OpenClaw

终极记忆系统

AI 智能体上下文失忆问题的
综合解决方案

技术报告与实施指南

发布时间：2026 年春节
来源：linux.do 论坛帖子
分析日期：2026 年 2 月 17 日

协作工具：Antigravity & Cursor
后端：qmd（混合搜索与重排序）
运行时：OpenClaw Cron 系统

目录

1. 概述

OpenClaw 终极记忆系统是一套由社区设计的解决方案，旨在解决 AI 编程智能体中持久存在的「失忆问题」——具体来说是 OpenClaw（又名 Claude Code）的上下文遗忘问题。该系统发布于 2026 年春节期间，由作者与 Antigravity 工具及 Cursor 协作，在大约四五个小时内设计完成——据作者本人所述，是一边看着「太难看了」的春晚一边完成的。

该系统提供了一条结构化的自动化流水线，用于捕获、压缩、蒸馏和归档 OpenClaw 会话中的上下文信息，确保 AI 智能体在跨会话时保持有意义的长期记忆，而无需人工干预。

1.1. 核心问题

像 OpenClaw 这样的 AI 编程助手存在 上下文失忆 问题：每次新会话都从零开始。智能体对之前的对话、决策、用户偏好和项目历史毫无记忆。这迫使用户反复解释自己的偏好、重新描述项目架构、重新说明重要决策——这是一个乏味且容易出错的过程，严重降低了用户体验和工作效率。

1.2. 方案一览

终极记忆系统通过三层时间架构解决这一问题：

层级	存储位置	保留策略
每日日志	`memory/YYYY-MM-DD.md`	原始会话摘要，活跃保留 7 天
每周摘要	`memory/weekly/YYYY-MM-DD.md`	压缩后的周度摘要，长期保留
长期记忆	`MEMORY.md`	蒸馏后的关键知识，上限 80 行

两个自动化 Cron Job 驱动整个流水线：

1. **memory-sync**（每日 4 次）：将会话内容捕获到每日日志文件中。
2. **memory-tidy**（每夜 1 次）：将旧的每日日志压缩为周度摘要，将洞察蒸馏到 `MEMORY.md`，并归档过期文件。

1.3. 关键设计指标

指标	数值
所需 Cron Job 数量	2 个
MEMORY.md 硬性上限	80 行 / 5 KB
同步频率	每日 4 次（10:00、14:00、18:00、22:00）
整理频率	每日 1 次（03:00）
最低会话阈值	2 条用户消息
幂等性机制	Session ID 前 8 字符去重
长期记忆准入条件	4 个条件同时满足

备份策略	修改前自动备份
每日保留窗口	7 天后压缩
外部依赖	`qmd` 混合搜索引擎

2. 现有方案与竞品分析

在设计终极记忆系统之前，作者对 OpenClaw/Claude Code 记忆生态中的现有方案进行了系统性调研。本章介绍四种主要的先行方案及其优缺点。

2.1. Calicastle 三层架构

来源：X/Twitter（@calicastle），社区分享方案。

Calicastle 方法提出了三层记忆层次结构：

1. **短期**：当前会话上下文
2. **中期**：近期会话摘要
3. **长期**：持久化知识库

优点：
- 架构清晰直观
- 简单易懂，容易实现
- 三层分离提供了自然的时间分段

缺点：
- **无去重机制**：如果相同信息在多个会话中被捕获，会不断累积而不合并，导致记忆文件膨胀。
- **捕获频率低**：系统每天仅捕获一次，意味着如果用户不手动触发保存，日内会话可能会丢失。
- **无验证逻辑**：没有机制验证被捕获的记忆在进入长期存储前是否仍然相关、准确或非冗余。

2.2. openclaw-memory（官方 Skill）

来源：GitHub 仓库（geq1fan），定位为 OpenClaw 的官方或半官方 Skill。

该方案将记忆实现为 OpenClaw Skill，具备自动验证能力：

优点：
- 支持自动验证，包括过时检测、重复检测和冲突检测
- 作为原生 OpenClaw Skill 集成，减少使用摩擦

缺点：
- **架构复杂**：需要两个独立的 Cron Job，编排逻辑较为复杂。
- **Python 依赖**：需要 Python 运行时环境，为未配置 Python 的用户增加了安装和维护开销。

2.3. openclaw-engram（社区插件）

来源：GitHub 社区插件。

Engram 是生态中功能最丰富的方案，实现了一套受认知科学启发的复杂记忆模型：

核心功能：
- **10 种记忆分类**：将记忆分为十种不同类型（如程序性记忆、情景记忆、语义记忆等）
- **信心分数**：每条记忆条目附带一个表示可靠性的数值信心分数
- **实体档案**：为跨会话提及的实体（人物、项目、技术）维护结构化档案
- **丰富的元数据**：提供广泛的标签和交叉引用系统

优点：

- 社区方案中功能最全面
- 精细的分类系统提供了对记忆类型的细粒度控制
- 实体档案支持关系感知的上下文检索

缺点：
- **复杂度高**：10 类分类系统和信心评分带来了显著的认知和实现开销。
- **需要 API Key**：部分功能需要外部 API Key，引入了外部依赖和潜在成本。
- **过度工程风险**：对大多数用户而言，其复杂度远超编程助手实际的记忆需求。

2.4. Supermemory / Mem0（云服务）

来源：官方插件生态，云端记忆服务。

这些方案通过云托管服务提供「即插即用」的记忆功能：

优点：
- **零配置**：安装即可使用，无需任何设置
- **托管基础设施**：无需本地存储管理、备份或维护

缺点：
- **云端依赖**：记忆数据存储在远程服务器上，存在可用性和延迟问题。
- **付费**：需要持续订阅费用。
- **隐私顾虑**：敏感的项目数据和用户偏好被传输到第三方服务器并存储——对于涉及专有或机密代码库的用户来说是重大隐患。

