前言

> *本小册是《千锋大前端小册》系列之 Electron 部分。内容包含 Electron 基础和实战项目。全部内容是依照《千锋教育大前端好程序员大纲-Electron》编写。*
>
> *—— 作者：千锋教育 ·古艺散人*

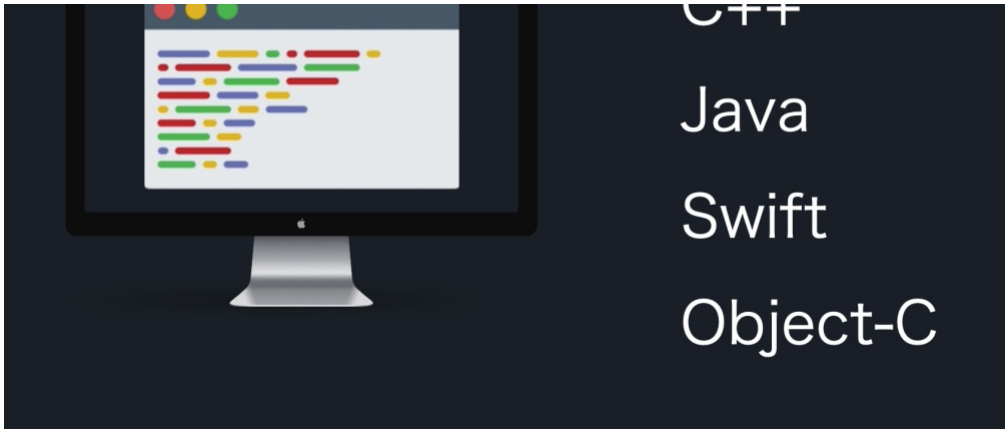
Electron 介绍

1、概览

想必你已经听说了可以应用electron来构建令人惊叹的桌面应用程序！

单纯使用JavaScript API 就可以构建Mac, windows或者Linux应用程序。

长期以来，很多开发语言都保留了生成桌面应用程序的功能，比如C和Java，但是用这些语言来构建应用程序是非常困难的。



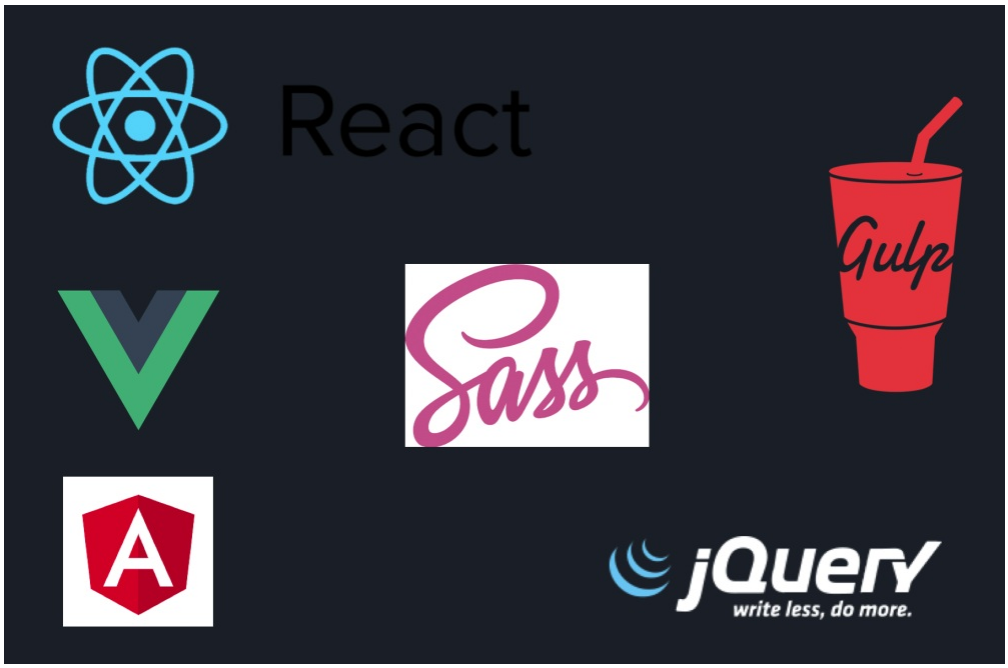
当然，近年来，构建本地应用程序变的更加灵活，但您仍然需要为每个操作系统学习不同的语言并使用非常特定的工具进行开发。

而如今，在Mac，Windows和Linux系统上，应用Electron技术，可以使我们前端开发人员运用现有的知识来解决这个问题了。

我们利用JavaScript，HTML和CSS这些Web技术来构建单个应用程序，然后为Mac windows 和 Linux 编译该应用程序。

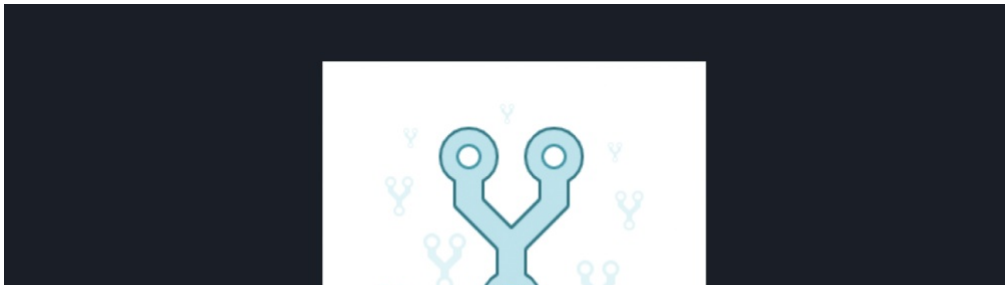
这样，我们就不用为特定的平台维护不同的应用程序了。

此外，我们还可以使用我们喜欢的框架和库来实现这个程序，比如 Vue, React 等前端框架。



Electron开发利用的是纯 Web 技术，她基于 Chromium 和 Node.js, 让你可以使用 HTML, CSS 和 JavaScript 构建应用。

Electron完全开源，她是一个由 GitHub 及众多贡献者组成的活跃社区共同维护的开源项目。



Electron完全跨平台，她兼容 Mac、Windows 和 Linux，可以构建出三个平台的应用程序。

如果你可以建一个网站，你就可以建一个桌面应用程序。 Electron 是一个使用 JavaScript, HTML 和 CSS 等 Web 技术创建原生程序的框架，它负责比较难搞的部分，你只需把精力放在你的应用的核心上即可。 Electron,一定比你想象的更简单！！

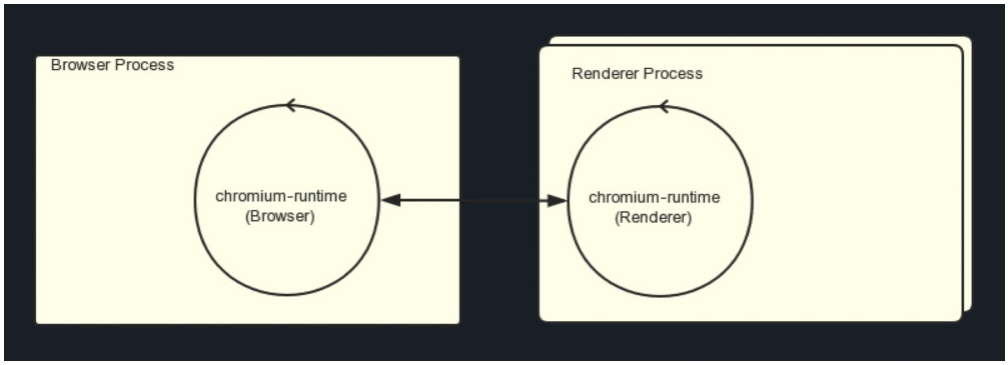
Electron 最初为 GitHub 开发 Atom 编辑器，此后被世界各地的公司采纳。比如Slack，甚至微软自己的 Visual Studio。

我们先来搭建一个Electron的运行环境

2、Electron 原理

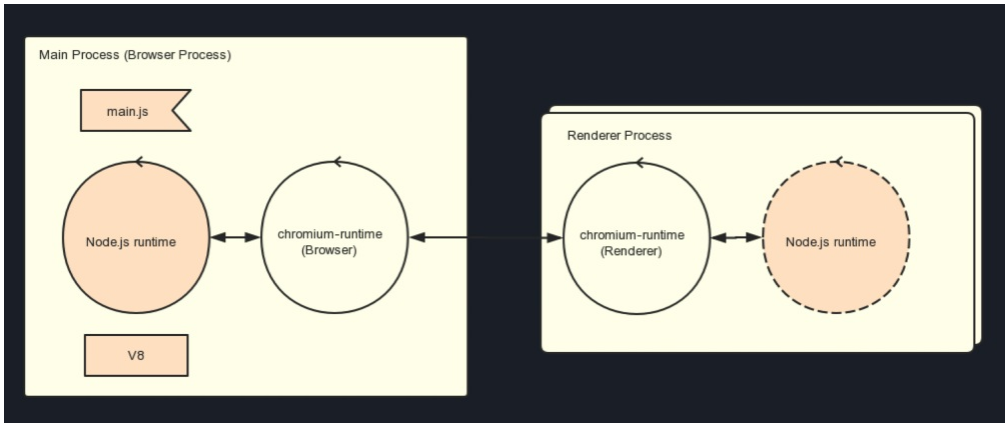
在深入学习Eelectron 之前，我们有必要了解一下Electron的应用架构。

Electron 运行在两类进程中，一类是主进程，一类是渲染进程
我们要知道，electron是基于chromium才能工作的，那我们就先简单看下chromium架构：



chromium运行时有一个Browser Process，以及一个或者多个Renderer Process。Renderer Process 顾名思义负责渲染Web页面。Browser Process则负责管理各个Renderer Process以及其他部分（比如菜单栏，收藏夹等等）。

