# 一、Electron 跨平台桌面应用开发

## 1、Electron 的介绍

Electron 是由 Github 开发的一个跨平台的桌面应用开发框架，可以让我们用 html css js

的技术开发桌面上面可以安装的软件。

实际上,桌面应用的历史并不算久远,不去查找各种资料,仅凭记忆,我能想到的曾经出现

过的桌面应用编写语言就有:C++、Delphi、VB、winForm、WPF、swing、awt、QT、flash、

Objective-C、Swift...或许还有更多。

**截止 2020-9-16，** Electron Github 的 star 已经达到 85.8k，那么他的用户量可能已经超 过 100 万了。还有大家知道的 github 的 atom 编辑器、微软的 vscode 编辑器、Facebook

Messenger、Microsoft Teams、包括阿里内部的一些软件也是用 electron 开发的。既然这么

热，那就一个字：学。

Github 地址：https://github.com/electron/electron

官网：<https://www.electronjs.org/>

## 2、为什么要学 Electron

想起在网路里流传很广的一句话“不要和老夫说什么 C++，Java，老夫行走江湖就靠一把 JS，遇到需求撸起袖子就是干”。是的 Electron 可以让我们用 html Css Js 的前端技术来开发跨平 台的桌面软件。

**如果你有下面的需求建议学一下 Electron:**

1、公司有开发桌面应用的需求。 （Electron 开发速度快、开发周期短、跨平台 、维护成

本低）

2、前端开发者想提升自己的技能，有必要学习一下 Electron，大家都会 Electron，所以我们

也得学。 （我们不知道 不代表别人也不知道）

3、开发开源软件 推广自己的产品 （如：atom 如：vscode ）。

4、同样的功能 开发软件比开发网站更赚钱。 （网站： 4k- 5 万 app： 5 万-50 万

桌面软件：2 万-50 万元 ）

## 3、关于NW.js和Electron的介绍

NW.js 和 Electron 都可以用前端的知识来开发桌面应用。NW.js 和 Electron 起初是同一 个作者开发。后来种种原因分为两个产品。一个命名为 NW.js（英特尔公司提供技术支持）、

另一命名为 Electron（Github 公司提供技术支持）。

NW.js 和Electron 可以用 Nodejs 中几乎所有的模块。NW.js 和 Electron 不仅可以把 html写的 web 页面打包成跨平台可以安装到电脑上面的软件，也可以通过 javascript 访问操作

系统原生的 UI 和 Api（控制窗口、添加菜单项目、托盘应用菜单、读写文件、访问剪贴板）。

目前用的比较多的还是 electron。

## 4、搭建 Electron 的环境

准备工作：

1、电脑必须安装 Nodejs, 推荐安装最新稳定版本（https://nodejs.org/en/）

2、电脑上面推荐安装 git

安装 electron 环境

npm install -g electron

或者

cnpm install -g electron

查看 Electron 的版本

# 二、搭建Electron项目

## 1、安装Electron

|  |
| --- |
| npm install -g electron / cnpm install -g electron |

## 2、克隆一个仓库、快速启动一个项目

要使用git的话首先电脑上面需要安装git

|  |
| --- |
| **克隆示例项目的仓库**  git clone https://github.com/electron/electron-quick-start  **进入这个仓库**  cd electron-quick-start  **安装依赖并运行**  npm install && npm start |

## 3、electron-forge搭建一个electron项目

electron-forge相当于electron的一个脚手架，可以让我们更方便的创建、运行、打包electron项目。

**Github地址：**<https://github.com/electron-userland/electron-forge>

**官网地址：**<https://www.electronforge.io/>****Note****: Electron Forge requires Node 10 or above, plus git installed.

### 3.1 npx或yarn创建项目

|  |
| --- |
| npx create-electron-app my-new-app  或者  yarn create electron-app my-new-app |

### **3.2运行项目**

|  |
| --- |
| cd my-new-app  npm start |

### 3.3也可以使用下面方法创建项目（老方法）

|  |
| --- |
| npm install -g @electron-forge/cli  electron-forge init my-new-app  cd my-new-app  npm start |

