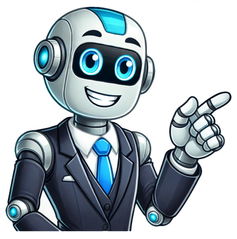
Continue



I have a question related to ICT terminology. Which is the correct/preferred preposition in the following sentence: Save the file to your portfolio/Save the file in your portfolio. Thanks I'd use "for" in here but you should give us more context because I don't understand what you are really implying. Thanks for your reply. I'm sorry I wasn't able to express myself clearly. The sentence will be used in a task in which pupils will be asked to record a short text and save the sound file to a portfolio (a folder) on the digital learning management platform that we use at school. I instinctively wrote "Save the file to your portfolio", but then I wasn't so sure. In various computer software manuals I have seen the phrase "save to file", and I suppose that is why I wrote "to". Prepositions are difficult in the first place, and with all this new computer technology as well, it is not easy to write correctly! In reference to a computer portfolio, you could say either "save in" or "save to". Thank you, I appreciate your help 最近在使用 Electron 做一个手写字体生成图片的工具。 不可避免的，遇到了通过Electron 往本地存文件的问题。 在Electron 中，存取本地文件，有很多种办法。本文介绍最常用的一种办法。 通过 Electron 框架提供的能力，和 Node.js 的 fs 文件管理模块实现本地文件的存取。使用 app.getPath 和 fs 模块存储文件首先，我们可以通过 app.getPath 来获取当前用户的 data 存放目录。 app.getPath('userData') 返回一个字符串，该字符串表示应用程序的用户数据目录的路径。这个目录通常是用来存储用户相关的数据，例如配置文件、缓存等。具体路径会根据操作系统而变化。windows 系统中， 会返回类似 (C:\Users\\AppData\Roaming\ ) 这样的结果。Mac 系统中，会返回 (/Users//Library/Application Support/) 这种结果。linux 系统中 则会返回 （/home//.config/ ） 这种结果。我们通过 node.js 的path 模块， 使用 path.join(app.getPath('userData'), 'myFile.txt') 就能得到一个完整的文件路径。 接着使用 fs 模块的 来写入内容即可。示例代码如下：const { app } = require('electron'); const fs = require('fs'); const path = require('path'); const filePath = path.join(app.getPath('userData'),'/myfile.txt'); // Example file path try { fs.writeFileSync(filePath, 'hello world', 'utf-8'); } catch (e) { console.error('Failed to save the file!'); } 这种做法，有一个问题。 那就是，用户不能在保存的时候，主动选择文件存放目录。使用 Dialog API 和 fs 模块配合Electron 提供了一个 Dialog API 来处理文件对话框。您可以专门调用一个保存对话框，让用户选择保存文件的位置。 下面是一个简单的，示例代码：const { dialog } = require('electron').remote; const fs = require('fs'); const options = { title: 'Save file', defaultPath: 'my_filename', buttonLabel: 'Save', filters: [{ name: 'txt', extensions: ['txt'] }, { name: 'All Files', extensions: ['*'] },], }; dialog.showSaveDialog(null, options).then(({ filePath }) => { if (filePath) { fs.writeFileSync(filePath, 'hello world', 'utf-8'); } });需要注意的点： 因为 fs 模块 和 Dialog 只能在，主进程中被调用。 但是我们的应用程序交互逻辑是在渲染进程，所以这段示例代码，只是演示了，如何去调用 Dialog 并手动选择文件存储位置。 并没有实际应用场景的参考意义。实际应用场景封装 调整对存取图片的封装想法跟之前的采集桌面思路基本一致（ 如果，没看过可以翻一下以前的文章 ）。 我们利用 Electron 的 ipcmain 模块在主进程中注册方法。 然后，在渲染进程去调用。 整理实现流程大概如下图所示。实例代码：// 主进程 -> main.js // .... ohter code // ... 在主进程注册我们封装后的 SaveFile 方法const { app, BrowserWindow, ipcMain, dialog} = require("electron"); const path = require("path"); const fs = require('fs'); /** * @param options: { title: String, defaultPath: String, buttonLabel: String, filters: area} * @param content: String * @returns Promise */ ipcMain.handle('saveFile', async (event, content, options) => { let path; try { const { filePath } = await dialog.showSaveDialog(null, options); path = filePath; } catch(e) { return Promise.reject({error: e, success: false}); } if(path) { try { fs.writeFileSync(path, content, 'utf-8'); return Promise.resolve({error: null, success: true}); } catch(e) { return Promise.reject({error: e, success: false}); } } else { return Promise.reject({error: e, success: false, canceled: true}); } }); // 其他代码 ....