

PaperBell: 基于 Obsidian 的学术写作管理系统

Shuang Song^{1,2*}, Sen Jiang²

¹ Max Planck Institute of Geoanthropology, Jena, Germany

² PaperBell Technology, Beijing, China

*Corresponding author: paperbell@songshgeo.com

Abstract — 本文介绍了 PaperBell 学术写作管理系统，一套基于 Obsidian、Longform、Pandoc 和 LaTeX 的现代化学术写作解决方案。传统的 Microsoft Word 写作存在版本追踪混乱、输入输出分离、格式调整分散注意力等问题。虽然 Markdown 写作具有纯文本、版本控制友好、易于自动化等优势，但传统 Markdown 工作流在长文本管理、导出格式和手动调整方面仍有局限。PaperBell 通过整合项目化管理、数据驱动写作、智能元数据管理和专业级 PDF 导出等核心特性，系统性地解决了这些问题。本文从引言、结果、讨论、方法四个方面全面介绍了 PaperBell 的设计理念、核心特性、配置方案、最佳实践和安装步骤，为学术研究者提供了一套完整的现代化写作工具链。

1 引言

1.1 传统学术写作的挑战

学术写作是科研工作者的核心技能之一，然而传统的 Microsoft Word 写作方式存在诸多局限性。首先，版本追踪混乱是困扰研究者的主要问题。当不断修改文章时，`manuscript_v1.docx`、`manuscript_v2_revised.docx`、`manuscript_final_really_final.docx` 等文件名层出不穷，版本管理混乱不堪（fig. 1）。本人写第一篇论文¹的时候就深受其苦，最后电脑里存了十几个版本。其次，输入与输出分离导致工作流割裂：研究者需要在文献管理软件（如 Zotero）、数据分析工具（如 R/Python）、写作软件（Word）之间频繁切换，手动复制粘贴数据和引用。第三，格式调整分散注意力：研究者常常陷入字体、行距、页边距的调整中，无法专注于内容创作本身。对我这样注重科研美学的人，经常调着调着格式就忘了自己的写作主线。

上述这些问题不仅降低了写作效率，还增加了错误风险。例如，当实验数据更新时，需要手动在文档中逐一查找并修改所有相关数值；当投稿不同期刊时，需要重新调整格式和引用样式，耗时耗力。

1.2 Markdown 写作的优势与局限

近年来，越来越多的学者开始在 Obsidian 等知识管理工具中使用 Markdown 进行写作。Markdown 是一种轻量级标记语言，具有显著优势：

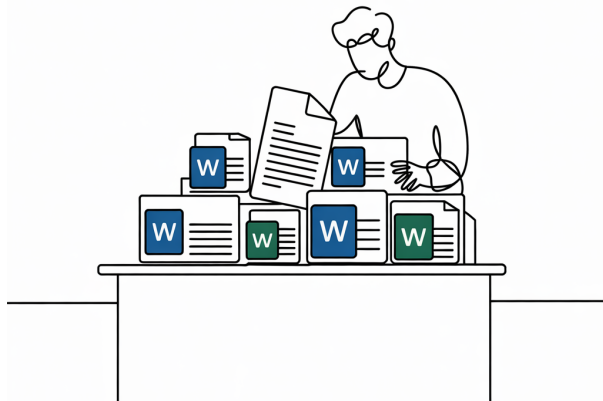


Figure 1 – 科研人日常：整理不同版本的文稿

1. 纯文本格式：文件体积小（通常只有 Word 文档的 1/10），不易损坏崩溃，即使在大型项目中也能保持稳定
2. 版本控制友好：纯文本易于使用 Git 等版本控制系统，可以精确追踪每一行修改，实现真正的协作写作
3. 自动化潜力：纯文本易于批量处理，支持正则表达式替换、脚本化操作和变量更新
4. 跨平台兼容：任何文本编辑器都能打开编辑，不依赖特定软件

然而，传统的 Markdown 学术写作仍存在痛点。长文本管理困难是首要问题：一篇完整的论文动辄数千字，单个 Markdown 文件难以管理和导航。导出格式不佳是另一大障碍：默认的 Markdown 转 PDF 功能生成的文档格式简陋，无法满足期刊投稿的专业要求。手动调整格式的问题依然存在：研究者仍需花费大量时间处理引用格式、图表编号、交叉引用等细节 (fig. 2)。

1.3 PaperBell 工作流的创新价值

PaperBell 是一套基于 Obsidian 的学术写作管理系统，通过整合 **Longform**、**Pandoc**、**LaTeX** 等先进工具，系统性地解决了上述问题：

1. 项目化管理：通过 Longform 插件实现论文项目的结构化组织，支持章节拆分、场景排序和多项目并行
2. 数据驱动写作：通过占位符系统（如 `{{results.correlation}}`）实现数据与文本的动态绑定，数据更新时自动同步
3. 专业级导出：通过自定义 Pandoc 模板和 LaTeX 引擎，生成符合期刊要求的高质量 PDF 文档
4. 完整工作流集成：从 Zotero 文献管理到最终 PDF 输出的全流程数字化，打通输入-处理-输出链路



Figure 2 – 使用 Markdown 进行写作时，如何转化成专业的论文排版是让人头疼的

PaperBell 的核心优势在于：保留 **Markdown** 写作的灵活性，同时提供 **Word** 级别的专业输出。研究者可以专注于内容创作，将格式调整、引用管理、数据更新等重复性工作交给自动化工具处理。

1.4 本文的目标与结构

本文旨在全面介绍 PaperBell 学术写作工作流的实现原理、配置方法和最佳实践。我们将从以下四个方面展开：

1. 引言（本章）：阐述传统写作方式的局限性和 PaperBell 的创新价值
2. 结果：展示 PaperBell 的核心特性和功能演示
3. 讨论：探讨常用配置方案、最佳实践和进阶自定义技巧
4. 方法：详细说明安装步骤、配置方法和故障排除指南

通过阅读本文，研究者将能够：

- 理解 PaperBell 工作流的设计理念和技术架构
- 掌握从项目创建到 PDF 导出的完整操作流程
- 学会针对不同期刊要求进行定制化配置
- 解决常见的技术问题和格式调整需求

PaperBell 不仅是一套工具集合，更是一种现代化的学术写作范式：以内容为核心，以自动化为手段，以专业输出为目标。让我们开始探索这套强大的学术写作系统。

2 结果

本章展示 PaperBell 学术写作系统的核心特性和功能实现，通过具体示例说明各个组件如何协同工作，实现从写作到发布的完整流程。

2.1 特性一：项目化的长文本管理

2.1.1 Longform 插件集成

PaperBell 通过 Longform 插件实现论文的项目化管理，解决了单文件长文本难以维护的问题。每个研究项目以文件夹形式组织：

```
Outputs/DEMO_Manuscript/  
├─ source/ # 章节源文件  
│   ├── Introduction.md  
│   ├── Methods.md  
│   ├── Results.md  
│   └── Discussion.md  
├─ imgs/ # 图片资源  
├─ Index.md # 元数据配置  
├─ results.json # 分析数据  
└─ manuscript.md # 编译输出
```

这种结构化组织具有显著优势：

- 模块化写作：每个章节独立编辑，降低认知负担（建议每章 < 500 行）
- 灵活排序：通过拖拽调整章节顺序，无需重命名文件
- 协作友好：多人可同时编辑不同章节，避免版本冲突
- 版本追踪：Git 可精确追踪每个章节的修改历史

2.1.2 场景（Scenes）系统

Longform 的场景系统支持复杂的文档结构。研究者可以创建章节组（如 Part I: Background），也可以将章节标记为草稿或待审阅状态：

```
---  
scene_title: "Methods"  
tags: [draft, peer-review]  
status: in-progress  
---
```

编译时，Longform 会按照预设顺序自动合并所有章节，生成完整手稿。需要注意小标题的层级排布：

标题层级说明

在 PaperBell 的 Longform 写作中，标题层级遵循以下规则：

1. 文档标题 (title in YAML) → LaTeX 的 \maketitle
2. **Scene** 标题 (scene_title in frontmatter) → LaTeX 的 \section (二级标题 ##)
3. **Scene** 内的标题 → 从 \subsection 开始 (三级标题 ###)

这样可以确保最终 PDF 文档的标题层级结构清晰一致。

2.2 特性二：数据驱动的动态写作

Warning

注意，本功能通过 Longform 的编译脚本 “Replace placeholders from JSON” 实现，如果你需要启用此功能，请将在 “PaperBell 编译标准流程” (PaperBell-Standard) 的基础上，将该脚本加入编译流程，并置于保存笔记 (Save as Note) 流程之前。