我们再来看看electron在chromium的基础上做了什么：



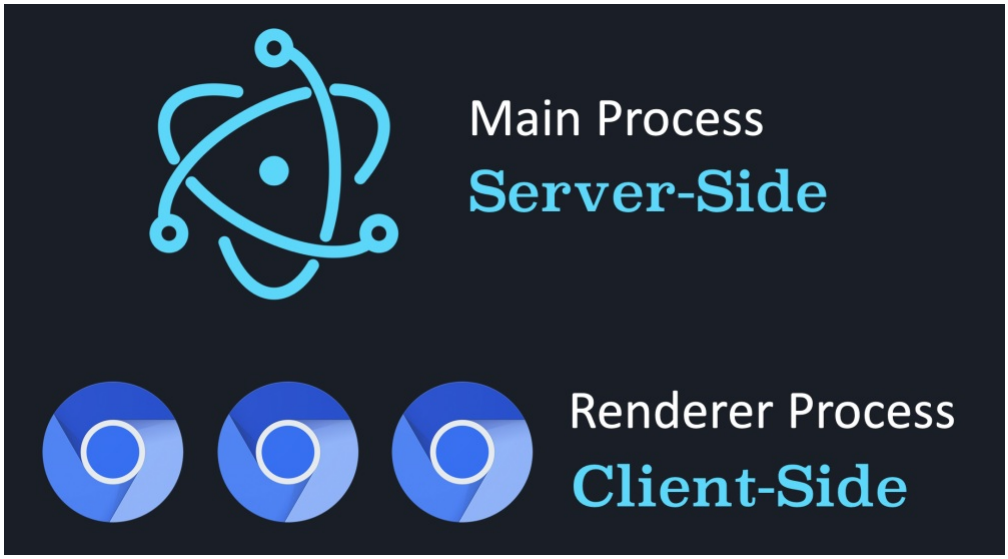
- Renderer Process

在electron中，仍然使用Renderer Process渲染页面，也就是说electron app使用Web页面作为UI显示，并且兼容传统的Web页面。不同的是electron app开发者可以可选的配置是否支持Node.js。

- Main Process

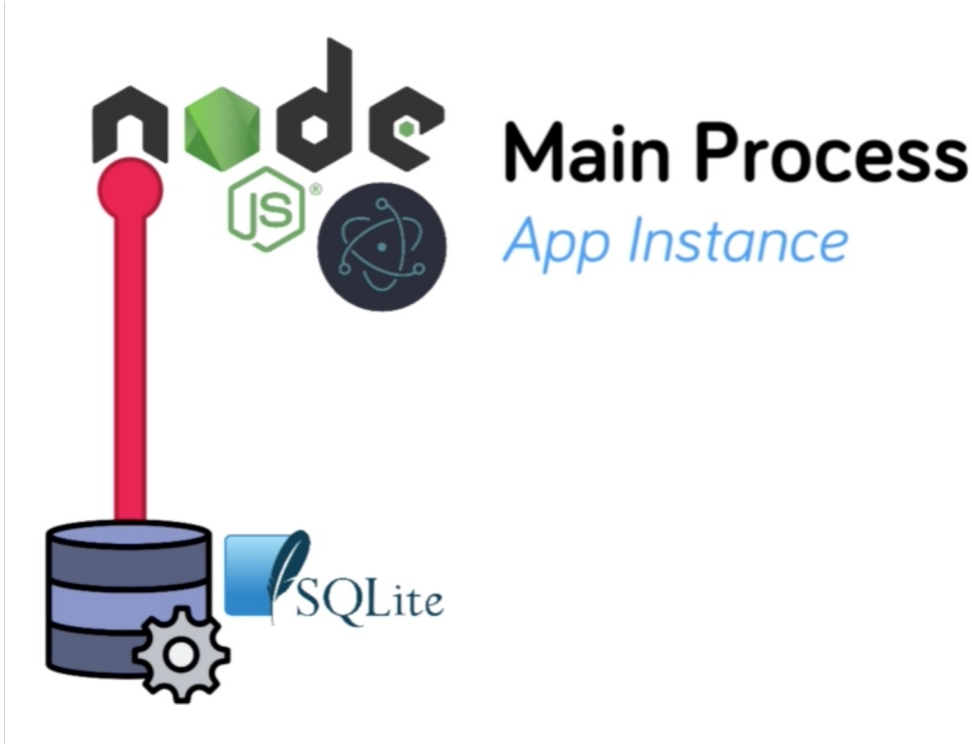
electron对Browser Process改动较大，干脆另起一个名字叫Main Process。Main Process 除了原来chromium的runtime，又添加了Node.js的runtime，main.js便运行在此之上。

electron将Node.js的message loop和chromium联系起来，使得js中可以灵活的控制页面显示，以及和Renderer Process的IPC通信。 进程间通信(IPC,Inter-Process Communication)指至少两个进程或线程间传送数据或信号的一些技术或方法。
当然原生的Node API和第三方的node module同样支持，并且有electron API提供给开发者控制原生菜单和通知等。
有一点需要注意，Browser Process本来没有js运行时，所以还需要依赖V8（当然这是chromium中的V8，不是单独的V8库）。



总结一下，一个Main Process(主进程），一个或多个Rederer(渲染进程) 构成了Electron的运行架构。
我们姑且把主进程叫Server-side服务端，将rederen process叫客户端。

- electron 使用 Node.js 原生模块



Node.js 原生模块是用 C++ 编写的 Node.js 扩展。C++ 源码通过 node-gyp 编译为 .node 后缀的二进制文件（类似于 .dll 和 .so）。在 Node.js 环境中可以直接用 require() 函数将 .node 文件初始化为动态链接库。一些 npm 包会包含 C++ 扩展，例如： node-ffi、node-iconv、node-usb，但都是源码版本，在安装后需要编译后才能被 Node.js 调用。
Electron 同样也支持 Node 原生模块，但由于和官方的 Node 相比使用了不同的 V8 引擎，如果你想编译原生模块，则需要手动设置 Electron 的 headers 的位置。

搭建 Electron 运行环境

指定 Electron 运行环境

1、安装 Electron

```
npm install electron -S
```

2、尝试官网的一个案例

```
# 克隆这仓库
git clone https://github.com/electron/electron-quick-start
# 进入仓库
cd electron-quick-start
# 安装依赖库
npm install
# 运行应用
npm start
```

Main Process API

> *Electron API （Electron API 有三种）*

- Main Process （主进进程）
- Renderer Process（渲染进程）
- Share Modules（共享模块）

App

事件

ready:

> *当 Electron 完成初始化时被触发。*

- 两种使用方法

```
app.on('ready', createWindow)
app.on('ready', () => {
  console.log('App is ready!')
  createWindow()
})
```

```
});
```

- 检查应用是否登录：app.isReady()
- 如果应用没有Ready，app.isReady()的值为 false

```
console.log('应用是否登录：' + app.isReady())
```

- 此时应用应该已经Ready

```
setTimeout(() => {
 console.log('应用是否登录：' + app.isReady())
}, 2000)
```

before-quit

> *在应用程序开始关闭窗口之前触发。*

```
app.on('before-quit', (e) => {
 console.log('App is quiting')
 e.preventDefault()
})
```

browser-window-blur

> *在 browserWindow 失去焦点时发出*

```
app.on('browser-window-blur', (e) => {
 console.log('App unfocused')
})
```

browser-window-focus

> *在 browserWindow 获得焦点时发出*

```
app.on('browser-window-focus', (e) => {
 console.log('App focused')
})
```

方法

app.quit()

```
app.on('browser-window-blur', (e) => {
 setTimeout(() => {
```

```
  app.quit()
 }, 3000)
})

app.on('browser-window-blur', (e) => {
 setTimeout(app.quit, 3000)
})
```

app.getPath(name)

```
app.on('ready', () => {
 console.log(app.getPath('desktop'))
 console.log(app.getPath('music'))
 console.log(app.getPath('temp'))
 console.log(app.getPath('userData'))

 createWindow()
})
```

BrowserWindow

> *electron.BrowserWindow: 创建和控制浏览器窗口*

实例方法

win.loadURL(url[, options]): 和 loadFile 互斥

```
mainWindow.loadURL('https://www.baidu.com')
```

优雅的显示窗口

- 使用ready-to-show事件

```
let mainWindow = new BrowserWindow({ show: false })
mainWindow.once('ready-to-show', () => {
 mainWindow.show()
})
```

- 设置 backgroundColor

```
let win = new BrowserWindow({ backgroundColor: '#2e2c29' })
```

父子窗口

父子窗口

- 窗口定义

```
secondaryWindow = new BrowserWindow({
 width: 600,
 height: 600,
 webPreferences: { nodeIntegration: true }
})

secondaryWindow.loadFile('index.html')

secondaryWindow.on('closed', () => {
 mainWindow = null
})
```

- 窗口关系

```
secondaryWindow = new BrowserWindow({
 parent: mainWindon, // 定义父窗口
 modal: true // 锁定在主窗口
})
```

- 子窗口显示和隐藏

```
secondaryWindow = new BrowserWindow({
 show: false
})

setTimeout(() => {
 secondaryWindow.show()
 setTimeout(() => {
 secondaryWindow.hide()
 }, 3000)
}, 2000)
```

无边框窗口

> *Frameless Window*

```
mainWindow = new BrowserWindow({
 frame: false
})
```

让页面可拖拽

```
<body style="user-select: none; -webkit-app-region:drag;">
```

no-drag 修复下面控件的bug

```
<input style="-webkit-app-region: no-drag;" type="range" name="range" min="0" max="10">
```

显示红绿灯

```
mainWindow = new BrowserWindow({
 titleBarStyle: 'hidden' // or hiddenInset 距离红绿灯更近
})
```

属性与方法

minWidth && minHeight

```
mainWindow = new BrowserWindow({
 minWidth: 300,
 minHeight: 300
})
```

更多详见：https://electronjs.org/docs/api/browser-window#new-browserwindowoptions

窗口焦点事件

```
secWindow = new BrowserWindow({
 width: 400, height: 300,
 webPreferences: { nodeIntegration: true },
})

mainWindow.on('focus', () => {
 console.log('mainWindow focused')
})

secWindow.on('focus', () => {
 console.log('secWindow focused')
})

app.on('browser-window-focus', () => {
 console.log('App focused')
})
```

静态方法

- getAllWindows()

```
let allWindows = BrowserWindow.getAllWindows()
console.log(allWindows)
```

更多详见: https://electronjs.org/docs/api/browser-window#%E9%9D%99%E6%80%81%E6%96%B9%E6%B3%95

实例属性

- id

```
console.log(secWindow.id)
```

更多详见: https://electronjs.org/docs/api/browser-window#%E5%AE%9E%E4%BE%8B%E5%B1%9E%E6%80%A7