### 3.4手动搭建一个electron项目

1. 新建一个项目目录 例如：**electrondemo01**
2. 在**electrondemo01**目录下面新建三个文件: index.html、main.js 、package.json
3. index.html 里面用css 进行布局（以前怎么写现在还是怎么写）
4. 在main.js 中写如下代码：

|  |
| --- |
| const { app, BrowserWindow } = require("electron")  const path = require("path")  const createWindow=()=>{  const mainWindow = new BrowserWindow({  width: 800,  height: 600,  });  mainWindow.loadFile(path.join(\_\_dirname, 'index.html'));  // mainWindow.loadURL('https://github.com');  }  //监听electron ready事件创建窗口  app.on('ready', createWindow);  //监听窗口关闭的事件，关闭的时候退出应用，macOs需要排除  app.on('window-all-closed', () => {  if (process.platform !== 'darwin') {  app.quit();  }  });  //Macos中点击dock中的应用图标的时候重新创建窗口  app.on('activate', () => {  if (BrowserWindow.getAllWindows().length === 0) {  createWindow();  }  }); |

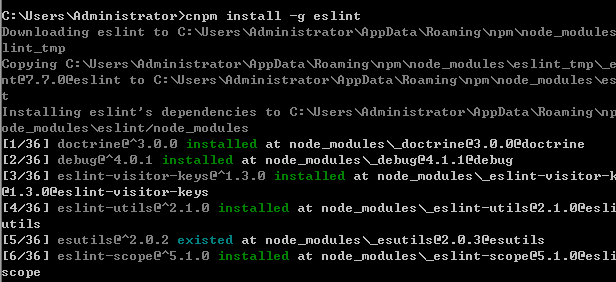
### 3.5运行

|  |
| --- |
| electron . 注意：命令后面有个点 |

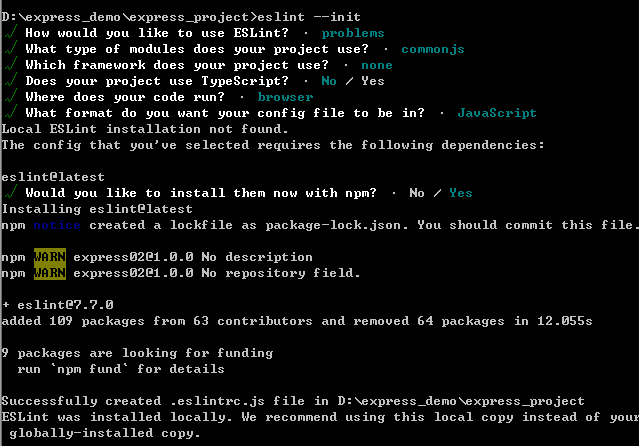
## 4、开发工具中配置Eslint

ESLint最初是由Nicholas C. Zakas 于2013年6月创建的开源项目。它的目标是**提供一个插件化的javascript代码检测工具**。

1. 安装cnpm install -g eslint



1. 在你的项目里面运行 eslint --init



# 三、Electron 运行流程、主进程渲染进程

## 1、Electron 运行的流程

## 2、Electron 主进程和渲染进程

**主进程和渲染器进程：**

package.json 中定义的入口被**称为主进程**。 在主进程中实例化 BrowserWindow 创建的

Web 页面被称为渲染进程。一个 Electron 应用**只有一个主进程，**但是可以有多个渲染进程，

每个 **Electron 中的 web 页面运行在它自己的渲染进程中**。

主进程使用 BrowserWindow 实例创建页面。 每个 BrowserWindow 实例都在自己的

渲染进程里运行页面。 当一个 BrowserWindow 实例被销毁后，相应的渲染进程也会被终

止。**进程（了解）**：进程（Process）是计算机中的程序关于某数据集合上的一次运行活动，是

系统进行资源分配和调度的基本单位，是操作系统结构的基础。

**线程（了解）：**在⼀一个程序里的一个执行路线就叫做线程（thread）。更准确的定义是：

线程是“一个进程内部的控制序列”**。**

**线程和进程（了解）：**一个程序至少有一个进程,一个进程至少有一个线程

## 3、Electron 主进程和渲染进程中使用 Nodejs 以及 Nodejs 第三方模块（注意）

Electron5.x 之前默认可以在主进程以及渲染进程中直接使用 nodejs, 但是在 Electron5.x

默认没法在渲染进程中直接使用 nodejs，如果我们想在渲染进程中使用 nodejs 的话需要进

行如下配置。

https://www.electronjs.org/docs/api/browser-window

mainWindow=new BrowserWindow({width:800,height:600, webPreferences: {

nodeIntegration: true,

contextIsolation:false

}});

### 3.1.1渲染进程中使用 Nodejs 读取本地文件

var fs = require('fs');

var content = document.getElementById('content');

var button = document.getElementById('button');

button.addEventListener('click',function(e){

fs.readFile('package.json','utf8',function(err,data){

content.textContent = data;

console.log(data);

});

});

