

目录

1 前言	2
2 第一次完整转换	3
2.1 安装与最短用法	3
2.2 manifest 的作用	3
2.3 如何阅读诊断	3
2.4 退出码	4
2.5 格式识别	4
3 一个共享的文档表示	5
4 Reader、Writer 与 Bundle	6
5 Office 与其他容器格式	7
6 唯一的 Writer	8
7 把输入视为恶意数据	9
8 嵌入与 Filter	10
8.1 Zig API	10
8.2 Python API	10
8.3 浏览器 API	10
9 CLI 参考	11
10 性能测试	12
11 Server	13
12 进一步阅读	14

1 前言

文档承载着作者与读者之间的约定，而文件格式只是历史形成的容器。zenfmt 是一个用 Zig 编写的文档转换器。它读取 19 种输入格式并输出 GitHub Flavored Markdown。项目规模不大，也不打算声称能够覆盖 Pandoc 的全部用途。它更关注紧凑的引擎、明确的资源限制、可复现的输出，以及把转换过程中丢失的信息如实告诉使用者。

本书面向需要使用 zenfmt 的读者。它与英文版采用相同的章节顺序，介绍 CLI、文档表示、格式 reader、Markdown writer、资源限制、Zig 与 Python API、浏览器版本、server 和 benchmark。代码中的名称、命令行选项、report code、API 字段和文件格式使用原文，因为这些名称需要能够直接复制和搜索。

设计决策保存在 docs/zds/ 中的 Zen Discussion，也就是 ZDS 记录。ZDS 是项目的设计历史，因此仍然只提供英文版。本书会说明如何使用已经实现的行为；需要了解某项设计为什么这样选择时，可以继续阅读相应的 ZDS。

zenfmt 有三项贯穿始终的约定。

- 信息损失会被报告，而不是隐藏。转换产生的诊断会写入终端，也会进入相邻的 manifest。
- core 不包含任何具体格式的知识。每种格式由独立 reader 或 writer plugin 提供。
- 所有文档都按不可信输入处理。archive、stream、递归深度、输出大小和内存都有明确上限。

2 第一次完整转换

2.1 安装与最短用法

从 GitHub Releases 下载适合当前平台的原生压缩包，解压后即可运行。这个可执行文件同时包含 CLI 和 server，不需要 Java、Python、npm、OCR、VLM 或模型文件。Python 用户也可以安装 `zenfmt` wheel。macOS 用户也可以使用 repository 中的 Homebrew cask。它会直接从 GitHub Releases 下载适合 Apple Silicon 或 Intel 的同一个自包含压缩包。

```
brew install --cask \
  https://raw.githubusercontent.com/insanai/zenfmt/main/packaging/homebrew/Casks/zenfmt.
rb
```

```
zenfmt report.docx
```

默认情况下，`zenfmt` 在输入文件旁创建 `report.md` 和 `report.md.zenfmt.json`。第一个文件是 Markdown artifact，第二个是 canonical JSON manifest。希望只把文档内容写到标准输出时使用：

```
zenfmt report.docx --stdout
```

显式指定输出路径时，artifact、manifest 和媒体资源会以 transaction 方式提交。转换失败或被严格模式拒绝时，不会留下只写了一部分的目标文件。

```
zenfmt report.docx --output build/report.md
```

查看当前二进制包含的 reader 与 writer：

```
zenfmt --list-formats
```

2.2 manifest 的作用

manifest 是转换过程的 custody record。它包含 source 与 artifact 的名称、识别出的格式、plugin id 和 BLAKE3-256 digest，也包含 AST schema、文档 metadata、diagnostic reports、plugin preservation data 和 facet 摘要。相同输入、相同版本和相同选项会生成相同字节。

当后续转换看到相邻 manifest 时，会先核对 digest。匹配时可以继续携带 plugin data；不匹配时会报告 manifest 已过期并忽略它，不会把旧 metadata 当成可信信息。

2.3 如何阅读诊断

诊断采用 Elm 风格的结构，依次回答四个问题。

- 问题是什么。
- 问题出现在哪里。
- `zenfmt` 因此采取了什么行动。
- 使用者下一步可以怎么做。

每条报告还有稳定的 code，例如 `docx.merged-cells-degraded`。自动化程序应根据 code 判断，而不要匹配可能改进的自然语言。报告中的 loss tier 区分简化后仍保留内容的 `degraded` 与内容已不存在的 `dropped`。