// Vue 文件、 // 在我们，的Vue 业务中。直接通过 ipc 调用import {reactive, ref, onMounted} from "vue"; const textContent = ref('hello world'); const { ipcRenderer } = require('electron'); const exportImg = async () => { try { // 注意-> 第一个参数，为 主进程注册进来的方法名称。 // 其他参数为 ， 主进程注册的函数参数。 await ipcRenderer.invoke('saveFile', 'hello', { title: '导出图片', buttonLabel: '导出', defaultPath: 'text.txt', filters: [{ name: 'All Files', extensions: ['*'] }] }) } catch(e) { console.error('出错了！', e) } } 导出 总结在Electron 中，向本地存储和读取文件（文本或者是图片）， 都离不开 node.js 的 fs 模块， 这个是文件系统处理的核心模块。 然后，我们搭配， path 模块。 dialog 模块，可以实现，用户主动选择存放位置。 或者 直接存到默认软件系统的 data 目录中。 当然，这些模块都只能在主进程中使用。 所以，我们不可避免的用到了， 之前的老朋友， ipc 进程间通讯方式。 本文，基于我在做一个手写照片 模拟器中遇到存储问题而来。 软件正在书写中。 后续会开源，并放出使用的视频。 请关注我的后续文章。 Do you say, Saved in my folder or saved on my folder. Thanks for the help saved in my folder - it is saved inside a metaphorical folder, not on top of one. Exactly. However there are some cases where it's tough to decide on whether to use "in" or "on." A store can be located "on" or "at" the corner (of two streets). Since we don't think if it as being in the middle of the intersection, we don't say "in the corner." However a chair or a person can be situated "in the corner" (of a room). Best, cew I always thought it was “the documents are saved in my folder”. I was asking the question because some of my colleagues always say "saved on my folder" and they are English native speakers I am not. Thanks you for the clarification. I always thought it was “the documents are saved in my folder”. I was asking the question because some of my colleagues always say "saved on my folder" and they are English native speakers I am not. Thanks you for the clarification. You are right, however note that the terminology varies depending on where you are saving it: "They are saved on my computer." "They are saved on my hard drive." "They are saved on this disk." "They are saved in a folder." Thank you for the clarification We use "in" for books and folders. Thus if I were reading a book and did not want to converse, I might say, "Not now, I'm in the middle of a good book." I've never heard anyone who spoke English as a native say that something was saved "on" his/her folder. We say, "That is saved on my desktop" (of a computer) but of course it is "saved in" the "My Documents" folder. Best, cew Hi! I always thought it was “the documents are saved in my folder”. I was asking the question because some of my colleagues always say "saved on my folder" and they are English native speakers I am not. Thanks you for the clarification. I just realized that we can say that something (an address, or another record usually) "is on file" (which means that we have it either in our file folders that we maintain or in our computers files or both--and thus that we don't need to obtain it again or elaborate more about it perhaps). Thus, "on file" means "in the files we maintain," and it's used to describe the general location of data records (such as addresses). However, what it really says is that we have these data records and can access them/look up the data in them rather than specifying the location of the data records. I hope this is not confusing. Best, cew Yes, I thing I got it when we say “on file” is when we have recorded something and when “is