2.2.1 占位符系统

PaperBell 的占位符系统实现了数据与文本的动态绑定。研究者在写作时使用占位符引用数据：

我们整合了 `{{results1.n_datasets}}` 个重建数据集，
Pearson 相关系数为 `{{results1.corr}}` ($p < 0.01$)。

对应的 `results.json` 文件包含实际数值：

```
{
  "results1": {
    "n_datasets": 10,
    "corr": 0.41,
    "n_pass_years": 30,
    "sig_sites_ratio": 0.9
  }
}
```

编译时，自定义脚本 替换结果占位符 .js 自动将占位符替换为实际数值，生成：

我们整合了 10 个重建数据集，
Pearson 相关系数为 0.41 ($p < 0.01$)。

2.2.2 路径解析能力

占位符系统支持复杂的数据结构访问：

- 嵌套对象：`{{study.phase1.participants}}` → 访问 `study.phase1.participants`
- 数组索引：`{{datasets[0].name}}` → 访问 `datasets` 数组第一个元素的 `name` 属性
- 混合路径：`{{results.sites[2].correlation}}` → 嵌套对象与数组组合

这种灵活性使得研究者可以高效管理复杂的分析结果，实现真正的数据驱动写作。

2.2.3 自动化优势

当实验数据更新时，研究者只需修改 `results.json` 文件，重新编译即可更新文档中所有相关数值。这避免了手动查找替换的错误风险，特别适合多次修订的论文项目。

2.3 特性三：智能化的元数据管理

2.3.1 YAML 前置元数据

PaperBell 使用 YAML 格式集中管理论文元数据，包括标题、作者、机构、摘要、关键词等：

```
---
title: "Archival documentation reveals perceptual bias"
date: "2025-10-11"

authors:
  - name: "Shuang Song"
    affiliation: [1, 2, 3]
    corresponding: "song@gea.mpg.de"
  - name: "Bo Hu"
    affiliation: [4]

affiliations:
  - index: 1
    name: "Max Planck Institute of Geoanthropology"
  - index: 2
    name: "Beijing Normal University"

abstract: "It is well-documented that..."
keywords: ["collective memory", "extreme events"]

target: "Nature Human Behaviour"
acronym: "DEMO"
csl: "nature"
---
```

2.3.2 自动化元数据注入

Longform 自定义脚本 编译后增加头文件 .js 在编译时自动读取元数据，并注入到最终文档。该脚本具有以下特点：

1. 用户选项优先：如果在编译时手动指定选项（如引用样式），会覆盖元数据中的默认值
2. 完整作者信息处理：自动解析作者-机构索引关系，标记通讯作者
3. 灵活格式配置：支持行号、图表位置、引用样式等动态选项

2.3.3 多期刊适配

同一份手稿可以通过调整元数据快速适配不同期刊：

```
# 投稿 Nature 系列
csl: "nature"
target: "Nature Human Behaviour"

# 投稿 APA 期刊
csl: "apa"
target: "Journal of Environmental Psychology"
```

编译时自动应用对应的引用格式和排版样式，无需手动调整。

2.4 特性四：专业级的 PDF 导出

2.4.1 PaperBell 学术模板

PaperBell 提供了基于 Eissvogel 定制的 LaTeX 模板 paperbell.tex，专为学术手稿设计。模板特性包括：

1. 统一字体系统：全文使用 Times New Roman（标题、正文、图表标题、页眉页脚），符合国际期刊标准
2. 智能行号控制：支持 lineno: "true" 选项显示行号，便于审稿和讨论；参考文献部分自动关闭行号
3. 图表位置灵活性：支持 figures-at-end: "true" 将所有图表置于文末（参考文献之前）
4. 页眉页脚定制：
 - 左页眉：Manuscript: {acronym}
 - 右页眉：{date}
 - 左页脚：Submission: {target}
 - 右页脚：页码
5. 图表标题格式化：无缩进、两端对齐、Times New Roman 字体、加粗标签
6. 紧凑学术格式：优化标题间距，移除不必要的空白

2.4.2 Pandoc 编译流程

PaperBell 使用 Pandoc 作为文档转换引擎，配置文件 `pdf.yaml` 定义了完整的编译流程：

```
from: markdown+tex_math_single_backslash+wikilinks_title_after_pipe
pdf-engine: xelatex
template: paperbell.latex

filters:
  - pandoc-crossref # 交叉引用
  - citeproc # 引用处理

metadata:
  numbersections: true
  link-citations: true
```

编译过程如下：

1. **Markdown 解析**：支持扩展语法（数学公式、Wiki 链接、表格）
2. 交叉引用处理：pandoc-crossref 解析 `{#fig:label}` 和 `[@eq:label]`
3. 引用格式化：citeproc 根据 CSL 文件格式化参考文献
4. **LaTeX 渲染**：xelatex 生成最终 PDF

2.4.3 多种导出选项

PaperBell 支持通过 Obsidian Enhancing Export 插件或命令行进行 PDF 导出：

插件导出（推荐）：

- 在 Obsidian 中右键点击 `manuscript.md`
- 选择 “Export with Enhancing Export”
- 选择 “PaperBell Academic” 配置
- 自动生成 PDF

命令行导出：

```
pandoc manuscript.md -o output.pdf \
  --defaults="40 - Obsidian/脚本/Pandoc/defaults/pdf.yaml" \
  -M lineno=true \
  -M figures-at-end=false
```

命令行方式适合批量编译和自动化流程（如 GitHub Actions）。

2.5 特性五：完整的引用管理集成

2.5.1 Zotero 联动

PaperBell 通过 ZotLit 插件实现与 Zotero 的深度集成：

1. 文献导入：在 Zotero 中标注文献后，右键选择“Create Literature Note(s)”，自动导入到 Obsidian
2. 高亮同步：Zotero 中的彩色高亮和批注自动转换为 Obsidian 笔记
3. 引用键生成：自动生成引用键（如 song2025collective），用于文中引用

2.5.2 引用 workflow

在写作时使用 Pandoc 引用语法：

已有研究表明集体记忆影响气候适应 `[@song2025collective; @smith2023memory]`。
如 `@jones2024perception` 所述，认知偏差广泛存在。

编译时自动格式化为：

已有研究表明集体记忆影响气候适应 (Song et al., 2025; Smith et al., 2023)。如 Jones et al. (2024) 所述，认知偏差广泛存在。

参考文献列表自动生成并格式化：

References

- Jones, A., et al. (2024). Perception and cognition. *Nature*, 123, 456-789.
- Smith, B., et al. (2023). Memory studies. *Science*, 234, 567-890.

2.5.3 多样化的引用样式

PaperBell 支持数千种期刊的引用样式（通过 CSL 文件）：

- **Nature** 系列：数字上标 [1, 2]
- **APA** 格式：作者-日期 (Author, 2024)
- 国标格式：[1] 作者, 标题, 期刊, 年份

只需在元数据中指定 `csl: "nature"` 或 `csl: "apa"`，即可自动应用对应样式。

2.6 特性六：交叉引用与编号系统

2.6.1 图表自动编号

使用 `pandoc-crossref` 过滤器，PaperBell 实现了智能的图表编号：

![研究区域地图] (imgs/study_area.png) {#fig:map}

如 [fig:map] 所示，研究区域位于...

编译后自动生成：

Figure 1. 研究区域地图

如 Figure 1 所示，研究区域位于...

2.6.2 公式编号

数学公式同样支持自动编号：

我们使用以下模型：

$y = \beta_0 + \beta_1 x + \epsilon$ {#eq:model}

根据 [eq:model]，我们估计参数...

编译后：

我们使用以下模型：

$$y = \beta_0 + \beta_1 x + \epsilon \quad (1)$$

根据 Equation 1，我们估计参数...

