OhMyMN
MarginNote 插件开发框架

同时也是一个可以自动处理摘录的工具箱

使用指南 | 开发文档 | API 文档 | 立即下载 | 查看源码

>>> 给插件开发者带来

现代化的开发框架

使用 TypeScript 开发，ESbuild 打包。封装大量 API，助力你开发更加强大的插件。

控制面板及配置管理

轻松创建具有控制面板的插件，拥有强大的配置管理功能。

模块化开发

像搭积木一样不断累加模块，最终开发出强大的插件。

>>> 给普通用户带来

自定义

有了控制面板之后，MarginNote 插件终于可以自定义了。

自动化

无需手动开关插件，一切都将按照你的要求自动执行。

相互协作

不同模块之间可以相互协作来实现更神奇的作用。

手势和快捷键

通过手势或者快捷键快速对脑图卡片进行处理，效率满分。

贡献者

简介

> **模块**
>
> 可以类比插件，是 OhMyMN 内部的插件。

首先要明白 OhMyMN 本质上是一个工具箱。里面所有的功能都是单独的模块，每个模块都被赋予了三种能力：

1. 摘录时，修改摘录的内容或者获取摘录内容并进行其他操作。
2. 脑图中选中卡片后，对卡片进行修改或获取信息并进行其他操作。
3. 文档中选中文字后，获取文字或选区信息并进行其他操作。

有的模块可能三种能力都用了，有的可能只用了其中一个，也有的一个都没有，仅仅只是一些选项。

使用第一种能力的模块通常以 Auto 开头，比如 AutoTitle，AutoDef，表示可以在摘录时自动执行（默认不执行，需要开启 `摘录时自动执行` ）。使用第一种能力的模块通常还会使用第二种能力，以便处理已经存在的卡片。

第二种能力和第三种能力也通常同时使用，比如用来复制，搜索，导出。它们有一个共同的名字 —— 动作（Action）。所有模块的动作都会出现在 MagicAction for Card 和 MagicAction for Text 中，也就是一个按钮，点击就会执行相应的动作。

除此之外，所有模块被分为了两大类：

1. 必选模块：无法关闭的模块。
   - OhMyMN
   - MagicAction for Card：一些与卡片有关的动作
   - MagicAction for Text：一些与文本有关的动作
2. 可选模块：可以选择开启的模块，可以在 `OhMyMN-模块快捷开关` 中启用
   - Shortcut：使用 URL Scheme 触发动作，可自行设置快捷键来打开 URL Scheme。
   - Gesture：使用手势触发动作。
   - CopySearch：复制或搜索选中的文字或选中的卡片。
   - AI: AI 动作，自定义 Prompts。
   - AutoX
     - Another AutoTitle：自动转标题。

    - Another AutoDef：自动拆分摘录为标题和摘录两部分，提取标题。
    - AutoFormat：自动格式化摘录，比如自动添加空格。
    - AutoComplete：自动补全英文单词词形，填充单词信息，制成单词卡片。
    - AutoReplace：自动替换摘录中的内容。
    - AutoList：自动在指定位置换行，添加序号。
    - AutoTag：自动添加标签或者提取部分内容为标签。
    - AutoComment：自动添加评论或者提取部分内容为评论。
    - AutoStyle：自动设置摘录颜色和填充样式。
    - AutoOCR：自动对摘录的选区进行 OCR 识别或者矫正。
    - AutoTranslate：自动翻译摘录的内容。
    - AutoSimplify：自动将繁体转为简体中文。
- ~~Export to X~~
    - ~~Export to Flomo~~
    - ~~Export to Anki~~
    - ~~Export to Devonthink~~

注意事项

1

OhMyMN 的使用逻辑与以往的插件均不相同，尤其是以 Auto 开头的模块，是真正的自动化。你需要设定执行的条件并开启 **摘录时自动执行**，让它可以在你需要的时候自动执行。模块通常会提供预设，你可以直接选用，也可以自定义。OhMyMN 不推荐无差别执行，所以没有提供全局的开关来一键开启或关闭 OhMyMN。

2

所有模块的预设中自定义的优先级始终最高。

3

如果你正在创建笔记本，需要在创建完成后重新进入，让 OhMyMN 知道你刚导入的文档。

4

OhMyMN 只能处理摘录，从浏览器中拖进来的或者自己输入的都属于评论。

5

不要将 OhMyMN 和其他在摘录时自动执行的插件同时使用，可能会出现冲突。

6

自定义输入时需要 **回车** 确认，会出现保存成功的提示，同时输入法关闭。否则就是输入错误。

7

在插件设置中停用 OhMyMN 可以 **清空配置**，如果出现错误导致崩溃，可以停用再启用插件。当然，大部分情况下都不需要清空配置，不然重新设置起来会非常的麻烦。

8

如果你划重点，MarginNote 会重新获取新的摘录内容，进而导致 OhMyMN 的处理失效。可以在划完重点后稍微修改一下摘录选区，即可重新处理摘录。

9

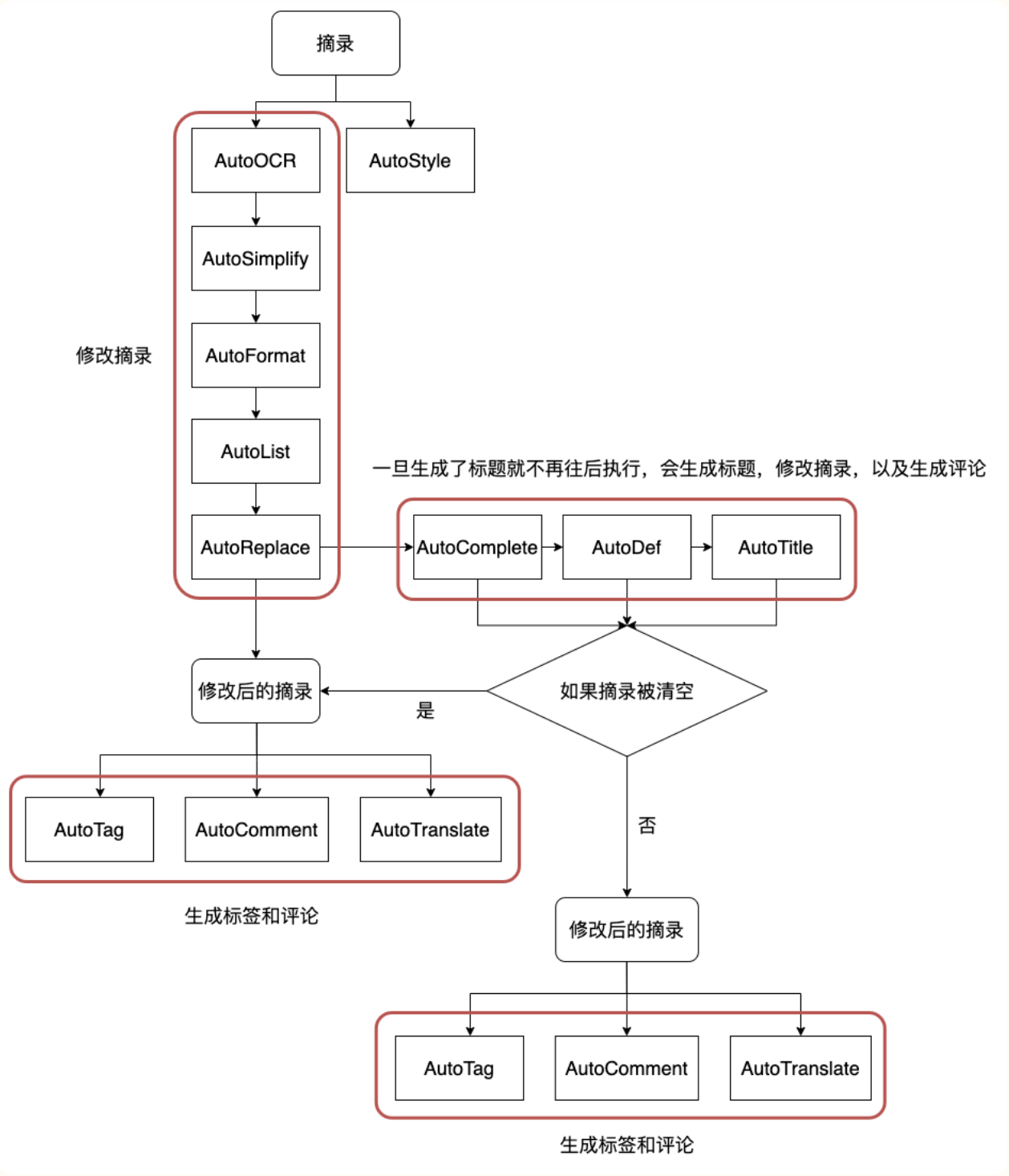
划完重点后，OhMyMN 获取到的重点两边会出现两个 `*`，比如 `**这是重点**`，在替换，筛选，提取等操作时要格外注意。

10

如果安装插件后看不到笑脸 Logo，可以尝试重现安装，如果还是不行，说明系统版本太低，可以尝试升级系统。

11

想要让 Auto 模块协同工作，必须先了解一下模块的执行顺序。



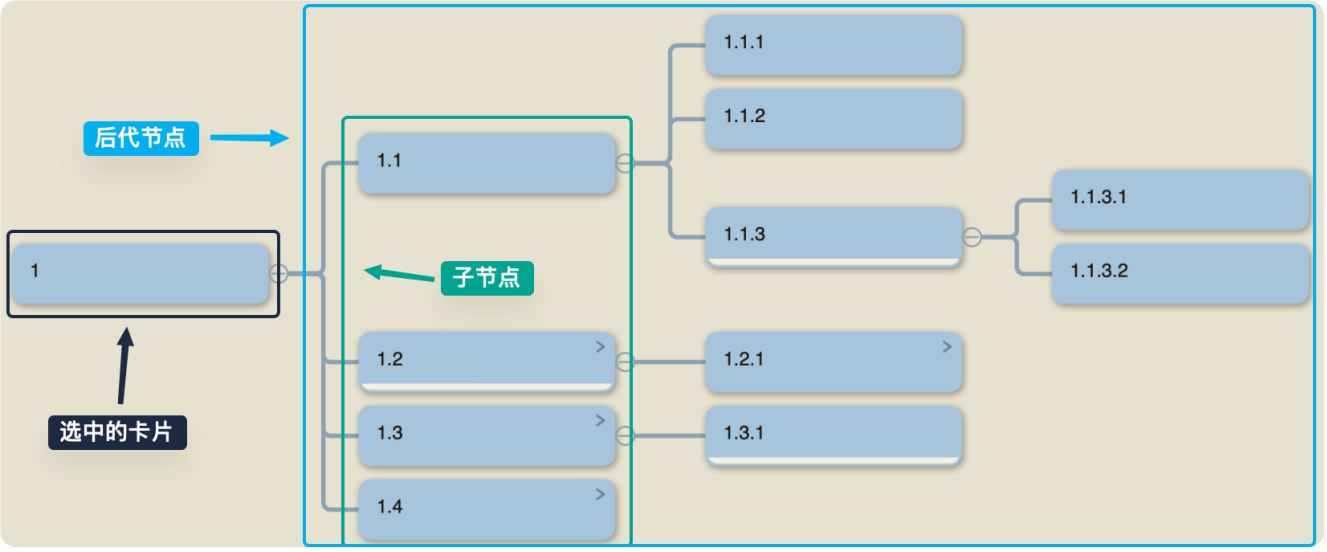
基本概念

摘录、笔记、卡片、评论

- 摘录（Excerpt），既可以当作动词，也可以当作名词。既是指从文档中选择一段文字或图片，给它加上颜色，变成脑图卡片这一过程，也是指这个文字或图片本身。这里特指从文档中摘录，从其余地方，比如浏览器中拖入 MarginNote 的都是属于评论。
- 笔记（Note），摘录其实就是笔记，每条摘录都会给它分配一个 noteid，卡片的 URL 就是这个 noteid。
- 卡片（Card/Node），Node 是在脑图语境下节点的意思，也是指的这张卡片。一张卡片里可以包含摘录，评论，标签，链接等很多内容，一张卡片原则上只有一条摘录。
- 评论（Comment），其实上面所说的标签和链接本质是也是评论。评论的特点就是不与文档关联。而摘录就会有一些文档相关的属性。一张卡片原则上只会有一条摘录，但是当你合并两张卡片时，被合并卡片中的摘录就会变为一个伪装成评论的摘录，不过在 OhMyMN 中，仍旧认为此为摘录。评论有个很难受的地方就是没法直接修改，只能删除后重新添加，所以 OhMyMN 中很多针对摘录的处理都没法处理评论。

卡片/节点、父子卡片、祖先卡片，后代卡片

前面说了卡片就是节点，节点就是卡片，文档中可能哪个顺口就写哪个。学过数据结构就知道有一种结构就是树，MarginNote 的脑图就是树结构的。



父子节点应该好理解，父节点就是与当前节点相连的上一个节点，一个节点只会有一个父节点，子节点可以有很多个。

祖先节点和后代节点是相对的。祖先节点顾名思义就是与当前节点相连或间接相连的前面所有节点，也就包括了父节点的父节点等。后代节点也是如此，与当前节点相连或间接相连的后面所有节点，包括了子节点的子节点等。

手型工具、文本选择工具、矩形选择工具、多边形选择工具

就和 PhotoShop 的选区工具一样，MarginNote 也有很多选区工具，让你对文档中的某块区域或文字进行选择并摘录。除了手型工具外，其他几个选择工具会在选择后自动摘录。

我通常是使用手型工具，拖拽选区进入脑图中自动会摘录并创建卡片，这样也可以控制卡片位置。

使用其他三种选择工具时会开启 `自动添加到脑图` 的功能，自动插入位置设置为 `选中位置`，这样就可以快速批量进行摘录，形成脑图。

标题链接、合并标题

标题链接非常好用，可以在文档中高亮卡片的标题，并链接到这张卡片。一张卡片可以通过 `;` 或者 `；` 来设置多个标题，每个标题都会链接到这张卡片。OhMyMN 一个创造性突破就是可以自动生成标题，并且合并已有的标题，让这个过程变得自动化。

拖拽选择区进入脑图

在 MarginNote 的首页设置中，有一个 `拖拽选择区进入脑图` 的选项，当你将选区拖到脑图中，在已有卡片上松手，可以有两个效果

1. 添加成为子节点
2. 合并入

两个选项各有优劣，`添加成为子节点` 可以方便摘录，整理脑图结构。配合 AutoStyle 来自动将同层级卡片设置相同颜色。而 `合并入` 在 OhMyMN 中更是重要，配合 AutoTitle，AutoDef 等可以

生成标题的模块，同样可以将新的摘录转为标题继续合并，从而利用标题链接。

那你可能会好奇有没有可能两者兼顾，其实是有的，当你拖拽选区到文档中已经摘录的选区中，就算你设置的是 `添加成为子节点` ，也会直接合并。

配置管理

配置结构

> **TIP**
>
> 笔记本配置和文档配置都会有注释说明，没有说明的情况下都是全局配置。
>
> - 【仅当前笔记本】
> - 【仅当前文档】

- 全局配置 x 4：可可以为不同笔记本选择不同的全局配置。多套全局配置用于多个不同的学习场景。比如 4 个科目，那么刚好就可以使用 4 套配置文件。如果你觉得 4 套不够，那么我认为你应该尽可能的让他们兼容，找出不同学习场景的最大公约数，兼顾灵活和方便。
  - 全局配置中又包含了所有模块的配置。
- 文档配置：每个文档不一样。
- 笔记本配置：每个笔记本不一样。

初始化

如果你觉得初始配置不符合你的需求，不同笔记本，文档一一修改太过繁琐，放心，我肯定是想到了。你可能已经发现了 `OhMyMN-选择配置文件` 中最后一项叫做 `初始化`，当你在这个配置文件中修改时，它会同步到所有文档，笔记本。不过要注意，`OhMyMN-选择配置文件`、`OhMyMN-模块快捷开关` 这两个选项不会参与同步。

> **更新**
>
> v4.1.1 改进，会作为新文档和新笔记本的默认配置。

~~对于文档配置和笔记本配置，初始化只会作用于已经打开过的文档或笔记本。~~

导出、导入

你还可以将所有的配置导出到脑图卡片中，这样就可以随着 MarginNote 一起使用 iCloud 同步，也可以通过导出笔记本来分享配置文件。更多内容可以继续阅读 MagicAction for Card-配置管理。

重置、同步多窗口的配置

OhMyMN 在多窗口的情况下修改配置不会立刻同步过去，你可以通过 MagicAction for Card-配置管理-同步多窗口的配置 来强制同步。

使用 MagicAction for Card-配置管理-重置配置 来重置配置。在停用或者卸载插件时也会提供选项来重置配置。

将自定义设置写入卡片中

基于 MarginNote 的脑图，OhMyMN 还可以将自定义的设置写入脑图卡片中，轻松实现排列组合，更多内容可以继续阅读 自定义输入格式。

正则表达式

在 OhMyMN 中，正则表达式无处不在。所谓正则表达式，简单点说，就是用来匹配某种特定格式字符串的一种表达式。

一般的搜索就是，你输入单词 `baby`，那就会搜出所有的 `baby`，其中也会包括 `angelababy`。其实我只想搜索单独的 `baby`，怎么办呢。单词与单词之间肯定是非单词的字符，所以我们可以用正则表达式 `\bbaby\b`，`\b` 表示所在位置的一侧为单词字符，另一侧为非单词字符。

补充知识

1. 不同的编程语言以及不同的浏览器，对正则支持情况都有所不同，MarginNote 使用的是 Safari 的 JavaScriptCore 引擎。很多特性都不支持，比如向后断言 `(?<=y)x`，向后否定断言 `(?<!y)x`。

2. JavaScript 语言中，正则有固定的写法，比如 `/\bbaby\b/g`，用两个 `/` 来包裹。第二个斜杠后面可以添加标志，用来改变匹配模式，下面这 5 个比较常用：

   - `/xxx/g` 表示全局匹配，在 Replace() 函数 中会替换掉所有匹配到的字符串，否则只会替换第一个。
   - `/xxx/i` 表示忽略大小写。
   - `/xxx/s` 表示 `.` 可以匹配换行符，这里要注意，默认情况下 `.` 无法匹配换行符的。
   - `/xxx/m` 表示多行匹配，`^$` 将不再只匹配整个字符串的开头和结尾，而是每一行的开头和结尾。
   - `/xxx/u` 表示启用 Unicode 匹配，用来匹配中文或者 Emoji 时非常有用，自行查看 具体属性。

3. 可以使用 Regex-Vis 或 iHateRegex 进行测试和可视化。

4. 可以使用 Regex Learn 通过答题来入门正则表达式，我相信会对你有所帮助。

元字符

元字符是在正则表达式中具有特殊含义的符号或字符，正则表达式本质上就是通过元字符实现字符串精准匹配的。接下来，我讲的所有符号都是元字符，下面这些是简单常用的元字符。

符号	说明
`.`	匹配除换行符以外的任意字符
`\w`	匹配字母或数字或下划线
`\s`	匹配任意的空白符
`\d`	匹配数字
`\b`	匹配单词的开始或结束
`^`	匹配字符串的开始
`$`	匹配字符串的结束

`^`说清楚点就是匹配每一行的开始位置，`$`匹配的是每一行的结尾位置。只要有了`^`，那就只会匹配每一行开头的字符串，而不会匹配每一行中间的，而`$`就是匹配每一行结尾的字符串，两个结合到一起，常用于单行字符串的匹配。

反义

反义一般用上面元字符的大写表示，比如`\d`匹配任意数字，而`\D`匹配除数字外的所有字符，其他的也一样。使用`^`来匹配除方框里给出的字符之外的所有字符。

符号/语法	说明
`\W`	匹配任意不是字母，数字，下划线，汉字的字符
`\S`	匹配任意不是空白符的字符
`\D`	匹配任意非数字的字符
`\B`	匹配不是单词开头或结束的位置
`[^x]`	匹配除了 x 以外的任意字符
`[^aeiou]`	匹配除了 aeiou 这几个字母以外的任意字符

限定符

限定符是跟在其他元字符后面的，用于限定元字符匹配字符的重复次数。

符号/语法	说明
`*`	重复零次或更多次
`+`	重复一次或更多次
`?`	重复零次或一次
`{n}`	重复n次
`{n,}`	重复n次或更多次
`{n,m}`	重复n到m次

这部分可能不太好理解，我来举几个例子

- `\d+` ：作用是匹配由数字构成的字符串 `\d` 是匹配数字，`+` 相当于无数个 `\d`，数量取决于什么时候遇到非数字，必须连续。
- `\d{1,}`：作用和上面一毛一样，也是匹配由数字构成的字符串，只是可以自定义最少有几位，比如 `\d{3,}` 表示这个数字至少有三位。
- `*` 相当于 `{0,}` ，`+` 相当于 `{1,}` ，`?` 相当于 `{0,1}`，后三个只是自定义程度更高，前三个使用更方便。

字符转义

当你想搜索元字符本身怎么办，那就在前面加一个 `\` ，比如说想搜索 `.` ，就需要用 `\.` 。之前说了正则表达式里所有的特殊符号都是元字符，都需要转义。

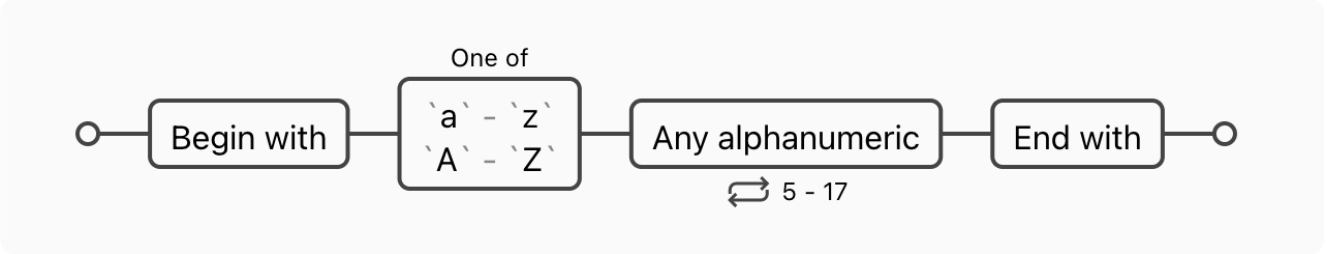
字符类

之前说的 `\w` ，`\d` ，`\s` 这些只能匹配任意的字母数字，而不能匹配特定的几个字母或者数字，只需要把你想匹配的装到方括号里，就像 `[12345]` 这样，你就能匹配到 12345 中任意一个数字了，同时你也可以用 `[1-5]` 表示。

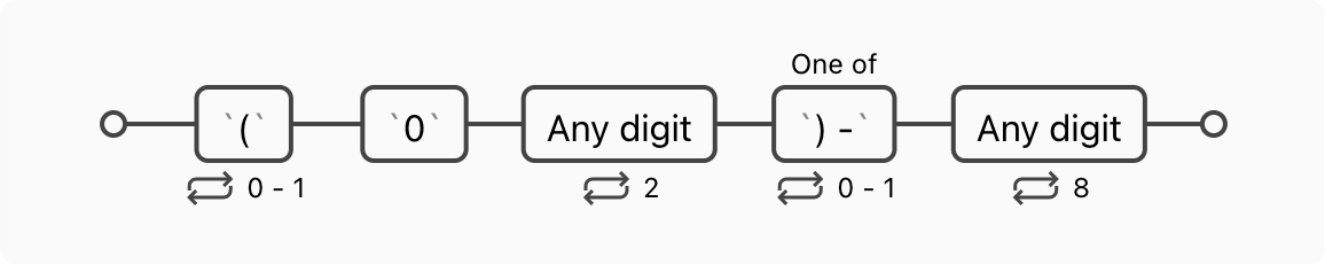
除了数字，其他的字符，字母都可以这样，并且在方括号里不用担心字符转义的问题，`[.*+?$]` 这些都可以直接匹配。但是用于反义的 `^`，我们如果想要匹配它就需要使用 `[\^]` 。

字符、数字、字母都可以放在一起，比如 `[0-9A-Za-z]` ，直接连在一起就行，相当于 `\w` 的效果。

举一个稍微复杂的例子，`^[a-zA-Z]\w{5,17}$`，用正则可视化我们可以看出这是一个校验密码的表达式，以字母开头，长度为6到18位。下面写的意思是再重复4到16次。用 `^$` 包在一起就表示是单独的一行。



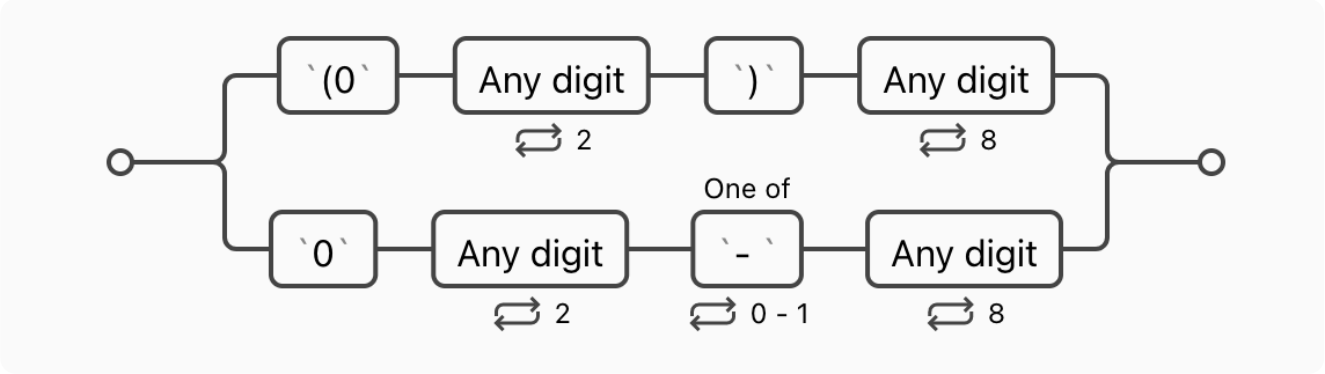
再举一个复杂点的例子，`\(?0\d{2}[) -]?\d{8}`，他其实能匹配四种格式的电话号码，比如 `(010)88886666` 或 `022-22334455` 或 `029 12345678` 或 `02912345678`，他们的特点在于前三位数字，有的是括号包围，有的后面跟着短横，有的后面跟着空格，有的什么都没有，这就用到了字符类，`[) -]` 这里面有是三种字符，包括一个空格。`?` 表示匹配0次或1次，所以一共四种情况。



分支条件

刚才举的第二个例子，可以匹配三种格式的电话号码，但你认真思考一下就会发现，它还会匹配 `010)12345678` 或者 `(022-87654321` 这些错误的格式。因为 `?` 不会进行判断，前面的字符存不存在都可以。但是我们更多的时候需要进行判断，不存在是什么格式，存在是什么格式。

拿这个例子来说，如果存在 `(`，那后面也必须是 `)`，要实现这个我们需要要用分支条件 `|`，这个符号就相当于取并集，两个条件满足一个即可。改写上面的表达式 `\(0\d{2}\)\d{8}|0\d{2}[- ]?\d{8}`。

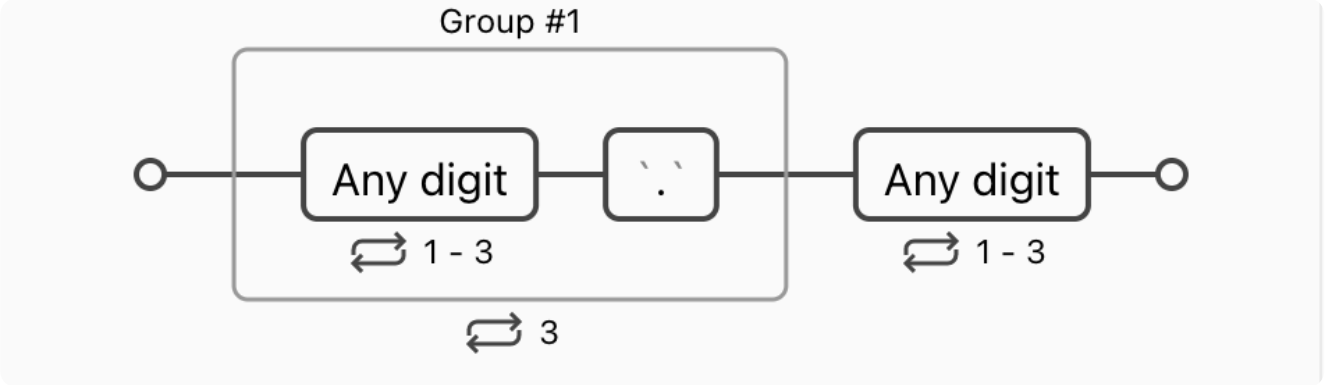


分成了两个分支，一个是有括号的，一个是没括号的。而没有括号的中间可以有一条短横，也可以没有。值得注意的是，分支条件优先匹配左边的条件，只要满足左边的条件，就不会看右边的条件了。

分组

之前说的在元字符后面加重复限定符就可以重复匹配这个字符，但是如果想重复匹配一个比较复杂的表达式呢，就需要把这个表达式放在括号 `()` 里面。

比如常用的 IP 地址匹配的表达式 `(\d{1,3}\.){3}\d{1,3}`，每 3 个数字（最多 3 个）一段，共四段，中间用 `.` 连接。可以看做三段 3 个数字（最多 3 个）加 1 个点，最后一段为 3 个数字（最多 3 个）。

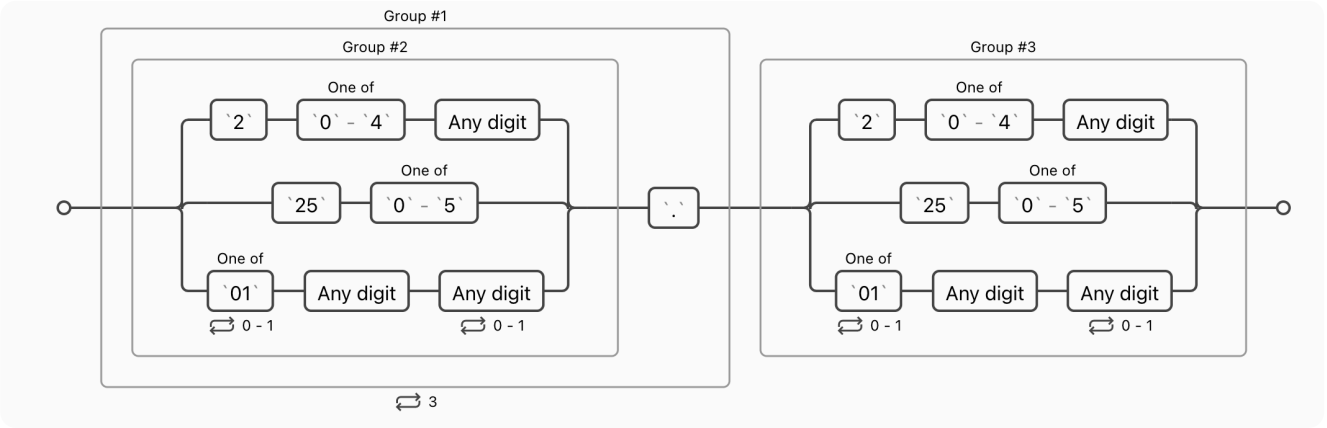


当然，这个表达式只是匹配个形式，IP 每一段有大小限制，不能大于 255，很遗憾正则表达式不能判断数字大小，所以我们只能把 3 个数字单独分开看，分为 3 个分支。

- 第一位为 `2`，第二位为 `0-4`，第三位为 `任意数字`。
- 前两位为 `25`，第三位 `0-5`。
- 第一位为 `0或1或者为空`，第二位为 `任意数字`，第三位为 `空或任意数字`。

然后将这3个数字的表达式分为一组即可。IP 地址完整匹配表达式

```
((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)
```



1. 捕获组

默认 `()` 就是捕获组，会将括号里匹配到的内容保存到内存中，你可以使用 `\1` 来引用它，当然，只能在括号后引用，这就是 后向引用 。

捕获组在 Replace ()函数 中非常有用。可以使用 `$1` 来引用捕获的内容。

2. 非捕获组

如果只是想分组，其实不需要用捕获组，可以使用 `(?:)` 来分组。

在 Split() 函数 中，如果使用捕获组，捕获的内容包括在结果中，会增加不确定性，一般就是用的非捕获组。

贪婪与懒惰

这其实是很多人不太明白，但却非常重要的知识点。

什么是贪婪，举个例子，如果一个字符串 `1010000000001` ，我们需要匹配 `101` 一般人就会使用 `1\d+1` ，但是这样你会匹配到 `1010000000001` 这整个字符串，这就是贪婪匹配，他会尽可能多的匹配字符。

而如果想匹配到 `101` ，我们就需要使用 `1\d+?1` ，在重复限定符后加一个 `?` ，就变成了懒惰匹配，会尽可能少的匹配。尽可能少重复，遇到第一个满足条件的就停止匹配。`?` 本身就是重复限定符，表示重复 0 次或 1 次，所以也有 `.??` 这种形式，至于这有什么作用，我也不知道。

零宽断言

可谓是正则里面最厉害的，可惜的是 MarginNote 的 JS 引擎对它的支持度不高。它不匹配任何字符串，只匹配一个位置，比如 `\b` `^` `$` 这些都是断言。

零宽好理解，匹配的只是一个位置，本身是没有宽度的。而断言，在调试代码中很常用，表示我断定这个条件是满足的，如果不满足就是出 bug 了。至于在这里嘛，可能就是断定这个位置的意思。

零宽断言有很多中别名，比如 `环视`，分为了 `肯定逆序环视`、`否定逆序环视`、`肯定顺序环视`、`否定顺序环视`。

这里我一般常用：

- 向前断言 `x(?=y)`，给出了一个位置，表示 y 的前面，所以我们匹配的就是在 y 前面的 x。也就说匹配的 x 必须有 y 跟在后面。注意断言只是一个位置，他不会被包含着匹配结果中。
- 向前否定断言 `x(?!y)` 顾名思义，给出的位置是一个字符的前面，但不是 y，匹配的是没有 y 跟在后面的 x。
- ~~向后断言 `(?<=y)x` 匹配的是有 y 在前面的 x。~~ `不支持`
- ~~向后否定断言 `(?<!y)x` 匹配的是没有 y 在前面的 x。~~ `不支持`

参考

1. 正则表达式30分钟入门教程
2. 正则表达式——MDN

Replace() 函数

为了使 OhMyMN 更加自由，更加强大，OhMyMN 中很多自定义都采用 Replace() 函数作为驱动。这使得你几乎可以对摘录进行任何处理，不过为了避免更多的导致插件崩溃的不确定因素，OhMyMN 限制了将函数作为参数，以及仅支持正则表达式。

替换

Replace() 函数的作用其实就是将匹配到的内容替换为给的文字，然后返回替换后的所有内容。

输入格式

```js
(/regex/, "newSubStr")
```

- `regex` 是 正则表达式，用来匹配需要替换的内容。
- `newSubStr` 为普通字符串，需要使用双引号包裹，比如 `"xxx"`，表示想要替换成的内容。 在 `newSubStr` 中，可以使用一些变量，来引用匹配到的内容：

1. `$&` 表示匹配的字符串。
2. `$\`` 表示匹配的字符串前面的内容。
3. `$'` 表示匹配的字符串后面的内容。
4. `$n` 如果你在 `regex` 中使用了捕获组，你就可以用 `$+数字` 来引用你捕获的内容。
5. 在 v4 版本中，某些情况下还可以使用模版变量，点击查看 模版语法。

