12个例子带你入门Electron[8000字附源码]

从零学习创建第一个**Electron**应用

随着前端快速的发展，在任何开发领域都有前端的一席之地。那如何使用前端进行桌面端应用的开发呢？Electron是一个不错的选择。

Electron是什么呢?

Electron就是使用 JavaScript，HTML 和 CSS 构建跨平台的桌面应用程序.

Electron是由github开发的开源框架

它允许开发者使用web技术开发跨平台的桌面应用

Electron = Chromium + Node.js + Native API

- Chromium：提供了强大的ui能力，利用强大的web生态来开发界面。

  Node.js：让 Electron有了底层才做能力，比如读写能力，并且可以使用大量的开源包来完成项目的开发。

  Native API：让Electron有了跨平台和桌面端的原生能力，比如有同意的原生界面、窗口、托盘。

> “
> 今天我们就来看看如何从零开始学习electron，用前端技术构建的桌面应用程序
> ”

创建第一个**Electron**应用

新建文件夹(命名记得不要出现中文)，初始化项目

```
npm init
会生成package.json 文件
文件内容如下

{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
别急
npm init 以后会让我们填写一些配置信息，不知道如何填写一路回车即可
当然也可以执行 npm init -y
```

安装electron

> “
> npm install --save-dev electron
> ”

如果安装的时候遇到什么问题(大部分报错或太慢)可以看看npm是否切换了淘宝镜像

> “
> npm config set registry https://registry.npm.taobao.org
> ”

也可以配置一下electron镜像

> “
> npm config set ELECTRON_MIRROR http://npm.taobao.org/mirrors/electron/
> ”

我们来测试一下electron是否安装成功

安装成功我们能看到electron的窗口打开

> “
> 命令行内执行 npx electron
> ”

看看electron的版本号

> “
> npx electron -v
> ”

根目录下创建main.js文件(主进程) electron主进程

```
const { app, BrowserWindow } = require('electron');
let mainWindow = null;

app.on('ready', () => {
   mainWindow = new BrowserWindow({    // 创建和控制浏览器窗口
      width: 800,                      // 窗口宽度
      height: 600,                     // 窗口高度
      webPreferences: {                // 网页功能设置
         nodeIntegration: true,      // 是否在node工作器中启用工作集成默认false
         enableRemoteModule: true,   // 是否启用remote模块默认false
      }
   });
   mainWindow.loadFile('index.html');  // 加载页面
   mainWindow.on('close', () => {      // 监听窗口关闭
      mainWindow = null
   })

})
```

根目录下创建index.html （需要加载的页面）

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <button id="btn">
        点击弹窗
    </button>
    <script src="./render/index2.js"></script>
</body>

</html>
```

修改paceage.json文件中的main的值，main表示需要启动的脚本， 将scripts 的test 修改为自己需要的名字或者只修改test 的值

> “
> 修改为如下
> ”

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "main.js",
  "scripts": {
    "start": "electron .",
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
  }
}
```

仅接着

> “
> npm run start 好的我们的第一个electron运行起来了
> ”

electron的运行流程

读取package.json文件中的入口文件，这里就是我们的main.js

main.js中我们引入electron 创建了渲染进程

index.html就是应用页面的布局和样式

使用IPC在主进程执行任务并获取信息

可能上面的前三点都能明白，但是第四点不知道是什么，没事我们现在只要知道他的运行流程即可内部实现先不做深究。

现在我们只需要知道main.js是主进程，index.html是渲染进程即可。

试试node.js的读取

我们先将目录结构规划一下

```
—package-lock.json
├—package.json
├—src
|  ├—main.js
|  ├—render
| |  ├—index.html
| |  ├—text.txt
| |  ├—yellow.html
| |  ├—js
| |  | ├—index.js
```

简单的划分后我们来看看是如何读取文件

> 我们在render目录下创建一个test.txt文件，里面写上被读取到啦。
>
> 接着在render中创建一个js文件夹在js文件夹内创建index.js文件。
>
> 在index.html中添加一个按钮，添加一个div标签(用于显示内容)，并且引入index.js。
>
> 接着在index.js中获取按钮元素，读取test.txt文件内容

代码如下:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <button id="btn">点击获取文件信息</button>
    <div id="filesData"></div>
    <script src="./js/index.js"></script>
</body>
</html>
```

index.js