### 3.1.2渲染进程和主进程中使用 Nodejs 第三方模块

1. 安装模块
2. 引入模块
3. 参考第三方文档使用

## 4、Electron 开启调试模式

mainWindow.webContents.openDevTools();

# 四、Electron 调用 h5 的拖放 api 结合 Nodejs fs 模块实现拖放打开文件功能

## 4.1拖拽事件的定义和用法：

ondragover 事件在可拖动元素或选取的文本正在拖动到放置目标时触发。

默认情况下，数据/元素不能放置到其他元素中。 如果要实现改功能，我们需要防止元

素的默认处理方法。我们可以通过调用 event.preventDefault() 方法来实现 ondragover 事

件。

**注意**： 为了让元素可拖动，需要使用 HTML5draggable 属性。

**提示**： 链接和图片默认是可拖动的，不需要 draggable 属性。

## 4.2在拖放的过程中会触发以下事件：

**在拖动目标上触发事件 (源元素):**

ondragstart - 用户开始拖动元素时触发

ondrag - 元素正在拖动时触发

ondragend - 用户完成元素拖动后触发

**释放目标时触发的事件:**

ondragenter - 当被鼠标拖动的对象进入其容器范围内时触发此事件ondragover - 当某被拖动的对象在另一对象容器范围内拖动时触发此事件

ondragleave - 当被鼠标拖动的对象离开其容器范围内时触发此事件

ondrop - 在一个拖动过程中，释放鼠标键时触发此事件

**注意**： 在拖动元素时，每隔 350 毫秒会触发 ondragover 事件。

## 4.3 Electron 拖放打开文件

var fs=require('fs');

var content=document.querySelector('#content');

content.ondragenter=content.ondragover=content.ondragleave=function(){

return false; /\*阻止默认行为\*/

}

content.ondrop=function(e){

//阻止默认行为

e.preventDefault();

var path=e.dataTransfer.files[0].path;

fs.readFile(path,'utf-8',(err,data)=>{

if(err){

console.log(err);

return false;

}

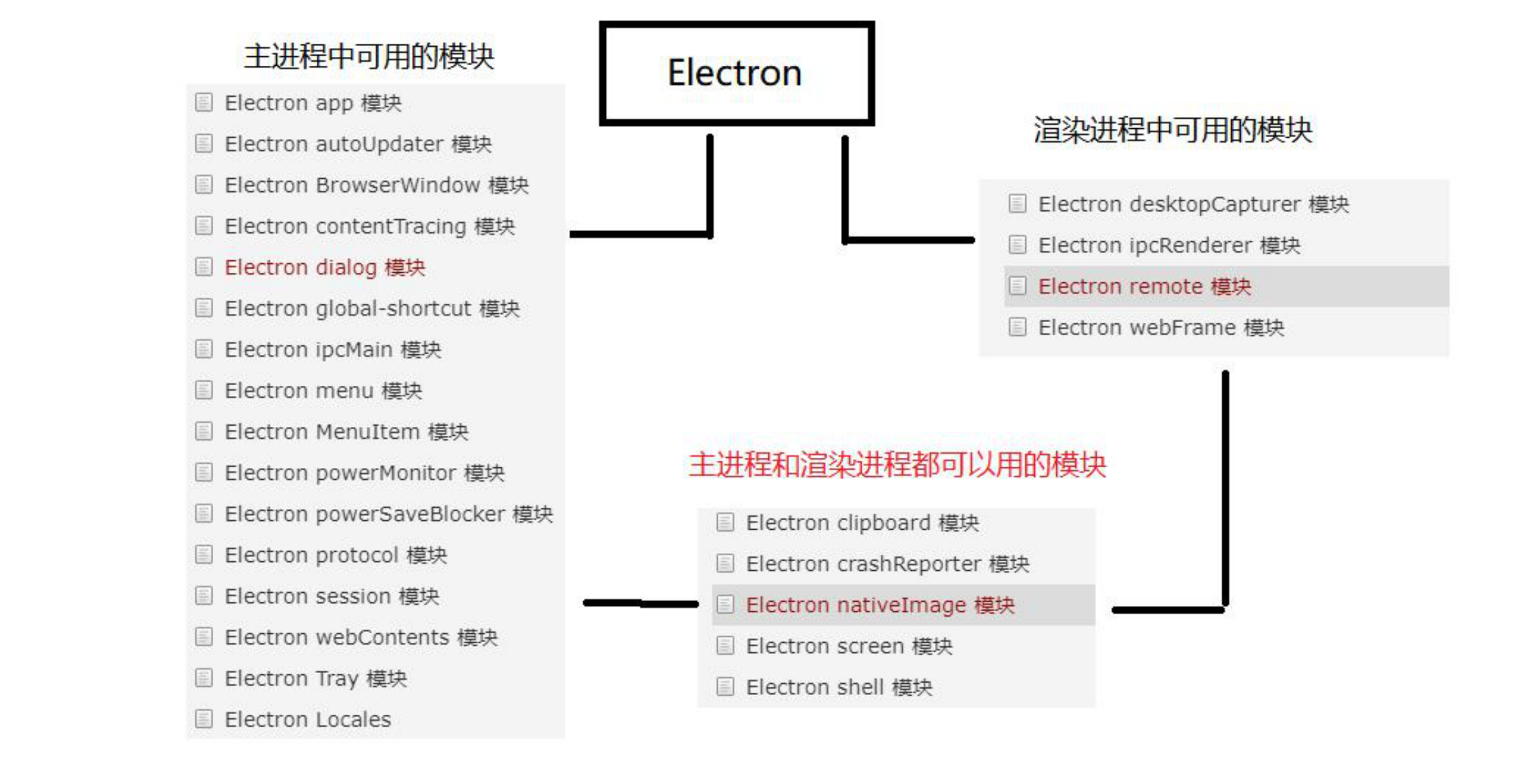
content.innerHTML=data;