提取

OhMyMN 中所有的提取操作都是直接将 `newSubStr` 作为了返回值。相当于只要正则匹配到了就返回 `newSubStr`，而使用捕获组，还可以将其捕获并返回。能实现这个效果，其实还需要用到 Match() 函数。当然，这个你不需要了解。只需要按照前面替换的语法一样使用即可，只是最后会返回 `newSubStr` 而不是整个字符串。

Split() 函数

Split() 函数目前只在 Another AutoDef 中使用，用于将摘录的内容分割成标题和摘录两部分。

只需要提供一个 正则表达式，作为分割点，将字符串分割成两部分。

这里要单独提一下是因为 split 有个特性，如果你正则表达式中使用了 捕获组，就会把捕获的内容放在分割后的结果中，导致分成了三个部分。 如果你必须要分组，可以使用 非捕获组。

其实捕获组也有个好处，就是当你找不到一个明确的分割点，比如选择题，你想把题目作为标题，选项作为摘录。这时候就必须用到捕获组，并且捕获的内容就是所有选项。

`[被定义项，定义项]` ⇒ `[被定义项，捕获组]`

另外，通过捕获组还可以对被定义项或者定义项进行限制。

模版语法

OhMyMN 在 v4 版本中加入了 Mustache 模版引擎。如果你熟悉 Anki 模版，就会发现其实 Anki 用的也是这个。标志就是两个大胡子 `{{}}`。相较于之前 AutoComplete 或者 CopySearch 我自己实现的模版~~引擎~~，Mustache 更加强大，带来了数组，对象，函数的支持。

在哪里可以使用

1. AutoComplete
2. CopySearch
3. ~~Export to Anki~~
4. ~~Export to Flomo~~

其实不仅如此，OhMyMN v4 将模版与 Replace() 函数相融合，你可以在 `newSubStr` 中使用模版。不过并不是任何地方使用都有意义，所以我限制了使用区域。

1. Another AutoDef：提取或设置标题。
2. AutoTag：提取或设置标签。
3. AutoReplace：修改摘录内容。
4. AutoComment：提取或设置评论。
5. MagicAction for Card
   - 重命名标题
   - 提取标题
   - 添加标签
   - 添加评论
   - 替换摘录文字

除了 AutoComplete 的数据来自于词典，其他地方均使用当前卡片或摘录笔记的数据，详细内容请查看 模版变量。

补充知识

1. 由于 Mustache 最初是用在 HTML 模版上，所以默认情况下解析出来的文本都会进行 HTML 转义。但我们这里用不着，所以我修改了下源码，改为了默认不转义，如果需要转义，可以使

用 `{{{titles}}}` 或 `{{& titles}}`。

2. 直接使用一个数组变量，数组元素会通过 `;` 合并成一个字符串。这和 Mustache 不一样。

变量

所谓变量嘛, 就是 key-value。输入 key，解析出来就成了 value。

要使用一个变量，直接 `{{titles}}` 即可，你就可以获取到这张卡片的标题。如果你变量名写错了，或者这个变量没有值，就整体为空。

对象

对象可以有很多变量, 比如

```
obj = {
  key1: "value1",
  key2: "value2",
}
```

你可以使用

```js
{{#obj}} {{key1}} {{key2}} {{/obj}}
> value1 value2
```

条件

可以把变量作为一个判断条件，像下面这样，包裹住一些文字或者变量，如果这个变量为空，那么整体就为空。有点类似 HTML 标签 `<a></a>`，以反斜杠结束。

```js
{{#titles}}有标题{{/titles}}
{{#titles}}有标题 {{id}}{{/titles}}
```

还可以当某个变量为空时才显示里面的内容，可以这样，把 `#` 换为 `^`。

```js
{{^titles}}没有标题{{/titles}}
{{^titles}}没有标题 {{id}}{{/titles}}
```

这样其实也就实现了 `if-else` 的效果。

数组/列表

> **TIP**
>
> 在代码世界里，一个数组或者列表都是从 0 开始数的。

这是我换成 Mustache 的最大原因。以前只能用 `title_1`，`tag_1` 表示第一个标题，第一个标签，现在可以使用 `titles.0`，`tags.0` 来表示。当然也可以 `titles.1` `titles.2`。

默认情况下，一个数组 `{ titles: ["aaa", "bbb", "ccc", "ddd"] }`，如果直接这样使用，输出的结果会用 `;` 隔开。

```js
{{titles}}
> aaa; bbb; ccc; ddd
```

通常是像下面这样给循环渲染出来，用 `{{ . }}` 来表示数组内的每一项：

```js
{{#titles}}{{.}}, {{/titles}}
> aaa, bbb, ccc, ddd,
```

其实这不是很智能，最后会多出一个 `,`。这时候我们可以用自定义的 join 函数，`{{#join}}{{titles}}, {{/join}}`

更神奇的是如果一个数组是对象数组，比如 模版变量 里的 `children` 变量，他是一个对象数组，每一个元素都有 `titles` 属性，类似

```js
{
  children: [
    {
      titles: ["aaa", "bbb"]
    },
    {
      titles: ["ccc", "ddd"]
    }
  ]
}
```

那么我们可以这样写：

```js
{{#children}}{{titles}}\n{{/children}}
>   aaa; bbb
    ccc; ddd
```

函数

函数也非常有用，在 模版变量 中，提供了很多函数。就已 `nohl` 举例吧。

默认情况下，如果你在 MarginNote 中划了重点，插件获取的重点就会变成 `**重点**`，这其实是 Markdown 里的语法。如果是直接在 Markdown 中粘贴就还好，但是粘贴到其他不支持 Markdown 的软件中，就比较不舒服了。

之前的解决办法是用两个变量，一个表示有 `**`，一个表示没有 `**`。现在你可以这样：

> **更新**
>
> v4.2.0 使用新的函数语法，支持函数参数。

```js
// 新写法
{{ excerpts.text.0 | nohl}}
// 旧写法
{{ #nohl }}{{ excerpts.text.0 }}{{ /nohl }}
```

就可以把 `excerpts` 里的所有 `**` 给删除。当然，你还可以用 `cloze` 将 `**重点**` 全部换成 `{{c1:重点}}`，快去试试吧。

模版变量

如果你已经学习了 模版语法，那你就可以随意使用下面这些变量了。

> **注意**
>
> Metadata 插件暂未发布，目前不开放下载。

> **更新**
>
> v4.2.0，使用 `date` 代替 `time` 变量。

变量

脑图卡片

变量名	类型	说明
`id`	字符串	Note ID
`url.pure`	字符串	Note URL
`url.md`	字符串	Note URL，添加 `[]()`
`url.html`	字符串	Note URL，添加 `<a>`
`page.start`	字符串	笔记在文档中的开始页码
`page.end`	字符串	笔记在文档中的结束页码
`page.real.start`	字符串	需要 Metadata 计算了偏移量后的开始页码。
`page.real.end`	字符串	需要 Metadata 计算了偏移量后的结束页码。
~~`time.creat`~~ `date.create`	字符串	笔记创建的时间
~~`time.modify`~~ `date.modify`	字符串	笔记最后修改的时间
~~`time.now`~~ `date.now`	字符串	当前时间
`allTextPic.text`	字符串	卡片中的所有文字，包括 Markdown 插件中的文字。

变量名	类型	说明
`allTextPic.ocr`	字符串	卡片中所有文字，图片摘录会自动 OCR，不包括图片评论。
`allTextPic.md`	字符串	卡片中的所有文字和图片，图片采用 base64 编码，添加 `![]()`
`allTextPic.html`	字符串	卡片中的所有文字和图片，图片采用 base64 编码，添加 `<img src>`
`tags`	字符串数组	所有标签，没有 #
`titles`	字符串数组	所有标题
`excerpts.text`	字符串数组	所有文字摘录
`excerpts.ocr`	字符串数组	所有摘录，图片摘录被 OCR 为文字
`excerpts.html`	字符串数组	所有摘录，图片摘录使用 base64 编码，添加 HTML 标签 `<img>`
`excerpts.md`	字符串数组	所有摘录，图片摘录使用 base64 编码，添加 `![]()`
`comments.text`	字符串数组	所有文字评论。包括 Markdown 插件中的文字。
`comments.html`	字符串数组	所有评论，包括图片，使用 base64 编码，添加 HTML 标签 `<img>`
`comments.md`	字符串数组	所有评论，包括图片，使用 base64 编码，添加 `![]()`

文档

当前卡片所属的文档

变量名	类型	说明
`doc.title`	字符串	文档标题
`doc.md5`	字符串	文档 md5
`doc.path`	字符串	文件路径

变量名	类型	说明
`doc.url.pure`	字符串	`笔记本里可用` 文档 URL，通过文档中最后一个笔记的链接来间接跳转。
`doc.url.md`	字符串	`笔记本里可用` 文档 URL，添加 `[]()`
`doc.url.html`	字符串	`笔记本里可用` 文档 URL，添加 `<a>`
`doc.reference`	字符串	`需要 Metadata` 参考文献/引用
`doc.citeKey`	字符串	`需要 Metadata` Citation key
`doc.pageOffset`	字符串	`需要 Metadata` 文档的页码偏移量
`doc.metadata`	对象	`需要 Metadata` 从 Zotero 中导入的所有元数据

笔记本

当前卡片所属的笔记本

变量名	类型	说明
`notebook.title`	字符串	笔记本标题
`notebook.id`	字符串	笔记本 ID
`notebook.url.pure`	字符串	笔记本 URL
`notebook.url.md`	字符串	笔记本 URL，添加 `[]()`
`notebook.url.html`	字符串	笔记本 URL，添加 `<a>`

父子卡片

你其实也可以获取到当前卡片的父卡片和子卡片上述所有的信息。

变量名	类型	说明
`parent`	对象	`parent.titles` 表示父卡片的 `titles`
`children`	对象数组	`children.0.titles` 表示第一个子卡片的 `titles`

函数

函数名	说明
`nohl`	删除 `**`，也就是重点
`blod`	将 `**重点**` 修改为 `<b>重点</b>`
`cloze`	将 `**重点**` 修改为 `{{c1::重点}}`
`clozeSync`	将 `**重点**` 修改为 `{{c1::重点}}`，同时显示答案
`upper`	全部转为大写
`lower`	全部转为小写
`join`	使用指定的前后修饰来合并字符串数组为一个新的字符串
`fmt`	格式化 `date` 变量

更新

v4.2.0 使用新的函数语法，新增 **`fmt`** 函数，比如 **`{{ date.now | fmt: "YYYY-mm-dd HH:MM" }}`**。

join 函数有两个参数，用来自定义前后修饰，两个参数之间用 `:` 隔开，比如：

```md
{{ titles | join: "" : "; " }}
```

titles 是一个数组，比如 `["标题1", "标题2"]`，那么上面的代码会输出 `标题1; 标题2`。最后一个元素不会添加后缀修饰，所以不会输出 `标题1; 标题2;`。

前缀修饰还可以 自动编号，使用 `$['1']`，比如

```md
{{ titles | join: "$['1']. " : "; " }}
```

输出 `1. 标题; 2. 标题`。

自定义输入格式

OhMyMN 中有大量的自定义，方便你定制。主要是三种格式：

1. 正则表达式
2. Replace() 函数
3. 模版

另外，由于正则表达式或者 Replace() 函数可以同时设置多个，也会涉及到执行顺序等问题，在 OhMyMN 的输入框中输入会比较麻烦，所以我创造性的将 MarginNote 的脑图卡片作为了自定义的输入框。

当然，我建议不管是自定义什么，都不要在 OhMyMN 的输入框中直接输入，而是其他地方写好了再粘贴进来。因为 OhMyMN 不是实时保存，最后需要敲回车确定后才会保存。

正则表达式

正则表达式 有两个作用：

1. 判断是否满足条件，比如 Another AutoTitle 中用来判断是否可以转为标题。
2. 作为分割点，将一段话分割为多个部分，比如 Another AutoDef 自定义定义联项，将定义分为定义项和被定义项两部分。

有六种输入格式：

1. 字符串 `xxx`，并非只是省略 `//` ，这里无法使用正则里特殊的字符。一般是用来直接匹配某个词。
2. 正则表达式 `/xxx/` 。
3. 正则表达式数组，`[/xxx/, /yyy/]` ，`,` 隔开。`与` 的关系，也就是全部匹配才算匹配成功。
4. 多个正则表达式 `/xxx/; /yyy/` ，`;` 隔开。`或` 的关系，一个匹配就算匹配成功。
5. 多个正则表达式数组，`[/xxx/, /yyy/]; [/xxx/]` ，`;` 隔开。`或` 的关系。建议使用这种写法，可以避免掉大多数解析错误的情况。
6. 正则表达式和正则表达式数组，`[/xxx/, /yyy/]; /xxx/; /yyy/` ，`;` 隔开。`或` 的关系。

如果特殊情况出现，我会在相应地方注明。

Replace() 函数

replace() 函数其实是对应着一种输入格式，`(regex, newSubStr)`，一个正则，一个字符串。比如 `(/regex/, "newSubStr")`。

其作用首先是进行正则判断，看是否满足条件，如果满足，

1. 替换，将正则匹配的部分替换为 `newSubStr`，返回替换后的内容。
2. 提取，返回 `newSubStr`。

其实还有第三个参数，`(regex, newSubStr, fnKey)`，`fnKey` 为整数，比如 `(/xxx/, "yyy", 0)`。只是 `FnKey` 默认为 0，可以不用写，用于一些特殊的设置。用到时自会标注出来。

可以写多个，用 `;` 隔开，比如

- `(/xxx/, "111"); (/yyy/, "222")`
- `(/xxx/, "111", 1); (/yyy/, "222")`

MNLink

所谓 MNLink 就是 `marginnote3app://note/F20F324D-61B3-4CA9-A64C-0C92645A1E33`，也就是笔记的链接。可以在这个地方获取到

上面说了，不管是 `/xxx/` 还是 `(/xxx/, "yyy")` 都可以使用 `;` 隔开设置多个。我创造性的将脑图卡片当作了编辑器，这样就可以轻松停用或启用其中每一个正则。也可以修改他们的先后顺序。

只需要创建一张卡片作为读取点，复制它的 MNLink 填入到对应的输入框中（支持上面这两种格式的都支持 MNLink）。然后实际读取的是子卡片的第一条评论，会把所有子卡片的评论通过 `;`

合并起来构造最终的输入。

> **注意**
>
> 如果你修改了卡片上的内容，OhMyMN 无法自动读取，你必须在填入自定义的地方手动敲一下回车来更新配置，同时会检查是否输入正确。

> **表示禁用的颜色是**
>
> 第 4 排第 2 个

在当前版本中，MagicAction 同样支持 MNLink，甚至不要求必须符合上面这两种格式。

模版

这个就没什么好说的了，就是 `{{变量}}`，看 模版语法 以及 模版变量。

自动编号

在 OhMyMN 里面随处可见自动编号的使用，比如给卡片编号，甚至是分层编号。合并卡片内的文字时给每一个评论添加编号。CopySearch 中搜索或复制多张卡片的内容时，也可以为每张卡片的内容进行编号。

在使用中会出现三种不同的写法

1. `%["1"]`，在 重命名标题 中为选中卡片依次进行编号。另外，在 合并卡片内文字 和 CopySeach 设置多张卡片的前后修饰都会用到。
2. `$["1"]`，本质上和 `%["1"]` 相同，用于 模版函数 里的 `join` 函数。至于为什么要设置不同的前缀，因为在重命名标题中可能会同时使用到这两个。
3. `#["1"]`，在 重命名标题 中为选中卡片的子卡片进行分层编号。

`%["1"]` 和 `$["1"]`

这两个的使用方法完全相同，就以 `%["1"]` 举例。

起始值和字符类型

首先，这个 `"1"` 里面的 1 是可以变的，可以是 2，也可是 100，甚至还可以是 `001`，给数字补 0，从而所有编号都能达到相同的宽度，这就是起始值。

不光如此，还可以是不同的编号字符：

- ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟㊱㊲㊳㊴㊵㊶㊷㊸㊹㊺㊻㊼㊽㊾㊿
- ❶❷❸❹❺❻❼❽❾❿⓫⓬⓭⓮⓯⓰⓱⓲⓳⓴
- `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
- `abcdefghijklmnopqrstuvwxyz`
- `壹贰叁肆伍陆柒捌玖拾`
- `一二三四五六七八九十`
- `ⅠⅡⅢⅣⅤⅥⅦⅧⅨⅩ`
- `ⅰⅱⅲⅳⅴⅵⅶⅷⅸⅹ`

所以说 `"1"` 既决定了编号的字符类型，也决定了起始值。使用的时候直接复制替换即可。不过要注意，如果编号不够，则会从头开始。

步长

`%["1"]` 的完整写法是 `%["1",1]`，后面这个 `1` 没有引号，代表着步长。步长就是间隔，步长 1 为默认值，所以可以省略。 如果你想 `1 11 21 31`，则可以写成 `%["1",10]`。

自定义编号字符

如果你想使用一套指定的编号或者前缀，则可以使用 `%["aaa", "bbb", "ccc", "ddd", "eee", "fff"]`。

#["1"]

目前就只有 重命名标题 的分层编号在使用。

`#["1"]` 的完整写法为 `#["1","1","1","1",[".",4, false]]`

表示第一层从 `1` 开始，第二层从 `1` 开始，第三和第四层也是从 `1` 开始。如果之后还有层级，默认使用提供的最后一个起始值，也就是 `1`。你脑图有多少层，你这里就可以设置多少层，也就是可以为每一层单独设置编号的字符。它支持上述所有的编号字符，但不自持设置步长，不支持自定义编号字符。

选项数组

最后的 `['.',4,false]` 数组中的三个元素分别表示 `连接符号`，`最大编号层级`，`是否只显示当前层的编号`。均可省略，没有顺序要求。默认值为 `[".",999,false]`。

- `连接符号` 为字符串 `"."`, 也就是 `1.1.1`。
- `最大编号层级` 为数字 `4`, 如果整个脑图有 10 层，你可以通过这个设置，来限定 4 层之后就不再编号。通常只需要编号三层即可。
- `是否只显示当前层的编号` 为布尔值 `false`，布尔值就是 `true/false`。将其设置为 `true` 后，每一层只会显示一个编号，而不会加上其父卡片的编号。 `1.1.2` → `2`

OhMyMN

这个板块中的选项主要是设置这个面板的一些属性，以及控制其他模块的。

选择全局配置

> **当前笔记本有效**
>
> 只切换全局配置，不包括笔记本配置和文档配置

配置管理 已经说过了 OhMyMN 的配置结构，这里不过多赘述。

模块快捷开关

停用或启用其他模块的，停用后不再显示其选项菜单。属于该模块的 Action 也不会显示在 MagicAction 中。

面板显示位置

> **更新**
>
> v4.0.6 新增：更多位置，支持自动跟随文档宽度调整。

- 文档内侧：默认，文档内侧显示，支持自动跟随文档宽度调整。
- 文档脑图中间：文档脑图中间显示，支持自动跟随文档宽度调整。
- 脑图内侧：脑图内侧显示，支持自动跟随文档宽度调整。
- 靠左
- 居中
- 靠右

面板显示高度

通常 12.9 寸的 iPad 可以选择 `高点`，11 寸一般还是用 `标准`，这样最下面的输入框不会被键盘挡住。

面板开关控制

> **更新**
>
> v4.1.0 移除 `双击面板关闭面板`，误触太严重。

- 双击图标打开面板：双击笑脸 😀 开启面板，这样可以避免误触。
- 动作执行完关闭面板：MagicAction 中的动作执行完就会关闭自动面板。

拖拽合并生成标题

> **更新**
>
> v4.0.10 优化了拖拽合并生成标题的逻辑。

Another AutoTitle, Another AutoDef, AutoComplete 都可以将摘录转为标题。

通过 拖拽选择区 合并进卡片中，有以下几种处理方式：

- 始终不生成标题。
- 满足条件时生成标题：通过上述模块来自动生成标题。
- 始终转标题：即使上面几个模块没有生成标题，也会直接转为标题。

> 已经有标题

如果会生成标题，但是卡片已经有标题了，有以下几种处理方式：

- 不转为标题
- 合并标题：使用 `;` 合并标题，可用于标题链接。
- 覆盖标题

> 拖入的摘录将

通过上述操作将摘录转为标题后，拖入的摘录会面临两个选择：

- 立即删除
- 等会删除：在下次摘录或者退出笔记本或者退出 MarginNote，或者 MarginNote 进入后台时进行删除。这么处理的原因是为了能够有调整摘录选区的机会。

保持屏幕常亮

> **更新**
>
> v4.0.4 删除：拆分为单独的插件，点击下载：Keep Screen On

双击打开链接

> **更新**
>
> v4.2.0 新增

双击控制面板里的链接才打开。防止误触。

使用 Markdown 评论 Only MNE

> **更新**
>
> v4.2.1 新增

所有通过 OhMyMN 生成的评论将直接使用 Markdown 语法。

锁定摘录文字

开启后，在滑动文档的时候如果你手滑修改了文档中的摘录选区，OhMyMN 会帮你还原摘录中的文字。在创建摘录的时候，只要摘录菜单不消失，就可以一直修改。一旦菜单消失再修改的话就

会被锁定。这个功能还可以做到，先摘录很长一段话，然后将选区缩短到一个词。

自动备份配置

更新

v4.0.3 改进：直接设置备份卡片的链接。

输入

卡片链接，例如 `marginnote3app://note/F20F324D-61B3-4CA9-A64C-0C92645A1E33`

自动将配置信息保存到 MarginNote 的脑图卡片中。同样请确保该卡片有子卡片，因为配置信息是写到子卡片的。子卡片越多越好，因为这个配置信息会不断膨胀，避免超过一张卡片的最大字数。

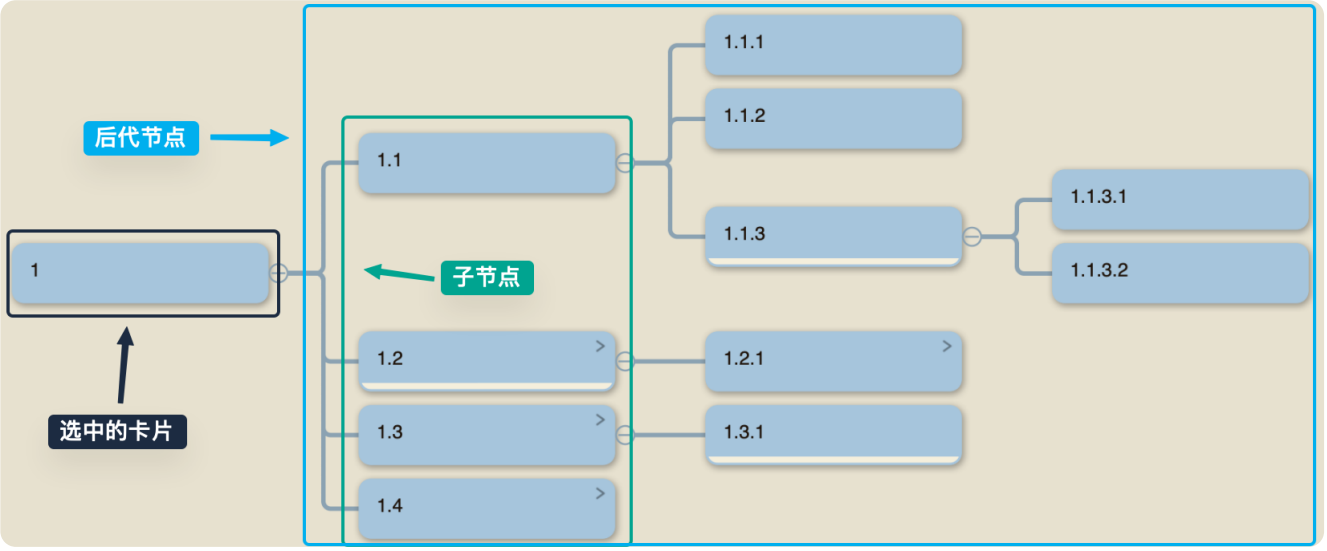
MagicAction for Card

一些与卡片相关的动作，其实就是按钮，先选中卡片（可以多选），再点击按钮来手动执行。部分动作来自于其他模块，需要启用后对应模块后才会显示动作，这些动作和模块相绑定，使用相同的配置，模块启用后才会在此显示相应动作。

智能选择

当你选中一个有子节点的卡片或者选中多个同层级并且均有子节点的卡片，此时 OhMyMN 会弹出菜单，询问是否选中子节点或者后代节点，这可以用于快速批量选中。

父子节点的 相关概念



动作

配置管理

更新

v4.0.6 新增：支持导出或导入单个模块的配置。支持手动重置配置和同步多窗口的配置。

注意

请先认真阅读 OhMyMN 的 配置结构 。

1. 写入配置：将配置写入选中卡片的子卡片中。请确保选中卡片有子卡片，子卡片越多越好，因为这个配置信息会不断膨胀，避免超过一张卡片的最大字数。
    - 所有配置
    - 全局配置 12345：全局配置还可以继续单独导出某一模块的配置。
    - 文档配置
    - 笔记本配置
2. 读取配置：读取卡片中的配置。可以只读取某一模块的配置，并写入到指定的全局配置中。
    - 所有配置 → 全局配置 → 模块配置
3. 重置配置：重置为默认配置。
4. 同步多窗口的配置：OhMyMN 在多窗口的情况下修改配置不会立刻同步过去，你可以该选项来强制同步。

筛选卡片

> **输入格式**
>
> 正则表达式——匹配

可以单独筛选标题，摘录，评论，标签。筛选后还可以继续使用其他动作。

虽然 OhMyMN 的选中和 MarginNote 的选中不一样，无法使用删除，剪切的功能。不过办法总是有的，你可以把选中的卡片添加一个特殊的标签，再通过标签筛选，即可快速重新选中。最后再统一删除这个标签即可。

同样，OhMyMN 无法通过颜色筛选，你可以使用 MarginNote 筛选后添加特殊的标签，再通过 OhMyMN 筛选并处理。最后再统一删除这个标签即可。

合并卡片

MarginNote 自带的合并功能实在鸡肋，不但无法合并标题，还会把标题作为评论合并进卡片。该功能解决了三个问题：

1. 合并标题。
2. 其它卡片标题不作为评论。
3. 合并标签。

重命名标题

OhMyMN 中最强大的功能前三吧，用于批量对卡片重命名，编号，分层编号。

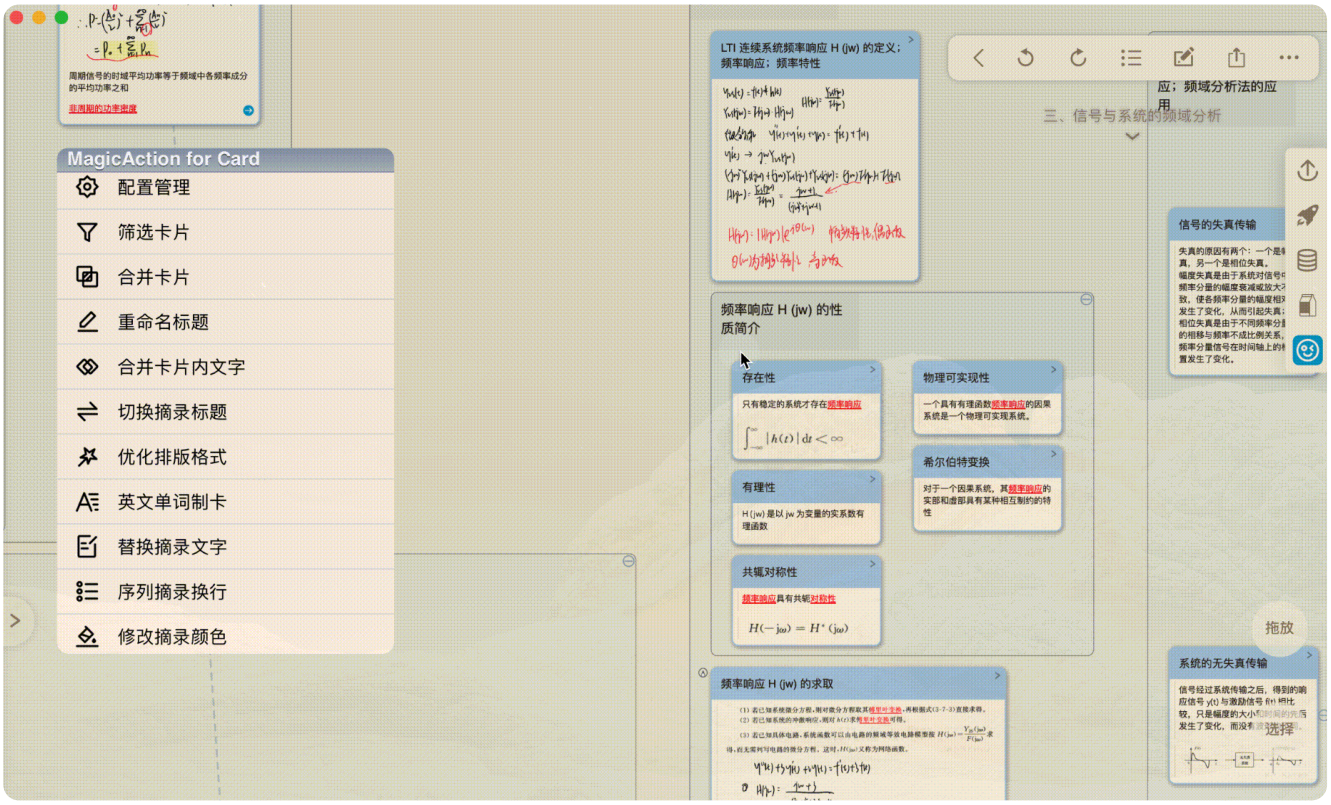
> 输入格式
>
> Replace() 函数格式——替换

由于大多数情况下是匹配整个标题，所以默认的正则就是 `/^.*$/`，你只需要输入新的标题即可，还可以用 `$&` 来引用当前标题。

1. 编号

> 输入
>
> `%["1"] $&`

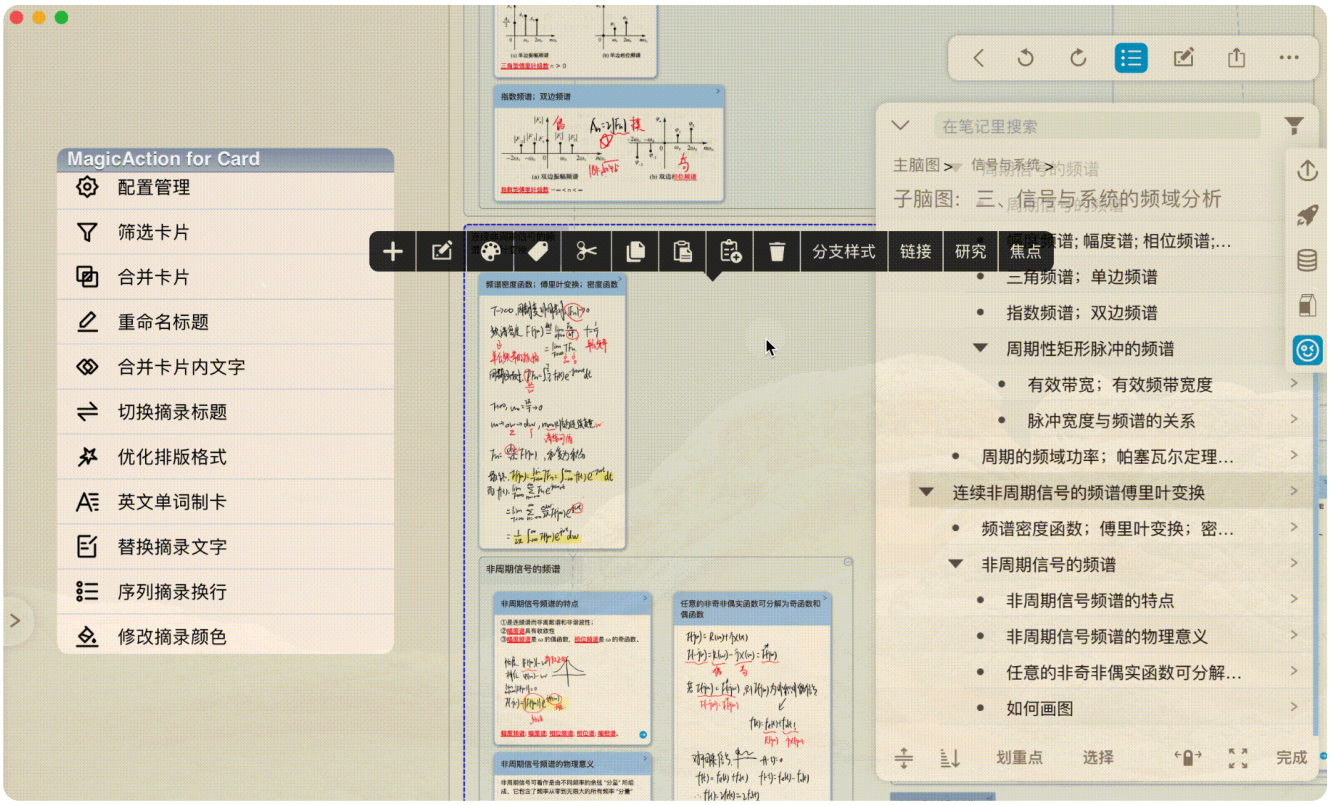
这里用到了一个魔法变量，`%["1"]`，每次调用时都会递增,请自行查看 语法。



2. 分层编号

> 输入
>
> `#["1"] $&`
>
> 分层编号是对其所有后代节点进行编号，自身不会编号。使用时请确保该卡片有子节点。

`#["1"]` 也是一个魔法变量，请自行查看语法。

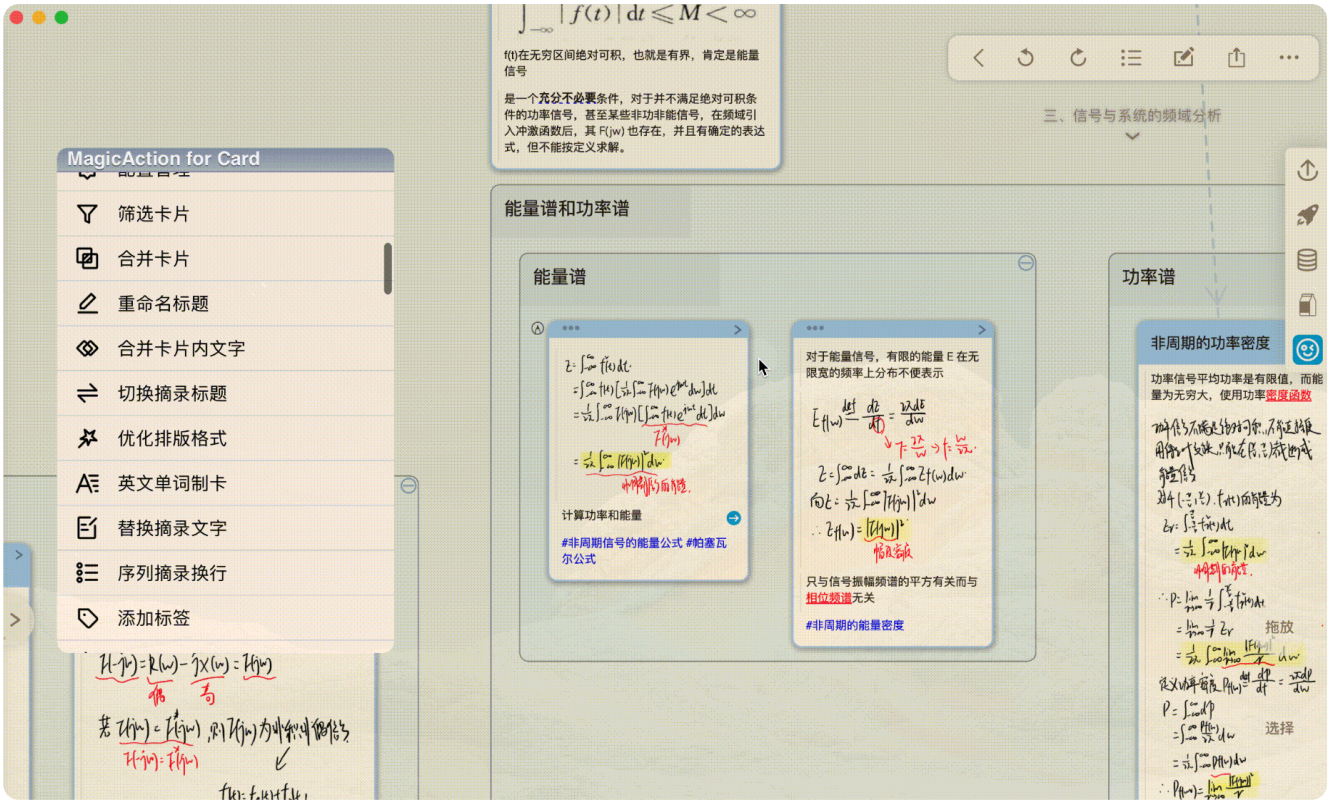


3. 将标签作为标题

输入

```
{{tags}}
```

这里用到的是模版，自行查看 模版语法 以及 模版变量。



合并卡片内文字

更新

v4.0.6 改进：摘录的图片不会自动 OCR 合并。

只会合并文字摘录和文字评论，图片以及 HTML 评论（Markdown）会自动置顶。标签和链接会自动置底。

你可以在这里设置前后修饰，比如开头加上编号，结尾加上换行。

默认为 `%["1"]. $&\n\n`

- `%["1"]` 前面提过，会自动递增的魔法变量，详细用法请查看语法。
- `$&` 表示当前的评论或者摘录。
- `\n` 表示换行，这个到处都会用到，在 OhMyMN 里想要换行，就使用 `\n`。