```
// read
const fs = require('fs');
const path = require('path')

const btnRead = document.querySelector('#btnRead');
const filesDaata = document.querySelector('#filesData');
const btnWrite = document.querySelector('#btnWrite');


window.onload = ()=>{
    btnRead.onclick = () => {
        fs.readFile(path.join(__dirname, 'text.txt'),(err,data)=>{
            console.log(data);
            filesDaata.innerHTML = data
        })
    }
    btnWrite.onclick = () => {
        fs.writeFile(path.join(__dirname,'text.txt'),'添加1111', 'utf8',err=>{
            console.log(err)
        })
    }
}
```

main.js

```
const { app, BrowserWindow } = require("electron");
let mainWindow = null;

app.on("ready", () => {
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
      enableRemoteModule: true,
    },
  });
  mainWindow.loadFile(require('path').join(__dirname, './render/index.html'));
  mainWindow.on("close", () => {
    mainWindow = null;
  });
});
```

上面就是利用了node.js的读写取能力。

remote模块的使用

说Remote之前我们要明确一点，当我们知道了Electron是分主进程、渲染进程后，还需要知道Electron的API方法和模块也是分主进程、渲染进程。

回到我们Remote的使用

Remote是渲染进程（web页面）和主进程通信（IPC）提供一种简单的方法。

在index.html中写上一个按钮，通过点击事件创建一个新的窗口

创建一个yellow.html用作新建窗口的渲染进程

> “
> 代码如下
> ”

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <button id="btn">
        打开新窗口
    </button>
    <script src="./js/index.js"></script>
</body>
</html>
```

yellow.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    我是新建的窗口
</body>
</html>
```

index.js

```
// 创建新窗口
// remote
const BrowserWindow = require('electron').remote.BrowserWindow;
const btn = document.querySelector('#btn');
window.onload = ()=>{

    btn.onclick = ()=>{
        newWindow = new BrowserWindow({
            width: 300,
            height:300,
        })
        newWindow.loadFile(require("path").join(__dirname, 'yellow.html'));
        newWindow.on('close',()=>{
            newWindow = null;
        })
    }
}
```

main.js

```
const { app, BrowserWindow } = require("electron");
let mainWindow = null;

app.on("ready", () => {
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
      enableRemoteModule: true,
    },
  });
  mainWindow.loadFile(require('path').join(__dirname, './render/index.html'));
  mainWindow.on("close", () => {
    mainWindow = null;
  });
});
```

好滴，新建窗口完成啦，别急，我们看看主进程中的一段代码在webPreferences 里写了一句enableRemoteModule: true 。

> “
> 这是Electron版本更新后修改的，默认不启用remote模块需要手动打开,别忘记了
> ”

创建菜单以及基本使用

新建main文件夹，在文件夹新建menu.js

在主线程中银日menu.jsm

> “
> 注意 Menu属于主线程下的模块，所以只能在主线程下使用。
> ”

代码如下:

menu.js

```
const {Menu} = require('electron');

const  temlpate = [
    {
        label: '第一项',
        submenu:[
            {label: '1.1'},
            {label: '1.2'}
        ]
    },
    {
        label: '第二项',
        submenu:[
            {label:'2.1'},
            {label:'2.2'}
        ]
    }
]

const menu = Menu.buildFromTemplate(temlpate);
Menu.setApplicationMenu(menu);
```

main.js

```
const { app, BrowserWindow } = require("electron");
let mainWindow = null;
require('./main/menu')

app.on("ready", () => {
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
      // enableRemoteModule: true,
    },
  });
  mainWindow.loadFile(require('path').join(__dirname, './render/index.html'));
  mainWindow.on("close", () => {
    mainWindow = null;
  });
});
```

既然是菜单那么当然是可以点击的，看看菜单中如何点击

menu.js

```
const { Menu, BrowserWindow } = require("electron");

const temlpate = [
  {
    label: "第一项",
    submenu: [
      {
        label: "1.1",
        click: () => {
          let win = new BrowserWindow({
            width: 300,
            height: 300,
          });
          win.loadFile(require('path').join(__dirname,'../render/yellow.html'));
          win.on('close',()=>{
              win = null;
          })
        },
      },
      { label: "1.2" },
    ],
  },
  {
    label: "第二项",
    submenu: [{ label: "2.1" }, { label: "2.2" }],
  },
];

const menu = Menu.buildFromTemplate(temlpate);
Menu.setApplicationMenu(menu);
```

我们还能通过快捷键实现