2.5. 对比分析总结

功能	Calicastle	openclaw-memory	engram	云服务	终极方案
去重	否	是	是	是	是
自动验证	否	是	是	是	是（4D）
大小控制	否	部分	否	N/A	是（80 行）
会话过滤	否	否	否	N/A	是（最少 2 条）
备份	否	否	否	云端	是（自动）
纯本地	是	是	部分	否	是
复杂度	低	中	高	低	低–中
Cron Job 数	1	2	N/A	0	2
外部依赖	无	Python	API Key	云订阅	`qmd`

Table 1: 各记忆方案功能对比

终极记忆系统有意占据了一个平衡点：比 Calicastle 更健壮（具备去重、验证和大小控制），比 engram 更简洁（2 个 Cron Job 对比 10 种记忆分类），且与云方案不同，完全在本地运行。

3. 架构设计

终极记忆系统采用 时间压缩流水线 架构，原始会话数据流经逐级收紧的过滤器，直到只有最关键的知识存活到长期记忆中。

3.1. 设计原则

系统建立在六条核心设计原则之上，每条原则都针对先行方案中观察到的特定失败模式：

3.1.1. 幂等操作

系统中的每个写操作都是幂等的：多次运行同一操作与运行一次产生相同结果。这通过基于 Session ID 的去重实现——写入前检查每个会话唯一标识符的前 8 个字符是否已存在于文件中。

设计理由：在基于 Cron 的系统中，任务可能因故障、时钟偏移或手动重新执行而被重试。没有幂等性，重复条目会随时间累积，污染记忆库。

3.1.2. 硬性大小限制

`MEMORY.md` 被限制在 80 行和 5 KB 的硬性上限内。当接近限制时，必须先压缩或合并现有条目，才能添加新条目。

设计理由：没有大小限制，记忆文件会无限增长。庞大的记忆文件会消耗上下文窗口空间，用可能过时或低价值的信息降低 AI 性能。80 行限制迫使持续策展，确保只有最高价值的知识得以保留。

3.1.3. 信噪比过滤

用户消息少于 2 条的会话在捕获时会被自动跳过。这过滤掉了意外会话、快速测试和琐碎交互。

设计理由：许多 OpenClaw 会话是简短的测试或一次性问题，不会产生持久知识。捕获这些内容会产生噪音，稀释真正洞察的价值。

3.1.4. 四维验证

在任何信息进入 `MEMORY.md`（长期记忆）之前，必须 同时满足 以下四个条件：

1. **错误预防**：没有这条信息，智能体会犯一个具体的、明确的错误。
2. **广泛适用**：该信息适用于许多未来的对话，而非仅限于某个特定任务。
3. **自包含**：条目本身可以独立理解，不需要额外上下文。
4. **非重复**：该信息在现有 `MEMORY.md` 中尚不存在。

此外，还需进行 反向检查：在写入条目之前，智能体必须明确说明没有这条信息会发生什么具体错误。如果无法给出具体错误，则拒绝该条目。

设计理由：这是系统中最关键的过滤器。长期记忆是宝贵的空间（80 行），每条记录都必须证明自己的价值。四维测试防止了常见的「有趣但不可操作」的条目占用空间却不提供价值的失败模式。

3.1.5. 自动备份

在对 `MEMORY.md` 进行任何修改之前，系统会在 `memory/archive/` 中创建带时间戳的备份副本：

```
mkdir -p memory/archive
cp MEMORY.md memory/archive/MEMORY.md.bak-$(date +%F)
```