2.4 退出码

退出码	含义
0	artifact 与 manifest 已提交。可能仍有 warning 或信息损失。
1	转换失败，例如损坏的容器、无效 XML 或 reader 发现矛盾。
2	命令用法错误，例如格式名未知、参数缺失或拒绝覆盖。
3	输入超过明确的资源上限。

2.5 格式识别

扩展名只是第一条提示。zenfmt 还检查 ZIP central directory 中的 part name、OpenDocument 与 EPUB 的 `mimetype`、CFB directory stream、PDF header 和 RTF signature。扩展名与内容冲突时，以内容为准并报告冲突。加密文档会被明确拒绝，不会静默跳过。

3 一个共享的文档表示

所有 reader 都生成同一种 document IR，所有 writer 都从它读取。IR v2 分为几个层次。

- block 与 inline AST 表示段落、heading、list、table、link、image 和文本等可见结构。
- attributes 保存 identifier、class 和 key value 属性。
- resource store 保存嵌入图片等按 digest 标识的二进制资源。
- facets 以稀疏的 stand off annotation 保存 style、修订、页与 slide geometry、formula 和 provenance。
- plugin preservation data 保存只对某个格式有意义、但未来 round trip 可能需要的信息。

AST 使用 preorder 的 struct of arrays 和 u32 index，而不是为每个 node 分配对象。一个 subtree 是连续区间，因此遍历、复制和限制检查较直接。不可变 transform 通过重建结果来工作；未改变的连续 subtree 可以按数组范围复制。

writer 不会因为 IR 中存在某项信息就假装自己能表达它。每个 writer 声明 capabilities，lowering planner 根据规则决定保留、简化或丢弃什么，并为每个损失生成 report。这样 strict mode 可以在写文件前做出一致决定。

4 Reader、Writer 与 Bundle

core 只协调 conversion pipeline，不识别 DOCX、PDF 或 Markdown 名称。format plugin 提供 descriptor、reader 或 writer、capabilities、limits 和 report catalog。默认 zenfmt bundle 在 compile time 把随项目发布的 plugin 组合起来。

Reader 的责任不只是解析成功。它还必须：

- 在读取前和扩展容器时执行 resource limits。
- 为不支持或降级的结构发出可操作 report。
- 生成结构合法且 index 有效的 document IR。
- 把媒体资源交给 resource store，而不是写入任意路径。
- 只在 namespace 中保存自己的 preservation data。

应用可以使用更小的 bundle，只编译需要的格式。Zig 项目也可以声明自己的 filter。filter 实现 visitBlock 或 visitInline，通过项目自己的 build.zig 编译进 pipeline，不需要嵌入 scripting runtime 或跨进程序列化。

5 Office 与其他容器格式

许多看起来不同的格式最终都是受约束的容器。

- DOCX、XLSX、PPTX、ODT、ODS、ODP 和 EPUB 通常是 ZIP package。
- 旧版 DOC、XLS 和 PPT 使用 CFB compound file。
- RTF 是带 group nesting 的 token stream。
- PDF 是 indirect object、xref、content stream 与 font mapping 的组合。

ZIP reader 会限制 entry 数量、单个展开大小、总展开大小和 compression ratio。它只按规范需要的 part name 读取内容，不会把 archive 解压到磁盘。CFB reader 会验证 sector chain、directory tree 和 stream boundary，并检测循环。XML parser 是 pull parser，限制 nesting、attribute 和 text 大小，也不会解析外部 entity。

Spreadsheet 的 formula、cell style 和 merged range 可能无法在 Markdown 中完整表达。Presentation 的 position、speaker note 和 animation 也类似。reader 会尽量保留可见文字和表格结构，并用 facet 或 report 说明其余信息。PDF reader 只做原生 Zig 文本提取，不运行 OCR；扫描图片本身没有可提取文字时会诚实返回相应限制。

6 唯一的 Writer

当前发布版只写 Markdown。较窄的输出面让许多 reader 共享同一个可检查的目标，也让 loss policy 更清楚。

Markdown writer 输出 CommonMark 加 GFM table。它统一空行、换行和 escaping，文件末尾只有一个 newline，不保留 trailing spaces。相同 IR 会产生相同字节。embedded resource 使用稳定名称发布到相邻媒体目录，并在 manifest 中记录。