```
const { Menu, BrowserWindow } = require("electron");

const temlpate = [
  {
    label: "第一项",
    submenu: [
      {
        label: "1.1",
        accelerator:`crtl+n`,
        click: () => {
          let win = new BrowserWindow({
            width: 300,
            height: 300,
          });
          win.loadFile(require('path').join(__dirname,'../render/yellow.html'));
          win.on('close',()=>{
              win = null;
          })
        },
      },
      { label: "1.2" },
    ],
  },
  {
    label: "第二项",
    submenu: [{ label: "2.1" }, { label: "2.2" }],
  },
];

const menu = Menu.buildFromTemplate(temlpate);
Menu.setApplicationMenu(menu);
```

右键菜单

右键是在渲染进程进行点击的，因此写在渲染进程中，且要用到remote模块

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <button id="btn">
        打开新窗口
    </button>
    <script src="./js/index3.js"></script>
</body>
</html>
```

index3.js

```
// 复制粘贴
const { remote } = require('electron');

const rightTemplate = [
    {label:'粘贴'},
    {label:'复制'},
]
const menu = remote.Menu.buildFromTemplate(rightTemplate);

window.addEventListener('contextmenu',(e)=>{
    // 阻止当前窗口默认事件
    e.preventDefault();
    //把菜单模板添加到右键菜单
    menu.popup({window:remote.getCurrentWindow()})
})
```

通过链接打开浏览器

在渲染进程中默认加入一个a标签，进行跳转默认是直接在窗口中打开，而不是在浏览器中打开的。代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <a href="https://www.baidu.com">123</a>
</body>
</html>
```

可以看到这个跳转是在页面内的跳转，那么我们来看看如何通过打开浏览器。

使用shell在浏览器打开

创建index4.js

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- shell通过链接打开浏览器 -->
    <a id="aHref" href="https://www.baidu.com">123</a>

    <script src="./js/index4.js"></script>
</body>
</html>
```

index4.js

```
// 通过链接打开浏览器
const { shell } = require('electron');

const aHref = document.getElementById('aHref');

aHref.onclick = (e)=>{
    // 阻止默认事件 因为默认是在应用中打开;
    e.preventDefault();
    // 获取链接
    const href = aHref.getAttribute('href');
    // 浏览器中打开
    shell.openExternal(href)
}
```

嵌入网页和打开子窗口

既然是嵌入网页，那当然是主进程中进行嵌入

BrowserView键入网页到应用

打开主进程main.js

```
const { app, BrowserWindow, BrowserView } = require("electron");
let mainWindow = null;
require('./main/menu')

app.on("ready", () => {
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
      enableRemoteModule: true,
    },
  });
  mainWindow.webContents.openDevTools()
  mainWindow.loadFile(require('path').join(__dirname, './render/index.html'));

  const viewWindow = new BrowserView(); // 定义实例
  mainWindow.setBrowserView(viewWindow); // 主窗口设置viewWindw可用
  viewWindow.setBounds({x:0,y:150,width: 600,height:600});
  viewWindow.webContents.loadURL('https://www.baidu.com')

  mainWindow.on("close", () => {
    mainWindow = null;
  });
});
```

上面通过BowserView嵌入网页， 下面来看看window.open打开子窗口

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- 打开子窗口-->
    <button id="btn">打开子窗口 </button>

    <script src="./js/index5.js"></script>
</body>
</html>
```

index5.js

```
// 创建子窗口
const btn = document.getElementById('btn');
btn.onclick = ()=>{
    window.open('child.html')
}
```

window.open子窗口和父窗口之间的通信

window.opener.postMessage(message,targetOrigin),是将消息发送给指定来源的父窗口，如果未指定来源则发送给*，即所有窗口。

创建了child.html index6.js

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- 子窗口父窗口通信 -->
    <button id="btn">打开子窗口 </button>
    <div id="message"></div>

    <script src="./js/index5.js"></script>
</body>
</html>
```

child.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>我是子窗口</div>
    <button id="btn">传递数据</button>
    <script src="./js/index6.js"></script>
</body>
</html>
```

index5.js

```
// 创建子窗口
const btn = document.getElementById('btn');
btn.onclick = ()=>{
    window.open('child.html')
}

// 子窗口与父窗口通信
const message = document.getElementById('message')
window.addEventListener('message',msg => {
    console.log(msg)
    message.innerHTML = msg.data
})
```

index6.js