2.6.3 表格编号

表格编号与图表一致：

变量	均值	标准差
温度	15.3	2.1
降水	800	120

Table: 描述统计 {#tbl:stats}

[tbl:stats] 展示了描述统计结果。

编译后自动编号并生成引用。

2.7 特性七：版本控制与协作

2.7.1 Git 集成

PaperBell 项目天然适合 Git 版本控制：

```
# 追踪修改
git add Outputs/DEMO_Manuscript/source/Results.md
git commit -m "Add correlation analysis"

# 创建版本标签
git tag -a v1.0-draft -m "First complete draft"

# 多人协作
git checkout -b coauthor-revision
```

纯文本格式使得 Git 可以精确显示每一行的修改，方便审阅和回滚。

2.7.2 协作写作场景

PaperBell 支持多种协作模式：

1. 分章节协作：作者 A 编辑 Methods.md，作者 B 编辑 Results.md，互不干扰
2. 审阅-修订循环：导出带行号的 PDF 供审稿人标注，根据行号快速定位修改
3. 版本对比：使用 Git diff 精确对比不同版本的修改内容

2.7.3 云同步支持

除 Git 外，PaperBell 项目也兼容各种云同步服务：

- **iCloud**：Mac 用户的自动同步方案
- **OneDrive/Dropbox**：跨平台云存储
- **Obsidian Sync**：官方加密同步服务

小文件特性确保同步速度快且不易产生冲突。

2.8 特性八：可扩展的自动化系统

2.8.1 自定义 Longform 脚本

PaperBell 提供了两个核心自定义脚本，用户也可以编写自己的脚本：

1. 编译后增加头文件.js:

- 从 Index.md 读取元数据
- 注入 YAML 前置内容
- 支持用户选项覆盖

2. 替换结果占位符.js:

- 解析 results.json
- 替换 {{path.to.value}} 占位符
- 支持嵌套对象和数组

2.8.2 Pandoc 过滤器

用户可以编写 Lua 过滤器扩展 Pandoc 功能:

```
-- 自动添加图表前缀
function Image(elem)
  elem.caption = pandoc.Str("Figure: ") .. elem.caption
  return elem
end
```

2.8.3 批量处理脚本

PaperBell 支持批量编译多个项目:

```
#!/bin/bash
for project in Outputs/*/manuscript.md; do
  pandoc "$project" --defaults=pdf.yaml -o "${project%.md}.pdf"
done
```

这些自动化能力使 PaperBell 能够适应各种复杂的科研写作场景。

2.9 小结

本章展示了 PaperBell 的八大核心特性: 项目化管理、数据驱动写作、智能元数据、专业导出、引用集成、交叉引用、版本控制和可扩展性。这些特性协同工作, 构建了一套完整的现代化学术写作系统。

下一章将讨论如何根据不同场景配置 PaperBell, 分享最佳实践和进阶自定义技巧。

3 讨论

本章探讨 PaperBell 在不同学术写作场景下的配置方案, 分享经过实践验证的最佳实践, 并介绍进阶自定义技巧。

3.1 常用配置方案

Pandoc Defaults 与 Template 配置说明

PaperBell 的导出系统基于 Pandoc 的 defaults 文件配置。在 40 - Obsidian/脚本/pandoc/defaults/ 目录下，我们提供了两个预配置的 defaults 文件：

- **paperbell.yaml** - macOS/Linux 系统使用（中文字体：Songti SC、Heiti SC、STFangsong）
- **paperbell-windows.yaml** - Windows 系统使用（中文字体：SimSun、SimHei、FangSong）

这两个配置文件的主要区别在于中文字体设置，以适应不同操作系统的字体可用性。

3.1.1 自动检测与手动指定

在 Longform 编译时，“Add YAML Metadata”脚本会：

1. 自动检测操作系统（如果 Template 选项留空）：
 - macOS/Linux → 使用 paperbell 模板
 - Windows → 使用 paperbell-windows 模板
2. 支持手动指定：在 Template 文本框中输入任何模板名称，如：
 - paperbell - 使用 Unix 版本
 - paperbell-windows - 使用 Windows 版本
 - eisvogel - 使用 Eisvogel 模板
 - my-custom - 使用你自己的自定义模板

3.1.2 创建自定义工作流

你完全可以定义自己的 defaults 配置文件和工作流：

1. 在 40 - Obsidian/脚本/pandoc/defaults/ 创建新的 .yaml 文件
2. 在 40 - Obsidian/脚本/pandoc/templates/ 创建新的 .latex 模板
3. 在 Longform 编译时的 Template 选项中指定你的配置名称

例如，创建 my-workflow.yaml 后，在编译时输入 my-workflow 即可使用。

相关文件路径：

```
40 - Obsidian/脚本/pandoc/
├── defaults/ # Pandoc 默认配置文件
│   ├── paperbell.yaml
│   ├── paperbell-windows.yaml
│   ├── beamer.yaml
│   └── crossref.yaml
├── templates/ # LaTeX 模板文件
│   └── paperbell.latex
├── filters/ # Pandoc Lua 过滤器
└── csl/ # 引用样式文件
```

3.1.3 方案一：快速投稿配置（Clean Submission）

适用场景：论文已基本定稿，准备首次投稿

配置要点：

```
---
title: "Your Research Title"
date: "2025-10-11"
target: "Nature Human Behaviour"
acronym: "YourAcronym"
csl: "nature"

# 关键设置
lineno: "false" # 不显示行号
figures-at-end: "false" # 图表嵌入正文
titlepage: true # 生成封面页
---
```

Longform 编译步骤：

1. Strip Frontmatter
2. Remove Links
3. Prepend Title
4. Concatenate Text
5. Add YAML Metadata (Note Name: Index)
6. Replace placeholders from JSON
7. Save as Note

导出设置：

- 使用 “PaperBell Academic” 配置
- 确保所有图片为高分辨率 (≥ 300 DPI)
- 检查参考文献格式与目标期刊一致

检查清单：

- ☐ 字数符合期刊要求
- ☐ 图表数量在限制内
- ☐ 所有引用在.bib 文件中
- ☐ 作者信息完整准确
- ☐ 摘要和关键词符合要求

3.1.4 方案二：审稿讨论配置（Review Version）

适用场景：发送给合作者或审稿人审阅

配置要点：

```
---
lineno: "true" # 显示行号便于标注
figures-at-end: "false" # 图表嵌入便于阅读
titlepage: true
---
```

为什么显示行号？

- 便于审稿人精确指出需要修改的位置
- 便于作者团队讨论时引用具体段落
- 便于记录修订历史（如 “Line 156: revised as suggested”）

协作建议：

- 导出 PDF 时在文件名中注明版本和日期：manuscript_v2_20251011_review.pdf
- 使用 Git 标签标记审稿版本：git tag -a review-v2 -m "Sent to co-authors"
- 创建修订记录文档，记录每个行号的修改内容

3.1.5 方案三：修订响应配置（Revision Response）

适用场景：响应审稿意见，准备修订稿

工作流程：

1. 创建修订分支：

```
git checkout -b revision-round1
```

2. 配置双版本对比：

```
# original.yaml
lineno: "true"
csl: "nature"

# revised.yaml
lineno: "true"
csl: "nature"
highlight-changes: true # 标记修改内容（需自定义过滤器）
```


3. 使用占位符记录修改：在 `results.json` 中添加修订信息：

```
{
  "revision": {
    "response_to_reviewer1": "We have added additional analysis as
    ↪ suggested...",
    "new_sample_size": 150,
    "previous_sample_size": 120
  }
}
```

在修订说明文档中引用：

```
**Reviewer 1, Comment 3**： Sample size seems insufficient.

**Response**： We have increased the sample size from
{{revision.previous_sample_size}} to {{revision.new_sample_size}}.
```

3.1.6 方案四：期刊特定配置

```
---
csl: "nature"
target: "Nature Human Behaviour"
reference-section-title: "References"

# Nature 特定要求
lineno: "false"
figures-at-end: "false" # 图表嵌入正文
titlepage: false # 不需要单独封面
numbersections: true
---
```

3.1.6.1 Nature 系列期刊 注意事项：

- Nature 系列使用数字上标引用 [1, 2]
- 图表标题格式： **Figure 1** | Title in sentence case.
- 参考文献按引用顺序排列

```
---
csl: "science"
target: "Science"
reference-section-title: "References and Notes"

# Science 特定要求
lineno: "true" # 初稿需要行号
figures-at-end: "true" # 图表置于文末
titlepage: false
---
```

3.1.6.2 Science 系列期刊 注意事项:

- Science 要求图表和说明分离
- 参考文献格式较为严格, 需仔细检查
- 补充材料需单独准备

```
---
csl: "apa"
target: "Journal of Environmental Psychology"
reference-section-title: "References"

# APA 格式要求
lineno: "false"
figures-at-end: "false"
titlepage: true # 需要封面页
abstract-title: "Abstract"
keywords-title: "Keywords"
---
```

3.1.6.3 社会科学期刊 (APA 格式) 注意事项:

- APA 使用作者-日期引用格式 (Author, 2024)
- 参考文献按字母顺序排列
- 标题层级有严格规定

3.2 最佳实践

3.2.1 实践一: 元数据集中管理

问题: 多个项目的元数据分散, 难以统一维护。

解决方案：创建元数据模板和复用策略。

实施步骤：

1. 创建作者信息库 40 - Obsidian/authors.yaml:

```
authors:
  songshgeo:
    name: "Shuang Song"
    email: "song@gea.mpg.de"
    affiliations:
      - "Max Planck Institute of Geoanthropology"
      - "Beijing Normal University"
    orcid: "0000-0002-XXXX-XXXX"

  coauthor1:
    name: "Bo Hu"
    email: "bohu@nju.edu.cn"
    affiliations:
      - "Nanjing University"
```

2. 在新项目中引用：

```
# 使用 templater 插件动态插入
<%*
const authors = await tp.file.include("authors");
%>
```

