开源浅思

https://github.com/lencx/ChatGPT

分享者：lencx

ChatGPT 桌面应用开发的心路历程。
将项目从默默无闻，做到 35K+ Stars 顶级开源。
成功具有偶然，不可复制性。
但很多因素凑在一起会让一些偶然成为必然，
希望通过我的分享可以给大家带来一些思考。

Let's GO →

内容大纲

1. 背景
2. Tauri 简介
    1. 项目结构
    2. 通信方式
        1. tauri::command & invoke
        2. Event: emit & listen
3. ChatGPT 桌面应用核心实现
    1. 代码示例：应用入口
    2. 代码示例：加载 URL 并注入脚本
    3. 代码示例：注入脚本中调用 Tauri API
4. 思考
    1. 行动大于空想
    2. 社区力量
    3. 产品思维
5. 如何学习?
6. 结束语

背景

一切技术的本质，都是为了解决问题，实战就是最好的学习。

`Web -> Rust -> WebAssembly -> Tauri -> ChatGPT`

- **结缘 Rust**：我非科班出身，毕业后从培训机构接触 Web，开始入行前端开发。因计算机基础薄弱，故希望学习一门系统语言，来提升一些自己对底层的认知。
- **开发 rsw 插件**：rsw-rs 算是我用 Rust 开发的第一个比较正式的工具。它是一个 CLI，旨在解决使用 Rust 开发 WebAssembly 时的热更新问题，提升开发体验。
- **Tauri 探索**：在开发 rsw-rs 之后，感觉 WebAssembly 应用于实际生产对我来说似乎有点遥远。所以我决定开始学习 Tauri（基于 Rust 的跨平台桌面应用开发框架，可以使用 Web 技术（React、Vue 等）来开发桌面应用），进一步在实战中学习 Rust。分享即是学习，我写下了《Tauri 教程》和《Rust 在前端》系列文章。

Tauri 简介

这里本不想多说，但考虑到这是一个技术分享，就简单介绍两个特点。具体细节及实现推荐看文档（重要的事情说三遍：看文档！看文档！看文档！），不过只看 Tauri 文档，有点不太够用，有能力的还是推荐去读一些 Tauri 源码和一些开源项目，会发现很多小技巧。

- **跨平台**：Tauri 支持 Windows、macOS 和 Linux，UI 部分使用 Web 技术（React、Vue 等）来开发。2.0 版本已支持移动端（Android 和 iOS）。
- **安装包体积小，内存占用小**：Hello World 应用一般在 3M 左右。但调用系统内置浏览器，兼容性会差一些。
- 系统菜单、系统托盘、权限管理、自动更新等等。

Electron vs Tauri

- **Electron = Node.js + Chromium**
- **Tauri = Rust + Tao + Wry**
  - Tao: 跨平台应用程序窗口创建库，支持所有主要平台，如 Windows、macOS、Linux、iOS 和 Android。
  - Wry: 跨平台 WebView 渲染库，支持所有主要桌面平台，如 Windows、macOS 和 Linux。

项目结构

项目结构简单，除标准的前端项目结构，外加 `src-tauri` 目录

```
[Tauri]
├── [src] # 前端代码
│   ├── main.js # 入口
│   └── ...
├── [src-tauri] # Rust 代码
│   ├── [src]
│   │   ├── main.rs # 入口
│   │   └── ...
│   ├── build.rs
│   ├── Cargo.toml # Rust 配置文件，类似于 package.json
│   ├── tauri.conf.json # 应用配置文件，包含权限，更新，窗口配置等等
│   └── ...
├── vite.config.ts # Vite 配置文件
├── package.json # 描述 Node.js 项目依赖和元数据的文件
└── ...
```

通信方式

- **tauri::command & invoke**: 前端通过 invoke API 调用 Rust 的 command 方法。command 可以接受参数并返回值。
- **Event: emit & listen**: 双向通信（Rust <-> Webview），emit 发送事件，listen 监听事件。

注意：Tauri 中所有的 API 都是异步的，在前端均以 Promise 的形式返回。

tauri::command & invoke

`src-tauri/src/main.rs`

```rust
#[tauri::command]
fn hello(name: String) -> String {
  format!("Hello, {}!", name)
}

fn main() {
  tauri::Builder::default()
    // 注册命令
    .invoke_handler(tauri::generate_handler![hello])
    .run(tauri::generate_context!())
    .expect("failed to run app");
}
```

`src/main.js`

```javascript
import { invoke } from '@tauri-apps/api/tauri';

// 调用 Rust 的 hello 方法
// 输出：Hello, ChatGPT!
await invoke('hello', { name: 'ChatGPT' });
```

Event: emit & listen

js 之间通信

```
// src/main.js
import { emit, listen } from '@tauri-apps/api/event';

// 监听事件
const unlisten = await listen('click', (event) => {
  // output: Hello, ChatGPT!
  console.log(event.theMessage);
})

// 发送事件
emit('click', {
  theMessage: 'Hello, ChatGPT!',
})
```

