LAURENT DÉSÉCHALLIERS

上下文

构建一个让 AI 理解重点、
连接知识，并将所学
用于实现目标的上下文。

Le Contexte

> 如何构建一个能让 *AI* 理解什么最重要、连接你的知识，并将你所学沉淀下来以服务于目标的上下文。

Laurent Deséchalliers

翻译说明

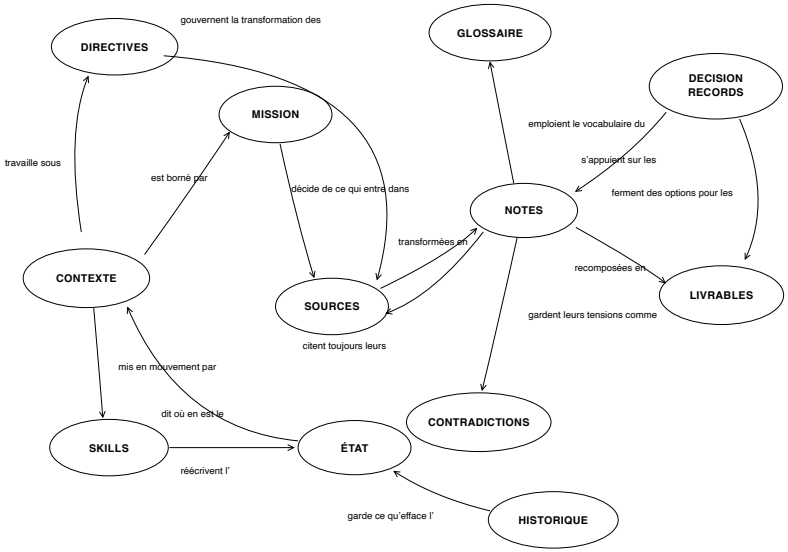
本版由人工智能根据法文原文自动翻译而成。

译文未经人工译者审校。书中的思想、结构和示例均忠实于原文，但有些表达可能不够自然，个别段落也可能留有源语言的痕迹。

为了让本书能以多种语言面世，而不只有法文版，这是我们愿意付出的代价。如对某处表述存疑，请以法文版为准。

上下文的定义

> 上下文是一个由人的使命划定边界的系统。在这个系统中，*AI* 会组织并调用我们知道什么、这些知识从何而来、各个要素如何关联，以及工作进展到了哪里，从而帮助人实现目标，而不必每次都从零开始。在这一过程中，系统会逐步建立一项长期的信息资产，并使其不断增值：已经产生的知识会被保留下来，随时间不断丰富，并可用于其他目标。



引言

我们从未拥有过如此丰富的信息。然而，我们仍要耗费大量精力去重新找到它们、整理它们，并重建几天前才刚刚理解的内容。

一个有用的想法，很少以整洁完整的形式出现。它可能萌生于一次交流，在一份文档中逐渐清晰，在会议上遇到质疑，促成一项决定，随后又变成一篇文章、一份规格说明或一个项目。我们的工具保存了每一道痕迹，却很难保留将它们串联起来的脉络。

人工智能改变了这个问题的一部分。它能够阅读、联系、改写和转化形态各异的材料。但与 AI 的一次对话依然十分脆弱：议题不断推进，对话越来越长，接下来又不得不重新说明目标、已经作出的决定、使用过的来源，以及尚未完成的工作。

正是为了解决这一困难，我构建了上下文。

一个上下文，是专门围绕某个主题建立的工作空间。人和 AI 在其中保存目标、来源、想法、规则和工作状态，以便持续产出成果，而不必在每次会话时把一切重新搭建一遍。

这样，AI 就能基于一段真实的历史开展工作。它可以找回一个想法的起点，提醒曾经的选择，指出矛盾，发现缺失的证据，并从草稿上次停下的地方继续。**它帮助组织思想，也帮助思想彼此检验。而意义与决策的责任始终由人承担。**

这个系统诞生于我的日常工作，起点只有一个文件，以及我向 AI 提出的一个极其简单的请求：整理我接下来要告诉它的内容。它在一次次尝试中发展起来，每遇到一个困难，就会暴露出一处缺失。它至今仍在演进：本书介绍的一些做法已经经受住了多种用途的检验，另一些则仍只是我为尚未解决的问题给出的临时答案。

因此，本书并非从一套已经完成的理论出发，而是从亲历的问题出发：五百封邮件变得无法追踪，想法散落在多个工具里，回复客户文档时又不能丢掉已经完成的工作，以及中断之后需要重新拾起的文章。

本书写给那些把信息转化为理解、决策和交付物的人：产品经理、顾问、作者、研究人员、分析师，以及更广泛的知识工作者。它提出了一种方法，让 AI 成为长期的工作伙伴，同时仍由人掌握自己的材料。

全书由相互补充的两册组成。**第一册——理解上下文** 讲述这个系统如何诞生，以及它怎样改变人与 AI 的关系。**第二册——构建并持续运用上下文** 说明如何建立它，并长期使用它。

两册既可顺序阅读，也可分别阅读。**按什么顺序阅读本书**一章介绍了几种可选的阅读路径。

示例说明

本书中的思考和示例，来自我在信息技术行业三十多年的实践。它们取材于不同项目和职业环境中遇到的情形，也来自我与其他公司的产品经理、CTO、工程经理等人的大量交流：涉及多个组织，也跨越了我职业生涯的不同时期。

为服务于本书的论述，我对这些情形进行了修改、概括或重新组合。书中的用例、数字和指标，用来说明反复出现的问题以及我从中得到的认识。它们都是经过构造的：没有一个示例描述真实的客户、产品或组织，也不反映我雇主的数据、实践或立场。

取自我个人生活的示例——例如小区业主事务、我的文章和我自己的文字——则源于我的真实经历。不过，某些可能识别出具体人物的细节经过了修改。

本书表达的分析、方法和观点均属个人意见，只代表我本人。

关于作者

我在软件行业工作了几十年，如今担任产品经理。我的职业经历从开发和架构开始，后来转向团队组织，再到产品、产品运营和市场营销。

这些职业变化也改变了我与信息打交道的方式。仅仅掌握一种技术方案或找回一份文档，已经不够了。我需要汇集分散的信号，理解它们合在一起说明了什么，在多个方向之间作出取舍，并为截然不同的读者写出合适的文字。

我很早就开始关注人工智能和分布式系统。如今，这份兴趣遇到了一个非常具体的问题：**如何与 AI 协作，同时不把自己的信息、决定和思考交由它掌控？**

我曾为专业媒体撰稿，也曾为 M2 Édition 和 Eyrolles 担任技术审校。现在，我在 malorean.net 上撰写有关产品管理、AI、工作方法和信息系统的文章。

我也为独立杂志 *Proposition Marketing* 供稿。这本杂志关注战略与运营营销。为来自市场营销和企业管理领域的读者写作，迫使我在解释产品问题时不能假定大家拥有共同的专业词汇。本书也遵循同样的要求。

我非常愿意与其他构建相近、不同或互补系统的人交流：这个主题值得讨论、碰撞，也值得借助其他实践不断丰富。我也希望与研究相近问题的学者交流——我曾在法国国家科学研究中心（CNRS）下属的两个研究单位工作过几年，若能重新与这个领域建立联系，我会很高兴。

Laurent Deséchalliers

- laurent@malorean.net
- https://malorean.net

致谢

我首先要感谢的是你。

是你冒出了一个略显疯狂的念头，翻开了这本书——这本书比预想的厚了一些，可能还有些重复，谈的又是一个相当晦涩的主题。

直到现在，我对上下文的研究大多是独自进行的。但我希望这种孤独不会持续下去：希望这个主题能吸引其他人来探索、讨论、批评，并推动它继续演进。也许，到那时，malorean.net 上接下来发表的文章，就会让我有机会感谢许多参与其中的人。