3. 维护机构索引映射表，确保一致性。

优势：

- 作者信息一处修改，所有项目同步更新
- 避免拼写错误和格式不一致
- 便于管理大型合作项目（多作者、多机构）

3.2.2 实践二：数据与文本分离

原则：所有可能变化的数值都应使用占位符，而非硬编码。

示例：

□ 不推荐：

我们分析了 150 个样本，发现相关系数为 0.45 ($p < 0.01$)。

□ 推荐：

我们分析了 `{{study.n_samples}}` 个样本，
发现相关系数为 `{{study.correlation}}` ($p < {{study.pvalue}}$)。

对应 `results.json`：

```
{
  "study": {
    "n_samples": 150,
    "correlation": 0.45,
    "pvalue": 0.01
  }
}
```

优势：

- 数据更新时无需逐行查找修改
- 降低人为错误（如漏改某个数值）
- 便于进行敏感性分析（替换不同数据集）
- 便于审稿人要求的额外分析

进阶技巧：

- 在 `results.json` 中包含数据来源和计算日期：

```
{
  "metadata": {
    "generated_date": "2025-10-11",
    "script": "analysis/main_analysis.R"
  },
  "study": {
    "n_samples": 150
  }
}
```

3.2.3 实践三：渐进式写作策略

问题：论文初稿往往需要反复调整结构和内容。

解决方案：采用渐进式写作，充分利用 Longform 的灵活性。

阶段一：大纲阶段（Outline Phase）

创建章节骨架，每个文件只包含标题和要点：

```
# Introduction

### Background
- Point 1: Traditional methods
- Point 2: Current challenges
- Point 3: Our innovation

### Research Questions
- RQ1: ...
- RQ2: ...
```

阶段二：初稿阶段（Draft Phase）

逐章节扩展内容，标记状态：

```
---
status: draft
completeness: 60%
next_steps:
  - Add literature review
  - Expand methodology
---
```

阶段三：修订阶段（Revision Phase）

使用 Git 分支管理不同版本：

```
git checkout -b draft-v2

# 大幅修改
git checkout main
git merge draft-v2
```

阶段四：抛光阶段（Polish Phase）

- 启用 **Linting** 检查拼写和语法
- 检查段落逻辑和过渡
- 统一术语和表达方式

优势：

- 降低初稿心理压力（先完成后完美）
- 便于追踪写作进度
- 易于回滚不成熟的修改

3.2.4 实践四：图表管理规范

问题：图片文件命名混乱，难以维护。

解决方案：建立图表命名和组织规范。

命名规范：

```
imgs/  
├─ fig1_study_area_map.png # 主文图  
├─ fig2_temporal_trends.png  
├─ fig3_spatial_patterns.png  
├─ figS1_validation_results.png # 补充图  
├─ figS2_sensitivity_analysis.png  
└─ raw/ # 原始图片  
    └─ fig1_raw_from_gis.png
```

命名模式：{type}{number}_{descriptive_name}.{ext}

在文中引用：

```
![Study area and sampling  
↪ sites] (imgs/fig1_study_area_map.png) {#fig:study-area width=80%}
```

As shown in `[@fig:study-area]`, our study area covers...

版本控制：

使用 `.gitattributes` 管理图片：

```
*.png filter=lfs diff=lfs merge=lfs -text  
*.jpg filter=lfs diff=lfs merge=lfs -text  
*.pdf filter=lfs diff=lfs merge=lfs -text
```

大型图片文件使用 Git LFS (Large File Storage)。

优势：

- 文件名即说明，易于理解
- 便于快速定位和替换
- 版本控制更高效

3.2.5 实践五：引用管理工作流

工作流程：

1. 文献阅读阶段（Zotero）：
- 使用彩色高亮标注关键内容：

• □ 黄色：重要发现

• □ 绿色：方法论

• □ 蓝色：理论框架

• □ 红色：局限性/疑问

• 添加标签：#project/DEMO、#method/ABM
2. 笔记导入阶段（ZotLit）：
- 右键选择 “Create Literature Note(s)”

• 自动导入到 Inputs/Zotero/

• 验证引用键格式（如 song2025collective）
3. 写作引用阶段（Obsidian）：
- 使用自动完成：输入 [@song 触发补全

• 多引用：[@author1; @author2; @author3]

• 文内引用：@author2024 demonstrated that...
4. 编译检查阶段（Pandoc）：
- 检查是否有未解析的引用（会显示为 [@unknown]）

• 验证参考文献格式与期刊要求一致

• 检查引用顺序（Nature 系列按引用顺序，APA 按字母顺序）

常见问题处理：

问题	解决方案
引用无法解析	检查.bib 文件路径，验证引用键拼写
参考文献格式错误	更换 CSL 文件，检查 Zotero 中的元数据
多次引用同一文献	使用 -@author 隐藏作者名，只保留年份

3.2.6 实践六：模板复用与项目快速启动

问题：每次创建新项目都需要重新配置，效率低下。

解决方案：创建项目模板和快速启动脚本。

创建项目模板：

1. 在 40 - Obsidian/模板/ 创建 Longform_Project_Template/:

```
Longform_Project_Template/  
├─ source/  
│   ├── 01_Introduction.md  
│   ├── 02_Methods.md  
│   ├── 03_Results.md  
│   └── 04_Discussion.md  
├─ imgs/.gitkeep  
├─ Index.md  
└─ results.json
```

2. Index.md 包含常用元数据框架:

```
---  
title: "{{TITLE}}"  
date: "{{DATE}}"  
authors:  
  - name: "{{AUTHOR}}"  
    affiliation: [1]  
    corresponding: "{{EMAIL}}"  
affiliations:  
  - index: 1  
    name: "{{INSTITUTION}}"  
abstract: ""  
keywords: []  
target: ""  
acronym: ""  
csl: "nature"  
lineno: "false"  
figures-at-end: "false"  
---
```

3. 使用 Templater 插件创建快速启动命令:

```
// QuickAdd script: Create Longform Project  
const projectName = await tp.system.prompt("Project name");  
const acronym = await tp.system.prompt("Project acronym");  
  
const template = "40 - Obsidian/模板/Longform_Project_Template";  
const target = `Outputs/${projectName}`;  
  
await tp.file.create_new("", projectName, false, tp.file.folder(target));  
// Copy template files...
```


优势:

- 新项目 1 分钟内完成初始化
- 确保项目结构一致性
- 避免遗漏必要文件

3.3 进阶自定义

3.3.1 自定义一：扩展占位符功能

需求：希望占位符支持计算和格式化（如保留小数位数、千位分隔符）。

解决方案：扩展 替换结果占位符.js 脚本。

示例代码：

```
// 支持格式化语法: {{value | format}}
const pattern = /{{\s*([^\|]+?)\s*(\\|\s*([^\|]+?)?)?\s*}}/g;

const replaced = input.contents.replace(pattern, (match, path, _, format) => {
  let value = getByPath(data, path.trim());

  if (format) {
    const fmt = format.trim();
    if (fmt.startsWith('toFixed')) {
      const digits = parseInt(fmt.match(/\d+/)[0]);
      value = Number(value).toFixed(digits);
    } else if (fmt === 'percent') {
      value = (Number(value) * 100).toFixed(1) + '%';
    } else if (fmt === 'comma') {
      value = Number(value).toLocaleString();
    }
  }

  return String(value);
});
```

使用示例：

```
相关系数为 {{study.correlation | toFixed2}}
样本量为 {{study.n_samples | comma}}
显著性比例为 {{study.sig_ratio | percent}}
```

输出：

相关系数为 0.45
样本量为 1,250
显著性比例为 87.3%

3.3.2 自定义二：多语言支持

需求：同一手稿需要中英文两个版本。

解决方案：使用条件渲染和多元数据文件。

实施方案：

1. 创建双语元数据：

```
{
  "title": {
    "en": "Perceptual bias in collective memory",
    "zh": " 集体记忆中的感知偏差"
  },
  "abstract": {
    "en": "It is well-documented that...",
    "zh": " 已有充分研究表明..."
  }
}
```

2. 修改脚本支持语言选择：

```
{
  id: "language",
  name: "Document Language",
  type: "Dropdown",
  options: [
    { value: "en", label: "English" },
    { value: "zh", label: " 中文" }
  ],
  default: "en"
}

// In compile function
const lang = context.optionValues["language"];
title = metadata.title[lang] || metadata.title;
```

3. 章节文本也可以使用占位符：

```
# {{headings.introduction}}
```

```
{{text.intro_para1}}
```