saved in a file“, it’s referred to location. Thanks for your help Luisiher Yes, that's right. You understand it. "On file" indicates that a record is availabe. "In the/a [particular] file" specifies the location something is saved in. The etymology of the English prepositions "in" and "on" is interesting--apparently, to some degree, they share a root: (According to the resource above and also according to my offline dictionary, "on" used to be used where we use "in" and "on" was in Old English an unstressed variant of in.) Best, cew Last edited: Jul 22, 2009 These prepositions drive crazy. Now another similar question (I am not sure if I should open a new thread as this is related) To improve on or to improve in I think is to improve in For example I need to improve in my piano skills Thanks again Hi. Yes this is related to the 'on' and 'in' question, but improve on /in is a different idiom than save on/in and has a different meaning. So I'd start a new thread (the moderators will probably suggest that you do so too). Best, cew 官网： ►►装依赖初始化package.jsonpnpm init安装依赖pnpm add -D electron安装报错解决方案： ►►打开npm的配置文件# cmd 运行打开配置文件 npm config edit2、在空白地方添加淘宝镜像，下面三个（ 缺什么补什么，但要是同一个公司单位的镜像 ）registry= electron_mirror= electron_builder_binaries_mirror= ►►动配置完重新安装即可pnpm add -D electron启动一个简单的项目1、修改package.json文件中的main与scripts配置段其中author和description为必填main 主进程脚本，这指向跟目录的main.jsstart 为启动命令：electron .{ "name": "Electron-补习所", "version": "1.0.0", "description": "electron-test", "main": "main.js", "scripts": { "start": "electron ." }, "keywords": [], "author": "wangfan", "license": "ISC", "devDependencies": { "electron": "^32.1.2" } }2、根目录新建main.jsconst { app, BrowserWindow } = require("electron"); // 监听app的ready事件 app.on("ready", () => { // 创建一个窗口 const win = new BrowserWindow({ width: 500, // 窗口宽度 height: 500, // 窗口高度 autoHideMenuBar: true, // 隐藏默认菜单 }); win.loadURL('); // 加载线上url页面 // BrowserWindow的更多配置可参考文档： }); 这样一个简单的项目就启动了。加载本地页面根目录新建pages文件夹，在里面写入index.html、index.css，正常的写一些内容main.js文件修改，通过loadFile方法加载本地页面const { app, BrowserWindow } = require("electron"); // 监听app的ready事件 app.on("ready", () => { const win = new BrowserWindow({ width: 500, height: 500, autoHideMenuBar: true, alwaysOnTop: true }); win.loadFile('./pages/index.html'); // 加载页面-指定路径 }); 启动项目：pnpm run start开发者工具，打开后会有一个内容安全策略警告解决方法是配置CSP，具体配置可上MDN上查看： ►►置完后重启项目，内容安全策略警告就会消失。完善窗口行为这里我将new BrowserWindow封装为一个函数createWindow，当应用ready(准备完毕)后，调用createWindow创建一个窗口。const { app, BrowserWindow } = require("electron"); // 创建一个窗口-封装 function createWindow() { // 创建一个窗口 const win = new BrowserWindow({ width: 800, // 窗口宽度 height: 500, // 窗口高度 autoHideMenuBar: true, // 隐藏默认菜单 }); win.loadFile("./pages/index.html"); // 加载页面 // BrowserWindow的更多配置可参考文档： } // 监听app的ready事件 app.on("ready", () => { createWindow(); // window所有窗口关闭时，并且不是苹果系统，退出应用-管理窗口的生命周期 app.on("window-all-closed", () => { if (process.platform !== "darwin") app.quit(); }); }); // 应用被激活时，窗口数量为0，自动创建一个窗口-管理窗口的生命周期 app.on("activate", () => { if (BrowserWindow.getAllWindows().length === 0) createWindow(); }); 窗口行为分两种，windows系统和mac系统，在windows系统中，所有窗口关闭后自动关闭应用，而mac系统则是自动缩小到任务栏不会关闭应用。所以这里需要做窗口行为管理，也就是官网所说的管理窗口的生命周期自动启动应用项目每次主进程(main.js)修改后，应用都需要手动重启，非常麻烦，我们可以安装nodemon来自动重启项目pnpm i nodemon -D在package.json中配置{ // 省略其它配置..... "scripts": { "start": "nodemon --exec electron ." // 通过nodemon --exec来启动electron . } } 这样每次修改主进程内容后应用会自动重启另外你也可以在文件中具体配置nodemon，来实现主进程、页面内容修改后自动重启功能。