链滴

Anki 对接思源开发笔记

作者：Clouder

原文链接：https://ld246.com/article/1627227554664

来源网站：链滴

前言

开篇警告：本文来自一个**业余编程爱好者**，多有误导性内容，仅供参考。

相关项目都没有 Warranty，请自行承担相关风险。

已完结，请关注 AnkiSiyuan

相关项目：AnkiLink AnkiIn

相关文章：AnkiImporter: Markdown 导入 Anki 的小工具 AnkiLink 完全体构想 Markdown 源文与 Anki 绑定同步的实现方案

终于，准备工作差不多了。在 Docker 部署思源实现多人协作 的尝试之后，我认为可以开始开发了。事实上，此时思源相关的 HTTP API 已经看上去能用了。实现最初步的导入应该是没什么问题的。

参考文档（比较零碎）

挂件块 Example

挂件块 Issue 中的 API 用法

对接 Web Clipper 中的相关 API 用法

现在开发有一点偷跑的意思、、、

思源侧知识准备

目前计划先实现最基本的：查询最近修改的有标记的块，导入 Anki 中。

做到这一点，只需要一个 API：SQL 查询。

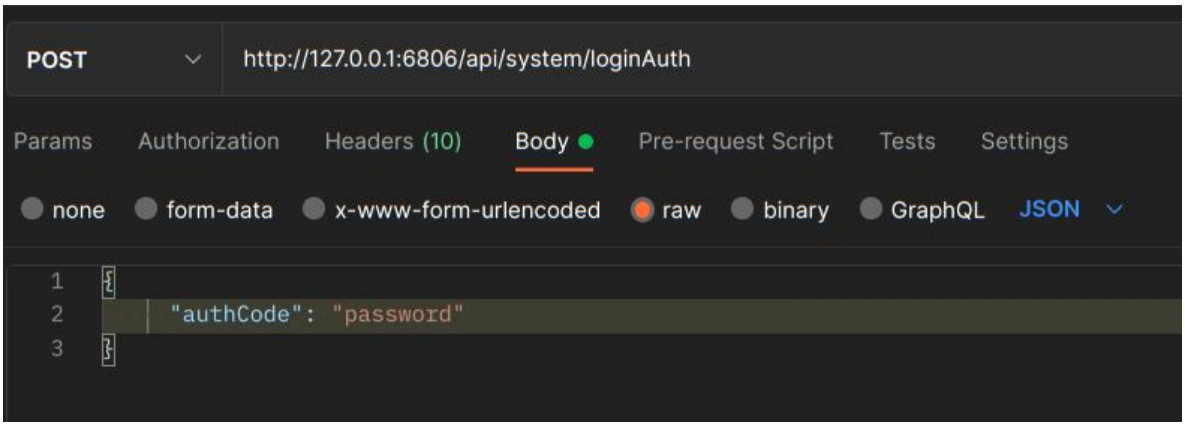
需要用到的 SQL 语句：

- 查询最近修改的、带有某种属性的块。
- 查找某个块的父亲、儿子们。
- 获取属性。

感觉还是很简单的，虽然我还没有学过 SQL，但现场拼凑一下应该也凑合。

鉴权

在 1.2.2 版本以后，API 也需要鉴权了，特此更新。



然后返回带有 Cookies，需要保存下来。

```
def login(password: str):
    res = post(API_URL + "system/loginAuth", authCode=password)
    if res.json()["code"] != 0:
        raise AuthCodeIncorrectException
    global AuthCookie
    AuthCookie = res.cookies.get("siyuan")
```

