Electron + Vue3 + TS + Vite项目搭建教程！

前言

Electron主要是用来搭建桌面应用程序的，虽然有许多人不喜欢它，但是也不能撼动目前它的王者地位！使用Electron可以让我们前端开发者快速搭建出桌面应用程序，我们所熟知的VS code就是使用Electron开发的。

由于Electron更新迭代非常快，加上Vue3已经发布很长一段时间了，所以我们很有必要将我们的项目升级一下，所以这里就利用electron+Vue3搭建一个万精油的项目框架！

1.初始化项目

我们需要先借助Vite初始化一个Vue3+TS 的项目，后面我们在逐步添加electron，在任何一个文件夹下：

执行命令：

```
npm create vite@latest my-vue-app -- --template vue-ts
```

安装依赖：

```
npm install
```

运行项目：

```
npm run dev
```

Hello Vue 3 + TypeScript + Vite

Recommended IDE setup: VS Code + Volar

See README.md for more information.

Vite Docs | Vue 3 Docs

count is: 0

Edit components/HelloWorld.vue to test hot module replacement.

这样一个最简单的Vue3 + TS + Vite的前端项目就初始化好了。

2.安装Electron相关包

初始化一个基本项目后，我们需要在项目中安装一些关于electron的包。

安装electron：

```
npm install electron -D
```

如果你安装electron不成功，建议使用cnpm安装。

安装electron-builder：

```
npm install electron -D
```

主要利用electron-builder来进行打包。

安装electron-devtools-installer：

```
npm install electron-devtools-installer -D
```

该包主要是为了方便我们开发和调试electron，可以去官网详细了解：electron-devtools-installer。

安装vite-plugin-electron：

```
npm install vite-plugin-electron -D
```

该包集成了Vite和Electron，比如使用它之后可以让我们方便的在渲染进程中使用Node API或者Electron API，详细使用用法可以去官网学习：vite-plugin-electron。

安装rimraf ：

```
npm install rimraf -D
```

该包主要是辅助作用，让我们快速删除某些文件和文件夹。

3.初始化Electron

我们都知道Electron项目分为了主进程和渲染进程，主进程其实就是我们的Electron，渲染进程就相当于我们的Vue项目。

3.1 新建主进程

为了方便修改代码和查看，我们在项目根目录新建主进程文件夹electron-main，然后在其目录下新建index.ts文件，编写主进程代码。

代码如下：

```
// electron-main/index.ts
import { app, BrowserWindow } from "electron";
import path from "path";

const createWindow = () => {
  const win = new BrowserWindow({
    webPreferences: {
      contextIsolation: false, // 是否开启隔离上下文
      nodeIntegration: true, // 渲染进程使用Node API
      preload: path.join(__dirname, "../electron-preload/index.js"), // 需要
    },
  });

  // 如果打包了，渲染index.html
  if (app.isPackaged) {
    win.loadFile(path.join(__dirname, "../index.html"));
  } else {
    let url = "http://localhost:3000"; // 本地启动的vue项目路径
    win.loadURL(url);
  }
};

app.whenReady().then(() => {
  createWindow(); // 创建窗口
  app.on("activate", () => {
    if (BrowserWindow.getAllWindows().length === 0) createWindow();
  });
});

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

上段代码只是一个最简单的Electron主进程的代码，大家也可以直接去官网看一下即可。这里有两个点需要大家**注意**：

- 渲染进程路径引用的是js而不是ts，因为我们的electron是不认识ts文件的，有些小伙伴可能不理解引用的js文件合适产生的，这都不用着急，我们安装的插件会帮我们解决的。
- app.isPackaged主要是用来判断应用是否已经打包了，打包了我们只需要引用相对路径的html文件即可。

3.2 新建预加载文件

electron中有一个预加载的概念，也就是我们常说的preload，在该文件里面可以在其它脚本文件执行之前运行，它可以调用一些Node API。

在项目根目录新建electron-preload文件夹，然后在其目录下新建index.ts文件，编写代码。

代码如下：