我尤其要感谢我的妻子 Christine。

她看着我为了这本书，一个个夜晚和周末都坐在电脑前。她在整个写作过程中一直支持我，同时也总能提醒我：已经很晚了，真的该把这台该死的电脑关掉了。

我保证，这只是一段插曲。

而且，远不止为了这本书，也谢谢你这么多年来一直陪伴着我，共同度过我们两人的美好人生。

按什么顺序阅读本书

本书按照事情在我身上发生的顺序展开：先是亲历的问题，再是临时拼凑的解决办法，最后才形成一个系统。因此，对上下文的完整说明要到第二册才会出现。

“理解”路径就是这个顺序。你从头读到尾，系统的每个组成部分，都会在催生它的缺失刚刚显现时出现。

“构建”路径面向那些已经经历过这个问题、想直接了解运作机制而非故事的人。

1. **上下文的定义**：用几行文字说明我们在谈论什么。
2. **第 5 部分——上下文的内部结构**：依次阅读其中六章，即全貌、打开、输入材料、转化、产出与决策、保存。
3. **第 6 部分——上下文所在的生态系统**：把你刚刚构建的内容与信息系统的其他部分连接起来。
4. **第一册**：当某个设计选择显得武断时，你会在这里找到促成它的问题。

这条捷径并非没有代价。第二册原本是为已经熟悉第一册各种情形的读者写的，也会引用其中的内容。因此，你会遇到一些略微缺少背景的示例——这就是代价；但如果你已经知道自己为何而来，这仍然是值得的。

阅读之前，先试一试

本书描述的是一种方法，而方法很难只靠讲述说清。你可以在阅读理论之前，或一边阅读一边，看它实际运行一次——这往往是理解某个部件用途的最佳时机。

github.com/laurent-malorean 上开放了三个代码库。

Contexte Nano Notion 安装最快。它既不需要开发环境，也不要求技术能力：一个 ChatGPT 加一个 Notion 就够了。ChatGPT 承担上下文的智能，Notion 则充当它的记忆——来源、笔记、决定和状态都写在那里，并能在下一次对话时重新读到。几分钟内，你就能拥有一个可以运作的上下文。

Contexte V2 是完整实现，也是我每天使用并用来撰写本书的版本。它可以通过 Claude Code 或 Codex 安装，任选其一；与前一个版本相比，需要多做一些设置。

LLM Wiki 不是上下文。它是一个从文档——书籍、标准、文章、课程——构建出来的知识库。AI 会从中提取简短且彼此关联的条目，之后便可用自然语言进行查询。上下文保留一项使命的脉络；LLM Wiki 则大规模保留我们读过的内容。第 6 部分会说明两者如何协同工作。

你完全不必从这里开始。不过，如果阅读在某个时刻让你觉得过于抽象，这是让它重新变得具体的最短路径。

同一作者的其他作品

Malorean Corp — 过去，把一个想法变成原型有时需要数周。如今，一个人在人工智能的帮助下，几天内就能做到。这一变化令人瞩目。但真正的巨变也许发生在别处：发生在我们共同工作的方式之中。

预计出版：*2026* 年 *11* 月底

更多信息请访问 https://malorean.net.

目录

卷 I

理解上下文

部分 1 – 一个需求的诞生 17
1.1 五百封邮件与一段消失的历史 18
1.2 第一个定义 24

部分 2 – 上下文改变了什么 29
2.1 从对话到可恢复的工作 30
2.2 两位客户、一份白皮书，以及一个需要保留的矛盾 35

部分 3 – 三种具体用途 41
3.1 回复招标文件，同时不丢失已经完成的工作 42
3.2 从尚未成形的思考出发写一篇文章 46
3.3 在混乱中让产品议题逐渐成熟 51

部分 4 – 交付物离开后留下了什么 57
4.1 单元格容得下答案，却容不下推理 58
4.2 六个月后，答案已经不够用了 61

卷 II

构建上下文，让它保持活力

部分 5 – 上下文的内部结构 67
5.1 完整地图 68
5.2 打开：找回使命与现状 77
5.3 输入：让材料进入 84
5.4 转化：从来源走向想法 89
5.5 产出与决策 95
5.6 保存与恢复工作 101

部分 6 – 生态系统中的上下文 107
6.1 查询结构化知识 108
6.2 查询真实产品 113
6.3 我拥有三个上下文的那一天 119
6.4 去邻居家阅读，但不搬进去 123
6.5 知识流动的代价 128

合上书之前

收尾 133
7.1 后记 134
7.2 概念渊源与相邻方法 136

卷 I

理解上下文

1 一个需求的诞生
2 上下文改变了什么
3 三种具体用途
4 交付物离开后留下了什么

部分 1

一个需求的诞生

上下文并非诞生于某个关于人工智能的构想。 它源于一个已经无法使用的邮箱，以及一个花了很久才逐渐成形的认识：问题并不在于邮件的数量。

这一部分会讲述它的起源，并由此给出第一个定义——足够简单，便于记住；也足够完整，可以作为阅读本书其余部分时的参照。

1.1

五百封邮件与一段消失的历史

第一次创建上下文时，我并没有打算发明一种与人工智能协作的方法。我只是想不再丢失自己小区业主事务的来龙去脉。

我是业主委员会的成员。因此，我会收到物业管理方、业主、租户和服务商发来的消息：

- 一封邮件可能宣布一个新问题，
- 下一封补充某份报价的细节，
- 第三封又推迟一次上门施工，
- 一周后，有人在同一个邮件串里回复，却谈起另一栋楼。

与此同时，另外三个议题又有了进展。

问题并不只在于数量。大约五百封邮件分散在十五个左右的议题中：

- 工程，
- 财务，
- 保险，
- 业主大会，
- 欠款人，
- 招标，
- 各类事故与请求
- ……

每个议题都按自己的节奏推进，有时会持续数月。

每天跟进一项工程的人，自然会把它的整个历程记在脑中。他知道哪家服务商来过，为什么日程发生了变化，还有哪项决定仍在等待答复。而我打开一封新邮件时，并不总能拥有这种连续的认识。我必须沿着邮件串往回翻，找出之前的交流，重建当时的情形，然后才谈得上理解新信息。

我希望能提出一些简单的问题：

- 屋顶工程进展到哪里了？

- 哪家服务商最近提交了檐沟的报价?
- 关于这处渗水，我们作出了什么决定?
- 物业管理方还有什么行动尚未完成?

我的邮箱里有这些答案。 但它无法给我全貌。

一个文件和一句话

我从最简单的办法开始：一个文件，再给 AI 一条指令。

> 汇总信息，并按主题拆分。

我把邮件交给它。 它把邮件归入不同议题，并为每个议题维护一份摘要。

- 工程相关的信息被放在一起。
- 财务问题形成另一组。
- 服务商、保险和业主大会也逐渐各归其位。

第一次尝试已经带来了一些帮助。我可以直接询问某个议题进展如何，而不必重读所有相关邮件。AI 不再只是在邮箱中寻找某个词句，而是向我呈现它对当前情形的理解。

随着材料不断进入，文件也在重新组织自己。

- 当一个新议题变得重要时，就出现一个新栏目。
- 当两个栏目开始混入不同的事情时，它们便彼此分开。

信息不再只是不断堆积。 它开始形成一种可以拿来工作的形态。

随后，这个最初的系统给出了一个正确却不够用的答案。

工程结束了，它的历史却消失了

一项工程结束时，摘要会写着：“工程已完成，验收合格”。

这确实是它的当前状态。 但这也意味着一种损失。

- 工程是什么时候开始的?
- 它在什么时候出现了延期?
- 日程为什么改变?

- 补充协议是什么时候签署的?