SQL 查询接口

拿出几天前刚装的 PostMan 比划一下。

首先按照对接 Web Clipper 中的信息：

然后去 PostMan 配置好 Content-Type，选择 Post，再参考 挂件块中的信息：

开发

开发者可以使用自己熟悉的框架（比如 React 或者 Vue.js 等）来开发挂件，示例项目。

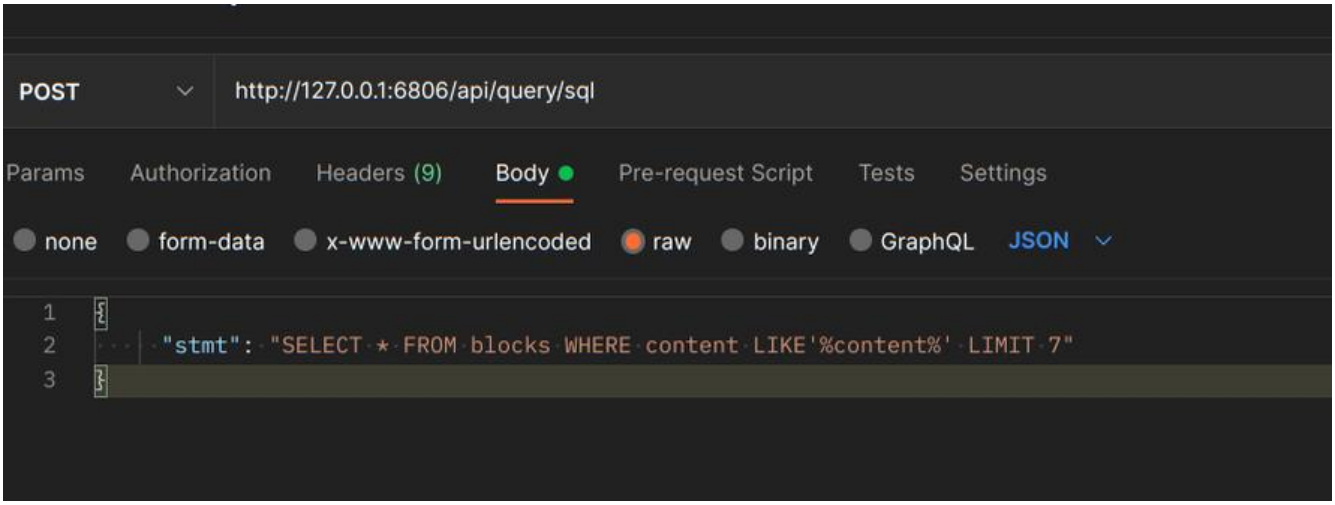
通过内核提供的 HTTP API 来读取数据，比如通过 SQL 查询接口查询内容块：

```
POST /api/query/sql

{
    "stmt": "SELECT * FROM blocks WHERE content LIKE'%content%' LIMIT 7"
}
```