3.3.3 自定义三：高级交叉引用

需求：希望引用时自动包含页码或章节信息。

解决方案：创建自定义 Pandoc Lua 过滤器。

示例过滤器（enhanced_crossref.lua）：

```
function Cite(elem)
  -- 为图表引用添加 "on page X"
  if elem.citations[1].id:match("^fig:") then
    return {
      pandoc.Str("Figure"),
      pandoc.Space(),
      elem,
      pandoc.Str(" (on page \\pageref{" .. elem.citations[1].id .. "})")
    }
  end
  return elem
end
```

在 pdf.yaml 中启用：

```
filters:
- enhanced_crossref.lua
- pandoc-crossref
- citeproc
```

3.3.4 自定义四：自动化补充材料

需求：自动生成补充材料文档（Supplementary Information）。

解决方案：创建独立的 Longform 项目和编译配置。

项目结构：

```
Outputs/DEMO_Manuscript/
├─ main/
└─ manuscript.md # 主文档
```

```
└─ supplementary/
  └─ source/
    ├── S1_Methods.md
    ├── S2_Results.md
    └── S3_Figures.md
  └─ Index_SI.md
  └─ supplementary.md # 补充材料编译输出
```

配置差异：

```
# Index_SI.md
---
title: "Supplementary Information"
subtitle: "Perceptual bias in collective memory"
numbersections: true
section-numbering: "S" # 章节编号前缀 S
figure-numbering: "S" # 图表编号前缀 S
---
```

批量编译：

```
# compile_all.sh
pandoc main/manuscript.md -o manuscript.pdf --defaults=pdf.yaml
pandoc supplementary/supplementary.md -o SI.pdf --defaults=pdf_si.yaml
```

3.3.5 自定义五：集成外部数据源

需求：直接从数据库或 API 获取最新数据，而非手动更新 JSON。

解决方案：创建数据同步脚本。

示例（Python）：

```
# sync_data.py
import json
import requests
from pathlib import Path

def fetch_analysis_results(project_id):
    """ 从远程服务器获取分析结果 """
```

```
response = requests.get(f"https://api.example.com/results/{project_id}")
return response.json()

def update_results_json(project_path, data):
    """ 更新 results.json """
    results_file = Path(project_path) / "results.json"
    with open(results_file, 'w') as f:
        json.dump(data, f, indent=2)

if __name__ == "__main__":
    data = fetch_analysis_results("DEMO_2025")
    update_results_json("Outputs/DEMO_Manuscript", data)
    print("✅ Data synchronized successfully")
```

配合 **Makefile** 自动化:

```
.PHONY: sync compile all

sync:
    python scripts/sync_data.py

compile:
    pandoc Outputs/DEMO_Manuscript/manuscript.md -o output.pdf
    ↪ --defaults=pdf.yaml

all: sync compile
```

运行 `make all` 即可自动同步数据并编译文档。

3.4 小结

本章讨论了 **PaperBell** 在不同场景下的配置方案，包括快速投稿、审稿讨论、修订响应和期刊特定配置。我们分享了六大最佳实践：元数据集中管理、数据文本分离、渐进式写作、图表管理规范、引用 workflow 和模板复用。最后，我们探讨了五种进阶自定义方案：扩展占位符、多语言支持、高级交叉引用、自动化补充材料和外部数据集成。

这些方案和实践经过实际项目验证，能够显著提升学术写作效率和质量。下一章将详细介绍 **PaperBell** 的安裝配置方法和故障排除技巧。

4 方法

本章提供 **PaperBell** 学术写作系统的完整安装指南、详细配置步骤和常见问题的解决方案。

4.1 前期准备

4.1.1 系统要求

4.1.1.1 操作系统

- macOS 10.15+
- Windows 10/11
- Linux (Ubuntu 20.04+)

4.1.1.2 必需软件

- Obsidian 1.0.0+
- Pandoc 2.19+
- LaTeX 发行版 (TeX Live 2022+ 或 MacTeX 2022+)
- Zotero 6.0+ (用于文献管理)

4.1.1.3 硬件建议

- RAM: $\geq 8\text{GB}$ (编译大型文档需要)
- 存储: $\geq 5\text{GB}$ 可用空间 (LaTeX 发行版约 4GB)
- 处理器: 现代多核处理器 (编译速度更快)

4.1.2 软件安装

4.1.2.1 1. 安装 **Obsidian** 访问 Obsidian 官网 下载对应平台的安装包。

```
brew install --cask obsidian
```

4.1.2.2 macOS

4.1.2.3 Windows 下载 .exe 安装程序

```
# Debian/Ubuntu
wget https://github.com/obsidianmd/obsidian-
↪ releases/releases/download/v1.4.0/obsidian_1.4.0_amd64.deb
sudo dpkg -i obsidian_1.4.0_amd64.deb
```

```
# Arch Linux
yay -S obsidian
```

4.1.2.4 Linux

4.1.2.5 2. 安装 Pandoc macOS:

```
brew install pandoc
```

Windows:

- 下载 Pandoc MSI 安装程序
- 或使用 Chocolatey: `choco install pandoc`

Linux:

```
# Debian/Ubuntu
sudo apt install pandoc

# Fedora
sudo dnf install pandoc
```

验证安装:

```
pandoc --version

# 应显示 pandoc 2.19 或更高版本
```

4.1.2.6 3. 安装 LaTeX macOS:

```
# 使用 Homebrew 安装 BasicTeX (轻量级)
brew install --cask basictex

# 或安装完整的 MacTeX (推荐, 约 4GB)
brew install --cask mactex

# 安装后更新包管理器
sudo tlmgr update --self
sudo tlmgr install collection-fontsrecommended
```

Windows:

- 下载 MiKTeX 或 TeX Live
- 运行安装程序，选择完整安装
- 在安装过程中启用“自动安装缺失包”

Linux:

```
# Debian/Ubuntu
sudo apt install texlive-full

# Fedora
sudo dnf install texlive-scheme-full
```

验证安装:

```
xelatex --version

# 应显示 XeTeX 版本信息
```

4.1.2.7 4. 安装 Pandoc Crossref macOS/Linux:

```
# 方法 1: 使用包管理器
brew install pandoc-crossref # macOS

# 方法 2: 手动下载
wget https://github.com/lierdakil/pandoc-crossref/releases/download/v0.3.17.0/pandoc-crossref-macOS.tar.xz
tar -xvf pandoc-crossref-macOS.tar.xz
sudo mv pandoc-crossref /usr/local/bin/
```

Windows:

- 下载 pandoc-crossref Windows 版本
- 解压后将 pandoc-crossref.exe 放入 Pandoc 安装目录

验证安装:

```
pandoc-crossref --version
```


4.1.2.8 5. 安装 Zotero 及插件 Zotero 主程序:

- 访问 Zotero 官网 下载安装

Better BibTeX 插件:

1. 下载 Better BibTeX XPI 文件
2. 在 Zotero 中: Tools → Add-ons → 齿轮图标 → Install Add-on From File
3. 选择下载的 .xpi 文件

配置 Better BibTeX:

- 在 Zotero 设置中, Better BibTeX → Citation Keys
- Citation key formula: [auth:lower][year][shorttitle1_1]
- 例如生成: song2025collective

4.2 配置 PaperBell

4.2.1 步骤一: 获取 PaperBell 模板

```
# 克隆 PaperBell 仓库
git clone https://github.com/SongshGeo/PaperBell.git

# 在 Obsidian 中打开 PaperBell 文件夹
```

4.2.1.1 方法 1: 克隆完整仓库 (推荐)

4.2.1.2 方法 2: 手动配置 如果只想使用核心组件, 可以手动创建目录结构:

```
mkdir -p YourVault/40\ -\ Obsidian/脚
↪ 本/{longform,Pandoc/{defaults,templates,filters}}
mkdir -p YourVault/Outputs
mkdir -p YourVault/Inputs/Zotero
```

4.2.2 步骤二: 安装 Obsidian 插件

在 Obsidian 中打开设置 (Settings → Community plugins):

4.2.2.1 核心插件列表

插件名称	用途	必需性
Longform	长文本项目管理	<input type="checkbox"/> 必需
Templater	模板引擎	<input type="checkbox"/> 必需
Obsidian Enhancing Export	PDF 导出	<input type="checkbox"/> 必需
ZotLit	Zotero 集成	<input type="checkbox"/> 必需
Dataview	数据查询展示	推荐
Obsidian Git	版本控制	推荐

安装步骤:

- 1. 关闭安全模式 (Turn off Safe Mode)
- 2. 点击 Browse 搜索插件名称
- 3. 点击 Install, 然后 Enable

4.2.3 步骤三: 配置 Longform 插件

基础设置 (Settings → Longform):

```
# Projects Folder
Projects folder: Outputs

# Scenes
Scene folder name: source
Include scene titles in compile: Yes

# Compile Workflow
Workflow folder: 40 - Obsidian/脚本/longform
Compile in place: Yes

# Draft Management
Draft folder: Drafts
Auto-save sessions: Yes
```