这些阶段都曾经十分重要。 最后一次更新却用最终状态把它们抹去了。

我要求的是一份摘要。 AI 用新状态替换旧状态，让摘要保持最新。 它遵守了指令，但这条指令并没有保护历史。

因此，我需要同一议题的两种视图。

- 第一种应当迅速回答“我们现在进展到哪里？”
- 第二种则应按顺序保存事件：工程开始、延期、达成新协议、上门施工、验收。
- 当前状态帮助我们采取行动。
- 历史则帮助我们理解如何走到当前状态，并在存疑时核实某一点。

这种区分源于一个非常具体的缺陷。起初，我只把它看成一种整理方式上的改进。 后来，它却成为我与 AI 协作时最重要的原则之一：**现在为真的事情，与曾经发生的事情，发挥的功能并不相同**。

AI 无法猜出我的规则

另一个问题出现在姓名上：

- 两家服务商的负责人拥有相同的姓氏。
  - 要区分他们，必须理解邮件讨论的议题。
- 一位女业主有时使用婚前姓氏，有时使用婚后姓氏。

如果没有特别说明，AI 可能把同一个人当成两个人。

还有一些差异来自我们的工作方式。

- 欠款低于一百欧元的人，不会按照其他欠款人的方式跟进。
- AI 不可能从任何普遍规则中知道这条界线。
- 它源于我们自己的工作方式。

我开始把这些知识明确写成长期有效的规则，完整地记录在文件中：

```
# 处理规则

- 在屋顶议题中提到的 Martin 是屋面施工方；在
  绿化议题中提到的 Martin 是园艺服务商。
- Dupont-Lemoine 与 Lemoine 是同一位女业主，房号 14。
- 将欠款低于一百欧元的人列在单独的清单中。
```


没有什么更复杂的东西。 只有三句话。原本每次对话时我都得再说一遍，但此后一次也不必重复。

每条规则都来自一次实际出现的错误或歧义。它捕捉到一种原本只存在于我脑中，而 AI 无法可靠重建的知识。

系统由此开始保存两种不同的材料：

- 关于小区业主事务的信息，
- 我希望如何处理这些信息。

原本提供帮助的文件开始造成妨碍

几个星期过去，文件越来越长。 它先是超过五百行，接着八百行，然后一千行。

这种增长恰恰证明了它有用：

- 各个议题得到了更好的跟进，
- 服务商也都对得上号，
- 作出的决定也能重新找到。

但最初让系统成为可能的东西，如今开始限制它的使用。

- 太多不同的信息挤在同一个空间里。
- 规则与事实混在一起。
- 财务、工程、人员、保险和招标彼此相邻，互相夹杂。

找到正确的材料，又一次变成需要费力的事。

我首先把处理规则分离出来。它们并不描述小区业主事务本身，而是说明应当如何处理相关信息。 将规则独立出来，更容易找到并应用它们。

接着，我把材料分配到多个主题之下。工程拥有了自己的空间，财务、服务商、业主大会决定和保险也是如此。一份总览说明每个空间包含什么。这样，AI 就能打开与问题有关的主题，而不必每次都遍历全部材料。

最后，我汇集了其中的专用词汇：法律术语、缩写和小区业主事务特有的表达。目的不是构建一部百科全书。而是让那些关系到交流能否被理解的词语，拥有稳定的含义。

当时，我脑中仍没有什么宏大的架构。只是每当一个问题让某个部件变得必要时，我就把它加进来。

在不失去来源的前提下信任

AI 有时会犯错。它可能把两条并非讨论同一议题的信息联系起来，也可能在两个同姓者中选错服务商。因此，系统必须加快理解，却不能成为唯一可用的事实版本。

我保留了所有原始邮件。当某个答案需要核实，或者事情变得重要时，我可以回到收到的原始材料。摘要提供快速入口。来源则让人能够核查。

正是这个选择，让系统真正可用。我可以容忍邮件整理中偶尔出现偏差，因为原始材料始终可以找到。同样的信任程度，并不适用于银行文件、合规义务或不可逆转的决定。风险始终取决于我们把什么交给系统，也取决于内容是否能够核实。

因此，上下文并没有取代我的邮箱。它把邮箱中的内容转化为一种工作记忆，同时仍为我保留了返回原始邮件的入口。

已经显现出来的东西

短短几周内，最初的起点——一个文件和一句话——已经变成一套能够满足多种需要的整体：显示当前状态、保存历史、应用小区业主事务特有的规则、区分议题、稳定词汇含义，以及返回来源。

收到新邮件时，我可以关注它改变了什么，而不必重建之前发生的一切。出现问题时，我可以用问题本身的语言直接提问，而不必在正确的邮件串中寻找正确的消息。AI 出错时，我可以修正规则，或回到原文。

我并不是一次性设计出整个上下文的。我是在一连串不足中，看着它逐渐成形：摘要抹去了历史，姓名彼此混淆，规则会丢失，文件变得过大。每一次修正，都让工作更持久，也更容易恢复。

这个起点很重要。上下文并非源于把内容分得更细的愿望。它诞生于这样的需要：理解不断变化的材料，保留能够核实材料的依据，并在不从零开始的情况下继续工作。

要点——同一议题的两种视图

状态回答“我们现在进展到哪里？”。历史回答“我们是怎样走到这里的？”。一份通过覆盖旧内容来保持最新的摘要，会保留前者，却丢失后者。两者都不可缺少：一种用来行动，另一种用来理解和核实。

1.2

第一个定义

从五百封邮件中诞生的系统，现在可以得到它的第一个定义了。

> 上下文是一个围绕某个主题建立的工作空间，人和 *AI* 在其中保存那些能让他们长期理解、决策和产出的内容。

这个定义有意保持简单。本书所描述的上下文拥有明确的组织方式，将在后半部分详细介绍。 要理解它能带来什么，开始时只需把握六个要点：

- 一个目标，
- 一批材料，
- 一些想法，
- 工作状态，
- 这项工作特有的规则，
- 需要产出的结果。

一个主题和一个目标

小区业主事务的上下文并不试图容纳我知道的一切。它有一个目标：理解正在进行的议题，保存它们的历史，并找回我在业主委员会采取行动所需的信息。

这个目标划出了一道边界。一封关于屋顶的邮件属于这里。一篇虽有意思却与小区业主事务无关的文章，则不属于这里。这条界线让系统不至于变成一个储藏室，把所有看似有趣的信息都堆进来。

因此，一个上下文始于一项意图。 它可以用来：

- 写一本书，
- 探索一个产品问题，
- 为一项决定做准备，
- 回复招标文件，
- 跟踪一个业务实体，

- ……

保留出处的材料

材料通过两扇门进入上下文。

第一扇门接收工作会话开始前已经存在的内容：

- 一封邮件，
- 一篇文章，
- 一本书，
- 一个网页，
- 一份访谈文字稿
- ……

它们是**文档来源**。这些内容进入时已经是成形的对象，并保持稳定。在小区业主事务的案例中，那封通知上门施工延期的邮件就属于第一类。

第二扇门接收与 AI 协作过程中逐渐表达出来的内容。当我说明一种情形、纠正一种理解、补充一个细微差别，或说出一个源于亲身经验的判断时，这些话就成为**会话来源**。它会随着交流逐步形成。当我说明两个姓名指的是同一位女业主时，我的话首先从第二扇门进入。

因此，两者的差别在于材料出现的时间和方式。

- 文档在会话前已经存在，随后被带入会话。
- 会话来源在会话中诞生，并随会话一起增长。

之后，这两类来源承担同一种功能：在解释之前保存材料。

- 它们并不保证其中一切都是真的。
- 它们让我们能够找回收到或表达过的内容，再判断其可靠性。

随后，上下文可以从这些材料中提取一个想法、一条规则或一种工作状态。这些经过转化的知识被放在另一个位置。 这样，如果某个日期似乎有误、两个人被搞混了，或某种解释变得值得质疑，我们就能回到它的出处。