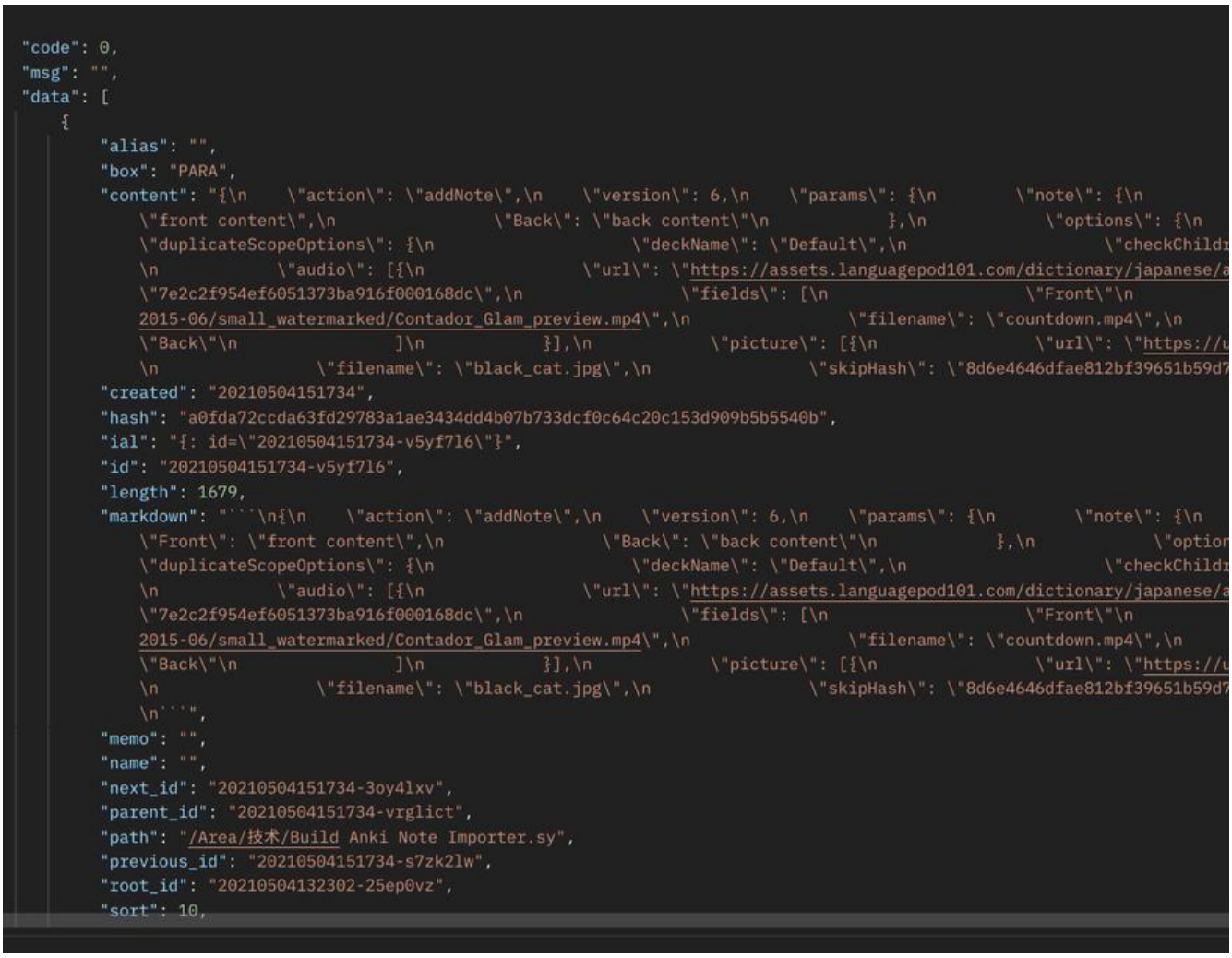
挂件块可以通过属性保存数据，即数据存储在挂件块自身的属性上：

```
POST /api/attr/setBlockAttrs

{
  "id": "挂件块 ID",
  "attrs": {
    "custom-attr1": "value1",
    "custom-attr2": "value2",
  }
}
```

最后这样请求一下：



得到返回值：



```
"code": 0,
"msg": "",
"data": [
    {
        "alias": "",
        "box": "PARA",
        "content": "{\n    \"action\": \"addNote\",\n    \"version\": 6,\n    \"params\": {\n        \"note\": {\n
            \"front content\",\n                \"Back\": \"back content\"\n            },\n            \"options\": {\n
            \"duplicateScopeOptions\": {\n                    \"deckName\": \"Default\",\n                    \"checkChildr
            \n            \"audio\": [{\n                \"url\": \"https://assets.languagepod101.com/dictionary/japanese/a
            \"7e2c2f954ef6051373ba916f000168dc\",\n                \"fields\": [\n                    \"Front\"\n
            2015-06/small_watermarked/Contador_Glam_preview.mp4\",\n                \"filename\": \"countdown.mp4\",\n
            \"Back\"\n                ]\n            }],\n            \"picture\": [{\n                \"url\": \"https://u
            \n                \"filename\": \"black_cat.jpg\",\n                \"skipHash\": \"8d6e4646dfae812bf39651b59d7
        "created": "20210504151734",
        "hash": "a0fda72ccda63fd29783a1ae3434dd4b07b733dcf0c64c20c153d909b5b5540b",
        "ial": "{: id=\"20210504151734-v5yf716\"}",
        "id": "20210504151734-v5yf716",
        "length": 1679,
        "markdown": "```\n{\n    \"action\": \"addNote\",\n    \"version\": 6,\n    \"params\": {\n        \"note\": {\n
            \"Front\": \"front content\",\n                \"Back\": \"back content\"\n            },\n            \"optior
            \"duplicateScopeOptions\": {\n                    \"deckName\": \"Default\",\n                    \"checkChildr
            \n            \"audio\": [{\n                \"url\": \"https://assets.languagepod101.com/dictionary/japanese/a
            \"7e2c2f954ef6051373ba916f000168dc\",\n                \"fields\": [\n                    \"Front\"\n
            2015-06/small_watermarked/Contador_Glam_preview.mp4\",\n                \"filename\": \"countdown.mp4\",\n
            \"Back\"\n                ]\n            }],\n            \"picture\": [{\n                \"url\": \"https://u
            \n                \"filename\": \"black_cat.jpg\",\n                \"skipHash\": \"8d6e4646dfae812bf39651b59d7
            \n```",
        "memo": "",
        "name": "",
        "next_id": "20210504151734-3oy4lxv",
        "parent_id": "20210504151734-vrglict",
        "path": "/Area/技术/Build Anki Note Importer.sy",
        "previous_id": "20210504151734-s7zk2lw",
        "root_id": "20210504132302-25ep0vz",
        "sort": 10,
```


