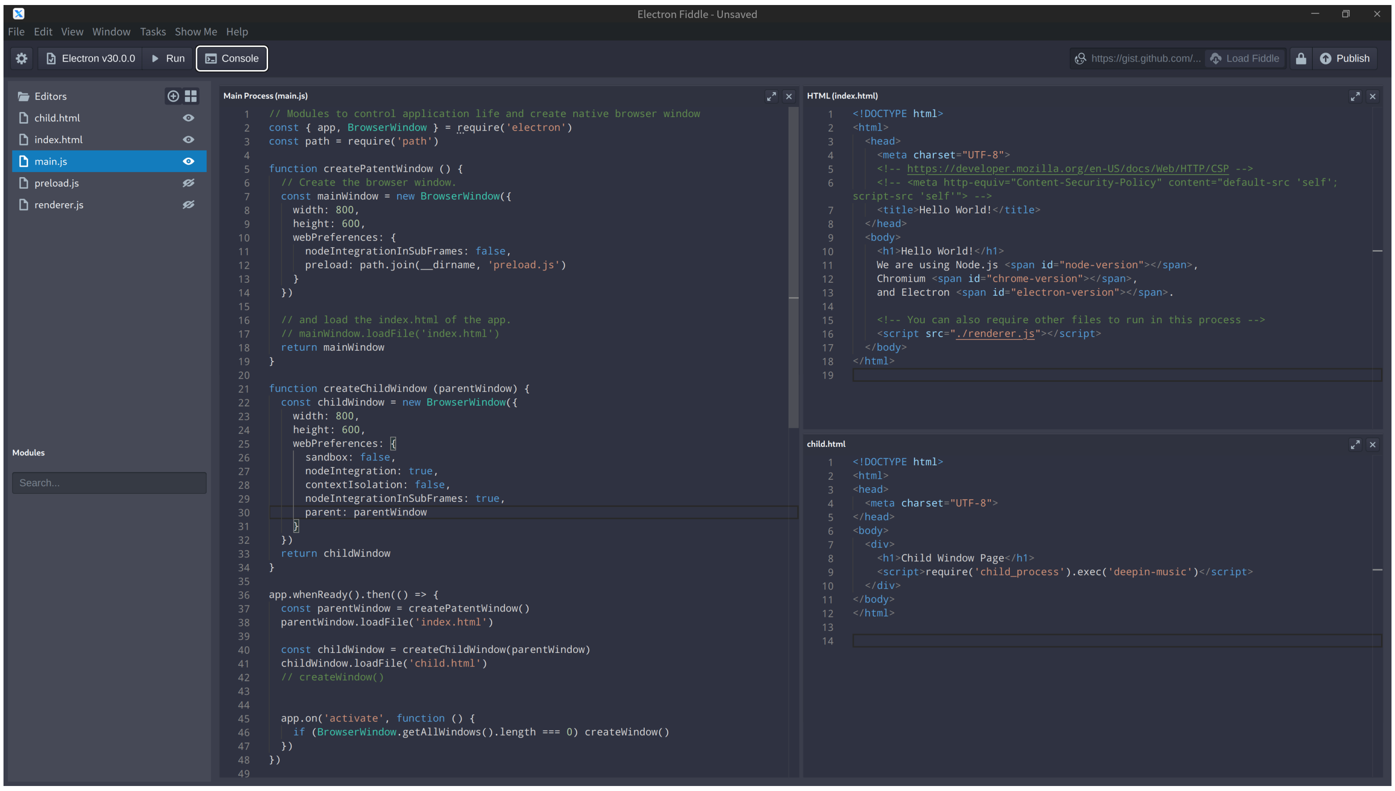
`index.html`

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src
'self'; script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <script src="./renderer.js"></script>
  </body>
</html>
```

`child.html` 这个是子窗口的主页面

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <div>
    <h1>Child Window Page</h1>
    <script>require('child_process').exec('deepin-music')</script>
  </div>
</body>
</html>
```

执行测试



Main Process (main.js)

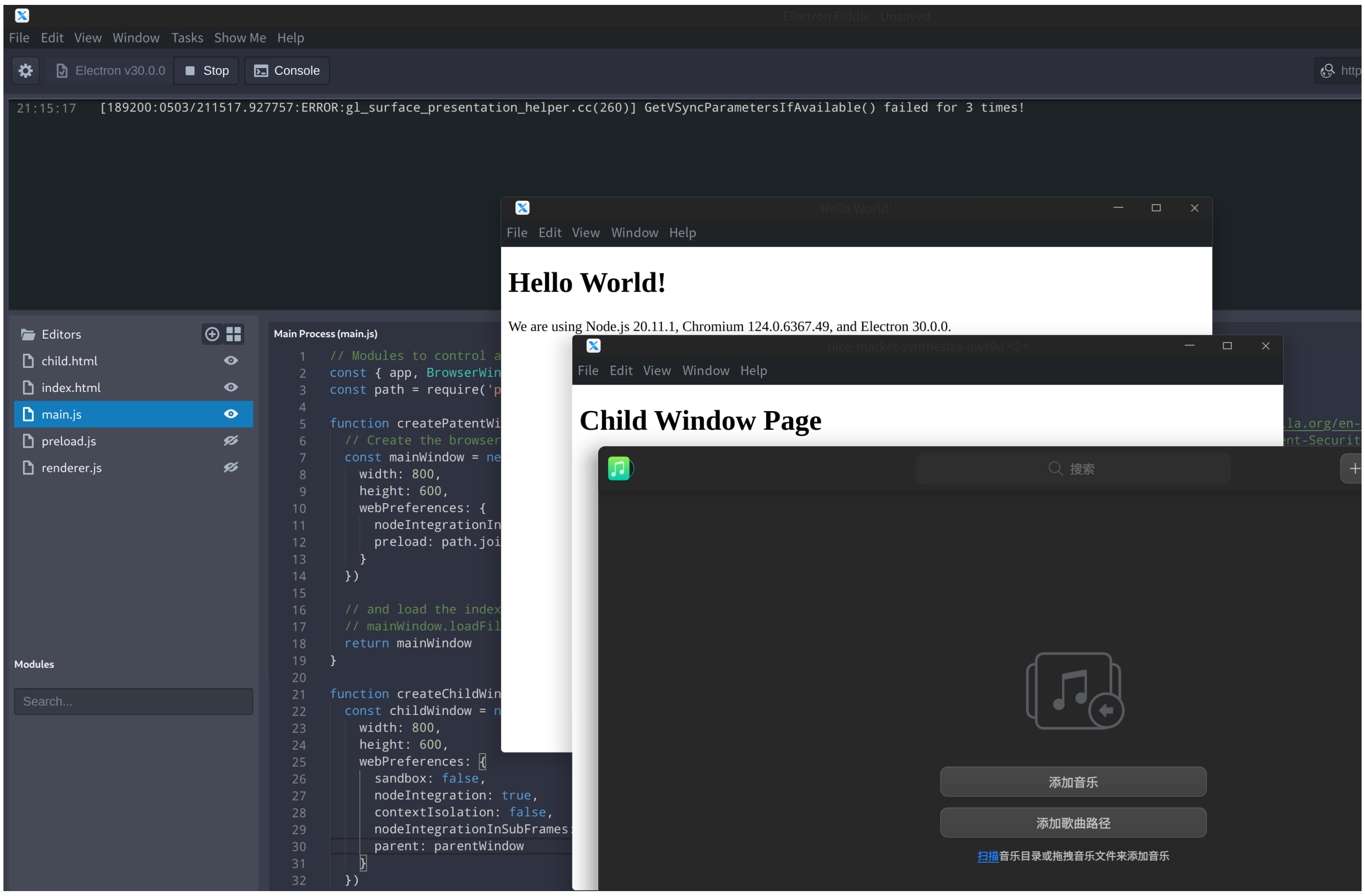
```javascript
// Modules to control application life and create native browser window
const { app, BrowserWindow } = require('electron')
const path = require('path')

function createPatentWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegrationInSubFrames: false,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  // mainWindow.loadFile('index.html')
  return mainWindow
}

function createChildWindow (parentWindow) {
  const childWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      sandbox: false,
      nodeIntegration: true,
      contextIsolation: false,
      nodeIntegrationInSubFrames: true,
      parent: parentWindow
    }
  })
  return childWindow
}

app.whenReady().then(() => {
  const parentWindow = createPatentWindow()
  parentWindow.loadFile('index.html')

  const childWindow = createChildWindow(parentWindow)
  childWindow.loadFile('child.html')
  // createWindow()


  app.on('activate', function () {
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

```

HTML (index.html)

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <script src="./renderer.js"></script>
  </body>
</html>

```

child.html

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <div>
    <h1>Child Window Page</h1>
    <script>require('child_process').exec('deepin-music')</script>
  </div>
</body>
</html>

```



```
21:15:17   [189200:0503/211517.927757:ERROR:gl_surface_presentation_helper.cc(260)] GetVSyncParametersIfAvailable() failed for 3 times!
```

```
// Modules to control a
const { app, BrowserWin
const path = require('p

function createPatentWi
  // Create the browser
  const mainWindow = ne
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegrationIn
      preload: path.joi
    }
  })

  // and load the index
  // mainWindow.loadFil
  return mainWindow
}

function createChildWin
  const childWindow = n
    width: 800,
    height: 600,
    webPreferences: {
      sandbox: false,
      nodeIntegration: true,
      contextIsolation: false,
      nodeIntegrationInSubFrames:
      parent: parentWindow
    }
  })
```

Hello World!

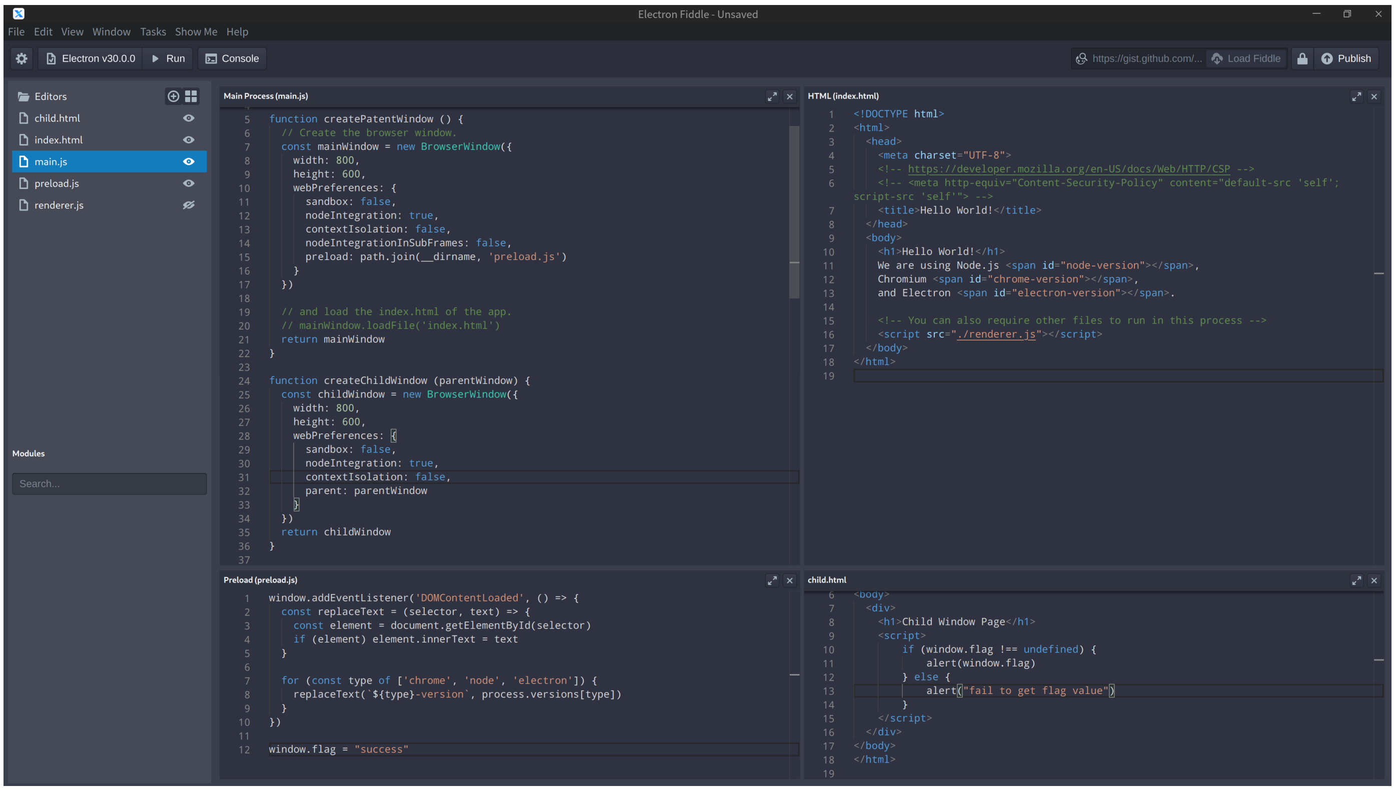
We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