让想法变得可用

仅仅积累来源还不够。五百封保存完好的邮件，仍然是五百封需要重读的邮件。

因此，上下文会逐渐转化这些材料。 它揭示出下列内容：

- 议题，
- 重要事实，
- 规则，
- 区分，
- 尚未解决的问题，
- 关联。

一段孤立的交流被纳入它所属工程的历史。 对同一家服务商的多次提及，共同丰富了相关知识。 一次纠正，则防止同一种混淆再次发生。

这个中间阶段之所以重要，是因为它让信息变得可用。

- 收到的材料保留原始痕迹。
- 组织好的知识让人能够理解并行动。

在更复杂的工作中，有些想法可以反复使用：同一份产品知识可以支持一项决定、一份规格说明和一份文档；同一个经过验证的事实，可以帮助回答多份客户问卷。 结果会变化，而想法及其出处仍然可用。

一种让工作可以继续的状态

一项仍在进行的工作，总有它的现在：已经知道什么、什么仍不确定、作出了哪些决定，以及接下来要发生什么。

在小区业主事务中，状态回答的是一个眼前的问题：这项工程今天进展到哪里？它的历史则回答另一个问题：我们是怎样走到这里的？系统必须同时拥有两者，工作才能真正接续。

这种区分并不只适用于工程跟踪。当我中断一篇文章的写作，或暂停对一个产品问题的探索时，我希望找回的不只是一批文档。 我想知道：

- 哪些内容已经确定，
- 哪一点仍然薄弱，
- 什么行动能让工作重新启动。

上下文在对话之外保存这种连续性。

- 与 AI 的会话可以结束。

- 工作仍保有一种状态，下一次会话可以由此继续。

这项工作特有的规则

AI 拥有一般知识。 但它不知道大量让具体情形变得独特的内容：

- 两家服务商的人拥有同一个姓氏，
- 一个团队如何界定客户信号，
- 发表一项论断前需要达到怎样的证据水平
- 一本书选择了什么语气，
- ……

上下文把这些规则与受其约束的工作保存在一起。 当经验显露出规则的局限时，可以修正它们。 此后，规则会持续生效，而人不必在每个请求中重新说明。

这些规则并没有把工作的责任交给 AI。它们只是明确表达了人对 AI 的期待。 人确定目标，提供或验证材料，纠正解释，并作出决定。 AI 则在这个框架内吸收、联系、组织、质疑和改写材料，并准备产出结果。

适应需要的结果

上下文始终以它要帮助完成的事情为导向。

在小区业主事务中，结果可能是：

- 对一个问题的回答，
- 一份工程进展，
- 为一次讨论做准备。

在其他用途中，它可能是：

- 一篇文章，
- 一份招标文件，
- 一份文档。

同一个空间可以产出多种结果，因为它会在材料形成最终形态之前保存它们。来源不会被封闭在第一次使用它们的文档里。想法仍可用于另一个问题或另一类读者。

这种能力让上下文区别于简单的档案。

- 档案让人能够保存。
- 上下文则组织已保存内容的接续与转化。

一种可以非常简单地开始的形态

在我的实践中，上下文由一组文件组成。无论借助什么工具处理它们，我都可以独立阅读、移动和保存这些文件。这种形态满足了我对掌控力和持久性的需要。 只要能保留同样的功能，也可以有其他实现方式。

然而，第一个上下文并没有任何复杂之处。 它只存在于一个文件里。 它是在自身局限逐渐显现后才成长起来的。这个过程仍然很重要：你不必先构建出本书描述的整套系统，才能开始体会这一想法。

一个主题、一个目标和第一批材料就足够了。

- AI 帮助组织它们。
- 人观察还缺什么，添加一条规则，保存一个来源，或把发挥不同作用的两条信息分开。
- 架构随着遇到的问题逐步显现。

因此，上下文所指的，与其说是一种特定的目录树，不如说是人、AI 和一批朝向某个目标的材料之间，一种持久的工作关系。

下一部分将探讨这种关系改变了什么。 与 AI 的一次对话，本来就可以富有成效。而当这场对话进入一个能在当下之外继续保存目标、历史和状态的环境时，它便开始转变为上下文。

要点——上下文的六个要点

一个划出边界的目标；一批保留出处的材料；一些变得可用的想法；一种让工作可以继续的状态；这项工作特有的规则；需要产出的结果。开始时，把握这六点就够了。 架构会随后出现——当遇到的问题让它变得必要时。

部分 2

上下文改变了什么

仅仅与 AI 对话，就已经能产生很多东西。 它会解释、改写、提出建议，而且给出的答案往往比我们独自找到的更好。

因此，这一部分要探讨的是上下文在此基础上又增加了什么。答案不在于获得的回复质量，而在于两次会话之间有什么得以延续：一项能够从上次停下之处重新开始的工作，以及一些保留彼此分歧、而不是将分歧磨平的文档。

2.1

从对话到可恢复的工作

与 AI 的一次对话，可以非常简单地开始。

我给它一个还很模糊的想法。 它换一种方式表达。 我让它给出一个提纲，然后再来一个。它提出一条我没有想到的反对意见。我进一步说明自己的意图，否定一个方向，再深入另一个方向。 短短几分钟，这场讨论就推动了原本可能在空白页面前停滞很久的工作。

这是聊天的一大优势：只要一个问题，就能开始。这种形式很适合短任务。我让它做一份摘要、给出一种解释或列出几种可能。答案出现，我拿来使用，然后对话就可以结束。

当工作持续更久时，这种关系便发生了变化。

写作之前，先说五句话

为了继续上周开始的一篇文章，我重新打开六天前的一场对话。还没写下任何正文，我先输入了这些内容：

> 继续写那篇关于作业报告质量的文章。 读者是运营负责人，不是开发人员。 我们已经排除了缩短表单的想法，现在还不到作出结论的时候。有关审计的数字只来自一次访谈，请把它视为假设。我们上次写到了第三节。

五句话，只为了让 AI 重新进入六天前就已经摆在它面前的工作。

我刚输入的一切，在输入之前就已经存在： 角度、读者、决定、薄弱之处以及文章写到的位置，全都在上一场对话里。我只是凭记忆把它们重新找了出来，因为对话是按说话的先后顺序存放这些内容的。

而工作遵循的是另一种组织方式。

- 一开始给出的要求，到最后仍然必须适用。
- 昨天加入的来源，应当能支持明天写出的段落。
- 过去作出的决定，应当避免同一场讨论再次开启。

- 一个待处理的问题，应当在合适的时候重新出现。

对话越长，人就越需要在脑中维持这种组织。新的会话开始时，人要为 AI 重新交代来龙去脉：这是目标，这是发生的变化，这是语气，这是已经作出的决定，这是我们上次停下的位置。

聊天依然有用。 但维持连续性的负担，落在了使用它的人身上。

为需要延续的内容提供一个位置

上下文为工作提供一个区别于对话的空间，从而改变了这种关系。

讨论仍然是交流发生的地方。我们在其中提问、探索、澄清和纠正。而需要延续的内容，则进入工作空间：

- 目标，
- 来源，
- 规则，
- 已经稳定下来的想法，
- 决定，
- 文本，
- 工作进展。

这种差别看起来可能很小。 它就像一场会议与会议所讨论的项目之间的差别。一场好会议可以推动项目，但整个项目并不会只存在于会议的记忆里。 会议结束后，文档、决定和行动仍然继续存在。

保存完整的聊天记录，并不能产生同样的效果。如果想找回某句话的原貌，它当然很有价值；但它遵循的是对话的形态：一个问题、一个回答、一次纠正、一段离题，然后再回到主题。要恢复工作，仍需重读这一连串内容，并判断此刻什么才重要。