设计理由：记忆蒸馏是一个有损过程——将每日日志压缩为 80 行必然会丢弃信息。如果蒸馏过程做出了糟糕的决定（例如删除了关键条目），备份可以恢复。

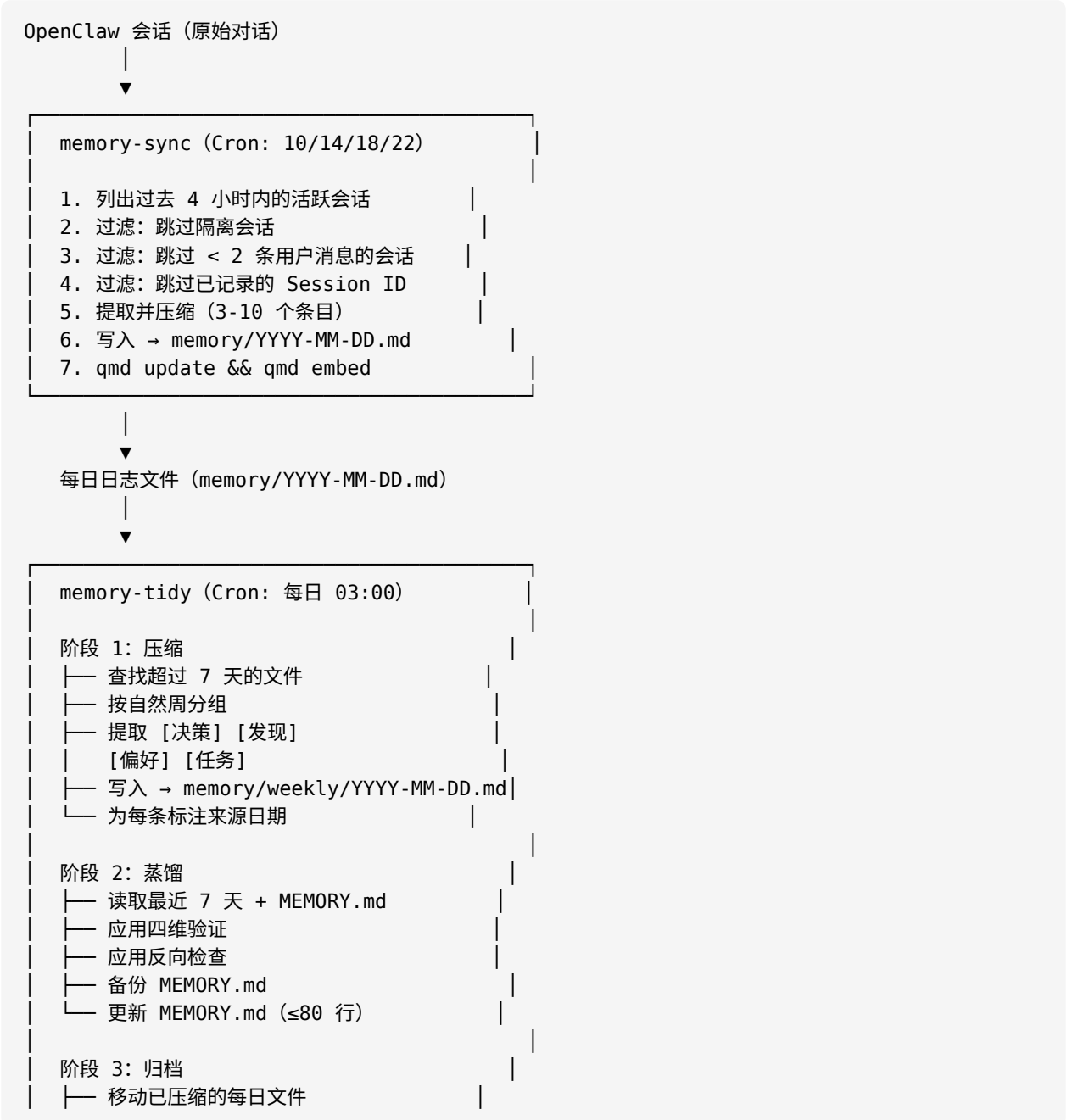
3.1.6. 精简架构

整个系统仅靠 2 个 Cron Job 运行。除了 OpenClaw 本身提供的功能外，不需要额外的服务、数据库、API Key 或后台守护进程。

设计理由：复杂性是可靠性的敌人。每增加一个组件都会引入潜在的故障模式、配置负担和维护开销。作者明确选择牺牲某些功能（如 engram 的实体档案），以换取运维简洁性。

3.2. 数据流架构

系统的完整数据流可以描述为一条流水线：



```
│  │   → memory/archive/YYYY/           │
│  └── qmd update && qmd embed           │
└────────────────────────────────────────┘
        │
        ▼
   MEMORY.md（80 行，长期记忆）
   每周摘要（memory/weekly/）
   归档文件（memory/archive/YYYY/）
```

这条流水线实现了 时间压缩：原始会话（可能数千行）被压缩为每日摘要（数十行），再压缩为每周摘要（每周几十行），最终蒸馏为代表智能体永久知识库的 80 行 MEMORY.md。

3.3. 三层存储模型

3.3.1. 第一层：每日日志（memory/YYYY-MM-DD.md）

每日日志是原始捕获层。每个文件包含当天的会话摘要，格式如下：

```
## HH:MM session:FIRST8 | N messages

- 用户请求了 X
- 智能体决定了 Y
- 重要结果：Z
```

特征：

- 由 memory-sync 自动创建
- 在活跃存储中保留 7 天
- 每个会话压缩为 3–10 个摘要条目
- Session ID 去重防止重复
- 由 qmd 索引以支持混合搜索

3.3.2. 第二层：每周摘要（memory/weekly/YYYY-MM-DD.md）

每周摘要是第一级压缩层。它们将上一周的每日日志聚合为分类摘要：

```
### YYYY-MM-DD
[决策] 切换到 PostgreSQL 作为持久化方案（src: 2026-02-10）
[发现] qmd 支持带重排序的混合搜索（src: 2026-02-11）
[偏好] 用户偏好 Typst 而非 LaTeX 写报告（src: 2026-02-12）
[任务] 完成了 API 迁移到 v2（src: 2026-02-13）
```

特征：

- 以每周的周一日期命名
- 四个分类：决策、发现、偏好、任务
- 每条标注来源日期以保证可追溯性
- 幂等：已存在的周度段落不会被重新处理
- 无限期保留（不受压缩影响）

3.3.3. 第三层：长期记忆（MEMORY.md）

最终蒸馏层。仅包含通过四维验证测试的信息：

```
# Long-Term Memory

> Only write info here that you'd make mistakes without.
> Event logs stay in daily files.
> Hard limit: 80 lines / 5KB. Must compress before adding when over limit.

## User Preferences

## Active Projects

## Key Decisions

## Important Contacts
```

特征：

- 硬性上限：80 行 / 5 KB
- 四个预定义分区
- 每条记录必须满足全部四个验证条件
- 每次修改前自动备份
- 加载到每个 OpenClaw 会话上下文中

4. memory-sync Cron Job

memory-sync 任务是系统的数据摄取层。每天运行四次，负责将会话内容捕获到结构化的每日日志文件中。

4.1. 调度与配置

```
openclaw cron add \
  --name "memory-sync" \
  --cron "0 10,14,18,22 * * *" \
  --tz "Asia/Shanghai" \
  --session isolated \
  --wake now \
  --delivery none
```

参数	值与设计理由
--cron	0 10,14,18,22 * * * — 每天在北京时间 10 点、14 点、18 点、22 点各捕获一次
--tz	Asia/Shanghai — 时区感知调度
--session isolated	在独立的隔离会话中运行，避免污染用户对话
--wake now	立即唤醒，触发后无延迟
--delivery none	静默执行，不向用户发送通知

