



链滴

思源源码解析（二）：启动顺序

作者：[zuo ez02](#)

原文链接：<https://ld246.com/article/1676187755406>

来源网站：[链滴](#)

许可协议：[署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

<p>系列文章: </p>

<p>思源源码解析（一）：整体架构</p>

<h2 id="0x00-Electron-启动">0x00 Electron 启动</h2>

<p>思源桌面都基于 Electron，当思源启动的时候，首先启动的就是 electron 的代码，也即是 elect on 的主线程代码 <code>app/electron/main.js</code>。</p>

<p> <code>main.js</code> 主要包含了以下几个内容: </p>

<p>错误显示 <code>showErrorWindow</code> </p>

<p>日志输出 <code>writeLog</code> </p>

<p>引导 <code>boot</code> </p>

<p>初始化内核 <code>initKernel</code> </p>

<p>窗口控制</p>

<p>electron app 初始化及后续事件监听，包括: </p>

ipcMain 事件监听，即主线程与渲染线程的交互

app 事件监听

<p>作为 electron 程序，思源程序的生命周期实际上是包含在 electron app 的生命周期之中的。除函数声明和监听之外，我们可以看到的整体流程为: </p>

思源初始化文件存储

Electron APP 初始化

Electron 菜单初始化

开启第一个 workspace 的窗口

判断是否未初次使用用户，是则等待渲染进程中发送 siyuan-first-init 事件，否则直接使用端口和 workspace 配置，初始化内核

内核初始化成功完成后，boot 引导创建主窗口。

<p>其余的多 workspace 管理、进程退出管理、系统休眠、关机等事件处理，在此先不做研究。</p>

<p>思源启动的重点，主要在于 initKernel 和 boot 两个部分</p>

<h3 id="initKernel">initKernel</h3>

<p>思源启动时，我们首先看到的是一个半透明的加载窗口（Linux 为非透明），这个窗口显示着系内核目前的加载状态，当失败时也会显示对应的失败信息。而背后的工作，实际就是思源的 electron 的 app 在获取可用端口和确定内核文件存在之后，使用 nodejs 的 <code>child_process.spawn</code> 方法，创建了内孩子进程。并持续到维护管理内核的运行状态，例如接收系统启动信息，内核常退出管理等。当内核启动失败时，会通过 showErrorWindow 的方式显示错误，关闭程序。通过询问内核接口的方式，获取启动进度和信息，接收到内核启动 100% 时，完成 initKernel。</p>

<h3 id="boot">boot</h3>

<p>当 boot 方法执行时，就是我们实际使用中初始化窗口关闭，显示思源主窗体的过程。</p>

主窗口 `currentWindow` 的内容信息是通过访问内核的界面接口获取的

```
language-js highlight-chroma"><span class="highlight-line"><span class="highlight-cl"><span class="highlight-nx">currentWindow</span><span class="highlight-".</span><span class="highlight-nx">loadURL</span><span class="highlight-p">(</span><span class="highlight-nx">getServer</span><span class="highlight-p">()</span><span class="highlight-o">+</span><span class="highlight-s1">'stage/build/app/index.html?v='<span><span class="highlight-o">+</span><span class="highlight-k">new</span><span class="highlight-nb">Date</span><span class="highlight-p">().</span><span class="highlight-t-nx">getTime</span><span class="highlight-p">())</span></span></span></code></pre>
```

主窗体加载完成后，加入到 workspaces 中，剩余的启动相关内容都是在主窗体的渲染进程中进的。

0x01-主窗体运行

根据目前的加载方式和架构可以发现，思源与传统的“前后端分离架构”非常类似。而现在，我就从主窗体的界面加载进行探索。

主模板入口

通过查看 `webpack.desktop.js` 可知，前端界面的 html 是通过 `app/rc/assets/template/index.tpl` 渲染来获得的。当然，这个 tpl 中并没有包含后端模板渲染内容。可以直接当做 html 来了解。

出去主入口 js 会后续添加到 head 中之外，主窗口的结构包括了

- `loading` 为加载界面，也就是一上来我们看到的包含思源 logo 的黑屏加载面
- `toolbar`、`dockTop`、`dockerLeft`、`layouts`、`dockRight`、`dockerBottom`、`status` 均为页面骨架组件
- `commonMenu`、`message` 为其他相关的工具内容

主脚本入口

主脚本入口为编译之后的 `src/index.ts`，我们从源文件开始看起。

`index.ts` 中包含了 `App` 这个唯一的类，并对齐进行了单创建操作。在其构造函数中，显示了 App 的创建过程。

- 同步添加脚本 `lute`，`lute` 是由思源开发者创建的 Markdown 引擎。

Lute 是一款结构化的 Markdown 引擎，完整实现最新的 GFM / Commo Mark 规范，对中文语境支持更好。
- 异步加载 `protyle`，`protyle-html` 是 `HTMLElement` 的继子类，可以看作是一类思源中特有的 DOM 元素，与 `lute` 相关，后续我们再做深度挖掘。
- `addBaseURL()` 根据后端内容初始化部分配置。
- `window.siyuan`，是在 `window` 对象上创建的 `siyuan` 公开 API，包含了大的原始信息，可以作为插件或者挂件需要的内容进行调用。

<p>▯<code>ws</code>▯，我们可以看到是一个创建的 Model 对象，Model 对象可以在 src/layout/Model.ts 查询到源码。Model 类包含的主要工作为维护 websocket，思源的 app 除了 http 请求内核进行消息传输之外，还可以通过 websocket 技术从思源内核中主动获取到信息。例如在 App 的 Model 对象 ws 中，我们可以看到在此处它进行了多种类型消息的处理，例如进程加载处理、配置值化处理、下载进度处理、状态栏内容处理等。</p>

<p>▯<code>menus</code>▯，思源右键的初始化菜单。</p>

<p>配置加载，我们可以看到 <code>fetchPost</code> ▯方法调用内核的 <code>/api/system/getConf</code> ▯接口，然后再获取本地存储接口，获取多语言国际化配置接口，然后进行了 <code>ootSync()</code> ▯同步启动，再获取云端用户信息。
上面这些调用是必须在一个完成之后调用另一个，我们看到思源这部分代码是通过回调控制的，调用多，缩进越深，也就是我们常说的“回调地狱”。</p>

<p>▯<code>setNoteBook()</code>▯，笔记本数据查询</p>

<p>▯<code>initBlockPopover()</code>▯，初始化块的悬浮弹出框。</p>

<p>▯<code>promiseTransactions()</code>▯，用于循环控制内核发出的操作执行。例如引用快嵌入快的更新等。</p>

<p>基于以上步骤还很难了解具体界面是如何出来的，主要内容可以查看 <code>onGetConfig()</code> ▯这个方法。</p>
<h2 id="0x03-启动时间轴--">0x03 启动时间轴▯▯</h2>
<p>综合前后端来看，我们可以尝试画出时间轴</p>
<p>▯▯</p>
<h2 id="0x04-小结">0x04 小结</h2>
<p>思源桌面版启动流程比较简洁，而真正复杂度都在内核的启动流程与 APP 从入口到真正渲染完这两个部分，后续，我们继续对这两部分进行探讨。</p>
<p>▯</p>
<p>▯</p>
<p>内容首发 b3log 平台，未经允许，请勿转载</p>