Child Window Page

子窗口可以成功执行 `Node.js`

这样以来， `nodeIntegrationInSubFrames` 对子窗口 `Node.js` 的执行就没有影响了呀，而且经过我的测试，在生命周期方面，关闭父窗口，子窗口并不会跟着关闭

测试一下 `nodeIntegrationInSubFrames: true` 时子窗口是否能够读取父窗口的 `Preload` 中的内容



Main Process (main.js)

```javascript
function createPatentWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      sandbox: false,
      nodeIntegration: true,
      contextIsolation: false,
      nodeIntegrationInSubFrames: false,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  // mainWindow.loadFile('index.html')
  return mainWindow
}

function createChildWindow (parentWindow) {
  const childWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      sandbox: false,
      nodeIntegration: true,
      contextIsolation: false,
      parent: parentWindow
    }
  })
  return childWindow
}

```

HTML (index.html)

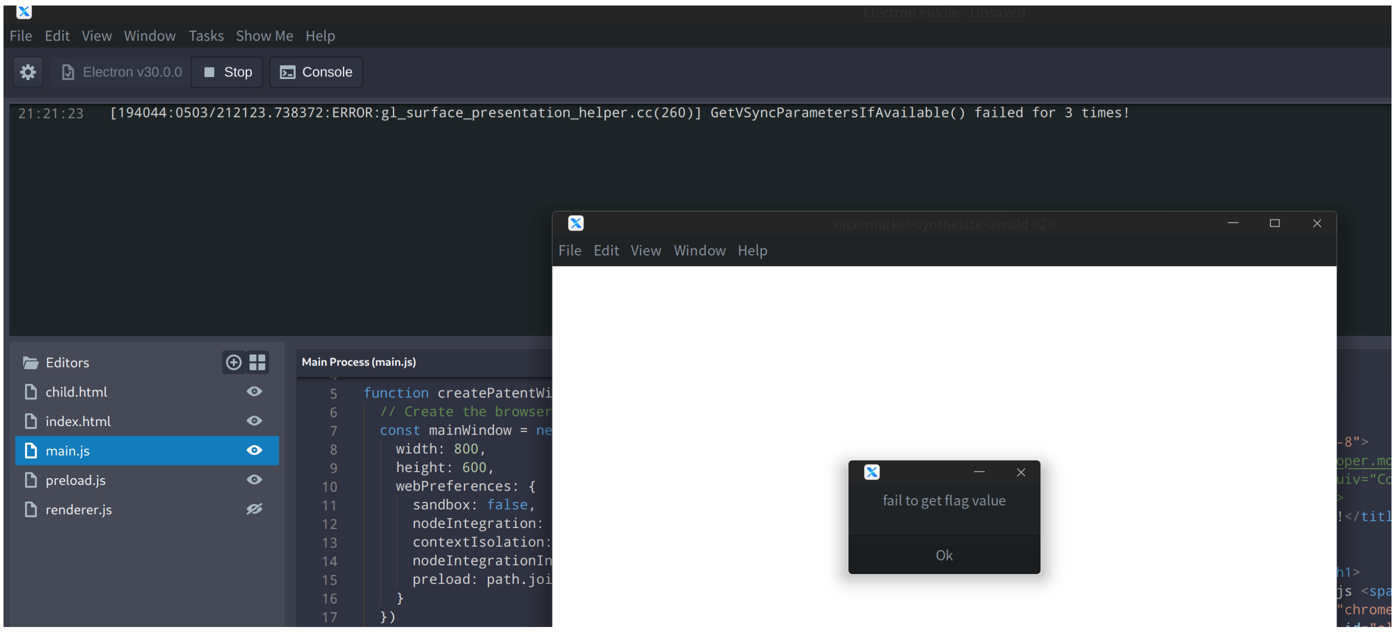
```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <script src="./renderer.js"></script>
  </body>
</html>

```

Preload (preload.js)

```javascript
window.addEventListener('DOMContentLoaded', () => {
  const replaceText = (selector, text) => {
    const element = document.getElementById(selector)
    if (element) element.innerText = text
  }

  for (const type of ['chrome', 'node', 'electron']) {
    replaceText(`${type}-version`, process.versions[type])
  }
})

window.flag = "success"
```

child.html

```html
<body>
  <div>
    <h1>Child Window Page</h1>
    <script>
        if (window.flag !== undefined) {
            alert(window.flag)
        } else {
            alert("fail to get flag value")
        }
    </script>
  </div>
</body>
</html>

```



```
21:21:23   [194044:0503/212123.738372:ERROR:gl_surface_presentation_helper.cc(260)] GetVSyncParametersIfAvailable() failed for 3 times!
```

fail to get flag value

获取失败，看起来官方文档中描述的 `child window` 并不是官方文档其他部分中的 `child window`

> https://www.electronjs.org/docs/latest/api/browser-window#new-browserwindowoptions
>
> https://www.electronjs.org/docs/latest/api/browser-window#parent-and-child-windows

0x05 探索可能的子窗口

既然子窗口不是指主进程创建的窗口之间的父子关系，那么和 `iframe` 比较类似的应该就是 `<webview>` 和 `WebContentsView` 了，还有 HTML 中的 `object` 和 `embed`

> https://www.electronjs.org/zh/docs/latest/tutorial/web-embeds
>
> https://www.electronjs.org/zh/docs/latest/api/webview-tag
>
> https://www.electronjs.org/zh/docs/latest/api/web-contents-view

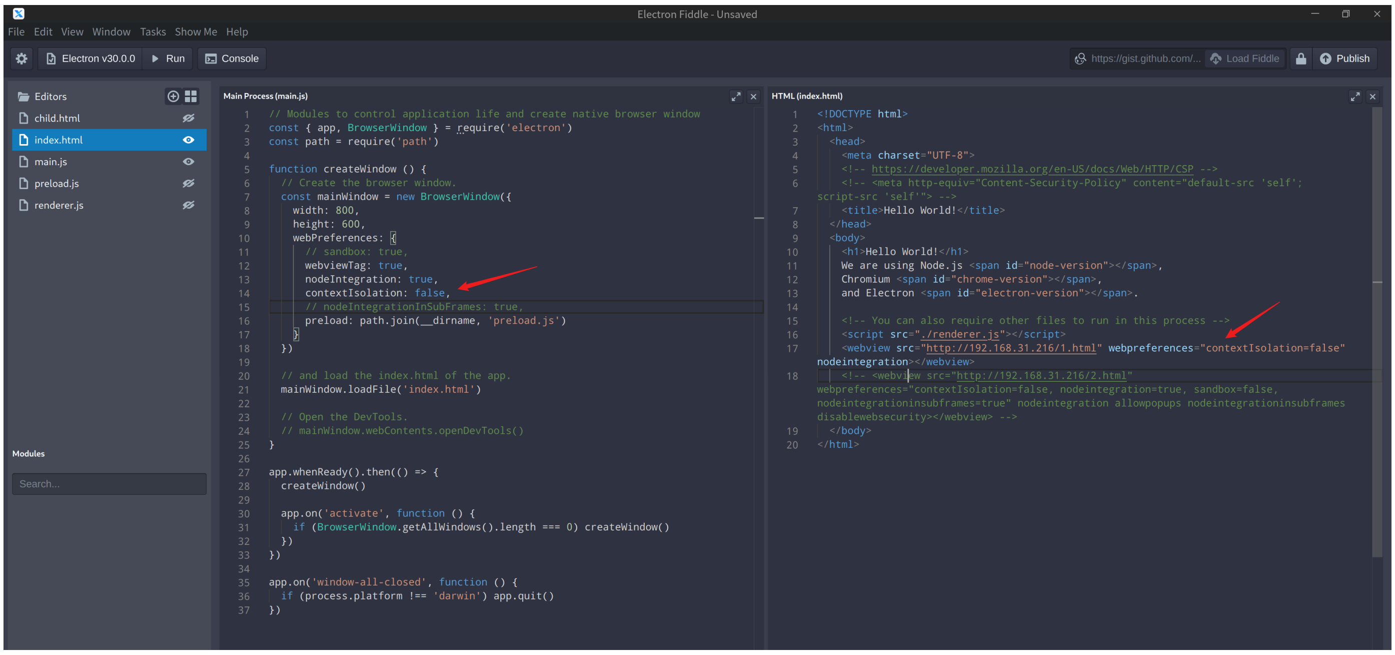
官方是不建议使用 `<webview>` 标签来实现嵌入其他页面的， `WebContentsView` 是 `ELectron 30.0.0` 新添加的功能，用来替代原本的 `BrowserViews`

1. webview 标签

对于 `webview` 标签，在 `Electron >= 5.0` 版本后，默认不允许，使用的话必须在创建父窗口时显式地设置 `webviewTag: true`

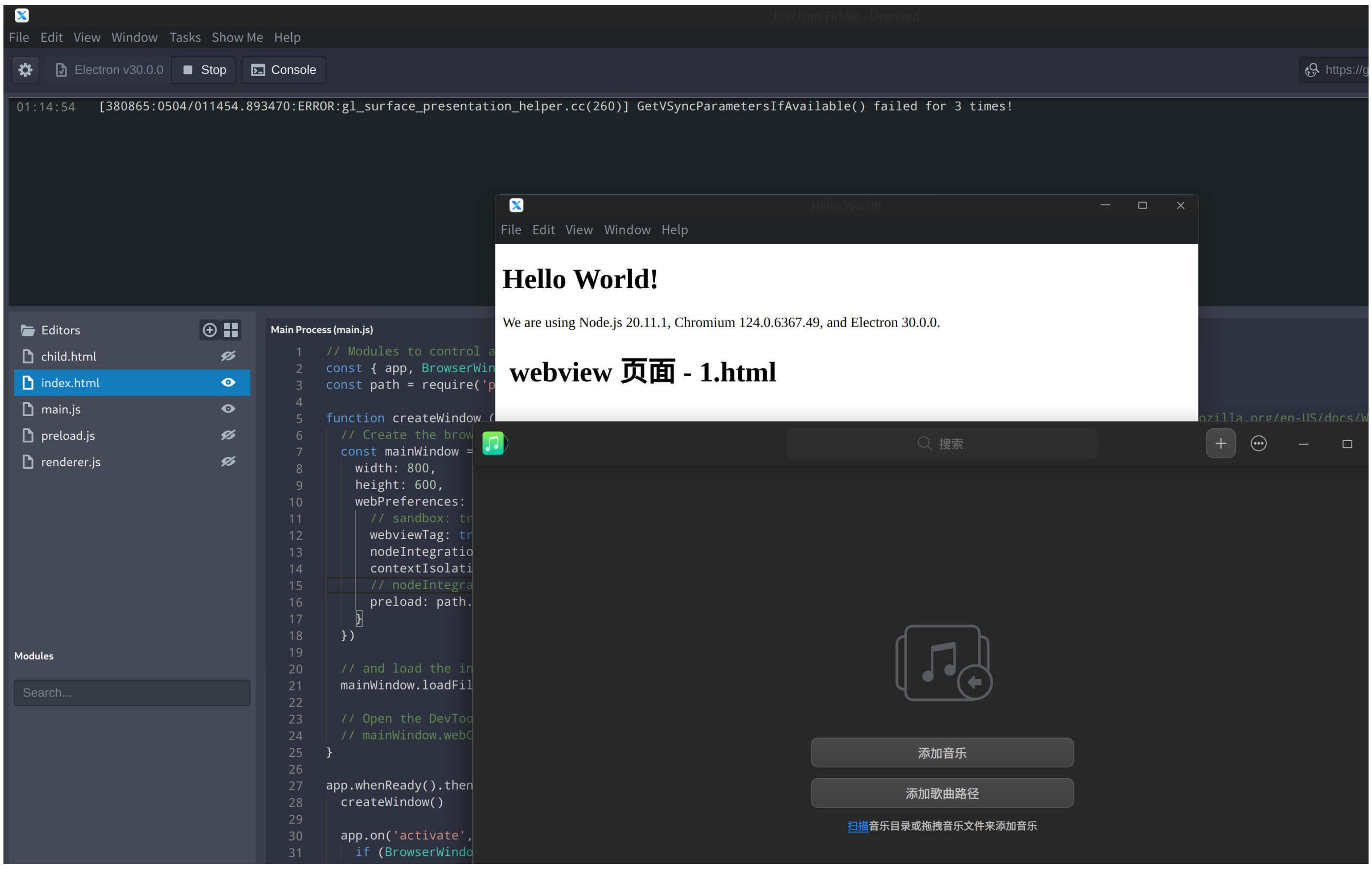
直接使用上面测试 `iframe` 执行 `Node.js` 的服务器即可

经过测试发现， `webview` 标签加载嵌入的内容是否可以执行 `Node.js` 与 `nodeIntegrationInSubFrames` 并不相关，主要与父窗口安全配置以及 `webview` 标签本身配置有关系



```javascript
// Modules to control application life and create native browser window
const { app, BrowserWindow } = require('electron')
const path = require('path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      // sandbox: true,
      webviewTag: true,
      nodeIntegration: true,
      contextIsolation: false,
      // nodeIntegrationInSubFrames: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  // mainWindow.webContents.openDevTools()
}

app.whenReady().then(() => {
  createWindow()

  app.on('activate', function () {
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

app.on('window-all-closed', function () {
  if (process.platform !== 'darwin') app.quit()
})
```