})

}

# 五、Electron 模块介绍、remote 模块、通过 BrowserWindow 打开新窗口

## 5.1、Electron 主进程和渲染进程中的模块（介绍）



## 5.2、Electron remote 模块

**remote 模块**提供了一种在渲染进程（网页）和主进程之间进行进程间通讯（IPC）的简便途径。

**Electron 中, 与 GUI 相关的模块（如 dialog, menu 等)只存在于主进程，而不在渲染进程中** 。

为了能从渲染进程中使用它们，需要用 ipc模块来给主进程发送进程间消息。**使用 remote 模**

**块，可以调用主进程对象的方法**，而无需显式地发送进程间消息，这类似于 Java 的 RMI。

**注意：**Electron10.x 以后要使用 remote 模块的话必须得在 BrowserWindow 中开启

参考：https://www.electronjs.org/docs/api/browser-window

mainWindow=new BrowserWindow({width:800,height:600,webPreferences: {

nodeIntegration: true,

enableRemoteModule: true,

}});

## 5.3、Electron 渲染进程中通过 remote 模块调用主进程中

## 的 BrowserWindow 打开新窗口

**主进程部分代码：**

mainWindow=new BrowserWindow({width:800,height:600,webPreferences: {

nodeIntegration: true,

enableRemoteModule: true, //要开启 remote 模块

}});

**渲染进程代码：**

const btn = document.querySelector('#btn');

const path = require('path');

const BrowerWindow = require('electron').remote.BrowserWindow;

btn.onclick = () => {

win = new BrowerWindow({

width: 300,

height: 200,

frame: false,

transparent: true

// fullscreen:true

})

win.loadURL(path.join('file:',\_\_dirname,'news.html'));

win.on('close',()=>{win = null});

}

# 六、Electron 自定义软件顶部菜单、右键菜 单以及绑定快捷键

## 6.1 Electron 自定义软件顶部菜单以及绑定快捷键

**Electron 中 Menu 模块**可以用来创建原生菜单，它可用作**应用菜单**和 **context 菜单**。

这个模块是一个**主进程的模块**，并且可以通过 remote 模块给渲染进程调用。const { Menu }=require('electron');

var menuTemplate=[

{

label:"文件",

submenu:[

{

label:'新建文件',

accelerator:'ctrl+n',

click:function(){

console.log('ctrl+n');

}

},

{

label:'新建窗口',

click:function(){

console.log('new window');

}

}

]

},

{

label:'编辑',

submenu:[

{

label:'复制',

role:'copy'

},

{

label:'截切',

role:'cut'

}

]

}

]

var menuBuilder=Menu.buildFromTemplate(menuTemplate)

Menu.setApplicationMenu(menuBuilder)

## 6.2 Electron 自定义右键菜单

const { remote } = require('electron');

const Menu = remote.Menu;

const contextMenuTemplate = [

{

label: '复制',

role: 'copy'

},

{

label: '黏贴',

role: 'paste'

},

{ type: 'separator' }, //分隔线

{

label: '其他功能',

click: () => {

console.log('click')

}

}

];

var menuBuilder = Menu.buildFromTemplate(contextMenuTemplate)

window.addEventListener('contextmenu', (e) => {

e.preventDefault();

menuBuilder.popup({ window: remote.getCurrentWindow() });

}, false);