如果你想直接合并两段文字，不留空隙，输入 `$&` 即可。

选项：

- 合并为摘录：摘录可以使用 MagicAction 中的其他功能进行进一步操作，同时会保留重点。
- 合并为评论。

切换摘录标题

- 切换为不存在的

> **更新**
>
> v4.0.11 新增更多交换方式

- 交换标题和摘录：标题或摘录只有一个存在的情况下，和 `切换为不存在的` 一样。在标题和摘录都有的情况下，会有以下处理方式：
  - 摘录 → 标题
  - 摘录 ← 标题
  - 摘录 ⇄ 标题

提取标题

来自于 Another AutoDef

拆分摘录

> **更新**
>
> v4.0.6 新增

来自于 Another AutoDef

优化摘录排版

来自于 AutoFormat

英文单词制卡

来自于 AutoComplete

替换摘录文字

来自于 AutoReplace

添加换行或序号

来自于 AutoList

修改摘录颜色

来自于 AutoStyle

修改摘录样式

来自于 AutoStyle

搜索卡片内容

来自于 CopySearch

复制卡片内容

来自于 CopySearch

添加标签

来自于 AutoTag

翻译摘录内容

> **更新**
>
> v4.0.16 新增

来自于 AutoTranslate

添加评论

来自于 AutoComment

转换为简体中文

更新

v4.0.6 新增

来自于 AutoSimplify

基于卡片回答

更新

v4.2.1 新增

来自于 AI

AI 动作 (Prompts)

更新

v4.2.0 新增

来自于 AI

MagicAction for Text

这里的动作和选中文字有关，大部分只是将结果复制到剪贴板上。所有动作都来自于各个模块，需要启用后对应模块后才会显示动作。在选中文字后，点击按钮来对选中的文字进行搜索或者复制。不止文字，框选一块区域也可以，甚至可以用来进行公式 OCR。

预先 OCR

> **仅当前文档**
>
> 需要启用 AutoOCR 模块，但不需要打开 **摘录时自动执行** 。

使用 AutoOCR 来进行小语种的文字识别，从而方便后续的复制，搜索，以及翻译操作。

预先转为简体中文

> **仅当前文档**
>
> 需要启用 AutoSimplify 模块，但不需要打开 **摘录时自动执行** 。

使用 AutoSimplify 进行繁简转换。

预先格式化

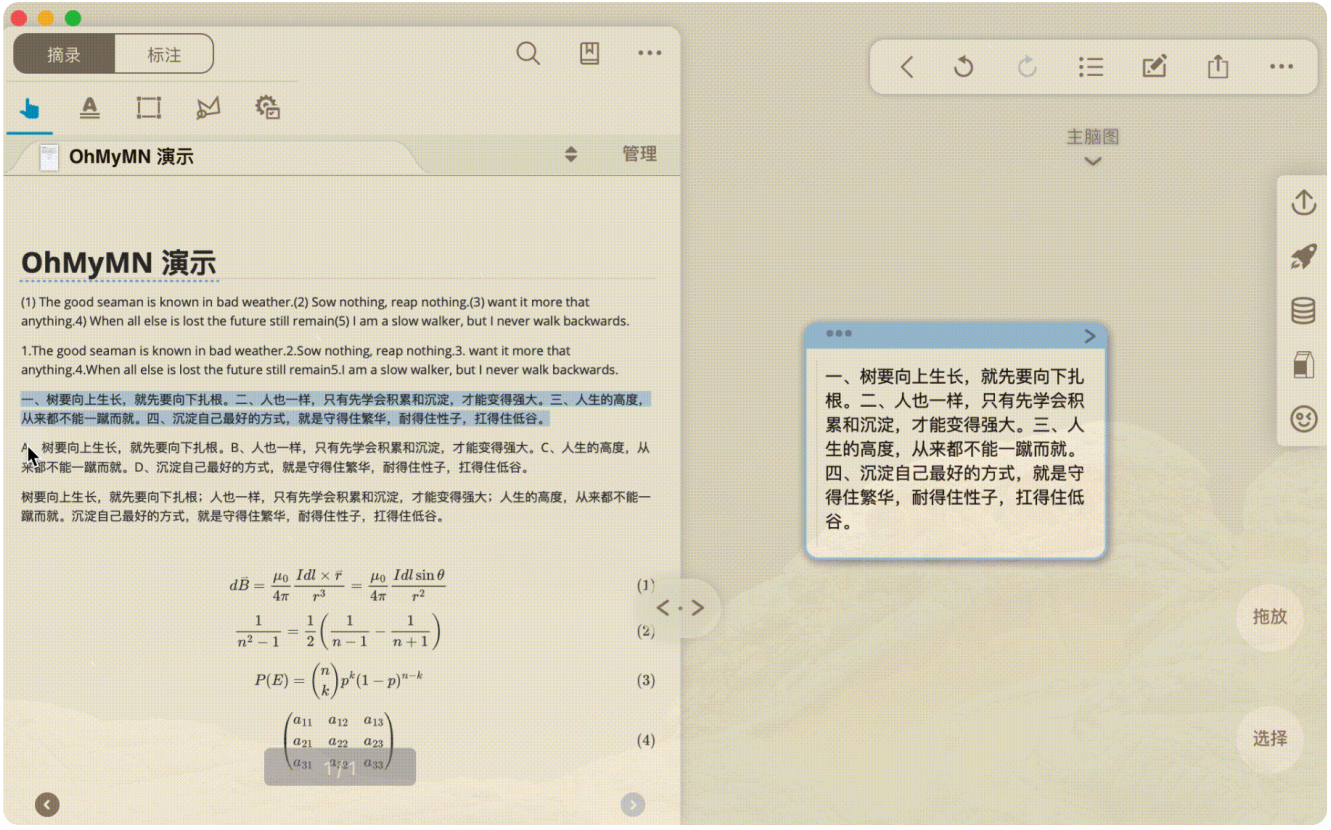
> **仅当前文档**
>
> 需要启用 AutoFormat 模块，但不需要打开 **摘录时自动执行** 。

> **更新**
>
> v4.0.6 新增

使用 AutoFormat 进行排版优化。

弹出更多选项

不知道你发现了没有，当你在文档中选中了一块摘录后，紧接着再用手型工具去选中一段文字，菜单里就会出现 设置标题 加为评论 等选项。这样就可以直接把你选中的这段文字直接作为标题或者评论加到之前选中的摘录里。



OhMyMN 利用了这一特性。当你在文档中选中了一块摘录后，紧接着再用手型工具去选中一段文字，此时再执行动作，某些会将执行结果复制到剪贴板的动作就会弹出更多选项，比如设置标题，合并标题，设为评论等。

> **TIP**
>
> 这里的动作指的是 OhMyMN 里的动作，也就是👇这些动作。

动作

复制选中文字

> **更新**
>
> v4.0.6 新增，v4.1.0 修改，不再属于 CopySearch。

这个动作看似没用，但当你搭配 `弹出更多选项` 时，就知道有多好用了。

搜索选中文字

来自于 CopySearch

公式识别

来自于 AutoOCR

文字识别

来自于 AutoOCR

手写识别

来自于 AutoOCR

二维码识别

来自于 AutoOCR

翻译选中文字

来自于 AutoTranslate

转为简体中文

> **更新**
>
> v4.0.6 新增

来自于 AutoSimplify

AI 动作 (Prompts)

更新

v4.2.0 新增

来自于 AI

Shortcut

更新

v4.0.6 新增

注意

该功能完全由 OhMyMN 提供，与 MarginNote 无关。

通过 URL Scheme 来触发 MagicAction 中的动作，在 Mac 上可以设置快捷键打开 URL。

1. 打开 `marginnote3app://addon/ohmymn?type=card&shortcut=1` 就可以触发第一个卡片动作。
2. 打开 `marginnote3app://addon/ohmymn?type=text&shortcut=2` 就可以触发第二个文字动作。

自定义捷径

更新

v4.0.14 改进。捷径 Pro 更名为自定义捷径，并改进了语法。

可以为每个动作以及任意输入值设置 URL，并将其设置成快捷键。甚至可以同时执行多个动作，通过选择的顺序来改变执行的顺序。

使用快捷键打开 URL

iPad

1. 点击安装快捷指令 https://www.icloud.com/shortcuts/d9027fc514f04fc4add78ae506baba8d
2. 设置-辅助功能-键盘-全键盘控制-命令-划到最下面就可以给快捷指令设置快捷键了。

Mac

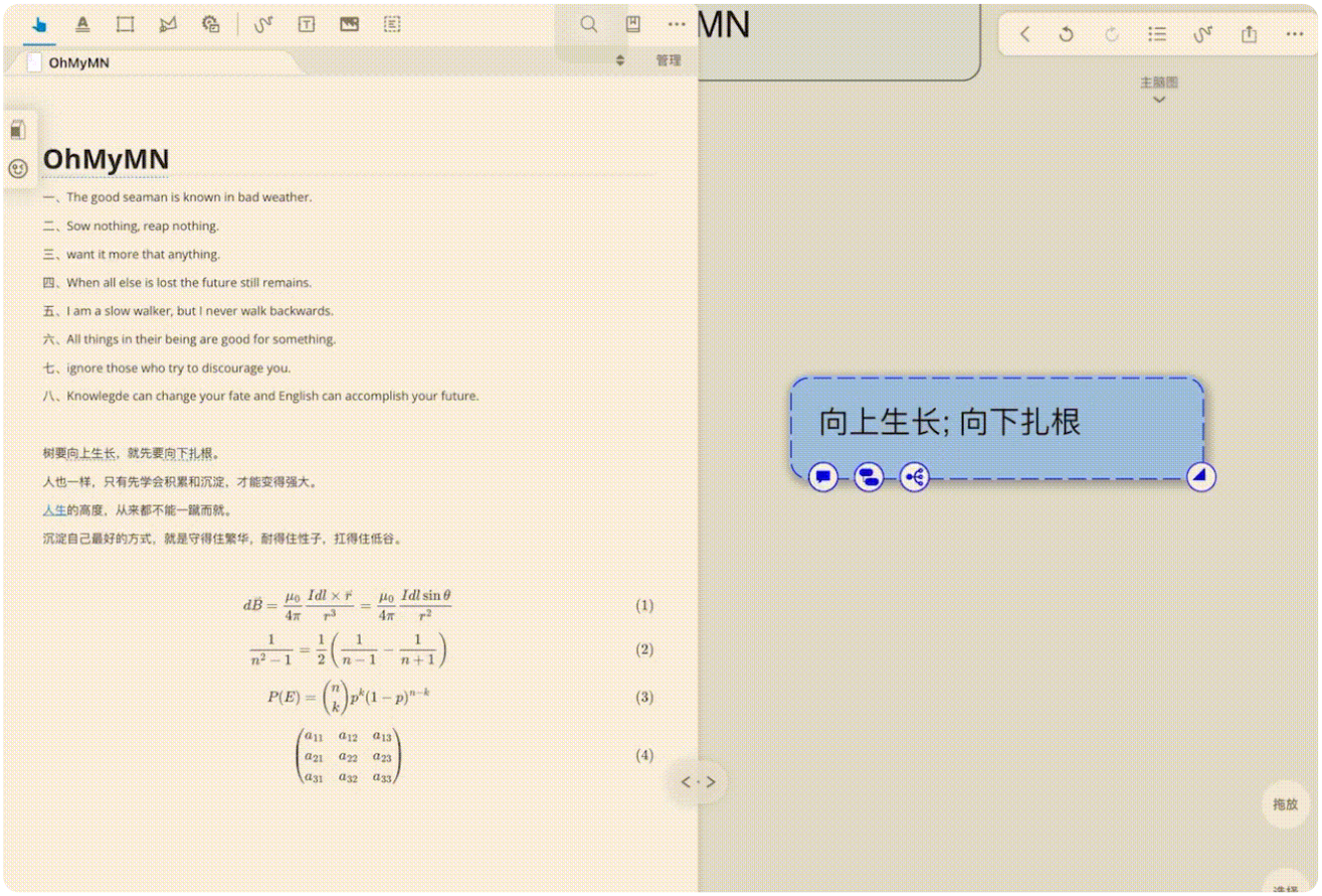
Mac 上的工具就很多了，我通常是使用 Karabiner-Elements，这个还可以单独给 MarginNote 设置快捷键，免费。Raycast 也非常合适，甚至更加简单。

使用手势打开 URL

iPad 上可以使用 Gesture 模块来打开 URL，从而使用捷径的强大功能。

Gesture

> **注意**
>
> 仅 iPad 可用。该功能完全由 OhMyMN 提供，与 MarginNote 无关。



使用手势来触发 MagicAction 中的动作。

> **TIP**
>
> 尽量在中间区域滑动，滑动速度慢且距离长。

通过在不同工具栏上 `上下左右` 滑动来直接触发 MagicAction 上的动作，而不用打开控制面板，一步到位。

自定义捷径

更新

v4.0.14 新增。现在你可以用手势触发自定义捷径。你必须启用 Shortcut 模块，并打开 自定义捷径 的开关。

TIP

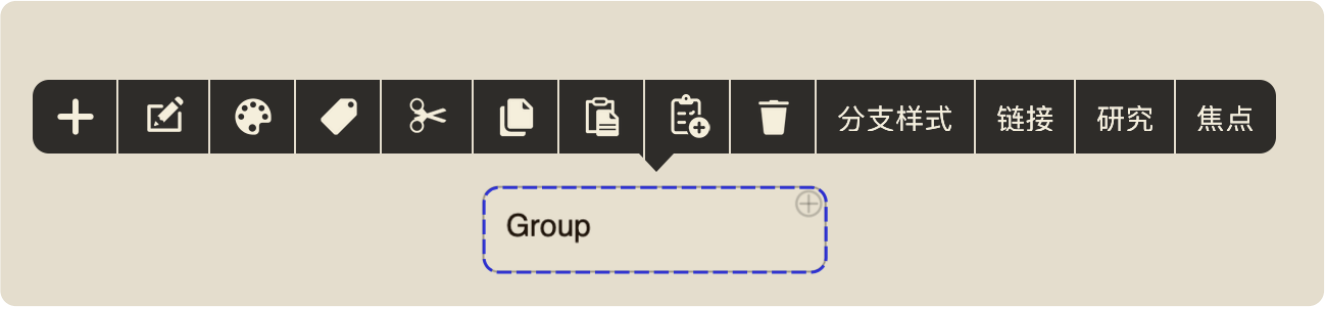
为了让其具有更高的可操作性，我只是简单的打开这个 URL，而没有将其限定在 OhMyMN 中。这意味着你可以用手势触发其他插件中的捷径。以后捷径会成为每个插件的标配功能。

手势识别区域

卡片相关工具栏

目前有三个区域四个方向的手势，两个区域是卡片相关，用于触发 MagicAction for Card 上的动作。

卡片选择工具栏



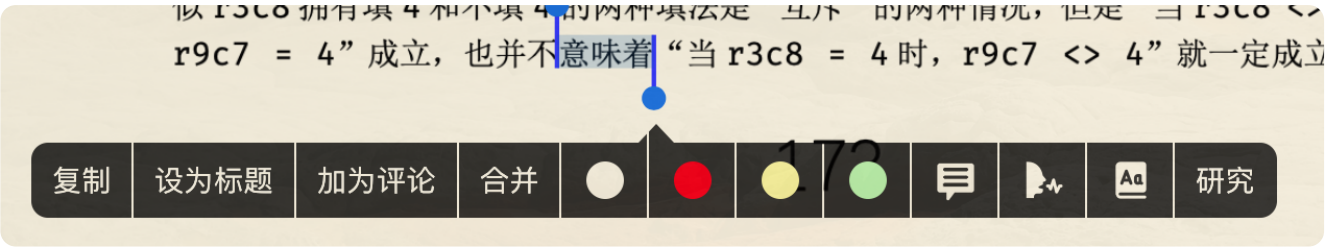
卡片多选工具栏

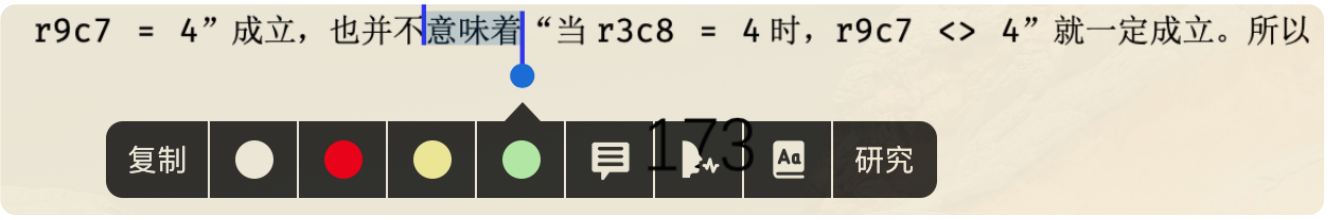
注意

如果只选中单张卡片，但手动开启了多选工具栏，此次多选工具栏的手势不会响应。

文字选择工具栏

这个区域是文字选择相关，一般是用手指或者手型工具选择文字时会出现，用于触发 MagicAction for Text 上的动作。框选一个区域也是一样的。





这两个的菜单样式有所不同，但都属于文字选择工具栏。第一个会出现 设置标题 加为评论 等选项，这是因为在选中这段文字之前，你已经选中了一段摘录的笔记。

MagicAction for Text 已经对此进行特别处理，具体可以自行查看，可以实现了公式 OCR 后直接添加到卡片中。

调整文字选择工具栏识别区域

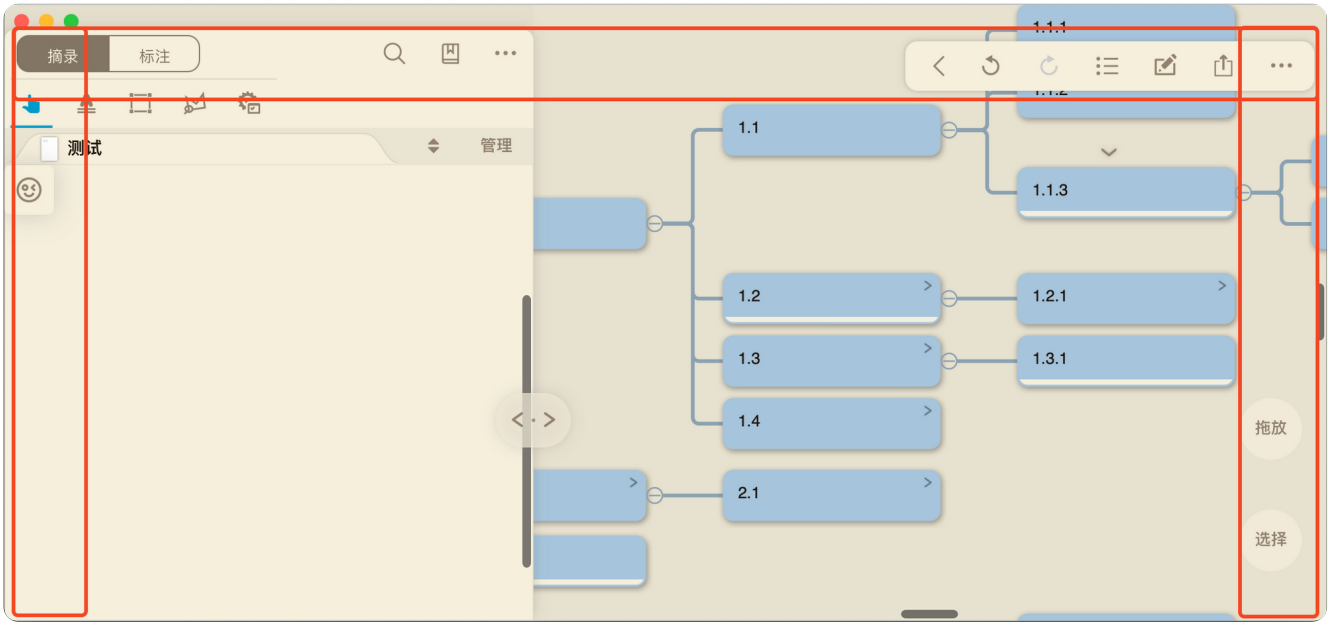
鉴于部分人始终无法成功触发文字选择工具栏上的手势，我怀疑是设备屏幕大小导致，所以开放了自定义。你可以按照以下步骤进行调整。注意，需要在工具栏顶部横向滑动，来获取最接近的值。由于我固定了工具栏高度，所以你只需要调整顶部的坐标即可，但不要随意填写。

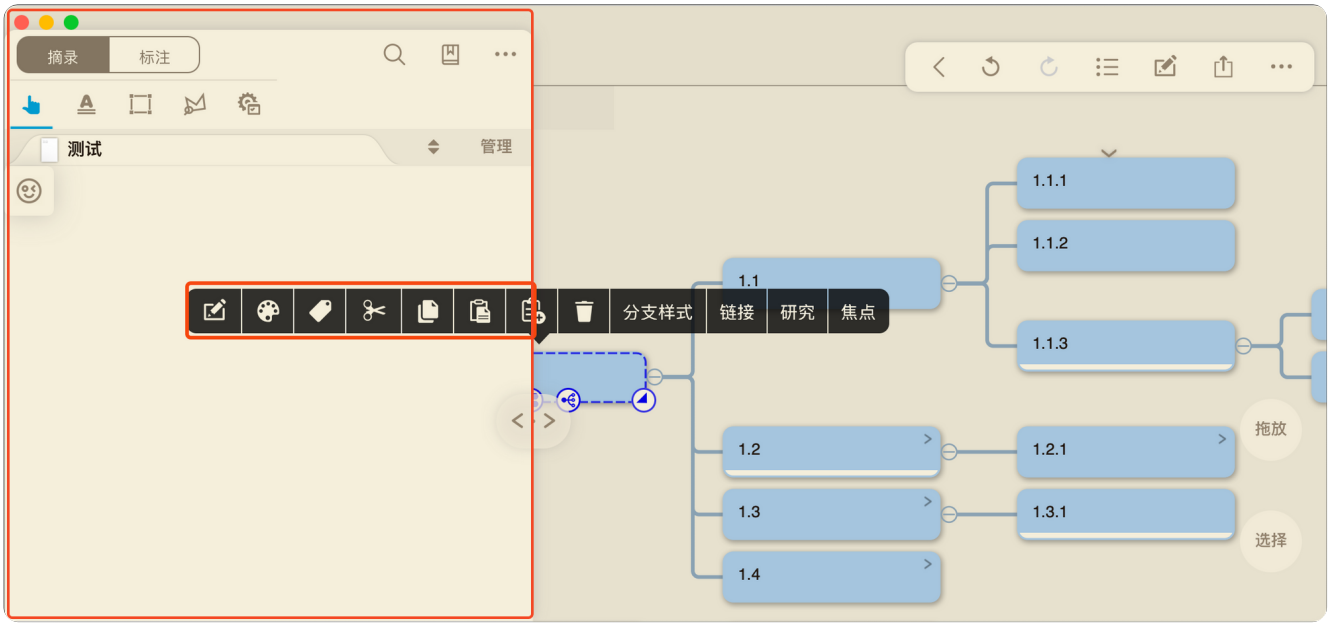
手势屏蔽区域

其实这个手势检测是加在了整个 MarginNote 界面上，理论上在任何地方滑动，OhMyMN 都会接收到信息。只是我做了屏蔽，使其只在指定几个区域上做出反应。

但是这几个工具栏位置的判断并没有那么准确，所以可能会导致没有在工具栏上滑动，OhMyMN 却执行了动作。为了减少这种情况发生，我进一步限制了识别的区域，尤其是 `卡片单选工具栏`，限制最大。

如图所示，如果 `卡片单选工具栏` 出现在红框内，并在红框区域内滑动，均不会响应。




摘录
标注
测试
管理
分支样式
链接
研究
焦点
1.1
1.1.1
1.1.2
1.1.3
1.2
1.2.1
1.3
1.3.1
1.4
拖放
选择

Another AutoTitle

什么样的摘录该设置为标题?

预设

> **TIP**
>
> 预设只需要满足一个就会自动转为标题

1. 字数：这是最容易想到的判断标准，当你某一次只摘录了两个字，必然想把这个摘录作为标题。只要不超过设置的字数，就会自动转为标题。如果中文句子中出现数字或者单词，并不会分开计算，一个单词或者数字也会占用一个类中文字数。
   - 类中文字数：不是由字母组成的都属于类中文。
   - 类英文字数：也就是单词数，这里我把数字也看作一个单词。
2. 不含有点号：所谓点号，标点标点，包含标号和点号，点号就是指表示停顿的一类标点符号，比如 `。.、？?! !，,；;：:` 。没有表示停顿的符号，大概率是作为标题。

自定义

使用正则表达式来判断摘录的文字是否满足要求，满足就转为标题。

> **输入格式**
>
> 正则表达式——判断

示例

比如你如果希望在全中文下，没有标点，字数不超过 10，就会自动转为标题，则可以写成 `/^[\u4e00-\u9fa5]{0,10}$/` 。

~~标题摘录始终为标题~~

更新

v4.0.10 移除

Another AutoDef

该功能与 Another AutoTitle 的区别在于，Another AutoTitle 主要用于将摘录转换为标题，而 Another AutoDef 更多是将摘录的部分内容作为标题，剩余部分作为摘录，起到的是分割和提取的作用。

至于为什么叫做 AutoDef，因为这个功能最常用到的就是定义。

> **对定义的定义**
>
> 定义是透过列出一个事件或者一个物件的基本属性来描述或规范一个词或一个概念的意义。被定义的事物或者物件叫做 **被定义项**，其定义叫做 **定义项**。例如，在“大象是一种原产于亚洲和非洲的大型灰色动物”的定义中，“大象”这个词是被定义项，“是”之后的所有词都是定义项，而“是”可以称为 **定义联项**。

这个定义联项既可以作为分割点，将其分为标题和摘录两部分，也可以通过这个定义联项来判断摘录的内容是否是一个定义，从而自动进行处理。

预设

我已经设置了几个常见的定义联项，为了避免影响正常的摘录行为，所以会比较克制。`xxx` 为定义项，`yyy` 为被定义项。

- xxx : yyy `/[: :]/`
- xxx —— yyy `/[—\-—]{1,2}/`
- xxx ，是(指) yyy `/[,, ]\s*(?:通常|一般)*是指?/`
- xxx 是(指)， yyy `/(?:通常|一般)*是指?\s*[,, ]/`
- xxx 是指 yyy `/(?:通常|一般)*是指/`
- yyy，___称(之)为 xxx `/[,, ].*称之?为/y`
- yyy(被)称(之)为 xxx `/(?:通常|一般)?被?称之?为/y`

> **更新**
>
> v4.0.6 删除该选项，现在默认开启该选项，无法关闭。如果想要只提取请使用 **自定义提取标题** 来代替。

~~默认开启 `摘录仅保留定义项`，也就是将被定义项作为标题，定义项作为摘录。如果关闭，则只会把被定义项作为标题，摘录内容不变，相当于提取出子标题。~~

自定义定义联项

自定义格式

正则表达式——分割

v4.0.7 进行改进：如果使用了正则表达式数组，数组中第一个正则作为分割点，其余正则作为该点的限制条件。

上面的预设我已经写清楚了背后的正则表达式，你可以自定义你需要的定义联项。这里的工作原理是 Split() 函数，把定义联项作为分割点，自然而然就分成了被定义项和定义项。

你可能已经发现了，前面预设中后两个的正则用了一个标志 `y`，并且它们都属于被定义项在后面的情况。`y` 在正则中用的很少，所以我将其作为了一个参数，你只要用了 `y`，AutoDef 就会自动交换定义项和被定义项。

自定义提取标题

自定义格式

Replace() 函数格式——提取

直接从摘录中提取出标题。

更新

v4.0.16 删除了 fnKey。自定义提取标题现在也支持了 **别名转为多个标题**。

如果你想将上面的定义联项预设转为 `自定义提取标题` 的语法，实现只提取标题，你可以这样写：

- xxx：yyy `(/^(.+)[：:]/, "$1")`
- xxx —— yyy `(/^(.+)[—\-]{1,2}/, "$1")`
- xxx，是(指) yyy `(/^(.+)[，,]\s*(?:通常|一般)*是指?/, "$1")`
- xxx 是(指)， yyy `(/^(.+)(?:通常|一般)*是指?\s*[，,]/, "$1")`
- xxx 是指 yyy `(/^(.+)(?:通常|一般)*是指/, "$1")`
- yyy，__称(之)为 xxx `(/[，,].*称之?为(.+)$/, "$1")`
- yyy(被)称(之)为 xxx `(/(?:通常|一般)?被?称之?为(.+)$/, "$1")`

自定义别名分词

生成的标题中通常会有别名，如果你打开 `别名转为多个标题` 选项，AutoDef 会自动将其拆分为多个标题，供标题链接使用。

自带的分词策略:

- 默认：`/或者?|[简又]?称(?:之?为)?/`
- 标点符号：`/[、。，,‘’“”"『』()（） 【】「」《》«»\/\[\]]/`

> **自定义格式**
>
> 正则表达式——分割
>
> v4.0.7 进行改进：如果使用了正则表达式数组，数组中第一个正则作为分割点，其余正则作为该点的限制条件。

MagicAction for Card

提取标题

> **自定义格式**
>
> Replace() 函数格式——提取

- `使用 AutoDef 的设置`：使用 `自定义提取标题` 中输入的自定义表达式。

v4.0.7 改进： 如果卡片已有标题，会自动合并。

拆分摘录

> **更新**
>
> v4.0.6 新增

> **自定义格式**
>
> 正则表达式——分割

- `使用 AutoDef 的设置`：使用 AutoDef 中的预设进行拆分，不包括 `自定义提取标题`。

如果卡片已有标题，会自动合并。

AutoFormat

Powered by Pangu.js

> **有研究表明**
>
> 打字的时候不喜欢在中文和英文之间加空格的人，感情路都走得很辛苦，有七成的比例会在 34 岁的时候跟自己不爱的人结婚，而其余三成的人最后只能把遗产留给自己的猫。毕竟爱情跟书写都需要适时地留白。

这个模块可以给中英文之间加空格，并且会将标点符号修改正确，中文用中文符号，英文用英文符号。不过对于引号和括号，目前无法实现自动转换。除此之外，还会删除中文之间的空格，以及重复的空格。

预设

1. 去除全部空格：有时候 PDF OCR 后会出现大量的空格，可以使用这个功能来去除全部空格，但是仅限于没有单词的情况下，否则单词会合在一起。
2. 半角转全角：中文使用全角符号，英文使用半角符号。
3. 中英文加空格
4. 去除中文间空格。
5. 去除重复空格: 将连续多个空格变为 1 个。

开启多个预设，会从上到下依次执行。

自定义

> **输入格式**
>
> Replace() 函数格式——替换

英文标题规范化

Powered by to-title-case

开启该选项后，AutoFormat 会按照规则将自动生成的标题规范化。请注意，是自动生成的标题，即通过 Another AutoTitle，Another AutoDef 以及 AutoComplete 生成的标题。

> **注意**
>
> 标题会首字母大写。但是如果标题全部大写，说明是开启了 MarginNote 首页设置里的标题大写。

规则：

1. By default, capitalize all words
2. Always capitalize the first and last word in titles and subtitles
3. Capitalize both parts of hyphenated words
4. Lowercase articles: a, an, the
5. Lowercase conjunctions: and, but, or, nor
6. Lowercase short prepositions: as, at, by, for, in, of, on, per, to, via
7. Lowercase versus: vs., vs, v., v
8. Lowercase NYT words*: en, if
9. Let intentional capitalization stand

MarginNote for Card

优化摘录排版

使用预设以及自定义进行优化。只能优化摘录和标题，无法优化评论，他们间的 区别 可以点击查看。

- 标题：如果开启了 `英文标题规范化`，此时优化标题会也进行英文标题规范化。
- 摘录

AutoComplete

Powered by ECDICT & API

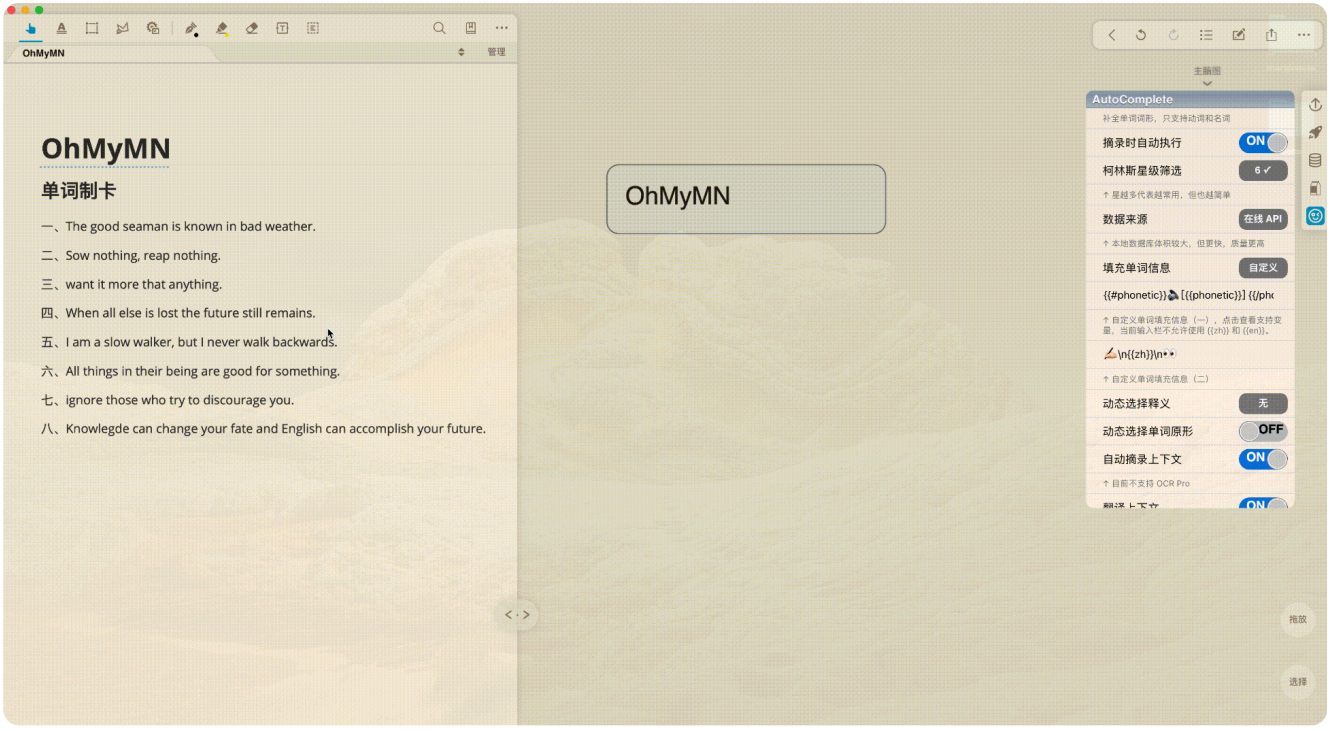
本模块会用到一个在线 API 来获取数据，需要联网，并且由于服务器在国内，国外用户有可能无法正常使用。目前该 API 使用免费，但不保证长期有效。 v4 版本提供了本地数据库版本，可以自行选择下载和开启，可在 QQ 频道 中获取。数据库较大，首次安装会解压数据库，请耐心等待。还需要在设置中将数据来源改为本地数据库。