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <script src="./renderer.js"></script>
    <webview src="http://192.168.31.216/1.html" webpreferences="contextIsolation=false"
nodeintegration></webview>
    <!-- <webview src="http://192.168.31.216/2.html"
webpreferences="contextIsolation=false, nodeintegration=true, sandbox=false,
nodeintegrationinsubframes=true" nodeintegration allowpopups nodeintegrationinsubframes
disablewebsecurity></webview> -->
  </body>
</html>
```



```
01:14:54    [380865:0504/011454.893470:ERROR:gl_surface_presentation_helper.cc(260)] GetVSyncParametersIfAvailable() failed for 3 times!
```

```
// Modules to control a
const { app, BrowserWin
const path = require('p

function createWindow (
  // Create the brow
  const mainWindow =
    width: 800,
    height: 600,
    webPreferences:
      // sandbox: tr
      webviewTag: tr
      nodeIntegratio
      contextIsolati
      // nodeIntegra
      preload: path.
    }
  })

  // and load the in
  mainWindow.loadFil

  // Open the DevToo
  // mainWindow.webC
}

app.whenReady().then
  createWindow()

  app.on('activate',
    if (BrowserWindo
```

Hello World!

We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

webview 页面 - 1.html

2. WebContentsView

> https://www.electronjs.org/zh/docs/latest/api/web-contents-view#class-webcontentsview-extends-view

```
const { WebContentsView } = require('electron')

const view = new WebContentsView()
view.webContents.loadURL('https://electronjs.org/')
```

我们尝试在 `baseWindow` 中添加两个 `WebContentsView`，看看 `WebContentsView` 的行为是不是受 `baseWindow` 的 `nodeIntegrationInSubFrames` 参数的影响

`main.js`

```
// Modules to control application life and create native browser window
const { app, BrowserWindow } = require('electron')
const { BaseWindow, WebContentsView } = require('electron')

app.whenReady().then(() => {
  const win = new BaseWindow({ width: 800, height: 400 , nodeIntegration: 
true, contextIsolation: false, sandbox: false, nodeIntegrationInSubFrames: 
true})

  const view1 = new WebContentsView()
  win.contentView.addChildView(view1)
  view1.webContents.loadURL('http://192.168.31.216/1.html')
  view1.setBounds({ x: 0, y: 0, width: 400, height: 400, nodeIntegration: 
true, contextIsolation: false, sandbox: false})

  const view2 = new WebContentsView()
  win.contentView.addChildView(view2)
  view2.webContents.loadURL('http://192.168.31.216/2.html')
  view2.setBounds({ x: 400, y: 0, width: 400, height: 400, nodeIntegration: 
true, contextIsolation: false, sandbox: false,nodeIntegrationInSubFrames: 
true })

  app.on('activate', function () {
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

app.on('window-all-closed', function () {
  if (process.platform !== 'darwin') app.quit()
})
```

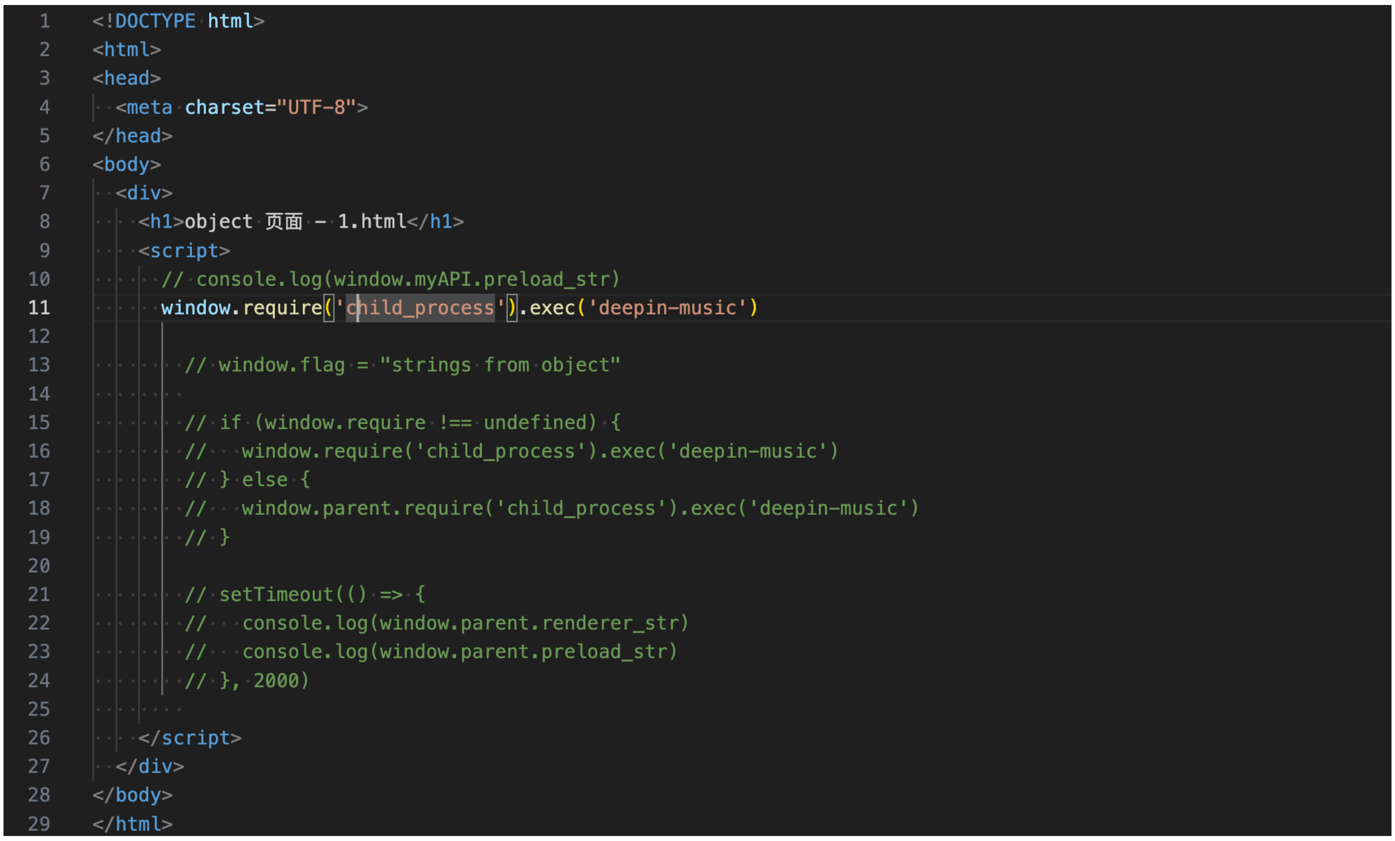
但比较遗憾的是，没有明确在官网找到更多的信息，尝试了过后也没有发现可以执行 `Node.js` 的，所以也就没有办法测试 `nodeIntegrationInSubFrames` 这个参数了

3. object

1) embed 服务器

`object` 远程加载页面内容

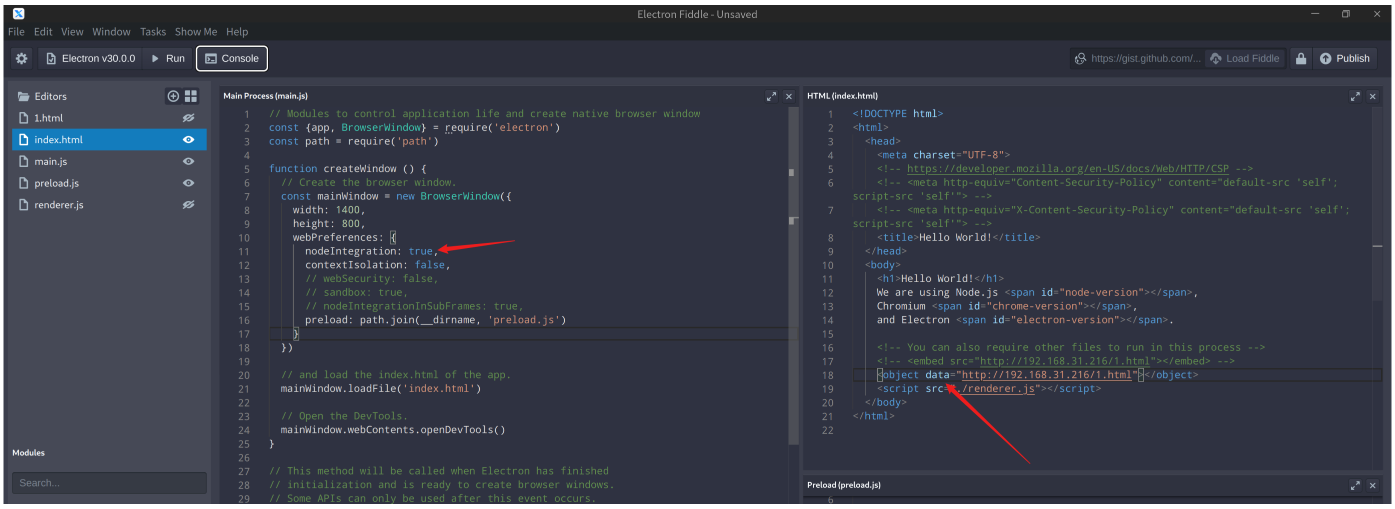
`1.html`



```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <div>
    <h1>object 页面 - 1.html</h1>
    <script>
      // console.log(window.myAPI.preload_str)
      window.require('child_process').exec('deepin-music')

        // window.flag = "strings from object"

        // if (window.require !== undefined) {
        //   window.require('child_process').exec('deepin-music')
        // } else {
        //   window.parent.require('child_process').exec('deepin-music')
        // }

        // setTimeout(() => {
        //   console.log(window.parent.renderer_str)
        //   console.log(window.parent.preload_str)
        // }, 2000)

    </script>
  </div>
</body>
</html>
```

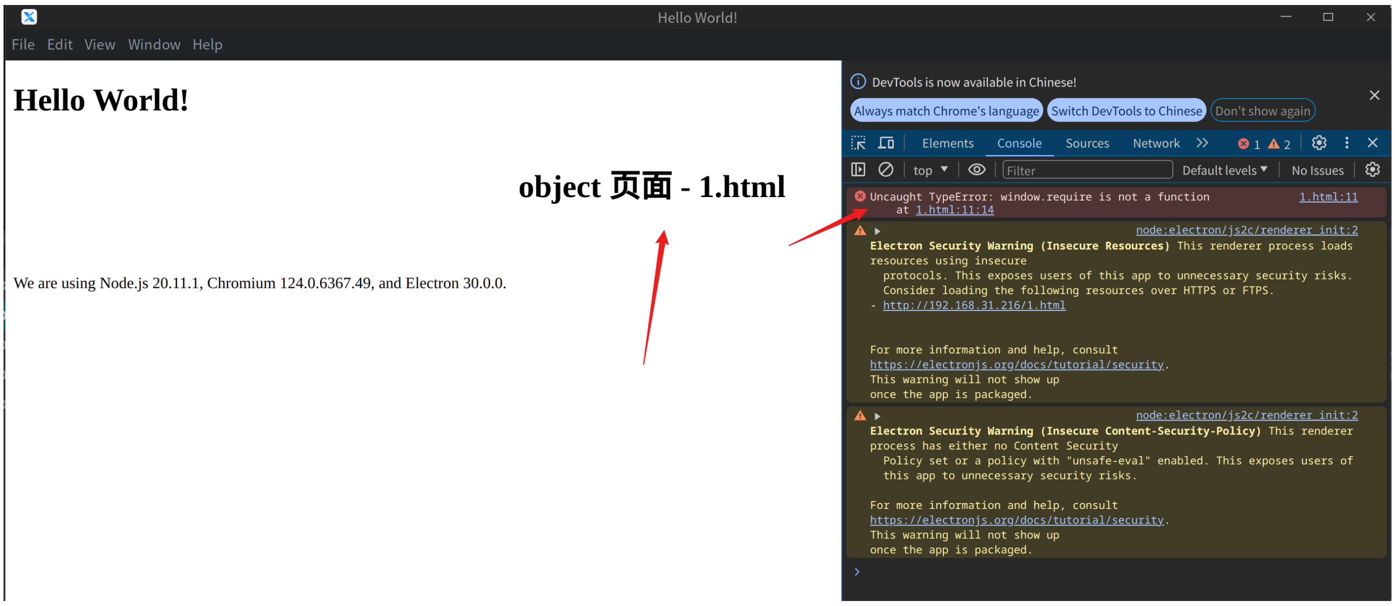
2) 测试执行 Node.js

开启 `nodeIntegration` ，关闭上下文隔离进行测试



```javascript
// Modules to control application life and create native browser window
const {app, BrowserWindow} = require('electron')
const path = require('path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 1400,
    height: 800,
    webPreferences: {
      nodeIntegration: true,
      contextIsolation: false,
      // webSecurity: false,
      // sandbox: true,
      // nodeIntegrationInSubFrames: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  mainWindow.webContents.openDevTools()
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
```

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <!-- <meta http-equiv="X-Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <!-- <embed src="http://192.168.31.216/1.html"></embed> -->
    <object data="http://192.168.31.216/1.html"></object>
    <script src="./renderer.js"></script>
  </body>
</html>
```


Hello World!
File Edit View Window Help
DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to Chinese
Don't show again
Elements
Console
Sources
Network
top
Filter
Default levels
No Issues

Hello World!