观察了一下，这个返回的其实是一个块的 List？那么我们先学习一下块的数据结构。这个在用户手册其实有相关内容。

Field	Description
id	Content block ID
parent_id	Parent block ID, If the content block s a document block, this field is empty
root_id	Root block ID, which is the document lock ID
box	Notebook name
path	Document path where content block is l cated
name	Content block name
alias	Content block alias
memo	Content block memo
content	Text with Markdown markers removed
markdown	Text with complete Markdown ma kers
type	Content block type, please refer to"here"

subtype	Content block subtype, please refer to here"
ial	Inline attributes list, like{: name="value"}
sort	For sorting, the smaller the value, the higher the sort
created	Create time
updated	Update time

找出几个我们要用到的：markdown parent_id updated ial

在这之前，先造个轮子。

需要用到 Requests 库，从 pip 安装。

```
import requests
import json


API_URL = "http://127.0.0.1:6806/api/"
HEADERS = {"Content-Type": "application/json"}


def post(url: str, **params):
    try:
        response = requests.post(url, data=json.dumps(
            params), cookies={"siyuan": AuthCookie})
        return response
    except Exception:
        raise Exception


class ApiException(Exception):
    pass


def query_sql(SQL: str):
    result = post(API_URL + "query/sql", stmt=SQL).json()
    if result["code"] == 0:
        return result["data"]
    raise ApiException(result)
```