实例方法

- maximize()

```
secWindow.on('closed', () => {
 mainWindow.maximize()
})
```

更多详见: https://electronjs.org/docs/api/browser-window#%E5%AE%9E%E4%BE%8B%E6%96%B9%E6%B3%95

state

> *electron-window-state 保存窗口的状态*
> `npm install electron-window-state`

webContents

> *webContents 是 EventEmitter 的实例， 负责渲染和控制网页，是 BrowserWindow 对象的一个属性。*

```
let wc = mainWindow.webContents
console.log(wc)
```

方法 getAllWebContents()

- 返回 WebContents[] - 所有 WebContents 实例的数组。 包含所有Windows，webviews，opened devtools 和 devtools 扩展背景页的 web 内容

```
const {app, BrowserWindow, webContents} = require('electron')
console.log(webContents.getAllWebContents())
```

实例事件

- did-finish-load
- dom-ready

```
<div>
 <img src="https://placekitten.com/500/500" alt="">
</div>
<script>
 let wc = mainWindow.webContents
 wc.on('did-finish-load', () => {
  console.log('Conent fully loaded')
 })
 wc.on('dom-ready', () => {
  console.log('DOM Ready')
 })
</script>
```

- new-window

```
<div>
 <a target="_blank" href="https://placekitten.com/500/500"><h3>Kitten</h3></a>
</div>

<script>
 wc.on('new-window', (e, url) => {
  e.preventDefault()
  console.log('DOM Ready')
 })
</script>
```

- before-input-event

```
 wc.on('before-input-event', (e, input) => {
  console.log(`${input.key} : ${input.type}`)
 })
```

- login
- did-navigate

```
 mainWindow.loadURL('https://httpbin.org/basic-auth/user/passwd')

 wc.on('login', (e, request, authInfo, callback) => {
  console.log('Logging in:')
  callback('user', 'passwd')
 })
```

```
wc.on('did-navigate', (e, url, statusCode, message) => {
  console.log(`Navigated to: ${url}, with response code: ${statusCode}`)
  console.log(message)
 })
```

- media-started-playing
- media-paused

```
 <div>
  <video src="./mgm.mp4" controls width="400"></video>
 </div>
 <script>
 wc.on('media-started-playing', () => {
  console.log('Video Started')
 })
 wc.on('media-paused', () => {
  console.log('Video Paused')
 })
 </script>
```

- context-menu : 右键上下文信息

```
 wc.on('context-menu', (e, params) => {
  console.log(`Context menu opened on: ${params.mediaType} at x:${params.x}, y:${params.y}`)
 })

 wc.on('context-menu', (e, params) => {
  console.log(`User seleted text: ${params.selectionText}`)
  console.log(`Selection can be copied: ${params.editFlags.canCopy}`)
 })
```

实例方法

- executeJavaScript()

```
 wc.on('context-menu', (e, params) => {
  wc.executeJavaScript(`alert('${params.selectionText}')`)
 })
```

Session

管理浏览器会话、cookie、缓存、代理设置等

> 管理浏览器会话、cookie、缓存、代理设置等。

起步

- 创建session对象

```
let session = mainWindow.webContents.session
console.log(session) // {}
```

- 在chromium 创建localStorage，然后创建两个窗口，两个session共享

```
mainWindow = new BrowserWindow({
 width: 1000, height: 800,
 webPreferences: { nodeIntegration: true }
})

secWindow = new BrowserWindow({
 width: 500, height: 400,
 webPreferences: { nodeIntegration: true }
})

let session = mainWindow.webContents.session
let session2 = mainWindow.webContents.session
console.log(Object.is(session, session2)) // true

// Load index.html into the new BrowserWindow
mainWindow.loadFile('index.html')
secWindow.loadFile('index.html')

// Open DevTools - Remove for PRODUCTION!
mainWindow.webContents.openDevTools();
secWindow.webContents.openDevTools();

// Listen for window being closed
mainWindow.on('closed', () => {
 mainWindow = null
})
secWindow.on('closed', () => {
 secWindow = null
})
```

- defaultSession

```
const {app, BrowserWindow, session} = require('electron')
```

```
let ses = mainWindow.webContents.session
console.log(Object.is(session.defaultSession, ses)) // true
```

- 自定义session

```
let customSes = session.fromPartition('part1')
console.log(Object.is(customSes, ses)) //false, 此时customSes 还是共享session

secWindow = new BrowserWindow({
 width: 500, height: 400,
 webPreferences: {
 nodeIntegration: true,
 session: customSes // 定义session, 此时子窗口有自己的session
 }
})

// 在子窗口里创建localstorge: winName/secWindow
// 关闭所有窗口，发现创建的localstorage又消失了，因为此时的session存储在内存里，重新启动应用又消失了。可以加前缀persist，使其变为永久存储:

let customSes = session.fromPartition('persist:part1')

// 或者:

secWindow = new BrowserWindow({
 width: 500, height: 400,
 webPreferences: {
 nodeIntegration: true,
 - session: customSes
 + partition: 'persist:part1'
 }
})
```

- 实例方法

```
ses.clearStorageData() // 删除主窗口的的storage
```

cookie

> *查询和修改一个会话的cookies*

```
// 查询所有 cookies
session.defaultSession.cookies.get({})
  .then((cookies) => {
```

```
  .then((cookies) => {
  console.log(cookies)
  }).catch((error) => {
  console.log(error)
  })
// 查询所有与设置的 URL 相关的所有 cookies
session.defaultSession.cookies.get({ url: 'http://www.github.com' })
  .then((cookies) => {
  console.log(cookies)
  }).catch((error) => {
  console.log(error)
  })
// 设置一个 cookie，使用设置的名称；
// 如果存在，则会覆盖原先 cookie.
const cookie = { url: 'http://www.github.com', name: 'dummy_name', value: 'dummy' }
session.defaultSession.cookies.set(cookie)
  .then(() => {
  // success
  }, (error) => {
  console.error(error)
  })
```

downloadItem

> *控制来自于远程资源的文件下载。*

```
<h2><a href="https://picsum.photos/5000/5000/" download>Download Image</a></h2>
<progress value="0" max="100" id="progress"></progress>

<script>
  window.progress = document.getElementById('progress')
</script>
// main.js
let ses = session.defaultSession

ses.on('will-download', (e, downloadItem, webContents) => {

  let fileName = downloadItem.getFilename()
  let fileSize = downloadItem.getTotalBytes()

  // Save to desktop
  downloadItem.setSavePath(app.getPath('desktop') + `/${fileName}`)
```

```
downloadItem.setSavePath(app.getPath('desktop') + `/${fileName}`)

downloadItem.on('updated', (e, state) => {

let received = downloadItem.getReceivedBytes()

if (state === 'progressing' && received) {
let progress = Math.round((received/fileSize)*100)
webContents.executeJavaScript(`window.progress.value = ${progress}`)
}
})
})
```

dialog - 对话框

> *显示用于打开和保存文件、警报等的本机系统对话框*

```
const {app, BrowserWindow, dialog} = require('electron')

mainWindow.webContents.on('did-finish-load', () => {
dialog.showOpenDialog({
buttonLabel: '选择',
defaultPath: app.getPath('desktop'),
properties: ['multiSelections', 'createDirectory', 'openFile', 'openDirectory']
}, filepaths => {
console.log(filepaths)
})
})
dialog.showSaveDialog({}, filename => {
console.log(filename)
})
const answers = ['Yes', 'No', 'Maybe']

dialog.showMessageBox({
title: 'Message Box',
message: 'Please select an option',
detail: 'Message details.',
buttons: answers
}, response => {
console.log(`User selected: ${answers[response]}`)
})
```

快捷键+系统快捷键

> ***快捷键：****定义键盘快捷键。*
> ***系统快捷键：****在应用程序没有键盘焦点时，监听键盘事件。*

快捷键可以包含多个功能键和一个键码的字符串，由符号+结合，用来定义你应用中的键盘快捷键

示例：

- CommandOrControl+A
- CommandOrControl+Shift+Z

快捷方式使用 register 方法在 globalShortcut 模块中注册。

globalShortcut 模块可以在操作系统中注册/注销全局快捷键，以便可以为操作定制各种快捷键。

注意：快捷方式是全局的；即使应用程序没有键盘焦点，它也仍然在持续监听键盘事件。 在应用程序模块发出 ready 事件之前，不应使用此模块。

```
const {app, BrowserWindow, globalShortcut} = require('electron')

globalShortcut.register('G', () => {
 console.log('User pressed G')
})
globalShortcut.register('CommandOrControl+Y', () => {
 console.log('User pressed G with a combination key')
 globalShortcut.unregister('CommandOrControl+G')
})
```

Menu

1、index.html

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
 <title>Hello World!</title>
 </head>
```

```
  <body>
  <h1>Hello World!</h1>

  <textarea name="name" rows="8" cols="80"></textarea>

  <!-- All of the Node.js APIs are available in this renderer process. -->
  We are using Node.js <strong><script>document.write( process.versions.node)</script></strong>,
  and Electron <strong><script>document.write( process.versions.electron )</script></strong>.

  <script>
  // You can also require other files to run in this process
  require('./renderer.js')
  </script>
  </body>
 </html>
```

2、main.js

```
 // Modules
 const {app, BrowserWindow, Menu, MenuItem} = require('electron')

 // Keep a global reference of the window object, if you don't, the window will
 // be closed automatically when the JavaScript object is garbage collected.
 let mainWindow

 let mainMenu = Menu.buildFromTemplate( require('./mainMenu') )


 // Create a new BrowserWindow when `app` is ready
 function createWindow () {

  mainWindow = new BrowserWindow({
  width: 1000, height: 800,
  webPreferences: { nodeIntegration: true }
  })

  // Load index.html into the new BrowserWindow
  mainWindow.loadFile('index.html')

  // Open DevTools - Remove for PRODUCTION!
```

```
  // Open DevTools - Remove for PRODUCTION!
  mainWindow.webContents.openDevTools();

  Menu.setApplicationMenu(mainMenu)

  // Listen for window being closed
  mainWindow.on('closed', () => {
  mainWindow = null
  })
 }

// Electron `app` is ready
app.on('ready', createWindow)

// Quit when all windows are closed - (Not macOS - Darwin)
app.on('window-all-closed', () => {
 if (process.platform !== 'darwin') app.quit()
})

// When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
app.on('activate', () => {
 if (mainWindow === null) createWindow()
})
```

3、mainMenu.js

```
module.exports = [
 {
 label: 'Electron',
 submenu: [
 { label: 'Item 1'},
 { label: 'Item 2', submenu: [ { label: 'Sub Item 1'} ]},
 { label: 'Item 3'},
 ]
 },
 {
 label: 'Edit',
 submenu: [
 { role: 'undo'},
 { role: 'redo'},
 { role: 'copy'},
 { role: 'paste'},
```

```
    ]
    },
    {
    label: 'Actions',
    submenu: [
    {
    label: 'DevTools',
    role: 'toggleDevTools'
    },
    {
    role: 'toggleFullScreen'
    },
    {
    label: 'Greet',
    click: () => { console.log('Hello from Main Menu') },
    accelerator: 'Shift+Alt+G'
    }
    ]
    }
]
```

Context Menus

1、index.html

```
<!DOCTYPE html>
<html>
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
  <title>Hello World!</title>
  </head>
  <body>
  <h1>Hello World!</h1>

  <textarea name="name" rows="8" cols="80"></textarea>

  <!-- All of the Node.js APIs are available in this renderer process. -->
  We are using Node.js <strong><script>document.write( process.versions.node)</script></strong>,
```

```
  and Electron <strong><script>document.write( process.versions.electron )</script></strong>.

