思源笔记缘起

作者：88250
原文链接：https://ld246.com/article/1619868273581
来源网站：链滴

<h2 id="动机">动机</h2>
在选择了很长一段时间的笔记应用后我发现，市面上还没有任何一款笔记应用能同时满足这两个求：

支持标准 Markdown，支持所见即所得。分屏预览带来的割裂感让我难以将注意力完全集中在容创作上
支持块级引用和双向链接。通过块链我可以更高效地发现和整理已有内容，形成知识体系且方便版输出

这两个需求拆分开来都有现成的产品能够实现，可一旦结合就没有好用的产品了。既然没有，那创造一个吧。
<h2 id="存储格式">存储格式</h2>
Markdown 的成功除了语法简洁之外，很大一个因素是因为它是一种开放的纯文本格式，使用通的文本编辑器就可以打开查看和编辑。但随着 Markdown 用户的增多，更多的“高阶”需求逐渐出，为了满足这些需求，有不少项目在 Markdown 基础语法的基础上设计了扩展语法。
经过一段时间的调研和尝试，我们确认了使用标准 Markdown 语法（CommonMark/GFM）无法实现块级引用的，必须引入扩展语法。在众多的 Markdown 扩展语法中，我们选择了 kramdown 提供的内联属性列表（Inline Attribute Lists）语法给块级元素做标识，让每个块元素都拥有唯一的 ID。这样在引用的时候就能精确地获取到该块。保持 Markdown 文本人类易读写前提下最大限度地实现通用性和互操作性。
目前并没有太多平台支持 kramdown 扩展语法，如果需要将笔记内容发布到其他平台上，可以过导出功能来导出标准的 Markdown 文件。
<h2 id="本地离线">本地离线</h2>
笔记应用应该是完全支持离线使用的，只有这样才能保证在最大程度上的可用。在数据同步需求，已经有很多优秀的解决方案，比如各式各样的云盘服务。
笔记可以说是我们一生的知识积累。从可用性和安全性的角度，在几十年的时间跨度上，没有任一款云笔记能够保障持续可用，本地文件的可用性是远远大于在线应用的。本地文件只要做好备份（以定时拷贝到其他离线的存储设备上，也可以通过云盘），基本不会丢失和泄露。云笔记就不一样了有非常多的可能性导致数据丢失或者泄露，想要通过其他机制做备份也基本不现实。可以说云笔记中数据是不受用户自己掌控的。
使用云笔记的唯一优势是方便分享，但作为个人知识管理工具来说，分享并不是刚需，或者说并是高频需求。综上，我们的结论是：无离线，不笔记。
<h2 id="拥抱开源">拥抱开源</h2>
数据可以完全离线以后，还剩下最后一个问题：软件生命周期。
没有任何软件开发团队能保证软件能够持续可用，从开发团队来说，发生软件终止维护的可能性多了，比如：经费不足、研发目标转移或者遇到不可抗力（比如开发人员突然离世）。
有一种有效的方法可以在最大限度上解决软件停更造成的不可迁移，那就是开源。如果软件有足够价值且是完整开源的，即使主创团队无法继续维护了，其他开发者也能接过重。从开发团队实际情况出发，完全开源是比较难办到的，虽然开源商业化模式在很多项目上已经大获功，但并不是每一个项目都适合这样做或者说需要等待适合的时机。
<h2 id="技术架构">技术架构</h2>
<h3 id="B-S-风格">B/S 风格</h3>
思源笔记整体上采用前后端分离的技术架构，通过 HTTP 实现前后端通讯。
<h4 id="前端">前端</h4>
前端目前主要是基于浏览器技术，实现用户交互界面和操作系统平台相关的业务逻辑等。

Electron
浏览器

<h4 id="后端">后端</h4>
后端是常驻内存的 HTTP 服务器，实现核心业务逻辑和状态持久化等。默认在 <code>127.0.0.1

/code> 上监听 <code>6806</code> 端口。
<h3 id="数据状态">数据状态</h3>

持久化数据基于操作系统的文件系统实现，以文件、文件夹存放

只读数据存储于 SQLite 数据库

编辑变更会自动同步到该数据库
数据库临时创建，只在内核运行时存在

<h3 id="发布包">发布包</h3>

桌面应用：基于 Electron 打包，主进程启动后拉起 kernel 内核进程。由 kernel HTTP 伺服界相关静态资源，Electron 主窗口通过 <code>loadURL</code> 加载界面
服务器应用：仅打包 kernel 内核和外观、文档等资源，通过 Docker 镜像 <code>b3log/siyua</code> 分发
手机应用：通过 golang mobile 编译为移动端库，通过 WebView 加载界面

<h2 id="抽象泄漏和奥康的剃刀">抽象泄漏和奥康的剃刀</h2>
<h3 id="抽象泄露">抽象泄露</h3>
<blockquote>
All non-trivial abstractions, to some degree, are leaky.
所有重大的抽象机制在某种程度上都是有漏洞的。
</blockquote>
“抽象泄露”是软件开发领域的术语，它可以推广到很多领域。把较为底层的结构直接暴露出来将抽象的自由度留给用户是最佳选择。
<h3 id="奥卡姆剃刀原理">奥卡姆剃刀原理</h3>
<blockquote>
奥卡姆剃刀原理（Occam's Razor, Ockham's Razor）又称“奥康的剃刀”，它是由 14 世纪英兰的逻辑学家、圣方济各会修士奥卡姆的威廉（William of Occam，约 1285 年至 1349 年）提出。个原理称为“如无必要，勿增实体”，即“简单有效原理”。正如他在《箴言书注》2 卷 15 题说“勿浪费较多东西去做，用较少的东西，同样可以做好的事情。”
</blockquote>
在设计方面，我们应该尽量遵循该原理。所以在思源笔记中，我们基于内容块来构建所有的数据并使用统一的引用语法，尽量降低用户的使用负担。