

技术方案

技术栈

层级	选型	理由
语言	TypeScript	类型安全；AI 编程工具生成代码的正确率更高
插件框架	WXT	约定式开发；自动生成 manifest.json；内置 HMR
UI	原生 HTML + CSS + TS	Popup 和 Options 页逻辑相对简单，不引入 Vue/React
PDF 生成	pdf-lib	纯前端拼装 PDF，支持图片/PDF 合并
LLM	OpenAI 兼容 API	语义字段匹配 + VLM 图片理解；兼容任何 OpenAI 格式端点（OpenAI / DeepSeek / Ollama 等）

项目结构

```
auto-filler/
├── wxt.config.ts           # WXT 配置（权限、manifest 覆盖、快捷键）
├── entrypoints/
│   ├── background.ts      # Service Worker: 扫描调度、LLM 匹配、流式编排、文件匹配、命令处理
│   ├── content.ts         # 注入网页: DOM 扫描、字段填充、打字机动画
│   └── popup/             # 点击图标弹出的面板（纯 UI 层）
│       ├── index.html
│       ├── main.ts        # 状态机: idle → scanning → result → filling → filled + stream
│       └── style.css
│   └── options/           # 插件内设置页: 个人信息 + 材料管理 + API 配置 + PDF 合并
│       ├── index.html
│       ├── main.ts
│       └── style.css
├── utils/
│   ├── matcher.ts         # LLM 语义匹配 + 增量 JSON 解析器 + 流式 SSE 生成器
│   ├── vlm.ts             # VLM 视觉理解（图片描述、PPT 解析）[新增]
│   ├── db.ts              # IndexedDB 封装（textFields + blockCategories + fileRecords）
│   ├── storage.ts         # chrome.storage.local 封装（API 配置）
│   └── pdf-merge.ts       # PDF 合并（pdf-lib）
├── public/
│   └── icon/              # 插件图标 + SVG
└── docs/                  # 项目文档
```

核心流程

流程 1：传统扫描填充（显式确认）

1. 用户在 Options 页录入个人信息（KV 字段 + 结构化 block）+ 管理材料文件 + 配置 API
2. 用户打开任意报名/申请页面，点击 Popup "开始扫描"
3. Content script 扫描页面所有可填充元素
4. Background 调用 LLM API 做语义匹配 → 返回字段映射
5. 用户在 Popup 中确认匹配结果 → 一键填充

流程 2：流式自动填充（零确认）

1. 用户点击 Popup "流式自动填充" 或按快捷键 `Alt+Shift+F`
2. Background 发送 `fillStreamInit` 初始化 content 端打字队列
3. Background 发起 SSE 流式 LLM 调用，`IncrementalJsonParser` 边走边解析

4. 短字段：完整 JSON 对象解析完 → `fillField` 即时填充
5. 长文本字段 (`fillMode=long`)：JSON 对象还在解析中 → `fillTypeChunk` 每 5 字符推送到 content → 30ms/字打字机动画 → `fillTypeCommit` 提交
6. 每个字段填充完推进度到 Popup (如 Popup 已打开)
7. 浏览器自动 `scrollIntoView({ behavior: 'smooth' })` 到当前填充字段

流程 3：文件智能解析 [新增]

1. 用户上传文件到材料管理页面 (图片 / 文本 / PDF / PPT)
2. 后台自动检测文件类型，调用对应处理管线：
 - 图片 (jpg/png/webp) → VLM 生成自然语言描述文本 → 存入 `FileRecord.textContent`
 - 纯文本 (txt/csv/json/xml/md 等) → 直接读取文本内容 → 存入 `FileRecord.textContent`
 - PPT (pptx) → 逐页渲染为图片 → 逐页调用 VLM 解析 → 拼接存入 `FileRecord.textContent`
 - PDF → 由 pdf-lib 处理合并，不调用 VLM (PDF 通常包含表单元数据)
3. 解析后的 `textContent` 作为文件语义索引，后续匹配长文本字段时可被 LLM 引用

文件智能解析详细设计

VLM 模块： `utils/vlm.ts` [新增]

```
// 调用 VLM 生成图片描述
async function describeImage(base64Image: string, mimeType: string, apiConfig: ApiConf

// 读取纯文本文件内容
async function extractTextFile(file: Blob): Promise<string>

// 解析 PPT 文件 → 逐页渲染为图片 → 逐页调用 VLM → 拼接描述
async function parsePptx(file: Blob, apiConfig: ApiConfig, onProgress?: (page: number,

// 统一入口：根据 MIME 类型分发到对应管线
async function processFile(file: File, apiConfig: ApiConfig, onProgress?: FileProcessP
```