```
// electron-preload/index.ts
import os from "os";
console.log("platform", os.platform());
```

上段代码只是一个示例，我们只是简单的打印了一下系统信息罢了，重点是需要大家理解预加载这个概念，以及预加载可以做什么。

4.修改tsconfig.json

前面我们已经建好了渲染进程和预加载文件，但是我们是放在项目根目录里面的。自动生成的Vue3+ts项目只初始化了src目录下的文件监听，所以我们需要修改一下tsconfig.json配置文件。

在include属性里新增关于electron文件监听的配置项。

代码如下：

```
"include": [
  "src/**/*.ts",
  "src/**/*.d.ts",
  "src/**/*.tsx",
  "src/**/*.vue",
  "electron-main/**/*.ts",
  "electron-preload/**/*.ts"
],
```

5.修改vite.config.ts

虽然我们建好了electron的主进程文件和预加载文件，但是如果我们不做任何处理，这两个文件就和普通的脚本文件没有任何区别了。所以我们需要修改vite.config.ts配置文件，以此将

electron和vite项目结合起来。

代码如下：

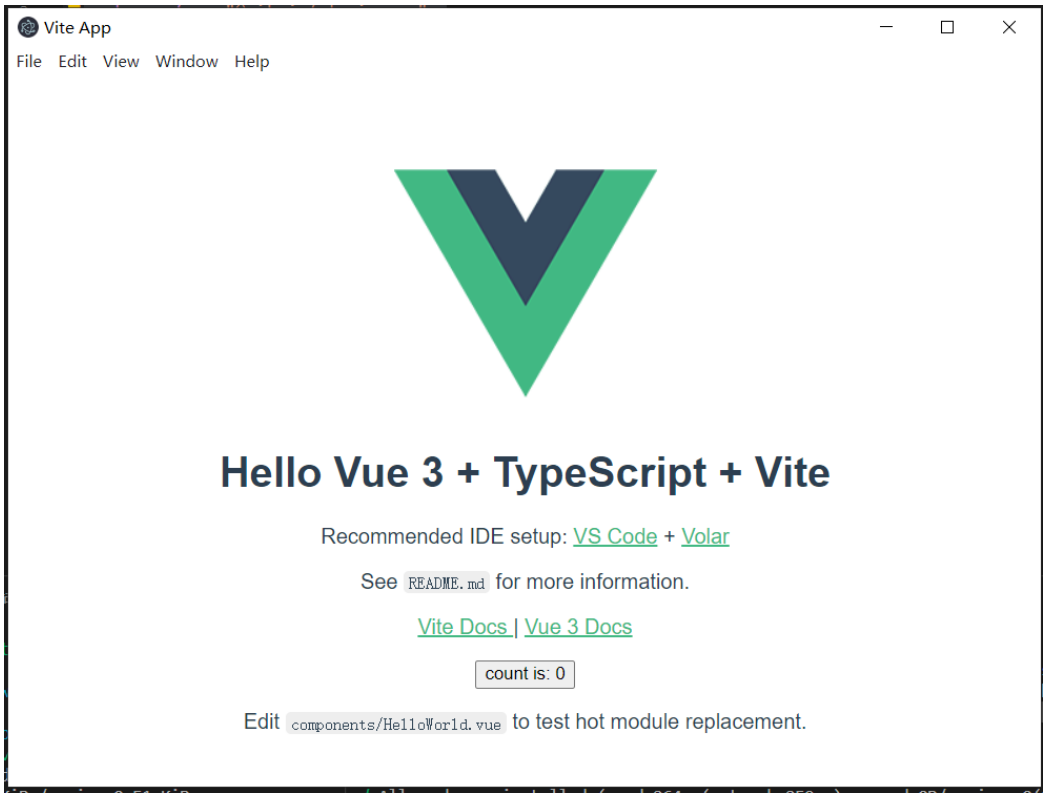
```
import { defineConfig } from "vite";
import vue from "@vitejs/plugin-vue";

import * as path from "path";
import electron from "vite-plugin-electron";
import electronRenderer from "vite-plugin-electron/renderer";
import polyfillExports from "vite-plugin-electron/polyfill-exports";

export default defineConfig({
  plugins: [
    vue(),
    electron({
      main: {
        entry: "electron-main/index.ts", // 主进程文件
      },
      preload: {
        input: path.join(__dirname, "./electron-preload/index.ts"), // 预加载
      },
    }),
    electronRenderer(),
    polyfillExports(),
  ],
  build: {
    emptyOutDir: false, // 默认情况下，若 outDir 在 root 目录下，则 Vite 会在构建
  },
});
```

在上面的配置中，我们使用到了vite-plugin-electron插件，它将我们的electron和vite很好的结合了起来。

其实到这里大家启动项目后就会发现，electron出面其实已经就出现了，界面如下：



6.修改package.json

package.json作为一个项目里面很重要的一个文件，我们自然是不能忽略的。在这里我们需要配置我们项目的入口文件以及打包的相关配置。

代码如下：