然后造一个思源的 Block 的对象。这里省略了。

查询带有某种属性的块

这里的属性指的是 Attribute，也就是支持自定义那个。

大概是在 ial 这个属性中的，我们创建一个测试块进行相关查询。

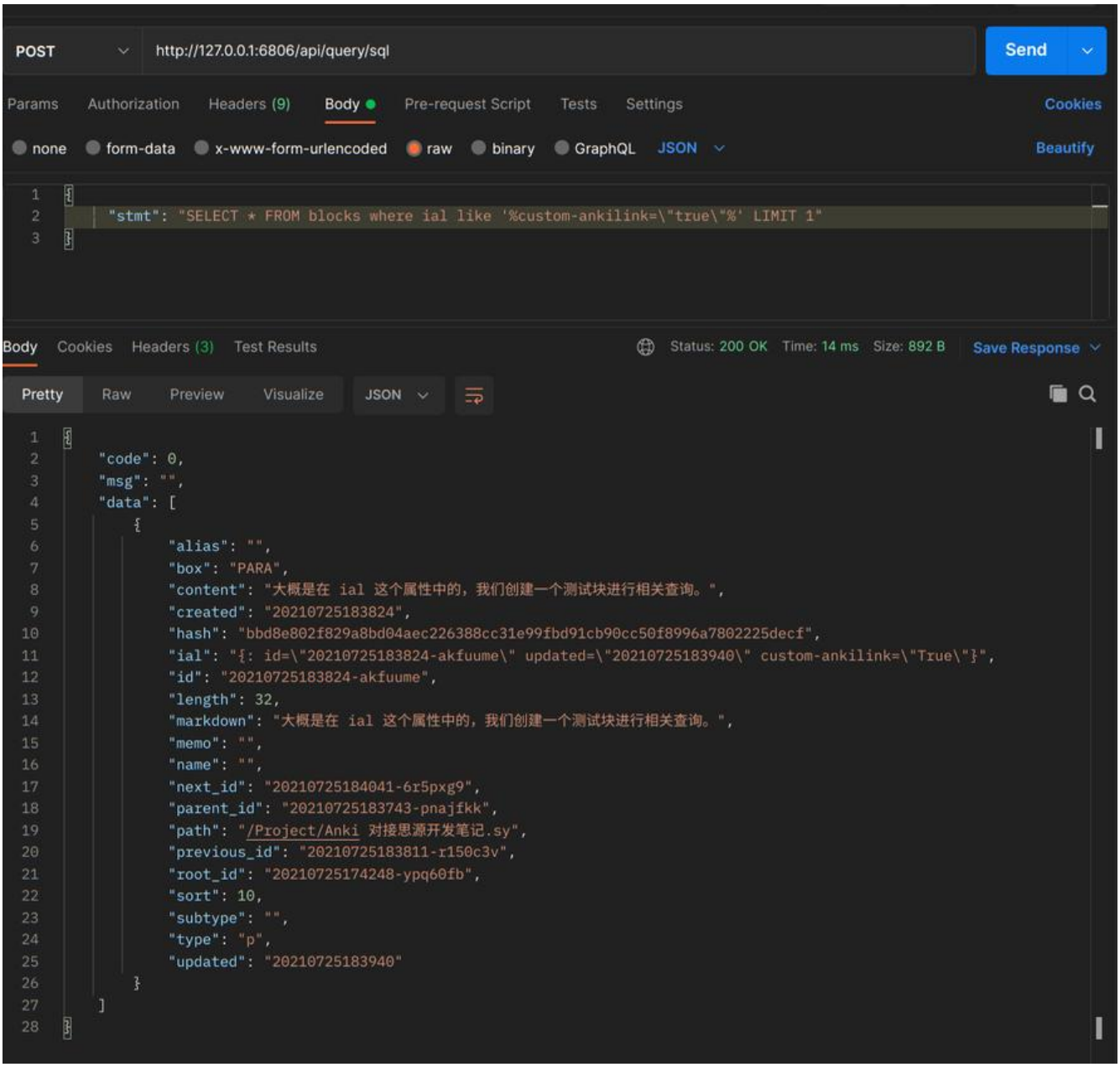
"ial": "{: id=\"20210725183824-akfuume\" updated=\"20210725183940\" custom-ankilink=\"true\"}",

可以看到，自定义的属性会以 custom 开头，并且名称会统一转换为小写。有趣的是，值不会被转换小写。

经过一番折腾，我们发现这个 ial 实际上类型是文本、、、那么我们来写一条 SQL 语句：

SELECT * FROM blocks where ial like '%custom-ankilink=\"true\"%' LIMIT 1

大小写是不敏感的。



查询某个时间点之后的块

这个时间格式让我有些迷糊、、、

感觉我已经对 SQL 理解大增了！

那么要筛选某个时间更新后，无非加个 AND 条件，然后用大于号比较一下。

直接大力字符串比较即可。原来这样存时间别有深意、、、

```
SELECT * FROM blocks where ial like '%custom-ankilink=\"true\"%' AND updated > '20210725185700'
```

确实很简单啊。

按 ID 查找块

这也不是个困难的活。

```
SELECT * FROM blocks where id='20210725183824-akfuume'
```

这是一个很常用的语句，我们写到代码里。

```
def find_by_id(id: str) -> Block:
    res = query_sql(r"SELECT * FROM blocks where id='{}'".format(id))
    if len(res) <= 0:
        raise NullBlockException
    return res[0]
```

找某个块的父亲

不难发现，一个 Block 的属性中带有 parent_id 这一项，那么这个很自然就实现了。

找某个块的儿子

我以为思源内部是按树形组织的，但实际上似乎存储的数据结构与我料想的有所出入。

事实上，每个块有一个唯一的父亲，这当然是一棵树。但我们无法从父亲直接获得儿子的 List，而只获得一个儿子，也就是线性文本排列时的下一个。

这其实让我想起了一种多叉树转二叉树的手法：左儿子作为真正的儿子，而右儿子作为兄弟。

当然了，这里没有那么复杂，只是一个类似链表的结构罢了。

说了这么多，其实找某个块的儿子用 SQL 一下就做到了。

```
SELECT id FROM blocks where parent_id='20210725192017-w3xxrdb'
```