上下文则会转化对话中的内容。

- 一次纠正会进入能够防止错误重现的位置。
- 一项决定仍以决定的身份清晰可见。
- 一个来源继续与它所支持的内容相连。
- 一篇文章以最新的完整版本存在。

这样，记忆便转向了行动。它不只为了保存过去。它也为下一次行动做好准备。

用一页找回三十二篇文本

当我重新开始撰写本书时，切身体会到了这种变化。

当时，书稿包含三十二篇文本。其中一些部分已经写好，另一些仍然空着。有一节因为所占篇幅过大，已经被拆成两节。 有两个段落在等待我审阅。还有一项论断没有任何亲身案例支撑，必须重写。

单独来看，这些内容都不难理解。但它们合在一起，便构成一种必须先重建、然后才能写出任何有用文字的情形。

我可以打开三十二篇文本，重读提纲，翻阅过去的对话，再寻找那些仍待处理的批注。这种考古会耗费时间，而且最后仍然会留下疑问：我是否忘记了那个本应阻止一次错误的要点？

但我选择打开这本书的上下文。它递给我一个工作交接文件。那天早上，其中的内容大致如下：

```
# 工作交接

## 当前进展
从提纲到结论，已经有三十二篇文本。
技术部分已经写完；最初的两个部分仍然过于理论化。

## 继续之前需要知道的事
目标读者初次接触这个主题：如果概念先于亲身案例出现，他们就会失去方向。
关于文档系统失败的论断，目前还没有任何个人经历作为依据。

## 下一步行动
围绕小区业主事务中五百封邮件的案例，重写开篇。

## 仍未解决的事项
有两个段落等待我审阅。
有几个来源已经进入系统，却尚未转化为笔记。
```

一页，不到一分钟就能读完。 我知道该做什么，而且没有打开那三十二篇文本中的任何一篇。

继续做，不等于恢复工作

继续做，是知道下一步行动是什么。恢复工作，是在采取行动之前，重新找回这项工作的意义。

任务清单可以告诉我某个来源需要处理。但它不会说明为什么这个来源应当优先、它应该为书中的哪一部分提供支撑，也不会告诉我它要帮助纠正什么风险。

要恢复工作，我需要一组更丰富的信息：

- 工作追求的目标；
- 工作实际进展到的位置；
- 仍在影响后续方向的决定；
- 不能遗忘的薄弱之处；
- 一个足够具体、能够重新启动工作的下一步行动。

工作交接文件也正是按这个顺序阅读的。“三十二篇文本”提供了一个尺度。“如果概念先于亲身案例出现，他们就会失去方向”指出了方向。下一步行动由这两点推导而来，并明确指出了应当用来开启全书的具体案例。

恢复工作时，重建的是一段推理，而不只是清单中的某个位置。

但它也不是要在我脑中重现上一次会话的每个细节。摘要太短，会抹去一个选择背后的原因。报告太长，又会推迟工作真正重新开始的时刻。

中断从不挑选时机

最初，我以为只有妥善收尾之后，才能恢复工作。会话结束时，状态会得到认真总结，下一步行动也会被记录下来。

实践很少遵循这样的剧本。

- 一场会议开始了。
- 电池没电了。
- 一天比预想中更早结束。
- 另一件急事取代了眼前的主题。

恢复工作所需的信息最有用的时刻，恰恰是我们没有为结束做准备的时候。

因此，上下文会在工作进行期间持续维护恢复工作的能力。决定在作出时归位，任务在提出时归位，来源在进入时归位。

于是，恢复工作成为系统的一种属性，而不再要求使用者拥有英雄般的自律。

尽管如此，人依然承担决定性的角色。他可以判断，系统建议的下一步行动已不再优先，过去的某项决定需要重新讨论，或一项新事实改变了目标。记忆不会把路线冻结。当我们决定改道时，它让我们看清自己是从哪条路线出发的。

过去每次重新打开对话时都要再次输入的五句话，如今我不必再输入。它们写在某个地方，每次会话开始时，AI 都会替我重读一遍。

要点——聊天、环境与工作交接

聊天是交流发生的地方；上下文是需要延续之物所在的地方。 对话按照信息被说出的顺序保存信息；工作需要另一种顺序。继续做，是知道下一步行动；恢复工作，是在行动前重新找回工作的意义。中断不会预先通知：恢复工作的能力要在工作过程中维护，而不是等到关闭工作时才准备。

2.2

两位客户、一份白皮书，以及一个需要保留的矛盾

到目前为止，我介绍了上下文，也说明了它会产生的几种效果。但真正打开它时，它到底是什么样子？文件中能看到什么？多个来源如何揭示出一些分别看来都正确、却又难以调和的想法？

这个虚构上下文的主题很简单：在决定产品是否需要改变之前，先弄清为什么有些技术人员提交的作业报告并不完整。

它的使命只有短短几行：

> 了解哪些因素降低了作业报告的质量。区分填写困难、组织限制和数据运用需求。在对照多个来源之前，不要选择解决方案。

这项使命为工作指出方向。 它要求我们理解问题，而不是立即设计功能。

第一条反馈带来一个想法

加入上下文的第一份文档，是对一位运营负责人的访谈文字稿。

她解释道：

> “技术人员经常到星期五才填写作业报告。在现场，他们觉得表单太长。到了周末，他们已经忘记了部分细节。”

AI 将这次访谈保存为一个来源。随后，它把其中的一个想法提取到单独的条目里，也就是我所说的原子笔记：

> 过于繁琐的填写要求，可能使一些技术人员推迟完成作业报告，并因此丢失细节。

这种表述并没有说所有技术人员都会这样做，也没有说表单长度是唯一原因。 它保留了证词的核心：在所描述的组织中，填写工作被推迟，而一部分信息在现场作业与记录之间消失了。

来源仍然保留。笔记并不取代来源；它只是让主要想法更容易被找回，也更容易与后续内容对照。

第二条反馈丰富了同一则笔记

几周后，第二次访谈进入上下文。

这一次，另一位运营负责人说：

> “为了提高数据质量，我们把十二个字段设成了必填项。现在，有些技术人员会写‘无异常’，或者只填一个短横线，目的只是通过验证。”

这段证词描述的并非完全相同的行为。第一次访谈中的技术人员会推迟填写。 第二次访谈中的技术人员会填写表单，但会设法规避要求。

不过，两个来源都指向同一个核心想法：一项原本为了获取更多信息而设置的约束，最终可能降低实际录入内容的质量。

因此，第二次访谈加入了第一则**原子笔记**。它没有取代第一个来源，也没有形成一个相互竞争的想法。它通过揭示同一问题的另一种形态，丰富了这则笔记：先是延迟填写，现在又出现了表面上的完整。

笔记现在可以这样表述：

> 过于繁琐的填写要求，可能使一些技术人员推迟或设法规避作业报告的填写，损害所收集的信息。

其中引用了两个来源。每个来源都保有自己的作用和局限。笔记变得更加扎实，却没有声称两份证词足以衡量这一现象的规模。

白皮书带来另一个想法

接下来，我加入了一份讨论现场作业数据运用的白皮书。

这份文档提出了另一种要求。要理解反复发生的故障、比较设备并为某些审计做准备，企业需要结构足够清晰的作业报告。如果每位技术人员都在自由文本区随意填写——或者什么也不填——录入或许变得更容易了，但数据会失去很大一部分集体使用价值。