四个每日窗口（10:00、14:00、18:00、22:00）的选择覆盖了典型的工作时间段，间隔 4 小时，确保没有会话超过 4 小时而未被捕获。回溯窗口与此间隔匹配：每次运行扫描过去 4 小时的会话。

4.2. 执行流水线

memory-sync 任务执行一个精心排序的 11 步流水线：

4.2.1. 步骤 1：会话发现

```
使用 sessions_list 获取过去 4 小时内有活动的会话
```

任务查询 OpenClaw 会话 API，获取在 4 小时回溯窗口内有活动的所有会话。返回的会话元数据包括 Session ID、消息数量和会话类型。

4.2.2. 步骤 2：隔离过滤

```
跳过隔离会话
```

标记为 isolated 的会话（包括 memory-sync 任务自身的会话）将被过滤。这防止记忆系统记录自己的操作，避免产生递归捕获循环。

4.2.3. 步骤 3：会话内容获取

```
对每个会话，使用 sessions_history 读取对话内容
```

对每个候选会话，获取完整的对话历史。包括用户消息和助手回复，为摘要提取提供原始素材。

4.2.4. 步骤 4：信号阈值过滤

```
跳过 user_message_count < 2 的会话
```

用户消息少于 2 条的会话将被丢弃。这排除了：

- 意外启动的会话（0 条消息）
- 快速的一次性问题（1 条消息）
- 不会产生持久知识的测试/调试会话

阈值设为 2 是作为「有意义对话」的最低标准——至少有一个问题和一次跟进。

4.2.5. 步骤 5：提前退出检查

```
如果没有剩余的有效会话，什么都不做 --- 回复 ANNOUNCE_SKIP 并停止。
```

如果所有会话都被过滤掉了（在非活跃时段很常见），任务立即以 `ANNOUNCE_SKIP` 响应终止。这避免了不必要的文件 I/O 和 `qmd` 索引更新。

4.2.6. 步骤 6：每日文件初始化

```
读取当天的 memory/YYYY-MM-DD.md（如不存在则创建）
```

任务读取或创建当前日期的每日日志文件。如果文件不存在（当天首次捕获），则初始化为空文件。

4.2.7. 步骤 7：幂等性检查

```
如果某个 session_id 的前 8 个字符已出现在文件中，则跳过该会话
```

这是关键的去重步骤。每个会话的唯一标识符被截断为前 8 个字符，并在每日文件中搜索此前缀。如果找到，说明该会话已被记录，予以跳过。

为什么是前 8 个字符？ Session ID 通常是 UUID（128 位，32 个十六进制字符）。前 8 个字符提供 $16^8 = 4.3 \times 10^9$ 种唯一组合，在一天内的会话数量（通常 5–20 个）中发生碰撞的概率极低。

4.2.8. 步骤 8：内容提取与压缩

```
提取：用户的关键请求、助手的结论/决策、
重要的操作结果。将每个会话压缩为 3-10 个摘要条目。
```

对每个未记录的会话，智能体提取三类信息：

- **用户请求**：用户要求了什么？
- **智能体结论/决策**：智能体决定或推荐了什么？

- **操作结果**：执行操作的结果是什么？

每个会话被压缩为 3–10 个要点摘要条目。这个范围确保了有意义的会话有足够的细节，同时防止冗长的会话主导每日日志。

4.2.9. 步骤 9：格式化追加

```
以如下格式追加到每日文件：
## HH:MM session:FIRST8 | N messages
```

提取的摘要以结构化标题追加到每日日志中：
- `HH:MM`：捕获的时间戳
- `session:FIRST8`：Session ID 的前 8 个字符（用于幂等性追踪）
- `N messages`：消息总数（用于了解会话深度）

4.2.10. 步骤 10：索引更新

```
运行 qmd update && qmd embed
```

写入新条目后，更新 `qmd` 搜索引擎。包括两个操作：
- `qmd update`：更新文件索引以包含新建/修改的文件
- `qmd embed`：生成向量嵌入以支持语义搜索

这确保新捕获的记忆可以立即通过 `qmd query` 进行搜索。

4.2.11. 步骤 11：静默完成

```
完成后回复 ANNOUNCE_SKIP。
```

由于指定了 `--delivery none`，任务以 `ANNOUNCE_SKIP` 响应静默完成。不向用户发送任何通知。

4.3. 幂等性深入分析

幂等性机制值得特别关注，因为它是本系统区别于先行方案的显著特征。

问题：Cron 任务可能在同一窗口内被多次触发，原因包括：
- 系统时钟调整
- 用户手动重新触发
- OpenClaw 调度器在临时故障后重试
- 时区转换（如夏令时变更）

解决方案：Session ID 前缀检查为每个会话创建了 一次性写入 保证。检查针对每日文件的原始文本内容执行，简单而健壮：

```
# 伪代码
for session in active_sessions:
    prefix = session.id[:8]
    if prefix in daily_file_content:
        continue  # 已记录
```

```
    else:
        append(daily_file, summarize(session))
```

这种方法以少量存储开销（每个标题中的 `session:FIRST8` 标签）换取完整的幂等性，而无需单独的状态文件或数据库。

5. memory-tidy Cron Job

memory-tidy 任务是系统的智能层。每夜运行一次，执行三个顺序阶段：压缩旧的每日日志、将洞察蒸馏到长期记忆、以及归档已处理的文件。

5.1. 调度与配置

```
openclaw cron add \
  --name "memory-tidy" \
  --cron "0 3 * * *" \
  --tz "Asia/Shanghai" \
  --session isolated \
  --wake now \
  --deliver
```

参数	值与设计理由
--cron	0 3 * * * — 每天北京时间凌晨 3:00 运行
--session isolated	隔离会话，确保安全修改 MEMORY.md
--deliver	交付摘要（与 memory-sync 的静默模式不同）