更新

v4.2.0 移除在线 API，必须如果想用 AutoComplete 必须下载数据库版本。

该模块用于解决使用 MarginNote 学习英语，摘录英文单词的一个困扰，那就是在实际文章中，英文单词往往都不是原型，无法很好的利用标题链接。开启该功能后，会自动补全第三人称，复数，过去，完成，比较级等等形式。

更为强大的是，不管你摘录的是不是原型，都可以自动判断，然后补全所有形式。



柯林斯星级筛选

越常用的单词柯林斯星级就越高，一共 6 个等级，0-5 星。通常 5 星的单词都是比较简单的单词，可以排除。

填充单词信息

可以将单词的部分信息添加为评论，达到自动制卡的目的。

自定义

> **自定义格式**
>
> 模版

有以下几个变量

变量名	备注
`word`	单词原型
`phonetic`	音标，大部分为英式音标
`zh`	中文释义
`en`	英文释义
`tags`	高考、四六级等等标签
`collins`	柯林斯星级

有两个输入栏，可以生成两个评论，通常第一栏填入音标，标签等信息，第二栏填入中文或者英文释义。这样方便在复习模式中将释义单独放在卡片背面。

默认填充 1: `{{#phonetic}}🔊 [{{phonetic}}] {{/phonetic}} {{collins}}{{#tags}}\n🏷️ {{tags}}{{/tags}}`

默认填充 2: `✍️ \n{{zh}}\n👀`

动态选择释义

开启此选项后，摘录的时候会弹出弹窗来选择当前文中的含义（最多 9 个），如果没有，则可以在输入框中输入来自定义。

多选释义

通过在输入框中输入指定的变量，来多选释义。

- `[all]`：所有释义。
- `[1-9]`：编号 1-9 的释义。
- `[123789]`：编号 1，2，3，7，8，9 的释义，依次类推。
- `[adj]`：词性为 adj 的释义，依次类推。

多选释义和自定义释义可以同时使用，比如 `[all] v.新的释义` 。

动态选择单词原形

有些单词比如 lay，既是 lie 的过去式，也是 laid 的原形。这时候就需要主动选择一下。

自动摘录上下文

> **更新**
>
> v4.0.10 上下文作为摘录而不是评论。

顾名思义，可以自动摘录当前单词所在的句子，从而保留语境，方便复习记忆。

> **限制**
>
> 目前不支持 OCR Pro，如果 PDF 本身没有文字层，则无法正常工作。如果非常依赖这个功能，可以使用 Abbyy 来 OCR 整本书。 目前识别算法还不够完善，有时候会识别错误，或者识别不全，这时候可以手动调整选区。

~~翻译上下文~~

> **更新**
>
> v4.0.10 移除，请直接用 AutoTranslate。

MagicAction for Card

英文单词制卡

> **更新**
>
> v4.0.16 改进，多选模式下自动关闭词义选择和原型选择。

使用相同的配置。AutoComplete 生成的信息都属于评论，评论无法被修改，只能删除重新添加。

- 追加：添加新的评论。
- 替换: 先删除旧评论，再添加新的评论（如果有图片，可能会跑顶上去）。

AutoReplace

在摘录的时候自动替换某些字或词。就是常规 replace 函数的使用，不多说。

自定义

输入格式

Replace() 函数格式——替换

OCR Pro 常见错误（待补充）

MagicAction for Card

替换摘录文字

自定义格式

Replace() 函数格式——替换

AutoList

在摘录列表，选择题等等，往往需要单独换行，而 OCR 无法实现。

预设

提供了 3 个预设，每种预设包括自定义必须匹配到两个序号才会执行。

`(/\s*([A-Za-z][.、，,])/g, "\n$1")`

- 字母 ABCD，其实也包括 abcd。
  - 当字母后跟着 `.、，,` 时有效，且**仅中文下有效**

`(/\s*([其第]?[一二三四五六七八九十]{1,2}[.、，,])|\s*([其第][一二三四五六七八九十]{1,2}是?[.、，,]?)/g, "\n$1$2")`

- 一二三四，这么复杂的正则就是为了尽可能的避免影响到正常的摘录。
  - 当一二三四前跟着 `其|第` 时有效
  - 当一二三四后跟着 `.、，,` 时有效

`/\s*([\( （【\[]?\s*[0-9]{1,2}\s*[\)） \]】]?[.、，,]\D)|\s*([\( （【\[]\s*[0-9]{1,2}\s*[\)） \]】][.、，,]?)/g`

- 1234
  - (1) （1） [1] 【1】有效
  - 1 `.、，,` 有效

自定义

> **输入格式**
>
> Replace() 函数格式——替换

在匹配到的字符串前面或者后面加上 `\n` 即可，这就是换行符。

为了给每一行添加编号，这里要用到 replace 函数的第三个参数 `fnKey`，通过为其设置不同的数字来编号。

fnKey	编号类型
1	1. 2. 3.
2	A. B. C.
3	a. b. c.
4	壹、贰、叁
5	一、二、三
6	① ② ③
7	❶ ❷ ❸

示例

- `(/[;；]/, "$&\n", 1)`
  - 表示在 `;` 或者 `；` 后换行。并且每一行用 1. 2. 3. 来编号

MarginNote for Card

添加换行或序号

和自定义一致

AutoTag

在匹配到正则的情况下自动添加指定标签，也可以从摘录中提取特定内容为标签。

更新

v4.0.11 支持记录自动生成的标签，在修改摘录时自动删除之前生成的标签。

自定义

自定义格式

Replace() 函数格式——提取

例

- `(/^.+$/gs, "这是一个例子")` 即可每次都添加一个标签为“这是一个例子”。

更新

v4.0.11 支持图片摘录自动添加标签。 v4.0.16 通过空格来添加多个标签。标点符号自动替换为 `_` 。

- `(/@picture/gs, "这是一张图片")` 摘录图片时自动添加标签。

MagicAction for Card

添加标签

输入格式

Replace() 函数格式——提取，传入卡片中的摘录。

由于大部分情况下只是为了添加标签，而无须提取，所以你可以直接输入标签内容。

AutoStyle

AutoStyle 可以单独设置并且固定文本摘录和图片摘录（划框和套索）默认的颜色和样式，同时还可以根据各种预设来自动设置颜色和样式。

预设

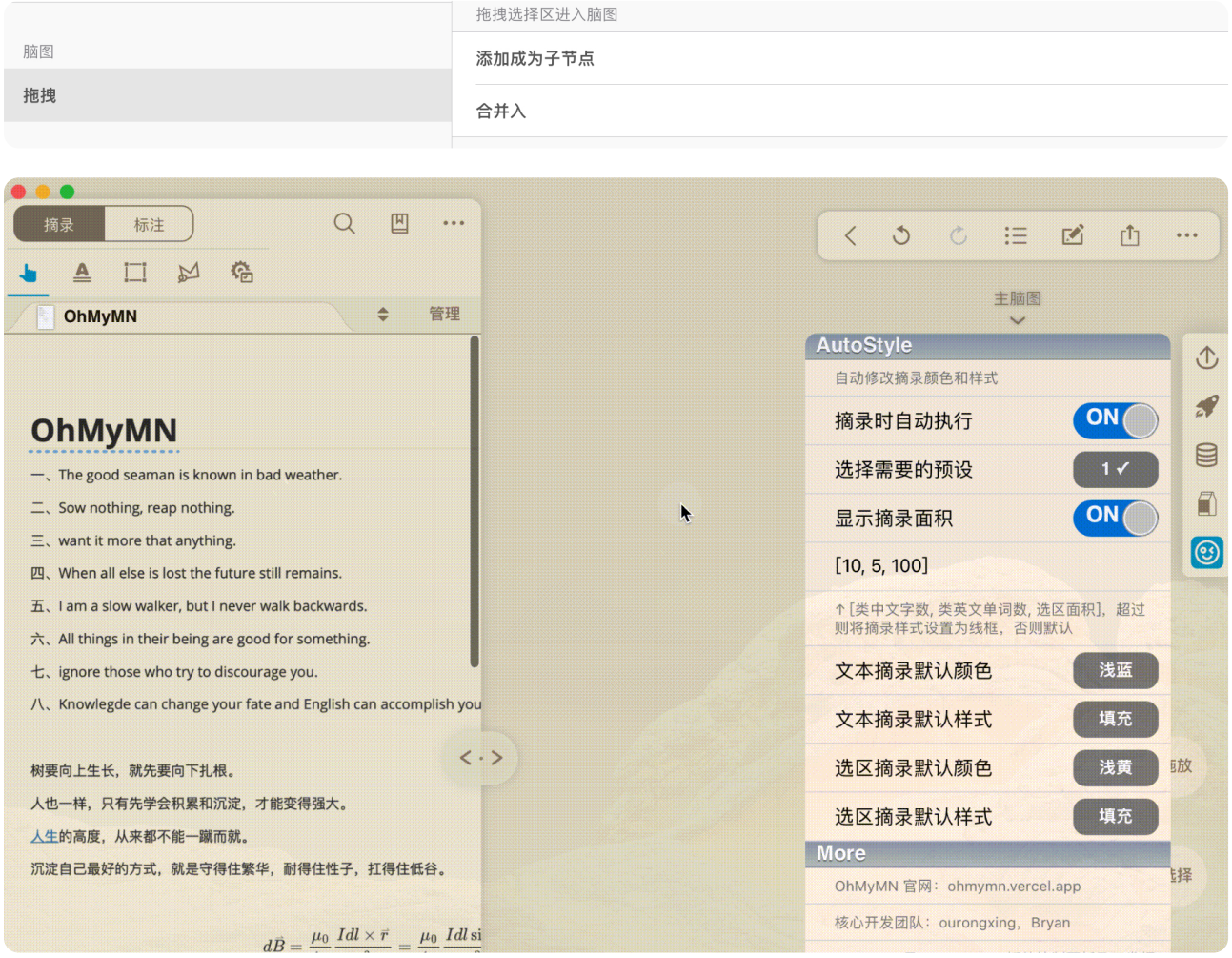
> **WARNING**
>
> OhMyMN 只能在摘录或修改摘录的时候才能触发，拖拽卡片，合并卡片均不会触发，但是可以用手势配合 `修改摘录颜色-使用 AutoStyle 的设置` 来刷新颜色。

有四个预设，一个关于样式，三个关于颜色。

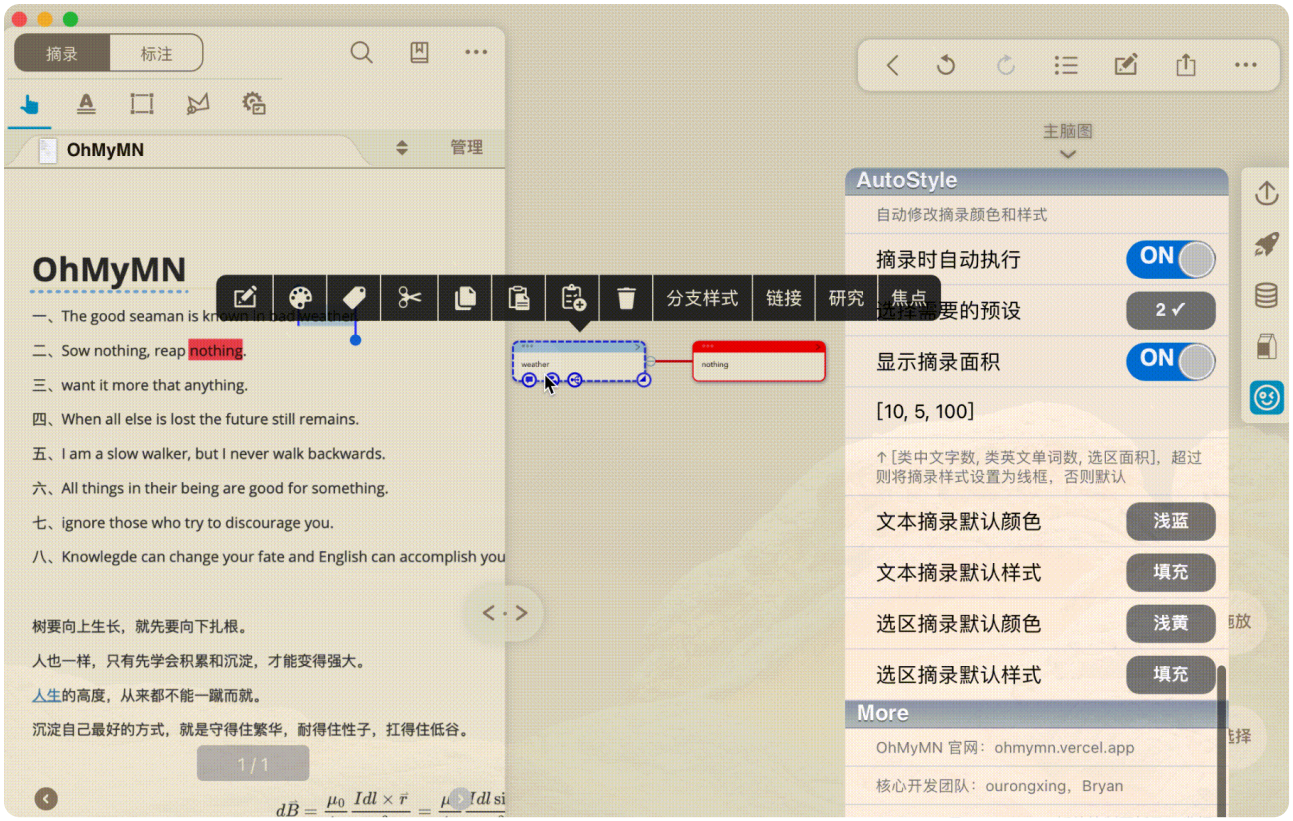
1. 样式由字数或面积决定：因为一旦字数多了或面积大了，如果用填充，就会出现大面积的色块，从而分散注意力，此时应该切换为线框。
    - 输入：`[类中文字数，类英文字数，选区面积]`
    - 这里的字数和 Another AutoTitle 中的字数是同一个概念。
    - 选区面积可以通过打开 `显示选区面积` 来获取。

> **TIP**
>
> 颜色预设有优先级，**颜色跟随卡片** ⇒ **颜色跟随兄弟卡片** ⇒ **颜色跟随父卡片** ⇒ **默认**
>
> 也就是说同时开启这三个预设，如果合并进卡片里，就跟随卡片。如果是作为子节点，有兄弟卡片，就跟随兄弟卡片，如果没有就跟随父卡片。

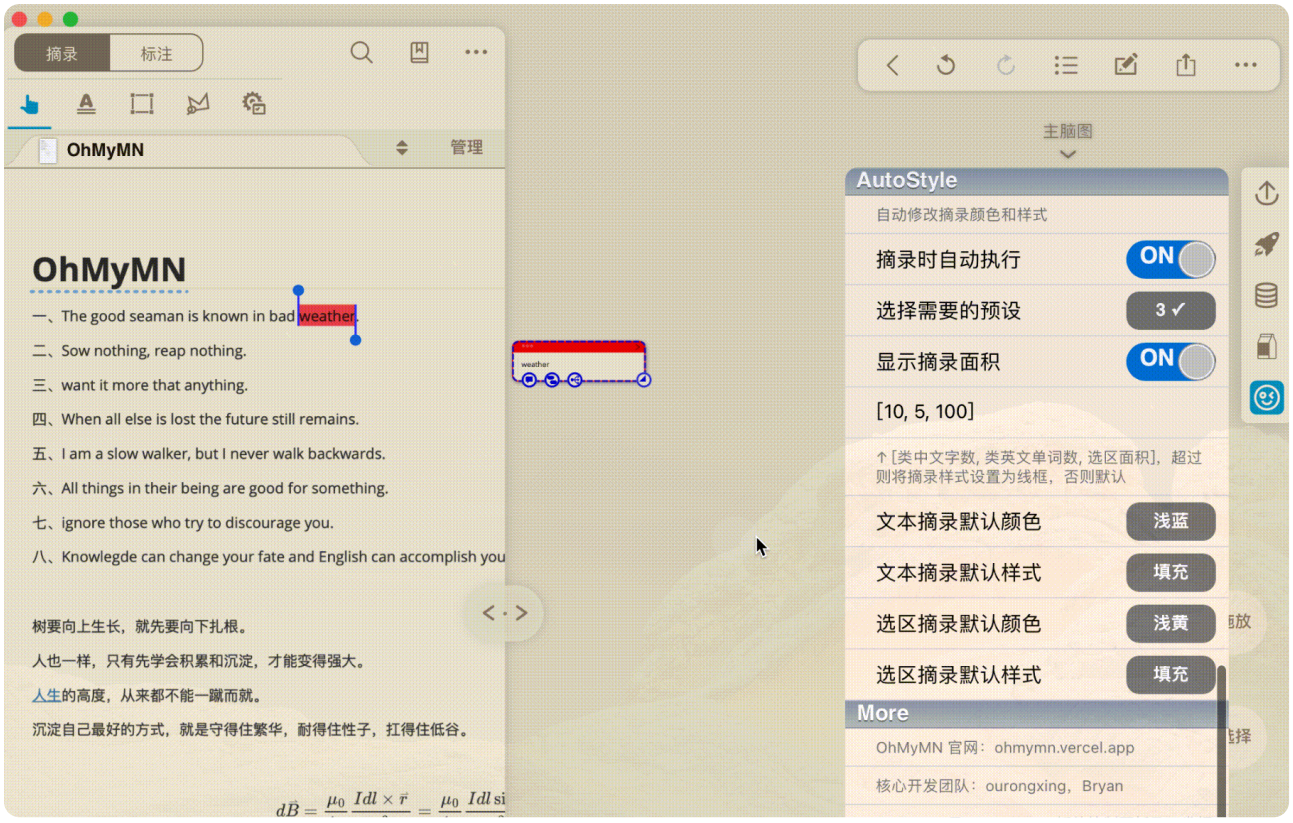
2. 颜色跟随卡片：将下方选项（MarginNote 主页设置）设置为 `合并入`，便可以在摘录时自动将颜色修改为拖拽合并进的卡片的颜色。



3. 颜色跟随兄弟卡片：所谓兄弟卡片，就是同一个父卡片的卡片，我设置的是跟随第一个兄弟卡片。使用该预设最好是将上方选项设置为 `添加为子节点`。



4. 颜色跟随父卡片：将上方选项设置为 `添加为子节点`，就可以跟随其父卡片。



MagicAction for Card

修改摘录颜色

- 输入颜色索引，1-16，也就是色盘从左到右，从上到下。
- `使用 AutoStyle 的设置`：使用预设来刷新颜色。

修改摘录样式

- `使用 AutoStyle 的设置`：使用预设来刷新样式。

CopySearch

> **TIP**
>
> CopySearch 没有以 Auto 开头，说明没法摘录时自动执行，可以通过手势或者在 MagicAction 中手动点击执行。

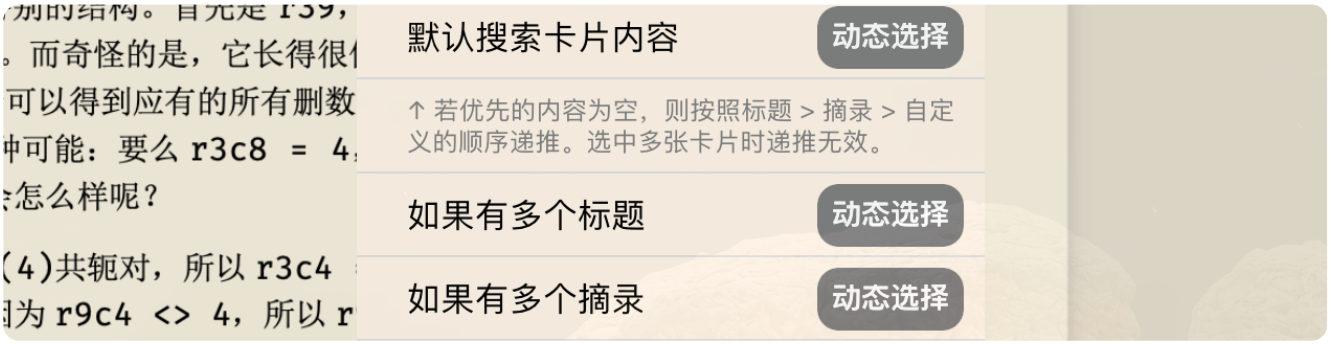
CopySearch 可以让你搜索和复制这张卡片上的一切，不管是看得见的摘录或者标题，还是看不见的 URL 或者修改时间。你还可以将各种属性进行自由组合，通过自定义 URL，将这些信息传入其他软件中，不管是导出还是搜索，一切都有可能实现。

选择卡片内容

卡片的组成非常复杂，可能 OhMyMN 看到的卡片和你眼中的卡片还有所不同，一张卡片中可以有多个标题，多个摘录，多个评论。当你搜索或者复制的时候，如何精准选中你真正想要的。

CopySearch 给出的解决方案是 `动态选择`

> **TIP**
>
> **默认搜索卡片内容** 只针对 **搜索卡片内容** 。 **复制卡片内容** 会在执行时弹出该选项，这样方便通过不同手势一步到位。由于 **搜索卡片内容** 既要选择搜索内容，也要选中搜索引擎。我觉得给不同搜索引擎设置不同手势会更重要一点。



`动态选择` 会给出所有标题，所有的摘录以及自定义，让你在搜索的时候进行选择。

除此之外，还有三个选项:

- 优先标题
- 优先摘录
- 自定义

为什么是优先，当优先的内容为空时，就会按照 标题→摘录→自定义，往下递推，直到有不为空的出现。

自定义

> **自定义格式**
>
> 模版

- 比如最常用的 MarkDown 格式 MNLink: `[{{titles.0}}]({{url.pure}})`
- 再或者是第一条评论: `{{comments.text.0}}`

> **更新**
>
> v4.0.6 改进：自定义复制和自定义搜索可以单独设置。

多张卡片

如果选中了多张卡片，就没法动态选择了，就默认第一个。对于多张卡片，不管是搜索还是复制，都是将所有卡片的指定内容合并在一起，而不是单独复制或搜索。

这里涉及到了合并时进行编号或者换行。这个和 MagicAction for Card -- 合并卡片内文字 其实是一样的，完全可以照搬过来，这里就不多说了。

搜索 URL

可以是网址，也可是其他软件的 URLScheme。

1. URL Schemes 使用详解 - 少数派
2. 入门 iOS 自动化：读懂 URL Schemes - 少数派
3. URL Scheme 查询指南 - 少数派

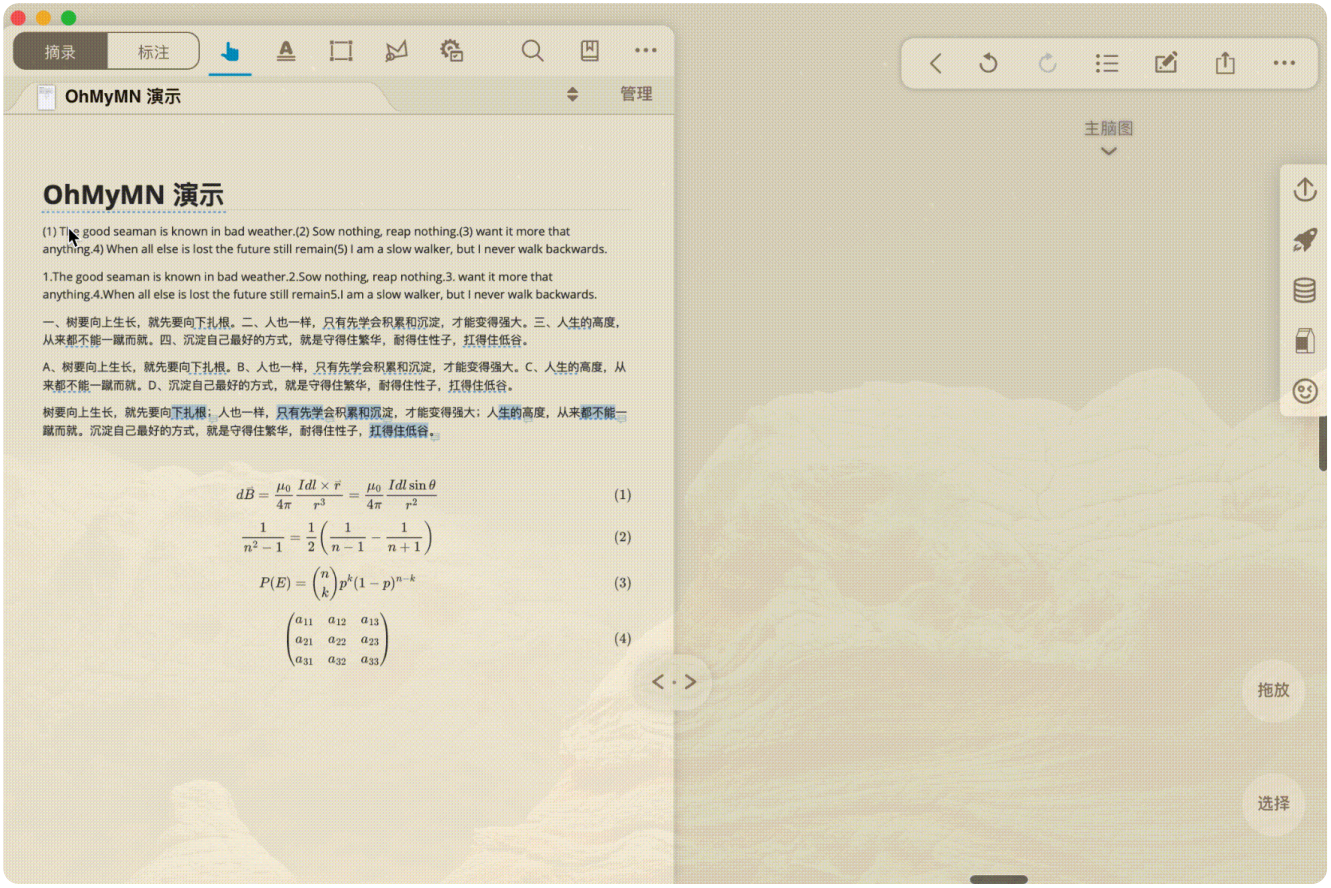
格式：将 URL 搜索关键词的部分换成

- 欧陆词典：`eudic://dict/`
- 百度搜索：`https://www.baidu.com/s?wd=`

- 指定 Edge 浏览器打开百度搜索：`microsoft-edge-https://baidu.com/s?wd=`

AutoTranslate

AutoTranslate 的主要职责是在摘录的时候自动翻译（翻译的结果单独作为评论）。



获取 API 密钥

首先你需要知道这个服务是调用的第三方的翻译服务，所以需要你自行去获取这个服务的密钥，没办法直接使用。我提供了百度翻译和彩云小译可以选择，百度翻译以及彩云小译都有免费的额度，足够你使用。

百度翻译

> **注意**
>
> 自行搜索 如何申请百度翻译 API，具体价格和额度以官网为准，所产生的任何费用与 OhMyMN 无关。

AutoTranslate 使用的是百度翻译高级版本，会更准确一点，支持的语言会更多，还支持自定义术语库，可以自定义一些专业术语的对应关系，从而精确翻译。

彩云小译

> **注意**
>
> 自行搜索 如何申请彩云小译 API，具体价格和额度以官网为准，所产生的任何费用与 OhMyMN 无关。

彩云小译只支持中英日三国语言的互译，对大部分国内学生来说就够用了，可以在百度翻译的免费额度用完后切换到彩云小译。

限制触发的条件

虽然我只提供了字数上的限制，低于某个字数就不会执行，这个和 Another AutoTitle 中的字数限制一模一样，分了类中文和类英文，这里不多说。

底层我还加了一层限制，那就是如果摘录的语言本身不属于你选择的输入语言，就不会执行。当然，我也只能基于是否有字母，是否是半角来判断，可能不会太精确，比如英语和法语之间就没办法。

MagicAction for Card

翻译摘录内容

> **更新**
>
> v4.0.16 新增

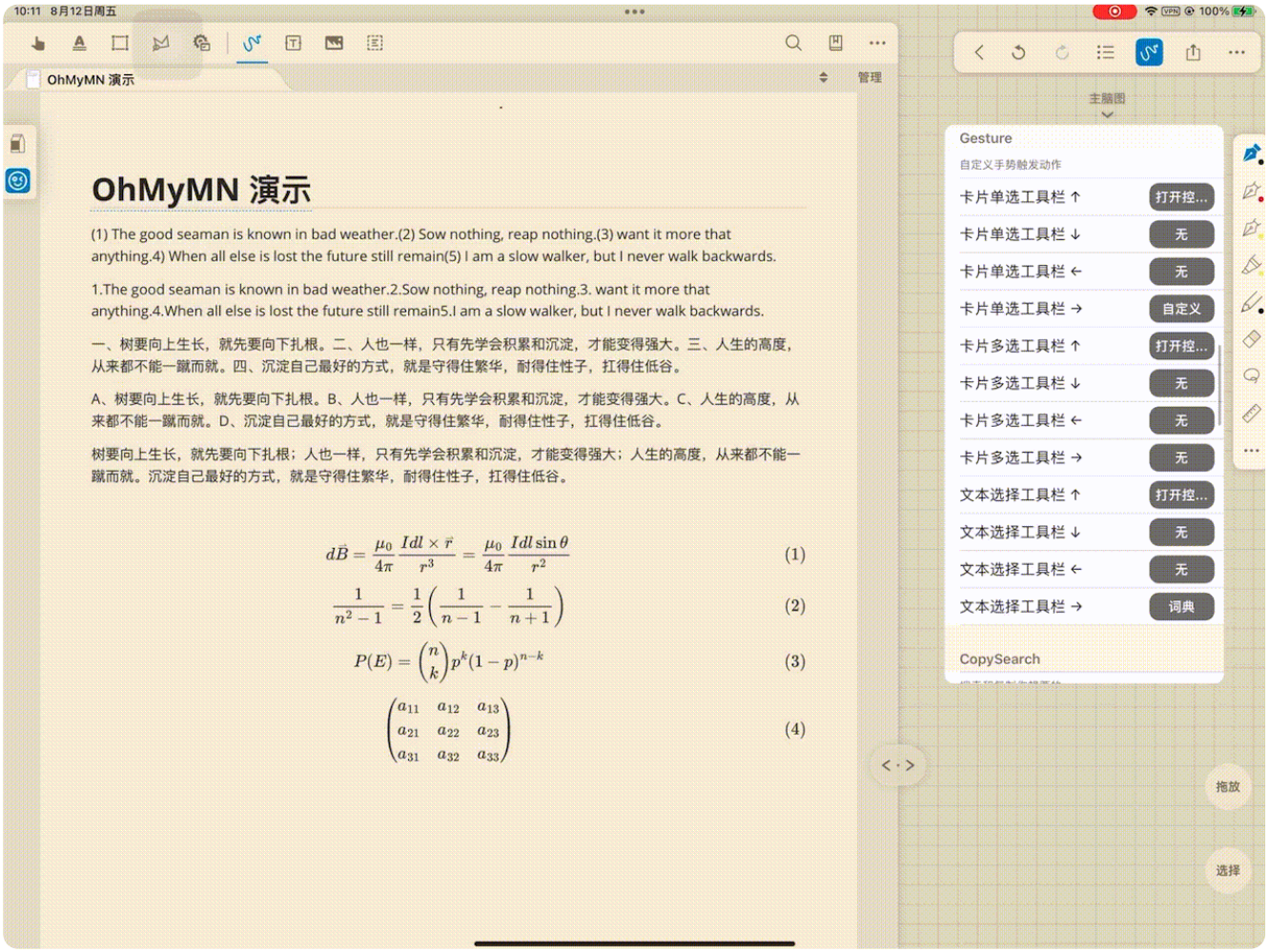
支持翻译卡片中的所有摘录，翻译结果作为评论添加到卡片中。请勿同时翻译过多内容，以免触发 API 限制。