无法直接表达的结构按规则 lowering。例如 merged cell 的内容保留在第一个 cell，其余 covered position 为空；page break 通常被丢弃；复杂 style 不会伪装成视觉上等价的 Markdown。每个决定都成为 report。

严格模式控制可接受的损失：

```
zenfmt report.docx --strict=content
zenfmt report.docx --strict=structure
zenfmt report.docx --strict=exact
```

`content` 拒绝内容丢失，`structure` 也拒绝结构损失，`exact` 拒绝任何已知损失。默认 `off` 会完成转换并报告损失。strict refusal 在 publication 前发生。

7 把输入视为恶意数据

zenfmt 不根据一台开发机能够承受多少来猜测安全边界。每项限制都有名称、默认值、report code 和建议。主要边界包括输入字节数、archive entry 数、展开字节数、compression ratio、XML depth、AST node 数、text bytes、resource bytes、输出字节数和 conversion 时间。

命令行可在允许的范围内调整部分限制。server 还会在 request body、concurrency、rate limit 和 timeout 层增加边界。达到限制属于正常拒绝，退出码为 3，不是 crash。

浏览器 profile 更保守，因为 tab 的可用内存与执行时间更有限。WebAssembly module 没有 filesystem、network、clock 或随机数 host import。转换在 worker 中运行，主页面保持响应；cancel 和 timeout 会终止当前工作，而不是让页面无限等待。

处理来源不明的文件时，建议保留默认 limits，检查 manifest 与 reports，并在需要保证时选择合适 strict level。不要因为某次输入可以成功转换，就在外层 service 去掉 size 或 time limit。

8 嵌入与 Filter

8.1 Zig API

应用可调用与 CLI 相同的 engine:

```
const zenfmt = @import("zenfmt");

const result = try zenfmt.convert(gpa, io, .{
    .input = .{ .path = "report.docx" },
    .output = .{ .memory = {} },
});
defer result.deinit(gpa);
```

实际类型和选项以当前 package API 为准。调用者提供 allocator 与 I/O 能力，转换结果包含 artifact、manifest、resources 和 reports。希望缩小 binary 时，可从 `zenfmt_core.Bundle` 组合较少的 plugin。Filter 属于应用自己的编译产物。它接收 document node 并返回重建后的内容。不要在遍历中保存失效 index，也不要绕过 builder 直接制造不合法关系。filter 产生的损失也应使用统一 report 机制表达。

8.2 Python API

Python distribution 是 typed、dependency free 的 wheel，并带有对应平台的 native bridge。转换期间会释放 GIL。

```
import zenfmt

conversion = zenfmt.convert("report.docx")
print(conversion.text)
for report in conversion.reports:
    print(report.code, report.problem)
```

`str` 始终表示 path。内存中的内容应显式传 `bytes` 并提供文件名：

```
conversion = zenfmt.convert(
    uploaded_bytes,
    name="upload.docx",
    to="markdown",
)
```

指定 `output` 会使用 transaction publication，并在目标旁创建 manifest 与媒体资源。`zenfmt.Converter` 是不可变的可复用 policy value，不读取全局配置、环境变量或网络。失败会抛出紧凑的 exception hierarchy，例如 `ConversionError`、`LimitExceededError` 和 `UnknownFormatError`，消息与 CLI 一样给出问题、后果与下一步。

8.3 浏览器 API

release 中的 `wasm32-freestanding` bundle 包含 module、ES module adapter、worker 和 TypeScript declaration。adapter 返回 artifact ensemble，不会回退到远程 server。调用方应从静态 origin 以正确的 `application/wasm` type 提供文件，并设置自己的 UI timeout 与 size policy。

Web application 也可以通过 `npm install @insanai/zenfmt` 获取相同的 dependency free browser distribution。npm package 包含经过检查的 module、adapter、worker、declarations 和 capability contract。原生 CLI 与 server 仍然不需要 Node 或 npm。

9 CLI 参考

常用命令：

```
zenfmt INPUT
zenfmt INPUT --output OUTPUT
zenfmt INPUT --stdout
zenfmt INPUT --from docx --to markdown
zenfmt INPUT --strict=structure
zenfmt INPUT --preserve-facets
zenfmt --list-formats
zenfmt --help
zenfmt serve --help
```