凌晨 3:00 的调度是有意为之：在通常没有活跃会话的非工作时间运行，避免与 memory-sync 捕获和用户会话冲突。--deliver 标志确保用户收到简要的变更摘要，为记忆维护过程提供透明度。

任务提示中出现了一条关键说明："你被明确授权在此隔离会话中读取和修改 *MEMORY.md*。" 这种显式授权是必要的，因为 OpenClaw 的安全机制可能会阻止自动化任务修改 MEMORY.md 这样重要的文件。

5.2. 阶段 1：压缩

压缩阶段将单个每日日志转换为每周摘要，减少文件数量并提取结构化分类。

5.2.1. 步骤 1：文件发现

```
列出 memory/ 中所有以日期命名的文件（YYYY-MM-DD.md）
```

任务扫描 memory/ 目录，查找匹配 YYYY-MM-DD.md 命名模式的文件。

5.2.2. 步骤 2：时间过滤

```
识别超过 7 天的文件。如果没有，跳过此阶段。
```

只有超过 7 天的每日日志才有资格被压缩。这确保近期日志保持其原始的详细形式，便于即时参考。

7 天保留窗口的选择平衡了以下因素：

- **时效性**：近期会话经常被引用，受益于详细的日志
- **数据量**：7 天后，每日日志积累了足够的数据来产生有意义的周度摘要

- **对齐**：自然周边界与人类的工作周期吻合

5.2.3. 步骤 3：按周分组

```
按自然周分组，生成 memory/weekly/YYYY-MM-DD.md（以周一命名）
```

每日日志按其所在的自然日历周分组，每组生成一个以该周周一日期命名的周度摘要文件。例如：

- `2026-02-03.md`、`2026-02-04.md`、…、`2026-02-07.md` → `memory/weekly/2026-02-03.md`

5.2.4. 步骤 4：分类提取

```
提取 [决策] [发现] [偏好] [任务]
为每条标注 (src: YYYY-MM-DD)
```

压缩过程将信息提取到四个语义分类中：

分类	说明	示例
[决策]	影响未来工作的选择	切换到 PostgreSQL 持久化方案（src: 2026-02-10）
[发现]	获得的新知识或洞察	qmd 支持否定查询（src: 2026-02-11）
[偏好]	用户明示或推断的偏好	偏好 Typst 而非 LaTeX（src: 2026-02-12）
[任务]	已完成或进行中的工作	完成了认证模块重构（src: 2026-02-13）

每条记录标注其来源日期 `(src: YYYY-MM-DD)`，保持出处信息以支持追溯。

5.2.5. 步骤 5：每周幂等性

```
幂等：如果 ### YYYY-MM-DD 段落已存在于每周文件中，则跳过
```

如果某个每日日志对应的段落已存在于每周摘要中，则跳过。这防止了 tidy 任务在连续日期运行时因每日日志重叠而重复处理。

5.3. 阶段 2：蒸馏

蒸馏阶段是最具智力挑战性的步骤。它决定哪些信息值得在 `MEMORY.md` 中永久保留。

5.3.1. 步骤 6：来源读取

```
读取最近 7 天的每日文件 + 当前 MEMORY.md
```

蒸馏过程读取：

- 过去 7 天的所有每日日志文件（活跃窗口）
- `MEMORY.md` 的当前内容

这为智能体提供了新的候选条目和现有的长期条目，以供比较。

5.3.2. 步骤 7：四维验证

```
必须同时满足全部四个条件：
(a) 没有这条信息，智能体会犯一个具体的错误
(b) 适用于许多未来的对话
(c) 自包含且可独立理解
(d) 在现有 MEMORY.md 中无重复
```

每个候选条目同时按四个维度评估。任何一个维度不通过即被淘汰：

维度 (a)：错误预防

最强的过滤器。该信息必须能防止一个 具体的、明确的 错误。模糊的好处如「知道挺好的」或「可能有用」是不够的。

通过示例："用户的项目使用 Svelte 5 的 Runes 语法，而非 Svelte 4"（没有这条信息，智能体会生成过时的 Svelte 4 代码）。

不通过示例："用户提到他们喜欢 Python"（没有这条信息不会导致具体错误）。

维度 (b)：广泛适用性

该信息必须跨多个未来对话相关，而非仅限于单个任务。

通过示例："在此项目中始终使用 `pnpm` 而非 `npm`"（适用于每次包管理交互）。

不通过示例:"Issue #42 中的 bug 是由空指针引起的"（仅与该特定 issue 相关）。

维度 (c)：自包含性

条目必须无需额外上下文即可理解。孤立阅读时应传达完整、可操作的信息。

通过示例："数据库连接使用连接池，通过 `pg` 库，最大 10 个连接。"

不通过示例："使用我们讨论过的那个方案"（需要未来会话中不可用的上下文）。

维度 (d)：非重复性

该信息不得以任何形式已存在于现有 `MEMORY.md` 中。

5.3.3. 步骤 8：反向检查

```
写入前问自己：没有这条信息会发生什么具体错误？
如果答不上来，就不要写。
```

这是一个元验证步骤，迫使明确表达条目的价值。它作为最后一道屏障，防止那些「感觉重要但实际不重要」的条目通过模糊推理通过四维测试。

反向检查将评估从「这值得保留吗？」（主观的、宽松的）转变为「没有这个什么会坏？」（客观的、严格的）。

5.3.4. 步骤 9：备份

```
mkdir -p memory/archive
cp MEMORY.md memory/archive/MEMORY.md.bak-$(date +%F)
```