MagicAction for Text

翻译选中文字

可以实现所谓的划词翻译。

利用 Gesture 模块，添加触发手势，真正实现划词翻译，还不用担心误触。



AutoOCR

AutoOCR 的本职工作是进行小语种的在线矫正，在摘录时自动执行。公式识别只能在 MagicAction for Text 手动调用，无法摘录时自动识别。

获取 API 密钥

和 AutoTranslate 一样，AutoOCR 也是使用的第三方服务，需要你自行获取密钥。

百度 OCR

> **注意**
>
> 自行搜索 如何申请百度 OCR API，具体价格和额度以官网为准，所产生的任何费用与 OhMyMN 无关。

AutoOCR 采用高精度版本，免费额度较低。额度用完会自动停止服务。

MathPix

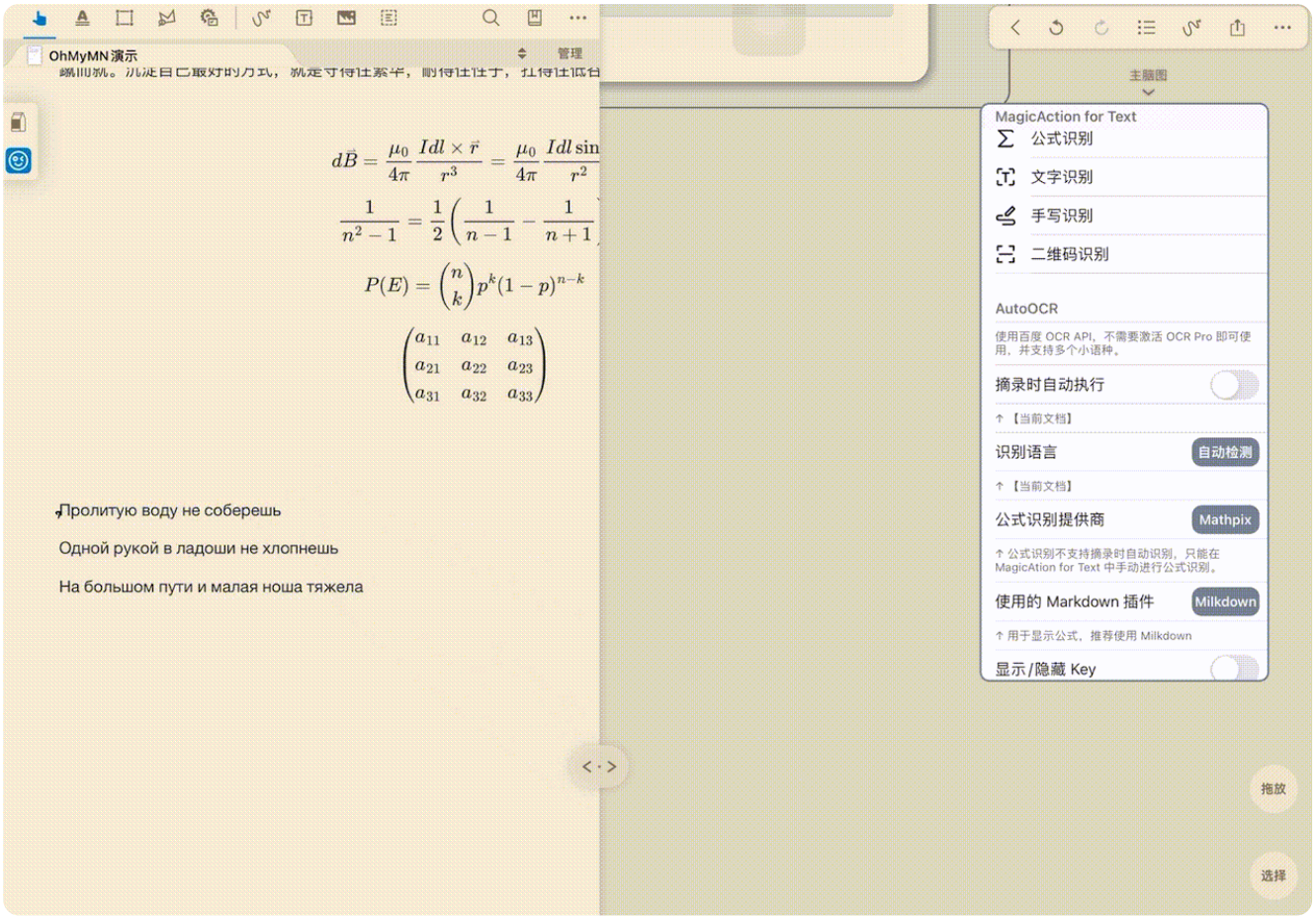
> **注意**
>
> 自行搜索 如何申请 MathPix API，具体价格和额度以官网为准，所产生的任何费用与 OhMyMN 无关。

MathPix 仅用于公式识别，其准确度比百度要高。 需要注意的是，MathPix 价格经常变化，并且价格不菲。

小语种在线矫正

> **注意**
>
> **摘录时自动执行** 和 **识别语言** 均为当前文档有效，这样可以为不同的文档单独设置。

MarginNote 的 OCR Pro 不支持很多小语种，导致无法正常摘录，比如俄语（现在已经支持）。Auto OCR 利用百度的 OCR 服务来重新进行在线矫正。



搭配上 AutoTranslate 还可以实现小语种的自动翻译。

MagicAction for Text

公式识别

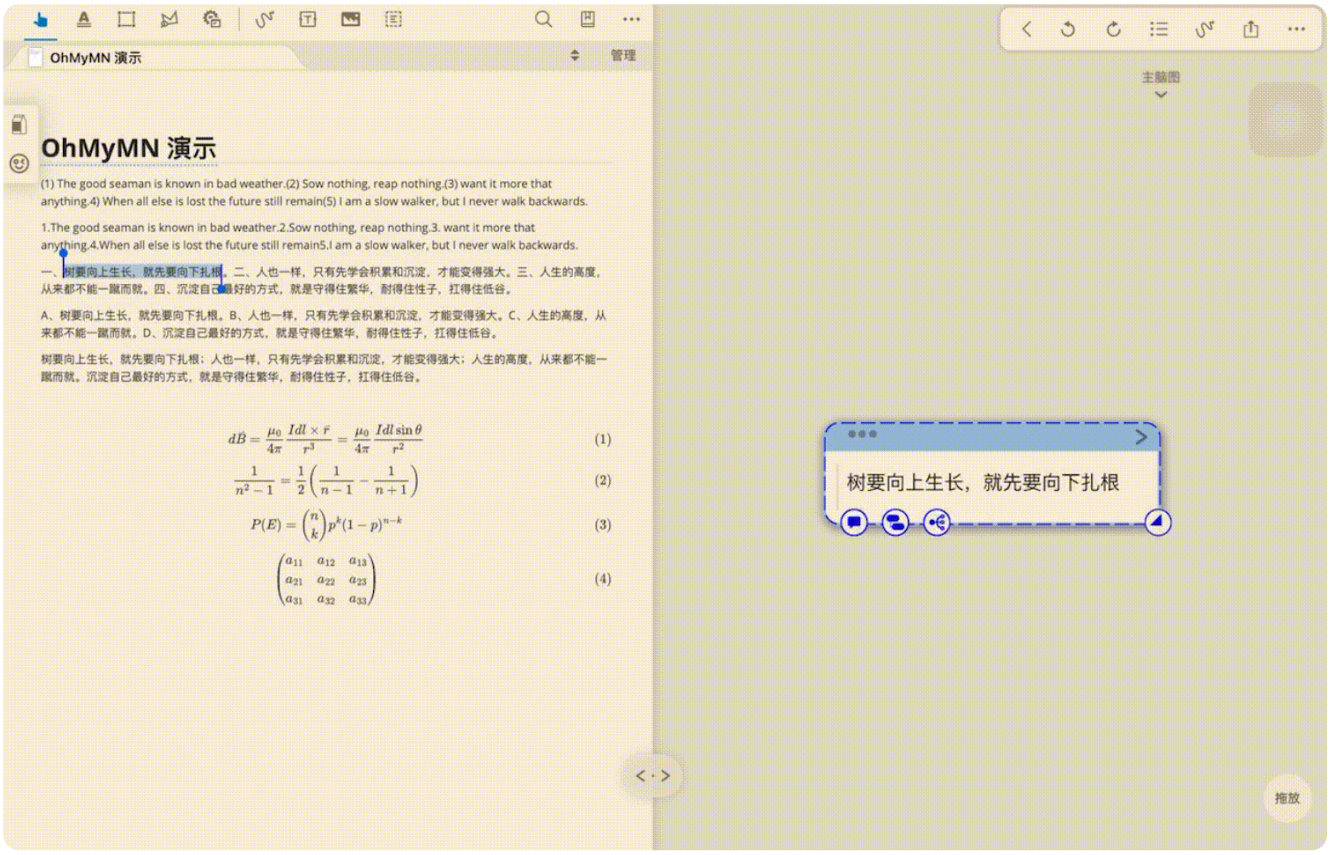
> **注意**
>
> 公式识别只能在 MagicAction for Text 手动调用，无法摘录时自动识别。不要框选超过两行的文字，容易失败，并且在卡片中显示不完整。

需要利用 Markdown 插件来显示公式，所以你必须提前安装好 Markdown 插件。目前有三款 Markdown 插件，推荐使用 Milkdown。

该功能有三个选项

- `pure latex`：MarkDown 插件请选择该选项，包括 Milkdown 开启兼容 MarkDown 后也需要选择该选项。请注意 Milkdown 和 MarkDown 这两个插件在公式这方面语法不相同，请认真思考后再选择是否兼容。

- `$$latex$$` `$latex$`：myMarkdown 和 Milkdown 都可以选择，至于区别自行了解。



请做好心理准备，虽然看上去很爽，但实现起来很复杂：

1. 申请百度 OCR 或者 MathPix 的 API 密钥。
2. 填入 AutoOCR 中，并且选择对应的公式识别服务商。
3. 安装任意一款 Markdown 插件，并且选择对应的 Markdown 插件。
4. MagicAction for Text —— 弹出更多选项，开启 `添加为评论`。
5. **先选中一条摘录或者卡片**（否则只能将结果复制到剪贴板上），再框选，再点击 `公式识别`，识别好的公式 Latex 会自动写入之前选中的卡片中。使用 Gesture 模块来调用执行体验更佳。

文字识别

对选中的文字或者区域进行文字识别，并将结果复制到剪贴板上。

手写识别

使用百度 OCR 服务来进行手写识别，注意免费额度。

二维码识别

使用百度 OCR 服务来进行二维码识别，注意免费额度。

AutoComment

在匹配到正则的情况下自动添加指定评论，也可以从摘录中提取特定内容为评论。

自定义

> **输入格式**
>
> Replace() 函数格式——提取

例

- `(/^.+$/gs, "这是一个例子")` 即可每次都添加一条评论为“这是一个例子”。

> **更新**
>
> v4.0.11 支持图片摘录自动添加评论。

- `(/@picture/gs, "这是一张图片")` 摘录图片时自动添加评论。

MagicAction for Card

添加评论

> **输入格式**
>
> Replace() 函数格式——提取，传入卡片中的摘录。

由于大部分情况下只是为了添加评论，而无须提取，所以你可以直接输入评论内容。

AutoSimplify

更新

v4.0.6 新增

Powered by OpenCC

自动将摘录转换为简体中文。

自定义

输入格式

Replace() 函数格式——替换

异体字

中国台湾和中国香港对于异体字的定义有所不同，因此在这两个地区的繁体中文中，可能会出现不同的异体字。

AI

Powered by OpenAI

使用本模块，请准备好网络以及 **OpenAI API Key**。OpenAI API 不支持中国大陆以及中国香港访问。

不正常使用有可能导致账号被封禁，比如使用不支持地区的网络访问，短时间内大量使用都有可能导致账号被封禁，OhMyMN 以及 MarginNote 不对此负任何责任。

OhMyMN 完全开源，不会上传数据，如果遇到 API Key 被盗用，请检查自身原因，及时重置 Key。

更新

v4.2.0 新增

为 OhMyMN 提供 AI 能力。由于 OpenAI API 速度太慢，极大降低了摘录速度，违背了 OhMyMN 的初衷，所以我不会提供任何自动执行的 AI 模块。但是你可以自行编写 Prompt 手动执行。你也可以自己开发 新的模块，比如 AI 翻译。可以查看 AIAssistant。

OpenAI API 服务器地址

默认

api.openai.com

网络上有不少代理 OpenAI API 服务器，可以在国内使用。但安全性未知，不建议使用。

OpenAI 模型

- `gpt-3.5-turbo`：速度较快，Max Tokens 为 4k，也就是 4076。
- `gpt-4`：回答质量较高，速度较慢，Max Tokens 为 8k，目前还处于内测阶段，需要自行申请。
- `gpt-4-32k`：Max Tokens 为 32k，目前还没有开放使用。

token

OpenAI API 有输入+输出长度限制，也就是 Max Tokens，而且不光包括输入，还包括输出。比如 Max Tokens 为 4k，输入 2k，输出只能 2k。

一般一个汉字占 2 个 token，一个英文单词占 1 个 token。

思维发散程度

也就是 Temperature 参数，0-2，越高越发散，越低越收敛。

如果需要准确答案，建议调低。如果需要创造性答案，比如写一篇小说，可以适当调高，但不应该大于 1。

Prompts 数据源

需要填入一张卡片的链接，比如 `marginnote3app://note/BF594D1D-AC4E-46DC-8F13-87B7018E414D` 。

注意

序号为自动生成。每次修改需要重新在这里回车确认，更新数据。更新后，序号会重新生成。请不要调整卡片顺序。

你只需要按照这个格式在子卡片中填写 Prompt 即可，好的 Prompt 可以让 OpenAI 无所不能。

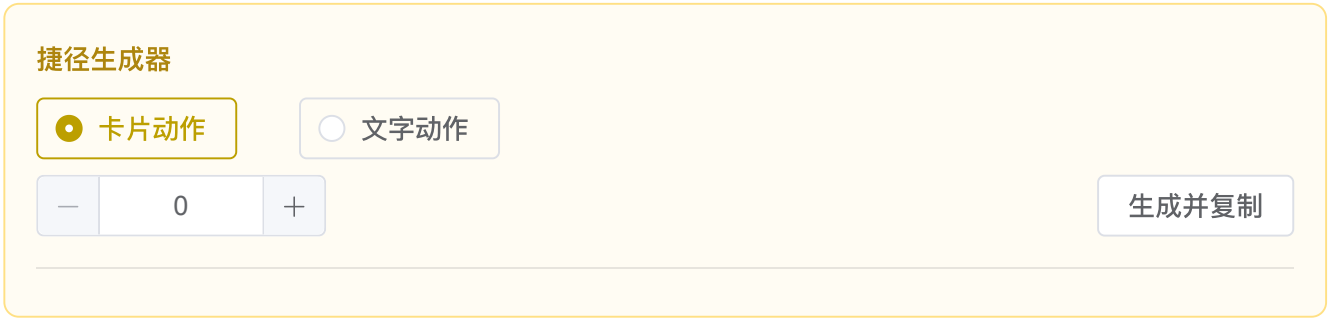
- 标题：作为 Prompt 的描述信息，选择 Prompts 的时候会显示。
- 第一条评论：注意是评论，作为 Prompt。
- 第二条评论: 参数。
  - `io`：输入输出，`title2comment` 表示将标题作为输入，拼接到 Prompt 后面，得到的结果作为评论。其他同理，excerpt 表示摘录，card 表示卡片里所有的摘录和评论。可以写多个，用 `,` 隔开。比如 `io: title2comment,excerpt2comment`。如果不填写，会在使用时手动选择。
    - `title2comment`
    - `title2title`
    - `excerpt2title`
    - `excerpt2comment`

    - `card2title`
    - `card2tag`
    - `card2comment`
    - `selected_text`：用于 MagicAction for Text 的 AI 动作 Prompts。
- `model`：模型，在这里可以设置不同的 Prompt 使用不同的模型。降低成本。
    - `gpt-3.5-turbo`
    - `gpt-4`
    - `gpt-4-32k`
- `temperature`：也就是前面提到的思维发散程度。可以根据 Prompt 填写。
- `max_tokens`：注意，这是回答的 max_tokens，和前面说的不一样。填小一点可以让回答更快，但是可能会不完整。

如何快速触发特定 Prompt

目前只能通过 Shortcut 模块的自定义捷径功能。启用 Shortcut 模块，打开自定义捷径。

在下方输入 Prompt 前的序号，序号为自动生成。

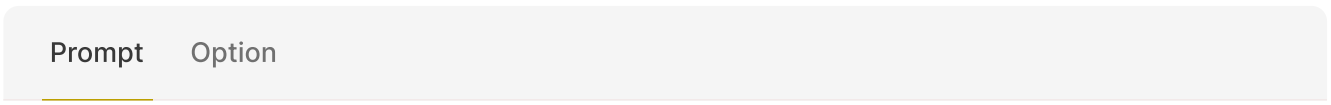


用 Raycast 或者其他快捷键设置工具，设置快捷键打开链接即可。在 iPad 上通过手势也可以打开链接。

Prompts 推荐

Prompt 推荐用英文写，AI 更容易理解。可以借鉴 Ask Prompts、Awesome ChatGPT Prompts 和 ChatGPT 中文调教指南。

1. 补全词形。用来替代 AutoComplete。并且支持更多词形，以及更多语言，自行调整即可。



```txt
Complete the word forms of this word. If the given form is not the base form
```

2. 英译中。

Prompt Option

```txt
You are a professional translation engine, please translate the text into a
```

3. 单词模式，可以更详细的翻译结果，包括：音标、词性、含义、双语示例。

Prompt Option

```txt
你是一个翻译引擎，请将翻译给到的文本，只需要翻译不需要解释。当且仅当文本只有一个单词时，请给
[<语种>] · / <单词音标>
[<词性缩写>] <中文含义>]
例句：
<序号><例句>(例句翻译)
```

4. 查询中文词组，展示多种翻译结果，并阐述适用语境。

Prompt Option

```txt
你是一个翻译引擎，请将给到的文本翻译成 English。请列出3种（如果有）最常用翻译结果：单词或
<序号><单词或短语> · /<音标>
[<词性缩写>] <适用语境（用中文阐述）>
例句：<例句>(例句翻译)
```

MagicAction for Card

基于卡片回答

基于卡片内容回答问题并生成评论，你可以用自然语言指定标题，摘录，评论以及标签。

AI 动作 (Prompts)

读取 Prompts 数据源，选择 Prompt 执行。

MagicAction for Text

AI 动作 (Prompts)

读取 Prompts 数据源，选择 Prompt 执行。

简介

首先要明白 MN 的插件底层 API 是基于 Objective-C 的，虽然看上去插件是用 JS 写，但其实只是通过 JSBridge 调用 Objective-C 的 API 来实现。

API 分为四个部分：

- OhMyMN
- MarginNote
  - High-Level API
  - Low-Level API
- Foundation
- UIKit

OhMyMN 的 API 用于 OhMyMN 插件的开发。

MarginNote 的 API 又分为 Low-Level API 和 High-Level API。

Low-Level API 实际上就是将 Objective-C 语法直接转换为 JS 语法，提供类型声明文件，使其可以在 TypeScript 中使用。High-Level API 是对 Low-Level API 的封装，使用起来更加方便。

> **TIP**
>
> 事实上，High-Level API 就是从 OhMyMN API 中抽取出来的通用部分，方便在其它插件中使用。所以目前提供的 High-Level API 并不完整，只覆盖掉了 OhMyMN 常用的部分。

Foundation 和 UIKit 是 Apple 提供的两个框架，用于开发 iOS/MacOS 应用。MN 插件对其提供有限的支持。同样属于 Objective-C Low-Level API。目前只转换了部分 API 到 TypeScript。High-Level API 其实也封装了部分 Foundation 和 UIKit 的 API。

所有的 API 代码都可以在 marinnoteapp/ohmymn/packages/api 找到。其中包括了 Objective-C 和 TypeScript 两种语言的代码。

推荐使用 VSCode 作为代码编辑器，可以检查类型，自动补全。导入 `marginnote` 包，以获得更好的开发体验。这个包里还包括了部分 Foundation 和 UIKit API。

```shell
pnpm install marginnote
```

Objective-C API 转换

MarginNote 一开始只提供了 Objective-C 版的 API，而 TypeScript 版的 API 则是通过 Objective-C API 转换而来。这里简单介绍一下转换的过程，以及注意事项。目前 UIKit 和 Foundation 的 API 只有部分有 TypeScript 版本，如果需要使用，可以参考这里的方法自行转换。也欢迎参与到 API 转换工作中来。

- Objective-C 版 API
- TypeScript 版 API

JSExport

官方文档：https://developer.apple.com/documentation/javascriptcore/jsexport

通过 JSExport 协议导出的类，会在 JavaScript 中表现为一个构造函数对象，可以通过 new 来创建实例。

对于导出的每个实例方法，会作为原型上的方法。对于导出的每个 Objective-C 属性，会作为原型上的属性。对于导出的每个类方法，会作为构造函数上的方法。

> **TIP**
>
> ES6 中的 class 其实就是构造函数的语法糖。如果对 JavaScript 原型链不熟悉的话，可以将构造函数看作对象，原型看作实例。构造函数上的方法就是对象的静态方法，原型上的方法就是对象的实例方法。
>
> 最终的转换可以理解为
>
> - OC 类 → JS 对象
> - OC 类的实例 → JS 对象的实例
> - OC 类方法 → JS 对象的静态方法
> - OC 类属性 → JS 对象的实例属性
> - OC 实例方法 → JS 对象的实例方法

而变量也会自动转换为 JavaScript 对应的类型，如下表所示：

Objective-C type	JavaScript type
nil	undefined

Objective-C type	JavaScript type
NSNull	null
NSString	string
NSNumber ,NSInteger	number
NSDictionary	Object, {}
NSArray	Array
NSDate	Date
JSValue	Function
id	any，调试查看
Class	构造函数

下面的代码示例说明如何采用 JSExport 协议:

```c
#import "MbBookNote.h"

@import Foundation;
@import JavaScriptCore;
@class MbTopic;
@class MbBook;
// 表示将 JSBMbBookNote 类及其实例方法、类方法和属性导出到 JavaScript 代码。
@protocol JSBMbBookNote <JSExport>

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wdeprecated-declarations"

// 属性，可读可写
@property (nonatomic,readwrite,getter=highlight_text,setter=_setHighlightTex

// 属性，只可读
@property (nonatomic,readonly,getter=noteid) NSString * noteId;

// 实例方法，返回值类型为 NSString，没有参数
- (NSString*) allNoteText;

// JSExportAs 表示重命名导出
// 如果没有重命名，导出的方法名应该为 appendHtmlCommentTextTag，这种有多个参数的函数可
JSExportAs(appendHtmlComment,
- (void)appendHtmlComment:(NSString*)html text:(NSString*)text tag:(NSStrin
// 实例方法，没有返回值，参数有 text，类型为 NSString
- (void)appendTextComment:(NSString*)text;

// 类方法，前面是 + 号。在 JS 中的方法名应该是 createWithTitleNotebookDocument。返
+ (MbBookNote *)createWithTitle:(NSString *)title notebook:(MbTopic*)topic c

#pragma clang diagnostic pop
@end
```

转换到 TypeScript 之后，会变成这样：

```ts
declare class MbBookNote {
  excerptText: string
  readonly noteId: string
  allNoteText(): string
  appendHtmlComment(html: string, text: string, tag: string): void
  appendTextComment(text: string): void
}

declare global {
  const Note: {
    createWithTitleNotebookDocument(
      title: string,
      notebook: MbTopic,
      doc: MbBook
    ): MbBookNote
  }
}
```

这样做的好处是 MbBookNote 可以作为类型进行导出和导入，而不是作为全局变量，避免全局变量满天飞。而 Note 则是作为全局变量，可以直接使用。会使用到静态方法的对象都必须作为全局变量。就算这里不是 Note，而是 MbBookNote，也仍然需要这样。实际上，所有的 low-level API 都是全局变量，运行时会被自动注入。

⚠ 注意

1. 创建实例
   - 如果有类方法，可以直接使用类方法创建实例。比如上面的 `Note.createWithTitleNotebookDocument()`
   - 大多数时候会有一个默认的 `.new()` 的类方法。比如 `NSNull.new()`
   - 如果 `.new()` 不行，可以使用 new 关键字。如 `new UISwitch()`，并且 UISwitch 必须作为全局变量。
2. 要导入类型，必须使用 `import type`，而不是 `import`。尤其是存在同名全局变量时，如果不加 type，会导致类型错误。

```
import type { MbBookNote } from 'marginnote'
```

3. 多个参数的函数名称。 JSExportAs 表示重命名导出。如果没有重命名，导出的方法名应该为 appendHtmlCommentTextTag，这种有多个参数的函数可以理解为将函数名拆开，逐个解释其参数。你只需要将冒号前的参数名拼接起来，并且首字母大写，就是导出的方法名。

```c
JSExportAs(appendHtmlComment,
- (void)appendHtmlComment:(NSString*)html text:(NSString*)text tag:(NSStr
```

4. 多个参数的函数，参数名有可能写全了不一定能运行，有时候少一个参数才能运行，需要自己尝试。

MN

```ts
import { MN } from "marginnote"
```

这可能会是你在开发 MarginNote 插件时最常用的一个变量。

有以下属性

app

Application 实例。

```ts
readonly app = Application.sharedInstance()
```

db

Database 实例。

```ts
readonly db = Database.sharedInstance()
```

currentWindow

当前窗口。

```ts
get currentWindow() {
  return this.app.focusWindow
}
```

studyController

当前窗口的 studyController。

```ts
get studyController() {
  return this.app.studyController(this.app.focusWindow)
}
```

notebookController

当前窗口的 notebookController

```ts
  get notebookController() {
    return this.studyController.notebookController
  }
```

currentDocumentController

当前窗口的 currentDocumentController

```ts
  get currentDocumentController() {
    return this.studyController.readerController.currentDocumentController
  }
```

currentNotebookId(){

当前笔记本的 id

```ts
  get currnetNotebookId() {
    return this.notebookController?.notebookId
  }
```

}

currentDocmd5

当前文档的 md5。如果笔记本中没有文档，也会随机产生一个 32 位的 md5，我在这里让这个值固定，8 个 0，避免生成无效配置。

```ts
get currentDocmd5() {
  try {
    const { docMd5 } = this.currentDocumentController
    if (docMd5 && docMd5.length === 32) return "00000000"
    else return docMd5
  } catch {
    return undefined
  }
}
```

currentAddon

从 `self.addon` 中获取当前插件的信息。需要手动写入到 `self.addon` 中。默认值为 `mnaddon` 和 `MarginNote`。通常是在 sceneWillConnect 阶段写入。

```ts
get currentAddon(): {
  key: string
  title: string
} {
  return {
    key: self.addon?.key ?? "mnaddon",
    title: self.addon?.title ?? "MarginNote"
  }
}
```

isZH

软件语言是否为中文

```ts
readonly isZH = NSLocale.preferredLanguages()?.[0].startsWith("zh")
```

version

版本号

```ts
readonly version = this.app.appVersion ?? "3.7.21"
```

isMNE, isMac, isMacMNE, isMacMN3

不同版本判断

```ts
readonly isMNE = gte(this.version, "4.0.0")
readonly isMac = this.app.osType === OSType.macOS
readonly isMacMNE = this.isMac && gte(this.version, "4.0.2")
readonly isMacMN3 = this.isMac && !this.isMacMNE
```

currentThemeColor

当前主题色

```ts
get currentThemeColor(): UIColor {
  return this.themeColor[this.app.currentTheme]
}
```

themeColor

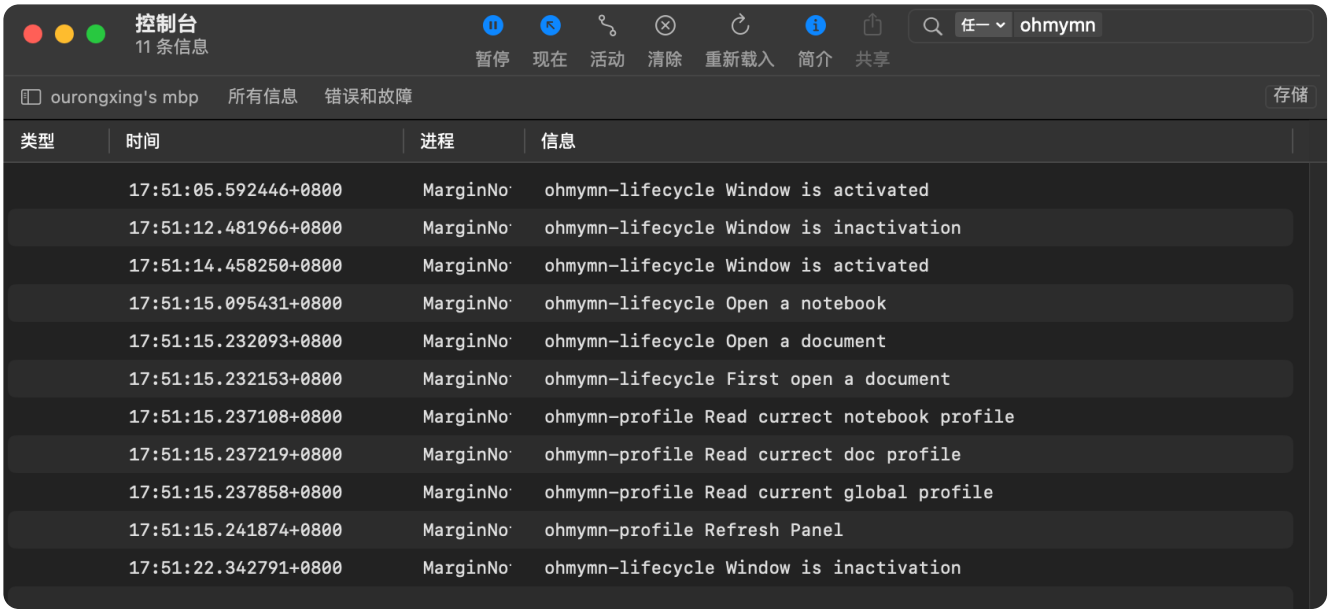
```ts
readonly themeColor = {
  Gray: UIColor.colorWithHexString("#414141"),
  Default: UIColor.colorWithHexString("#FFFFFF"),
  Dark: UIColor.colorWithHexString("#000000"),
  Green: UIColor.colorWithHexString("#E9FBC7"),
  Sepia: UIColor.colorWithHexString("#F5EFDC")
}
```

---

log, error

```ts
/**
 * You need watch log in console.app not in Browser
 * @param obj any object
 * @param suffix default is "normal"
 * @param args any params
 */
log(obj: any, suffix?: string, ...args: any[]): void;
/**
 * You need watch log in console.app not in Browser
 * @param obj any object
 * @param suffix default is "error"
 * @param args any params
 */
error(obj: any, suffix?: string, ...args: any[]): void;
```

用来调试，可以理解为 `console.log` 和 `console.error`。但是你需要在 `Console.app` 中查看，而不是在浏览器中。



通过插件 key 筛选即可。`suffix` 参数默认为 `normal` 和 `error`，也是为了方便筛选。OhMyMN 里有大量的调试输出。

其实也有办法在浏览器里调试，可以查看 这篇文章。并且将 `self.useConsole` 设置为 `true`，这样就可以在浏览器中调试了。可以尝试，但不一定有效。

开发相关

一些简化开发的工具方法。主要为了 TypeScript 类型检查。

---

defineLifycycleHandlers

```ts
declare function defineLifecycleHandlers(events: {
    instanceMethods?: Partial<JSExtensionLifeCycle.InstanceMethods>;
    classMethods?: Partial<JSExtensionLifeCycle.ClassMethods>;
}): {
    instanceMethods?: Partial<Partial<{
        sceneWillConnect(): void;
        sceneDidDisconnect(): void;
        sceneWillResignActive(): void;
        sceneDidBecomeActive(): void;
        notebookWillOpen(notebookid: string): void;
        notebookWillClose(notebookid: string): void;
        documentDidOpen(docmd5: string): void;
        documentWillClose(docmd5: string): void;
    }>> | undefined;
    classMethods?: Partial<Partial<{
        addonDidConnect(): void;
        addonWillDisconnect(): void;
        applicationDidEnterBackground(): void;
        applicationWillEnterForeground(): void;
        applicationDidReceiveLocalNotification(notify: any): void;
    }>> | undefined;
};
```

提供生命周期函数的类型声明。

举个例子：

```ts
import { MN, defineLifycycleHandlers } from "marginnote";

export default defineLifecycleHandlers({
  instanceMethods: {
    sceneWillConnect() {
      MN.log("sceneDidBecomeActive");
    },
    notebookWillOpen(notebookid) {
      MN.log(notebookid);
    }
  },
});
```

defineEventHandlers

```ts
declare function defineEventHandlers<K extends string>(h: {
    [M in K extends `on${string}` ? K : `on${K}`]: (sender: {
        userInfo: UserInfo<M>;
    }) => void;
}): { [M in K extends `on${string}` ? K : `on${K}`]: (sender: {
    userInfo: UserInfo<M>;
}) => void; };

type UserInfo<K> = K extends
  | "onPopupMenuOnSelection"
  | "onClosePopupMenuOnSelection"
  ? {
      winRect: string
      arrow: DirectionOfSelection
      documentController: DocumentController
    }
  : K extends "onPopupMenuOnNote" | "onClosePopupMenuOnNote"
  ? { note: MbBookNote }
  : K extends "onChangeExcerptRange" | "onProcessNewExcerpt"
  ? { noteid: string }
  : K extends "onAddonBroadcast"
  ? { message: string }
  : Record<string, any>
```

提供事件处理函数的类型声明，进行类型检查。

举个例子：

```ts
const events = [
  "ClosePopupMenuOnNote",
  "ClosePopupMenuOnSelection"
] as const

export default defineEventHandlers<(typeof events)[number]>({
   onClosePopupMenuOnNote(sender) { }
   onClosePopupMenuOnSelection(sender) { }
})
```

---

eventObserverController

```ts
declare function eventObserverController(observers: ({
    event: string;
    handler?: string;
} | string)[]): {
    add: () => void;
    remove: () => void;
};
```

事件监听控制器，可以用于添加和移除监听。

```ts
const events = [
  "ClosePopupMenuOnNote",
  "ClosePopupMenuOnSelection"
] as const

export const eventObservers = eventObserverController(events)

// lifecycle.ts
notebookWillOpen(){
    eventObservers.add()
}
notebookWillClose(notebookid: string) {
    eventObservers.remove()
}
```

---

i18n

```ts
declare function i18n<M, N>(lang: {
    zh: M;
    en: N extends M ? M : M;
}): M;
```

提供国际化的支持，根据系统语言自动切换。目前支持中文和英文，提供类型检查。

举个例子：

```ts
import { i18n } from "marginnote"

export default i18n({
  zh: {
    none: "无",
  },
  en: {
    none: "None",
  }
})
```

getObjCClassDeclar

```ts
/**
 * @param name Objective-C Class name
 * @param type Objective-C Class type
 */
declare function getObjCClassDeclar(name: string, type: string): string;
```

提供 JSB 类的声明，用于 JSB.defineClass 的第一个参数。 举个例子，OhMyMN 就是这样定义的

```ts
JSB.defineClass(getObjCClassDeclar("OhMyMN", "JSExtension"), {}, {})
```

实际上返回的值就是 `OhMyMN: JSExtension` 。

笔记相关

Code

与笔记相关的有两个比较重要的类，

- MbBookNote: MarginNote 里的笔记都属于 MbBookNote。MbBookNote 一定是笔记，但不一定是一张卡片。
- NodeNote: 对 MbBookNote 的扩展，NodeNote 一定是一张卡片。

除了这两个外，还有一些封装的工具，方便对笔记进行操作或读取。

---

undoGroupingWithRefresh

```ts
declare function undoGroupingWithRefresh(f: () => void): void;
```

所有对笔记的操作都应该用这个函数包裹，包裹在里面的操作可以被撤销。同时会刷新界面，呈现修改后的效果。

> **注意**
>
> 这个函数里不能有异步操作，必须提到外面。

举个例子

```ts
const { value } = await select([1,2,3])
undoGroupingWithRefresh(()=>{
  note.title = value
})
```

---

isNoteExist

```ts
/**
 * @param note MbBookNote or noteid
 */
declare function isNoteExist(note: MbBookNote | string): boolean;
```

判断笔记是否存在。可以传入一个笔记或者笔记的 id。因为有可能这个笔记被删除了。

---

isNoteLink

```ts
/**
 * @param url note link
 */
declare function isNoteLink(url: string): boolean;
```

判断一个 url 是否是笔记链接。笔记链接通常是 `marginnote3app://note/xxxx` 这样的。

---

removeHighlight

```ts
declare function removeHighlight(text: string): string;
```

MarignNote 里的高亮实际上是 `**内容**`，获取到的摘录内容里会包含这些高亮。这个函数可以把 `*` 删掉。

NodeNote

Code

NodeNote 是对 MbBookNote 的扩展，用于表示脑图中的一个节点，或者一张卡片。

一张卡片里实际上可能存在多个 MbBookNote，比如你合并里两张卡片。其中一个 MbBookNote 可以拥有控制这张卡片的能力，而另一个 MbBookNote 就只能作为评论存在。

不管你传入的是哪一个 MbBookNote，NodeNote 都可以控制这张卡片。

```ts
import { NodeNote } from "marginnote"

const node = new NodeNote(note)
```

静态方法

getSelectedNodes

这是 NodeNote 的静态方法，用于获取脑图中选中的卡片。

```ts
const nodes = NodeNote.getSelectedNodes()
```