# 七、Electron 主进程与渲染进程之间的通信(同步通信、异步通信)

有时候我们想在渲染进程中通过一个事件去执行主进程里面的方法。或者在渲染进程中通知 主进程处理事件，主进程处理完成后广播一个事件让渲染进程去处理一些事情。这个时候就

用到了主进程和渲染进程之间的相互通信。

**Electron 主进程和渲染进程的通信主要用到两个模块：ipcMain 和 ipcRendereripcMain：**当在主进程中使用时，它处理从渲染器进程（网页）发送出来的异步和同步信息, 当然也有可能从主进程向渲染进程发送消息。

**ipcRenderer：** 使用它提供的一些方法从渲染进程 (web 页面) 发送同步或异步的消息到主

进程。 也可以接收主进程回复的消息。

## 场景 1：渲染进程给主进程发送异步消息

//渲染进程

const { ipcRenderer } = require('electron')

ipcRenderer.send('msg',{name:'张三'}); //异步

//主进程

const { ipcMain } = require('electron');

ipcMain.on(''msg'',(event,arg) => {

})

## 场景 2：渲染进程给主进程发送异步消息，主进程接收到异步消息以后通知渲染进程

//渲染进程

const { ipcRenderer } = require('electron')

ipcRenderer.send('msg',{name:'张三'}); //异步

//主进程

const { ipcMain } = require('electron');

ipcMain.on('msg',(event,arg) => {

event.sender.send('reply', 'pong');

})

//渲染进程

const { ipcRenderer } = require('electron')

ipcRenderer.on('reply', function(event, arg) {

console.log(arg); // prints "pong"

});

## 场景 3：渲染进程给主进程发送同步消息//渲染进程

const { ipcRenderer } = require('electron')

const msg = ipcRenderer.sendSync('msg-a');

console.log(msg)

//主进程

ipcMain.on('msg-a',(event)=> {

event.returnValue = 'hello';

})

## 场景 4：主进程通知渲染进程执行操作

//主进程

BrowserWindow.getFocusedWindow().webContents.send('replay','new 111');

//渲染进程

const { ipcRenderer } = require('electron')

ipcRenderer.on('reply', function(event, arg) {

console.log(arg); // prints "pong"

});

# 八、Electron 渲染进程与渲染进程之间的通信

## 8.1 Electron渲染进程通过 localstorage给另一个渲染进程传值

localStorage.setItem(key,value);localStorage.getItem(key);

## 8.2 通过 BrowserWindow 和 webContents 模块实现渲染进程和渲染进程的通信

webContents 是一个 事件发出者.它负责渲染并控制网页，也是 BrowserWindow 对象的属

性。

https://www.electronjs.org/docs/api/web-contents

**需要了解的几个知识点：**

**1）获取当前窗口的 id：**

const winId = BrowserWindow.getFocusedWindow().id;

**2）监听当前窗口加载完成的事件**

win.webContents.on('did-finish-load',(event) => {

})

**3）同一窗口之间广播数据**

win.webContents.on('did-finish-load',(event) => {

win.webContents.send('msg',winId,'我是 index.html 的数据');

})

**4）通过 id 查找窗口**

let win = BrowserWindow.fromId(winId);

# 九、Electron Shell 模块在用户默认浏览器 **中打开 URL 以及 Electron webview**

## 9.1 Electron shell 模块 在用户默认浏览器中打开URL的

## 示例

https://www.electronjs.org/docs/api/shellshell 模块提供了集成其他桌面客户端的关联功能.

var {shell}=require('electron')

shell.openExternal('https://github.com');

## 9.2 Electron 中的 webview

https://www.electronjs.org/docs/api/webview-tag

Electron5.x 之后官方不建议使用 webview 标签，可以使用 iframe 替代 webview

<iframe id="myWebview" src="https://www.itying.com" style="position:fixed;width:

100%;height: 100%; border: none;" >

</iframe>

# 十、Electron dialog 弹出框

dialog 模块提供了 api 来展示原生的系统对话框，例如打开文件框，alert 框，

所以 web 应用可以给用户带来跟系统应用相同的体验。

https://www.electronjs.org/docs/api/dialog

dialog.showErrorBox('title','content');remote.dialog.showMessageBox({

type: 'info',

title: '提示信息',

message: '内容',

buttons: ['ok', 'no']

}).then(result => {

console.log(result);

if ( result.response == 0 ) {

console.log('You click ok.');

} else {

console.log('You click cancel');

}

}).catch(err => {

console.log(err)

})

remote.dialog.showOpenDialog({

properties:['openDirectory','openFile']