这个来源并没有削弱之前证词的真实性。其中描述的技术人员仍然会推迟或规避填写。 这份 PDF 揭示的是另一种现实：取消所有约束同样会付出代价。

AI 可能会想把这些内容合并进第一则笔记，再把它改写成一种普遍的折中方案。 这样做是错误的。 白皮书并不是在为访谈中观察到的事实补充限定；它提出的是另一个命题。

因此，它产生了第二则**原子笔记**：

> 要比较现场作业并运用相关数据，作业报告必须采用结构化形式。

现在，上下文中包含两个不同的想法：

- 填写约束可能降低实际收集到的信息质量；
- 缺乏结构可能降低这些信息日后的利用价值。

两则笔记可以同时为真。 然而，它们却把产品决策拉向相反的方向。

笔记之间的对立必须保持可见

上下文的作用，不是不断修改第一则笔记，直到矛盾消失。它必须保留两个命题，并明确呈现它们之间的关系。

这时，一则**主题笔记**可以说清：把两者放在一起，我们能理解到什么：

> 作业报告的质量，取决于填写时所需投入与后续运用所需结构之间的张力。

这第三则笔记不是各打五十大板式的折中。 它不取代任何一则原子笔记。它显示的是，把两者放在一起阅读时出现的新问题。

这个矛盾也进入工作状态：如何在不导致推迟或规避填写的情况下，获得结构化的信息？ 三个来源都还无法回答这个问题。

屏幕上呈现的内容

这个示例所用的上下文，以一个小型文件夹的形式存在。它的结构并不等于概念本身：这只是我目前赋予它具体形态的方式。

```
contexte-qualite-comptes-rendus/
├── 00-frame/
│   ├── mission.md
│   └── directives.md
├── 01-state/
│   ├── state.md
│   └── contradictions.md
├── 03-sources/
│   ├── SRC00001-entretien-exploitation.md
│   ├── SRC00002-entretien-exploitation.md
│   └── SRC00003-livre-blanc-donnees-intervention.md
├── 04-notes/
│   ├── atomic/
│   │   ├── ATO00001-la-contrainte-de-saisie.md
│   │   └── ATO00002-des-comptes-rendus-structures.md
│   └── thematic/
│       └── THM00001-la-qualite-depend-de-la-saisie.md
└── 05-outputs/
    └── questions-prochain-entretien.md
```

现在，你可以把本书从一开始使用的词语，与一些看得见的对象联系起来。

- **使命**说明这个文件夹为何存在。
- **来源**保存解释之前收到的内容。
- **原子笔记**让彼此不同的想法稳定下来。
- **主题笔记**让想法之间的张力变得可以理解。
- **状态**说明这种张力留下了哪些尚未解决的问题。
- **交付物**为下一步行动做好准备。

这一切都不需要什么神秘的软件。在这里，VS Code（也可以是 Codex 应用或 Claude Cowork）只是我访问上下文的窗口。上下文本身存在于文件、文件发挥的功能，以及文件之间保留下来的关系之中。

反馈变得贴合具体情形

几天后，我重新开始工作，向 AI 提议：

> “也许我们应该把所有字段都设为选填。”

在孤立的对话中，AI 可能会为这个想法辩护，提出更轻量的界面，或列出它的种种优点。而在上下文中，它会遇到两则原子笔记以及连接它们的张力。于是，它可以回答：

> 这项建议很可能会减少访谈中描述的部分阻力。不过，它与白皮书指出的结构化数据需求相冲突。来源显示，两方面的约束都真实存在；但它们还不足以让我们在简化表单、渐进式填写、语音输入，或根据现场作业类型调整字段之间作出选择。

这种反馈并非来自产品管理的一般知识。它植根于正在进行的工作。AI 知道我们正试图理解什么问题、哪些想法由哪些来源支持，以及哪个矛盾仍未解决。

它不会替我作决定。 它只是防止我新产生的直觉抹去一种已经确立的现实。

第一份交付物为下一个来源做好准备

上下文的状态并不会写着：“设计一份新表单”。它把看似已经确定的内容与仍不确定的内容分开。

> **看似已经确定的内容** 填写约束可能导致延迟和规避。要运用作业报告，其中仍须保留足够的结构。
>
> **仍未解决的内容** 我们还不知道哪一种阻力影响最大，也不知道哪些信息必须采用结构化形式。
>
> **下一步行动** 准备一次访谈，用来区分问题的成因，而不是立即提出解决方案。

因此，第一份交付物很朴素：一份用于新访谈的问题指南。它要帮助我们理解填写发生的时机、现场的限制、不同字段的实际用途，以及规避填写的各种方式。

这份交付物并不解决矛盾。 它只是为将来或许能够解决矛盾的材料做好准备。

一个工作对象，而不是把一切融合起来的机器

这个示例也展示了一种 AI 可能犯下的错误：因为议题相似，便想把新来源塞进已有的笔记。

当多个来源讨论同一个命题时，一则**原子笔记**可以继续增长。两次访谈就是这种情况。但白皮书并不是支持这一命题的新证据。它揭示出一个不同的想法，值得拥有自己的笔记。

这种区分很重要。如果每次出现对立都重写之前的笔记，“想法的世界”最终就会磨平所有张力。 知识看似自洽，却不再对应来源真正带来的内容。

上下文无法保证 AI 总能选对边界。人必须能够拒绝合并，要求建立新笔记，并核查笔记之间建立的关系。结构让这种修正成为可能，也让它能够长期保留；但结构不会自动作出决定。

现在，你已经看到了这个对象：

- 保持原样的**来源**，
- 各自保有核心的**原子笔记**，
- 将想法联系起来却不加以合并的**主题笔记**，
- 一个仍然开放的**矛盾**，以及一份为后续工作做准备的**交付物**。

接下来的章节会重新讨论上下文的用途，然后以这些已经变得可见的要素为基础，更具体地打开它的架构。

要点——保留矛盾

两个来源可以同时为真，却把决定拉向相反方向。 当一个来源加强同一命题时，原子笔记会继续增长；不同的命题则应拥有自己的笔记。为了让张力消失而强行合并，只会产生一种看似自洽、却不再对应来源的知识。

部分 3

三种具体用途

前两个部分说明了上下文改变了什么。 这一部分则通过三种我确实在实践的用途，展示它如何工作。

回复一份招标文件；从尚未成形的思考出发写一篇文章；在支离破碎的一天里，让一个产品议题逐渐成熟。 三种时长，三类材料，三种读者——而每次都遵循同一个过程：找回已经理解的内容，加以转化，再产出成果，同时不耗尽支撑这份成果的材料。

3.1

回复招标文件，同时不丢失已经完成的工作

我用这个系统处理的第一份招标文件有四十三个问题。

我们需要回答有关产品架构、安全性、备份、服务连续性和数据保护的问题。 一部分信息存在于我们的文档中。 另一些深藏在配置文件里。 还有一些必须询问负责服务运营的团队。

我们不得不汇集七个技术来源，并与团队往返确认一次。这些工作还不包括真正的写作：理解潜在客户在问什么，找回信息，核实它是否仍然准确，决定哪些内容可以对外传达，最后再把答案放进对方要求的格式中。

这并不是什么特殊情况。每位新的潜在客户都会发来自己的文档。有时是电子表格，有时是 Word 文件或 PDF。 参照标准不同，提问方式也不同，但相同的主题会反复出现。

然而，我们每次都觉得自己在从头开始。 完成回复需要三到六个小时，还要动用多个人，而且无法保证潜在客户最终会成为客户。随后，文档被发出去，所有劳动则被封闭在已经发送的文件里。

因此，我最初的目标很简单：下一份问卷到来时，我不想再从零开始。

看似可信，却是错的答案

最明显的做法，是保存过去的问卷并重复使用其中的答案。

我们很快就发现了为什么这样还不够。

收到的某些文档中已经带有答案。它们可能来自模板、另一家供应商或过去的一次交流。 其中一份文档写着，ISO 9001 认证取得于 2017 年。 这句话看起来完全可以直接使用。它很具体，已经写好，而且正好放在对应的问题旁边。