找父亲则直接找到块，然后得到 parent_id 即可。

解析块属性

这是为了后续的 AnkiLink 配置项服务的。

这个 ial 的文本格式是 Inline Attribute List 之类的，总之不是我认识的格式。

不过看起来很简单，直接正则解析就可以了。

ial 格式：{: id="20210725192239-sm6uroj" updated="20210725192314"}

```
(.+?)="(.+?)"

def parse_ial(text: str):
    ret = {}
    subs = re.finditer(r" (.+?)=\"(.+?)\"", text)
    for sub in subs:
        ret[sub.group(1)] = sub.group(2)
    return ret
```

```
>>> from AnkiIn.helper.siyuanHelper import *
>>> t = find_by_id("20210725185945-rykie12")
>>> print(t.properties)
{'id': '20210725185945-rykie12', 'updated': '20210725191455'}
>>>
```

差不多了，还写了一些其他的乱七八糟的函数。

相关页面：https://github.com/Clouder0/AnkiIn/blob/feature/SiYuan/AnkiIn/parser/siyuan.py

AnkiIn 侧

准备工作终于完成了、、、有点疲惫。

根据"AnkiLink 完全体构想"中的描述走吧。

请注意：以下内容描述混乱，阅读需要强大的语文理解能力与初中生级别的数据结构、算法功底。由本人乱写一通，建议各位跳过与算法有关部分。

找到上一次同步后更新的所有带有 ankilink=true 属性的块，然后进行同步。

这里有一个非常严重的问题：某个块更新了，其标题对应的更新时间不会改变。

但是超级块是可以的。

这不禁让我有些混乱，那只能选择比较随便的方法了：获取某时间后更新过的所有块，然后查找它的宗们是否被标记。

将某时间更新后的所有块保存下来可能耗费大量内存，因此只 Select ID.

然后开始对每个块爬树，查找其祖宗对应的 ial 信息。

在查找信息的时候，可以顺路把配置传递下来。

感觉略有一些复杂，那干脆直接把语义树建出来好了。

建语义树的话，由于省略了大量的节点，导致我们**不可避免地丢失了原先的顺序信息**，只能留下层级信息。

如果思源后续添加相关信息的话，或许还能把顺序搞回来。

事实上，只需要向上爬，找到第一个被标记了的祖先，就可以直接返回了。

因为节点**不应该**有多个被标记的祖先，任何一个标记的节点都支配了其所在子树，而因此其祖宗若被记，其祖宗的支配子树一定包含其支配子树，从而使这个标记无意义。

理论上增加程序鲁棒性确实需要处理这种情况，然而其实我们只需要稍加思考，就能意识到不处理也以得到完全正确的结果。（大概是这样的？）因为如果有这个祖宗的支配内而不在当前支配子树内的点修改且需要更新，那么它会自然地找到被标记的祖宗而获得更新。实质上，这将在原树上联通的两分分拆成了两棵子树，但对于关键点的访问是没有影响的。

但这也启发我们必须对访问过的节点进行标记，否则就可能出现由于多个标记根互为父子而导致的重导入现象。

此外，这可能会导致来自祖宗的配置失效。由于我们自下向上爬，难以获取节点深度，因此暂且不管。

后续重构可以考虑牺牲常数来做些更简单的实现、、、

来描述一下算法：

- 从某个叶子节点出发，向上查找父亲。
- 如果其父亲曾经被遍历过，则连边后直接退出。
- 如果其父亲未被遍历过，则递归向上遍历父亲。
- 若当前节点已经被标记，则加入根数组，并直接退出。

曾经有一个比这个复杂得多的算法、、、然而 O(n) 复杂度大概不用这样过度设计罢、、、

建出树之后，进行一次深度优先搜索。

在搜索过程中下传配置项即可。

我计划将配置项存放在容器块的块属性中，然而目前不支持多行，那么以后再说吧。

在这里遇到了一个特别阴间的 Bug，调试得我怀疑人生、、、Python 究竟是什么奇怪的语言、、、

```
class SyntaxNode:
    def __init__(self, id: str, parent=None, sons=[]):
        self.id = id
        self.parent = parent
        self.sons = sons
```