}).then(result => {

console.log(result.canceled)

console.log(result.filePaths)

}).catch(err => {

console.log(err)

})remote.dialog.showSaveDialog({

title: 'save file',

defaultPath: "aaa.jpg",

filters: [

{ name: 'Images', extensions: ['jpg', 'png', 'gif'] },

{ name: 'Movies', extensions: ['mkv', 'avi', 'mp4'] },

{ name: 'Custom File Type', extensions: ['as'] },

{ name: 'All Files', extensions: ['\*'] }

]

}).then(result => {

console.log(result.canceled)

console.log(result.filePath)

}).catch(err => {

console.log(err)

})

# 十一、Electron 系统托盘 托盘右键菜单、托 盘图标闪烁 点击右上角关闭按钮隐藏到托盘（仿杀毒软件）

## 11.1、Electron 创建任务栏图标以及任务栏图标右键菜单

var { Menu, Tray,app,BrowserWindow } = require('electron');

const path = require('path');

var appIcon = new Tray(path.join(\_\_dirname,'lover.png'));

const menu = Menu.buildFromTemplate( [

{

label: '设置',

click: function () {} //打开相应页面

},

{

label: '帮助',

click: function () {}

},

{

label: '关于',

click: function () {}

},

{

label: '退出',

click: function () {

//

BrowserWindow.getFocusedWindow().webContents().send('close-main-window');

app.quit();

}

}

])

appIcon.setToolTip('my best app');

appIcon.setContextMenu(menu);

## 11.2 监听任务栏图标的单击、双击事件

var { Menu, Tray,app,BrowserWindow } = require('electron');

var appIcon = new Tray(path.join(\_\_dirname,'lover.png'));

appIcon.on('double-click',()=>{

console.log(win);

win.show();

})

## 11.3 Electron 点击右上角关闭按钮隐藏任务栏图标

const win = BrowserWindow.getFocusedWindow();

win.on('close',(e)=>{

console.log(win.isFocused());

if(!win.isFocused()){

win=null;

}else{

e.preventDefault(); /\*阻止应用退出\*/

win.hide(); /\*隐藏当前窗口\*/

}

})

## 11.4 Electron 实现任务栏闪烁图标

timer=setInterval(function() {

count++;

if (count%2 == 0) {

appIcon.setImage(path.join(\_\_dirname,'empty.ico'))

} else {

appIcon.setImage(path.join(\_\_dirname,'lover.png'))

}

}, 500);

# 十二、Electron 实现消息通知、监听网络变 化、网络变化弹出通知框

## 12.1 Electron 实现消息通知

Electron 里面的消息通知是基于 h5 的通知 api 实现的。const option = {

title: 'title',

body: 'body',

icon: path.join('main-process/favicon2.ico')

}

const myNotification = new window.Notification(option.title,option);

myNotification.onclick = () =>{

console.log('clicked');

}

## 12.2 Electron 监听网络变化

window.addEventListener('online', function(){});

window.addEventListener('offline', function(){});

# 十三、Electron 注册全局快捷键 （globalShortcut）以及 clipboard 剪 切板事件以及 nativeImage 模块（实现类似播放器点击机器码自动复制功能）

## 13.1 Electron 注册全局快捷键（globalShortcut）

**var** app = require('app');

**var** globalShortcut = require('electron').globalShortcut;

app.on('ready', **function**() {

// Register a 'ctrl+x' shortcut listener.

**var** ret = globalShortcut.register('ctrl+x', **function**() {

console.log('ctrl+x is pressed');

})

**if** (!ret) {

console.log('registration failed');

} // Check whether a shortcut is registered.

console.log(globalShortcut.isRegistered('ctrl+x'));

});

app.on('will-quit', **function**() {

// Unregister a shortcut.

globalShortcut.unregister('ctrl+x');

// Unregister all shortcuts.

globalShortcut.unregisterAll();

});

## 13.2 clipboard 剪切板事件 clipboard 模块以nativeImage 模块

const { clipboard, nativeImage } = require('electron');

clipboard.writeText("这是一个机器码");

console.log(clipboard.readText());

let img = nativeImage.createFromPath('static/favicon2.ico');

clipboard.writeImage(img);

const imgDataURL = clipboard.readImage().toDataURL();

const img3 = new Image();

img3.src = imgDataURL;

document.body.appendChild(img3);

# 十四、Electron+Vue 跨平台桌面软件开发之 electron-vue 的使用

## 14.1 electron-vue 的一些资源

**Github 地址**：https://github.com/SimulatedGREG/electron-vue

**Electron-vue 文档**：https://simulatedgreg.gitbooks.io/electron-vue/content/cn/

## 14.2 electron-vue 环境搭建、创建项目

npm install -g vue-cli

vue init simulatedgreg/electron-vue my-project

cd my-project

yarn # or npm install

yarn run dev # or npm run dev

## 14.3 electron-vue 目录结构分析

**14.4 electron-vue 创建组件、配置路由**

看教程演示...