JS 和 Rust 之间通信

JS -> Rust

```
app.get_window("main").unwrap().emit("rust2js", Some("Hello from Rust!"));
```

```
import { listen } from '@tauri-apps/api/event';
listen('rust2js', (event) => {
  console.log(event.theMessage); // output: Hello from Rust!
})
```

Rust -> JS

```
import { emit } from '@tauri-apps/api/event';
emit('js2rust', {
  theMessage: 'Tauri is awesome!',
})
```

```
app.get_window("main").unwrap().listen("js2rust", |msg| {
  // output: Event { id: EventHandler(xxxxxxxx), data: Some("{\"theMessage\":\"Tauri is awesome!\"}") }
  println!("js2rust: {:?}", msg);
});
```

ChatGPT 桌面应用核心实现

项目的灵感来自于机器人指令，如果经常玩 TG 或者 Discord 的朋友应该都比较熟悉（通过输入斜杠指令来调用机器人的功能。比如：`/help`、`/start`、`/ping` 等等）。而 ChatGPT 经常需要重复性输入 Prompt，所以我想到了通过指令的方式来调用 ChatGPT 的功能（据我所知，这个功能应该是我最早实现，后来就出现了许多类似浏览器插件）。

桌面应用是基于 Tauri 的套壳实现，简单来说就是直接在 Webview 中加载网站 URL。通过注入脚本的方式来实现对网站功能的拓展。主要有以下几点：

- 如何加载 URL 到窗口?
- 加载的网址中如何注入脚本?
- 注入脚本中如何调用 Tauri API?

代码示例：应用入口

```rust
// main.rs
#[tauri::command]
pub fn hello(name: String) {
  println!("Hello, {}!", name);
}
fn main() {
  tauri::Builder::default()
    .invoke_handler(tauri::generate_handler![hello]) // 注册命令
    .plugin() // 注册插件，如果命令过多可以考虑写成插件，方便管理
    .setup() // 初始化
    .system_tray() // 系统托盘
    .menu() // 系统菜单
    .on_menu_event() // 菜单事件
    .on_system_tray_event() // 托盘事件
    .on_window_event() // 窗口事件
    .run(tauri::generate_context!())
    .expect("error while running NoFWL application");
}
```

```javascript
// main.js
import { invoke } from '@tauri-apps/api';
await invoke('hello', { name: 'lencx' });
```

代码示例：加载 URL 并注入脚本

```rust
// main.rs
tauri::Builder::default()
  .setup(|app| {
    tauri::WindowBuilder::new(
      app,
      "main", // 窗口 ID
      tauri::WindowUrl::App("https://chat.openai.com".into()) // 加载 URL
    )
      .initialization_script(include_str!("./scripts/core.js")) // 注入脚本
      .title("ChatGPT") // 标题
      .inner_size(800.0, 600.0) // 窗口大小
      .resizable(true) // 是否可调整窗口大小
      .build()
      .unwrap();
  })
  .run(tauri::generate_context!())
  .expect("error while running ChatGPT application");
```

代码示例：注入脚本中调用 Tauri API

这一部分比较复杂，因为 Tuari 的架构设计本身就是为安全而生的，所以如果应用程序选择通过加载远程 URL 的方式来创建窗口时，Tauri 不会为该窗口注入 Tauri API。我是从源码中获得技巧，通过 Tauri 暴露的 `__TAURI_POST_MESSAGE__` 底层 API 自己来模拟出上层 `invoke` API。这部分代码有点多，就不展示了，具体可以参考：

`https://github.com/lencx/ChatGPT/blob/main/src-tauri/src/scripts/core.js`

Tauri 套壳 ChatGPT，代码实现到这里，整个应用程序的核心逻辑就算跑通了。即：

1. `tauri::WindowBuilder::new` 加载 URL https://chat.openai.com
2. `initialization_script` 注入脚本
3. `invoke` 调用 `tauri::command`
4. `tauri::command` 实现操作系统文件读写

思考

程序员最不缺的就是编码力和创造力，但能够成为独立开发者的人却少之又少。我认为，主要有以下原因：

- 眼高手低，或不屑于去做。那不就是个套壳吗，有什么可搞的？
- 缺少开发独立产品思维，虽然在公司做过的项目挺多，但自己独立完成整个产品闭环时却有点茫然（功能实现，页面交互，界面排版，项目架构，项目推广等等）。
- 对信息的敏锐性，和技术的学习力下降。上班已经那么卷了，下班或周末就会选择躺平，不愿意走出舒适区。
- 缺乏分享意识。虽然平时技术群，各大社区没少吹牛，但能够正真沉淀下来的东西少之又少。
- 以及其他一些因素。