根目录新建：nodemon.json{ "ignore": ["node modules", "dist"], "restartable": "r", "watch": ["*.*"], "ext": "html,js,css" } 主进程与渲染进程在electron中，主进程只有一个，渲染进程可以有n个main.js就是主进程，主进程是node环境运行的在html中运行的js就是渲染进程，渲染进程是在web环境中运行的新建一个js文件，在index.html中引入它修改js文件// render.js let btn = document.getElementById("btn1"); btn.addEventListener("click", () => { alert("弹窗"); });主进程(node环境)与渲染进程(web环境)作用的环境不同，他们之间是隔离开的，web端不能使用nodeAPI而node也无法使用webAPI。web端由于是沙箱环境，如果你想在web端获取一些系统、环境、硬件的信息，可能就需要借助node能力与系统交互，那么electron就为我们提供了主进程与渲染进程之间的通信。主进程与渲染进程之间通过预加载脚本通信，预加载脚本在渲染端(web环境)运行，虽然预加载脚本在web端运行，但是它可以访问一部分的nodeAPI，预加载脚本就是主进程与渲染进程之间的桥梁。根目录新建预加载脚本：preload.js这样我们的目录结构就一目了然了，pages文件夹下管理渲染进程，preload.js管理预加载脚本，main.js管理主进程我们随便在preload.js中写点东西打印// preload.js console.log('预加载脚本')然后在主进程中引入预加载脚本，预加载脚本只能使用绝对路径，这里使用node模块path引入根目录下的preload.js// main.js const { app, BrowserWindow } = require("electron"); const path = require("path"); // 创建一个窗口-封装 function createWindow() { const win = new BrowserWindow({ // 省略其它配置项... webPreferences: { preload: path.resolve(__dirname,'./preload.js'), // 加载预加载脚本，绝对路径 } }); win.loadFile("./pages/index.html"); // 加载页面 } // 监听app的ready事件 app.on("ready", () => { createWindow(); }); // 省略其它配置项... 启动项目后会发现窗口打印了预加载脚本的输出语句应用的运行顺序是：主进程 -> 预加载脚本 -> 渲染进程现在我们来看预加载脚本如何与渲染进程通信的使用 contextBridge 来选择要从预加载脚本中暴露哪些 API，通过exposeInMainWorld向渲染器进程暴露一个全局的 window.abc变量。// preload.js const { contextBridge } = require("electron"); // 向渲染进程暴露全局myAPI变量 contextBridge.exposeInMainWorld("myAPI", { mytext: "这是暴露的变量,预加载进程可使用部分nodeAPI", version: process.version }); 然后在渲染进程中打印window你会发现暴露的myAPI直接挂载在window身上若是挂在在window身上，你可以直接获取myAPI变量btn.addEventListener("click", () => { // alert("弹窗"); // console.log(window); console.log(myAPI); });这样就做到了渲染进程访问预加载脚本的能力。进程通讯(IPC)-重要需求：点击按钮，在系统D盘创建一个hello.text文件，文件内容来自用户输入渲染进程向主进程通信（ 单项 ）先编写页面 Document 欢迎光临 查看弹窗

向D盘写入hello.txt // render.js let btn = document.getElementById("btn1"); let setBtn = document.getElementById("setBtn"); let input = document.getElementById("input"); btn.addEventListener("click", () => { console.log(myAPI); }); // 设置写入按钮的点击事件 setBtn.onclick = () => { // 调用预加载脚本中定义的函数saveFile并传入input的输入内容 myAPI.saveFile(input.value) }; 编写预加载脚本在预加载脚本中，通过ipcRenderer将消息发送到主进程创建的监听器// preload.js const { contextBridge, ipcRenderer } = require("electron"); contextBridge.exposeInMainWorld("myAPI", { mytext: "这是暴露的变量", saveFile: (data) => { // 通过ipcRenderer.send向主进程发送消息 ipcRenderer.send("file-save", data); }, }); 编写主进程在主进程中，通过ipcMain监听预加载脚本发送的消息，监听的时机在加载页面之前。1、创建窗口2、订阅预加载脚本3、加载页面// main.js const { app, BrowserWindow, ipcMain } = require("electron"); const path = require("path"); const fs = require("fs"); // 引入node的fs模块操纵文件 // 自定义的函数，用于写入文件 // 回调函数有两个参数：一个IpcMainEvent结构和传入的变量 // 但是这里event用不上，所以用_占位 function writeFile(_, data) { // 在用户D盘下写入hello.txt文件，文件内容为data fs.writeFileSync("D:/hello.txt", data); } // 创建一个窗口-封装 function createWindow() { // 创建一个窗口 const win = new BrowserWindow({ width: 800, // 窗口宽度 height: 500, // 窗口高度 autoHideMenuBar: true, // 隐藏默认菜单 webPreferences: { preload: path.resolve(__dirname, "./preload.js"), // 加载预加载脚本，绝对路径 }, }); // 订阅预加载脚本，在加载页面之前，调用自定义函数writeFile ipcMain.on("file-save", writeFile); win.loadFile("./pages/index.html"); // 加载页面 } // 监听app的ready事 件 app.on("ready", () => { createWindow(); // 省略其它... }); 根据上面的步骤，就可以在系统D盘创建一个hello.txt文件，内容为输入框的内容。渲染进程向主进程通信（ 双向 ）读取我们之前写入的hello.txt文件内容在页面中写好结构 Document 欢迎光临 查看弹窗

向D盘写入hello.txt

