Electron + Vite + TS + Vue3打开新窗口实战

前言

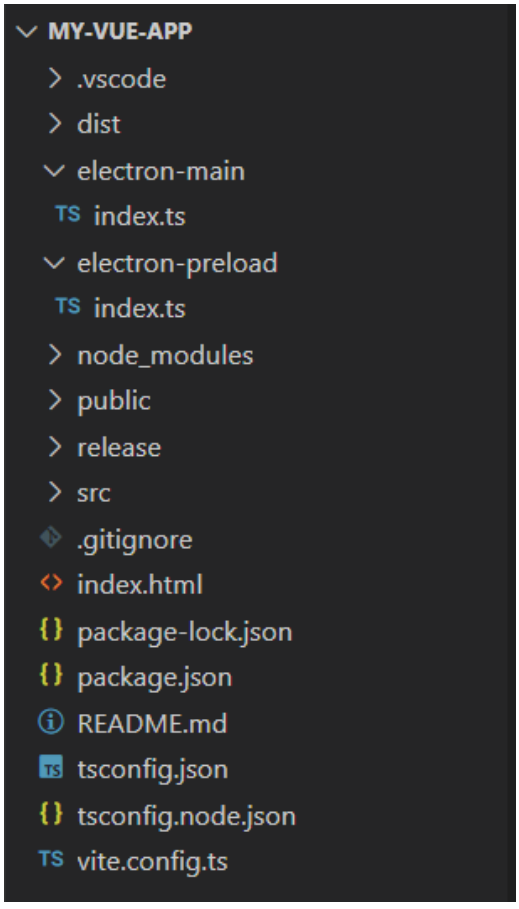
我们在使用 Electron 编写桌面应用时，打开新窗口可以说是一个非常常见的场景了。很多刚接触 Electron 的小伙伴面对这样一个问题可能都会显得比较棘手，比如打开新窗口如何知道渲染哪一个页面？打开的新窗口如何与其它窗口产生联系，比如父子窗口？...等等一系列问题。

