做时间的朋友 —— 用印象笔记打造时间记录工具

原创 太阳雪 Python技术 1周前

相信很多朋友听说或读过李笑来写的《把时间当作朋友》，其中介绍了一个记录时间日志的方法，用来帮助我们了解自己的状态，培养对时间的感知。但是实践起来并不容易，特别是没有好的记录工具，今天介绍下我是如何利用 Python 来分析时间日志的

由来

在实践记录时间日志的过程中，我尝试过很多记录工具，如浏览器插件，APP，甚至纸质记事本，但没有特别合适的，要么记录不方便，要么统计不方便。

最终我选择用 印象笔记 记录时间日志，因为它支持多种客户端，而且形式随意，最重要的是我习惯用它记录各种东西。

经过各种日志版本的时间，最终形成了适合自己的风格：

- 每天一个日志，日志日期命名，如 2020-03-19
- 日志存放在 “时间的朋友” 笔记本里
- 日志标签为 “时间账”
- 每天开始是记录一条开始时间
- 要进行下一件事时，记录一条当前做的事情
- 每条记录由 结束时间、内容、分类、耗时 四个节段组成，之间用 tab 或 4个空格 分隔

例如：

```
09:05    开始工作
09:09    冥想    健康    4
09:32    开完早例会    工作    23
09:41    整理会议记录    工作    9
09:56    协调项目工作    工作    15
10:06    看网页休息    休息    10
...
```

最后每天结束时，需要做下统计，以增强对时间的感觉，例如我的统计大概是这样：

```
总时长：8h

事业:4.7h(36%)
工作:3.9h(30%)
成长:0.41h(3%)
...
```

但是，每天都忙的要死，在结束一天工作时又困又累，做一个需要计算的统计工作实在很为难，果然，没坚持几天。

有个偶然的机会，得知印象笔记提供 API，喜出望外，可以用编程的方式来处理笔记，做个统计分析不在话下，于是有这样的解决方案：

1. 用 Python 调用 印象笔记 API 读取时间日志
2. 对日志内容进行解析，做统计分析
3. 将统计分析结果回写日志

实践

概念和原理

要用好 印象笔记 的 API 需要了解一些概念和原理。

概念

- `NoteStore`： `NoteStore` 是印象笔记的主模块，通过 `NoteStore` 实例可以访问所有的 API 操作
- `guid`：印象笔记中各个对象的唯一标识，如标识具体的 笔记本、标签、笔记等，要操作某个实体，需要先获取到它的 `guid`
- 数据结构：印象笔记中将除了 `NoteStore` 以外的模块或类都叫数据结构，通过 `evernote.edam.type.ttypes` 和 `evernote.edam.notestore.ttypes` 两个模块来定义，在接口中提到的

原理

- 为了提高传输效率，将数据分为属性部分和内容部分，例如在获取列表类的操作中，得到的是对象的属性集合，如果要获取内容，通过 guid 调用单独的接口
- 接口中，除了 `guid` 和 `token` 外，其他参数都是通过相应对象实体传递的，例如查询笔记接口 findNotes，具体条件通过 NoteFilter 结构传递

申请开发者 token

印象笔记提供两种授权模式

- 沙箱模式，支持 OAuth 验证，可以为多个用户提供基于 印象笔记 的服务
- 普通模式，仅支持自己的账号，可以用 API 来操作自己的资源

我采用普通模式，一是申请方便，二是只用来处理自己笔记

请求普通开发模式的地址是：https://app.yinxiang.com/api/DeveloperToken.action

填写自己的登录信息（印象笔记的用户名，密码），进入获取 开发者 token 的页面，点击创建 token 按钮，创建 token，复制并像保存账号密码一样妥善保存 token。

如果泄露或者丢失，还来这里重置。token 有效期为一周，过期需要重新申请。

安装 SDK

印象笔记提供了多种语言的 SDK，我们选用 Python3 版的，地址是：https://github.com/evernote/evernote-sdk-python3

下载源码后，解压，在解压文件夹下执行:

```
$ python setup.py install
```

一切正常的话，执行下面语句，不会报错：

```
$ python -c 'from evernote.api.client import EvernoteClient'
```

> 注意：通过 `pip install evernote` 也能安装，不过是 Python2.x 版的

API 测试

运行下面测试代码，如果不报错，说明 API 调用成功：

```
from evernote.api.client import EvernoteClient

auth_token = "S=s1:U<...>5369"  # 开发 token

client = EvernoteClient(token=auth_token, sandbox=False,china=True)

note_store = client.get_note_store()
```