VLM API 调用格式

使用 OpenAI 兼容的 vision API，发送 base64 图片 + 指令 prompt：

```
const response = await fetch(`${baseUrl}/chat/completions`, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${apiKey}`,
  },
  body: JSON.stringify({
    model,
    messages: [{
      role: 'user',
      content: [
        {
          type: 'image_url',
          image_url: { url: `data:${mimeType};base64,${base64Image}` },
        },
        {
          type: 'text',
          text: '请详细描述这张图片的内容 ... ',
        },
      ],
    }],
    max_tokens: 1024,
  }),
});
```

文件类型路由表

文件扩展名	MIME 类型	处理方式
.jpg/.jpeg	image/jpeg	VLM 描述
.png	image/png	VLM 描述
.webp	image/webp	VLM 描述
.txt	text/plain	直接读取
.csv	text/csv	直接读取（前 2000 字符）
.json	application/json	直接读取（提取 key 摘要）
.xml	text/xml, application/xml	直接读取（前 2000 字符）
.md	text/markdown	直接读取
.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation	逐页 VLM 解析
.docx	application/vnd.openxmlformats-officedocument.wordprocessingml.document	提取文本 [待定]
.pdf	application/pdf	pdf-lib 合并，不调用 VLM

PPT 解析流程 (pptx)

- 1. 使用 JSZip 解压 pptx（本质是 zip 包）
- 2. 读取 ppt/slides/slide*.xml 获取每页的文本内容（可选：用文本作为 VLM prompt 上下文）
- 3. 使用 ppt2canvas 或手动解析将每页 Slide 渲染为 canvas → toDataURL → base64
- 4. 逐页调用 VLM：

"你是一个文档解析助手。这是演示文稿的第 {i}/{total} 页。
请提取本页中的所有文字内容，并简要描述本页的视觉布局和图表信息。"

- 5. 拼接所有页面的描述结果，格式：

【第 1 页】标题: xxx 内容: xxx 图表: xxx
【第 2 页】...

存储变更

`FileRecord` 新增字段:

```
interface FileRecord {  
  // ... 现有字段  
  textContent: string; // 【新增】VLM/文本提取后的自然语言描述，用于语义索引  
  textContentStatus: 'pending' | 'processing' | 'done' | 'error'; // 【新增】  
  textContentError?: string; // 【新增】处理失败时的错误信息  
}
```

无需新建 Object Store, `textContent` 作为 `fileRecords` 的新字段即可。需要升级数据库到 v4, 新增索引 `textContentStatus`。

处理时机

- **上传时异步处理:** 文件上传到 IndexedDB 后, 状态标记为 `pending`
- **后台处理队列:** background service worker 轮询 `pending` 状态的记录, 逐一调用 VLM
- **UI 状态显示:** Options 材料列表展示处理状态图标 (⌚ 等待中 / 🔄 解析中 / ✅ 已完成 / ❌ 失败)
- **失败重试:** 用户可手动点击重试

架构决策

LLM 接入: OpenAI 兼容 API

采用 OpenAI 兼容的 `/v1/chat/completions` 接口, 用户自行配置:

- **API Base URL** (默认 `https://api.openai.com/v1`, 可改为任何兼容端点)
- **API Key**
- **Model 名称**

匹配任务用小模型即可 (gpt-4o-mini 级别), 成本极低。VLM 任务建议使用支持视觉的模型 (如 gpt-4o-mini / gpt-4o), 或 DeepSeek-VL2 等兼容端点。

数据存储

存储	内容	理由
<code>chrome.storage.local</code>	API 配置	数据量小，API 简单
IndexedDB <code>autoFillerDB</code>	个人信息 (KV + block) 、材料文件及其解析文本	支持 Blob、索引查询、大容量

Popup / Background 职责分离

- **Background Service Worker**: 执行扫描、LLM 匹配、SSE 流式编排、文件匹配、VLM 文件解析、命令处理
- **Popup**: 纯 UI 展示层，通过 `chrome.runtime.sendMessage` （传统流程）或 `chrome.runtime.connect` （流式进度推送）与 Background 通信