  <script>
  // You can also require other files to run in this process
  require('./renderer.js')
  </script>
  </body>
</html>
```

2、main.js

```
// Modules
const {app, BrowserWindow, Menu} = require('electron')

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let mainWindow

let contextMenu = Menu.buildFromTemplate([
  { label: 'Item 1' },
  { role: 'editMenu' }
])

// Create a new BrowserWindow when `app` is ready
function createWindow () {

  mainWindow = new BrowserWindow({
  width: 1000, height: 800,
  webPreferences: { nodeIntegration: true }
  })

  // Load index.html into the new BrowserWindow
  mainWindow.loadFile('index.html')

  // Open DevTools - Remove for PRODUCTION!
  mainWindow.webContents.openDevTools();

  mainWindow.webContents.on('context-menu', e => {
  contextMenu.popup()
  })
```

```
30.
31.  // Listen for window being closed
32.  mainWindow.on('closed', () => {
33.  mainWindow = null
34.  })
35. }
36.
37. // Electron `app` is ready
38. app.on('ready', createWindow)
39.
40. // Quit when all windows are closed - (Not macOS - Darwin)
41. app.on('window-all-closed', () => {
42.  if (process.platform !== 'darwin') app.quit()
43. })
44.
45. // When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
46. app.on('activate', () => {
47.  if (mainWindow === null) createWindow()
48. })
```

Tray（托盘）

1、main.js

```
1.  // Modules
2.  const {app, BrowserWindow, Tray, Menu} = require('electron')
3.
4.  // Keep a global reference of the window object, if you don't, the window will
5.  // be closed automatically when the JavaScript object is garbage collected.
6.  let mainWindow, tray
7.
8.  let trayMenu = Menu.buildFromTemplate([
9.   { label: 'Item 1' },
10.  { role: 'quit' }
11. ])
12.
13. function createTray() {
14.
15.  tray = new Tray('trayTemplate@2x.png')
16.  tray.setToolTip('Tray details')
17.
```

```
  tray.on('click', e => {

  if (e.shiftKey) {
  app.quit()
  } else {
  mainWindow.isVisible() ? mainWindow.hide() : mainWindow.show()
  }
  })

  tray.setContextMenu(trayMenu)
}

// Create a new BrowserWindow when `app` is ready
function createWindow () {

  createTray()

  mainWindow = new BrowserWindow({
  width: 1000, height: 800,
  webPreferences: { nodeIntegration: true }
  })

  // Load index.html into the new BrowserWindow
  mainWindow.loadFile('index.html')

  // Open DevTools - Remove for PRODUCTION!
  mainWindow.webContents.openDevTools();

  // Listen for window being closed
  mainWindow.on('closed', () => {
  mainWindow = null
  })
}

// Electron `app` is ready
app.on('ready', createWindow)

// Quit when all windows are closed - (Not macOS - Darwin)
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') app.quit()
})
```

```
59.
60. // When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
61. app.on('activate', () => {
62.  if (mainWindow === null) createWindow()
63. })
```

powerMonitor（电源指示器）

```
1.  // Modules
2.  const electron = require('electron')
3.  const {app, BrowserWindow} = electron
4.
5.  // Keep a global reference of the window object, if you don't, the window will
6.  // be closed automatically when the JavaScript object is garbage collected.
7.  let mainWindow
8.
9.  // Create a new BrowserWindow when `app` is ready
10. function createWindow () {
11.
12.  mainWindow = new BrowserWindow({
13.  width: 1000, height: 800,
14.  webPreferences: { nodeIntegration: true }
15.  })
16.
17.  // Load index.html into the new BrowserWindow
18.  mainWindow.loadFile('index.html')
19.
20.  // Open DevTools - Remove for PRODUCTION!
21.  mainWindow.webContents.openDevTools();
22.
23.  // Listen for window being closed
24.  mainWindow.on('closed', () => {
25.  mainWindow = null
26.  })
27.
28.  electron.powerMonitor.on('resume', e => {
29.  if(!mainWindow) createWindow()
30.  })
31.
32.  electron.powerMonitor.on('suspend', e => {
```

```
  console.log('Saving some data')
  })
 }

 // Electron `app` is ready
 app.on('ready', createWindow)

 // Quit when all windows are closed - (Not macOS - Darwin)
 app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') app.quit()
 })

 // When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
 app.on('activate', () => {
  if (mainWindow === null) createWindow()
 })
```

Renderer Process API

Renderer API 主要包括 `remote、Browser window proxy、desktop Capture`

> *Renderer Process API*

- remote
- Browser Window Proxy
- desktop Capture

1、remote（服务端对象）

1.1 index.html

```
 <!DOCTYPE html>
 <html>
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
  <title>Hello World!</title>
  </head>
  <body>
```

```
  <h1>Hello World!</h1>

  <button type="button" name="button" id="test-button">Fullscreen</button><br>

  <!-- All of the Node.js APIs are available in this renderer process. -->
  We are using Node.js <strong><script>document.write( process.versions.node)</script></strong>,
  and Electron <strong><script>document.write( process.versions.electron )</script></strong>.

  <script>
  // You can also require other files to run in this process
  require('./renderer.js')
  </script>
  </body>
 </html>
```

1.2 renderer.js

```
 // This file is required by the index.html file and will
 // be executed in the renderer process for that window.
 // All of the Node.js APIs are available in this process.

 const remote = require('electron').remote

 const { app, dialog, BrowserWindow } = remote

 const button = document.getElementById('test-button')

 button.addEventListener('click', e => {

  // dialog.showMessageBox({ message: 'Dialog invoked from Renderer process' })

  // let secWin = new BrowserWindow({
  // width: 400, height: 350
  // })
  // secWin.loadFile('index.html')

  // console.log( remote.getGlobal('myglob') )

  // app.quit()
```

```
23.
24.  let win = remote.getCurrentWindow()
25.  win.maximize()
26.
27. })
```

1.3 main.js

```
1.  // Modules
2.  const {app, BrowserWindow} = require('electron')
3.
4.  global['myglob'] = 'A var set in main.js'
5.
6.  // Keep a global reference of the window object, if you don't, the window will
7.  // be closed automatically when the JavaScript object is garbage collected.
8.  let mainWindow
9.
10. // Create a new BrowserWindow when `app` is ready
11. function createWindow () {
12.
13.  mainWindow = new BrowserWindow({
14.  width: 1000, height: 800,
15.  webPreferences: { nodeIntegration: true }
16.  })
17.
18.  // Load index.html into the new BrowserWindow
19.  mainWindow.loadFile('index.html')
20.
21.  // Open DevTools - Remove for PRODUCTION!
22.  mainWindow.webContents.openDevTools();
23.
24.  // Listen for window being closed
25.  mainWindow.on('closed', () => {
26.  mainWindow = null
27.  })
28. }
29.
30. // Electron `app` is ready
31. app.on('ready', createWindow)
32.
33. // Quit when all windows are closed - (Not macOS - Darwin)
34. app.on('window-all-closed', () => {
```

```
  if (process.platform !== 'darwin') app.quit()
})

// When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
app.on('activate', () => {
  if (mainWindow === null) createWindow()
})
```

2、Browser Window Proxy （浏览器窗口代理）

```
<!DOCTYPE html>
<html>
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
  <title>Hello World!</title>
  </head>
  <body>
  <h1>Hello World!</h1>

  <h3><a href="#" onclick="newWin()">New Window</a></h3>
  <h3><a href="#" onclick="closeWin()">Close Window</a></h3>
  <h3><a href="#" onclick="styleWin()">Bad Fonts</a></h3>

  <script>

  let win

  const newWin = () => {
  win = window.open('https://electronjs.org', '_blank', 'width=500,height=450,alwaysOnTop=1')
  }

  const closeWin = () => {
  win.close()
  }

  const styleWin = () => {
  win.eval("document.getElementsByTagName('body')[0].style.fontFamily = 'Comic Sans
```

```
MS'")
  }

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

3、webFrame

```
 <!DOCTYPE html>
 <html>
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
  <title>Hello World!</title>
  </head>
  <body>
  <h1>Hello World!</h1>

  <img src="https://placekitten.com/450/300" alt=""><br>

  <button onclick="zoomUp()">Increase Zoom</button>
  <button onclick="zoomDown()">Decrease Zoom</button>
  <button onclick="zoomReset()">Reset Zoom</button>

  <script>

  const { webFrame } = require('electron')

  const zoomUp = () => {
  webFrame.setZoomLevel( webFrame.getZoomLevel() + 1 )
  }
  const zoomDown = () => {
  webFrame.setZoomLevel( webFrame.getZoomLevel() - 1 )
  }
  const zoomReset = () => {
  webFrame.setZoomLevel( 1 )
  }

```

```
  console.log( webFrame.getResourceUsage() )

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

4、desktopCapturer（桌面快照）

4.1 index.html

```
 <!DOCTYPE html>
 <html>
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
  <title>Hello World!</title>
  </head>
  <body>
  <h1>Hello World!</h1>

  <img width="100%" src="" id="screenshot"><br>
  <button id="screenshot-button">Get Screenshot</button>