# 十五、electron-vue UI 框架ElementUi 的使用、 electron-vue 中使用 sass

**15.1、electron-vue UI 框架 ElementUi 的使用**

**地址**：http://element-cn.eleme.io/#/zh-CN

**15.2、electron-vue 中使用 sass**

**地址**：https://simulatedgreg.gitbooks.io/electron-vue/content/cn/using\_pre-processors.html

**安装 sass-loader：**

npm install --save-dev sass-loader node-sass

**vue 文件中修改 style 为如下代码：**

<style lang="scss">

body {

/\* SCSS \*/

}

</style>

# 十六、Electron、electron-vue 中隐藏顶部菜单，隐藏顶部最大化、最小化、关闭按钮 自定最大化、最小化 、关闭按钮

## 16.1 electron-vue 中隐藏顶部菜单

mainWindow.setMenu(null)

## 16.2 electron-vue 中隐藏关闭 最大化 最小化按钮

mainWindow = new BrowserWindow({

height: 620,

useContentSize: true,

width: 1280

frame: false /\*去掉顶部导航 去掉关闭按钮 最大化最小化按钮\*/

})

## 16.3 electron-vue 自定义关闭 最大化 最小化按钮

ipc.on('window-min',function(){

mainWindow.minimize();

})

//登录窗口最大化

ipc.on('window-max',function(){

if(mainWindow.isMaximized()){

mainWindow.restore();

}else{

mainWindow.maximize();

}

})

ipc.on('window-close',function(){

mainWindow.close();

})

## 16.4 electron-vue 自定义导航可拖拽

**可拖拽的 css:**

-webkit-app-region: drag;

**不可拖拽的 css:**

-webkit-app-region: no-drag;

# 十七、Electron操作nedb数据库

## 17.1 electron应用操作数据的几种方法：

1、远程api接口 （多个客户端公用一套数据） \*

2、连接远程数据库 (局域网内使用 多个客户端公用一套数据 ) （不建议使用）

3、连接本地数据库 （nedb sqlite） 应用本地保存数据（localstore 5M） 用户设置信息 qq聊天记录

## 17.2 electron-vue读写本地数据库文件

https://simulatedgreg.gitbooks.io/electron-vue/content/cn/savingreading-local-files.html

## 17.3 electron app模块详细：

https://www.w3cschool.cn/electronmanual/electronmanual-electronapp.html

## 17.4 Nedb 数据库文档

https://github.com/louischatriot/nedb

## 17.5 electron-vue中使用Nedb 数据库

### 17.5.1安装nedb数据库

cnpm install nedb --save

### 17.5.2新建一个src/renderer/datastore.js

import Datastore from 'nedb'

import path from 'path'

import { remote } from 'electron'

export default new Datastore({

autoload: true,

filename: path.join(remote.app.getPath('userData'), '/data.db')

})

### 17.5.3 src/renderer/main.js

import db from './datastore.js'

/\* 其它代码 \*/

Vue.prototype.$db = db

### 17.5.4在vue的组件里面实现数据的增加 修改 删除 显示

this.$db.insert({},function(){

})

this.$db.find({},function(){

//获取查询的数据

})

this.$db.update({条件},{$set:{更改的数据}},function(){

})

this.$db.remove({条件},{},function(){

})