流式填充：增量 JSON 解析 + 打字机动画

- `IncrementalJsonParser` : 逐字符状态机，追踪 `[/ { / } /]` 深度、字符串边界、转义序列 (含 `\uXXXX` Unicode) 。每 5 字符 emit `value_chunk` , 对象闭合时 emit `match_complete`
- `matchFieldsStream()` : async generator, SSE `data:` 行解析 → feed parser → yield 事件
- **Content 端**: `TypingEntry` 管理每个字段的 buffer + pos + 30ms 定时器, `scrollIntoView` 自动跟随
- **双模式共存**: 传统 `matchFields()` 非流式接口保持不变

快捷键触发

- 注册 `commands.stream-fill` , 默认快捷键 `Alt+Shift+F`
- 用户可在 `chrome://extensions/shortcuts` 自定义
- 触发后直接执行流式填充，无需打开 Popup

Content Script 注入

采用 WXT 声明式注入 (`content_scripts` in manifest) , 匹配 `<all_urls>` 。

消息协议

传统流程 (sendMessage)

```
popup → background: { type: 'startScan' }
background → content: { type: 'scan' }
content → background: Array<{ index, field }>
background → popup: ScanSuccessResponse { matches, fields }
popup → background: { type: 'startFill', payload: { matches } }
background → content: { type: 'fill', items }
content → background: FillResult { success, failure }
```

流式流程 (sendMessage + connect port)

Popup ↔ Background (`chrome.runtime.connect` port, 名为 `stream-fill`) :

```
popup → background: { type: 'startStreamScan' }
background → popup: { type: 'streamProgress', matched, total, latestLabel }
background → popup: { type: 'streamComplete', matched, errorCount }
background → popup: { type: 'streamError', error }
```

Background → Content (sequential `chrome.tabs.sendMessage`) :

```
{ type: 'fillStreamInit', items: Array<{ index, fillMode }> }
{ type: 'fillField', index, value }           // 短文本即时填充
{ type: 'fillTypeChunk', index, chunk }      // 长文本逐字流
{ type: 'fillTypeCommit', index }            // 长文本提交
{ type: 'fillStreamComplete' }               // 流式完成
```

快捷键流程 (无 Popup)

```
chrome.commands.onCommand('stream-fill')
  → handleStreamScan() // port = undefined, 静默完成
  → content 端同样走 fillStreamInit → fillField/fillTypeChunk → fillStreamComplete
```


存储结构

chrome.storage.local — API 配置

```
interface ApiConfig {
  baseUrl: string;    // 默认 "https://api.openai.com/v1"
  apiKey: string;
  model: string;      // 默认 "gpt-4o-mini"
  providerId: string;
}
```

IndexedDB: **autoFillerDB** v4 [升级]

textFieldds — 键值文本字段

keyPath: **id** (autoIncrement)

```
interface TextField {
  id?: number;    // 自增主键
  key: string;    // 用户自定义 key, 如 "姓名"
  value: string;  // 对应值
}
```

fileRecords — 文件记录 [v4 新增字段]

keyPath: **id** (autoIncrement) 索引: **fileType**, **createdAt**, **categoryId**, **textContentStatus**

```
interface FileRecord {
  id?: number;
  filename: string;
  fileType: string;
  fileBody: Blob;
  fileSize: number;
  fileDescription: string;
  categoryId: number;
  createdAt: number;
  textContent: string;          // [v4] VLM/文本提取后的描述文本
  textContentStatus: 'pending' | 'processing' | 'done' | 'error'; // [v4]
  textContentError?: string;    // [v4]
}
```

blockCategories — 结构化数据块

keyPath: `id` (autoIncrement)

```
interface BlockCategory {
  id?: number;
  title: string;
  items: BlockItem[];
}
```

`categories` — 文件分类

keyPath: `id` (autoIncrement) 索引: `sortOrder`

```
interface Category {
  id?: number;
  name: string;
  icon: string;
  sortOrder: number;
  isDefault: boolean;
}
```

关键 Chrome API

- `chrome.storage.local` — 持久化存储 API 配置
- `chrome.tabs` — 与当前标签页通信
- `chrome.runtime.sendMessage` — Popup / Background / Content Script 间消息传递
- `chrome.runtime.connect` — Popup ↔ Background 双向持续通信 (流式进度推送)
- `chrome.commands` — 键盘快捷键触发流式填充
- `chrome.scripting` — 动态注入 content script (fallback)

开发命令

<code>npm run dev</code>	# Chrome 扩展开发模式 (HMR)
<code>npm run dev:firefox</code>	# Firefox 开发模式
<code>npm run build</code>	# 生产构建
<code>npm run zip</code>	# 打包 .zip (Chrome Web Store)
<code>npm run compile</code>	# 仅类型检查 (<code>tsc --noEmit</code>)