添加自定义编译脚本:

- 1. 将 编译后增加头文件.js 放入 40 - Obsidian/脚本/longform/
- 2. 将 替换结果占位符.js 放入同一目录
- 3. 重启 Obsidian 或重新加载 Longform 插件

验证脚本加载：

- 在 Longform 面板中选择任一项目
- 点击 Compile → Configure Steps
- 应能看到 “Add YAML Metadata” 和 “Replace placeholders from JSON”

4.2.4 步骤四：配置 Pandoc 模板

PaperBell 提供了完整的 Pandoc 配置系统，包括 defaults 文件、LaTeX 模板、Lua 过滤器和 CSL 引用样式。

4.2.4.1 理解 Pandoc Defaults 文件 PaperBell 在 40 - Obsidian/脚本/pandoc/defaults/ 目录下提供了预配置的 defaults 文件：

文件名	用途	主要差异
paperbell.yaml	macOS/Linux 系统	中文字体：Songti SC、Heiti SC、STFangsong
paperbell-windows.yaml	Windows 系统	中文字体：SimSun、SimHei、FangSong
beamer.yaml	演示文稿导出	使用 Beamer 模板
crossref.yaml	交叉引用配置	pandoc-crossref 插件配置

为什么需要两个 **paperbell** 配置？
不同操作系统的中文字体名称不同：

```
# macOS/Linux (paperbell.yaml)
metadata:
  CJKmainfont: Songti SC # 宋体
  CJKsansfont: Heiti SC # 黑体
  CJKmonofont: STFangsong # 仿宋

# Windows (paperbell-windows.yaml)
metadata:
  CJKmainfont: SimSun # 宋体
  CJKsansfont: SimHei # 黑体
  CJKmonofont: FangSong # 仿宋
```

4.2.4.2 自动检测机制 Longform 的 “Add YAML Metadata” 脚本会自动检测操作系统：

```
// 自动检测逻辑
if (!template || template === "") {
  const platform = process.platform;
  if (platform === "darwin" || platform === "linux") {
    template = "paperbell"; // 使用 paperbell.yaml
  } else if (platform === "win32") {
    template = "paperbell-windows"; // 使用 paperbell-windows.yaml
  }
}
```

4.2.4.3 手动指定模板 在 Longform 编译时，可以在“Pandoc Template”文本框中手动指定：

```
# 使用 Unix 版本（即使在 Windows 上）
Pandoc Template: paperbell

# 使用 Windows 版本（即使在 macOS 上）
Pandoc Template: paperbell-windows

# 使用其他模板
Pandoc Template: eisvogel
Pandoc Template: my-custom-template
```

4.2.4.4 创建自定义 Defaults 文件 你可以创建自己的 defaults 配置文件：

1. 创建新的 **defaults** 文件：

在 40 - Obsidian/脚本/Pandoc/defaults/ 创建 my-workflow.yaml：

```
---

## General options
standalone: true
pdf-engine: xelatex
data-dir: ${.}/..

## Templates
template: ${USERDATA}/templates/my-template.latex

## Bibliography
bibliography: ${USERDATA}/../..mybib.bib
csl: ${USERDATA}/my-style.csl
```

```
## Filters
filters:
- ${USERDATA}/filters/shift_headings.lua
- pandoc-crossref
- citeproc

## Metadata
metadata:
  CJKmainfont: "Your Preferred Font"
  mainfont: "Your Preferred Font"
  numbersections: true
  link-citations: true
```

2. 在 **Longform** 中使用:

编译时在 **Template** 选项中输入 my-workflow。

```
40 - Obsidian/脚本/pandoc/
|— defaults/ # Pandoc defaults 配置文件
|   |— paperbell.yaml # Unix 系统配置
|   |— paperbell-windows.yaml # Windows 系统配置
|   |— beamer.yaml # 演示文稿配置
|   |— crossref.yaml # 交叉引用配置
|— templates/ # LaTeX 模板文件
|   |— paperbell.latex # PaperBell 学术模板
|— filters/ # Pandoc Lua 过滤器
|   |— shift_headings.lua # 调整标题层级
|   |— image.lua # 图片处理
|   |— callout.lua # Callout 渲染
|   |— ... # 其他过滤器
|— cs1/ # 引用样式文件
|   |— nature.csl
|   |— apa.csl
|   |— ...
|— preamble.sty # LaTeX 导言区自定义
```

4.2.4.5 文件结构说明

```
# 查看 defaults 文件内容
cat "40 - Obsidian/脚本/pandoc/defaults/paperbell.yaml"

# 测试 defaults 配置
cd "40 - Obsidian/脚本/pandoc"
pandoc --defaults=paperbell.yaml test.md -o test.pdf

# 查看可用的过滤器
ls "40 - Obsidian/脚本/pandoc/filters/"
```

4.2.4.6 验证配置

4.2.5 步骤五：配置 Zotero 导出

配置 **Better BibTeX** 自动导出：

1. 在 Zotero 中选择你的文献库
2. 右键 → Export Library
3. 格式选择：Better BibTeX
4. 勾选 “Keep updated”
5. 导出位置：YourVault/40 - Obsidian/mybib.bib

配置 **ZotLit** 插件（Obsidian）：

Settings → ZotLit:

```
# Zotero Connection
Zotero Path: /Users/yourname/Zotero # Zotero 数据目录

# Note Templates
Literature Note Template: 40 - Obsidian/模板/Literature Note.md
Note Location: Inputs/Zotero

# Annotation Settings
Color Mapping:
- Yellow (#ffd400): Highlight
- Green (#5fb236): Method
- Blue (#2ea8e5): Theory
- Red (#ff6666): Question

# Citation Key
Use Better BibTeX Key: Yes
```

Zotlit 文献笔记模板:

在 Zotlit 插件的 `template` 界面或者在 40 - Obsidian/模板文件夹找到下面 md 文件进行修改:

zt-annot.eta:

```
<%
// 从 paperbell 插件获取配置
let label = {};
let noteLabel = 'Note'; // 默认标签
try {
  const paperbellPlugin = app.plugins.plugins.paperbell;
  if (paperbellPlugin && paperbellPlugin.settings &&
    paperbellPlugin.settings.ZotLitColors &&
    paperbellPlugin.settings.ZotLitColors.mapping) {
    const colorMapping = paperbellPlugin.settings.ZotLitColors.mapping;
    // 将 mapping 对象转换成我们需要的格式 {colorName: label}
    Object.keys(colorMapping).forEach(color => {
      if (colorMapping[color] && colorMapping[color].callout) {
        label[color] = colorMapping[color].callout;
      }
    });
  }
} catch (e) {
  console.error(" 无法读取 paperbell 插件配置:", e);
}
// 如果读取失败, 使用默认映射作为备份
if (Object.keys(label).length === 0) {
  label = {
    "red": "Conclusion",
    "orange": "Question",
    "yellow": "Highlight",
    "gray": "Comment",
    "green": "Quote",
    "cyan": "Task",
    "blue": "Definition",
    "navy": "Definition",
    "purple": "Question",
    "brown": "Source",
    "magenta": "To Do"
  };
}
// 获取当前注释的标签
noteLabel = label[it.colorName] ? label[it.colorName] : 'Note';
%>
```

```
[!<%= noteLabel %>] Page <%= it.pageLabel %>

<%= it.imgEmbed %><%= it.text %>
<% if (it.comment) { %>
---

<%= it.comment %>
<% } %>
```