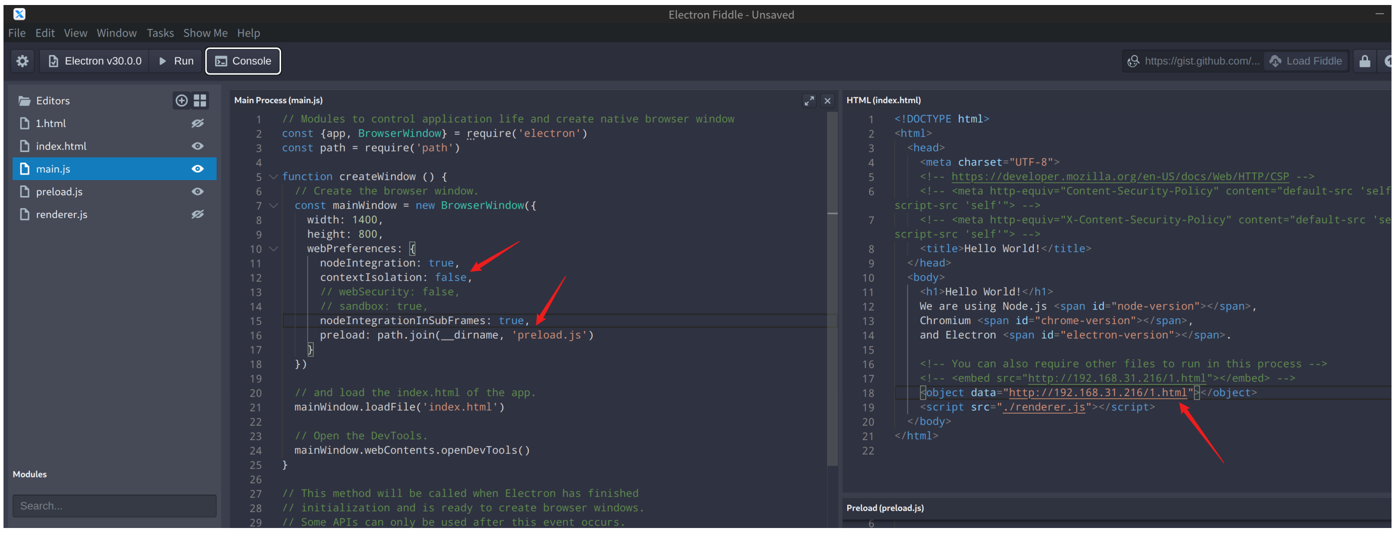
object 页面 - 1.html

We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

```
Uncaught TypeError: window.require is not a function                1.html:11
    at 1.html:11:14
                                          node:electron/js2c/renderer_init:2
Electron Security Warning (Insecure Resources) This renderer process loads
resources using insecure
  protocols. This exposes users of this app to unnecessary security risks.
  Consider loading the following resources over HTTPS or FTPS.
- http://192.168.31.216/1.html


For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
                                          node:electron/js2c/renderer_init:2
Electron Security Warning (Insecure Content-Security-Policy) This renderer
process has either no Content Security
  Policy set or a policy with "unsafe-eval" enabled. This exposes users of
  this app to unnecessary security risks.

For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
```

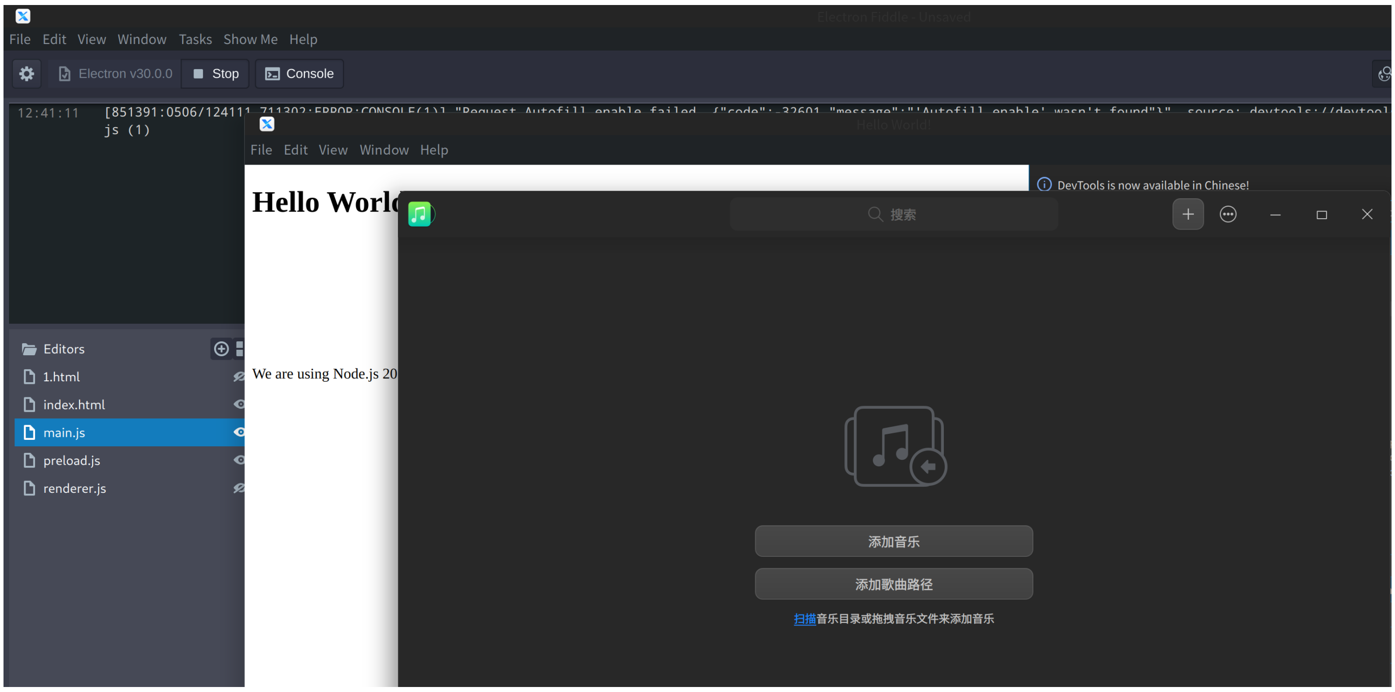
页面正常嵌入了，但是 `Node.js` 代码没有执行

添加 `nodeIntegrationInSubFrames: true`



```
// Modules to control application life and create native browser window
const {app, BrowserWindow} = require('electron')
const path = require('path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 1400,
    height: 800,
    webPreferences: {
      nodeIntegration: true,
      contextIsolation: false,
      // webSecurity: false,
      // sandbox: true,
      nodeIntegrationInSubFrames: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  mainWindow.webContents.openDevTools()
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self
script-src 'self'"> -->
    <!-- <meta http-equiv="X-Content-Security-Policy" content="default-src 'se
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <!-- <embed src="http://192.168.31.216/1.html"></embed> -->
    <object data="http://192.168.31.216/1.html"></object>
    <script src="./renderer.js"></script>
  </body>
</html>
```


File Edit View Window Tasks Show Me Help
Electron v30.0.0
Stop
Console
Editors
1.html
index.html
main.js
preload.js
renderer.js
File Edit View Window Help
Hello World
We are using Node.js 20
DevTools is now available in Chinese!
搜索
添加音乐
添加歌曲路径
扫描音乐目录或拖拽音乐文件来添加音乐

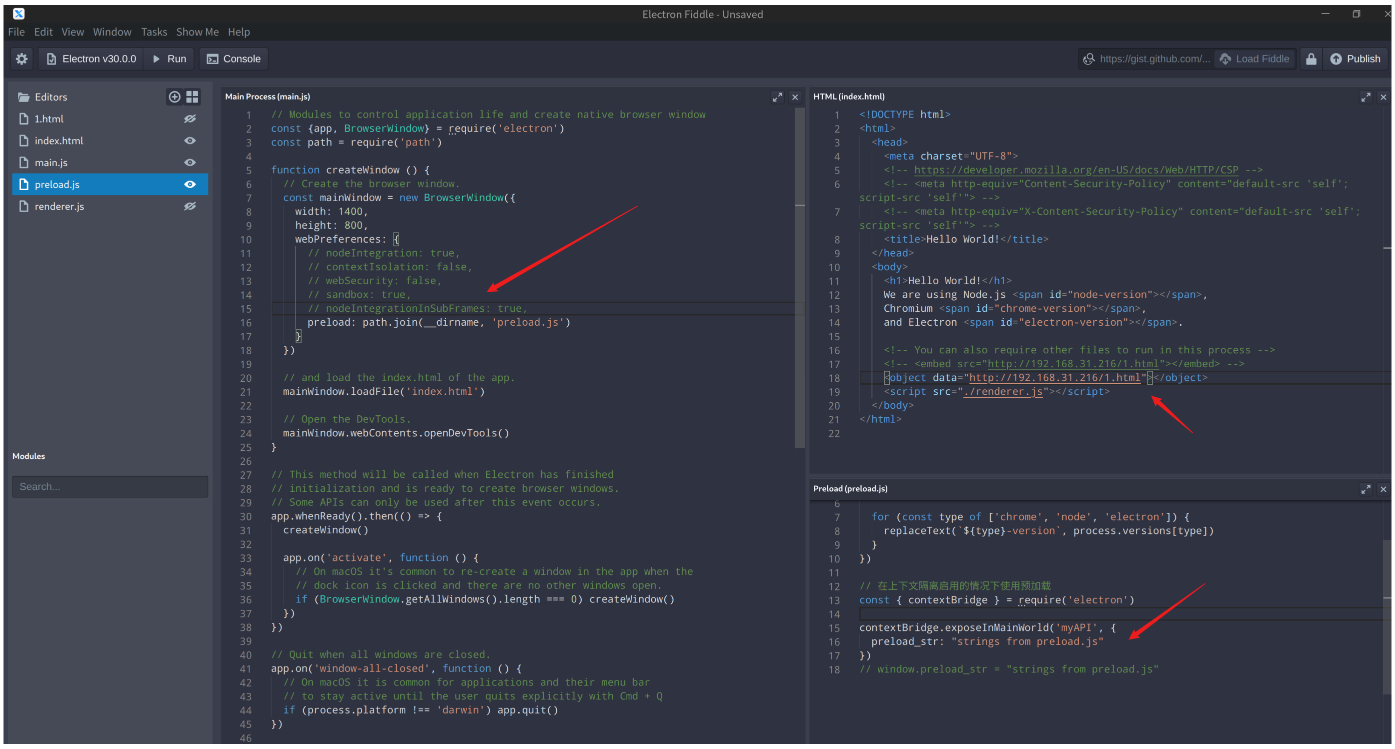
```
12:41:11   [851391:0506/124111.711302:ERROR:CONSOLE(1)] "Request Autofill.enable failed. {"code":-32601,"message":"'Autofill.enable' wasn't found"}", source: devtools://devtool
           js (1)
```

成功执行，经过测试， `iframe` 执行 `Node.js` 的条件与 `iframe` 一致

3) 测试预加载脚本

修改 `object` 服务器内容，获取并控制台输出预加载脚本暴露给渲染进程的值

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <div>
    <h1>object 页面 - 1.html</h1>
    <script>
      console.log(window.myAPI.preload_str)
      // window.require('child_process').exec('deepin-music')

        // window.flag = "strings from object"

        // if (window.require !== undefined) {
        //   window.require('child_process').exec('deepin-music')
        // } else {
        //   window.parent.require('child_process').exec('deepin-music')
        // }

        // setTimeout(() => {
        //   console.log(window.parent.renderer_str)
        //   console.log(window.parent.preload_str)
        // }, 2000)