  <script>
  // You can also require other files to run in this process
  require('./renderer.js')
  </script>
  </body>
 </html>
```

4.2 renderer.js

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

 document.getElementById('screenshot-button').addEventListener('click', () => {

  desktopCapturer.getSources({ types:['window'], thumbnailSize: {width:1920, height:1080} }, (error, sources) => {

  console.log(sources)

```

```
  document.getElementById('screenshot').src = sources[0].thumbnail.toDataURL()
  })

})
```

IPC 通信

1、index.html

```
<!DOCTYPE html>
<html>
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
  <title>Hello World!</title>
  </head>
  <body>
  <h1>Hello World!</h1>

  <button type="button" id="talk">Talk to main process</button><br>

  <!-- All of the Node.js APIs are available in this renderer process. -->
  We are using Node.js <strong><script>document.write( process.versions.node)</script></strong>,
  and Electron <strong><script>document.write( process.versions.electron )</script></strong>.

  <script>
  // You can also require other files to run in this process
  require('./renderer.js')
  </script>
  </body>
</html>
```

2、renderer.js

```
// This file is required by the index.html file and will
// be executed in the renderer process for that window.
```

```
// All of the Node.js APIs are available in this process.

const { ipcRenderer } = require('electron')

let i = 1
setInterval( () => {
 console.log(i)
 i++
}, 1000)

document.getElementById('talk').addEventListener('click', e => {

 // ipcRenderer.send( 'channel1', 'Hello from main window')

 let response = ipcRenderer.sendSync( 'sync-message', 'Waiting for response')
 console.log(response)

})

ipcRenderer.on( 'channel1-response', (e, args) => {
 console.log(args)
})

ipcRenderer.on( 'mailbox', (e, args) => {
 console.log(args)
})
```

3、main.js

```
// Modules
const {app, BrowserWindow, ipcMain} = require('electron')

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let mainWindow

// Create a new BrowserWindow when `app` is ready
function createWindow () {

 mainWindow = new BrowserWindow({
 width: 1000, height: 800, x: 100, y:140
```

```
  width: 1000, height: 800, x: 100, y:110,
  webPreferences: { nodeIntegration: true }
  })

  // Load index.html into the new BrowserWindow
  mainWindow.loadFile('index.html')

  // Open DevTools - Remove for PRODUCTION!
  mainWindow.webContents.openDevTools();

  mainWindow.webContents.on( 'did-finish-load', e => {

  // mainWindow.webContents.send( 'mailbox', {
  // from: 'Ray',
  // email: 'ray@stackacademy.tv',
  // priority: 1
  // })
  })

  // Listen for window being closed
  mainWindow.on('closed', () => {
  mainWindow = null
  })
 }

 ipcMain.on( 'sync-message', (e, args) => {
  console.log(args)

  setTimeout( () => {
  e.returnValue = 'A sync response from the main process'
  }, 4000)

 })

 ipcMain.on( 'channel1', (e, args) => {
  console.log(args)
  e.sender.send( 'channel1-response', 'Message received on "channel1". Thank you!')
 })

 // Electron `app` is ready
 app.on('ready', createWindow)
```

```
53.
54.  // Quit when all windows are closed - (Not macOS - Darwin)
55.  app.on('window-all-closed', () => {
56.   if (process.platform !== 'darwin') app.quit()
57.  })
58.
59.  // When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
60.  app.on('activate', () => {
61.   if (mainWindow === null) createWindow()
62.  })
```

共享API (Shared API)

本节重点讲解 `process` 、 `screen` 、 `shell` 、 `nativeImage` 、 `clipboard` 几个部分内容。

1、process (进程)

1.1 index.html

```
1.  <!DOCTYPE html>
2.  <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-
    inline'">
6.   <title>Hello World!</title>
7.   </head>
8.   <body>
9.   <h1>Hello World!</h1>
10.
11.   <!-- All of the Node.js APIs are available in this renderer process. -->
12.   We are using Node.js <strong><script>document.write( process.versions.node)</script>
    </strong>,
13.   and Electron <strong><script>document.write( process.versions.electron )</script>
    </strong>.
14.
15.   <br><button type="button" onclick="process.hang()">Hang Renderer</button>
16.   <br><button type="button" onclick="process.crash()">Crash Renderer</button>
17.
```

```
  <script>
  // let i = 1
  // setInterval(() => {
  // console.log(i)
  // i++
  // }, 500)

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

1.2 main.js

```
// Modules
const {app, BrowserWindow} = require('electron')

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let mainWindow

// Create a new BrowserWindow when `app` is ready
function createWindow () {

  mainWindow = new BrowserWindow({
  width: 1000, height: 800,
  webPreferences: { nodeIntegration: true }
  })

  // Load index.html into the new BrowserWindow
  mainWindow.loadFile('index.html')

  // Open DevTools - Remove for PRODUCTION!
  mainWindow.webContents.openDevTools();

  mainWindow.webContents.on( 'crashed', mainWindow.reload )

  // Listen for window being closed
  mainWindow.on('closed', () => {
  mainWindow = null
  })
}
```

```
29.
30. // Electron `app` is ready
31. app.on('ready', createWindow)
32.
33. // Quit when all windows are closed - (Not macOS - Darwin)
34. app.on('window-all-closed', () => {
35.  if (process.platform !== 'darwin') app.quit()
36. })
37.
38. // When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
39. app.on('activate', () => {
40.  if (mainWindow === null) createWindow()
41. })
```

2、screen（屏幕）

1.1 main.js

```
1. // Modules
2. const electron = require('electron')
3. const {app, BrowserWindow} = electron
4.
5. // Keep a global reference of the window object, if you don't, the window will
6. // be closed automatically when the JavaScript object is garbage collected.
7. let mainWindow
8.
9. // Create a new BrowserWindow when `app` is ready
10. function createWindow () {
11.
12.  let primaryDisplay = electron.screen.getPrimaryDisplay()
13.
14.  mainWindow = new BrowserWindow({
15.  x: primaryDisplay.bounds.x, y: primaryDisplay.bounds.y,
16.  width: primaryDisplay.size.width/2, height: primaryDisplay.size.height,
17.  webPreferences: { nodeIntegration: true }
18.  })
19.
20.  // Load index.html into the new BrowserWindow
21.  mainWindow.loadFile('index.html')
22.
23.  // Open DevTools - Remove for PRODUCTION!
```

```
  mainWindow.webContents.openDevTools();

  // Listen for window being closed
  mainWindow.on('closed', () => {
  mainWindow = null
  })
}

// Electron `app` is ready
app.on('ready', createWindow)

// Quit when all windows are closed - (Not macOS - Darwin)
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') app.quit()
})

// When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
app.on('activate', () => {
  if (mainWindow === null) createWindow()
})
```

1.2 renderer.js

```
const electron = require('electron')

const displays = electron.screen.getAllDisplays()

console.log( `${displays[0].size.width} x ${displays[0].size.height}` )
console.log( `${displays[0].bounds.x}, ${displays[0].bounds.y}` )
console.log( `${displays[1].size.width} x ${displays[1].size.height}` )
console.log( `${displays[1].bounds.x}, ${displays[1].bounds.y}` )


electron.screen.on( 'display-metrics-changed', (e, display, metricsChanged) => {
  console.log( metricsChanged )
})

document.getElementsByTagName('body')[0].addEventListener( 'click', e => {
  console.log( electron.screen.getCursorScreenPoint() )
})
```

3、shell

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
 <title>Hello World!</title>
 </head>
 <body>
 <h1>Hello World!</h1>

 <button onclick="showSite()">Launch Electron.js Site</button><br>
 <button onclick="openSplash()">Open Splash.png</button><br>
 <button onclick="showSplashFile()">Show Splash.png</button><br>
 <button onclick="deleteSplashFile()">Delete Splash.png</button><br>

 <script>

 const { shell } = require('electron')

 const showSite = e => {
 shell.openExternal('https://electronjs.org')
 }


 const splashPath = `${__dirname}/splash.png`

 const openSplash = e => {
 shell.openItem(splashPath)
 }
 const showSplashFile = e => {
 shell.showItemInFolder(splashPath)
 }
 const deleteSplashFile = e => {
 shell.moveItemToTrash(splashPath)
 }

 </script>
 </body>
```

```
</body>
</html>
```

4、nativeImage（本地图片）

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
 <title>Hello World!</title>
 </head>
 <body>
 <h1>Convert splash.png:</h1>

 <button onclick="toPng()">PNG</button>
 <button onclick="toJpg()">JPG</button>
 <button onclick="toTag()">Show</button>
 <br><img src="" id="preview">


 <script>

 const fs = require('fs')
 const { nativeImage, remote } = require('electron')

 const splash = nativeImage.createFromPath(`${__dirname}/splash.png`)

 const saveToDesktop = (data, ext) => {

 let desktopPath = remote.app.getPath('desktop')
 fs.writeFile( `${desktopPath}/splash.${ext}`, data, console.log )
 }

 const toTag = e => {

 let size = splash.getSize()

 let splashURL = splash.resize({ width: size.width/4, height: size.height/4 }).toDataURL()
```

```
  document.getElementById('preview').src = splashURL
  }
  const toPng = e => {
  let pngSplash = splash.toPNG()
  saveToDesktop( pngSplash, 'png' )
  }
  const toJpg = e => {
  let jpgSplash = splash.toJPEG(100)
  saveToDesktop( jpgSplash, 'jpg' )
  }

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

5、clipboard（剪贴板）

5.1 index.html

```
 <!DOCTYPE html>
<html>
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
  <title>Hello World!</title>
  </head>
  <body>
  <h1>Hello World!</h1>

  <!-- All of the Node.js APIs are available in this renderer process. -->
  We are using Node.js <strong><script>document.write( process.versions.node)</script></strong>,
  and Electron <strong><script>document.write( process.versions.electron )</script></strong>.

  <br><button onclick="makeUpper()">Make clipboard uppercase</button>
  <br><button onclick="showImage()">Show clipboard image</button>

  <br><img src="" id="cbImage">
```

```
<script>

const { clipboard } = require('electron')

console.log( clipboard.readText() )

const showImage = e => {
let image = clipboard.readImage()
document.getElementById('cbImage').src = image.toDataURL()
}

const makeUpper = e => {
let cbText = clipboard.readText()
clipboard.writeText( cbText.toUpperCase() )
}

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

5.2 main.js

```
// Modules
const {app, BrowserWindow, clipboard} = require('electron')

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let mainWindow

// Create a new BrowserWindow when `app` is ready
function createWindow () {

clipboard.writeText('Hello from the main process!')