```
// 子窗口发来信息
const btn = document.getElementById('btn');
btn.onclick = ()=>{
    window.opener.postMessage('子窗口发来信息')
}
```

选择文件对话框

我看看打开文件对话框的API是什么dialog.showOpenDialog(), 这个方法可以接收两个参数，一个是基本的属性设置，一个是回调函数，如果异步可以使用then基本设置有

title：对话框名字

defaultPath: 默认打开路径

buttonLabel: 确认按钮的自定义标签、若为空则使用默认标签

filters: 文件选择器，定义后可以对文件类型进行筛选

properties: 打开文件的属性，如：能否多选，选择文件的类型等

下面来看看代码

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body
    <!-- 打开文件 -->
    <button id="btn">打开文件</button>


    <script src="./js/index7.js"></script>
</body>
</html>
```

index7.js

```
// 选择文件对话框
const {dialog} = require('electron').remote;
const btn = document.getElementById('btn')

btn.onclick = () =>{
    dialog.showOpenDialog({
        title:'请选择你需要的文件',   // 对话框标题
        defaultPath: '默认路径',     // 默认打开路径
        filters:[                   // 文件选择过滤
            {
                name:'jpg',
                extensions:['jpg']
            }
        ],
        buttonLabel:'是否确认',
        properties: ['openFile', 'multiSelections']   // 对话框使用的功能，允许选择文件、允许多选
    }).then(res => {
        console.log(res)
    }).catch(err=>{
        console.log(err)
    })
}
```

保存对话框的操作

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- 保存对话框 -->
    <button id="btn">保存对话框</button>
    <script src="./js/index8.js"></script>
</body>
</html>
```

index8.js

```
// 选择文件
const btn = document.getElementById('btn');
const fs = require('fs');
const {dialog} = require('electron').remote;
btn.onclick = ()=>{
    dialog.showSaveDialog({
        title: '选择文件'
    }).then(res => {
        console.log(res);
        fs.writeFileSync(res.filePath, '你好啊');
    }).catch(err =>{
        console.log(err)
    })
}
```

当打开窗口输入文件名后会在对应的文件夹下创建相对于的文件，我们通过node.js的fs对相应的文件进行操作，写入了你好啊

> “
> 注意 保存文件是dialog.showSaveDialog
> ”

消息对话框（制作一个确认对话框）

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- 消息对话框 -->
    <button id="btn">打开消息对话框</button>
    <script src="./js/index9.js"></script>
</body>
</html>
```

index9.html

```
// 消息对话框
const btn = document.getElementById("btn");
const { dialog } = require('electron').remote

btn.onclick = () => {
    dialog.showMessageBox({
    type: "warning",
    title: "警告",
    message: "这是一个警告对话框",
    buttons: ["知道了", "无所谓"],
  }).then(res => {
      console.log(res)
  });
};
```

断网提醒

通过window.addEventListener进行监听online offline判断网络是否正常

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- 断网提醒 -->
    <div>断网提醒</div>

    <script src="./js/index10.js"></script>
</body>
</html>
```

index10.js

```
// 断网提醒
window.addEventListener('online',()=>{
    alert('网络正常，放心使用')
})

window.addEventListener('offline',()=>{
    alert('网络异常，请检查网络是否连接')
})
```

> “
> 注意：运行electon后不会有反应，先进行断网，在对网络进行连接即可看到效果
> ”

底部通知消息提醒

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- 底部通知消息提醒 -->
    <button id="btn">通知消息</button>

    <script src="./js/index11.js"></script>
</body>
</html>
```

index11.js

```
// 底部消息提醒
const btn = document.getElementById('btn')

const option = {
    title: '新消息提醒',
    body: '你有新的消息提醒'
}
btn.onclick = ()=>{
    new window.Notification(option.title,option)
}
```