```
{
  "name": "my-vue-app",
  "private": true,
  "version": "0.0.0",
  "main": "dist/electron-main/index.js",
  "scripts": {
    "dev": "vite",
    "build": "rimraf dist && vite build && electron-builder",
    "preview": "vite preview"
  },
  "dependencies": {
    "vue": "^3.2.25"
  },
  "devDependencies": {
    "@vitejs/plugin-vue": "^2.3.3",
    "electron": "^19.0.0",
    "electron-builder": "^23.0.3",
    "electron-devtools-installer": "^3.2.0",
    "rimraf": "^3.0.2",
    "typescript": "^4.5.4",
    "vite": "^2.9.9",
    "vite-plugin-electron": "^0.4.5",
    "vue-tsc": "^0.34.7"
```

```
  }
}
```

我们配置了main入口文件，由于electron还未支持ts，所以我们需要引用打包后的index.js文件。

除此之外，我们修改了build命令，利用electron-builder进行electron项目的打包。

7.配置electron-builder打包脚本

想要顺利打包项目，我们还需要写一些关于electron-builder的打包脚本代码，如果想要详细了解脚本的各项配置项的作用的，可以去官网学习一些：electron-builder。

修改package.json文件，**代码如下：**

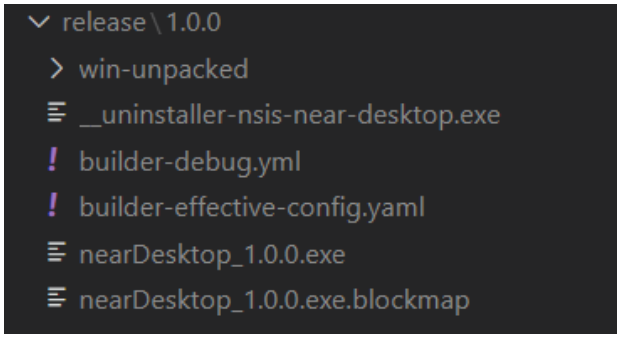
```
{
  ......
  "build": {
    "appId": "com.smallpig.desktop",
    "productName": "smallpig",
    "asar": true,
    "copyright": "Copyright © 2022 smallpig",
    "directories": {
      "output": "release/${version}"
    },
    "files": [
      "dist"
    ],
    "mac": {
      "artifactName": "${productName}_${version}.${ext}",
      "target": [
        "dmg"
      ]
    },
    "win": {
      "target": [
        {
          "target": "nsis",
          "arch": [
            "x64"
          ]
        }
      ],
      "artifactName": "${productName}_${version}.${ext}"
    },
    "nsis": {
      "oneClick": false,
      "perMachine": false,
      "allowToChangeInstallationDirectory": true,
      "deleteAppDataOnUninstall": false
    },
```

```
    "publish": [
      {
        "provider": "generic",
        "url": "http://127.0.0.1:8080"
      }
    ],
    "releaseInfo": {
      "releaseNotes": "版本更新的具体内容"
    }
  }
}
```

执行打包：

```
npm run build
```

打包完成的相关文件都会放到release目录下面，目录如下：



我们经常会打包失败，很大部分原因就是因为资源下载不下来，可能是由于国内的网络环境，建议大家多试几次，比如出现下面这种问题：