在任何修改之前创建带时间戳的备份。备份文件名包含日期，提供每日粒度的回滚能力。

5.3.5. 步骤 10：带大小约束的更新

```
更新 MEMORY.md。硬性上限：80 行。
如果超限，先压缩/合并现有条目。
```

更新过程必须遵守 80 行硬性上限。如果添加新条目会超过限制，智能体必须先压缩或合并现有条目以腾出空间。

压缩策略包括：

- 将相关条目合并为单个更通用的陈述
- 移除已过时或不再相关的条目
- 当项目完成时合并项目特定的条目
- 通过更简洁的措辞提高现有条目的密度

5.4. 阶段 3：归档

归档阶段将已处理的每日日志移动到长期存储。

5.4.1. 步骤 11：文件移动

```
将已压缩到每周摘要的每日文件移动到 memory/archive/YYYY/
```

已成功压缩到每周摘要的每日日志文件被移动到 `memory/archive/YYYY/`，按年份组织。这保持了活跃的 `memory/` 目录整洁，同时保留原始数据以供未来可能的参考。

5.4.2. 步骤 12：索引更新与报告

```
运行 qmd update && qmd embed
如果有变更，发送简要摘要。
如果没有变更：回复「memory-tidy 完成，无变更。」
```

更新 `qmd` 索引以反映文件移动和新的每周摘要。与 `memory-sync` 不同，此任务通过 `--deliver` 标志向用户交付摘要，为维护过程提供透明度。

6. qmd 集成层

qmd 工具作为记忆系统的搜索与检索引擎，提供对记忆文件存储的混合搜索与重排序能力。

6.1. 在架构中的角色

Cron Job 负责 写入 记忆，而 qmd 负责 读取 记忆。它使 AI 智能体能够高效地跨所有记忆层搜索，无需读取整个文件：

操作	qmd 命令
语义搜索	`qmd query "<问题>"` — 带重排序的混合搜索
片段检索	`qmd get <文件>:<行号> -l 20` — 拉取特定行
索引更新	`qmd update` — 刷新文件索引
嵌入生成	`qmd embed` — 生成向量嵌入

6.2. AGENTS.md 集成

系统通过强制性的记忆检索指令扩展了 AGENTS.md：

```
### Memory Retrieval (MANDATORY)

When you need to recall past events, **search first — never read all files**:
1. `qmd query "<question>"` — hybrid search with reranking
2. `qmd get <file>:<line> -l 20` — pull only the snippet you need
3. Only fall back to reading files directly if qmd returns nothing
```

「先搜索，不要读取所有文件」的原则对性能至关重要。随着记忆文件累积，读取所有文件会消耗大量上下文窗口空间。qmd 的混合搜索（结合关键词匹配与通过向量嵌入的语义相似度）能够在不进行全文件扫描的情况下实现精确检索。

6.3. 混合搜索架构

qmd 实现了两阶段检索流水线：

1. **阶段 1 — 检索**：关键词搜索（BM25）和语义搜索（向量嵌入）并行执行，产生两个排序的候选列表。
2. **阶段 2 — 重排序**：候选列表被合并，并使用交叉编码器模型重新排序，该模型比任何单一方法都能更深入地考虑查询-文档相关性。

这种混合方法结合了关键词匹配的精确性（精确术语、文件名、日期）和语义搜索的召回率（概念相似性、改写查询）。

6.4. 实时写入补充

系统还指示智能体实时写入记忆，而非仅依赖 Cron 捕获：

```
### Memory Writing - Don't Wait for Cron

When you make a decision, the user says "I prefer X", or a key task
completes — **immediately append to `memory/YYYY-MM-DD.md`**.
Cron captures sessions automatically every few hours, but it's just
the safety net. Write it down in the moment.
```

这种双重写入策略确保重要信息被立即捕获（由活跃智能体），同时如果智能体遗忘或会话突然结束，也会被安全网（`memory-sync` Cron Job）捕获。

7. 文件系统布局

终极记忆系统在 OpenClaw 工作区下创建了组织良好的目录结构：

7.1. 完整目录树

```
~/.openclaw/workspace/
├── MEMORY.md                    # 第三层：长期记忆（上限 80 行）
├── AGENTS.md                    # 已更新记忆检索规则
└── memory/
    ├── 2026-02-17.md           # 第一层：今天的每日日志
    ├── 2026-02-16.md           # 第一层：昨天的每日日志
    ├── 2026-02-15.md           # 第一层：...
    ├── ...                     # （最多 7 天的活跃每日日志）
    ├── weekly/
    │   ├── 2026-02-03.md       # 第二层：2 月 3 日所在周的摘要
    │   ├── 2026-01-27.md       # 第二层：1 月 27 日所在周的摘要
    │   └── ...                 # （持续增长的每周归档）
    └── archive/
        ├── MEMORY.md.bak-2026-02-17  # MEMORY.md 备份
        ├── MEMORY.md.bak-2026-02-16  # 上一次备份
        └── 2026/
            ├── 2026-02-03.md   # 已归档的每日日志
            ├── 2026-02-04.md   # 已归档的每日日志
            └── ...             # （每周压缩后的每日日志）
```

7.2. 文件命名约定

所有基于日期的文件均使用 ISO 8601 格式 `YYYY-MM-DD`，以确保一致性和可排序性：

- 每日日志：`memory/YYYY-MM-DD.md`
- 每周摘要：`memory/weekly/YYYY-MM-DD.md`（周一日期）
- 备份：`memory/archive/MEMORY.md.bak-YYYY-MM-DD`
- 归档每日日志：`memory/archive/YYYY/YYYY-MM-DD.md`

7.3. MEMORY.md 模板

初始 `MEMORY.md` 模板建立了结构和约束：

```
# Long-Term Memory

> Only write info here that you'd make mistakes without.
> Event logs stay in daily files.
> Hard limit: 80 lines / 5KB. Must compress before adding when over limit.

## User Preferences

## Active Projects

## Key Decisions
```

```
## Important Contacts
```