正确年份是 2018 年。

错误只有一年，却揭示出一个严重得多的问题。过去的答案看起来就像知识。 它采用肯定而令人安心的表达。 AI 在文档中遇到它时，很容易直接

沿用。但这句话的存在并没有告诉我们是谁写的、依据是什么，也不能说明它描述的确实是我们的产品。

于是我明白，我们不应构建一个可以复制答案的资料库。真正需要构建的，是一份记忆，记录我们确实有把握回答的内容。

收到的问卷仍有价值：它说明客户想知道什么。 但它本身无法证明答案。这种区分后来成为上下文的一条规则：客户文档保存需求；答案则必须与我们已经核实的信息保持关联。

保存事实，而不是表单

我们开始从招标文件中提取那些可以服务于下一份文件的内容。

问题不再只按客户或编号归档。我们根据主题把它们联系起来：备份、身份验证、加密、分包商、托管、服务连续性。这样，一位潜在客户标为“14.6.l”的问题，如果与另一位客户标为“CL-072”的问题讨论同一主题，就可以归到一起。

对于每个主题，上下文都会保存经过验证的事实及其出处。答案可以依据官方文档、经过核实的配置、一次测试，或主管团队的确认。我们还会保留验证日期，以及可以对外承诺到什么程度。

这些工作并不总能产生完整答案。 它也会暴露出缺口。

缺少信息时，系统必须明确说出来。 它会为相关团队准备一个具体问题，并指出答案可以补全问卷中的哪些行。这样，信息缺失就不再隐藏在一句貌似可信的话背后。 它变成一项可以识别的行动。

最后，我们会把验证答案所需的内容，与可以发送给客户的内容分开。某项配置的细节，对内部审阅者可能必不可少，却不适合出现在公开文档中。因此，上下文会准备面向潜在客户的表述，同时在旁边保留验证所需的证据与保留意见。

渐渐地，目标发生了变化。我们不再只是要完成眼前的问卷。**每一个经过验证的答案，都应减少处理后续问卷所需的工作。**

第四份问卷

第四份招标文件有一百一十五个问题，比第一份多得多。

这一次，我们不再从一堆零散文件出发。上下文已经熟悉反复出现的主题，也知道去哪里寻找对应事实。其中九十六个完整答案，可以直接根据积累的记忆和产品来源生成，无需重新收集信息。

系统也对文档的其余部分进行了分类。有些答案依据充分，但仍需最后确认。另一些仍不确定。还有少数问题，现有信息不足以回答。

更重要的是，缺口的性质发生了变化。我们不再花费数小时，漫无目的地寻找谁知道什么。尚未解决的问题，要么对应商业决定——例如承诺期限——要么只是需要个别技术确认。剩余工作清晰可见，可以分派，而且范围有限。

这个结果让我理解，真正“自动回复”一份招标文件意味着什么。它不是让 AI 信心十足地填满所有格子。而是立即准备所有能够准备的内容，显示每个答案的可信程度，并清楚隔离那些仍然需要人作出判断的部分。

十分钟并不意味着百分之百确定

在我的实践中，准备一份招标文件的时间，从三到六小时缩短到大约十分钟，足以产出文档的 80% 至 90%。

这个数字可能造成误解。它并不意味着 80% 至 90% 的答案会立即变得确定，并足以代表公司作出承诺。它意味着文档已经大体准备就绪：已知答案填到了相应位置，来源已经找回，薄弱环节得到标记，缺少的信息也转化成了具体请求。

人工审阅仍然必不可少。不过，审阅集中在真正需要关注的地方。审阅者不再把时间花在翻找旧文件、再次询问同一批人上，而可以核实承诺、决定哪些内容可以披露，并专注于新的或仍不确定的答案。

因此，最大的收获不只是从数小时缩短到几分钟。更重要的是，工作的性质发生了变化。

第一份问卷需要一次调查。后续问卷则受益于这次调查带来的全部认识。一个日期错误变成了一条谨慎规则。从某个团队获得的确认，变成了可以重复使用的事实。一个没有答案的问题，变成了已知的缺口，而不再是遗忘。

潜在客户仍然会收到自己的文档。

而企业不再丢失为了准备这份文档而学到的东西。

要点——答案的价值来自证据

过去的答案看起来就像知识：它语气肯定、形式完整，却对出处和日期保持沉默。 应保存经过核实的事实及其来源，而不是保存已经填好的表单。 明确的缺口胜过貌似可信的句子：它会变成一个交给某个人回答的问题。

3.2

从尚未成形的思考出发写一篇文章

我在博客上发表的文章，很少始于电脑前一张精心准备的空白页面。

一个想法通常在遇到困难、试用工具、进行讨论或改变看法之后出现。有时，它会在步行、乘坐公共交通工具或坐在沙发上时到来。我感觉它值得写成一篇文章，却还不知道确切的角度，也不知道该按什么顺序讲述。

这时，我通常会拿起手机。

我并不只是用它录一段语音笔记，或口述一篇稍后再誊写的文字。**我会与 AI 展开对话**。在我思考时，它倾听、回答，也继续追问。这场对话进入同一个工作环境，之后我会在电脑上重新找到它。因此，我可以远离书桌开始工作，而不会创造一则日后还得转移并重新解释的孤立笔记。

把正在成形的思考倒出来

我创建一个上下文，目标很简单：写出这篇文章。然后，我把脑中的一切都告诉它。

有时一说就是两个小时。我先讲第一点，又转向另一点，然后再回到第一点。我讲述一段经历，突然跳到结论，补充一个细节，打开一条新思路，又暂时放下上一条。同一个想法，我可能会重复四次，因为它以不同形式反复回到脑中。我也可能先肯定一件事，一个小时后又说出相反的话，因为我的思考在对话中已经发生了变化。

这种混乱并不是开始前需要避免的意外。一个想法仍在诞生时，这往往正是它真实的样子。

与 AI 对话，让我无需过早施加结构就能继续下去。一条回答促使我补充某段回忆。一个问题揭示出我刚刚跳过了一步。一次改写让我发现，自以为显而易见的事情其实还没有说清。我纠正、补充限定，再转向另一个方向。

上下文把这些话保存为会话来源。它并非只留下最整洁的句子。它也保留反复、犹豫和矛盾，因为这些内容能帮助理解思路是怎样一步步变化的。

随后，AI 会把同一个想法的四种表述联系起来，而不是把它们视为四个不同的想法。它会指出两项互不相容的论断，找回一小时后出现的限定，并把一个示例与它所说明的原则放在一起。**它并不让混乱消失，而是让混乱变得可以加工。**

这第一步让我不必过早作出选择。我还不需要决定用哪句话开篇、某个想法是否值得单设一节，或事件应当按照什么顺序讲述。我首先要把脑中的内容全部倒出来，不让那些一闪而过的想法溜走。

不过，上下文知道这些材料为何存在。 它接收的不是一场没有目的地的对话，而是一篇有待构建的文章尚且杂乱的材料。

把 AI 当作帮助想法成熟的伙伴

放入第一批材料后，我会询问 AI 还缺什么。

它指出一个过于草率的论断、一个从未解释的前提，或一个只有熟悉我的经历才能理解的段落。它要求我给出示例。它把在不同时刻表达的两个想法联系起来。有时，它会提出一个看似合理的方向，而我会拒绝，因为它不符合我的真实经历。

我回答、补充和纠正。这场对话还没有产出文章。它只是在让文章的材料变得更充实。

AI 的作用很重要，但它的位置依然清楚。它掌握我放入的全部内容，能够比我更快地从中寻找张力。但只要我还没有表达出来，它就不知道我真正想主张什么。 它打开问题；由我决定这个问题是否值得改变文章的走向。

这种对话缩短了发现薄弱之处所需的时间。如果独自工作，我或许要到两天后才会发现少了一个示例，或者结论建立在一个从未解释的步骤上。AI 会在想法仍然容易修改时，促成这次发现。