这样写，所有的 SyntaxNode 的 sons 指向的**居然是同一个 List**、、、这是什么奇怪的设计啊。就这个 Bug 消磨我了一个小时、、、

相关页面：https://github.com/Clouder0/AnkiIn/blob/feature/SiYuan/AnkiIn/helper/siyuanHeler.py

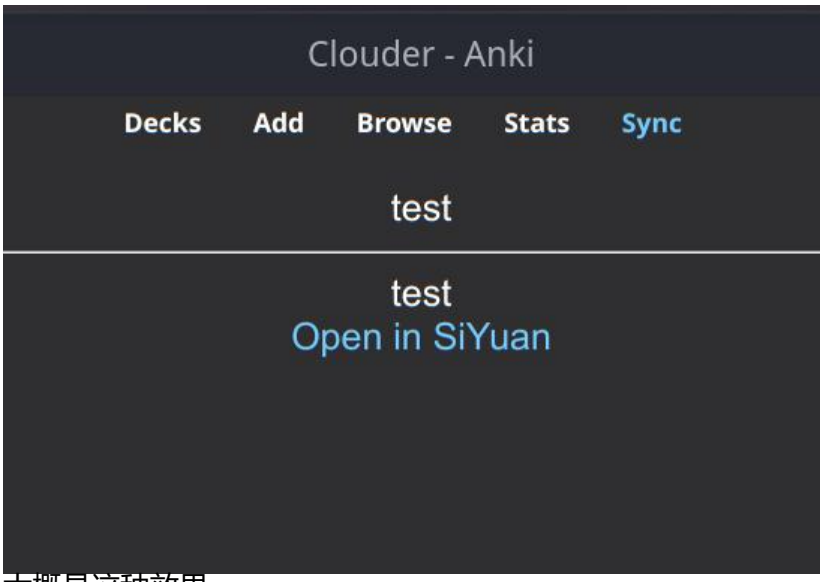
配套数据结构

思源的块 ID 是非常有用的信息。

AnkiLink 在最初的 Note 类中并没有加入 ID 属性，事实上 Anki 似乎也没有这个属性（或者我不知），但我们可以定制 Anki 的 NoteType，加入一个 Field 专门用于存放思源 ID.

事实上还可以做进一步的设计，在 Anki 模板中直接加入利用思源 ID 跳转回思源的超链接。

可惜我是孤儿的 Linux 用户，思源超链接无法使用、、、不过还是加上这功能。



大概是这种效果。

由于与之前的内置 NoteTypes 的识别方式完全一致，仅仅多了一个传入的 SiyuanID 的 Field，所以概是直接复用代码罢。

这里涉及到从父类生成子类的一个小问题，用不太优雅的方法处理了一下。具体的，直接更新内置的 __dict__ 变量，然后给 fields 加一项用于保存思源块 ID 的键值对。

代码质量对我来说已经没那么重要了，能用是第一优先级。至于代码质量，可以日后慢慢重构。对于 nkiLink 这个量级的小工具，一天之内就能把项目代码全部重构一遍。

移植了如下 Note Types：

- Question&Answer
- Multiple-Line Question&Answer
- Cloze
- ListCloze
- TableCloze

把选择题砍掉了，因为我现在已经不用这个类型了。

经过了一番折腾，将 Note Types 移植过来之后，大概是能获取 noteList 了。

后续还需要实现配置项的内联，不过这也是后话了。

最后发现设计得有点问题、、、

AnkiSiyuan 侧（用户交互）

库写好了，再去写用户脚本。

按照计划，更新之类的操作是在用户脚本中判断的，而库只负责获取到对应的 noteList.

需要保存一个上次同步的时间，避免重复同步。需要鉴权码配置项。

查看更多，访问 AnkiSiyuan.

结语

其实后面这些内容都与思源没什么关系了。

AnkiSiyuan 目前差不多到达了勉强能用级别，进行一定的测试后就会发布了。

只实现了导入和同步思源内容更新到 Anki，后续可能要开发 Anki 插件来实现反向同步，也可能直接了。