```
  • packaging       platform=win32 arch=x64 electron=19.0.0 appOutDir=release\0.0.0\win-unpacked
  × Get "https://github.com/electron/electron/releases/download/v19.0.0/electron-v19.0.0-win32-x64.zip": dial tcp 20.205.243.166:443: connectex: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.
github.com/develar/app-builder/pkg/download.(*Downloader).follow.func1
        /Volumes/data/Documents/app-builder/pkg/download/downloader.go:206
github.com/develar/app-builder/pkg/download.(*Downloader).follow
        /Volumes/data/Documents/app-builder/pkg/download/downloader.go:234
github.com/develar/app-builder/pkg/download.(*Downloader).DownloadNoRetry
        /Volumes/data/Documents/app-builder/pkg/download/downloader.go:128
github.com/develar/app-builder/pkg/download.(*Downloader).Download
        /Volumes/data/Documents/app-builder/pkg/download/downloader.go:112
github.com/develar/app-builder/pkg/electron.(*ElectronDownloader).doDownload
```

打包完成后，我们只需要安装生成出来的exe文件即可，非常简单。

8.主进程与渲染进程通信

为了方便我们主进程与渲染进程的通信，我们借助vueuse插件库中的一个插件@vueuse/electron来简化我们的工作，具体使用方式可以查看官网：@vueuse/electron。

安装@vueuse/electron：

```
npm install @vueuse/electron -D
```

渲染进程App.vue，使用示例代码如下：

```vue
<script setup lang="ts">
import { useIpcRenderer } from "@vueuse/electron";
const ipcRenderer = useIpcRenderer();
ipcRenderer.send("window-new", "im render"); // 向主进程通信
</script>
```

electron-main/index.ts中主进程监听事件：

```ts
import { app, BrowserWindow, ipcMain } from "electron";
// 监听渲染进程方法
ipcMain.on("window-new", (e: Event, data: string) => {
  console.log(data);
});
```

运行项目后，我们查看命令行控制台打印结果：

```
13:56:03 [vite] hmr update /src/App.vue (x2)
im render
```

这个时候我们渲染进程和主进程之间就可以正常通信了。

9.打包常见错误

错误一：

```
  - Adjust chunk size limit for this warning via build.chunkSizeWarningLimit.
    • electron-builder  version=23.0.3 os=10.0.19043
    • loaded configuration  file=package.json ("build" field)
    • description is missed in the package.json  appPackageFile=D:\electron\near-desktop\package.json
    • author is missed in the package.json  appPackageFile=D:\electron\near-desktop\package.json
    • writing effective config  file=release\1.0.0\builder-effective-config.yaml
    • packaging       platform=win32 arch=x64 electron=18.2.3 appOutDir=release\1.0.0\win-unpacked
    • default Electron icon is used  reason=application icon is not set
    • building        target=nsis file=release\1.0.0\nearDesktop_1.0.0.exe archs=x64 oneClick=false perMachine=false
    × C:\Users\lanyuan\AppData\Local\electron-builder\Cache\nsis\nsis-3.0.4.1\Bin\makensis.exe process failed ERR_ELECTRON_BUILDER_CANNOT_EXECUTE
Exit code:
1
Output:
```

可以尝试以下方法解决错误：

- 删除C:\Users\lanyuan\AppData\Local\electron-builder\cache缓存文件
- 重新安装electron

错误二：

```
    • packaging       platform=win32 arch=x64 electron=19.0.0 appOutDir=release\0.0.0\win-unpac
ked
    × Get "https://github.com/electron/electron/releases/download/v19.0.0/electron-v19.0.0-win3
2-x64.zip": dial tcp 20.205.243.166:443: connectex: A connection attempt failed because the c
onnected party did not properly respond after a period of time, or established connection fai
led because connected host has failed to respond.
github.com/develar/app-builder/pkg/download.(*Downloader).follow.func1
        /Volumes/data/Documents/app-builder/pkg/download/downloader.go:206
github.com/develar/app-builder/pkg/download.(*Downloader).follow
        /Volumes/data/Documents/app-builder/pkg/download/downloader.go:234
github.com/develar/app-builder/pkg/download.(*Downloader).DownloadNoRetry
        /Volumes/data/Documents/app-builder/pkg/download/downloader.go:128
github.com/develar/app-builder/pkg/download.(*Downloader).Download
        /Volumes/data/Documents/app-builder/pkg/download/downloader.go:112
github.com/develar/app-builder/pkg/electron.(*ElectronDownloader).doDownload
```

解决办法：

- 尝试解决网络问题
- 尝试将该文件下载下来，本地配置

总结

到这里我们electron + Vue3 + TS + Vite的项目架子就搭建完成了，这只是一个最简单的项目框架，还需要有非常多的地方需要我们去填充，比如打开新窗口，桌面提醒等等，这也会在后续的文章中提及。