- 引入 印象笔记 客户端模块
- 用 `EvernoteClient` 创建客户端实体，因为是普通开发模式， `sandbox` 和 `china` 参数设置为 `False` 和 `True`,与沙箱模式相反，请注意
- 通过客户端实体，获得 `note_store`，如果能成功执行，说明一切顺利

实现

SDK 中提供了很多方法，不过使用并不方便，所以创建了一个代理类 `EverNote`，将需要用的操作包装成习惯用的，创建类的好处还有，不必将 token 和 evernote 终端实体到处传。

EverNote 定义

```
class EverNote():
    def __init__(self, auth_token):
        self.auth_token = auth_token
        self.client = EvernoteClient(token=auth_token, sandbox=False,china=True)
        self.note_store = self.client.get_note_store()

        self.template = '<?xml version="1.0" encoding="UTF-8"?>'
        self.template +='<!DOCTYPE en-note SYSTEM "http://xml.evernote.com/pub/enml2.dtd">'
        self.template +='<en-note>%s</en-note>'
```

- 构造方法接受一个参数 `auth_token`
- `auth_token` 被设置为内部变量，为了反复使用
- 创建 evernote 终端实例， client，由于是普通模式，设置 sandbox 为 False, china 为 True
- 通过 终端实例 client 获得 NoteStore 实例，后续操作都基于这个实例
- template 是 evernote 笔记的 xml 框架，笔记内容放在 `en-note` 标记之间

工具方法

SDK 虽然提供了很多方法，但并不方便用，例如没有提供通过 标签名 获取 标签 guid 的方法，所以需要基于现有 API 写一个，例如 `getTagGuid` ：

```
def getTagGuid(self, name):
    if name is None:
        return None

    for tag in self.note_store.listTags():
        if tag.name == name:
            return tag.guid
```

- 方法接受 标签名 作为参数
- 如果提供的 标签名 为空，则返回空
- 如果 标签名 不为空，通过接口 `listTags` 获取所有标签，遍历以找到 标签名 匹配的标签，返回其 guid

除了 `getTagGuid`，还定义了：

- `getNotebookGuid` 根据名称获取 笔记本（ `notebook` ） 的 guid
- `getNoteGuidByTagGuid` 根据 标签（ `tag` ） guid 获取 笔记（ `note` ） 的 guid
- `getNoteContent` 根据 笔记（ `note` ） 的 guid，获取笔记的原始内容（xml 格式）
- `getNoteText` 根据 笔记（ `note` ） 的 guid，获取笔记的文本内容（text 格式）
- `createNote` 创建新笔记
- `updateNote` 更新已有笔记，例如用来将统计信息写入笔记
- `createTag` 创建标签，为 `createNote` 方法提供支持

具体代码可从文后代码示例中获取

解析笔记

直接获取到的 日志内容 是 xml 格式的，需要将它转化为 text 格式，以便分析，这里借助 `html2text` 模块，可以用 pip 安装：

```
pip install html2text
```

最终将获取笔记的 text 格式，封装成 `getNoteText` 方法，返回的内容和通过 印象笔记 软件看到的一样。

由于记录日志时采用了特定的格式，解析比较方法，大体是将笔记切分成行，逐个处理每一行，并按分类进行了累计，

需要注意的是：

1. 记录笔记时输入的 tab 会转化为 4 个空格
2. 每行会被 两个 回车符 分隔

下面是解析代码：

```
def analysisNote(self, notetext):
    # 日志行格式：
    # 0时间 1结束时间 2内容 3分类 4耗时

    rows = []
    statistics = {  # 统计结果
        "total": 0,  # 总时间
        "group": {}  # 按分类分组统计
    }
```

```
    }

    # 当天记录的开始时间
    lines = notetext.split('\n\n')
    startTime = lines[0].split(" "*4)[0]

    begin = startTime  # 每个记录的开始时间
    for l in lines[1:]:
        if not l.strip():
            continue
        row = {}
        parts = l.split(" "*4)
        row['begin'] = begin
        row['end'] = parts[0]
        row['description'] = parts[1]
        row['type'] = parts[2]
        row['cost'] = parts[3]
        rows.append(row)

        # 分组统计
        group = statistics["group"].get(row['type'], 0)
        group += int(row['cost'])
        statistics["group"][row['type']] = group

        begin = row['end']  # 当前行时间为下一行开始时间

    # 计算总体时间
    if startTime > begin:
        statistics['total'] = difftime("2020-01-02 " + begin + ":00", "2020-01-01 " + startTime + ":
    else:
        statistics['total'] = difftime("2020-01-01 " + begin + ":00", "2020-01-01 " + startTime + ":

    return  rows, statistics
```