zt-annots.eta

```
<%
// 尝试从 paperbell 插件获取配置
let label = {};
let readSuccess = false; // 添加一个标志来跟踪是否成功读取数据
let readSource = " 未知"; // 记录数据来源
try {
  // 假设可以通过 app.plugins.plugins 访问插件
  const paperbellPlugin = app.plugins.plugins.paperbell;
  if (paperbellPlugin && paperbellPlugin.settings &&
    paperbellPlugin.settings.ZotLitColors &&
    paperbellPlugin.settings.ZotLitColors.mapping) {
    // 从插件设置中提取颜色标签映射
    const colorMapping = paperbellPlugin.settings.ZotLitColors.mapping;
    // 将 mapping 对象转换成我们需要的格式 {colorName: label}
    Object.keys(colorMapping).forEach(color => {
      if (colorMapping[color] && colorMapping[color].label) {
        label[color] = colorMapping[color].label;
      }
    });
    // 如果至少读取到一个颜色标签，则标记为成功
    if (Object.keys(label).length > 0) {
      readSuccess = true;
      readSource = "paperbell 插件";
    }
  }
} catch (e) {
  console.error(" 无法读取 paperbell 插件配置:", e);
}
console.log("paperbell 插件检测:", !!app.plugins.plugins.paperbell);
console.log("paperbell 设置检测:", !(app.plugins.plugins.paperbell &&
  ↪ app.plugins.plugins.paperbell.settings));
console.log("ZotLitColors 检测:", !(app.plugins.plugins.paperbell &&
  app.plugins.plugins.paperbell.settings &&
```



```

    app.plugins.plugins.paperbell.settings.ZotLitColors));
// 如果读取失败，使用默认映射作为备份
if (Object.keys(label).length === 0) {
    label = {
        "red": "Conclusion",
        "orange": "Keyword",
        "yellow": "Highlight",
        "gray": "Comment",
        "green": "Quote",
        "cyan": "Task",
        "blue": "Definition",
        "navy": "Definition",
        "purple": "Question",
        "brown": "Source",
        "magenta": "To Do"
    };
    readSource = " 默认映射";
    console.log(" 使用默认映射进行笔记提取，请检查 PaperBell 插件配置。")
}
const byColor = Object.groupBy(it, (annot) => annot.colorName);
// 保持原来的逻辑
const colorSet = new Set([...Object.keys(label), ...Object.keys(byColor)]);
%>
<% for (const color of colorSet) {
    if (!(color in byColor)) continue
-%>

### <%= label[color] ?? color %>

<%=_for (const annot of byColor[color]) { %>
<%=~ include("annotation", annot) %>
<%%>
<%=_ } %>
<%= } %>

```

zt-cite.eta

```
[<%= it.map(lit => `@${lit.citekey}`).join("; ") %>]
```

zt-cite2.eta

```
<%= it.map(lit => `@${lit.citekey}`).join("; ") %>
```

zt-colored.eta

```
<mark style="
<%- if (it.color) { _%> color: <%= it.color %>; <%-_ } -%>
<%- if (it.bgColor) { _%> background-color: <%= it.bgColor %>; <%-_ } -%>
"><%= it.content %></mark>
```

zt-field.eta

```
title: "<%= it.title %>"

citekey: "<%= it.citekey %>"

tags: [paper, <% = it.tags.filter(t => t.name &&
  ↳ t.name.startsWith('#')).map(t => '"' + t.name.slice(1) + '"').join(', ')
  ↳ %>]

cate: 论文

concepts: [<%let excludeEndings = ['更新', '推荐', '关联', '检索', '浏览', '初
  ↳ 读', '精读', '星标'];
let filteredConceptTags = (Array.isArray(it.tags) ? it.tags : []).filter(t =>
  t.name &&
  !t.name.startsWith('#') &&
  !excludeEndings.some(ending => t.name.endsWith(ending))
).map(t => '"' + t.name + '"');
%> <%= filteredConceptTags.join(', ') %>]

read: [<% let endings = ['浏览', '初读', '精读']; let filteredTags =
  ↳ it.tags.filter(t => t.name && endings.some(ending =>
  ↳ t.name.endsWith(ending))); if (filteredTags.length === 1) { %> "<%=
  ↳ filteredTags[0].name %>" <% } else if (filteredTags.length > 1) { %> 错误:
  ↳ 存在多个符合条件的标签。 <% } else { %> 错误: 没有找到符合条件的标签。 <% } %>]

source: [<% let endings_2 = ['更新', '推荐', '关联', '检索']; let filteredTags_2
  ↳ = it.tags.filter(t => t.name && endings_2.some(ending =>
  ↳ t.name.endsWith(ending))); if (filteredTags_2.length === 1) { %> "<%=
  ↳ filteredTags_2[0].name %>" <% } else if (filteredTags_2.length > 1) { %>
  ↳ 错误: 存在多个符合条件的标签。 <% } else { %> 错误: 没有找到符合条件的标签。 <% } %>]

authors: [<%= it.authors %>]
```

```

journal: <%= it.publicationTitle %>

paper_date: <%= it.date %>

date: <%= (new Date(it.dateModified || Date.now())).toISOString().slice(0,
  ↪ 10) %>

<%
let isImportant = it.tags.some(t => t.name === '🌟 星标');
%>

important: <%= isImportant ? 'True' : 'False' %>

```

zt-note.eta

```

| Zotero | File | Journal |
| ----- | ----- |
↪ ----- |
| Zotero | <%= it.fileLink %> | <%= it.publicationTitle %> |

## Annotations

<%~ include("annots", it.annotations) %>

```

4.2.6 步骤六: 配置 Enhancing Export

Settings → Obsidian Enhancing Export:

添加 **PaperBell Academic** 导出配置:

```

{
  "name": "PaperBell Academic",
  "type": "pandoc",
  "arguments": "-f ${fromFormat} --resource-path=\"${currentDir}\"
  ↪ --resource-path=\"${currentDir}/imgs\"
  ↪ --resource-path=\"${currentDir}/figs\"
  ↪ --resource-path=\"${attachmentFolderPath}\" --pdf-engine=xelatex
  ↪ --template=paperbell -o \"${outputPath}\" -t pdf",
  "customArguments": "--bibliography=\"${vaultDir}/40 - Obsidian/mybib.bib\"
  ↪ --filter pandoc-crossref --number-sections --citeproc -M
  ↪ reference-section-title=References --listings",
  "extension": ".pdf",

```

```
"outputPath": "${currentDir}/${fileName}.pdf"
}
```

添加带行号版本:

```
{
  "name": "PaperBell Academic (Line Numbers)",
  "type": "pandoc",
  "arguments": "-f ${fromFormat} --resource-path=\"${currentDir}\"
  ↪ --resource-path=\"${currentDir}/imgs\" --pdf-engine=xelatex
  ↪ --template=paperbell -M lineno=true -o \"${outputPath}\" -t pdf",
  "customArguments": "--bibliography=\"${vaultDir}/40 - Obsidian/mybib.bib\"
  ↪ --filter pandoc-crossref --number-sections --citeproc -M
  ↪ reference-section-title=References",
  "extension": ".pdf",
  "outputPath": "${currentDir}/${fileName}_lineno.pdf"
}
```

4.3 创建第一个项目

4.3.1 步骤一: 创建 **Longform** 项目

1. 打开 Obsidian 左侧边栏的 Longform 图标
2. 点击 **New Project**
3. 配置项目:
 - Name: MyFirstPaper
 - Type: Manuscript
 - Location: Outputs/
4. 点击 **Create**

4.3.2 步骤二: 添加项目结构

手动创建以下文件和文件夹:

```
cd Outputs/MyFirstPaper
mkdir imgs
touch results.json
```

创建章节文件:

```
cd source
touch Introduction.md Methods.md Results.md Discussion.md
```

4.3.3 步骤三：配置元数据

编辑 Index.md:

```
---
title: "My First Academic Paper with PaperBell"
date: "2025-10-11"

authors:
  - name: "Your Name"
    affiliation: [1]
    corresponding: "your.email@institution.edu"

affiliations:
  - index: 1
    name: "Your Institution"

abstract: "This is my first paper using PaperBell workflow."
keywords:
  - "academic writing"
  - "Obsidian"
  - "automation"

target: "Test Journal"
acronym: "MyPaper"
csl: "nature"

lineno: "false"
figures-at-end: "false"
titlepage: true
---
```

4.3.4 步骤四：编写示例内容

source/Introduction.md:

```
# Introduction

This is a demonstration of the PaperBell workflow.
```

```
We analyzed {{demo.n_samples}} samples and found a correlation of
↪ {{demo.correlation}}.
```

```
Previous studies have shown [example2024paper] that this approach is
↪ effective.
```

```
![Example figure](imgs/demo.png){#fig:demo width=80%}
```

```
As shown in [fig:demo], our method works well.
```

```
results.json:
```

```
{
  "demo": {
    "n_samples": 100,
    "correlation": 0.75,
    "pvalue": 0.001
  }
}
```

4.3.5 步骤五：添加引用和图片

添加测试引用：

在 40 - Obsidian/mybib.bib 中添加：

```
@article{example2024paper,
  title={An Example Paper},
  author={Author, A. and Coauthor, B.},
  journal={Example Journal},
  volume={1},
  pages={1--10},
  year={2024},
  publisher={Example Publisher}
}
```

添加测试图片：

将任意图片复制到 imgs/demo.png。

4.3.6 步骤六：编译项目

1. 在 Longform 面板中选择 MyFirstPaper

2. 点击 **Compile** 选项卡
3. 配置编译步骤（按顺序）：
 - Strip Frontmatter
 - Remove Links
 - Prepend Title
 - Concatenate Text
 - **Add YAML Metadata** (Note Name: Index)
 - **Replace placeholders from JSON**
 - Save as Note (输出: manuscript.md)