行动大于空想

当时我在 Tauri 群里聊开发桌面应用的想法时，有些群友表示不看好，认为已经有人开发过了，你完全可以给别人做贡献（提 PR）。而不是重复造轮子，同期类似项目还有两个：

- sonnylazuardi/chat-ai-desktop：基于 Tauri 开发
- vincelwt/chatgpt-mac：基于 Electron 开发

做一件事情时，身边必然会出现一些不和谐的声音，但他们的观点并不可以左右你的行动。就个人而言，我不喜欢被束缚，因为提 PR 就意味着你必须按照别人的想法去做，事情会变得不可控（通过/拒绝）。我创建项目的初衷并不是为了服务于人（也没想着会火），主要是为验证自己的一些想法。

社区力量

项目早期想要获得关注是很困难的一件事情。在早期我做了两件事情，现在看来，正是这两件事让它迅速段时间内获得了巨大关注（向两个开源项目提了 issues）：

- liady/ChatGPT-pdf：PDF，图片导出功能
- f/awesome-chatgpt-prompts：斜杠指令的 prompt 数据源

首先对两个库作者的工作表示感谢，并告知他们我已经将他们所做的工作集成到了 ChatGPT 应用中。Awesome ChatGPT Prompts 作者认为我这个想法很棒，并且愿意在 README 中添加我的项目链接。相互成就，才能走的更远。

项目早期还是比较辛苦的，不但要开发功能，还要回复 issues，Fix Bug。随着项目的发展，也有一些小伙伴参与进来，贡献 PR。这里我想感谢每一个参与开源的人，正是因为 TA 们，开源的生态才能不断发展壮大。

产品思维

公众号「浮之静」可查看原文《流量密码：ChatGPT 开源的一些思考》

- **产品闭环**：它可以很小，功能可以很简陋，但是必须要形成最小闭环，保证其可用性（产品核心功能可以正常使用）。
- **速度要快**：开发速度，更新速度，问题相应速度都要快，因为它可以帮助你抢占第一波用户（种子用户积累很重要，可以形成口碑，帮助产品二次传播）。
- **用户体验**：这是需要花心思的，虽然你是一名开发者，但是你更是一名使用者。所以没有产品，你就是产品；没有设计，你就是设计（你就是用户，甚至你要比用户更懂用户，学会取舍）。
- **产品计划**：你对产品未来方向的规划，计划加入什么牛逼的功能，需要在文档里写清楚。它就相当于是在给用户画饼，可以打动一些想要长期追随它的用户（注意：画饼不代表天马行空的想法，而是根据实际情况，可实现但因时间原因暂时无法实现的计划）。
- **差异化**：因为当你发现机会的时候，别人可能早已经在里面开始收割了，所以产品功能的差异化，将是你的突破口（人无我有，人有我有优）。
- **稳定性**：产品的初期的架构很重要，它可能会伴随其一生。重构有时候并不现实，因为它需要牵扯到很多的历史包袱，数据兼容，人力成本等等（可扩展性很重要）。

如何学习？

现在的我们正在面临各种碎片化的冲击。海量信息，短视频让人的思维愈发碎片化（许多人表示很难静下心来读一篇大几千字，上万字的文章，更别谈思考或输出了）。“卷”这个字也是近些年最火的一个字，没有信息让人焦虑，信息爆炸会让人变得更加焦虑。

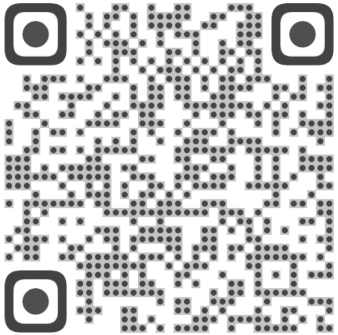
我也是在开发桌面应用之后，才开始接触 AI 这个领域。写的文章多了（大约输出了 35+ 篇 AI 系列文章），也莫名成了别人眼中的大佬（自己有多菜只有自己清楚）。

`未知知识学习 = 扩展阅读 + 信息源 + 已有知识 + 经验推导`

- 扩展阅读：善用~~搜素引擎~~ ChatGPT，检索文章中的未知术语或名词（不过我更倾向于在 ChatGPT 给出结论后，自己再用搜索引擎复核一下）
- 信息源：尽可能去靠近信息源，关注领域大牛。信息具有时效性，二手信息会造成信息差，交智商税，走弯路是必然的。
- 分享输出：分享是最高效的学习方式。动手写或给别人讲，都会让你发现很多之前注意不到的细节（看往往是浮于表面，细节和坑都隐藏在更深处）。

结束语

身为一名程序员我很自豪，虽然足不出户，指尖却有着可以改变~~世界（可能有点大子）~~自己的力量。即使不能实现，将其作为努力的目标也不错。



微信扫码或搜索：**浮之静**

关注公众号：了解更多 AI 资讯，技术思考等