---

getter

getter 不同于函数，不用加括号，直接调用就行。可以看作一个 readonly 的属性。

descendantNodes

用来获取当前卡片的后代卡片，返回所有的后代卡片和后代卡片的树状结构。

所谓后代卡片，就是当前卡片的子卡片，子卡片的子卡片，子卡片的子卡片的子卡片，以此类推。

treeIndex 为二维数组，第一个数字表示第几层，第二个数字表示这一层的第几个。

```ts
const { descendant, treeIndex } = node.descendantNodes
```

OhMyMN 的 分层编号 就用到了这个 treeIndex。

ancestorNodes

获取当前卡片的祖先卡片。祖先卡片就是父卡片，父卡片的父卡片，父卡片的父卡片的父卡片，以此类推。

childNodes, parentNode

获取当前卡片的子卡片和父卡片。

notes

获取当前卡片的所有 MbBookNote。

isOCR

获取当前卡片是否被 OCR。一旦被 OCR，之后合并进入的卡片都会被 OCR。因为这个值是绑定到卡片上的。

excerptsText, excerptsTextPic

```ts
get excerptsText(): string[]
get excerptsTextPic(): {
    ocr: string[]
    html: string[]
    md: string[]
}
```

获取从 PDF 中摘录的文字，如果是图片，会被 OCR，也可以将图片转为 base64，然后返回 html 和 md 格式的图片。如果不需要图片，可以直接用 `excerptsText`。

注意这个返回的都是数组，因为一个卡片可能有多个摘录。

commentsText, commentsTextPic

```ts
    get commentsText(): string[]
    get commentsTextPic(): {
        html: string[]
        md: string[]
    }
```

同理，获取卡片中的评论。但是评论里的图片无法被 OCR。

allText, allTextPic, excerptsCommentsText

```ts
    get excerptsCommentsText(): string[];
    get allText(): string
    get allTextPic(): {
        html: string
        ocr: string
        md: string
    }
```

相当于 `excerptsText` 和 `commentsText` 加起来。他们的顺序就是卡片里的顺序。注意，`allText` 直接返回的字符串，而不是数组。他们之间用 `\n` 合并。如果需要返回数组，可以用 `excerptsCommentsText`。

---

setter

有 setter 也会有 getter，所以上面的 getter 没有列举完。留到这里一起说。

setter 不同于函数，可以直接赋值。可以看作一个 writable 的属性。不过写入和读取的值不一样。

title, titles

```ts
    get titles(): string[]
    set titles(titles: string[])
    get title(): string
    set title(title: string)
```

`title` 是直接返回卡片的标题，而 `titles` 会将标题按 `;` 分割成数组返回。设置标题会覆盖之前的标题，如果要追加标题，可以用 `appendTitles` 。

```ts
note.title = "a;b;c"
note.titles // ["a", "b", "c"]
note.titles = ["e", "f", "g"]
note.title // "e;f;g"
```

mainExcerptText

```ts
get mainExcerptText(): string;
set mainExcerptText(text: string);
```

设置卡片的摘录内容。这里的主摘录可以看作卡片的第一个摘录。

tags

```ts
get tags(): string[];
set tags(tags: string[]);
```

设置标签，会覆盖之前的标签，不需要 `#` 。如果要追加标签，可以用 `appendTags` 。

```ts
note.tags = ["a", "b", "c"]
note.tags // #a #b #c
note.tags = ["d", "e", "f"]
note.tags // #d #e #f
```

实例方法

appendTitles

```ts
appendTitles(...titles: string[]): this;
```

追加标题，会在原来的标题后面加上新的标题。这个方法可以传入任意数量个参数。

```ts
node.appendTitles("a")
node.appendTitles("b", "c")
```

> TIP
>
> 这个方法返回的是 `this`，也就是当前的卡片，所以可以链式调用。

appendTags

```ts
appendTags(...tags: string[]): this;
```

同上，追加标签。不需要 #，会自动加上。

appendTextComments

```ts
appendTextComments(...comments: string[]): this;
```

同上，追加评论。

tidyupTags

```ts
tidyupTags(): this;
```

整理标签，会去掉重复的标签，并且统一放置在最后。

getCommentIndex

```ts
getCommentIndex(comment: MbBookNote | string): number;
```

获取评论在卡片中的位置，从 0 开始，如果没有找到，返回 -1。 这里可以传入评论的内容，也可以传入合并的笔记。合并的笔记也会作为评论，所以也可以用这个方法获取合并的笔记在卡片中的位置。

removeCommentButLinkTag

```ts
/**
 * @param filter not deleted
 * @param f call a function after deleted, before set tag and link
 */
removeCommentButLinkTag(filter: (comment: NoteComment) => boolean, f?: (node
```

作用是移除除了 filter 筛选出的评论之外的所有评论。移除后执行 f 函数，然后重新添加标签和链接，使其出现在最后。

如何使用：

```ts
// 这个的作用是保留卡片中的 PaintNote 和 HtmlNote，其他的都删除。然后添加新的评论。
node.removeCommentButLinkTag(
    k =>
        k.type === "PaintNote" ||
        k.type === "HtmlNote",
    n => {
        n.appendTextComments(...comments)
    }
)
```

弹窗

Code

---

showHUD

在屏幕上临时显示一条消息。

```ts
/**
 * @param message
 * @param duration default `2`, unit `s`
 * @param window  default `MN.currentWindow`
 */
declare function showHUD(message: string, duration?: number, window?: any):
```

HUDController

在屏幕上显示一条消息，并且可以控制何时消失。

```ts
declare const HUDController: {
    /**
     * @param message
     * @param window default `MN.currentWindow`
     */
    show(message: string, window?: any): void;
    /**
     * @param message If set, temporarily shows message on the screen when t
     * @param duration default `2`, unit `s`
     * @param window  default `MN.currentWindow`
     */
    hidden(message?: string, duration?: number, window?: any): void;
};
```

如何使用：

```ts
import { HUDController } from "marginnote"
HUDController.show("Loading...")
// do something
HUDController.hidden("Done")
```

Alert

显示一个提示框，需要点击才能关闭。但是无法获取结果，如果获取点击结果，可以使用 confirm。

```ts
declare function alert(message: string): void;
```

popup

弹出对话框，输入框，选择框等。

```ts
declare function popup<T>({ title, message, type, buttons, canCancel, multiL
    title?: string;
    message: string;
    buttons?: string[];
    type?: UIAlertViewStyle;
    canCancel?: boolean;
    multiLine?: boolean;
}): Promise<{
    inputContent: string | undefined;
    buttonIndex: number;
}>;
```

参数，是一个 options 对象，包含以下属性：

- `title` 默认 `MN.currentAddon.title`
- `message`
- `type` 默认 `UIAlertViewStyle.Default`，只有按钮.如果需要输入文字，请使用 `UIAlertViewStyle.PlainTextInput` 或者 `UIAlertViewStyle.`, 但是现在不支持

`UIAlertViewStyle.LoginAndPasswordInput`.

- `buttons` 默认 `[lang.sure]`，也就是确定。
- `canCancel` 默认 `true`, 如果为 `false`，你必须选择一个选项。
- `multiLine` 默认 `false`, 如果为 `true`，按钮文本将移除 `\n`，并被减少到 40 个字节。

UIAlertViewStyle:

```ts
export const enum UIAlertViewStyle {
  /**
   * The default alert view style. The default.
   */
  Default,
  /**
   * Allows the user to enter text, but the text field is obscured.
   */
  SecureTextInput,
  /**
   * Allows the user to enter text.
   */
  PlainTextInput,
  /**
   * Allows the user to enter a login id and a password.
   * @deprecated not support yet
   */
  LoginAndPasswordInput
}
```

返回值：

- `inputContent` 输入框里的输入，如果 `type` 为 `UIAlertViewStyle.Default` 返回 `undefined`。
- `buttonIndex` 点击的按钮的索引。如果取消该对话框，它将为 -1。

如何使用:

```ts
import { popup } from "marginnote"
async function confirm(title = MN.currentAddon.title, message = "") {
  const { buttonIndex: option } = await popup({
    title,
    message,
    buttons: [lang.sure],
    multiLine: false,
    canCancel: true
  })
  return option === 0
}
```

confirm

弹出一个确认对话框，并返回结果。

```ts
/**
 * @param title default `MN.currentAddon.title`
 * @param message optional
 */
declare function confirm(title?: string, message?: string): Promise<boolean>
```

select

弹出一个选项列表对话框，并返回选定的值。

```ts
/**
 * @param title default `MN.currentAddon.title`
 * @param message optional
 * @param canCancel default `false`
 */
declare function select(options: string[], title?: string, message?: string,
    value: string;
    index: number;
}>;
```

返回值:

- `value` 选择的值
- `index` 选择的值的索引

如何使用:

```ts
const { value, index } = await select(['a', 'b', 'c'])
```

网络请求

Code

```ts
import { fetch } from "marginnote"
```

在插件里是无法使用 JS 的 fetch 方法的。我利用 Objective-C 的 NSURLConnection 实现了一个 fetch 方法，用法和 JS 的 fetch 一样。但也有不少问题。

```ts
declare type RequestOptions = {
    method?: "GET" | "POST" | "PATCH";
    timeout?: number;
    headers?: Record<string, any>;
} & XOR<XOR<{
    body?: string;
}, XOR<{
    json?: Record<string, any>;
}, {
    form?: Record<string, string | number | boolean>;
}>>, {
    search?: Record<string, string | number | boolean>;
}>;
declare function fetch(url: string, options?: RequestOptions): Promise<Respo
```

问题

1. 只能解析 JSON 的返回数据。
2. 无法获取到 status code。

示例

GET

```ts
const res = await fetch("http://dict.e.opac.vip/dict.php?sw=" + word).then(
  res => res.json()
)
```

POST JSON

```ts
const res = (await fetch(
  "http://api.interpreter.caiyunai.com/v1/translator",
  {
    method: "POST",
    headers: {
      "X-Authorization": `token ${caiyunToken}`
    },
    json: {
      source: [text],
      trans_type: `${fromLangKey[fromLang]}2${toLangKey[toLang]}`,
      request_id: "ohmymn",
      detect: true
    }
  }
).then(res => res.json())) as {
  target: string[]
}
```

POST form

```ts
const res = (await fetch(
  `https://aip.baidubce.com/rest/2.0/ocr/v1/handwriting?access_token=${toker
  {
    method: "POST",
    headers: {
      "Content-Type": "application/x-www-form-urlencoded"
    },
    form: {
      image: imgBase64
    }
  }
).then(res => res.json())) as {
  words_result: { words: string }[]
} & BaiduOCRError
```

等待/间隔

Code

---

delay

等待一段时间，单位是秒

```ts
/**
 * @param sec unit `s`
 */
declare function delay(sec: number): Promise<NSTimer>;
```

如何使用

```ts
import { delay } from "marginnote"
await delay(1)
// do something after 1 second
```

---

loopBreak

每隔一段时间执行一次 `condition` 函数，执行 `times` 次，如果 `condition` 返回 `true`，则停止执行，立即返回 `true`，否则执行 `times` 后返回 `false`。

这个方法的作用是尽可能缩短等待时间，不断地检查某个条件是否满足，如果满足则立即返回，否则继续等待。这个执行次数其实就相当于超时。

```ts
/**
 * @param times loop times
 * @param sec unit `s`, duration of each loop
 * @param condition end condition
 * @returns `true` if `condition` return `true`, otherwise `false`
 */
declare function loopBreak(times: number, sec: number, condition: () => bool
```

setTimeInterval

每隔一段时间执行一次 `f` 函数，需要手动停止。返回一个 `Timer` 对象，调用 `invalidate` 方法停止定时器。

通常用于后台执行，不打断用户操作，比如每隔一段时间保存一次笔记。

```ts
/**
 * @param sec unit `s`
 * @param f
 * @returns Timer, call `invalidate` to stop this timer
 */
declare function setTimeInterval(sec: number, f: () => any): Promise<Timer>;

declare type Timer = {
    /**
     * Stop this timer
     */
    invalidate: () => void;
};
```

文件操作

Code

一些文件操作

Path

一些常用的可读写的文件夹位置

- `MN.app.cachePath`：缓存文件夹。
- `MN.app.tempPath`：临时文件夹，在退出后随时有可能被删除。
- `Addon.path`：当前插件所在的文件夹。

要想临时写入文件，比如需要 AirDrop 分享，可以使用 `MN.app.tempPath`。

> **注意**
>
> 目前删除文件有问题，无法执行。

---

isfileExists

文件是否存在。

```ts
declare function isfileExists(path: string): boolean;
```

---

copyFile

复制文件，返回复制是否成功。

```ts
declare function copyFile(src: string, dest: string): boolean;
```

writeTextFile

写入文字到文件，如果文件不存在则创建，如果文件存在则覆盖。

```ts
declare function writeTextFile(path: string, text: string): void;
```

readJSON

读取 JSON 文件。

```ts
declare function readJSON(path: string): any;
```

writeJSON

写入 JSON 到文件。

```ts
declare function writeJSON(path: string, data: any): void;
```

saveFile

交互式保存文件，需要用户选择保存位置。iOS 上使用分享面板。

> **TIP**
>
> UTI 就是 Uniform Type Identifier，比如 `public.plain-text`，`public.png`，`public.jpeg`，`public.zip` 等等。用来标明文件类型。

```ts
/**
 * @param file file path
 * @param UTI
 */
declare function saveFile(file: string, UTI: string): void
```

saveTextFile

交互式保存文本文件，需要用户选择保存位置。iOS 上使用分享面板。只要是纯文本，都可以使用这个方法保存。指定文件后缀和 UTI 即可。

```ts
/**
 * Create a text file with given content, default .txt file
 * @param UTI @default public.plain-text
 */
declare function saveTextFile(
  content: string,
  fileName: string,
  UTI?: string
): void
```

openFile

从文件管理器打开文件，可以多选。iOS 上不可用。

```ts
/**
 * @warning not working on iPad
 */
declare function openFile(...uti: string[]): Promise<string | undefined>
```

其它

都是一些常用的方法，进行一些简单的封装。

---

setLocalDataByKey, getLocalDataByKey

```ts
export function getLocalDataByKey(key: string) {
  return NSUserDefaults.standardUserDefaults().objectForKey(key)
}
export function setLocalDataByKey(data: any, key: string) {
  NSUserDefaults.standardUserDefaults().setObjectForKey(data, key)
}
```

持久化数据，也就是存到本地。但这并不是文件。可以直接存储 JSON 对象，不需要转换成字符串。但注意不要有 undefined，否则会报错。

---

genNSURL, openURL

```ts
export function genNSURL(url: string, encode = false) {
  url = url.trim()
  if (!/^[\w\-]+:\/\//.test(url)) url = `https://${url}`
  return NSURL.URLWithString(encode ? encodeURI(url) : url)
}

export function openURL(url: string, encode = false) {
  MN.app.openURL(genNSURL(url, encode))
}
```

打开链接，需要构建 NSURL 对象。如果链接不是以协议开头，会自动加上 https://。如果链接中有中文，需要设置 encode=true。

---

postNotification

```ts
export function postNotification(key: string, userInfo: any) {
  NSNotificationCenter.defaultCenter().postNotificationNameObjectUserInfo(
    key,
    self,
    userInfo
  )
}
```

发送事件通知，key 为事件名，userInfo 传递的信息，可以是任意类型。

在其它插件或者类中，可以通过 监听事件 来获取信息。在 OhMyMN，控制面板的触发动作就是通过这个方法获取到的。

---

copy

```ts
export function copy(text: string, hud = true) {
  if (text) {
    UIPasteboard.generalPasteboard().string = text.trim()
    hud && showHUD(lang.copy_success)
  } else hud && showHUD(lang.copy_empty)
}
```

复制文字到剪贴板上，如果 hub 为 true，会显示一个复制提示。

---

isNSNull, NSNull2Null

```ts
export function isNSNull(obj: any): obj is NSNull {
  return obj === NSNull.new()
}
export function NSNull2Null<T>(k: T) {
  return isNSNull(k) ? null : (k as Exclude<T, NSNull>)
}
```

NSNull 和 null 是不同的，在 fetch 中，如果返回的数据中有 null，会被转换成 NSNull。这个方法可以将 NSNull 转换成 null。

NSValue2String

```ts
export function NSValue2String(v: NSValue) {
  return Database.transArrayToJSCompatible([v])[0] as string
}
```

NSValue 是一种数据类型，可以存储任意类型的数据。但是在 JS 中，无法直接使用，需要转换成字符串。这个方法可以将 NSValue 转换成字符串。

CGRectString2CGRect, CGSizeValue2CGSize

```ts
/**
 * @param str string like "{x,y}, {h,w}"
 */
export function CGRectString2CGRect(str: string): CGRect {
  const arr = str.match(/\d+\.?\d+/g)!.map(k => Number(k))
  return {
    x: arr[0],
    y: arr[1],
    height: arr[2],
    width: arr[3]
  }
}
export function CGRectValue2CGRect(v: NSValue) {
  return CGRectString2CGRect(NSValue2String(v))
}
```

将 CGRect 类型的 NSValue 转换成 JS 可读的 CGRect 对象。

CGSizeString2CGSize, CGSizeValue2CGSize

```ts
/**
 * @param str string like "{x,y}"
 */
export function CGSizeString2CGSize(str: string): CGSize {
  const arr = str.match(/\d+\.?\d+/g)!.map(k => Number(k))
  return {
    width: arr[0],
    height: arr[1]
  }
}
export function CGSizeValue2CGSize(v: NSValue) {
  return CGSizeString2CGSize(NSValue2String(v))
}
```

将 CGSize 类型的 NSValue 转换成 JS 可读的 CGSize 对象。

Application

```ts
const Application: {
  /**
   * Create an Application instance
   * @recommended use {@link MN.app}
   */
  sharedInstance(): Application
}
```

```ts
export declare type Application = {
  /**
   * @value 4.0.2(97)
   *
   * 4.0.2 is version, 97 is build num
   * @recommended use {@link MN.version}
   */
  readonly appVersion: string
  /**
   * @value 4.0.2(97)
   */
  readonly build: string
  /**
   * Current theme
   * @recommended use {@link MN.currentThemeColor}
   */
  readonly currentTheme: "Gray" | "Default" | "Dark" | "Green" | "Sepia"
  /**
   * default tint color for dark background
   */
  readonly defaultTintColorForDarkBackground?: UIColor
  /**
   * default tint color for selected
   */
  readonly defaultTintColorForSelected?: UIColor
  /**
   * default tint color
   */
  readonly defaultTintColor?: UIColor
  /**
   * default book page color
   */
  readonly defaultBookPageColor?: UIColor
  /**
   * default note book color
   */
  readonly defaultNotebookColor?: UIColor
  /**
   * default text color
   */
  readonly defaultTextColor?: UIColor
  /**
```

```
     * default disable color
     */
    readonly defaultDisableColor?: UIColor
    /**
     * default highlight blend color
     */
    readonly defaultHighlightBlendColor?: UIColor
    /**
     * Focus window
     */
    readonly focusWindow?: UIWindow
    /**
     * Database path
     */
    readonly dbPath?: string
    /**
     * Document relative path
     */
    readonly documentPath?: string
    /**
     * Cache path
     */
    readonly cachePath?: string
    /**
     * Temp path
     */
    readonly tempPath?: string
    /**
     * OS type
     */
    readonly osType: OSType
    /**
     * Refresh Note data
     */
    refreshAfterDBChanged(notebookid: string): void
    queryCommandWithKeyFlagsInWindow(
      command: string,
      keyFlags: number,
      window: UIWindow
    ): DictObj
    processCommandWithKeyFlagsInWindow(
      command: string,
      keyFlags: number,
```

```
    window: UIWindow
  ): void
  /**
   * @recommand {@link openURL}
   */
  openURL(url: NSURL): void
  /**
   * @recommand {@link alert}
   */
  alert(message: string): void
  /**
   * @recommand {@link showHUD}
   */
  showHUD(message: string, view: UIView, duration: number): void
  /**
   * @recommand {@link HUDController}
   */
  waitHUDOnView(message: string, view: UIView): void
  /**
   * @recommand {@link HUDController}
   */
  stopWaitHUDOnView(view: UIView): void
  /**
   * @recommand {@link saveFile}
   */
  saveFileWithUti(mfile: string, uti: string): void
  /**
   * @recommand {@link MN.studyController}
   */
  studyController(window: UIWindow): StudyController
  /**
   * Check the notify sender is current window.
   * @param obj Usually sender
   * @param window
   * @recommand MN.currentWindow === window
   */
  checkNotifySenderInWindow(obj: any, window: UIWindow): boolean
  /**
   * @recommand {@link openFile}
   */
  openFileWithUTIs(
    types: string[],
    controller: UIViewController,
```

```
      callback: (file: string) => void
    ): void

    /**
     * Register a html comment editor
     * @param commentTag The markdown editor plugin id
     * @see "https://github.com/marginnoteapp/milkdown/blob/main/src/jsExtensi
     */
    regsiterHtmlCommentEditor(
      commentEditor: DictObj,
      htmlEditor: JSValue,
      htmlRender: JSValue,
      commentTag: string
    ): void

    /**
     * Unregister a html comment editor
     * @param commentTag The markdown editor plugin id
     */
    unregsiterHtmlCommentEditor(commentTag: string): void
  }
```

Database

从全局读取数据，是对数据库操作的封装。

```ts
const Database: {
  /**
   * @recommended use {@link MN.db}
   */
  sharedInstance(): MbModelTool
  /**
   * Transfrom unreadable OC dictionary to JS compatible
   */
  transDictionaryToJSCompatible(dic: any): any
  /**
   * Transfrom unreadable OC array to JS compatible
   */
  transArrayToJSCompatible(arr: any): any
}
```

```ts
export declare class MbModelTool {
  getNotebookById(notebookid: string): MbTopic | undefined
  getMediaByHash(hash: string): NSData | undefined
  getNoteById(noteid: string): MbBookNote | undefined
  getDocumentById(md5: string): MbBook | undefined
  /**
   * Get note in review mode
   */
  getFlashcardByNoteId(
    noteid: string,
    notebookid: string
  ): MbBookNote | undefined
  /**
   * Get notes in review mode
   */
  getFlashcardsByNoteId(noteid: string): MbBookNote[] | undefined
  /**
   * Whether has note in review mode
   */
  hasFlashcardByNoteId(noteid: string): boolean
  savedb(): void
  /**
   * Fetch all notebooks
   */
  allNotebooks(): MbTopic[]
  /**
   * Fetch all documents
   */
  allDocuments(): MbBook[]
  setNotebookSyncDirty(notebookid: string): void
  saveHistoryArchiveKey(notebookid: string, key: string): any[]
  loadHistoryArchiveKey(notebookid: string, key: string): any[]
  deleteBookNote(noteid: string): void
  /**
   * Delete note and its all descendant notes
   */
  deleteBookNoteTree(noteid: string): void
  /**
   * Clone notes to a notebook, and return the cloned notes
   */
  cloneNotesToTopic(notes: MbBookNote[], notebookid: string): MbBookNote[]
  cloneNotesToFlashcardsToTopic(
```

```
    notes: MbBookNote[],
    notebookid: string
  ): MbBookNote[]
  exportNotebookStorePath(notebookid: string, storePath: string): boolean
  importNotebookFromStorePathMerge(storePath: string, merge: boolean): any
  /**
   * Get handwriting notes in notebook mindmap
   */
  getSketchNotesForMindMap(notebookid: string): MbBookNote[]
  getSketchNotesForDocumentMd5Page(
    notebookid: string,
    docmd5: string,
    page: number
  ): MbBookNote[]
}
```

SQLiteDatabase

如果你想直接访问数据库，十分不建议，数据库数据结构非常复杂，而且直接修改容易损坏数据库。

```ts
declare global {
  const SQLiteDatabase: {
    databaseWithPath(path: string): SQLiteDatabase
  }
}

export declare type SQLiteDatabase = {
  open(): boolean
  close(): boolean
  executeQueryWithArgumentsInArray(sql: string, args: any[]): SQLiteResultSe
}

export declare class SQLiteResultSet {
  stringForColumn(columnName: string): string
  next(): boolean
  close(): void
  resultDictionary(): DictObj
}
```

但是你确实可以通过这个读取 SQLite 数据库。比如 OhMyMN 就通过这个方法来读取来一个词典数据库，查看 源码。

StudyController

学习模式下整个界面的控制器。

UI 的嵌套结构如下：

- StudyController
  - NotebookController
    - OutlineView
    - MindMapView
      - MindMapNode
  - ReaderController
    - DocumentController

```ts
export declare class StudyController extends UIViewController {
  /**
   * View of the study controller
   * {@link UIView}
   */
  view: UIView
  /**
   * Study Mode
   */
  readonly studyMode: StudyMode
  /**
   * Narrow Mode
   */
  readonly narrowMode: boolean //when narrowmode, book split mode 1 is disab
  /**
   * DocMap Split Mode
   * {@link DocMapSplitMode}
   */
  docMapSplitMode: DocMapSplitMode
  /**
   * Right Map Mode
   */
  rightMapMode: boolean
  /**
   * Get notebook controller
   * @recommended use {@link MN.notebookController}
   */
  readonly notebookController: NotebookController
  /**
   * Get reader controller
   */
  readonly readerController: ReaderController
  /**
   * @param noteId NSString*
   */
  focusNoteInMindMapById(noteId: string): void
  /**
   * @param noteId NSString*
   */
  focusNoteInDocumentById(noteId: string): void
```

```ts
  refreshAddonCommands(): void
}
```

NotebookController

```ts
export declare class NotebookController {
  /**
   * View of notebook Controller
   */
  readonly view: UIView
  /**
   * Outline view
   */
  readonly outlineView: OutlineView
  /**
   * MindMap view
   */
  readonly mindmapView: MindMapView
  /**
   * Notebook id
   */
  readonly notebookId?: string
  /**
   * Focus note
   */
  readonly focusNote?: MbBookNote
  /**
   * Visible focus note
   */
  readonly visibleFocusNote?: MbBookNote
}
```

OutlineView

```ts
export declare class OutlineView {
  noteFromIndexPath(indexPath: NSIndexPath): MbBookNote
}
```

MindMapView

```ts
export declare class MindMapView extends UIView {
  /**
   * MindMap Nodes
   */
  readonly mindmapNodes?: MindMapNode[]
  /**
   * Nodes of selected
   */
  readonly selViewLst?: {
    note: MindMapNode
    view: UIView
  }[]
}
```

MindMapNode

```ts
export declare class MindMapView extends UIView {
  /**
   * MindMap Nodes
   */
  readonly mindmapNodes?: MindMapNode[]
  /**
   * Nodes of selected
   */
  readonly selViewLst?: {
    note: MindMapNode
    view: UIView
  }[]
}
```

ReaderController

```ts
export declare class ReaderController {
  /**
   * @recommended use {@link MN.currentDocumentController}
   */
  readonly currentDocumentController: DocumentController
  /**
   * Document controllers
   */
  readonly documentControllers?: DocumentController[]
  /**
   * view of ReaderController
   * {@link UIView}
   */
  view: UIView
}
```

DocumentController

```ts
export declare class DocumentController {
  readonly document?: MbBook
  /**
   * MD5 of the document.
   */
  readonly docMd5?: string
  /**
   * ID of Notebook
   */
  readonly notebookId?: string
  /**
   * Focus note of document, usually the note you are clicking on
   */
  readonly focusNote?: MbBookNote
  /**
   * Last focus note, only valid when you are selecting text
   */
  readonly lastFocusNote?: MbBookNote
  /**
   * Visible focus note
   */
  readonly visibleFocusNote?: MbBookNote
  /**
   * Text you are selecting
   */
  readonly selectionText?: string
  /**
   * Image from selection, usually converted to base64 to use.
   */
  imageFromSelection(): NSData
  /**
   * Image from focusNode
   */
  imageFromFocusNote(): NSData
  /**
   * start from 1. The virtual page has a large number of discontinuous page
   * */
  readonly currPageNo: number
  /**
   * start from 0, but if page deleted, the index will be 0.
   * */
  readonly currPageIndex: number
```

```
    /**
     * convert page index to page number
     */
    indexFromPageNo(pageNo: number): number
    /**
     * convert page number to page index
     */
    pageNoFromIndex(index: number): number
    /**
     * Jump to the page index
     */
    setPageAtIndex(index: number): void
    /**
     * Get all page indices from page number, which is not one-to-one mapping.
     */
    indicesFromPageNo(pageNo: number): number[]
}
```

MbBookNote

完整定义

MbBookNote 是 MarginNote 中的笔记对象。推荐使用 NodeNote 的相关方法来操作卡片。

静态方法

```ts
const Note: {
    createWithTitleNotebookDocument(title: string, notebook: MbTopic, doc: M
};
```

用来创建一个 MbBookNote 对象。注意，这并不是 MbBookNote 上的静态方法，MarginNote 将其暴露在 Note 对象上。

```ts
Note.createWithTitleNotebookDocument('title', current-topic, current-book)
```

目前还无法指定创建的笔记在脑图中的位置，也无法改变。

---

writeable 属性

```ts
    /**
     * Excerpt text of the note
     */
    excerptText?: string
    /**
     * Title of the note
     */
    noteTitle?: string
    /**
     * A int value, 0-15
     * Index of the color
     */
    colorIndex: number
    /**
     * A int value, 0-2
     * Index of the fill type
     */
    fillIndex: number
    /**
     *
     * not working
     */
    mindmapPosition: CGPoint
```

- `excerptText`：从 PDF 中摘录的内容。
- `noteTitle`：卡片标题。
- `colorIndex`：笔记颜色, 0-15，对着颜色面板数一数就知道了。
- `fillIndex`：笔记在 PDF 中选区的填充类型, 0-2。

> **TIP**
>
> 是不是发现没有评论，是的，评论无法修改，只能删除后重新添加。

---

readonly 属性

```ts
/**
 * Note id
 */
readonly noteId: string
/**
 * MD5 of the document
 */
readonly docMd5?: string
/**
 * Notebook id
 */
readonly notebookId?: string
/**
 * Page number of the start position of the note
 */
readonly startPage?: number
/**
 * Page number of the end position of the note
 */
readonly endPage?: number
/**
 * Start position of the note, like x,y
 */
readonly startPos?: string
/**
 * End position of the note, like x,y
 */
readonly endPos?: string
/**
 * Excerpt picture of the note, just the area of you selected
 */
readonly excerptPic?: excerptPic
/**
 * Date of the note created
 */
readonly createDate: Date
/**
 * Date of the note modified
 */
readonly modifiedDate?: Date
/**
 * List of media hash value seprated by '-'
```

```
     * @example
     * "mediaHash1-mediaHash2-mediaHash3"
     * note.mediaList?.split("-").map(hash => MN.db.getMediaByHash(hash))
     */
    readonly mediaList?: string
    /**
     * Origin note id, will be valid after merging
     */
    readonly originNoteId?: string
    /**
     * Whether the note branch in mindmap is closed
     */
    readonly mindmapBranchClose?: number
    /**
     * All the note text
     * @recommand {@link NodeNote.allText}
     */
    readonly notesText?: string
    /**
     * It Will be valid after merging itself into another note. It's the note id
     */
    readonly groupNoteId?: string
    /**
     * Comments of the note, different from the excerptText
     */
    readonly comments: NoteComment[]
    /**
     * Parent-notes of the note
     */
    readonly parentNote?: MbBookNote
    /**
     * List of Linked-note ID, used to locate the linked note card
     */
    readonly linkedNotes: {
      summary: boolean
      /**
       * nodeid of the linked note
       */
      noteid: string
      /**
       * text of the linked note
       */
      linktext: string
```

```
    }[]
    /**
     * Child-notes of the note
     */
    readonly childNotes?: MbBookNote[]
    /**
     * Array of summarized note-id
     */
    readonly summaryLinks: string[]
    /**
     * A int value
     */
    readonly zLevel?: number
    /**
     * Whether the card is hidden
     */
    readonly hidden?: boolean
    /**
     * A int value
     */
    readonly toc?: number
    readonly annotation?: boolean
    /**
     * Whether the image has been OCR to text
     */
    readonly textFirst: boolean
    /**
     * Mindmap group mode of the node branch
     */
    readonly groupMode?: GroupMode
    /**
     * A int value
     * Whether the note has a flashcard
     */
    readonly flashcard?: number
    /**
     * A int value
     */
    readonly summary: number
    /**
     * A int value
     */
    readonly flagged?: number
```

```
    readonly textHighlight?: {
      highlight_text: string
      coords_hash: string
      maskList?: string[]
      textSelLst?: any[]
    }
    readonly options?: DictObj
```

- `noteID`：笔记 ID
- `docMd5`：笔记所在的 PDF 的 MD5
- `notebookId`：笔记所在的脑图的 ID
- `groupNoteId`：当笔记被合并到另一个笔记时，这个属性会被赋值为目标笔记的 ID。

comments

评论列表，是一个数组，每个元素都是一个评论对象。

```ts
    comments: NoteComment[]
```

```ts
export type NoteComment = TextComment | HtmlComment | LinkComment | PaintCom

/**
 * Basic Comment, just text you typed
 * @see {@link NoteComment}
 */
export interface TextComment {
  type: "TextNote"
  /**
   * Get the content of the comment
   */
  text: string
  /**
   * NoteID of the note, is only valid after merging the notes
   */
  noteid?: string
}

/**
 * Generate when html copied to note
 * @see {@link NoteComment}
 */
export interface HtmlComment {
  type: "HtmlNote"
  /**
   * Size of the render image
   */
  htmlSize: DictObj
  /**
   * RTF
   */
  rtf: DictObj
  /**
   * HTML code
   */
  html: string
  /**
   * Text
   */
  text: string
  /**
   * NoteID of the note
```

```
     */
    noteid?: string
}

/**
 * Picture comment
 * @see {@link NoteComment}
 */
export interface PaintComment extends MNPic {
    type: "PaintNote"
}

/**
 * It not means link to another note and it will be generated when merge not
 * The notes merged into is the LinkComment
 * @see {@link NoteComment}
 */
export type LinkComment = LinkCommentText | LinkCommentPic

/**
 * @see {@link LinkComment}
 */
export interface LinkCommentText {
    type: "LinkNote"
    /**
     * NoteID of the note
     */
    noteid: string
    /**
     * Text of the comment
     */
    q_htext: TextComment["text"]
}

/**
 * @see {@link LinkComment}
 */
export interface LinkCommentPic {
    type: "LinkNote"
    /**
     * NoteID of the note
     */
    noteid: string
```

```
    /**
     * Text of the comment : {@link TextComment.text}
     */
    q_htext?: TextComment["text"]
    /**
     * Image of the comment : {@link MNPic}
     */
    q_hpic: MNPic
}
```

需要注意的是里面的 `LinkComment` 类型。它的 type 是 `LinkNote`，但是这个 `LinkNote` 并不是指它是一个链接，而是它被合并到了当前笔记中，作为评论存在。

excerptPic

摘录的选取区域的渲染图。

```ts
excerptPic?: ExcerptPic
```

```ts
/**
 * Base type of the picture in MarginNote
 */
interface MNPic {
  /**
   * A hash value. Use it to get the picture from {@link MN.db.getMediaByHas
   * @example
   * MN.db.getMediaByHash(pic.paint)?.base64Encoding()
   */
  paint: string
  /**
   * CGSize, use {@link CGSizeValue2CGSize} to convert it to {@link CGSize}
   */
  size: NSValue
}
/**
 * The area of the excerpt
 */
interface ExcerptPic extends MNPic {
  selLst: {
    [key: number]: {
      /**
       * Rotation of the picture
       */
      rotation: number
      /**
       * CGRect Value, the position of the picture in the note
       * use {@link CGRectValue2CGRect} to convert it to {@link CGRect}
       */
      rect: NSValue
      /**
       * CGRect Value, same as rect
       */
      imgRect: NSValue
      pageNo: number
    }
  }
}
```

如果想要获取到这个图片，可以使用

```ts
MN.db.getMediaByHash(note.excerptPic.paint)?.base64Encoding()
```

