

我开发了一个 MCP  
文档转换工具，让 AI  
轻松搞定格式转换

我开发了一个 MCP  
文档转换工具，让 AI  
轻松搞定格式转换

大家好，我是玄同765（xt765）。作为一名开发者，我一直想让 AI 更好地帮助我们处理日常工作。最近我遇到了一个痛点：如何让 AI 直接处理不同格式的文档？

比如，我想让 AI 读取一个 PDF 文件并总结内容，或者把 Markdown 转成 Word 文档发给同事。但 AI 往往只能输出文本，不能直接操作文件格式。这让我萌生了一个想法：开发一个能让 AI 直接调用的文档转换工具。

于是，MCP Document Converter诞生了。

## 为什么选择 MCP 协议？

在开发这个工具之前，我深入研究了 Anthropic 在 2024 年 11 月推出的 MCP（Model Context Protocol，模型上下文协议）。

MCP 的设计理念让我眼前一亮：它就像 USB 接口统一了各种外设连接一样，旨在用一套标准协议打通所有 AI 工具与外部系统的连接。简单来说，MCP 就是 AI 世界的“通用语言”。

任何支持 MCP 的 AI 助手（如 Trae IDE、Claude Desktop 等），都可以通过统一的接口调用外部工具的能力。这正是我想要的！

## 我设计的核心功能

### 1. 支持 5 种主流格式

我选定了最常用的 5 种文档格式，覆盖绝大多数使用场景：

### 2. 25 种转换组合

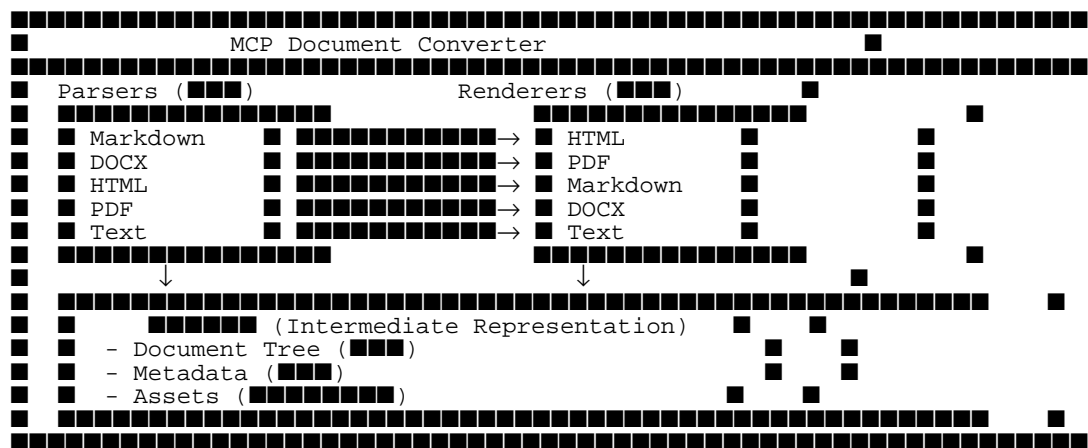
我实现了任意格式之间的双向转换，形成 $5 \times 5 = 25$ 种转换组合：

Markdown ↔ HTML ↔ DOCX ↔ PDF ↔ Text

无论你的源文档是什么格式，都可以轻松转换为目标格式。

### 3. 插件化架构

在设计架构时，我特别注重可扩展性。整个系统采用高度解耦的设计：



核心设计思想：所有格式都先解析为统一的中间表示（DocumentIR），然后再渲染为目标格式。这样做的好处是：

#### 4. 代码高亮与样式定制

作为开发者，我特别在意代码的可读性。因此 HTML 和 PDF 输出都支持语法高亮：

```
def hello_world():
    print("Hello, MCP Document Converter!")
```

同时支持自定义 CSS 样式，满足个性化需求。

### 如何快速使用？

## 安装

最简单的方式是通过uvx安装（无需手动配置 Python 环境）：

```
# ■■ GitHub ■■
uvx --from git+https://github.com/xt765/mcp-document-converter mcp-document-converter

# ■■■ Gitee ■■■■■■■■■■
uvx --from git+https://gitee.com/xt765/mcp-document-converter mcp-document-converter
```

## 在 Trae IDE 中配置

将以下内容添加到 Trae IDE 的 MCP 配置中：

```
{
  "mcpServers": {
    "mcp-document-converter": {
      "command": "uvx",
      "args": [
        "--from",
        "git+https://github.com/xt765/mcp-document-converter",
        "mcp-document-converter"
      ]
    }
  }
}
```

配置完成后，AI 助手就可以直接调用文档转换工具了！

## 实际使用示例

### 示例 1：Markdown 转 HTML

```
# ■ AI ■■ Markdown ■■■ HTML
convert_document(
  source_path="document.md",
  target_format="html",
```

```
        output_path="document.html"
    )
```