    </script>
  </div>
</body>
</html>
```



Main Process (main.js)

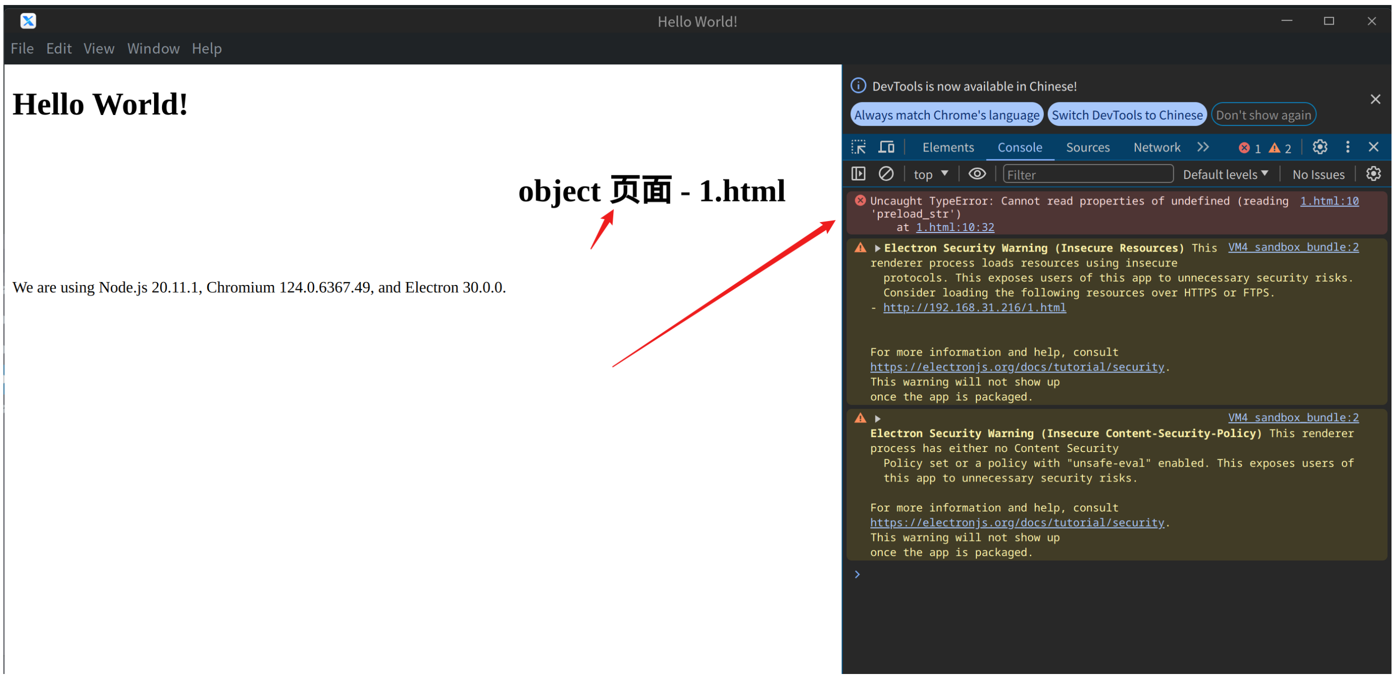
```js
// Modules to control application life and create native browser window
const {app, BrowserWindow} = require('electron')
const path = require('path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 1400,
    height: 800,
    webPreferences: {
      // nodeIntegration: true,
      // contextIsolation: false,
      // webSecurity: false,
      // sandbox: true,
      // nodeIntegrationInSubFrames: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  mainWindow.webContents.openDevTools()
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.whenReady().then(() => {
  createWindow()

  app.on('activate', function () {
    // On macOS it's common to re-create a window in the app when the
    // dock icon is clicked and there are no other windows open.
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

// Quit when all windows are closed.
app.on('window-all-closed', function () {
  // On macOS it is common for applications and their menu bar
  // to stay active until the user quits explicitly with Cmd + Q
  if (process.platform !== 'darwin') app.quit()
})
```

HTML (index.html)

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <!-- <meta http-equiv="X-Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <!-- <embed src="http://192.168.31.216/1.html"></embed> -->
    <object data="http://192.168.31.216/1.html"></object>
    <script src="./renderer.js"></script>
  </body>
</html>
```

Preload (preload.js)

```js

  for (const type of ['chrome', 'node', 'electron']) {
    replaceText(`${type}-version`, process.versions[type])
  }
})

// 在上下文隔离启用的情况下使用预加载
const { contextBridge } = require('electron')

contextBridge.exposeInMainWorld('myAPI', {
  preload_str: "strings from preload.js"
})
// window.preload_str = "strings from preload.js"
```


Hello World!
File Edit View Window Help
DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to Chinese
Don't show again
Elements
Console
Sources
Network
1
2
top
Filter
Default levels
No Issues

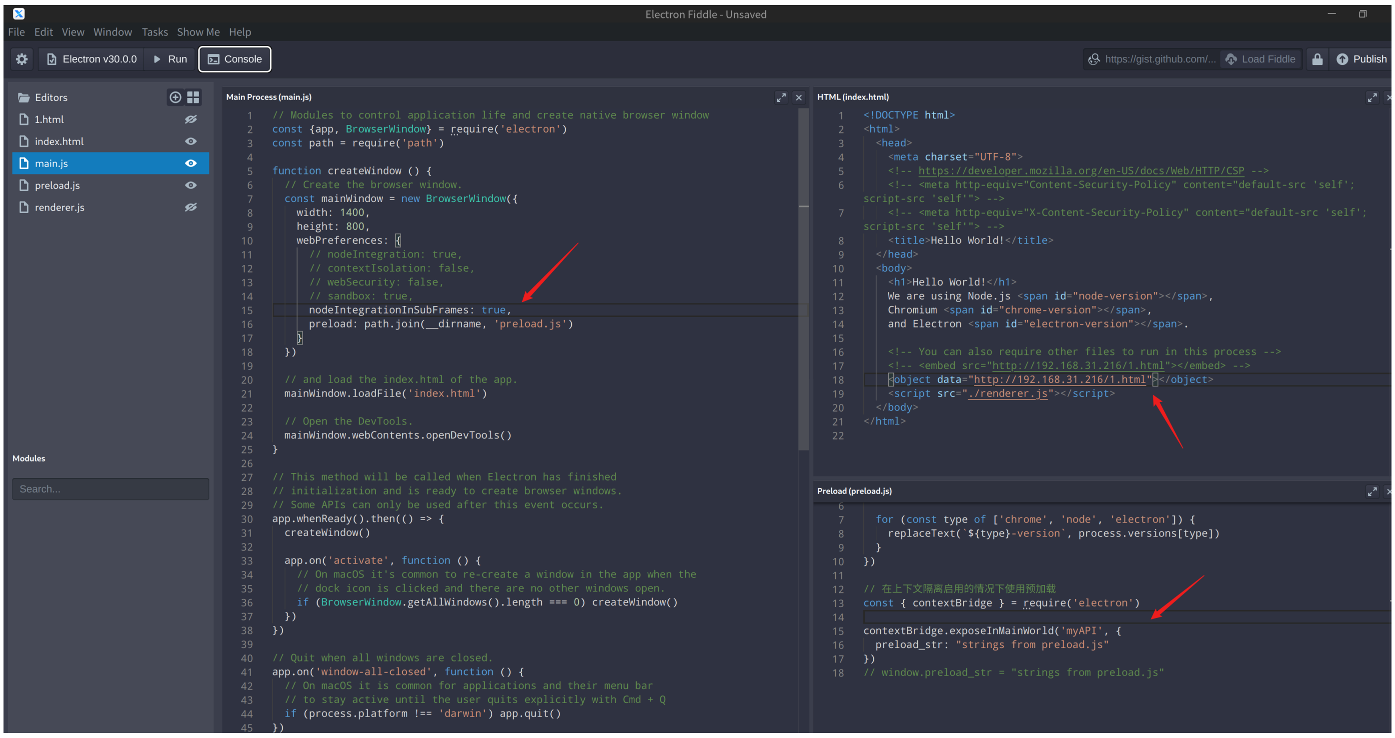
Hello World!

object 页面 - 1.html

We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

```
Uncaught TypeError: Cannot read properties of undefined (reading      1.html:10
'preload_str')
    at 1.html:10:32

▸ Electron Security Warning (Insecure Resources) This   VM4 sandbox bundle:2
renderer process loads resources using insecure
  protocols. This exposes users of this app to unnecessary security risks.
  Consider loading the following resources over HTTPS or FTPS.
- http://192.168.31.216/1.html


For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.

▸                                                        VM4 sandbox bundle:2
Electron Security Warning (Insecure Content-Security-Policy) This renderer
process has either no Content Security
  Policy set or a policy with "unsafe-eval" enabled. This exposes users of
  this app to unnecessary security risks.

For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
```

设置 `nodeIntegrationInSubFrames: true`



Main Process (main.js)

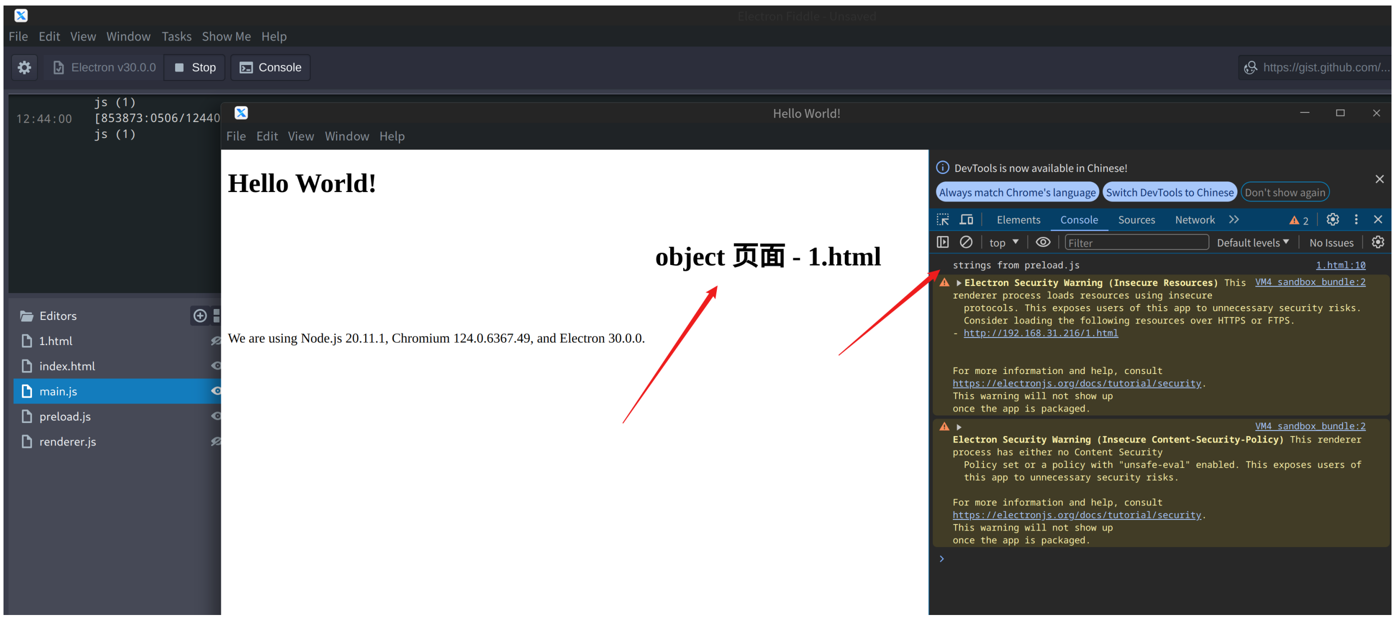
```
// Modules to control application life and create native browser window
const {app, BrowserWindow} = require('electron')
const path = require('path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 1400,
    height: 800,
    webPreferences: {
      // nodeIntegration: true,
      // contextIsolation: false,
      // webSecurity: false,
      // sandbox: true,
      nodeIntegrationInSubFrames: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  mainWindow.webContents.openDevTools()
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.whenReady().then(() => {
  createWindow()

  app.on('activate', function () {
    // On macOS it's common to re-create a window in the app when the
    // dock icon is clicked and there are no other windows open.
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

// Quit when all windows are closed.
app.on('window-all-closed', function () {
  // On macOS it is common for applications and their menu bar
  // to stay active until the user quits explicitly with Cmd + Q
  if (process.platform !== 'darwin') app.quit()
})
```

HTML (index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <!-- <meta http-equiv="X-Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <!-- <embed src="http://192.168.31.216/1.html"></embed> -->
    <object data="http://192.168.31.216/1.html"></object>
    <script src="./renderer.js"></script>
  </body>
</html>
```

Preload (preload.js)

```

  for (const type of ['chrome', 'node', 'electron']) {
    replaceText(`${type}-version`, process.versions[type])
  }
})

// 在上下文隔离启用的情况下使用预加载
const { contextBridge } = require('electron')

contextBridge.exposeInMainWorld('myAPI', {
  preload_str: "strings from preload.js"
})
// window.preload_str = "strings from preload.js"
```


Electron Fiddle - Unsaved
File Edit View Window Tasks Show Me Help
Electron v30.0.0
Stop
Console
https://gist.github.com/...
js (1)
12:44:00 [853873:0506/12440
js (1)
Editors
1.html
index.html
main.js
preload.js
renderer.js
Hello World!
File Edit View Window Help
object 页面 - 1.html
DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to Chinese
Don't show again
Elements Console Sources Network
2
top
Filter
Default levels
No Issues

Hello World!

We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

```
strings from preload.js                                            1.html:10
⚠ ▸ Electron Security Warning (Insecure Resources) This   VM4 sandbox bundle:2
renderer process loads resources using insecure
   protocols. This exposes users of this app to unnecessary security risks.
   Consider loading the following resources over HTTPS or FTPS.
 - http://192.168.31.216/1.html


For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
⚠ ▸                                                      VM4 sandbox bundle:2
Electron Security Warning (Insecure Content-Security-Policy) This renderer
process has either no Content Security
   Policy set or a policy with "unsafe-eval" enabled. This exposes users of
   this app to unnecessary security risks.