四个分区提供语义组织：

- **User Preferences**：用户喜欢的工作方式（工具、风格、约定）
- **Active Projects**：当前活跃的项目上下文和架构
- **Key Decisions**：重要的技术或流程决策及其理由
- **Important Contacts**：用户经常提及的人物、团队或服务

8. 安全机制

终极记忆系统包含多重安全机制，以防止数据丢失、损坏和无限增长。

8.1. 幂等性保护

如第四章所述，Session ID 前缀检查确保没有会话被记录两次。这防止了：

- Cron 任务重试
- 运行间的时钟偏移
- 手动重新触发
- 并发捕获中的竞态条件

8.2. 大小约束

`MEMORY.md` 的 80 行 / 5 KB 硬性上限防止无限增长。在先行系统中，缺乏此限制的记忆文件已被观察到增长至数百或数千行，通过上下文窗口饱和降低了 AI 性能。

执行机制是行为性的而非技术性的：memory-tidy 智能体被指示在添加前先压缩，且约束在 `MEMORY.md` 头部醒目显示作为持续提醒。

8.3. 会话过滤

最低 2 条消息的阈值过滤掉了琐碎的会话。对典型 OpenClaw 使用模式的分析表明，有意义的会话（那些产生可操作知识的会话）几乎总是包含至少 2 条用户消息。

此过滤器防止了：

- 意外的会话启动（0 条消息）
- 快速的 API 检查或拼写纠正（1 条消息）
- 系统测试会话

8.4. 修改前备份

每次 `MEMORY.md` 修改之前都会自动备份。备份以带日期戳的文件名存储，提供每日粒度的回滚能力。

这防止了：

- 过于激进的压缩（删除后来证明重要的条目）
- 蒸馏过程中的智能体错误
- 写操作期间的文件损坏

8.5. 隔离会话执行

两个 Cron Job 都在 `--session isolated` 模式下运行，这：

- 防止干扰活跃的用户会话
- 确保 Cron Job 的上下文不会泄漏到用户对话中
- 防止 `memory-sync` 记录自身（通过步骤 2 的隔离过滤器）

8.6. 出处追踪

每一层的每条记录都包含出处信息：

- 每日日志：Session ID 前缀和时间戳
- 每周摘要：来源日期标签 (`src: YYYY-MM-DD`)
- MEMORY.md：条目可以通过 每周 → 每日 → 会话 的链条追溯

这条审计路径在记忆条目看起来不正确或过时时支持调试。

9. 验证与确认

原帖提供了一个四步验证流程，用于在部署后确认系统正确运行。

9.1. 步骤 1：Cron 注册确认

```
openclaw cron list
```

验证 memory-sync 和 memory-tidy 都出现在已注册的 Cron Job 列表中，且调度时间正确。

9.2. 步骤 2：捕获验证

与 OpenClaw 进行若干对话会话，然后等待下一个 memory-sync 窗口（北京时间 10:00、14:00、18:00 或 22:00）。窗口过后，检查每日日志是否已生成：

```
ls memory/2026-02-16.md  # 替换为当前日期
```

9.3. 步骤 3：搜索验证

测试 qmd 搜索功能：

```
qmd query "最近会话中的关键词"
```

应返回每日日志文件中的相关片段，确认捕获和索引流水线都在正常运行。

9.4. 步骤 4：每周压缩验证

运行一周后，检查每周摘要文件：

```
ls memory/weekly/
```

应至少存在一个以上一周周一命名的每周摘要文件。

10. 与 FlowCabal OpenViking 的对比

OpenClaw 终极记忆系统与 FlowCabal 中的 OpenViking 子系统存在有趣的相似性，尽管它们面向不同的使用场景。两者都解决了 AI 智能体的结构化知识持久化问题。

10.1. 架构对比

方面	**OpenClaw 记忆**	**FlowCabal OpenViking**
存储	扁平 Markdown 文件	SQLite 虚拟文件系统
层级	每日 → 每周 → MEMORY.md	`/meta`、`/entities`、`/manuscript`、`/summaries`
压缩	时间维度（7 天窗口）	多级摘要
搜索	`qmd` 混合搜索	递归上下文搜索
策展	长期记忆的 4D 验证	仅用户批准的输出
索引	通过 qmd 的向量嵌入	对策展输出的 OpenViking 索引
出处	来源日期标签	可追溯检索

10.2. 关键差异

范围：OpenClaw 记忆管理 智能体会话历史（关于用户及其偏好的元知识），而 OpenViking 管理创作内容（长篇写作场景中的手稿、实体、摘要）。

策展模型：OpenClaw 通过 4D 验证测试使用自动化策展，而 FlowCabal 使用人在回路的策展（仅用户批准的输出进入存储）。

存储后端：OpenClaw 使用扁平文件（简单、便携、可 grep），而 FlowCabal 使用 SQLite（可查询、事务性、有模式约束）。

两个系统共享一个根本性洞察：并非所有信息都值得持久化 — 激进的过滤对于维护长期知识库中的信号质量至关重要。

11. 局限性与未来工作

尽管终极记忆系统代表了对先行方案的重大进步，但仍然存在若干局限性和改进机会。

11.1. 当前局限性

11.1.1. 固定调度的不灵活性

四个固定的同步时间（10、14、18、22）可能不适合所有用户的工作模式。不同时区或非标准作息的用户可能错过捕获窗口或体验到次优的时机安排。

潜在改进：基于会话活动模式的自适应调度，或在每次会话结束后触发的事件驱动型触发器。

11.1.2. 单用户设计

系统面向个人使用设计。多用户场景（如团队共享开发环境）未被考虑。

潜在改进：在 `memory/` 目录中实现按用户命名空间隔离。

11.1.3. 无语义去重

幂等性检查使用 Session ID 前缀匹配，可以防止完全重复的会话，但无法检测跨会话的语义重复条目。如果同一决策在多个会话中被讨论，可能出现在多个每日日志中。