今天我们就将 Electron 打开新窗口的常见做法分享给大家，而且是基于最新的 TS 封装。

1.基础项目搭建

还没有简单基础项目的小伙伴赶紧搭建一个 Electron 项目，具体可以参考： Electron + Vue3 + TS + Vite 桌面应用项目搭建教程 。

我们先来看一下基础的项目目录结构吧，如下图：



本篇文章我们重点关注 electron-main 这个目录，该目录就是 electron 项目的主进程目录，我们将在这里面封装打开新窗口的一些方法。

2.实现目标

有了目标我们才能更好更快的去理清思路，我们可以回想一下平时使用的桌面程序它们打开新窗口都有哪些特点，比如腾讯视频、腾讯 QQ 等等。

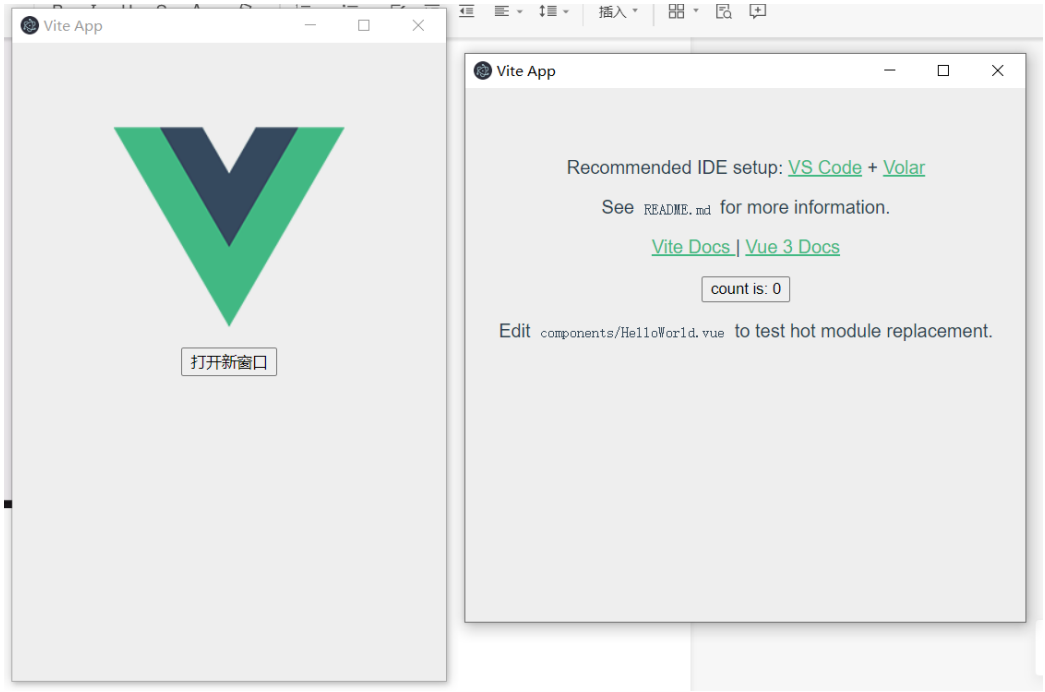
针对于我们当前的 Electron+Vue3+TS 项目，主要实现以下需求。

需求如下：

- 在渲染进程中，直接调用某个方法即可打开新窗口。
- 默认打开的新窗口是一个子窗口。
- 打开新窗口方法可以接收参数。
- 可以传入路由地址，新窗口渲染此路由地址页面。
- 可以传入窗口样式，如宽高、背景色、是否显示默认菜单栏等等。
- 可以单独关闭当前新打开的窗口。

上面几点需求大致就是我们此次打开新窗口需要实现的功能，当然，你还可以添加更多自定义需求。

先来简单看下效果：



上图中左侧是我们的主窗口，点击打开新窗口按钮时，便会打开右侧的子窗口，接下来我们就需要去写代码来实现了。

3.具体实现

3.1 改造 electron-main/index.ts 文件

既然我们要通过调用方法来打开新窗口，那么有必要将打开新窗口这类操作直接封装成方法，我们改造下主进程的入口文件。

代码如下：

```
// electron-main/index.ts
import { app, BrowserWindow } from "electron";
import { Window } from "./window"; // 具体方法放在此处
const isDevelopment: boolean = process.env.NODE_ENV !== "production";

// 创建主窗口
async function createWindow() {
  let window = new Window();
  window.listen(); // 设置监听事件，比如主进程与渲染进程之间的通信事件
  window.createWindows({ isMainWin: true }); // 创建窗口，默认为主窗口
  window.createTray(); // 创建系统托盘
}

// 关闭所有窗口
app.on("window-all-closed", () => {
  if (process.platform !== "darwin") {
    app.quit();
  }
});

app.on("activate", () => {
  if (BrowserWindow.getAllWindows().length === 0) createWindow();
});

// 准备完成，初始化窗口等操作
app.on("ready", async () => {
  createWindow();
});

// 根据环境处理不同操作
if (isDevelopment) {
  if (process.platform === "win32") {
    process.on("message", (data) => {
      if (data === "graceful-exit") {
        app.quit();
      }
    });
  } else {
    process.on("SIGTERM", () => {
      app.quit();
    });
  }
}
```

上段代码主要是一个入口文件，我们把创建创建窗口、创建监听事件、创建系统托盘等操作都风窗到了 window.ts 文件中，这里重点理解下面三个方法：

- window.listen()
- window.createWindows({ isMainWin: true })
- window.createTray();

3.2 新建 electron-main/window.ts 文件

前面的 index.ts 只是主进程的入口文件，接下来我们需要编写真正创建窗口、创建托盘、监听事件等方法的文件了：window.ts。

这个文件我们主要编写以下几个函数：

- getWindow(id: number)：获取当前窗口
- createWindows(options: object)：创建新的窗口
- createTray()：创建系统托盘
- listen()：开始事件监听