For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
>
```

成功获取到预加载脚本暴露给渲染页面的内容

4) 小结

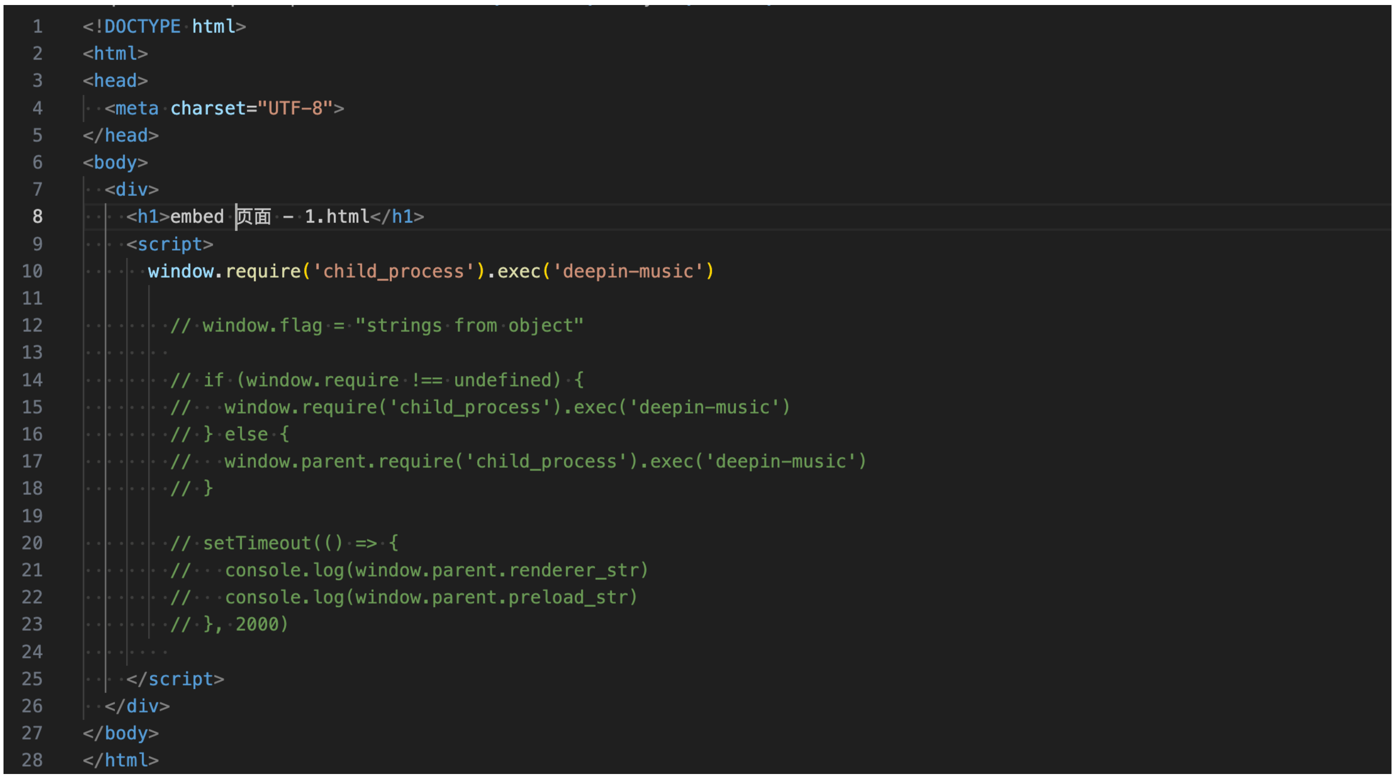
`nodeIntegrationInSubFrames` 对 `object` 的影响与 `iframe` 一致

4. embed

1) embed 服务器

`embed` 远程加载页面内容

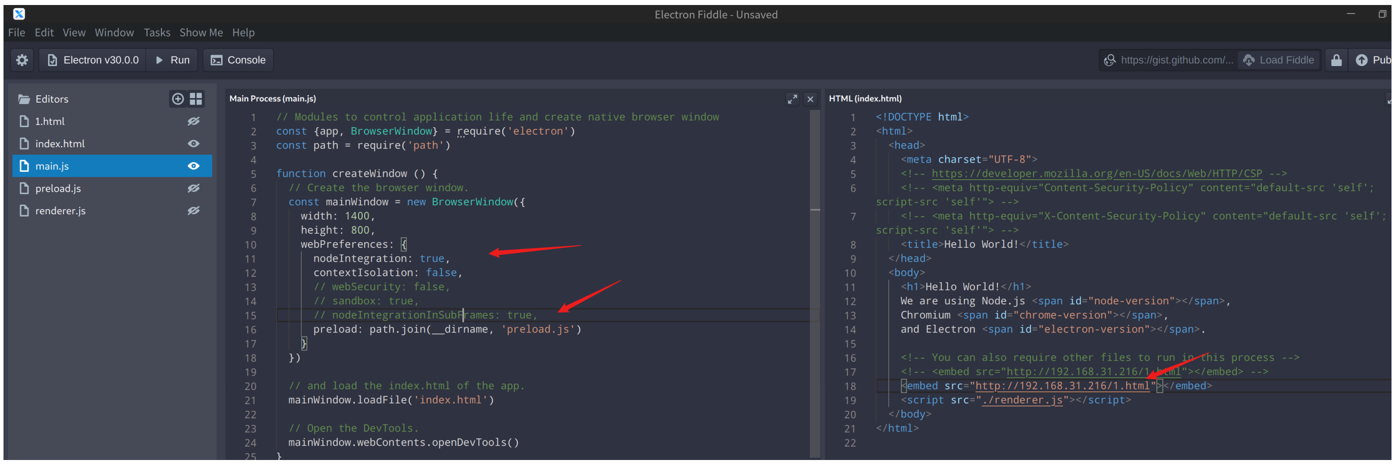
`1.html`



```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <div>
    <h1>embed 页面 - 1.html</h1>
    <script>
      window.require('child_process').exec('deepin-music')

      // window.flag = "strings from object"

      // if (window.require !== undefined) {
      //   window.require('child_process').exec('deepin-music')
      // } else {
      //   window.parent.require('child_process').exec('deepin-music')
      // }

      // setTimeout(() => {
      //   console.log(window.parent.renderer_str)
      //   console.log(window.parent.preload_str)
      // }, 2000)

    </script>
  </div>
</body>
</html>
```

2) 测试执行 Node.js

开启 `nodeIntegration` ，关闭上下文隔离进行测试

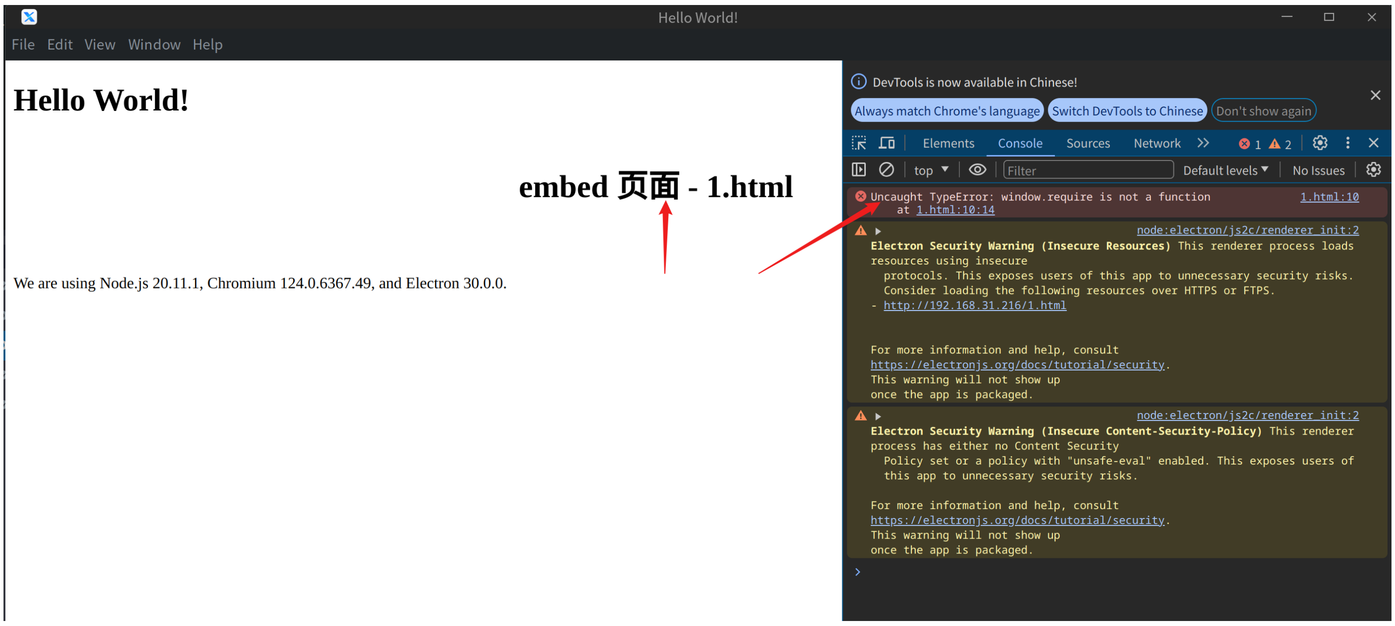

Electron Fiddle - Unsaved
File Edit View Window Tasks Show Me Help
Electron v30.0.0
Run
Console
https://gist.github.com/...
Load Fiddle
Pub
Editors
1.html
index.html
main.js
preload.js
renderer.js

Main Process (main.js)

```javascript
// Modules to control application life and create native browser window
const {app, BrowserWindow} = require('electron')
const path = require('path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 1400,
    height: 800,
    webPreferences: {
      nodeIntegration: true,
      contextIsolation: false,
      // webSecurity: false,
      // sandbox: true,
      // nodeIntegrationInSubFrames: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  mainWindow.webContents.openDevTools()
}
```

HTML (index.html)

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <!-- <meta http-equiv="X-Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <!-- <embed src="http://192.168.31.216/1.html"></embed> -->
    <embed src="http://192.168.31.216/1.html"></embed>
    <script src="./renderer.js"></script>
  </body>
</html>

```


Hello World!
File Edit View Window Help
DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to Chinese
Don't show again
Elements
Console
Sources
Network
top
Filter
Default levels
No Issues

Hello World!