mainWindow = new BrowserWindow({
width: 1000, height: 800,
webPreferences: { nodeIntegration: true }
})

// Load index.html into the new BrowserWindow
mainWindow.loadFile('index.html')
```

```
20.
21.  // Open DevTools - Remove for PRODUCTION!
22.  mainWindow.webContents.openDevTools();
23.
24.  // Listen for window being closed
25.  mainWindow.on('closed', () => {
26.  mainWindow = null
27.  })
28. }
29.
30. // Electron `app` is ready
31. app.on('ready', createWindow)
32.
33. // Quit when all windows are closed - (Not macOS - Darwin)
34. app.on('window-all-closed', () => {
35.  if (process.platform !== 'darwin') app.quit()
36. })
37.
38. // When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
39. app.on('activate', () => {
40.  if (mainWindow === null) createWindow()
41. })
```

特性和技巧

本节我们来学习一下 `Electron` 其他特性和使用技巧。

1、Offscreen 渲染

```
1.  // Modules
2.  const {app, BrowserWindow} = require('electron')
3.  const fs = require('fs')
4.
5.  // Keep a global reference of the window object, if you don't, the window will
6.  // be closed automatically when the JavaScript object is garbage collected.
7.  let mainWindow
8.
9.  app.disableHardwareAcceleration()
10.
11. // Create a new BrowserWindow when `app` is ready
```

```
// Create a new BrowserWindow when `app` is ready
function createWindow () {

  mainWindow = new BrowserWindow({
  width: 1000, height: 800,
  show: false,
  webPreferences: {
  nodeIntegration: true,
  offscreen: true
  }
  })

  // Load index.html into the new BrowserWindow
  mainWindow.loadURL('https://electronjs.org')

  let i = 1
  mainWindow.webContents.on('paint', (e, dirty, image) => {

  let screenshot = image.toPNG()
  fs.writeFile( app.getPath('desktop') + `/screenshot_${i}.png`, screenshot, console.log
)
  i++
  })

  mainWindow.webContents.on('did-finish-load', e => {
  console.log( mainWindow.getTitle() )

  mainWindow.close()
  mainWindow = null
  })

  // Open DevTools - Remove for PRODUCTION!
  // mainWindow.webContents.openDevTools();

  // Listen for window being closed
  // mainWindow.on('closed', () => {
  // mainWindow = null
  // })
}

// Electron `app` is ready
```

```
app.on('ready', createWindow)

// Quit when all windows are closed - (Not macOS - Darwin)
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') app.quit()
})

// When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
app.on('activate', () => {
  if (mainWindow === null) createWindow()
})
```

2、网络检测 (Network Detection)

```
<!DOCTYPE html>
<html>
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
  <title>Hello World!</title>
  </head>
  <body>
  <h1>App is: <u id="status"></u></h1>

  <!-- All of the Node.js APIs are available in this renderer process. -->
  We are using Node.js <strong><script>document.write( process.versions.node)</script></strong>,
  and Electron <strong><script>document.write( process.versions.electron )</script></strong>.

  <script>

  const setStatus = status => {
  const statusNode = document.getElementById('status')
  statusNode.innerText = status ? 'online' : 'offline'
  }

  setStatus( navigator.onLine )

  window.addEventListener('online', e => {
```

```
  setStatus(true)
  })
  window.addEventListener('offline', e => {
  setStatus(false)
  })

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

3、提醒（Notivications）

```
 <!DOCTYPE html>
 <html>
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
  <title>Hello World!</title>
  </head>
  <body>
  <h1>Hello World!</h1>

  <!-- All of the Node.js APIs are available in this renderer process. -->
  We are using Node.js <strong><script>document.write( process.versions.node)</script></strong>,
  and Electron <strong><script>document.write( process.versions.electron )</script></strong>.

  <script>

  const { remote } = require('electron')

  const self = remote.getCurrentWindow()

  setTimeout(() => {

  let notification = new Notification( 'Electron App', {
  body: 'Some notification info!'
  })

```

```
  notification.onclick = e => {
  if( ! self.isVisible() ) self.show()
  }

  }, 2000)

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

4、预加载脚本（Preload Scripts）

4.1 index.html

```
 <!DOCTYPE html>
 <html>
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'">
  <title>Hello World!</title>
  </head>
  <body>
  <h1>Hello World!</h1>

  <!-- All of the Node.js APIs are available in this renderer process. -->
  We are using Node.js <strong><script>document.write( versions.node)</script></strong>,
  and Electron <strong><script>document.write( versions.electron )</script></strong>.

  <br><textarea id="content" rows="8" cols="80"></textarea>
  <br><button id="save" onclick="saveText()">Save Content</button>

  <script>

  const saveText = e => {

  const text = document.getElementById('content').value

  writeToFile( text )
  }

```

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

4.2 main.js

```
// Modules
const {app, BrowserWindow} = require('electron')

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let mainWindow

// Create a new BrowserWindow when `app` is ready
function createWindow () {

 mainWindow = new BrowserWindow({
 width: 1000, height: 800,
 webPreferences: {
 nodeIntegration: false,
 contextIsolation: false,
 preload: __dirname + '/preload.js'
 }
 })

 // Load index.html into the new BrowserWindow
 mainWindow.loadFile('index.html')

 // Open DevTools - Remove for PRODUCTION!
 mainWindow.webContents.openDevTools();

 // Listen for window being closed
 mainWindow.on('closed', () => {
 mainWindow = null
 })
}

// Electron `app` is ready
app.on('ready', createWindow)

// Quit when all windows are closed - (Not macOS - Darwin)
```

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

// When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
app.on('activate', () => {
  if (mainWindow === null) createWindow()
})
```

4.3 preload.js

```
const { remote } = require('electron')
const fs = require('fs')

const desktopPath = remote.app.getPath('desktop')

window.writeToFile = text => {
  fs.writeFile( desktopPath + '/app.txt', text, console.log )
}


window.versions = {
  node: process.versions.node,
  electron: process.versions.electron
}
```

5、进度条（Progress Bar）

```
// renderer.js

const { remote } = require('electron')

const self = remote.getCurrentWindow()

let progress = 0.01

let progressInterval = setInterval(() => {

  self.setProgressBar(progress)

  if (progress <= 1) {
  progress += 0.01
```

```
  } else {
  self.setProgressBar(-1)
  clearInterval(progressInterval)
  }
 }, 75)
```

项目简介

本项目是应用 Electron + Vue.js 完成一个网站收集和网站浏览的功能。具体包括网站添加、网站浏览、列表项目删除、内容搜索、菜单定制及项目打包等功能。

1、目录：

01-环境搭建

02-构建项目基本结构

03-添加信息

04-获得屏幕快照

05-显示列表

06-打开网站窗口

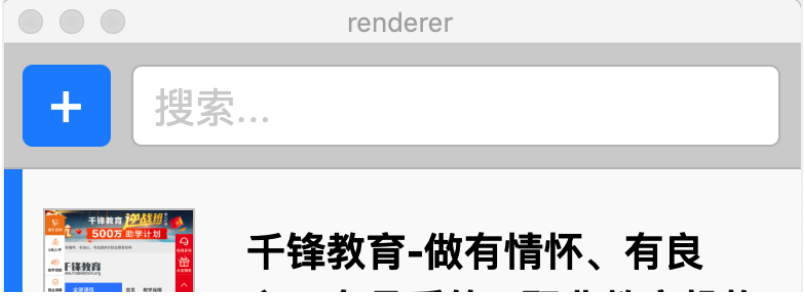
07-删除信息

08-搜索信息

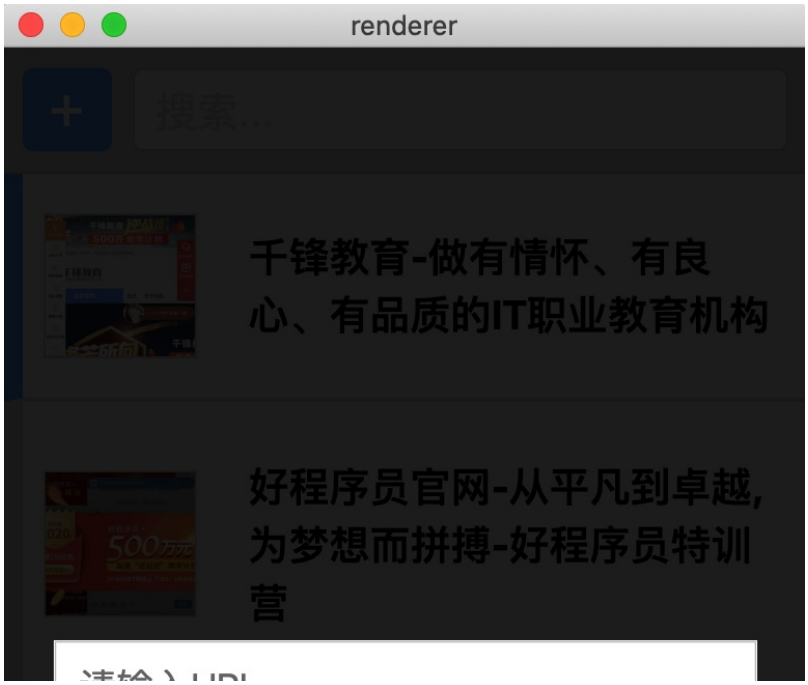
09-定制菜单

10-项目打包部署

2、项目部分截图




心、有品质的IT职业教育机构
好程序员官网-从平凡到卓越，为梦想而拼搏-好程序员特训营
扣丁学堂千锋教育旗下IT在线教育平台-在线培训学习直播视频课程
权威的HTML5培训课程|WEB前端培训H5培训-千锋教育


renderer
搜索...
千锋教育-做有情怀、有良心、有品质的IT职业教育机构
好程序员官网-从平凡到卓越，为梦想而拼搏-好程序员特训营

3、项目地址

```
https://github.com/lurongtao/felixbooks-electron
```

环境搭建

1、搭建 Electron 环境

在你认为合适的目录下 创建 readit-vue 目录，在终端命令行里输入命令：

```
cd 你认为合适的目录/readit-vue
npm init -y
npm install electron@latest -D
```