代码如下：

```
// electron-main/window.ts
import { app, BrowserWindow, ipcMain, Menu, Tray } from "electron";
import path from "path";
interface IWindowsCfg {
  id: number | null;
  title: string;
  width: number | null;
  height: number | null;
  minWidth: number | null;
  minHeight: number | null;
  route: string;
  resizable: boolean;
  maximize: boolean;
  backgroundColor: string;
  data: object | null;
  isMultiWindow: boolean;
  isMainWin: boolean;
  parentId: number | null;
  modal: boolean;
}
interface IWindowOpt {
  width: number;
  height: number;
  backgroundColor: string;
  autoHideMenuBar: boolean;
  resizable: boolean;
  minimizable: boolean;
  maximizable: boolean;
```

```
  frame: boolean;
  show: boolean;
  parent?: BrowserWindow;
  minWidth: number;
  minHeight: number;
  modal: boolean;
  webPreferences: {
    contextIsolation: boolean; //上下文隔离
    nodeIntegration: boolean; //启用 Node 集成（是否完整的支持 node）
    webSecurity: boolean;
    preload: string;
  };
}

// 新建窗口时可以传入的一些options配置项
export const windowsCfg: IWindowsCfg = {
  id: null, //唯一 id
  title: "", //窗口标题
  width: null, //宽度
  height: null, //高度
  minWidth: null, //最小宽度
  minHeight: null, //最小高度
  route: "", // 页面路由 URL '/manage?id=123'
  resizable: true, //是否支持调整窗口大小
  maximize: false, //是否最大化
  backgroundColor: "#eee", //窗口背景色
  data: null, //数据
  isMultiWindow: false, //是否支持多开窗口（如果为 false，当窗体存在，再次创建不会
  isMainWin: false, //是否主窗口(当为 true 时会替代当前主窗口)
  parentId: null, //父窗口 id  创建父子窗口 -- 子窗口永远显示在父窗口顶部 【父窗口
  modal: false, //模态窗口 -- 模态窗口是禁用父窗口的子窗口，创建模态窗口必须设置 pa
};
// 窗口组
interface IGroup {
  [props: string]: {
    route: string;
    isMultiWindow: boolean;
  };
}

/**
 * 窗口配置
 */
export class Window {
  main: BrowserWindow | null | undefined;
  group: IGroup;
  tray: Tray | null;
  constructor() {
    this.main = null; //当前页
    this.group = {}; //窗口组
    this.tray = null; //托盘
```

```
  }

  // 窗口配置
  winOpts(wh: Array<number> = []): IWindowOpt {
    return {
      width: wh[0],
      height: wh[1],
      backgroundColor: "#f7f8fc",
      autoHideMenuBar: true,
      resizable: true,
      minimizable: true,
      maximizable: true,
      frame: true,
      show: false,
      minWidth: 0,
      minHeight: 0,
      modal: true,
      webPreferences: {
        contextIsolation: false, //上下文隔离
        nodeIntegration: true, //启用 Node 集成 ( 是否完整的支持 node )
        webSecurity: false,
        preload: path.join(__dirname, "../electron-preload/index.js"),
      },
    };
  }

  // 获取窗口
  getWindow(id: number): any {
    return BrowserWindow.fromId(id);
  }

  // 创建窗口
  createWindows(options: object) {
    console.log("------------开始创建窗口...");
    let args = Object.assign({}, windowsCfg, options);
    // 判断窗口是否存在
    for (let i in this.group) {
      if (
        this.getWindow(Number(i)) &&
        this.group[i].route === args.route &&
        !this.group[i].isMultiWindow
      ) {
        console.log("窗口已经存在了");
        this.getWindow(Number(i)).focus();
        return;
      }
    }
    // 创建 electron 窗口的配置参数
    let opt = this.winOpts([args.width || 390, args.height || 590]);
    // 判断是否有父窗口
    if (args.parentId) {
```

```
      console.log("parentId：" + args.parentId);
      opt.parent = this.getWindow(args.parentId) as BrowserWindow; // 获取主
    } else if (this.main) {
      console.log('当前为主窗口');
    } // 还可以继续做其它判断

    // 根据传入配置项，修改窗口的相关参数
    opt.modal = args.modal;
    opt.resizable = args.resizable; // 窗口是否可缩放
    if (args.backgroundColor) opt.backgroundColor = args.backgroundColor; //
    if (args.minWidth) opt.minWidth = args.minWidth;
    if (args.minHeight) opt.minHeight = args.minHeight;

    let win = new BrowserWindow(opt);
    console.log("窗口 id：" + win.id);
    this.group[win.id] = {
      route: args.route,
      isMultiWindow: args.isMultiWindow,
    };
    // 是否最大化
    if (args.maximize && args.resizable) {
      win.maximize();
    }
    // 是否主窗口
    if (args.isMainWin) {
      if (this.main) {
        console.log("主窗口存在");
        delete this.group[this.main.id];
        this.main.close();
      }
      this.main = win;
    }
    args.id = win.id;
    win.on("close", () => win.setOpacity(0));

    // 打开网址（加载页面）
    let winURL;
    if (app.isPackaged) {
      winURL = args.route
        ? `app://./index.html${args.route}`
        : `app://./index.html`;
    } else {
      winURL = args.route
        ? `http://${process.env["VITE_DEV_SERVER_HOST"]}:${process.env["VITE
        : `http://${process.env["VITE_DEV_SERVER_HOST"]}:${process.env["VITE
    }
    console.log("新窗口地址:", winURL);
    win.loadURL(winURL);

    win.once("ready-to-show", () => {
      win.show();
```

```
    });
  }

  // 创建托盘
  createTray() {
    console.log("创建托盘");
    const contextMenu = Menu.buildFromTemplate([
      {
        label: "注销",
        click: () => {
          console.log("注销");
          // 主进程发送消息，通知渲染进程注销当前登录用户 --todo
        },
      },
      {
        type: "separator", // 分割线
      },
      // 菜单项
      {
        label: "退出",
        role: "quit", // 使用内置的菜单行为，就不需要再指定 click 事件
      },
    ]);
    this.tray = new Tray(path.join(__dirname, "../favicon.ico")); // 图标
    // 点击托盘显示窗口
    this.tray.on("click", () => {
      for (let i in this.group) {
        if (this.group[i]) this.getWindow(Number(i)).show();
      }
    });
    // 处理右键
    this.tray.on("right-click", () => {
      this.tray?.popUpContextMenu(contextMenu);
    });
    this.tray.setToolTip("小猪课堂");
  }

  // 开启监听
  listen() {
    // 固定
    ipcMain.on('pinUp', (event: Event, winId) => {
      event.preventDefault();
      if (winId && (this.main as BrowserWindow).id == winId) {
        let win: BrowserWindow = this.getWindow(Number((this.main as Browser
        if (win.isAlwaysOnTop()) {
          win.setAlwaysOnTop(false); // 取消置顶
        } else {
          win.setAlwaysOnTop(true); // 置顶
        }
      }
    })
```

```
233
234     // 隐藏
235     ipcMain.on("window-hide", (event, winId) => {
236       if (winId) {
237         this.getWindow(Number(winId)).hide();
238       } else {
239         for (let i in this.group) {
240           if (this.group[i]) this.getWindow(Number(i)).hide();
241         }
242       }
243     });
244
245     // 显示
246     ipcMain.on("window-show", (event, winId) => {
247       if (winId) {
248         this.getWindow(Number(winId)).show();
249       } else {
250         for (let i in this.group) {
251           if (this.group[i]) this.getWindow(Number(i)).show();
252         }
253       }
254     });
255
256     // 最小化
257     ipcMain.on("mini", (event: Event, winId) => {
258       console.log("最小化窗口 id", winId);
259       if (winId) {
260         this.getWindow(Number(winId)).minimize();
261       } else {
262         for (let i in this.group) {
263           if (this.group[i]) {
264             this.getWindow(Number(i)).minimize();
265           }
266         }
267       }
268     });
269
270     // 最大化
271     ipcMain.on("window-max", (event, winId) => {
272       if (winId) {
273         this.getWindow(Number(winId)).maximize();
274       } else {
275         for (let i in this.group)
276           if (this.group[i]) this.getWindow(Number(i)).maximize();
277       }
278     });
279
280     // 创建窗口
281     ipcMain.on("window-new", (event: Event, args) => this.createWindows(args
282   }
283 }
```