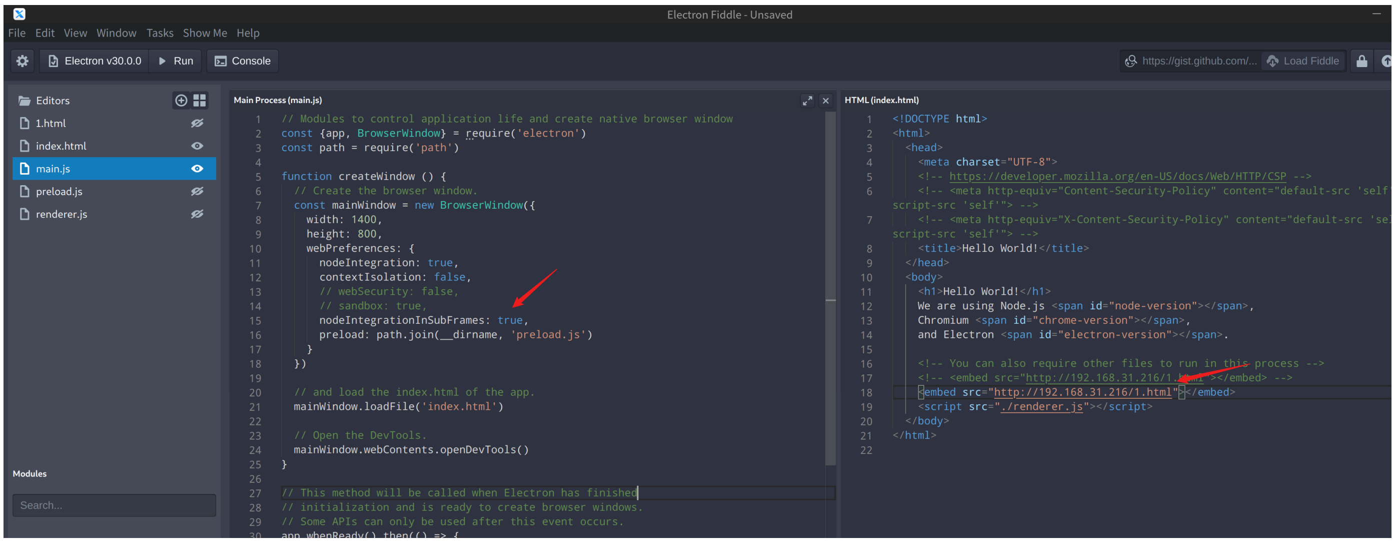
embed 页面 - 1.html

We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

```
Uncaught TypeError: window.require is not a function                1.html:10
    at 1.html:10:14
                                               node:electron/js2c/renderer_init:2
Electron Security Warning (Insecure Resources) This renderer process loads
resources using insecure
  protocols. This exposes users of this app to unnecessary security risks.
  Consider loading the following resources over HTTPS or FTPS.
- http://192.168.31.216/1.html


For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
                                               node:electron/js2c/renderer_init:2
Electron Security Warning (Insecure Content-Security-Policy) This renderer
process has either no Content Security
  Policy set or a policy with "unsafe-eval" enabled. This exposes users of
  this app to unnecessary security risks.

For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
```

页面正常嵌入了，但是 `Node.js` 代码没有执行

添加 `nodeIntegrationInSubFrames: true`

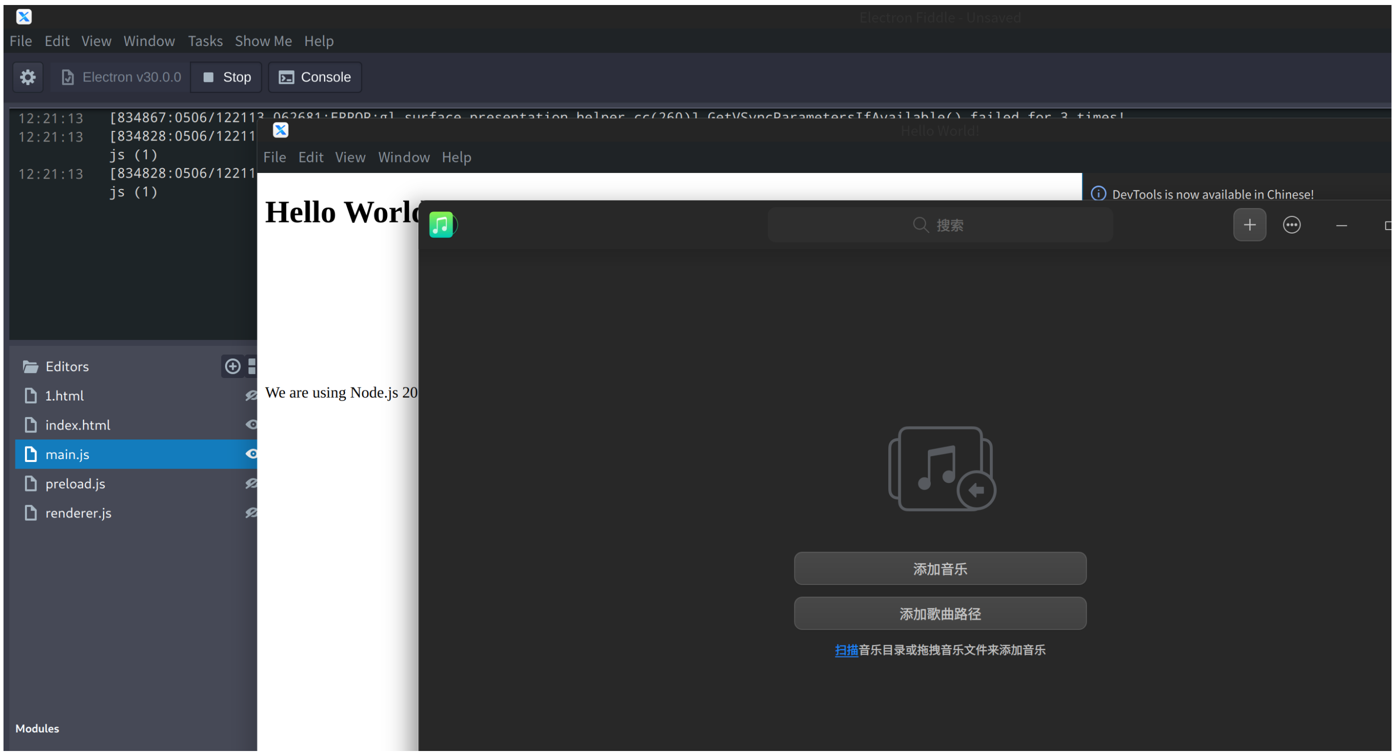


Main Process (main.js)

```
// Modules to control application life and create native browser window
const {app, BrowserWindow} = require('electron')
const path = require('path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 1400,
    height: 800,
    webPreferences: {
      nodeIntegration: true,
      contextIsolation: false,
      // webSecurity: false,
      // sandbox: true,
      nodeIntegrationInSubFrames: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  mainWindow.webContents.openDevTools()
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.whenReady().then(() => {
```

HTML (index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self'
script-src 'self'"> -->
    <!-- <meta http-equiv="X-Content-Security-Policy" content="default-src 'se
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <!-- <embed src="http://192.168.31.216/1.html"></embed> -->
    <embed src="http://192.168.31.216/1.html"></embed>
    <script src="./renderer.js"></script>
  </body>
</html>
```

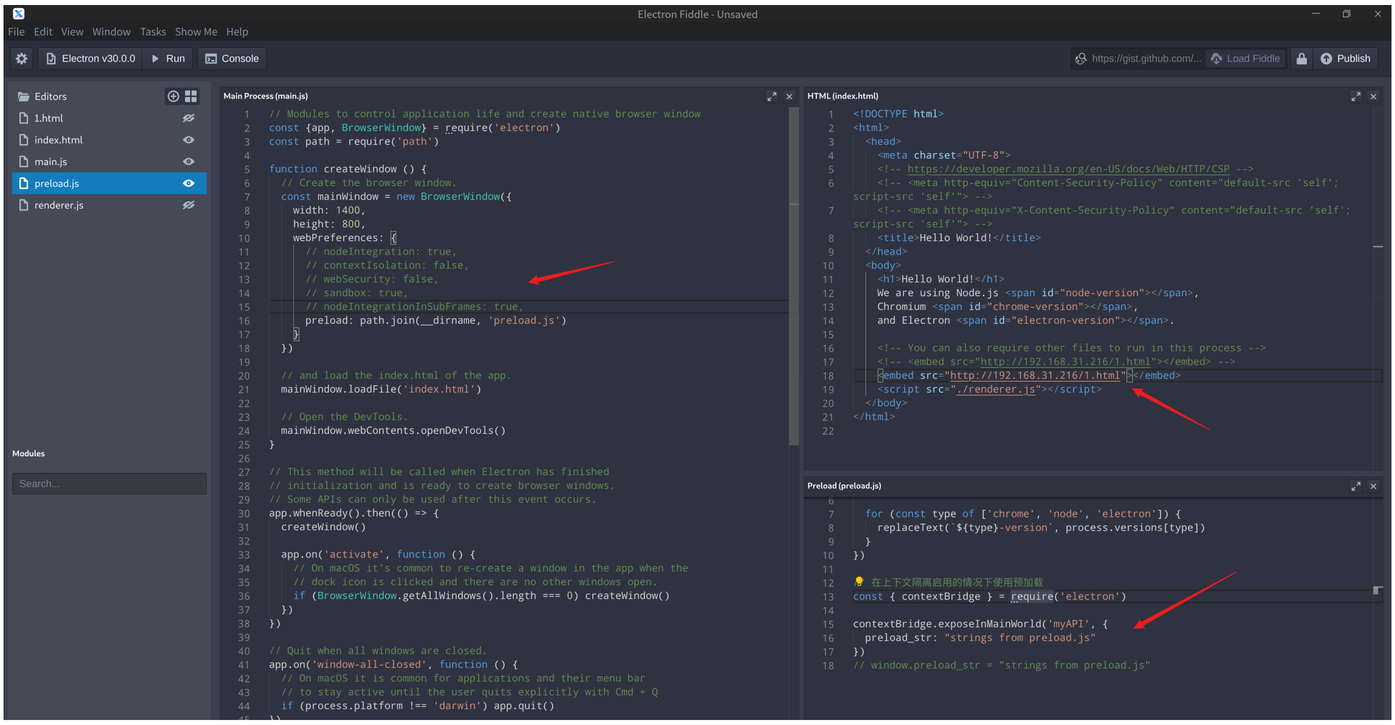

```
12:21:13    [834867:0506/122113.062681:ERROR:gl_surface_presentation_helper.cc(260)] GetVSyncParametersIfAvailable() failed for 3 times!
12:21:13    [834828:0506/12211
            js (1)
12:21:13    [834828:0506/12211
            js (1)
```

成功执行，经过测试， `embed` 执行 `Node.js` 的条件与 `iframe` 一致

3) 测试预加载脚本

修改 `embed` 服务器内容，获取并控制台输出预加载脚本暴露给渲染进程的值

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <div>
    <h1>embed 页面 - 1.html</h1>
    <script>
      console.log(window.myAPI.preload_str)
      // window.require('child_process').exec('deepin-music')

        // window.flag = "strings from object"

        // if (window.require !== undefined) {
        //   window.require('child_process').exec('deepin-music')
        // } else {
        //   window.parent.require('child_process').exec('deepin-music')
        // }

        // setTimeout(() => {
        //   console.log(window.parent.renderer_str)
        //   console.log(window.parent.preload_str)
        // }, 2000)

    </script>
  </div>
</body>
</html>
```



Main Process (main.js)

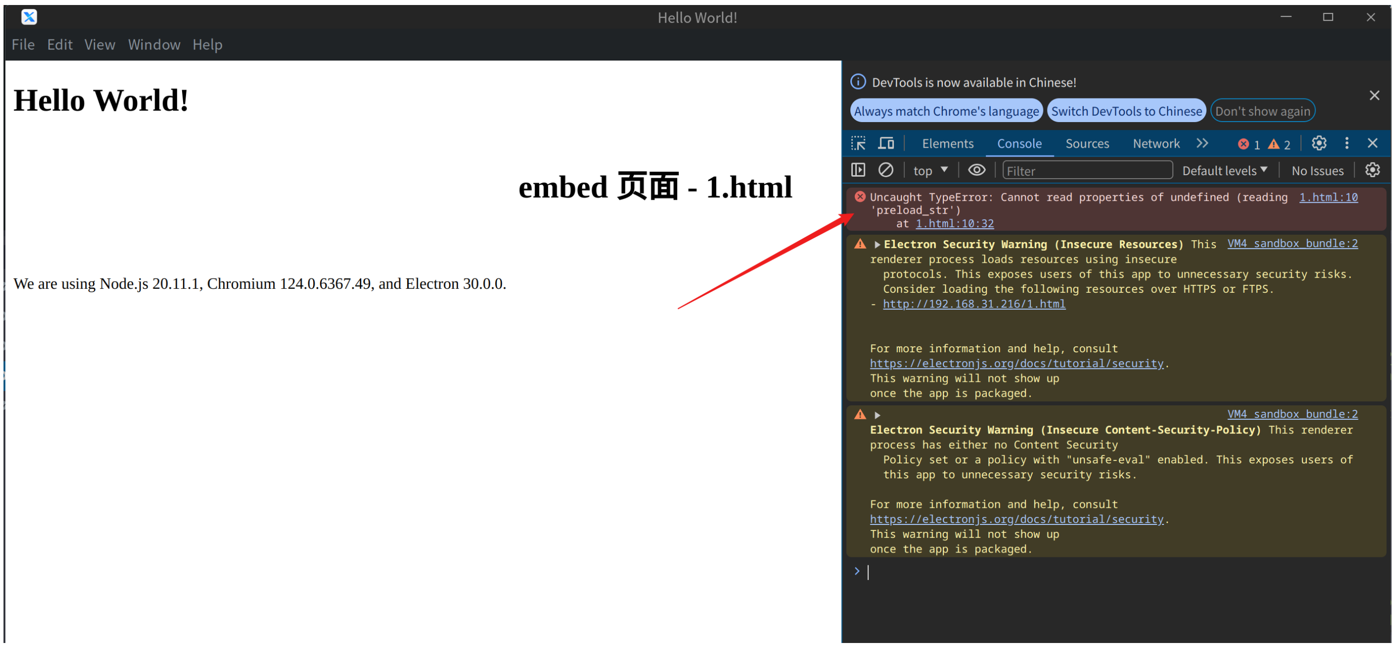
```
// Modules to control application life and create native browser window
const {app, BrowserWindow} = require('electron')
const path = require('path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 1400,
    height: 800,
    webPreferences: {
      // nodeIntegration: true,
      // contextIsolation: false,
      // webSecurity: false,
      // sandbox: true,
      // nodeIntegrationInSubFrames: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  mainWindow.webContents.openDevTools()
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.whenReady().then(() => {
  createWindow()

  app.on('activate', function () {
    // On macOS it's common to re-create a window in the app when the
    // dock icon is clicked and there are no other windows open.
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

// Quit when all windows are closed.
app.on('window-all-closed', function () {
  // On macOS it is common for applications and their menu bar
  // to stay active until the user quits explicitly with Cmd + Q
  if (process.platform !== 'darwin') app.quit()
```