注册全局快捷键

全局快捷加的注册是使用globalShortcut,globalShortcut是主线程的模块，因此写在main.js中

main.js 完整代码

```
const { app, BrowserWindow, BrowserView, globalShortcut } = require("electron");
let mainWindow = null;
require('./main/menu')

app.on("ready", () => {
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
      enableRemoteModule: true,
    },
  });
  globalShortcut.register('ctrl+y',()=>{
    mainWindow.loadURL('https://www.baidu.com')
  })
  globalShortcut.register('ctrl+z',()=>{
    mainWindow.loadFile(require('path').join(__dirname, './render/index.html'))
  })
  mainWindow.webContents.openDevTools()
  mainWindow.loadFile(require('path').join(__dirname, './render/index.html'));

  // const viewWindow = new BrowserView(); // 定义实例
  // mainWindow.setBrowserView(viewWindow); // 主窗口设置viewWindw可用
  // viewWindow.setBounds({x:0,y:150,width: 600,height:600});
  // viewWindow.webContents.loadURL('https://www.baidu.com')

  mainWindow.on("close", () => {
    mainWindow = null;
  });
});
```

通过globalShortcut.isRegistered()判断全局快捷键是否注册成，可以打开多个软件进行测试

```
const { app, BrowserWindow, BrowserView, globalShortcut } = require("electron");
let mainWindow = null;
require('./main/menu')

app.on("ready", () => {
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
      enableRemoteModule: true,
    },
  });
  globalShortcut.register('ctrl+y',()=>{
    let isRegister = globalShortcut.isRegistered(`ctrl+y`)?'true':'false'
    console.log(isRegister)
    mainWindow.loadURL('https://www.baidu.com')
  })
  globalShortcut.register('ctrl+z',()=>{
    mainWindow.loadFile(require('path').join(__dirname, './render/index.html'))
  })
  mainWindow.webContents.openDevTools()
  mainWindow.loadFile(require('path').join(__dirname, './render/index.html'));

  // const viewWindow = new BrowserView(); // 定义实例
  // mainWindow.setBrowserView(viewWindow); // 主窗口设置viewWindw可用
  // viewWindow.setBounds({x:0,y:150,width: 600,height:600});
  // viewWindow.webContents.loadURL('https://www.baidu.com')

  mainWindow.on("close", () => {
    mainWindow = null;
  });
});
```

注销快捷键

因为是全局快捷键所以页面关闭记得注销快捷键

下面是注册快捷键、 判断快捷键是否注册成功、注销快捷键

main.js

```
const { app, BrowserWindow, BrowserView, globalShortcut } = require("electron");
let mainWindow = null;
require('./main/menu')

app.on("ready", () => {
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
      enableRemoteModule: true,
    },
  });
  globalShortcut.register('ctrl+y',()=>{
    let isRegister = globalShortcut.isRegistered(`ctrl+y`)?'true':'false'
    console.log(isRegister)
    mainWindow.loadURL('https://www.baidu.com')
  })
  globalShortcut.register('ctrl+z',()=>{
    mainWindow.loadFile(require('path').join(__dirname, './render/index.html'))
  })
  mainWindow.webContents.openDevTools()
  mainWindow.loadFile(require('path').join(__dirname, './render/index.html'));

  // const viewWindow = new BrowserView(); // 定义实例
  // mainWindow.setBrowserView(viewWindow); // 主窗口设置viewWindw可用
  // viewWindow.setBounds({x:0,y:150,width: 600,height:600});
  // viewWindow.webContents.loadURL('https://www.baidu.com')

  mainWindow.on("close", () => {
    mainWindow = null;
  });

});

// 注销快捷键

app.on('will-quie',()=>{
  globalShortcut.unregister('ctrl+y');
  globalShortcut.unregister('ctrl+z');
  globalShortcut.unregisterAll()

})
```

剪切板事件使用

通过 clipboard模块实现

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- 剪切板  复制-->被复制的信息
    <div>
        复制信息: <span id="message">被复制的信息</span><button id="btn">点击复制</button>
    </div>

    <script src="./js/index12.js"></script>
</body>
</html>
```

index12.js

```
// 剪切板
const btn = document.getElementById('btn');
const message = document.getElementById('message');
const { clipboard } = require('electron');
btn.onclick = ()=>{
    clipboard.writeText(message.innerHTML)
    alert('复制成功')
}
```

打包桌面应用

这里只进行windows的打包测试

npm install electron-builder --save-dev\

安装完成以后在package.json文件中的scrit内添加一个属性

> “
> "build": "electron-builder --win --x64"
> ”

执行 npm run build 即可进行打包

代码可查看 https://github.com/barntet/electron

最后

欢迎加入我们【特皮技术团队】，后台回复：加群即可联系我们

觉得写得不错可以点个在看/赞,关注支持我们一下,让更多人看到好文章,我们会持续输出优质原创文章

-