代码思路：

- 定义两个接口 interface，用来规定窗口默认参数格式。
- 调用创建窗口方法时会传入一些配置项，方法内部需要合并这些配置项。
- 根据传入的路由地址，动态配置需要渲染的页面。
- 每一个新窗口都会产生一个窗口 id。
- 为了让每个窗口产生关联，需要给每个窗口配置参数中带上 parentId 字段。
- 关闭窗口或者其它 electron 操作事件时，都根据窗口 id 来获取到对应的窗口。

3.3 渲染进程调用

在渲染进程中我们使用@vueuse/electron 模块方便的进行主进程与渲染进程之间的通信，比如我们打开一个新窗口，可以像如下写法。

代码如下：

```
<template>
  <img alt="Vue logo" src="../assets/logo.png" />
  <div>
    <button @click="openNewWin">打开新窗口</button>
  </div>
</template>
<script setup lang="ts">
import { useIpcRenderer } from "@vueuse/electron";
const ipcRenderer = useIpcRenderer();
const openNewWin = () => {
  ipcRenderer.send("window-new", {
    route: "/helloworld",
    width: 500,
    height: 500,
  });
};
</script>
```

这样就简单实现了一个打开新窗口。

总结

针对于本篇文章对于 Electron 打开新窗口的封装，可能有些小伙伴觉得稍显复杂，但是长痛不如短痛，一次封装，多次获益！

我们需要搞懂以下几个问题，对我们的 Electron 打开新窗口就会有很多帮助的：

- 每个新窗口都会有 id
- 通过 parentId 来个每个窗口建立关联
- 把所有的窗口都使用一个窗口组对象保存下来