HTML (index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <!-- <meta http-equiv="X-Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <!-- <embed src="http://192.168.31.216/1.html"></embed> -->
    <embed src="http://192.168.31.216/1.html"></embed>
    <script src="./renderer.js"></script>
  </body>
</html>
```

Preload (preload.js)

```

  for (const type of ['chrome', 'node', 'electron']) {
    replaceText(`${type}-version`, process.versions[type])
  }
})

// 在上下文隔离启用的情况下使用预加载
const { contextBridge } = require('electron')

contextBridge.exposeInMainWorld('myAPI', {
  preload_str: "strings from preload.js"
})
// window.preload_str = "strings from preload.js"
```


Hello World!
File Edit View Window Help
DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to Chinese
Don't show again
Elements
Console
Sources
Network
top
Filter
Default levels
No Issues

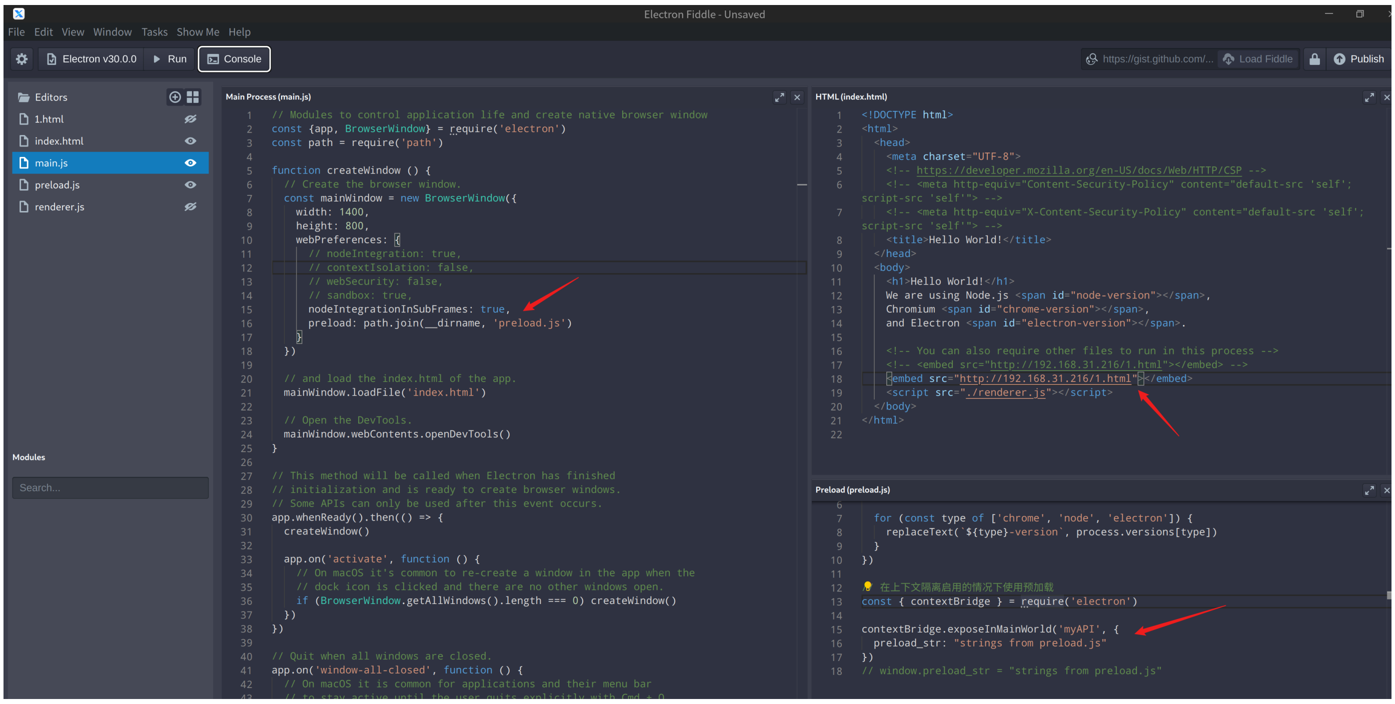
Hello World!

embed 页面 - 1.html

We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

```
Uncaught TypeError: Cannot read properties of undefined (reading     1.html:10
'preload_str')
    at 1.html:10:32

▶ Electron Security Warning (Insecure Resources) This  VM4 sandbox_bundle:2
renderer process loads resources using insecure
  protocols. This exposes users of this app to unnecessary security risks.
  Consider loading the following resources over HTTPS or FTPS.
- http://192.168.31.216/1.html


For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.

▶                                                      VM4 sandbox_bundle:2
Electron Security Warning (Insecure Content-Security-Policy) This renderer
process has either no Content Security
  Policy set or a policy with "unsafe-eval" enabled. This exposes users of
  this app to unnecessary security risks.

For more information and help, consult
https://electronjs.org/docs/tutorial/security.
This warning will not show up
once the app is packaged.
```

设置 `nodeIntegrationInSubFrames: true`



Main Process (main.js)

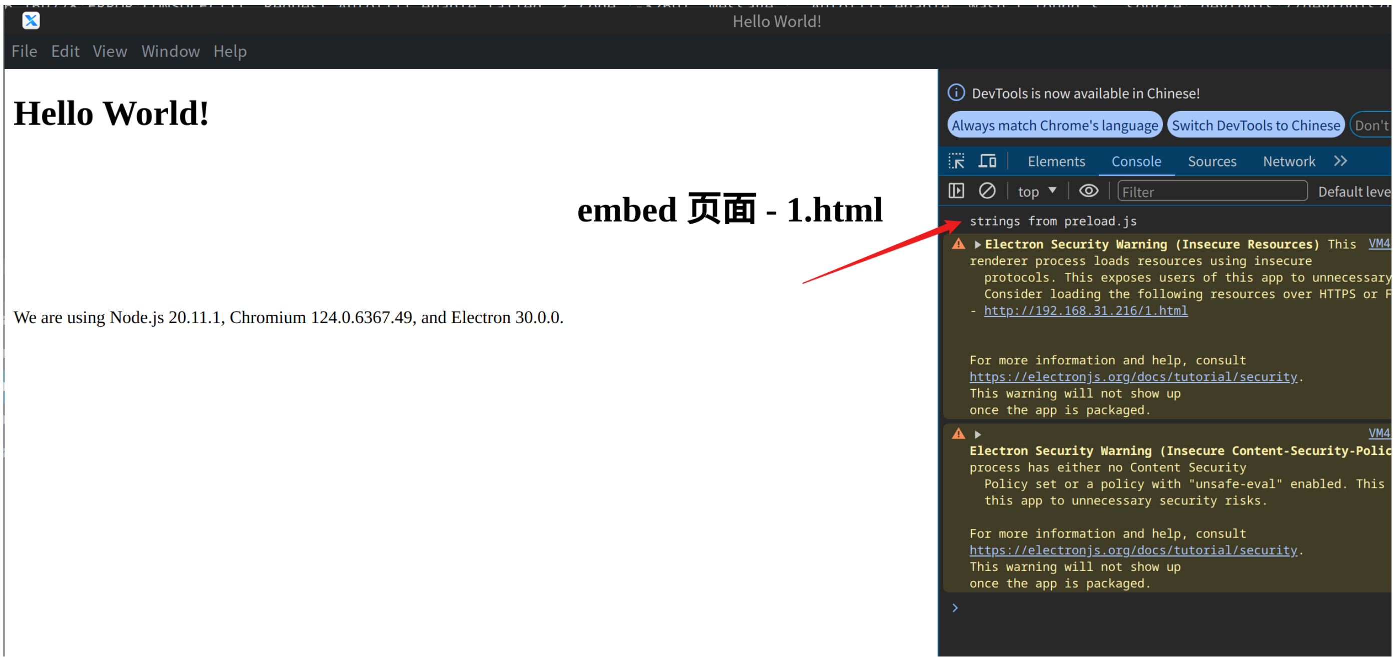
```javascript
// Modules to control application life and create native browser window
const {app, BrowserWindow} = require('electron')
const path = require('path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 1400,
    height: 800,
    webPreferences: {
      // nodeIntegration: true,
      // contextIsolation: false,
      // webSecurity: false,
      // sandbox: true,
      nodeIntegrationInSubFrames: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  mainWindow.webContents.openDevTools()
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.whenReady().then(() => {
  createWindow()

  app.on('activate', function () {
    // On macOS it's common to re-create a window in the app when the
    // dock icon is clicked and there are no other windows open.
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

// Quit when all windows are closed.
app.on('window-all-closed', function () {
  // On macOS it is common for applications and their menu bar
  // to stay active until the user quits explicitly with Cmd + Q
```

HTML (index.html)

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <!-- <meta http-equiv="X-Content-Security-Policy" content="default-src 'self';
script-src 'self'"> -->
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using Node.js <span id="node-version"></span>,
    Chromium <span id="chrome-version"></span>,
    and Electron <span id="electron-version"></span>.

    <!-- You can also require other files to run in this process -->
    <!-- <embed src="http://192.168.31.216/1.html"></embed> -->
    <embed src="http://192.168.31.216/1.html"></embed>
    <script src="./renderer.js"></script>
  </body>
</html>
```

Preload (preload.js)

```javascript

  for (const type of ['chrome', 'node', 'electron']) {
    replaceText(`${type}-version`, process.versions[type])
  }
})

// 在上下文隔离启用的情况下使用预加载
const { contextBridge } = require('electron')

contextBridge.exposeInMainWorld('myAPI', {
  preload_str: "strings from preload.js"
})
// window.preload_str = "strings from preload.js"
```



Hello World!
File Edit View Window Help
DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to Chinese
Don't
Elements
Console
Sources
Network
top
Filter
Default leve

Hello World!

embed 页面 - 1.html

We are using Node.js 20.11.1, Chromium 124.0.6367.49, and Electron 30.0.0.

```
strings from preload.js
⚠ ▸Electron Security Warning (Insecure Resources) This   VM4
  renderer process loads resources using insecure
    protocols. This exposes users of this app to unnecessary
    Consider loading the following resources over HTTPS or F
  - http://192.168.31.216/1.html

  For more information and help, consult
  https://electronjs.org/docs/tutorial/security.
  This warning will not show up
  once the app is packaged.
⚠ ▸                                                       VM4
  Electron Security Warning (Insecure Content-Security-Polic
  process has either no Content Security
    Policy set or a policy with "unsafe-eval" enabled. This
    this app to unnecessary security risks.

  For more information and help, consult
  https://electronjs.org/docs/tutorial/security.
  This warning will not show up
  once the app is packaged.
>
```

成功获取到预加载脚本暴露给渲染页面的内容

4) 小结

`nodeIntegrationInSubFrames` 对 `embed` 的影响与 `iframe` 一致

0x06 总结

`nodeIntegrationInSubFrames` 这个配置项的含义随着其他配置项而呈现不同效果，目前来看，影响的对象主要是 `iframe` 、 `object` 、 `embed`

- 如果 `nodeIntegrationInSubFrames` 设置为 `true` 时， `preload` 脚本中暴露的方法和值等将向 `iframe` 、 `object` 、 `embed` 内暴露，也就是说 `iframe` 、 `object` 、 `embed` 内部的内容中的 `JavaScript` 可以直接使用 `Preload` 脚本中定义好的功能和值
- 如果嵌入 `iframe` 、 `object` 、 `embed` 的宿主页面的安全配置为
  - `sandbox: false`
  - `nodeIntegration: true`
  - `contextIsolation: false`
  - `nodeIntegrationInSubFrames: true`

其中 `sandbox` 为 `false` 或默认即可，此时页面中嵌入的 `iframe` 、 `object` 、 `embed` 的内容可执行 `Node.js`