这样就得到了一个 base64 编码的图片了。

如果想知道图片的大小或者选区的大小，比如 OhMyMN 中 AutoStyle 可以根据选区的大小调整填充样式。可以使用

```ts
import { CGSizeValue2CGSize } from "marginnote"
const { width, height } = CGSizeValue2CGSize(note.excerptPic!.size)
```

mediaList

笔记中所有的媒体文件的 hash 值，用 `-` 分割。

```ts
note.mediaList?.split("-").map(hash => MN.db.getMediaByHash(hash))
```

这样就可是得到所有的媒体文件的 NSData 对象了。

linkedNotes

笔记中的链接。

```ts
/**
 * List of Linked-note ID, used to locate the linked note card
 */
readonly linkedNotes: {
  summary: boolean
  /**
   * nodeid of the linked note
   */
  noteid: string
  /**
   * text of the linked note
   */
  linktext: string
}[]
```

注意。要区别与前面 comments 中 LinkComment 的 type 值 `LinkNote`，这里的 `linkedNotes` 才是链接。

实例方法

```ts
paste(): void
/**
 * Clear format of the note
 */
clearFormat(): void
/**
 * @recommand {@link NodeNote.allText}
 */
allNoteText(): string
/**
 * Merge another note to this note
 */
merge(note: MbBookNote): void
/**
 * Append HTML comment to the note
 * @param html HTML text of the comment
 * @param text Pure text of the comment
 * @param size Size of the comment
 * @param tag Markdown editor plugin id. The HTML comment will be rendered b
 * @example
 * note.appendHtmlComment(
 *    "```math\n" + res + "\n```",
 *    res,
 *    { width: 420, height: 100 },
 *    "MarkDownEditor"
 *  )
 */
appendHtmlComment(html: string, text: string, size: CGSize, tag: string): vo
/**
 * Append one text comment to the note
 */
appendTextComment(text: string): void
/**
 * Append Note Link to the note
 *
 */
appendNoteLink(note: MbBookNote): void
/**
 * Remove comment by index
 */
removeCommentByIndex(index: number): void
/**
```

```
     * Number of handwritten strokes
     */
    getStrokesCount(): number
```

clearFormat

```ts
    clearFormat(): void
```

清除笔记的格式，包括卡片的尺寸。通常从浏览器中复制的内容会带有一些格式，使用这个方法可以清除格式。

merge

```ts
    merge(note: MbBookNote): void
```

合并其它笔记到当前笔记。

MbBook

也就是文档对象

```ts
export declare class MbBook {
  /**
   * Last notebook which the document is in and opened
   */
  readonly currentTopicId?: string
  /**
   * Date of last visit
   */
  readonly lastVisit?: Date
  /**
   * docMd5 of the document
   */
  readonly docMd5?: string
  /**
   * pathFile of the document
   */
  readonly pathFile?: string
  /**
   * Title of the document
   */
  readonly docTitle?: string
  /**
   * Page count of the document
   */
  readonly pageCount: number
  /**
   * Content of text layer of the document, not including OCR Pro layer
   * Each row and each character is an element of the array
   */
  textContentsForPageNo(pageNo: number): TextContent[][]
}
```

```ts
interface TextContent {
  /**
   * @example
   * String.fromCharCode(Number(char))
   */
  readonly char: string
  readonly rect: NSValue
}
```

`textContentsForPageNo` 用于获取指定页的文字。仅限 PDF 有文字层才可以获取到，OCR Pro 得到的文字层无法获取。

该方法已经封装好了，

```ts
import { getPageContent } from "marginnote"
```

如果想要了解更复杂的使用这个方法，可以查看 OhMyMN AutoComplete 的获取上下文的源码。

MbTopic

也就是笔记本对象

```ts
export declare class mbtopic {
  /**
   * notebook title, can be modified
   */
  title?: string
  /**
   * nodebook id
   */
  readonly topicid?: string
  readonly lastvisit?: date
  /**
   * main document md5
   */
  readonly maindocmd5?: string
  readonly historydate?: date
  readonly syncmode?: number | boolean
  readonly categorylist?: string
  readonly hashtags?: string
  readonly doclist?: string
  readonly options?: dictobj
  /**
   * doucments in the notebook
   */
  readonly documents?: mbbook[]
  /**
   * notes in the notebook
   */
  readonly notes?: mbbooknote[]
  readonly flags: notebooktype
  hidelinksinmindmapnode: boolean
}
```

Utility

SpeechManager

朗读

```ts
const SpeechManager: {
  sharedInstance(): SpeechManager
}
```

```ts
export declare type SpeechManager = {
  /**
   * start speech notes
   */
  startSpeechNotes(notes: any[]): void
  /**
   * stop speech
   */
  stopSpeech(): void
  /**
   * pause speech
   */
  pauseSpeech(): void
  /**
   * continue speech
   */
  continueSpeech(): void
  /**
   * previous speech
   */
  prevSpeech(): void
  /**
   * next speech
   */
  nextSpeech(): void
  canPrev(): boolean
  canNext(): boolean
  playText(text: string): void
  playTextLanguageTxt(text: string, languageTxt: string): void
  /**
   * If speaking
   */
  readonly speaking: boolean
  /**
   * If paused
   */
  readonly paused: boolean
  /**
   * when speak text
   */
  readonly sysSpeaking: boolean
  /**
```

```
     * Scene window
     */
    sceneWindow?: UIWindow
    languageCode?: string
}
```

如何使用: 来源于 read-aloud

```ts
class Speaker {
  constructor() {
    this.s = SpeechManager.sharedInstance()
    this.speakerStatus = "stop"
  }
  get status() {
    if (Date.now() - this.lastPlay < 300) return "playing"
    if (this.speakerStatus === "playing" && !this.s.sysSpeaking)
      return "over"
    return this.speakerStatus
  }
  play(content) {
    this.s.playText(content)
    this.lastPlay = Date.now()
    this.speakerStatus = "playing"
  }
  pause() {
    this.s.pauseSpeech()
    this.speakerStatus = "pause"
  }
  continue() {
    this.s.continueSpeech()
    this.lastPlay = Date.now()
    this.speakerStatus = "playing"
  }
  close() {
    this.s.stopSpeech()
    this.speakerStatus = "stop"
  }
}
```

UndoManager

撤销操作

```ts
const UndoManager: {
  sharedInstance(): UndoManager
}
```

```ts
export declare type UndoManager = {
  undoGrouping(actionName: string, notebookid: string, block: JSValue): voic
  /**
   * undo an action
   */
  undo(): void
  /**
   * redo an action
   */
  redo(): void
  /**
   * if can undo
   */
  canUndo(): boolean
  /**
   * if can redo
   */
  canRedo(): boolean
  /**
   * clear all actions
   */
  clearAll(): void
}
```

ZipArchive

压缩和解压缩

```ts
const ZipArchive: {
  initWithPath(path: string): ZipArchive
  unzipFileAtPathToDestination(path: string, destination: string): boolean
  unzipFileAtPathToDestinationOverwritePassword(
    path: string,
    destination: string,
    overwrite: boolean,
    password: string
  ): boolean
  createZipFileAtPathWithFilesAtPaths(path: string, filenames: any[]): boole
  createZipFileAtPathWithContentsOfDirectory(
    path: string,
    directoryPath: string
  ): boolean
}
```

```ts
export declare type ZipArchive = {
  open(): boolean
  writeFile(path: string): boolean
  writeDataFilename(data: NSData, filename: string): boolean
  close(): boolean
}
```

MenuController

选项菜单

```ts
class MenuController extends UIViewController {
  /**
   * new instance
   */
  static new(): MenuController
  menuTableView?: UITableView
  commandTable?: {
    title: string
    /** OC Object */
    object: any
    selector: string
    height?: number
    param: Record<string, any>
    checked: boolean
  }[]
  sections?: any[]
  rowHeight: number
  secHeight: number
  fontSize: number
  preferredContentSize: {
    width: number
    height: number
  }
}
```

Foundation

> **注意**
>
> MarginNote 插件只提供有限的支持，并且 API 版本比较老旧。如果有用不了的 API，是正常的。

Foundation 框架为应用程序和框架提供了一个基础功能层，包括数据存储和持久性、文本处理、日期和时间计算、排序和过滤以及网络。Foundation 定义的类、协议和数据类型在 macOS、iOS、 watch OS 和 tvOS SDK 中使用。

- Apple 官方文档
- TypeScript 版 API
- Objective-C 版 API
  - JSBNSArray.h
  - JSBNSAttributedString.h
  - JSBNSCharacterSet.h
  - JSBNSCountedSet.h
  - JSBNSData.h
  - JSBNSDate.h
  - JSBNSDateComponents.h
  - JSBNSDateFormatter.h
  - JSBNSDecimalNumber.h
  - JSBNSDictionary.h
  - JSBNSEnumerator.h
  - JSBNSError.h
  - JSBNSFileHandle.h
  - JSBNSFileManager.h
  - JSBNSFormatter.h
  - JSBNSHashTable.h
  - JSBNSHTTPURLResponse.h
  - JSBNSIndexPath.h
  - JSBNSIndexSet.h
  - JSBNSJSONSerialization.h
  - JSBNSKeyedArchiver.h

- JSBNSKeyedUnarchiver.h
- JSBNSLocale.h
- JSBNSMapTable.h
- JSBNSMutableArray.h
- JSBNSMutableAttributedString.h
- JSBNSMutableCharacterSet.h
- JSBNSMutableData.h
- JSBNSMutableDictionary.h
- JSBNSMutableIndexSet.h
- JSBNSMutableOrderedSet.h
- JSBNSMutableSet.h
- JSBNSMutableString.h
- JSBNSMutableURLRequest.h
- JSBNSNotification.h
- JSBNSNotificationCenter.h
- JSBNSNull.h
- JSBNSNumber.h
- JSBNSNumberFormatter.h
- JSBNSOperation.h
- JSBNSOperationQueue.h
- JSBNSOrderedSet.h
- JSBNSPointerArray.h
- JSBNSPredicate.h
- JSBNSProxy.h
- JSBNSScanner.h
- JSBNSSet.h
- JSBNSSimpleCString.h
- JSBNSTimer.h
- JSBNSTimeZone.h
- JSBNSURL.h
- JSBNSURLComponents.h
- JSBNSURLConnection.h

- JSBNSURLRequest.h
- JSBNSURLResponse.h
- JSBNSUserDefaults.h
- JSBNSUUID.h
- JSBNSValue.h

UIKit

> **注意**
>
> MarginNote 插件只提供有限的支持，并且 API 版本比较老旧。如果有用不了的 API，是正常的。

UIKit 为构建应用程序提供了各种功能，包括可用于构建 iOS、iPadOS 或 tvOS 应用程序核心基础设施的组件。该框架提供了用于实现 UI 的窗口和视图架构，用于向应用程序提供 Multi-Touch 和其他类型输入的事件处理基础设施，以及用于管理用户、系统和应用程序之间交互的主运行循环。

- Apple 官方文档
- TypeScript 版 API
- Objective-C 版 API
  - JSBNSMutableParagraphStyle.h
  - JSBNSParagraphStyle.h
  - JSBNSTextStorage.h
  - JSBUIActivityIndicatorView.h
  - JSBUIAlertView.h
  - JSBUIApplication.h
  - JSBUIBarButtonItem.h
  - JSBUIBarItem.h
  - JSBUIBezierPath.h
  - JSBUIButton.h
  - JSBUICollectionView.h
  - JSBUICollectionViewCell.h
  - JSBUICollectionViewController.h
  - JSBUICollectionViewLayout.h
  - JSBUICollectionViewLayoutAttributes.h
  - JSBUIColor.h
  - JSBUIControl.h
  - JSBUIDatePicker.h
  - JSBUIDevice.h
  - JSBUIEvent.h

- JSBUIFont.h
- JSBUIGestureRecognizer.h
- JSBUIImage.h
- JSBUIImagePickerController.h
- JSBUIImageView.h
- JSBUIKeyCommand.h
- JSBUILabel.h
- JSBUILocalNotification.h
- JSBUILongPressGestureRecognizer.h
- JSBUINavigationBar.h
- JSBUINavigationController.h
- JSBUINavigationItem.h
- JSBUIPageControl.h
- JSBUIPageViewController.h
- JSBUIPanGestureRecognizer.h
- JSBUIPasteboard.h
- JSBUIPickerView.h
- JSBUIPinchGestureRecognizer.h
- JSBUIPopoverController.h
- JSBUIResponder.h
- JSBUIRotationGestureRecognizer.h
- JSBUIScreen.h
- JSBUIScreenMode.h
- JSBUIScrollView.h
- JSBUISearchBar.h
- JSBUISegmentedControl.h
- JSBUISlider.h
- JSBUISwipeGestureRecognizer.h
- JSBUISwitch.h
- JSBUITabBar.h
- JSBUITabBarController.h
- JSBUITabBarItem.h

- JSBUITableView.h
- JSBUITableViewCell.h
- JSBUITableViewController.h
- JSBUITapGestureRecognizer.h
- JSBUITextField.h
- JSBUITextInputMode.h
- JSBUITextPosition.h
- JSBUITextRange.h
- JSBUITextView.h
- JSBUIToolbar.h
- JSBUITouch.h
- JSBUIView.h
- JSBUIViewController.h
- JSBUIWebView.h
- JSBUIWindow.h

模块

使用 `defineConfig` 定义模块，提供类型检查和代码提示。

```ts
import { defineConfig } from "~/profile"

export default defineConfig({
  name: "AutoReplace",
  key: "autoreplace",
  intro: lang.intro,
  link: doc("autoreplace"),
  settings: [
    {
      key: "on",
      type: CellViewType.Switch,
      label: lang.on,
      auto: {
        modifyExcerptText: {
          index: 999,
          method({ node, text }) {
            return replaceText(node, text)
          }
        }
      }
    },
    {
      key: "preset",
      type: CellViewType.MuiltSelect,
      option: lang.preset.$option1,
      label: lang.preset.label
    },
    {
      key: "customReplace",
      type: CellViewType.Input,
      help: lang.custom_replace.help,
      bind: ["preset", 0],
      link: lang.custom_replace.link
    }
  ],
  actions4card: [
    {
      type: CellViewType.ButtonWithInput,
      label: lang.replace_selected.label,
      key: "replaceCard",
      option: lang.replace_selected.$option2,
      method: ({ content, nodes, option }) => {
```

```
            undoGroupingWithRefresh(() => {
              // do something
            })
          }
        }
      ]
    })
```

完整的类型定义如下：

```ts
export type IConfig<T extends AllModuleKeyUnion | null = null> = {
  name: string
  key: T extends AllModuleKeyUnion ? T : string
  intro: string
  link?: string
  settings: ISetting<
    T extends AllModuleKeyUnion ? IAllProfile[T] : Record<string, any>
  >[]
  actions4card?: IAction<IActionMethod4Card>[]
  actions4text?: IAction<IActionMethod4Text>[]
}
```

---

name,key,intro,link

- `name` 模块名
- `key` 模块的唯一标识，不可重复，不需要与文件名相同。会根据 key 从 `profile/default.ts` 读取类型，提供类型检查，不同的选项类型有不同的属性。
- `intro` 模块介绍。
- `link` 模块文档的链接。

---

settings

模块的设置项。

type

设置菜单选项的类型有以下几种：

```ts
export const enum CellViewType {
  PlainText = 0,
  Switch = 1,
  Button = 2,
  ButtonWithInput = 3,
  Input = 4,
  InlineInput = 5,
  Select = 6,
  MuiltSelect = 7,
  Expland = 8
}
```

> **注意**
>
> Button 和 ButtonWithInput 两种类型属于按钮，只能在 actions 中使用。

对于不同的 type 有不同的属性

```ts
type HelpLink = XOR<{ help: string; link?: string }, {}>

type Bind<T> = {
  bind?: MaybeArray<
    MaybeArray<
      | [PickKeyByValue<T, number[]>, number | number[]]
      | [PickKeyByValue<T, boolean>, boolean]
      | ["quickSwitch", number | number[]]
    >
  >
}

type HelpLinkLabel = HelpLink & {
  label: string
}

export type ISettingInlineInput<T> = {
  key: PickKeyByValue<T, string>
  type: CellViewType.InlineInput
  check?: ICheckMethod
} & HelpLinkLabel &
  Bind<T>

export type ISettingInput<T> = {
  key: PickKeyByValue<T, string>
  type: CellViewType.Input
  help: string
  link?: string
  check?: ICheckMethod
} & Bind<T>

export type ISettingSwitch<T> = (
  | {
      key: Exclude<PickKeyByValue<T, boolean>, "on">
      type: CellViewType.Switch
    }
  | {
      key: "on"
      type: CellViewType.Switch
      auto: TypeUtilIndex<AutoUtilType>
    }
) &
```

```
  HelpLinkLabel &
  Bind<T>

export type ISettingExpland<T> = {
  key: PickKeyByValue<T, boolean>
  type: CellViewType.Expland
  label: [string, string]
} & Bind<T>

export type ISettingSelect<T> = {
  key: PickKeyByValue<T, number[]>
  type: CellViewType.Select | CellViewType.MuiltSelect
  option: string[]
} & HelpLinkLabel &
  Bind<T>

export type ISetting<T> =
  | ISettingInput<T>
  | ISettingSelect<T>
  | ISettingSwitch<T>
  | ISettingInlineInput<T>
  | ISettingExpland<T>
```

help,link,label

```ts
type HelpLink = XOR<{ help: string; link?: string }, {}>
```

- `help` 帮助信息
- `link` 帮助信息的链接，可以点击跳转
- `label` 选项文字描述

对于 `CellViewType.Input` 没有 `label` 属性，通常使用 help 作为 label。

填写了 `help` 属性才能使用 `link` 属性。

option

数组，选项的文字描述，只有 `CellViewType.Select` 和 `CellViewType.MuiltSelect` 有 `option` 属性。得到的值为数组下标，并且都返回数组。

check

`check` 属性是一个函数，用于检查用户输入的值是否合法，如果不合法，会在设置菜单中显示错误信息。`check` 函数的类型定义如下：

```ts
export type ICheckMethod = ({ input }: { input: string }) => MaybePromise<vo
```

只有 `CellViewType.Input`，`CellViewType.InlineInput`，以及 `CellViewType.ButtonWithInput` 有 `check` 方法。

只需要在函数里抛出一个 error 即可。

bind

```ts
type Bind<T> = {
  bind?: MaybeArray<
    MaybeArray<
      | [PickKeyByValue<T, number[]>, number | number[]]
      | [PickKeyByValue<T, boolean>, boolean]
      | ["quickSwitch", number | number[]]
    >
  >
}
```

当前选项的显示与否条件与其他选项绑定，当绑定选项的值为指定的值时，当前选项才会显示。

可以绑定 `Switch`，`Select`，`MuiltSwitch` 类型的选项。

举个例子

1. 当 `on` 为 `true` 时，当前选项才显示。

```ts
["on", true]
```

2. 当 `quickSwitch` 选择为 `0` 时，当前选项才显示。

```ts
["quickSwitch", 0]
```

3. 当 `quickSwitch` 选择为 `[0,1,2]` 中的一个时，当前选项才显示。

```ts
["quickSwitch", [0,1,2]]
```

4. 同时绑定多个选项，满足一个就会显示。

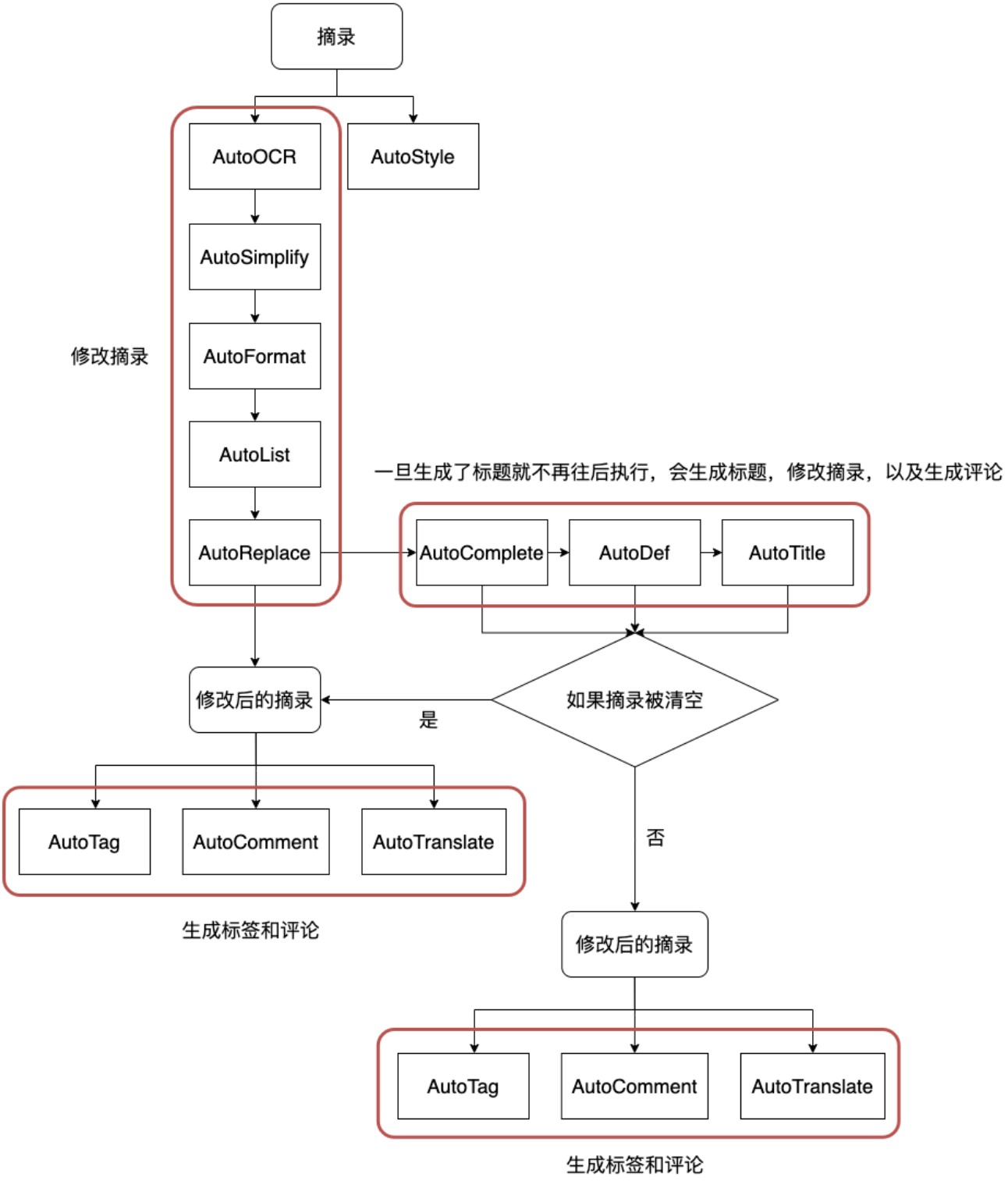
```ts
[
  ["quickSwitch", [0,1,2]],
  ["on", true]
]
```

5. 同时绑定多个选项，同时满足才会显示。

```ts
[
  [
    ["quickSwitch", [0,1,2]],
    ["on", true]
  ]
]
```

auto just ohmymn addon

OhMyMN 作为一个可以自动处理摘录的插件，提供了一系列注入点，可以在这些注入点中处理摘录，生成标题、标签、评论等。他们的执行顺序可以参考下图



当且仅当 key 为 `on` 时，才可以使用 auto 属性，auto 属性的类型定义如下：

```ts
export type AutoUtilType = {
  customOCR: ({ imgBase64 }: { imgBase64: string }) => string
  modifyExcerptText: ({
    node,
    note,
    text
  }: {
    node: NodeNote
    note: MbBookNote
    text: string
  }) => string
  generateTitles: ({
    node,
    text,
    note
  }: {
    node: NodeNote
    note: MbBookNote
    text: string
  }) => {
    title: string[]
    text: string
    comments?: string[]
  }
  generateTags: ({
    note,
    node,
    text
  }: {
    note: MbBookNote
    node: NodeNote
    text: string
  }) => string[]
  generateComments: ({
    node,
    note,
    text
  }: {
    note: MbBookNote
    node: NodeNote
    text: string
  }) => string[]
```

```
    modifyTitles: ({ titles }: { titles: string[] }) => string[]
    modifyStyle: ({ note }: { note: MbBookNote }) => {
      color: number | undefined
      style: number | undefined
    }
  }
```

可以通过 index 属性来控制某一个触发点里的所有函数的执行顺序，index 越大，越先执行。

```ts
  auto: {
    modifyExcerptText: {
      index: 999,
      method({ node, text }) {
        return replaceText(node, text)
      }
    }
  }
```

如果不需要指定执行顺序，可以简写，会安装模块注册的顺序执行。

```ts
  auto: {
    modifyExcerptText({ node, text }) {
        return replaceText(node, text)
      }
  }
```

`auto` 的函数里不需要直接修改笔记，只需要返回修改后的值即可。

---

actions

动作，也就是可以手动触发的按钮。会集中显示在 `MagicAction` 中。在 OhMyMN 中分为两种类型，一种是针对摘录的 `actions4card`，一种是针对文本的 `actions4text`。

just ohmymn addon

但是我提供的模版不区分类型，只有 `actions`，并且不会自动传入参数。需要自行获取选中的卡片或者选中的文字。

action 只能使用 `CellViewType.Button` 和 `CellViewType.ButtonWithInput` 两种类型。

```ts
export type IAction<T extends IActionMethod4Card | IActionMethod4Text> = {
  key: string
  label: string
  type: CellViewType.Button | CellViewType.ButtonWithInput
  /** auto generate. value is module's key*/
  module?: string
  /** auto generate. value is module's name*/
  moduleName?: string
  option?: string[]
  help?: string
  method: T
  check?: ICheckMethod
}
```

卡片动作

```ts
export type IActionMethod4Card = ({
  content,
  nodes,
  option
}: {
  content: string
  nodes: NodeNote[]
  option: number
}) => any
```

- `content` 输入框的值
- `nodes` 选中的卡片
- `option` 是选择的选项

文字动作

```ts
export type IActionMethod4Text = ({
  text,
  imgBase64,
  option
}: {
  text: string
  imgBase64: string
  option: number
}) => any
```

- `text` 选中的文字
- `imgBase64` 选区图片的 base64
- `option` 是选择的选项

输入处理

OhMyMN 中有大量的输入框，需要进行字符串处理，这些字符串处理的方法都在 `utils/input.ts` 中，可以直接在 `~/utils` 中导入。

---

reverseEscape

```ts
/**
 * Reverse escape a string
 * @param quote default false, if true, the str will be wrapped with double
 */
function reverseEscape(str: string, quote = false) : string
```

反转义字符串，比如将 `\n` 转换为换行，将 `\\` 转换为 `\`，将 `\"` 转换为 `"`，将 `\'` 转换为 `'`。

这个函数的原理是利用来 JSON.parse 来进行反转义，所以如果字符串不是 JSON 格式的话，会抛出异常。所以需要用 `""` 包裹字符串，提供 quote 参数，如果 quote 为 true，则会在字符串两边加上 `"`。如果字符串中包含 `"`，则还需要先转换为 `\"`。

---

escapeDoubleQuote

```ts
/**
 * Escape double quotes
 */
function escapeDoubleQuote(str: string): string
```

转义双引号，将 `"` 转换为 `\"`。结合 `reverseEscape` 使用。

---

isIntegerString, isNumberString

```ts
const isIntegerString = (text: string) => /^[0-9]+$/.test(text)
const isNumberString = (text: string) => /^[0-9]+\.?[0-9]*$/.test(text)
```

判断字符串是否是整数或者数字。

---

string2Reg

```ts
function string2Reg(str: string): RegExp
```

将字符串转为正则表达式对象。如果字符串不符合正则的格式，比如没有使用 // 包裹，则会当作普通字符串，会转义正则里的特殊字符。

```ts
string2Reg('a|b')

/a\|b/

string2Reg('/a|b/')

/a|b/
```

---

string2RegArray

```ts
function string2RegArray(str: string): RegExp[][]
```

将字符串转为正则表达式对象。并且是二维数组。 比如

```ts
string2RegArray('/a|b/')

[[/a|b/]]

string2RegArray('[/a|b/g, /$hello^/]; [/a|b/]')

[[/a|b/g, /$hello^/], [/a|b/]]
```

具体支持的格式可以查看 自定义格式

---

string2ReplaceParam

```ts
function string2ReplaceParam(str: string): ReplaceParam[]

interface ReplaceParam {
  regexp: RegExp
  newSubStr: string
  fnKey: number
}
```

将 replace 函数的自定义格式，转换为 ReplaceParam 数组。

举个例子

```ts
string2ReplaceParam('(/a|b/g, "new", 1)')

[{
  regexp: /a|b/g,
  newSubStr: "new",
  fnKey: 1
}]


string2ReplaceParam('(/a|b/g, "new", 1); (/a|b/g, "new", 1)')

[{
  regexp: /a|b/g,
  newSubStr: "new",
  fnKey: 1
},{
  regexp: /a|b/g,
  newSubStr: "new",
  fnKey: 1
}]
```

extractArray

```ts
function extractArray(text: string, params: ReplaceParam[]): string[]
```

实现 replace 函数格式 的提取作用。

regFlag

```ts
const regFlag: {
    add(reg: RegExp, flag: "g" | "i" | "m" | "s" | "y" | "u"): RegExp;
    remove(reg: RegExp, flag: "g" | "i" | "m" | "s" | "y" | "u"): RegExp;
}
```

添加或者删除正则表达式的 flag。

文字相关

可以从 `~/utils` 中导入。

CJKRegex, isCJK, notCJK

```ts
const CJK =
  "\u2e80-\u2eff\u2f00-\u2fdf\u3040-\u309f\u30a0-\u30fa\u30fc-\u30ff\u3100-\
const CJKRegex = new RegExp(`([${CJK}])`, "g")

const isCJK = (text: string) => CJKRegex.test(text)
const notCJK = (text: string) => !isCJK(text)
```

CJK 也就是中日韩统一表意文字，这个正则表达式会匹配所有的 CJK 字符。

---

isHalfWidth

```ts
const isHalfWidth = (text: string) => /^[\u0000-\u00ff]*$/.test(text)
```

判断字符串是否全都是半角字符，通常用于判断是否是英文。

---

isLanguage

```ts
const isLanguage = {
  // Russian
  Cyrillic: (text: string) => /\p{sc=Cyrillic}/u.test(text),
  // English, French, German...
  Latin: (text: string) => /\p{sc=Latin}/u.test(text),
  // Chinese, Janpanese, Korean。由于日语和韩语均有可能出现汉字，导致不能准确判断。
  CJK: (text: string) => CJKRegex.test(text),
  Han: (text: string) => /\p{sc=Han}/u.test(text),
  // not full
  Janpanese: (text: string) =>
    /[\u3000-\u303f\u3040-\u309f\u30a0-\u30ff]/u.test(text),
  Korea: (text: string) => /\p{sc=Hangul}/u.test(text)
}
```

使用 Unicode 的属性来判断字符串是否是某种语言。

---

countWord

```ts
function countWord(text: string): number
```

统计字符串的单词数，中文会被当作一个单词。

---

byteLength

```ts
function byteLength(text: string): number
```

统计字符串的字节数，中文会被当作两个字节。

基础

提醒

需要 Mac，需要购买 Mac 版 MarginNote

准备工作

1. 安装 pnpm

```shell
curl -fsSL https://get.pnpm.io/install.sh | sh -
```

2. 安装 nodejs

```shell
pnpm env use --global 18
```

3. 安装 vscode

首先要明白 MN 的插件底层 API 是基于 Objective-C 的，虽然看上去插件是用 JS 写，但其实只是通过 JSBridge 调用 Objective-C 的 API 来实现。

所以你要想开发插件，首先要了解 Objective-C。MN 的插件其实暴露了很多 API，只是比较底层。而且还有一个问题，就是 MN 使用的 JSBridge 框架比较老旧，很多新 API 不支持，Bug 也比较多，iOS 和 macOS 也会有不少兼容性的问题，所以要做好踩坑的准备。

也正是因为插件只是用了 JSBridge，你没法使用 NodeJS 的 API，也没法使用部分 Browser 的 API，比如 `fetch`。而 Browser 的 API 也是 Safari 实现，所以 Safari 不支持的 JS 特性插件也不支持。

目前来说，MN 插件开发门槛比较高，坑比较多，不过 OhMyMN 的出现让开发门槛降低了很多。比如 OhMyMN 使用 TypeScript 开发，在编译期就可以发现大部分的问题。OhMyMN 使用 ESBuild 打包，方便拆分模块，一个命令就可以快速打包一个插件，并且在开发期间还可以热更新。更为重要的是 OhMyMN 封装了大部分的 API，提供更加强大的功能，还不需要接触到底层的 Objective-C API。

插件开发的教程分为三个部分：

1. MN 插件（Lite）开发

2. OhMyMN 模块开发
3. MN 插件（OhMyMN）开发

MN 插件（Lite）开发 只适用于简单功能，所以我称之为 Lite 版，也就是在 OhMyMN 出现之前传统的用一个 `main.js` 就能搞定的功能，比如 `AutoTitle`，`SearchInEudic`。如果你的功能比较复杂，又和摘录，卡片有关，建议阅读 OhMyMN 模块开发。如果无关，建议阅读 MN 插件（OhMyMN）开发。

OhMyMN 模块就是 OhMyMN 内置的插件，独立于 MN 插件。OhMyMN 模块开发 非常简单，可以快速开发复杂的功能。

OhMyMN 不只是一个插件，而是一个开发框架，提供了一套完整的插件开发体系，只是由于 MN 插件底层种种问题导致很难像前端框架一样拆分功能，按需选择。目前 OhMyMN 作为框架来说还不够完善，但是已经可以用于开发复杂的功能了，我们提供了很多 模版 供你选择。

插件结构

MN 插件文件以 `.mnaddon` 为后缀，本质上是一个 `.zip` 压缩包，里面包含了插件的所有文件。mnaddon 工具提供了打包和解包的命令。

`mnaddon.json`

插件的描述清单，包含了插件的基本信息，比如名称、版本、作者等。

```json
{
  "addonid": "marginnote.extension.ohmymn",
  "author": "MarginNote",
  "title": "OhMyMN",
  "version": "4.2.0",
  "marginnote_version_min": "3.7.21",
  "cert_key": ""
}
```

- `addonid` 是插件的唯一标识，不能重复，统一使用 `marginnote.extension.` 开头，后面跟上你的插件 id/key。
- `marginnote_version_min` 是插件最低支持的版本，如果 MN 低于该版本，将无法安装。
- `cert_key` 是插件的签名，需要向 MarginNote 官方申请，申请方法。没有签名的插件默认无法安装，需要打开 `允许加载未经认证的插件` 选项，但是仍然会有弹窗警告。每次插件更新均需要重新申请签名。