从 stdin 读取时通常需要 `--from`，因为没有文件名可以提供初始提示。向 stdout 写 artifact 时，diagnostic 仍写 stderr，便于 shell pipeline 分开处理。除非明确允许，否则 zenfmt 不覆盖已有输出。支持的输入 family：

类别	格式
文字处理	docx、docm、doc、odt、rtf
电子表格	xlsx、xlsm、xlsb、xls、ods、csv、tsv
演示文稿	pptx、pptm、ppsx、ppsm、ppt、pps、pot、odp
出版与页面	epub、pdf、html
Markup	markdown、asciidoc、rst、text

详细 limit 默认值和所有 diagnostic code 以英文 reference chapter、`--help` 与 machine-readable capabilities 为准。程序应依赖稳定 code 和 schema version，而不是自然语言文本。

10 性能测试

benchmark 先问 correctness，再问速度。语料库 manifest 记录每个第三方文件的来源、格式、大小和 SHA-256，但不重新分发许可不同的原文件。`benchmarks/fetch_corpus.sh` 下载相同字节并验证 digest。测试把不同问题分开呈现。

- Native CLI 比较 zenfmt、Docling parser only、AnyDoc 与 Pandoc 的 coverage、wall time、CPU time 和 peak RSS。
- Browser 测试 WebAssembly 的下载大小、cold ready、warm conversion 和与 native artifact 的 parity。
- Server 测试长期运行的 zenfmt 与 Tika Server 的 startup、warm latency、peak RSS 和不同 concurrency 下的 throughput。
- Output preservation 按格式使用 tool neutral oracle，不把多项质量判断压缩成一个分数。

Docling 测试明确关闭 OCR、VLM、ASR、layout model、table model、enrichment 和 accelerator。这样它能在与 zenfmt 类似的普通机器上运行，比较的是 parser path，不是 AI pipeline。unsupported file 会显示为 unsupported，不会当成无限慢，也不会切换到 AI。

ratio 使用 comparison tool 除以 zenfmt。大于 1.0 表示本次记录中 comparison tool 在该指标上使用更多。ratio 只在双方都成功的 shared files 上计算 geometric mean。不同 release、不同机器或不同 benchmark lens 的数字不会合并。结果提供背景，不代表对所有文档的质量排名。

11 Server

同一个 `zenfmt` 可执行文件包含 server、OpenAPI 文档、database migration 和 web assets。

```
zenfmt serve
curl -s -T report.docx \
    "http://127.0.0.1:8998/api/v1/convert?to=markdown"
```

open mode 默认只监听 `127.0.0.1:8998`，无账户且保持 stateless，适合本机工具和受控 sidecar。访问 `/docs` 可查看 OpenAPI reference，`/openapi.json` 提供 OpenAPI 3.1 contract。未授权访问 web root 时会进入公开 API 文档，而不是显示不可用的管理页面。

需要共享部署时启用 secure mode：

```
zenfmt serve --secure --data-dir ./zenfmt-data
```

secure mode 增加 user、role、API key、session、audit log 和 administration UI。operator 明确选择 data directory，程序不会在其他位置偷偷创建状态。反向代理应负责 TLS，并限制可到达的 bind address。API key 只应显示一次，日志不应记录 credential 或文档正文。

转换 endpoint 支持普通 response 与 streaming。客户端应设置 request size、timeout 和重试策略，并根据 HTTP status 与 structured error envelope 判断失败。health endpoint 用于存活检查，metrics 用于运行观察，不应把包含敏感 filename 的高 cardinality value 作为 label。

12 进一步阅读

本书帮助使用 zenfmt 完成转换与部署。英文完整版包含更深入的 IR layout、各格式 container、diagnostic catalog、limit table 和 benchmark raw tables。设计选择及其未采用方案保存在英文 ZDS 中。

- 项目主页与浏览器转换器: <https://insanai.github.io/zenfmt/>
- release 与自包含程序包: <https://github.com/insanai/zenfmt/releases>
- source 与 issue: <https://github.com/insanai/zenfmt>
- API reference: 运行 `zenfmt serve` 后访问 `/docs`

如果转换结果对业务很重要, 请保留 source、artifact 和 manifest, 检查 reports, 并用适合需求的 strict level。zenfmt 希望把能够确定的事情做得清楚, 也把不能完整保留的部分说清楚。