2、创建 main.js 文件

在项目根目录下创建 main.js 文件：

```
// /main.js

// Modules
const {app, BrowserWindow} = require('electron')
```

```
const windowStateKeeper = require('electron-window-state')

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let mainWindow

// Create a new BrowserWindow when `app` is ready
function createWindow () {

 // Win state keeper
 let state = windowStateKeeper({
 defaultWidth: 500, defaultHeight: 650
 })

 mainWindow = new BrowserWindow({
 x: state.x,
 y: state.y,
 width: state.width,
 height: state.height,
 minWidth: 350,
 maxWidth: 650,
 minHeight: 300,
 webPreferences: {
 nodeIntegration: true
 }
 })

 // Load local vue server into the new BrowserWindow
 mainWindow.loadURL('http://localhost:8080')

 // Manage new window state
 state.manage(mainWindow)

 // Open DevTools - Remove for PRODUCTION!
 mainWindow.webContents.openDevTools();

 // Listen for window being closed
 mainWindow.on('closed', () => {
 mainWindow = null
 })
}
```

```
}

// Electron `app` is ready
app.on('ready', createWindow)

// Quit when all windows are closed - (Not macOS - Darwin)
app.on('window-all-closed', () => {
 if (process.platform !== 'darwin') app.quit()
})

// When app icon is clicked and app is running, (macOS) recreate the BrowserWindow
app.on('activate', () => {
 if (mainWindow === null) createWindow()
})
```

3、搭建 Vue 环境，启动 Vue 服务

在命令行里输入：

```
vue create vue-renderer
cd vue-renderer
yarn serve
```

4、配置 package.json npm 脚本

```
// /package.json
{
 // ...
 "scripts": {
 "start": "nodemon --exec 'electron .'"
 }
}
```

5、启动应用

```
npm start
```

构建 Vue 项目基本结构

准备 Header，Main，和 Modal 三个组件。

1、reset.css 样式

编写 reset.css 样式：

```
/* /src/assets/styles/reset.css */

html, body {
 height: 100%;
}

body {
 font: caption;
 margin: 0;
 display: flex;
 flex-flow: column;
}
```

2、App 根组件

编辑 `/renderer/src/App.vue` ：

```
<template>
 <div>
 <Header></Header>
 <Main></Main>
 <Modal></Modal>
 </div>
</template>

<script>
import Header from './components/Header'
import Main from './components/Main'
import Modal from './components/Modal'

export default {
 components: {
 Header,
 Main,
 Modal
 }
}
```

```
}
</script>

<style lang='stylus' scoped>
div
 height 100%
 position relative
</style>
```

3、Header 组件

在 components 文件夹下创建 Header.vue 组件： /src/components/Header.vue

```
<template>
 <header>
 <button id="show-modal">+</button>
 <input type="text" id="search" placeholder="Search">
 </header>
</template>

<script>
export default {

}
</script>

<style lang='stylus' scoped>
button {
 background: dodgerblue;
 color: white;
 border-radius: 5px;
 border: none;
 font-size: 20px;
 outline: none;
}

input {
 font-size: 20px;
 border-radius: 5px;
 border: 1px solid silver;
 padding: 0 10px;
}
```

```
}

input::placeholder {
 color: lightgray;
}

header {
 background: lightgray;
 display: flex;
 padding: 10px;
 font-weight: bold;
 border-bottom: 1px solid silver;
 box-shadow: 0px 10px 10px rgba(0,0,0,0.1);
}

#show-modal {
 padding: 0px 12px 5px;
 margin-right: 10px;
 font-size: 30px;
}

#search {
 flex-grow: 1;
}
</style>
```

4、Main 组件

在 components 文件夹下创建 Main.vue 组件： /src/components/Main.vue

```
<template>
 <main>
 <p id="no-items">No Items</p>
 <div id="items"></div>
 </main>
</template>

<script>
export default {

}
</script>
```

```
</script>

<style lang='stylus' scoped>
#items {
 flex-grow: 1;
}

#no-items {
 font-weight: bold;
 color: silver;
 text-align: center;
 width: 100%;
 position: absolute;
 top: 100px;
 z-index: -1;
}
</style>
```

5、modal 组件

在 components 文件夹下创建 Modal.vue 组件： /src/components/Modal.vue

```
<template>
 <div id="modal">
 <input type="text" id="url" placeholder="Enter URL">
 <button id="add-item">Add Item</button>
 <button id="close-modal">Cancel</button>
 </div>
</template>

<script>
export default {

}
</script>

<style lang='stylus' scoped>
#modal {
 position: absolute;
 top 0;
 left 0;
 width: 100%;
```

```
  width: 100%;
  height: 100%;
  background: rgba(0,0,0,0.85);
  display: flex;
  align-items: center;
  align-content: center;
  flex-wrap: wrap;
  display: none;
}

#url {
  flex-grow: 1;
  width: 100%;
  margin: 0 25px 15px;
  padding: 10px;
}

#modal button {
  padding: 10px;
}

#close-modal {
  background: white;
  color: black;
  margin-left: 15px;
}

#add-item {
  margin-left: 25px;
}

.read-item {
  display: flex;
  align-items: center;
  align-content: center;
  border-bottom: lightgray 2px solid;
  background: #FAFAFA;
  padding: 10px;
}

.read-item img {
```

```
  width: 20%;
  margin-right: 25px;
 }
</style>
```

添加一个新的信息

1、创建 Store

1.1 编辑 store

编辑 `/vue-renderer/src/store/index.js` ：

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
  isShowModal: false
  },

  mutations: {
  setModalVisible(state, show) {
  state.isShowModal = show
  }
  },

  actions: {
  setModalVisible({commit}, show) {
  commit('setModalVisible', show)
  }
  }
})
```

1.2 引入 store

编辑 `/vue-renderer/src/main.js` ：

```
// ...
```

```
import store from './store'
// ...
```

2、显示添加窗口

编辑 `/vue-renderer/src/components/Header.vue` :

```
<template>
 <header>
 <button id="show-modal" @click="setModalVisible(true)">+</button>
 // ...
 </header>
</template>

<script>
import { mapActions } from 'vuex'
export default {
 methods: {
 ...mapActions(['setModalVisible'])
 },
}
</script>
```

3、完善添加模态组件

编辑 `/vue-renderer/src/components/Modal.vue` :

```
<template>
 <div id="modal" v-show="isShowModal">
 <input type="text" id="url" :disabled="status" v-model="url" placeholder="输入 URL ...">
 <button id="add-item" :class="{disabled: status}" :disabled="status" @click="addItem">{{addButtonText}}</button>
 <button id="close-modal" v-show="!status" @click="setModalVisible(false)">取消</button>
 </div>
</template>

<script>
import { mapState, mapActions } from 'vuex'

```

```
export default {
 data() {
 return {
 url: '',
 status: false,
 addButtonText: '添加'
 }
 },

 created() {

 // Listen for new item from main process
 ipcRenderer.on('new-item-success', (e, newItem) => {
 console.log(newItem)

 this.status = false
 this.addButtonText = '添加'
 this.url = ''

 this.setModalVisible(false)
 })
 },

 computed: {
 ...mapState(['isShowModal'])
 },

 methods: {
 ...mapActions(['setModalVisible']),

 addItem() {
 if (this.url !== '') {

 // Send new item url to main process
 ipcRenderer.send('new-item', this.url)

 this.status = true
 this.addButtonText = '添加中...'
 }
 }
 }
```

```
}
</script>
```

4、完善主进程 main.js

编辑 `/main.js` ，在文件代码中的最外层添加 `ipcMain` 的 `new-item` 时间监听，重点是 `ipc` 通信：

```
//...

// Modules
const { ipcMain } = require('electron')

// Listen for new item request
ipcMain.on('new-item', (e, itemUrl) => {

 // Get new item and send back to renderer
 setTimeout(() => {
 e.sender.send('new-item-success', 'New item from main process')
 }, 2000)
})

// ...
```

获得屏幕快照

1、完善主进程处理

从渲染进程中拿到 `url` 后，通过 `offscreen` 获取屏幕快照。
在项目根目录下，创建 `readItem.js` ：

```
// /readItems

// Modules
const {BrowserWindow} = require('electron')

// Offscreen BrowserWindow
let offscreenWindow

// Exported readItem function
```

```
module.exports = (url, callback) => {

  // Create offscreen window
  offscreenWindow = new BrowserWindow({
  width: 500,
  height: 500,
  show: false,
  webPreferences: {
  offscreen: true
  }
  })

  // Load item url
  offscreenWindow.loadURL(url)

  // Wait for content to finish loading
  offscreenWindow.webContents.on('did-finish-load', e => {

  // Get page title
  let title = offscreenWindow.getTitle()

  // Get screenshot (thumbnail)
  offscreenWindow.webContents.capturePage( image => {

  // Get image as dataURL
  let screenshot = image.toDataURL()

  // Execute callback with new item object
  callback({ title, screenshot, url })

  // Clean up
  offscreenWindow.close()
  offscreenWindow = null
  })
  })
 }
```

2、更新 main.js

在 `/main.js` 文件里添加对 `readItem.js` 的引用：

```
// Modules
// ...
const readItem = require('./readItem')

// ...

// Listen for new item request
ipcMain.on('new-item', (e, itemUrl) => {

 // remove all codes here.