读取D盘中hello.txt的内容 修改render.js的内容，同样的，调用预加载脚本中的自定义函数// render.js let btn = document.getElementById("btn1"); let setBtn = document.getElementById("setBtn"); let input = document.getElementById("input"); let getBtn = document.getElementById("getBtn"); btn.addEventListener("click", () => { console.log(myAPI); }); setBtn.onclick = () => { console.log(input.value); myAPI.saveFile(input.value); }; // 读取D盘中hello.txt的内容 // 实际上ipcRenderer.invoke返回的是一个Promise，所以使用async-await获取 getBtn.onclick = async () => { let txt = await myAPI.readFile(); console.log(txt); }; 在预加载脚本中，通过invoke来向主进程暴露请求// preload.js const { contextBridge, ipcRenderer } = require("electron"); console.log("预加载脚本"); contextBridge.exposeInMainWorld("myAPI", { mytext: "这是暴露的变量", saveFile: (data) => { ipcRenderer.send("file-save", data); }, // 自定义读取函数 readFile: () => { // 通过ipcRenderer.invoke向主进程发送请求 // 并将请求回来的值返回出去，ipcRenderer.invoke返回的是一个Promise return ipcRenderer.invoke("file-read"); }, }); 在主进程中监听invoke请求并执行自定义函数操作，最后将结果返回通过ipcMain.handle监听invoke事件const { app, BrowserWindow, ipcMain } = require("electron"); const path = require("path"); const fs = require("fs"); function writeFile(_, data) { fs.writeFileSync("D:/hello.txt", data); } // 自定义函数，读取D盘下的hello.txt文件内容，并返回 function readFile() { // 利用node的fs模块操作文件，readFileSync读取文件，toString将buffer转为字符串返回 return fs.readFileSync("D:/hello.txt").toString(); } // 创建一个窗口-封装 function createWindow() { const win = new BrowserWindow({ width: 800, height: 500, autoHideMenuBar: true, webPreferences: { preload: path.resolve(__dirname, "./preload.js"), // 加载预加载脚本，绝对路径 }, }); // 订阅预加载脚本，在加载页面之前 // 监听预加载脚本send事件 ipcMain.on("file-save", writeFile); // 监听预加载脚本invoke事件 ipcMain.handle("file-read", readFile); win.loadFile("./pages/index.html"); // 加载页面 } // 监听app的ready事 件 app.on("ready", () => { createWindow(); }); 打包应用pnpm install electron-builder -D在package.json中配置{ "name": "original-electron", "version": "1.0.0", "description": "original electron", "main": "main.js", "scripts": { "start": "nodemon --exec electron .", "build": "electron-builder" // 打包命令 }, "build": { "appId": "myelectron-app", // 应用程序唯一标识符 "win": { "icon": "./logo.ico", // 应用图标 "target": [{ "target": "nsis", // 指定使用 NSIS 作为安装程序格式 "arch": ["x64"] // 生成64位安装包 }] }, "nsis": { "oneClick": false, // 设置为 false 使安装程序显示安装向导界面，而不是一键安装 "perMachine": true, // 允许每台机器安装一次，而不是每个用户都安装 "allowToChangeInstallationDirectory": true // 允许用户在安装过程中选择安装目录 } }, } 运行打包命令pnpm run build目录中会生成dist文件夹，内部就打包的产物参考文档：安装依赖失败： ►►网： ►►手架： Hello. I think that both 'save that' and 'except that' have the similar meaning. And I can't distinguish with Korean words. Then, can they be exchageable with each other? 1a. This house is very good except that it doesn't have the trees. 1b. This house is very good save that it doesn't have the trees. 2a. They look alike except that one is a few inches taller. 2b. They look alike save that one is a few inches taller. Do both (a) and (b) have the same meaning? Thank you always. Last edited by a moderator: Apr 3, 2013 To me, using "save that" sounds strange in both situations. That is not a common phrase. However, if I were to choose one situation in which it seems more correct, I would say 2b. Because you are referring to something numerical (the number of inches) "save that" seems appropriate. However, to say "save that" in reference to an absence of trees sounds awkward. "Save" in this usage is in my opinion a bit old-fashioned, thus unusual in contemporary speech. I wouldn't use it, myself.