4. 点击 **Compile**

验证编译结果：

打开 manuscript.md，检查：

- ☐ YAML 前置内容包含完整元数据
- ☐ 占位符 `{{demo.n_samples}}` 已替换为 100
- ☐ 章节按顺序合并

4.3.7 步骤七：导出 PDF

4.3.7.1 方法 1：使用 Enhancing Export

1. 右键点击 manuscript.md
2. 选择 “Export with Enhancing Export”
3. 选择 “PaperBell Academic”
4. 等待编译完成（首次编译可能需要 1-2 分钟）

```
cd Outputs/MyFirstPaper
pandoc manuscript.md -o output.pdf \
  --defaults="../../40 - Obsidian/脚本/Pandoc/defaults/pdf.yaml"
```

4.3.7.2 方法 2：使用命令行 检查 PDF 输出：

打开 output.pdf，验证：

- ☐ 标题页显示正确
- ☐ 作者和机构信息完整
- ☐ 参考文献正确格式化
- ☐ 图片正常显示
- ☐ 图表编号和引用正确

4.4 故障排除

4.4.1 问题 1: 占位符未替换

症状: 编译后的 `manuscript.md` 中仍有 `{{placeholder}}`

可能原因:

1. `results.json` 不在项目根目录或 `source/` 目录
2. JSON 语法错误
3. 路径拼写错误

解决方案:

```
# 1. 检查文件位置
ls Outputs/MyFirstPaper/results.json
ls Outputs/MyFirstPaper/source/results.json

# 2. 验证 JSON 语法
cat results.json | python -m json.tool

# 3. 启用调试模式

# 在 Longform 编译选项中勾选 "Enable debug log"

# 查看 Obsidian 控制台输出 (Cmd+Option+I)
```

4.4.2 问题 2: Pandoc 编译失败

症状: 导出 PDF 时报错

常见错误及解决方案:

```
# 验证 LaTeX 安装
which xelatex

# macOS: 添加到 PATH
echo 'export PATH="/Library/TeX/texbin:$PATH"' >> ~/.zshrc
source ~/.zshrc

# Windows: 在系统环境变量中添加 MiKTeX bin 目录
```

4.4.2.1 错误: `xelatex not found`


```
# 检查模板路径
ls "40 - Obsidian/脚本/Pandoc/templates/paperbell.latex"

# 使用绝对路径
pandoc manuscript.md -o output.pdf \
  --template="/full/path/to/paperbell.latex"
```

4.4.2.2 错误: Template paperbell.latex not found

```
# 安装缺失的 LaTeX 包
sudo tlmgr install tikz # macOS/Linux
mpm --install=tikz # Windows MiKTeX
```

4.4.2.3 错误: File 'tikz.sty' not found

```
# 验证安装
which pandoc-crossref

# 重新安装
brew reinstall pandoc-crossref # macOS
```

4.4.2.4 错误: pandoc-crossref not found

4.4.3 问题 3: 字体问题

症状: PDF 中文显示为方块或编译报错

解决方案:

```
# 检查系统字体
fc-list | grep "Times New Roman"
fc-list | grep "Songti" # 中文字体

# macOS: 安装中文字体
```

```
# 系统字体册 → 全部字体 → 确认有 " 宋体 "
```

```
# Linux: 安装中文字体
```

```
sudo apt install fonts-noto-cjk # Debian/Ubuntu
```

```
# 修改 pdf.yaml 使用系统可用字体
```

```
metadata:
```

```
  CJKmainfont: "Noto Serif CJK SC" # 替代宋体
```

```
  mainfont: "Liberation Serif" # 替代 Times New Roman
```

4.4.4 问题 4: 引用无法解析

症状: PDF 中显示 [unknown] 而非正确引用

诊断步骤:

```
# 1. 检查 .bib 文件路径
```

```
ls "40 - Obsidian/mybib.bib"
```

```
# 2. 验证引用键存在
```

```
grep "@article{unknown}" "40 - Obsidian/mybib.bib"
```

```
# 3. 测试 citeproc
```

```
pandoc test.md -o test.pdf \
```

```
  --bibliography="40 - Obsidian/mybib.bib" \
```

```
  --citeproc
```

常见问题:

- 引用键拼写错误 (区分大小写)
- .bib 文件编码问题 (应为 UTF-8)
- 引用键格式不符合 Better BibTeX 规则

4.4.5 问题 5: 行号显示异常

症状: 行号未显示或参考文献也有行号

解决方案:

```
# 确保使用最新的 paperbell.latex 模板
```

检查模板中是否有以下代码:

% 在参考文献前关闭行号

```
\AtBeginEnvironment{thebibliography}{  
  \linenomathNonumbers  
  \nolinenumbers  
}
```

如果模板过旧, 从 PaperBell 仓库下载最新版本。

4.4.6 问题 6: 图表位置错误

4.4.6.1 症状 启用 `figures-at-end: true` 但图表未移到文末

4.4.6.2 检查项

1. YAML 元数据中是否正确设置:

```
figures-at-end: "true" # 注意是字符串格式
```

2. 模板是否支持该功能 (检查 `paperbell.latex` 版本)
3. 图表标记是否正确:

```
![Caption] (image.png) {#fig:label} # ☐ 正确  
![Caption] (image.png) # ☐ 不会被识别
```

4.4.7 问题 7: Longform 脚本未加载

症状: 编译步骤中看不到自定义脚本

解决方案:

1. 检查脚本位置

```
ls "40 - Obsidian/脚本/longform/"
```

2. 检查脚本格式

确保文件名为 `.js` 且导出 `module.exports`

3. 验证脚本语法

```
node --check "40 - Obsidian/脚本/longform/编译后增加头文件.js"
```

```
# 4. 重启 Obsidian
```

```
# 或重新加载 Longform 插件 (Settings → Community plugins → Reload)
```

4.4.8 问题 8: 编译速度慢

优化方案:

1. 缓存图片:

```
# pdf.yaml
resource-path:
- .
- imgs
```

2. 使用增量编译:

```
# 仅在内容变化时重新编译
make -f Makefile
```

3. 减少过滤器:

```
# 调试时暂时禁用部分过滤器
filters:
# - some-slow-filter
- pandoc-crossref
- citeproc
```

4. 使用 **LuaLaTeX** 替代 **XeLaTeX** (如不需要复杂字体):

```
pdf-engine: lualatex
```

4.4.9 调试工具和日志

Obsidian 控制台:

```
// 在 Obsidian 中按 Cmd+Option+I (Mac) 或 Ctrl+Shift+I (Windows)
// 查看 Longform 脚本的 console.log 输出
```

Pandoc 详细日志:

```
pandoc manuscript.md -o output.pdf \
  --defaults=pdf.yaml \
  --verbose 2>&1 | tee pandoc.log
```

LaTeX 日志:

```
# 编译失败时查看详细日志
cat output.log | grep "Error"
```

测试最小示例:

创建 test.md:

```
---
title: "Test"
---

# Test

This is a test [example2024].
```

逐步测试:

```
# 1. 测试 Markdown 到 PDF
pandoc test.md -o test.pdf

# 2. 测试引用
pandoc test.md -o test.pdf --bibliography=mybib.bib --citeproc

# 3. 测试模板
pandoc test.md -o test.pdf --template=paperbell.latex
```

4.5 更新和维护

4.5.1 更新 PaperBell 模板

```
# 进入 PaperBell 目录
cd /path/to/PaperBell

# 拉取最新更新
git pull origin main

# 复制更新的模板文件
cp templates/paperbell.latex YourVault/40\ -\ Obsidian/脚本/Pandoc/templates/
```

4.5.2 更新插件

在 Obsidian 中:

1. Settings → Community plugins
2. 点击 “Check for updates”
3. 更新所有可用更新

4.5.3 备份策略

每日自动备份（使用 Obsidian Git）:

Settings → Obsidian Git:

```
Automatic pull/push/commit:
- Auto pull: Every 10 minutes
- Auto commit: Every 30 minutes
- Auto push: Every 60 minutes
```

手动备份:

```
# 备份整个 Vault
tar -czf paperbell_backup_$(date +%Y%m%d).tar.gz YourVault/

# 仅备份项目文件
tar -czf projects_backup_$(date +%Y%m%d).tar.gz YourVault/Outputs/
```

4.6 小结

本章详细介绍了 PaperBell 的完整安装配置流程，从系统准备、软件安装、插件配置到创建第一个项目。我们提供了常见问题的系统化诊断和解决方案，以及更新维护建议。

通过遵循本章的步骤，研究者可以在 1-2 小时内完成 PaperBell 的完整配置，开始享受现代化学术写作的便利。

相关资源：

- PaperBell GitHub
- Pandoc 文档
- Longform 插件
- 问题反馈



Figure 3 – 请关注我们 PaperBell 的官方账号

References

1. 宋爽. *et al.* 社会—生态系统适应性治理研究进展与展望. 地理学报 **74**, 2401–2410 (2019).