`main.js`

插件代码

`logo.png`

插件图标，尺寸必须是 44x44，否则会被缩放。文件名不限，需要在 `queryAddonCommandStatus` 中手动指定。

插件对象

如何创建一个插件？下面是通过 mnadon 工具创建的插件模版的一部分。

```js
JSB.newAddon = () => {
  const showHUD = (text, duration = 2) => {
    self.app.showHUD(text, self.window, duration)
  }
  const go = async () => {
    const { option } = await popup(
      "Template Popup",
      "Whether to view the Chinese development documents (old version, new v
    )
    if (option === -1) return
    self.app.openURL(
      NSURL.URLWithString(encodeURI("http://docs.test.marginnote.cn/"))
    )
  }
  // 返回一个 JSExtension 类，也就是插件
  return JSB.defineClass(
    Addon.name + ": JSExtension",
    // 实例方法
    {
      // 新的 MN 窗口打开
      sceneWillConnect() {
        self.status = false
        self.app = Application.sharedInstance()
        self.studyController = self.app.studyController(self.window)
      },

      // 设置插件按钮图标以及选中状态。点击插件按钮会触发 "onToggle" 方法。
      // 只在脑图模式下才显示图标。
      queryAddonCommandStatus() {
        return self.studyController.studyMode !== 3
          ? {
              image: "logo_44x44.png",
              object: self,
              selector: "onToggle:",
              checked: self.status
            }
          : null
      },

      // 点击插件图标执行的方法。效果就是按钮被选中，然后弹窗，然后取消选中。
      async onToggle() {
        self.status = true
```

```
            // 刷新插件按钮状态
            self.studyController.refreshAddonCommands()
            await go()
            self.status = false
            self.studyController.refreshAddonCommands()
          }
        },
        // 类方法
        {}
      )
```

从上面的示例可以看出，我们只需要给 `JSB.newAddon` 赋值一个函数，这个函数返回一个 `JSB.defineClass` 定义的 Objective-C 类，在 JS 中通常称之为对象，这个对象就是我们的插件。不需要我们手动 new 实例，会自动创建。

而这样的一个 `JSB.defineClass` 方法，需要传入 3 个参数：

1. 类的名称以及继承的父类，上面的 `Template: JSExtension` 就表示这是一个继承自 `JSExtension` 的类，这个类的名称是 `Template`。
2. 实例方法，主要包括：
   - 插件的生命周期
   - 插件接收到的事件处理方法。
   - 插件按钮刷新，按钮点击等等一系列的处理方法都需要作为实例方法。

```js
// OhMyMN 中的实例方法
 JSB.defineClass(
   getObjCClassDeclar(Addon.title, "JSExtension"),
   {
     ...lifecycle.instanceMethods,
     ...switchPanel,
     ...handleGestureEvent,
     ...handleReceivedEvent
   },
   lifecycle.classMethods
 )
```

3. 类方法。只需要了解插件安装和卸载的事件是静态方法，生命周期

self

self 指代的就是这个插件实例，注意，这是 Objective-C 里的对象。并且这个 self 只能在对象的实例方法中使用。

MN 支持多窗口，不同窗口的插件实例是不同的，所以 self 也是不同的。

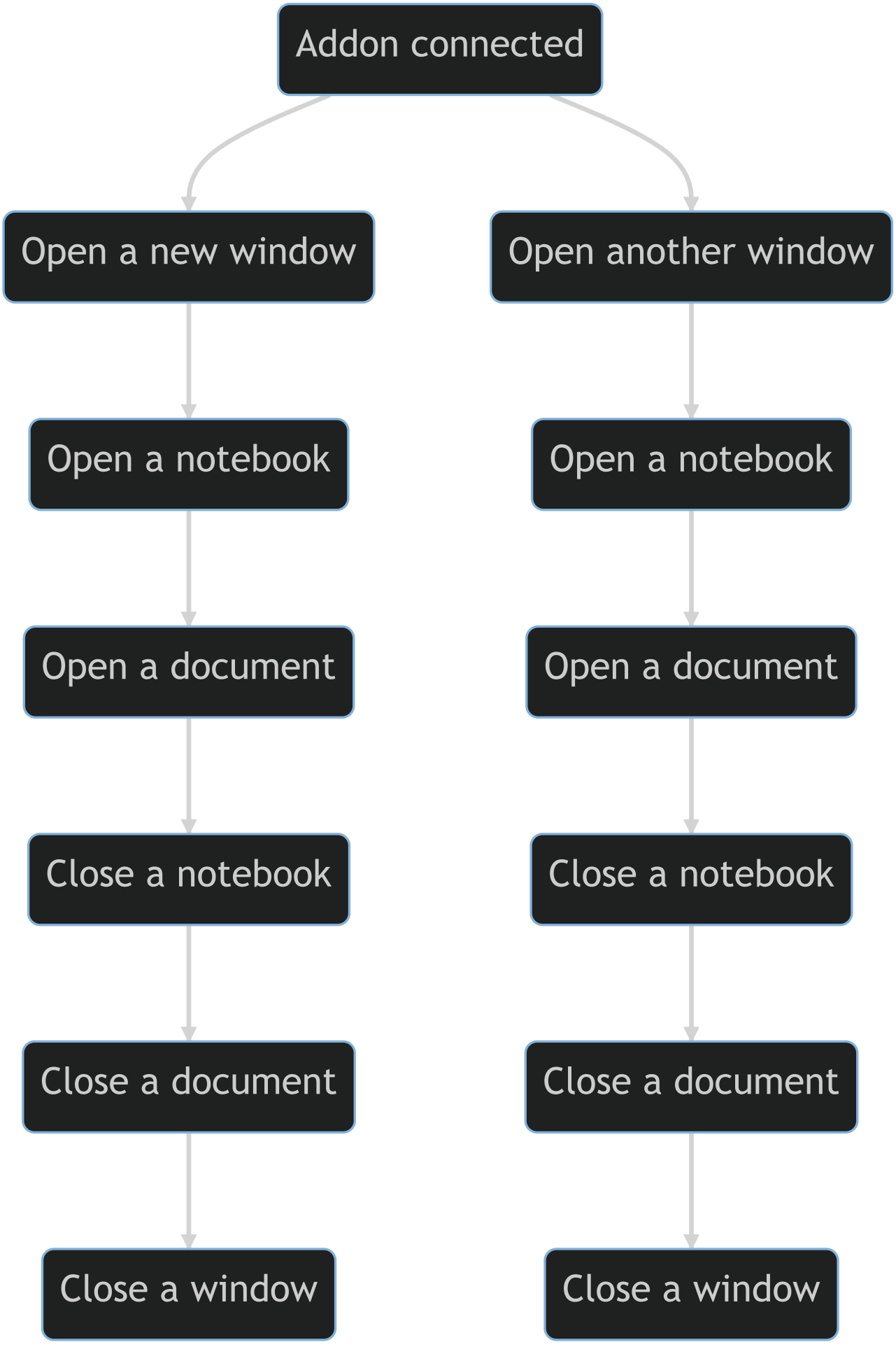
> **注意**
>
> 插件里的 JS 变量是多窗口共用的。要想区分开只能挂载到 self 上。

生命周期

所谓生命周期，也就是 MN 和插件在运行的各个阶段调用的特定方法。

实例方法

MN 的执行顺序


Addon connected
Open a new window
Open another window
Open a notebook
Open a notebook
Open a document
Open a document
Close a notebook
Close a notebook
Close a document
Close a document
Close a window
Close a window

注意

iPad 上直接划掉后台，不会触发关闭事件，包括关闭笔记本和关闭文档。请在进入后台的时候触发相关动作。

sceneWillConnect()

创建新的 MN 窗口时触发，MN 支持多窗口，每次新建窗口都会触发这个方法。

sceneDidDisconnect()

关闭 MN 窗口时触发。

sceneWillResignActive()

MN 窗口失去焦点时触发。

sceneDidBecomeActive()

MN 窗口获得焦点时触发。

notebookWillOpen(topicid: string)

打开笔记本时触发。可以获取笔记本的 id。

notebookWillClose(topicid: string)

关闭笔记本时触发。可以获取笔记本的 id。

documentDidOpen(docmd5: string)

打开文档时触发。可以获取文档的 md5。

documentWillClose(docmd5: string)

关闭文档时触发。可以获取文档的 md5。

静态方法

addonDidConnect()

插件安装，启用，注意启动 MN 时也会触发。

addonWillDisconnect()

插件卸载，停用时触发。

事件监听

在特定事件发生时会自动调用事件处理的方法。大多数事件都是摘录有关。

添加/移除事件监听

通常在打开笔记本的时候添加事件监听。 比如当 `ProcessNewExcerpt` 调用 `onProcessNewExcerpt` 方法，也就是摘录时触发。

```js
NSNotificationCenter.defaultCenter().addObserverSelectorName(
  self,
  "onProcessNewExcerpt:",
  "ProcessNewExcerpt"
)
```

注意这里有一个 `:` 。然后我们创建一个 `onProcessNewExcerpt` 函数，这个函数要作为插件对象的实例方法。

在关闭笔记本的时候移除事件监听。

```js
NSNotificationCenter.defaultCenter().removeObserverName(
  self,
  "ProcessNewExcerpt"
)
```

可以导入 事件监听控制器，同时监听多个事件。

```ts
import { eventObserverController } from "marginnote"
```

所有事件

事件处理函数可以接收到一个 `sender` 对象，可以通过 `sender.userInfo` 得到事件传递来的参数。

AddonBroadcast

打开 `marginnote3app://addon/ohmymn?type=card&shortcut=1` 这种格式的 URL 时触发。

`userInfo.message` 就会得到 `ohmymn?type=card&shortcut=1`

ProcessNewExcerpt

从 PDF 中摘录时触发。

`userInfo.noteid` 可以得到当前摘录的笔记 id。

ChangeExcerptRange

修改 PDF 中摘录选取的范围时触发。

`userInfo.noteid` 可以得到当前笔记的 id。

PopupMenuOnNote

点击笔记，弹出菜单时触发。点击 PDF 中的摘录选区以及脑图中的卡片都属于点击笔记，所有都会触发。

`userInfo.note` 可以得到点击的笔记对象。注意，这里直接是笔记对象。

ClosePopupMenuOnNote

笔记菜单消失时触发。

`userInfo.note` 可以得到点击的笔记对象。

PopupMenuOnSelection

在 PDF 中框选文字或区域后，弹出菜单时触发。

```js
userInfo = {
  arrow,
  documentController,
  winRect
}
```

如果想要得到选中的文字，可以用

`userInfo.documentController.selectionText`

得到选中的文字。

ClosePopupMenuOnSelection

选中文字的菜单消失时触发，同上。

OCRImageBegin

OCR 开始时触发，包括摘录时自动 OCR 以及手动 OCR。

OCRImageEnd

OCR 结束时触发

数据存储

```ts
function getLocalDataByKey(key: string) {
  return NSUserDefaults.standardUserDefaults().objectForKey(key)
}
function setLocalDataByKey(data: any, key: string) {
  NSUserDefaults.standardUserDefaults().setObjectForKey(data, key)
}
```

持久化数据，也就是存到本地。但这并不是文件。可以直接存储 JSON 对象，不需要转换成字符串。但注意不要有 undefined，否则会报错。

可以直接导入使用

```ts
import { getLocalDataByKey, setLocalDataByKey } from "marginnote"
```

MN 插件（Lite）

所谓 Lite 版插件，指一两百行代码的简单插件，并且使用 JavaScript 进行开发。这是为了区别于 OhMyMN 那种使用 TypeScript 开发，需要编译，有控制面板的插件。在 OhMyMN 发布之前的 MN 插件都属于 Lite 版插件。当然也并不是说 Lite 版插件无法开发复杂功能，只是不推荐。

一个 Lite 版插件通常只有三个文件：

1. logo.png
2. main.js
3. mnaddon.json

CLI

为了更方便的开发 Lite 版插件，我们开发了一个命令行工具 mnaddon。

该工具主要有以下命令

1. create: 使用模板创建新的插件项目。
2. resize: 调整 logo 大小为 44x44，这是 MN 插件要求的大小。
3. watch: 监听文件修改，并且将修改后的文件同步到 MN 插件目录。如果插件未安装，会自动安装。
4. restart: 每次修改代码后，需要重启 MN 才会生效。该命令还可以自动跳过未签名的警告。
5. dev: 打开 MN 和控制台，开始监听文件修改。
6. build: 打包成插件。
7. unpack: 解包插件。

不管是监听文件变化还是最终打包成插件，都只会处理 png，js，json 这三类文件。另外，请不要使用子文件夹，否则不会被监听或者打包。

安装

```shell
pnpm add mnaddon -g
```

使用

上面已经介绍了每个命令的作用，你还可以使用 `mnaddon help` 或者 `mnaddon help restart` 来查看每个命令具体的使用方法。 `<project-name>` 表示必填， `[output-name]` 表示可选。

```shell
# 创建新的插件项目，名为 template
mnaddon create template
# 进入项目目录
cd template
# 打包成插件
mnaddon build
```

示例

```js
;(function () {
  const zh = {
    confirm: "确定",
    cancel: "取消"
  }
  const en = {
    confirm: "OK",
    cancel: "Cancel"
  }
  const Addon = {
    name: "Template",
    key: "template"
  }
  // 同时适配中英文
  const lang = isZH() ? zh : en
  function isZH() {
    return (
      NSLocale.preferredLanguages().length &&
      NSLocale.preferredLanguages()[0].startsWith("zh")
    )
  }
  // 可以在 控制台.app 中查看 log 输出。可以通过 key 来筛选。
  const console = {
    log(obj) {
      JSB.log(`${Addon.key} %@`, obj)
    }
  }

  // 弹窗
  const popup = (title, message, buttons = [lang.confirm]) => {
    return new Promise(resolve =>
      UIAlertView.showWithTitleMessageStyleCancelButtonTitleOtherButtonTitle
        title,
        message,
        0,
        lang.cancel,
        buttons,
        (alert, buttonIndex) => {
          resolve({
            option: buttonIndex - 1
          })
        }
```

```
    )
  )
}

JSB.newAddon = () => {
  const showHUD = (text, duration = 2) => {
    self.app.showHUD(text, self.window, duration)
  }
  const go = async () => {
    const { option } = await popup(
      "Template Popup",
      "Whether to view the Chinese development documents (old version, new
    )
    if (option === -1) return
    self.app.openURL(
      NSURL.URLWithString(encodeURI("http://docs.test.marginnote.cn/"))
    )
  }
  // 返回一个 JSExtension 类，也就是插件
  return JSB.defineClass(
    Addon.name + ": JSExtension",
    {
      // 新的 MN 窗口打开
      sceneWillConnect() {
        self.status = false
        self.app = Application.sharedInstance()
        self.studyController = self.app.studyController(self.window)
      },

      // 设置插件按钮图标以及选中状态。点击插件按钮会触发 "onToggle" 方法。
      // 只在脑图模式下才显示图标。
      queryAddonCommandStatus() {
        return self.studyController.studyMode !== 3
          ? {
              image: "logo_44x44.png",
              object: self,
              selector: "onToggle:",
              checked: self.status
            }
          : null
      },

      // 点击插件图标执行的方法。效果就是按钮被选中，然后弹窗，然后取消选中。
```

```
            async onToggle() {
              self.status = true
              // 刷新插件按钮状态
              self.studyController.refreshAddonCommands()
              await go()
              self.status = false
              self.studyController.refreshAddonCommands()
            }
          },
          {}
        )
      }
    })()
```

这个插件的作用就是，点击插件按钮弹出一个对话框，点击确定后打开开发文档。

MN 插件（OhMyMN）

OhMyMN 是 MarginNote 插件开发框架，提供了一套完整的插件开发体系，是开发复杂插件的首选，OhMyMN 自身也是一个强大的插件，是你开发复杂插件最好的模板。

OhMyMN 作为插件时，专注于摘录自动处理和卡片处理。如果你想开发这部分的功能，建议直接开发 OhMyMN 模块。避免重复实现，以及避免冲突。

在这个部分，将暂时不介绍摘录处理和卡片处理相关的内容。如果需要了解，请查看 OhMyMN 模块。

OhMyMN 从插件起步，逐步发展出一套完整的插件开发体系，提供强大的配置管理能力，模块化能力，控制面板，以及手势，快捷键。但是由于 MN 插件底层的混乱，导致 OhMyMN 变得复杂，各部分耦合得比较严重，目前只抽取出了 MN 插件通用的 API，也就是 `marginnote` 包。但是其它的部分，仍然耦合在一起，无法变得可选。目前我采取的办法是提供模板。

如果你用过 OhMyMN 插件，完整看过 使用指南，那么你应该对 OhMyMN 有一个大致的了解。

OhMyMN 的大致逻辑：

- 将插件拆分成多个模块，每个模块负责一个功能，模块之间可以相互调用，也可以相互独立。可以选择启用。
- 所有模块手动执行的动作都统一放在一起，方便点击，形成了 MagicAction 模块。并且提供多种触发动作的方式，比如手势，URL Scheme。
- 复杂的配置，有 5 套全局配置，单独的笔记本配置，单独的文档配置。强大的配置管理功能，可以导出和导入配置，也可以初始化配置。

不过除了摘录场景外，很少需要这么复杂的功能。不需要有这么多的模块，也不需要这么复杂的配置。现在我提供了 3 个模板，可以根据自己的需求选择。

1. 基础模板

没有控制面板，没有配置，没有模块，没有动作，没有手势，没有快捷键。

```ts
pnpm create ohmymn -t base
```

2. 基础模板 + 控制面板

有控制面板，但是配置更加简单，不包括配置导出导入以及初始化，有动作，支持 URL Scheme 调用。

```ts
pnpm create ohmymn -t panel
```

3. 基础模板 + 控制面板 + 配置管理

有控制面板，有动作，支持 URL Scheme 调用。

```ts
pnpm create ohmymn -t profile
```

描述清单

也就是 `manifest.ts` 文件，用于设置插件的基本信息。最终会生成 mnaddon.json 文件。

```ts
interface Manifest {
  /**
   * Unique identifier of the addon. Not need "marginnote.extension.".
   */
  key: string
  author: string
  title: string
  version: string
  /**
   * Github repository url.
   */
  github?: string
  /**
   * The minimum version of MarginNote that the addon supports.
   */
  minMarginNoteVersion: string
  /**
   * Profile key, used to save addon settings.
   */
  profileKey?: {
    global: string
    doc: string
    notebook: string
  }
  certKey?: string
  /**
   * Panel color
   */
  color?: {
    border: string
    button: string
  }
  /** Chinese forum url */
  forumZH?: string
  forum?: string
  docZH?: string
  doc?: string
  /** Files to be copied to the addon folder */
  files?: string[]
}
```

key

key 不同于 addonid，不需要 `marginnote.extension.`，

```ts
addonid = "marginnote.extension." + key
```

key 不能重复，否则会覆盖同名插件。key 还会作为 MN.log 的筛选关键词。

minMarginNoteVersion

插件支持的 MarginNote 最低版本，如果低于这个版本，插件将无法安装。

profileKey

配置文件的 key，通过这个 key 来写入和读取配置。

- `global`：全局配置。
- `doc`：文档配置，当前文档有效。
- `notebook`：笔记本配置，当前笔记本有效。

certKey

插件签名。

> **注意**
>
> 签名和 `main.js` 文件是一一绑定的，而 OhMyMN 的打包是随机的，代码会发生变化。而签名只能在打包后申请，所以只能将申请签名的插件解压，然后填入签名，再打包。

color

面板颜色，目前只能设置边框和按钮颜色。只能填写 `#ffffff` 格式的颜色。

---

files

需要复制到插件目录的文件，可以是文件夹或文件。文件夹会递归复制。文件夹和文件的路径都是相对于插件根目录的。 比如 `["assets/logo.png", "assets/icon/"]`。

打包插件

OhMyMN 使用 esbuild 打包插件，将所有代码打包成一个文件。它很快，但不会检查 TypeScript 类型。在 OhMyMN 中，类型的正确非常重要。所以我使用了 ESBuild 的上层封装，也就是 estrella，它可以快速检查 TypeScript 类型。

在 `build.ts` 里修改打包的配置。

开发

```ts
pnpm dev
```

在开发阶段，可以监听文件变化，自动重新编译，生成 `main.js` 文件，然后将其复制到 MarginNote 的插件目录。

每次修改都会检查 TypeScript 类型，如果有错误，会提示错误信息。

> **注意**
>
> MarginNote 无法自动读取新的插件代码，需要手动重启 MarginNote。可以使用 mnaddon 工具快速重启 MarginNote，以及跳过弹窗。

构建

```ts
pnpm build
```

打包插件，在 dist 目录下生成 `.mnaddon` 文件，可以直接安装到 MarginNote 中。

为了更方便的在 iPad 上调试插件，可以使用

```ts
pnpm run build:iPad
```

会在打包后自动通过 AirDrop 将插件发送到 iPad 上。

TIP

需要安装 airdrop-cli，才能在命令行里使用 AirDrop。

```sh
brew install vldmrkl/formulae/airdrop-cli
```

项目结构

所有的代码都在 `src` 目录下，`src` 目录下的文件结构如下：

```
├── jsExtension
│   ├── handleExcerpt
│   ├── handleMagicAction
│   ├── fetchNodeProperties.ts
│   ├── handleGestureEvent.ts
│   ├── handleReceivedEvent.ts
│   ├── index.ts
│   ├── lang.ts
│   ├── lifecycle.ts
│   ├── mustacheFunc.ts
│   └── switchPanel.ts
├── settingViewController
│   ├── handleUserAction.ts
│   ├── index.ts
│   ├── lang.ts
│   ├── lifecycle.ts
│   └── settingView.ts
├── modules
│   ├── addon
│   ├── ai
│   ├── aiassistant
│   ├── aitranslte
│   ├── anotherautodef
│   ├── anotherautotitle
│   ├── autocomment
│   ├── autocomplete
│   ├── autoformat
│   ├── autolist
│   ├── autoocr
│   ├── autoreplace
│   ├── autosimplify
│   ├── autostyle
│   ├── autotag
│   ├── autotranslate
│   ├── copysearch
│   ├── gesture
│   ├── magicaction4card
│   ├── magicaction4text
│   ├── shortcut
│   └── index.ts
├── profile
│   ├── default.ts
```

```
│   ├── index.ts
│   ├── lang.ts
│   ├── profileAction.ts
│   ├── profileAuto.ts
│   ├── rewrite.ts
│   ├── typings.ts
│   ├── updateDataSource.ts
│   └── utils.ts
├── typings
│   ├── AutoUtils.ts
│   ├── DataSource.ts
│   ├── index.ts
│   ├── Module.ts
│   └── utils.d.ts
├── utils
│   ├── third party
│   ├── checkInput.ts
│   ├── index.ts
│   ├── input.ts
│   ├── lang.ts
│   ├── number.ts
│   └── text.ts
├── dataSource
│   ├── index.ts
│   ├── lang.ts
│   └── more.ts
├── addon.ts
├── coreModule.ts
├── main.ts
└── self.d.ts
```

可以发现 OhMyMN 是一个相当复杂的项目。当然里面大部分的代码都与具体的功能实现有关，也就是摘录处理和卡片处理，我会特别标注出来 just ohmymn addon，如果只想用 OhMyMN 模版开发新的插件，你可以跳过。

---

jsExtension

```
├── handleExcerpt
│   ├── genNewExcerpt.ts
│   └── index.ts
├── handleMagicAction
│   ├── handleCardAction.ts
│   ├── handleTextAction.ts
│   └── index.ts
├── fetchNodeProperties.ts
├── mustacheFunc.ts
├── handleGestureEvent.ts
├── handleReceivedEvent.ts
├── index.ts
├── lang.ts
├── lifecycle.ts
└── switchPanel.ts
```

这个文件夹就是整个插件的核心。

- `handleExcerpt` 自动处理摘录，将所有模块的摘录处理功能集中在这里，按流程自动执行 just ohmymn addon。
- `handleMagicAction` 执行动作，将所有模块的动作集中在这里执行。包括文字动作和卡片动作。
- `fetchNodeProperties.ts` `mustacheFunc` 用于模版引擎读取卡片属性。 just ohmymn addon
- `handleGestureEvent.ts` 手势相关。 just ohmymn addon
- `handleReceivedEvent.ts` 接收事件，处理事件。
- `switchPanel.ts` 切换面板显示。
- `lifecycle.ts` 插件生命周期。

---

settingViewController

settingViewController 也就是控制面板，

```
├── handleUserAction.ts
├── index.ts
├── lang.ts
├── lifecycle.ts
└── settingView.ts
```

- `lifecycle.ts` 控制面板的生命周期。
- `settingView.ts` 控制面板的 UI 渲染。
- `handleUserAction.ts` 处理用户的操作，比如点击按钮，输入框等，然后发送通知，这样就可以在 `jsExtension/handleReceivedEvent.ts` 中接收到通知，然后执行相应的操作。

需要注意的时，控制面板和插件是两个不同的 OC 对象，他们的通信是通过发送通知来实现的。而且他们的 self 是不一样的。需要在 `jsExtension/lifecycle.ts` 手动给 settingViewController 实例传递数据。

```ts
// 实例化
self.settingViewController = SettingViewController.new()
self.settingViewController.addon = self.addon
self.settingViewController.dataSource = self.dataSource
self.settingViewController.globalProfile = self.globalProfile
self.settingViewController.docProfile = self.docProfile
self.settingViewController.notebookProfile = self.notebookProfile
```

在点击插件图标的时候，通过

```ts
MN.studyController.view.addSubview(self.settingViewController.view)
```

在 MarginNote 界面上添加控制面板。这部分的代码在 `jsExtension/switchPanel.ts` 中。

---

modules

模块。每个模块有自己的选项，可以在控制面板中设置，有自己的动作。在 OhMyMN 中，模块分为了必选模块和可选模块，可选模块可以选择是否启用，但这属于 just ohmymn addon，在提供的模版中都为必选模块。

在 `modules/index.ts` 中注册。

profile

```
├── default.ts
├── index.ts
├── lang.ts
├── profileAction.ts
├── profileAuto.ts
├── rewrite.ts
├── typings.ts
├── updateDataSource.ts
└── utils.ts
```

Profile 配置文件。Profile 在 OhMyMN 中非常重要。

- `default.ts` 默认配置，其中的 key-value 将会用于控制面板的渲染，根据 value 类型限制其使用对应的菜单选项样式，并提供代码提示。所以要开发新的模块，必须要先在这里填入默认配置。
- `profileAction.ts` 配置管理的动作，用于导出和导入配置。
- `profileAuto.ts` 配置的读取和写入。
- `rewrite.ts` 重写配置文件，用于兼容旧版本。
- `updateDataSource.ts` 更新数据源，重新渲染控制面板。

dataSource

控制面板的数据源，用于渲染控制面板。根据模块中的 `settings` 生成。

coreModule.ts

整理模块中的属性和方法，汇集在一起。

addon.ts

一个全局变量，用于存储插件的一些信息以及数据。

> **注意**
>
> 全局变量在不同的 MN 窗口中共享。如果不想共享，请挂载到 self 上。

---

self.d.ts

`self` 的类型，用于代码提示。可以任意添加属性。self 上的属性在不同的 MN 窗口中无法共享。前面也说过，不同 OC 对象的 self 不一样。

---

main.ts

插件入口。

默认配置

在 `profile/default.ts` 文件中填写默认配置，分为

- `defaultGlobalProfile` 全局配置
- `defaultDocProfile` 文档配置，每个文档都有自己的配置。
- `defaultNotebookProfile` 笔记本配置，每个笔记本都有自己的配置。

除此之外，还有 `defaultTempProfile` ，但这属于 just ohmymn addon，用于缓存正则表达式对象，value 填写 `[]` 即可。

```ts
export const defaultTempProfile = {
  replaceParam: {
    customTag: [],
    customComment: [],
    customList: [],
    customReplace: [],
    customExtractTitle: [],
    customSimplify: [],
    customFormat: []
  },
  regArray: {
    customTitleSplit: [],
    customBeTitle: [],
    customDefLink: []
  }
}
```

用过 OhMyMN 的应该知道，OhMyMN 里正则表达式的使用主要分为 replace 函数 和 正则表达式（数组），分别对应了 `replaceParam` 和 `regArray` 。

默认配置填写的格式如下

```ts
export const defaultGlobalProfile = {
  autoreplace: {
    on: false,
    preset: [],
    customReplace: ""
  },
  additional: {
    lastVision: Addon.version
  }
}
```

默认配置与控制面板的菜单互相绑定，并且指定菜单选项的类型。请自行查看模块设置选项 API。

上面的 example 中，`on` 是 `Switch` 开关，`preset` 是 `MuiltSelect` 多选，`customReplace` 是 `Input` 输入框。对于多项选择，可以为空数组 `[]`，单项选择需要填入默认选项，比如 `[1]`。

`additional` 不属于控制面板的选项，通常用于存储一下额外数据。

在模块中，只要填写了 key，就会将默认配置自动绑定到控制面板的菜单中，并且进行类型检查。

```ts
export default defineConfig({
  name: "AutoReplace",
  key: "autoreplace",
  settings: [
    {
      key: "on",
      type: CellViewType.Switch,
      label: lang.on,
      auto: {
        modifyExcerptText: {
          index: 999,
          method({ node, text }) {
            return replaceText(node, text)
          }
        }
      }
    },
    {
      key: "preset",
      type: CellViewType.MuiltSelect,
      option: lang.preset.$option1,
      label: lang.preset.label
    },
    {
      key: "customReplace",
      type: CellViewType.Input,
      help: lang.custom_replace.help,
      bind: ["preset", 0],
      link: lang.custom_replace.link
    }
  ],
})
```

通过 `self.globalProfile.autoreplace.presets` 来读取全局配置。另外使用

- `self.docProfile` 读取当前文档的配置
- `self.notebookProfile` 读取当前笔记本的配置

对于 `defaultTempProfile`，需要使用 `self.tempProfile` 来读取。

国际化

通常我们创建一个 `lang.ts` 文件来存放两种语言的文本。使用 `i18n` 函数来创建一个国际化对象，会自动根据 MN 的语言得到对应的文字。

```ts
import { i18n } from "marginnote"

export default i18n({
  zh: {
    intro: `当前版本: ${Addon.version}`,
    double_link: "双击打开链接",
    profile: {
      $option5: [
        "配置 1",
        "配置 2",
        "配置 3",
        "配置 4",
        "初始化"
      ] as StringTuple<5>,
      label: "选择全局配置",
      help: "【仅当前笔记本】不同场景，不同配置。"
    }
  },

  en: {
    intro: `Current Version: ${Addon.version}`,
    double_link: "Double Click to Open Link",
    profile: {
      $option5: [
        "Profile 1",
        "Profile 2",
        "Profile 3",
        "Profile 4",
        "Initialize"
      ],
      label: "Select Global Profile",
      help: "[Only Current Notebook] Different Scenes, Different Profile."
    }
})
```

需要做一些约定：

OhMyMN 里菜单选项都是直接传入的字符串数组，虽然这样做非常简单，但是会直接影响到代码的逻辑，比如数组的长度不一样，所以我们需要使用 `StringTuple` 来约束数组长度。并且 key 必须以 `$` 开头，后面跟着数字，表示数组长度。

OhMyMN 模块开发

OhMyMN 是一个可以自动处理摘录的工具箱，也可以手动处理卡片。里面所有的功能都是模块化的，可以轻松扩展。

目前已经多达 15 个可选模块。点击查看更多细节。

如果你也想开发处理摘录或者处理卡片相关的功能，直接开发 OhMyMN 的模块会十分轻松，也更加强大。

OhMyMN 有个不成文的约定，如果会摘录时自动执行，模块名一般以 `auto` 开头，并且需要提供 `on` 的选项，用来关闭摘录时自动执行。

模块注册

所有的模块都需要在注册 `modules/index.ts`，才能被 OhMyMN 加载。分为可选模块和必选模块。

```ts
export const optionalModules = { shortcut }
export const requiredModules = { addon, magicaction4card, magicaction4text }
```

如何开发一个模块

目前 OhMyMN 还没有办法安装模块，只能通过修改源码来添加模块。

1. Fork ohmymn 仓库，然后将 fork 后的仓库 clone 到本地

```bash
git clone git@github.com:xxxxx/ohmymn.git
```

2. 使用 pnpm 安装依赖

```bash
pnpm install
```

3. 构建最新的 MarginNote API

```bash
pnpm run api build
```

4. ohmymn 是一个 monorepo 仓库，API，文档都在这一个仓库里，插件代码在 `packages/addon` 目录下，所有模块都在 `packages/addon/src/modules` 目录下。

```bash
pnpm run addon dev
```

启动插件开发模式，会自动将插件代码复制到 MarginNote 的插件目录下，就可以在 MarginNote 中看到插件了，如果之前没有安装，可能需要手动启用。之后修改代码，MarginNote 不会自动重新加载插件，需要重启 MarginNote。

5. pull request 到 ohmymn 仓库，等待审核。

填写默认配置

第一步是填写 默认配置，你需要想好模块的 key，key 是唯一的。以及控制面板的菜单选项。

创建一个新的模块

直接复制一个模块的文件夹最为简单，比如 `autoreplace`。`autoreplace` 是一个自动修改摘录的简单插件。这个插件由三个文件组成

```
├── index.ts
├── lang.ts
└── typings.ts
```

index.ts 是模块的入口文件，lang.ts 是模块的语言包，typings.ts 是模块的类型定义文件。

在模块中，只要填写了 key，就会将默认配置自动绑定到控制面板的菜单中，并且进行类型检查。`index.ts` 中的代码如下

```ts
import { type NodeNote, undoGroupingWithRefresh } from "marginnote"
import { renderTemplateOfNodeProperties } from "~/JSExtension/fetchNodePrope
import { defineConfig } from "~/profile"
import { CellViewType } from "~/typings"
import { doc, escapeDoubleQuote, string2ReplaceParam } from "~/utils"
import lang from "./lang"
import { AutoReplacePreset, ReplaceCard } from "./typings"

function replaceText(node: NodeNote, text: string) {
  const { preset } = self.globalProfile.autoreplace
  for (const set of preset) {
    switch (set) {
      case AutoReplacePreset.Custom:
        const { customReplace: params } = self.tempProfile.replaceParam
        if (!params?.length) continue
        text = params.reduce(
          (acc, param) =>
            acc.replace(
              param.regexp,
              renderTemplateOfNodeProperties(node, param.newSubStr)
            ),
          text
        )
    }
  }
  return text
}

export default defineConfig({
  name: "AutoReplace",
  key: "autoreplace",
  intro: lang.intro,
  link: doc("autoreplace"),
  settings: [
    {
      key: "on",
      type: CellViewType.Switch,
      label: lang.on,
      auto: {
        modifyExcerptText: {
          index: 999,
          method({ node, text }) {
```

```
              return replaceText(node, text)
            }
          }
        }
      },
      {
        key: "preset",
        type: CellViewType.MuiltSelect,
        option: lang.preset.$option1,
        label: lang.preset.label
      },
      {
        key: "customReplace",
        type: CellViewType.Input,
        help: lang.custom_replace.help,
        bind: ["preset", 0],
        link: lang.custom_replace.link
      }
    ],
    actions4card: [
      {
        type: CellViewType.ButtonWithInput,
        label: lang.replace_selected.label,
        key: "replaceCard",
        option: lang.replace_selected.$option2,
        method: ({ content, nodes, option }) => {
          undoGroupingWithRefresh(() => {
            if (option == ReplaceCard.UseAutoReplace) {
              nodes.forEach(node => {
                node.notes.forEach(note => {
                  const text = note.excerptText
                  if (text) note.excerptText = replaceText(node, text)
                })
              })
            } else if (content) {
              content = /^\(.*\)$/.test(content)
                ? content
                : `(/^.*$/gs, "${escapeDoubleQuote(content)}")`
              const params = string2ReplaceParam(content)
              nodes.forEach(node => {
                node.notes.forEach(note => {
                  const text = note.excerptText
                  if (text) {
```

```
                    note.excerptText = params.reduce((acc, params) => {
                      return acc.replace(
                        params.regexp,
                        renderTemplateOfNodeProperties(node, params.newSubStr)
                      )
                    }, text)
                  }
                })
              })
            }
          })
        }
      }
    ]
  })
```

详细 API 请查看 API 文档。

通常如果插件比较复杂，会将里面的函数抽离出来，放到一个单独的 `utils.ts` 文件`。
OhMyMN 自动执行的模块一般也会提供手动触发的动作，他们的代码是可以共用的。