处理过程比较简单，主要是对文本的处理，不同的记录格式有不同的解析方式，如果你在实践中格式不同，需要修改这个方法。

计算整体时间这里需要说明下

- 日志中第一个时间是当天开始时间，最后一行的时间是当天结束时间，会出现跨天的情况，例如早上 9点开始，临晨1点才结束工作，即跨天了，所以日期应该是第二天，为了方便直接用 2020-01-01 作为第一天，2020-01-02 作为第二天
- `difftime` 是个工具方法（具体实现参考文后代码示例），可以计算两个时间的差值，第三个参数是差值单位，着用的是 分钟，为了和记录的耗时单位匹配

统计分析

有了对日志的解析结果，就可以方便的计算出 总时长，以及每种分类的占比情况；还可以计算出有记录的时间长，和无记录时间长的比例，看看有多少暗时间；还可以定义有效标签，统计有效时间，例如将 工作、写作、健康 定义为有效活动，合并它们的时间就可以得到有效占比等等，总之这一步实现了非结构化信息到结构化信息的转变，为统计分析通过了数据基础。

解析结果还返回了日志行结构化数据，可以将其存储到数据库中，并且将统计信息按照方面（角度的另一种说法）存储起来，做时间跨度更大的统计分析，例如一周、一月、一个季度等，有了这些数据，结合前面 《Flask 数据可视化》中的方法，用图表展示。

回写笔记

统计分析完成后，将结果写到被解析的日志后面，已完成最初的目标 —— 代替手工统计

印象笔记 的笔记格式是 xml，直接写 xml 不是很方便，需要将 text 格式转换为 xml 格式，于是写了将 text 转化为 印象笔记特有的 xml 格式 方法 `formatNote`，以及支撑方法 `formatLine`：

```
def formatNote(self, text): # 将文字转换为印象笔记格式
    content = []
    for line in text.split("\n"):
        content.append(self.formatLine(line))
    return "".join(content)

def formatLine(self, line):
    tabstr = "    "
    if line:
        return "<div>" + line.replace(" "*4, tabstr).replace("\t", tabstr) + "</div>"
    else:
        return "<div><br/></div>"
```

- `formatNote` 接受 text 格式字符串，将其转化为 `en-note` 标记之间的内容
- `formatLine` 接受 text 格式的一行，转化为 `div` 标记格式
- 记录笔记时输入的 tab，会转化为 4 个空格，xml 格式为 `   `
- 空行要被转化为 `<div><br/></div>`

最终这些细节封装到 `updateNote` 方法中，提供要修改的笔记的 guid，要添加的内容，就可以通过 SDK 更新到印象笔记中，至此完成了最初的设想。

集成

前面过程组合起来，提供日志名，最后完成添加日志统计信息的功能：

```
auth_token = "S=s1:U<...>5369"
client = EverNote(auth_token)  # 创建代理实例

# 获取日志 text 内容
tagGuid = client.getTagGuid("时间账")
noteGuid = client.getNoteGuidByTagGuid("2020-03-18", tagGuid)
noteText = client.getNoteText(noteGuid)

ret = client.analysisNote(noteText)  # 解析并分析

client.updateNote(noteGuid, ret.result) # 添加分析结果
```

自动化

如果感觉每天调用一次脚本很麻烦，可以将其配置为自动执行

linux 操作系统，可以用 crontab 配置定时任务，定时运行脚本

windows 操作系统，可以配置计划任务

另外还利用定时框架，如 `schedule` 编写服务，有多种玩法可以尝试一下

需要注意的是 python 的运行环境问题，如果是在虚拟环境中的，需要在脚本中激活虚拟环境

总结

今天结合分析 时间日志 的实际场景，介绍了 印象笔记 SDK 的使用，从最初的思路，到最终的实现，希望对您有所启发，请参考示例代码做做实践，也许会有更好的玩法。另外在示例代码中附带了测试类，以便修改后检测功能。

“人生苦短，我用 python”，今天的实践，就是个很好的阐述。

参考

API: https://dev.yinxiang.com/doc/reference

SDK: https://github.com/evernote/evernote-sdk-python3

https://dev.evernote.com/doc/start/python.php

https://stackoverflow.com/questions/41710896/getuser-return-edamsystemexception-errorcode-8

https://www.jianshu.com/p/bda26798f3b3