# 十八、electron-vue 打包 Windows 上可以安 装的软件 以及修改软件名称 软件版本号等

## 18.1 electron 中构建应用最常用的模块

1、electron-packager

2、electron-builder

electron-packager 和 electron-builder 在自己单独创建的应用用也可以完成打包功能。但是由于配置太复杂所以我们不建议单独配置。

## 18.2 electron-forge

https://github.com/electron-userland/electron-forge

electron-forge package

electron-forge make

## 18.3、electron-vue

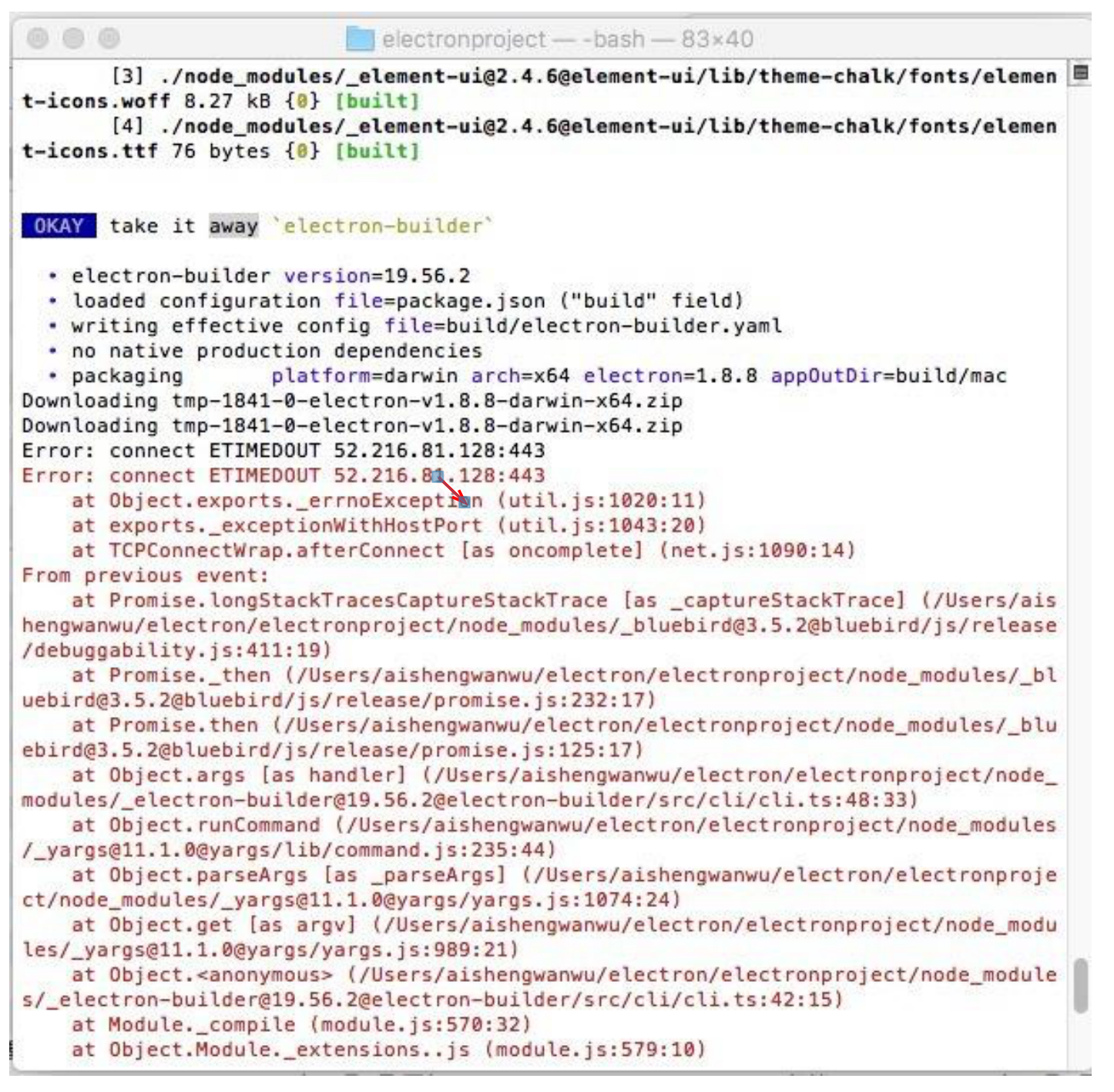
https://github.com/SimulatedGREG/electron-vuehttps://simulatedgreg.gitbooks.io/electron-vue/content/cn/using-electron-packager.

html

npm run build

修改应用名称、应用图标、应用版本号、应用appid3、 打包的时候可能遇到的错误。





### 18.3.1创建应用托盘的时候可能会遇到错误

(1) 把托盘图片放在根目录 static 里面，然后注意下面写法。

var tray = new Tray(path.join(\_\_static,'favicon.ico'));

(2) 如果托盘路径没有问题，还是包托盘相关错误的话，把托盘对应的图片换成.png

格式重试

### 18.3.2模块问题可能会遇到的错误解决方法：

(1) 删掉 node\_modules 然后重新用 npm install 安装依赖

(2) 如果上面方法还是解决不了。可以通过安装 yarn 用 yarn 来安装模块

(1) cnpm install -g yarn

(2) cd 到对于目录里面执行 yarn 替代（npm install /cnpm install）安装依赖

(3) 如果上面两个方法都不可行，用手机创建一个热点电脑连上热点。重试上面任

意一种方法