 // Get new item and send back to renderer
 readItem( itemUrl, item => {
 e.sender.send('new-item-success', item)
 })
})
```

显示列表

屏幕快照的图片获取生成以后，将返回的信息显示在列表里。

1、重新规划 Store

重新规划 Store，使用 `Vuex` 模块来分开管理数据。在 `/src/store/` 创建 `modules` 文件夹，在文件里创建 `main.js` 与 `modal.js` 两个文件。将 `/src/store/index.js` 文件里的代码迁移到 `modal.js` 里，做修改。三个文件的内容如下：

1.1 index.js

修改 `/src/store/index.js` ：

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

import modal from './modules/modal'
import main from './modules/main'

export default new Vuex.Store({
```

```
  modules: {
  modal,
  main
  }
})
```

1.2 modal

编辑 /src/store/modules/modal.js

```
const state = {
  isShowModal: false
}

const mutations = {
  setModalVisible(state, show) {
  state.isShowModal = show
  }
}

const actions = {
  setModalVisible({commit}, show) {
  commit('setModalVisible', show)
  }
}

export default {
  state,
  mutations,
  actions
}
```

1.3 main.js

编辑 `main.js` ，提供 Main.vue 管理的数据：

```
import store from 'store'

const state = {
  items: []
}

const mutations = {
```

```
  setItems(state, item) {

  state.items.push({
  id: new Date().getTime(),
  ...item
  })

  // 数据缓存
  store.set('items', state.items)
  },

  initItems(state, items) {
  state.items = items
  }
 }

 const actions = {
  setItems({commit}, item) {
  commit('setItems', item)
  },

  initItems({commit}, items) {
  commit('initItems', items)
  }
 }

 export default {
  state,
  mutations,
  actions
 }
```

2、修改 Modal.vue

`/src/components/Modal.vue` 获取到数据后，装填到 `Store` 中：

```
<script>
import { mapState, mapActions } from 'vuex'
export default {
 // ...
```

```
  created() {
  ipcRenderer.on('new-item-success', (e, newItem) => {

  this.setItems(newItem)

  // ...
  })
  },

  methods: {
  ...mapActions(['setModalVisible', 'setItems'])

  // ...
  },
 }
</script>
```

3、修改 Main.vue 组件

修改 `/src/components/Main.vue` 组件，用来响应的显示 `Store` 里的 `items` 数据。

```
<template>
 <main>
 <p id="no-item">暂无数据。</p>
 <div id="items">
 <div
 v-for="(item, index) in items"
 :key="item.id"
 class="read-item"
 :class="{selected: index === currentIndex}"
 @click="changeIndex(index)"
 >
 <img :src="item.screenshot" alt="item.title">
 <h2>{{item.title}}</h2>
 </div>
 </div>
 </main>
</template>

<script>
import { mapState, mapActions, mapGetters } from 'vuex'
```

```
import store from 'store'

export default {
 data() {
 return {
 currentIndex: 0
 }
 },

 created() {
 let items = store.get('items') || []
 this.initItems(items)
 },

 computed: {
 ...mapState({
 items: state => state.main.items
 })

 // ...
 },

 methods: {
 ...mapActions(['initItems']),

 changeIndex(index) {
 this.currentIndex = index
 }
 },
}
</script>

<style lang='stylus' scoped>
#items
 flex-grow 1

#no-item
 font-weight bold
 color silver
 text-align center
 width 100%
```

```
  position absolute
  top 100px
  z-index -1

.read-item
  display flex
  align-items center
  align-content center
  border-bottom lightgray 2px solid
  background #fafafa
  padding 10px
  border-left 10px solid lightgray
  -webkit-user-select none
  img
  width 20%
  margin-right 25px
  border solid 1px #ccc
  &:hover
  background #eee
  &.selected
  border-left-color dodgerblue
</style>
```

打开网站窗口

点击一个条目，根据获取到的网站 `URL` 信息，打开网站窗口。

1、修改 Main.vue

在 `/src/components/Main.vue` 组件里，给每个条目添加双击事件，双击后打开网站窗口，同时注入一段 `JS` 代码：

```
<template>
  <main>
  // ...
  <div id="items">
  <div
  // ...
  @dblclick="open(item.url, index)"
```

```
  >
  // ...
  </div>
  </div>
  </main>
</template>

<script>

// ...

import buttonJS from './button'

export default {
  // ...
  methods: {

  // ...

  open(url, index) {
  let readerWin = window.open(url, '', `
  maxWidth=2000,
  maxHeight=2000,
  width=1250,
  height=800,
  backgroundColor=#dedede,
  nodeIntegration=1,
  contextIsolation=1
  `)

  readerWin.eval(buttonJS)
  }
  }
}
</script>
```

2、Button.js

创建 `/src/components/button.js` ，编写要注入的 `JS` 代码：

```
export default `
```

```
  alert( hello. )
`
```

删除信息

在打开的窗口里注入按钮，点击按钮关闭窗口，同时删除相应的条目。

1、在打开的窗口中注入按钮

修改 `/src/components/button.js` ，编写创建的按钮 `JS` 代码，同时修改注入语句，将被点击条目的 `index` 值传递到窗口的按钮上。

1.1 button.js

```
export default `
 let readitClose = document.createElement('div')
 readitClose.innerText = '关闭窗口'

 readitClose.style.position = 'fixed'
 readitClose.style.bottom = '100px'
 readitClose.style.right = '30px'
 readitClose.style.padding = '5px 10px'
 readitClose.style.fontSize = '14px'
 readitClose.style.background = 'dodgerblue'
 readitClose.style.fontWeight = 'bold'
 readitClose.style.color = 'white'
 readitClose.style.borderRadius = '5px'
 readitClose.style.cursor = 'default'
 readitClose.style.boxShadow = '2px 2px 2px rgba(0, 0, 0, 0.2)'

 readitClose.onclick = e => {
 window.opener.postMessage({
 action: 'delete-reader-item',
 itemIndex: {{index}}
 }, '*')
 }

 document.querySelector('body').appendChild(readitClose)
`
```

1.2 修改 Main.vue

```
<script>
export default {
 // ...
 methods: {
 // ...

 ...mapActions(['initItems', 'removeItem']),

 open(url, index) {
 // ...
 readerWin.eval(buttonJS.replace('{{index}}', index))
 }
 },
}
</script>
```

2、删除条目

介绍到用户点击打开的按钮消息后，执行关闭窗口和删除条目的操作。

2.1 编辑 `/src/components/Main.vue`：

```
<script>

export default {
 // ...
 created() {

 // ...
 window.addEventListener('message', e => {

 if (e.data.action === 'delete-reader-item') {

 // 删除条目
 this.removeItem(e.data.itemIndex)

 // 更新当前高亮的 currentIndex
 if (this.currentIndex > 0) this.currentIndex--

 // 关闭打开的窗口
 e.source.close()
```

```
  }
  })
  }
 }
</script>
```

2.2 修改 Store

修改 `/src/store/modules/main.js` ，添加删除数据的功能：

```
// ...

const mutations = {

 // ...

 removeItem(state, index) {
 state.items.splice(index, 1)

 store.set('items', state.items)
 }
}

const actions = {
 // ...

 removeItem({commit}, index) {
 commit('removeItem', index)
 }
}

// ...
```

搜索信息

搜索信息的思路：在 `/src/components/Header.vue` 组件里获取到用户从搜索框里的关键字(keyword)，保存在 `Store` 里，再做个 `getter` ，过滤 `items` 信息，修改 `Main.vue` 组件的渲染信息源。

1、定制 Store

修改 `src/store/modules/main.js` ：

```
// ...

const state = {
 // ...
 keywords: ''
}

const mutations = {
 // ...

 changeKeywords(state, keywords) {
 state.keywords = keywords
 }
}

const actions = {
 // ...

 changeKeywords({commit}, keywords) {
 commit('changeKeywords', keywords)
 }
}

const getters = {
 filteredItems(state) {
 if (state.keywords) {
 return state.items.filter((value, index) => {
 return value.title.indexOf(state.keywords) != -1
 })
 }

 return state.items
 }
}

export default {
 // ...

```

```
  getters
}
```

2、修改 Header.vue

处理 `/src/componnent/Header.vue` 的 keywords 信息获取与存储：

```
<template>
 <header>
  // ...
  <input type="text" @keyup.enter="searchItem" v-model="keywords" id="search" placeholder="搜索...">
 </header>
</template>

<script>
 import { mapActions } from 'vuex'
 export default {
  data() {
  return {
  keywords: ''
  }
  },

  methods: {
  ...mapActions(['setModalVisible', 'changeKeywords']),

  searchItem() {
  this.changeKeywords(this.keywords)
  }
  },
 }
 </script>
```

3、修改 Main.vue

修改 `/src/components/Main.vue`，获取关键字和修改数据渲染数据源。

```
<template>
 <main>
  // ...
```

```
  <div id= items >
  <div
  v-for="(item, index) in filteredItems"
  // ...
  >
  // ...
  </div>
  </div>
  </main>
 </template>

 <script>
 // ...
 export default {
  // ...

  computed: {
  // ...

  ...mapGetters(['filteredItems'])
  },

  // ...
 }
 </script>
```

定制菜单

本节为大家介绍如何为我们的应用定制一个菜单，让它看起来更像一个原生的桌面端APP。

1、载入菜单模块

在 `renderer` 的 `/public/index.html` 里载入菜单模块：

```
 <script>
  const { remote, shell } = require('electron')
 </script>
```

2、定制菜单

修改 `/src/App.vue`，在 `mounted` 里定制菜单：

```
<script>
// ...

export default {
 // ...
 mounted() {
 // Menu template
 const template = [
 {
 label: 'Items',
 submenu: [
 {
 label: 'Add New',
 click: () => {
 this.setModalVisible(true)
 },
 accelerator: 'CmdOrCtrl+O'
 }
 ]
 },
 {
 role: 'editMenu'
 },
 {
 role: 'windowMenu'
 },
 {
 role: 'help',
 submenu: [
 {
 label: 'Learn more',
 click: () => { shell.openExternal('https://github.com/stackacademytv/master-electron')
}
 }
 ]
 }
 ]

 // Set Mac-specific first menu item
```

```
39.   if (process.platform === 'darwin') {
40.
41.   template.unshift({
42.   label: remote.app.getName(),
43.   submenu: [
44.   { role: 'about' },
45.   { type: 'separator'},
46.   { role: 'services' },
47.   { type: 'separator'},
48.   { role: 'hide' },
49.   { role: 'hideothers' },
50.   { role: 'unhide' },
51.   { type: 'separator'},
52.   { role: 'quit' }
53.   ]
54.   })
55.   }
56.
57.   // Build menu
58.   const menu = remote.Menu.buildFromTemplate(template)
59.
60.   // Set as main app menu
61.   remote.Menu.setApplicationMenu(menu)
62.   }
63.  }
64. </script>
```

项目打包部署

1、基本概念

本节我们将介绍打包和分发我们的项目，内容包括代码签名和添加发布自动应用程序更新的功能。

Code sign

Auto updates

为此，我们将使用electron builder模块。electron Builder 已成为打包 electron 几乎所有我们需要的所有功能，包括一个非常简单的使用 `electron` 更新。

Update Server

我们应该听说将应用程序更新推送到官方的服务器。这里 `Mac` 应用程序商店需要一个专用的应用程序更新服务器。在配置和维护时，这通常增加了复杂性。

所以在使用electron Builder时，我们将看到如何实现将本地的应用

发布到 `Github` 服务器上，只使用 `Github` 更新服务器。

2、Eletron-Builder

```
npm install -g electron-builder
electron-builder -m zip
```

感谢

鸣谢：
千锋教育大前端教研院