## 示例 2：DOCX 转 Markdown

```
# Word 转 Markdown
convert_document(
    source_path="report.docx",
    target_format="markdown"
)
```

## 示例 3：带自定义样式的转换

```
# 自定义 CSS
convert_document(
    source_path="document.md",
    target_format="html",
    options={
        "css": "body { font-family: Arial; font-size: 14px; }",
        "preserve_metadata": True
    }
)
```

## 示例 4：检查转换支持

```
# 检查 PDF 转 DOCX 支持
can_convert(
    source_format="pdf",
    target_format="docx"
)
# 返回: {"can_convert": true, "message": "Supported: pdf to docx"}
```

## 示例 5：列出支持的格式

```
# 列出支持的格式
list_supported_formats()
```

返回结果示例：

```
{
  "parsers": [
    { "format": "markdown", "extensions": [".md", ".markdown"] },
    { "format": "html", "extensions": [".html", ".htm"] },
    { "format": "docx", "extensions": [".docx"] },
    { "format": "pdf", "extensions": [".pdf"] },
    { "format": "text", "extensions": [".txt"] }
  ],
  "renderers": [...],
  "conversion_matrix": {
    "markdown": ["html", "pdf", "markdown", "docx", "text"],
    "html": ["html", "pdf", "markdown", "docx", "text"],
    ...
  },
  "summary": {
    "total_source_formats": 5,
    "total_target_formats": 5,
    "possible_conversions": 25
  }
}
```

## 开发过程中的技术挑战

在开发这个工具的过程中，我遇到了几个技术挑战：

### 1. Windows 上的 PDF 渲染

最初我使用 WeasyPrint 作为 PDF 渲染引擎，但它依赖 GTK 库，在 Windows 上安装非常麻烦。为了解决这个问题，我：

### 2. MCP 库的版本兼容性

MCP 库在 1.0.0 版本后有一些 API 变化，比如 `ListToolsRequestParams` 被移除。我及时更新了代码，确保兼容性。

### 3. 异步编程的处理

MCP 服务器使用异步编程模型，我设计了`main()`和`main_sync()`两个入口函数，确保 CLI 调用和 MCP 调用都能正常工作。

## 实际应用场景

这个工具在我的日常工作中已经帮了大忙：

### 场景 1：文档批量转换

我有一堆 Markdown 文档需要转换为 PDF 分享给客户：

```
import os

for file in os.listdir("docs"):
    if file.endswith(".md"):
        convert_document(
            source_path=f"docs/{file}",
            target_format="pdf",
            output_path=f"output/{file.replace('.md', '.pdf')}"
        )
```

### 场景 2：构建文档网站

将 Markdown 文档批量转换为 HTML，构建静态文档网站：

```
convert_document(
    source_path="README.md",
    target_format="html",
    options={
        "css": "custom.css",
```

```
        "template": "docs"
    }
)
```

## 场景 3：文档格式统一

团队成员使用不同格式编写文档，需要统一为

Markdown：

```
# ■■■ DOCX■PDF■Text ■■■■■■ Markdown
convert_document(source_path="report.docx", target_format="markdown")
convert_document(source_path="notes.txt", target_format="markdown")
```

## 与其他工具的比较

在开发之前，我调研了现有的文档转换工具：

我的设计理念：不是要做功能最全的工具，而是要做最适合 AI 时代的工具。

## 如何扩展？

如果你想添加新的格式支持，只需实现相应的解析器或渲染器：

```
from mcp_document_converter.core.parser import BaseParser
from mcp_document_converter.core.ir import DocumentIR, Node, NodeType

class MyParser(BaseParser):
    @property
    def supported_extensions(self) -> List[str]:
        return [".myext"]

    @property
    def format_name(self) -> str:
        return "myformat"
```



```
def parse(self, source, **options) -> DocumentIR:
    # 解析源
    document = DocumentIR()
    # ... 解析
    return document
```

然后注册到系统中：

```
from mcp_document_converter.registry import get_registry

registry = get_registry()
registry.register_parser(MyParser())
```

## 未来规划

这个工具还在持续迭代中，我计划添加以下功能：

## 邀请你一起参与

MCP Document Converter

是一个开源项目，我非常欢迎社区贡献：

项目地址： - GitHub:<https://github.com/xt765/mcp-document-converter> - Gitee（国内镜像）:<https://gitee.com/xt765/mcp-document-converter>

## 写在最后

开发 MCP Document Converter 的初衷很简单：让 AI 更好地为我们服务。

在 AI 时代，我们不应该被文档格式所束缚。无论是开发者、内容创作者还是办公人员，只要需要处理不同格式的文档，这个工具都能为你节省大量时间和精力。

如果你也觉得这个工具有用，请在 GitHub 上给我一个  
☐☐ Star，这是对我最大的鼓励！

立即体验：[-GitHub-Gitee](#)（国内镜像）

我是玄同765（xt765），一个热爱开源和 AI 的开发者。欢迎与我交流！