潜在改进：在蒸馏阶段使用语义相似度检查，利用 `qmd` 的向量嵌入检测近似重复条目。

11.1.4. 质量依赖于智能体

提取摘要的质量完全取决于 AI 智能体的理解力和判断力。糟糕的提取可能遗漏关键信息或包含噪音。

潜在改进：对每周摘要进行人工审核，或采用类似 engram 的信心评分方法。

11.1.5. 无冲突解决

如果 `MEMORY.md` 包含矛盾条目（如「用户偏好 Svelte」和「用户切换到了 React」），没有自动化机制来检测或解决冲突。

潜在改进：在蒸馏阶段进行矛盾检测。

11.1.6. 纯 Markdown 格式

所有记忆文件都是纯 Markdown，这限制了可以高效存储和查询的信息类型。结构化数据（如实体关系、版本号、日期）将受益于更结构化的格式。

潜在改进：混合格式，使用 YAML frontmatter 存放结构化元数据，Markdown 正文存放自由文本内容。

11.2. 潜在扩展

11.2.1. 跨项目记忆

目前，记忆的范围限于单个 OpenClaw 工作区。一个跨项目记忆层可以捕获适用于所有项目的通用用户偏好（如编码风格、工具偏好）。

11.2.2. 记忆可视化

一个基于 Web 的仪表板展示记忆增长、分类分布和时间模式，可以帮助用户理解和策展其 AI 智能体的知识库。

11.2.3. 主动遗忘

除了添加和压缩之外，主动遗忘机制可以主动移除与已完成项目、已弃用技术或已更改偏好相关的条目。

11.2.4. 与版本控制集成

将 `MEMORY.md` 纳入 Git 追踪可以实现：

- 记忆随时间演变的历史记录
- 回滚到任何先前的记忆状态
- 团队间共享已验证的记忆条目

12. 实施手册

本章提供逐步实施指南，将原始帖子中的所有配置步骤整合为单一参考。

12.1. 前置条件

要求	详情
OpenClaw	已安装并运行
qmd	已配置为记忆搜索后端
工作区	~/.openclaw/workspace/ 目录已存在
Cron 支持	OpenClaw cron 系统可操作

12.2. 逐步设置

12.2.1. 1. 创建目录结构

```
mkdir -p ~/.openclaw/workspace/memory/weekly
mkdir -p ~/.openclaw/workspace/memory/archive
```

12.2.2. 2. 创建 MEMORY.md

将以下模板写入 ~/.openclaw/workspace/MEMORY.md：

```
# Long-Term Memory

> Only write info here that you'd make mistakes without.
> Event logs stay in daily files.
> Hard limit: 80 lines / 5KB. Must compress before adding
> when over limit.

## User Preferences

## Active Projects

## Key Decisions

## Important Contacts
```

12.2.3. 3. 更新 AGENTS.md

在 AGENTS.md 的 ## Memory 部分添加记忆检索和写入规则。

12.2.4. 4. 注册 memory-sync Cron Job

执行 openclaw cron add 命令，使用完整的 memory-sync 提示（见第四章）。

12.2.5. 5. 注册 memory-tidy Cron Job

执行 `openclaw cron add` 命令，使用完整的 memory-tidy 提示（见第五章）。

12.2.6. 6. 验证安装

运行 `openclaw cron list` 并确认两个任务都已出现。

12.2.7. 7. 测试流水线

1. 开始一次有意义的对话（2 条以上消息）
2. 等待下一个同步窗口
3. 检查每日日志是否已创建
4. 测试 `qmd query` 搜索

13. 结论

OpenClaw 终极记忆系统是对 AI 辅助开发中最持久挑战之一——上下文失忆——的精心设计方案。通过时间压缩、激进过滤和自动化维护的组合，它为赋予 AI 编程智能体持久记忆提供了一种实用、低维护的方法。

该系统对 OpenClaw 记忆生态的核心贡献包括：

1. **幂等捕获**：基于 Session ID 的去重确保了可靠、可重复的数据摄取，不受 Cron 调度异常的影响。
2. **有界增长**：`MEMORY.md` 的 80 行硬性上限迫使持续策展，防止了困扰朴素方案的无限文件增长。
3. **四维验证**：多条件验证测试（错误预防、广泛适用、自包含、非重复）加上反向检查，为决定什么值得长期保留提供了严格的、有原则的框架。
4. **时间压缩流水线**：每日 → 每周 → 长期的层次结构提供了带有完整出处追踪的自然压缩，在近期日志的细节和长期记忆的密度之间取得平衡。
5. **运维简洁性**：两个 Cron Job、扁平 Markdown 文件和一个搜索工具（`qmd`）构成了整个系统——没有外部 API，没有云服务，没有复杂基础设施。

该系统在方案版图中占据了一个经过深思熟虑的位置：比 Calicastle 的极简方案更健壮，比 engram 的功能丰富但复杂的架构更简洁，且与依赖云端的方案不同，完全在本地运行。它证明了有效的 AI 记忆不需要复杂的基础设施——它需要的是有原则的信息管理。

正如作者所言，这是在一个春节晚上，一边看着一台他觉得「太难看了」的电视节目一边做出来的。或许这恰恰说明了系统的设计哲学：实用的方案不需要复杂，最好的架构是那些简单到足以在看着春晚的几个小时内完成设计的架构。

报告结束
由 Claude Code 生成 — 2026 年 2 月 17 日
来源：linux.do/t/topic/1621623
总页数：33