当经验不再足够时加入来源

我的文章常常从亲身经历出发。 但外部材料仍有它的作用：

- 一项参考资料为我观察到的现象提供名称；
- 一份文档验证某种运作方式；
- 另一位作者主张相近的立场，或指出一种我没有遇到的局限；

- 补充研究揭示，我的经验只覆盖一种特殊情况。

这些文档作为**文档来源**进入上下文。它们与我最初说出的话发挥不同作用。**会话来源**保存我所主张的内容，以及我对自身经历的理解。**文档来源**则带来可以与这些经历相互对照的材料。

于是，文章可以区分哪些是我的观察、哪些由来源确立，哪些仍然只是解释。这种区分能保护文本，避免一种常见诱惑：把个人经历当成普遍规律。

把来源转化为可重复使用的想法

到了这个阶段，上下文不会直接从来源跳到文章提纲。它会先把原始材料转化成**原子笔记**。

每则笔记都会稳定一个来自对话或文档、可以独立成立的想法：

- 一项我想主张的论断；
- 一个让它变得具体的示例；
- 一项限制其适用范围的限定；
- 一个文章必须回应的反对意见；
- 一个证实或反驳我经验的外部事实。

笔记会保留自己的出处。这样，我无需重读整场对话就能理解想法；如果需要核实某种表述或找回其语境，我仍可以回到来源。

这种转化改变了准备文章的方式。

- 对话跟着我的思路走；笔记则把讨论同一个想法的内容汇集起来，即使它们相隔一小时才出现，或来自两个不同来源。
- 一则核心笔记可以让文章的角度变得清晰。
- 其他笔记则为它提供示例、证据、反对意见或影响。

这些笔记还不是段落，也不是预先写好的提纲。**它们对最终交付物保持中立：同一个想法可以用于这篇文章、一次演示或另一项工作。**

先经过提纲，再开始写作

材料足够充实后，我会要求一份提纲。

这一步可以避免我从一场内容丰富的对话直接跳到一篇只是照搬对话顺序的文章。讨论通过回头、纠正和联想向前推进。文章则必须引导一个既不了解讨论、也不知道其历史的读者。

提纲确定这条推进路径。它说明首先要理解什么、主要示例放在哪里、需要处理哪项反对意见，以及结论究竟能够肯定到什么程度。

AI 根据已有材料提出结构。我会调整顺序、收紧内容，并拒绝那些让次要方向占据过多篇幅的部分。 通常，两三轮交流就足以形成清晰的方向。然后才进入写作。

到了这个阶段，AI 不再根据一条模糊指令凭空生成文章。它面对的是一个想法与主题的语料库，这些内容来自我讲述的经历、对问题的回答和加入的来源，还有一份已经确认的提纲。 它合并重复内容，重新安置示例，并把曲折的口述转化为可读的文字。

我会重读结果，专门寻找三种偏差：

- 消失的限定；
- 变得过于肯定的段落；
- 背离原意的表述。

我会修正这些偏差，然后决定最终发布哪个版本。

上下文能为短期工作带来什么

一篇文章可能只需一两天就能写完。它似乎不需要像持续数年的产品，或包含数十章的书那样拥有长期记忆。

但这个轻量案例恰好说明，时间并不是唯一的问题。

上下文把最初的话、后来加入的来源、提出的问题、选定的提纲和完整文章保存在一起。 如果工作中断，我能找回它的状态。 如果某个句子变得值得质疑，我可以回到它的出处。如果写作中出现的一个想法超出了这篇文章，它可以单独稳定下来，为未来的工作提供材料。

所有这些材料不一定都值得永久保存。一个小型上下文可以完成目标，产出文章，然后被归档。 它的价值，在于形成期间为工作提供的连续性，也在于留下了一条痕迹，让我们日后需要时能够理解文章为何走向这个方向。

发布本身还可能触发其他任务：准备摘要、把文章改编成多种语言，或生成上线所需的内容。这些操作延伸的是同一项成果，也遵循上下文已经知道的规则。

整个过程仍然很简单：一段经历变成口述，口述遇到问题和来源，材料形成提纲，提纲再变成文章。

AI 加速每一次转化，而上下文保留从一步走向下一步的脉络。

这一过程绝非长篇写作所独有。只要一个想法值得拥有比手机里一条被遗忘的语音笔记更好的归宿，它就已经开始。

要点——混乱也是材料

一个正在诞生的想法会重复、自相矛盾，也会回头：这是它的正常形态，不是开始之前需要修正的缺陷。上下文把这些话保存为会话来源，再让它们变得可以加工。提纲位于对话与文章之间，因为读者不会按照一场讨论的顺序前进。

3.3

在混乱中让产品议题逐渐成熟

前两章都是从头到尾只跟进一个议题。本章描述的，则是产品经理典型一天的现实：多个议题、请求和时间尺度同时向前推进。

我是公司唯一的产品经理。这意味着，与产品有关的大量信息最终都会汇到我这里。

同一天里，一位客户说了一句我觉得很重要的话。

- 它只有寥寥几个词。
- 我还不知道它揭示的是其他用户共有的问题，还是一个人的特殊困难，但我不想把它丢掉。

过一会儿，我拿到一份讨论某个行业议题演变的白皮书。

- 这一次，材料不再只有一句话，而是一份几十页的 PDF。
- 我还不知道其中什么对我有用，也不知道这份文档是否会证实我在现场观察到的现象。

接着，开发人员来和我讨论一个正在实现的功能。

- 我需要回答一个问题，明确某种行为，或重新审视几周前作出的假设。

与此同时，首席产品官希望回顾与产品另一部分有关的两项押注。

最后，与客服团队的一次对话又揭示出一份功能文档的缺失。

- 这份文档的一部分，目前还只存在于我的脑中。

一天结束时，我已经在三个或四个不同的议题之间来回切换（甚至更多）。我收到了一句短短的引语、一整份文档、一个关于开发中功能的问题、一项需要权衡的请求，以及一份文档需求。

这并不是特别糟糕的一天，而只是普通的一天。

大小不同、去向也不同的输入

产品管理的混乱，不只来自信息的数量。 也来自信息性质上的差异。

这五种输入分别指向产品生命周期中的不同时刻：

- 客户的一句话是一个微弱信号，我还不知道后面会不会有更多类似信号；
- 一份 PDF 提供对某个行业的总体看法；
- 与开发人员的讨论关乎我们此刻正在构建的东西；
- 客服团队的请求关乎已经存在、却需要解释的内容；
- 首席产品官的问题迫使我们比较几种可能的未来。

这些内容本就不应进入同一份文档。

- 它们并非都在讨论同一主题，也不处于产品生命周期的同一时刻。

如果只是把它们汇集进一个笔记清单，我保存了信息，却把理解它们的工作推迟到真正需要它们的那一天。

多个上下文让我能够同时维持多条线索。

- 那句引语会进入它似乎所属的议题，即使其含义仍不确定。
- PDF 会连同出处一起保存，并可接受分析，从中提取少数真正有用的想法（**原子笔记**）。
- 与开发人员的对话进入当前功能的工作之中。
- 客服团队指出的缺口，则成为一项需要完成的工作，而不再是两场会议之间听到后便消失的一句话。

并非所有内容都会立即转化为知识：

- 客户的话也许始终含义不明，
- 白皮书也许没有多大关系，
- 直觉也许会在下一周遭到反驳。

上下文不会让进入其中的内容自动变得有价值。 它只是给这些内容一个位置、一个出处，以及在恰当时刻被找回的机会。

上下文容纳混乱，却不假装能够消灭混乱。

信息不断进入，产出请求也从未停止

一天的工作并不只是接收和整理材料：我们还必须产出。

例如，客服团队会让我为一个正在开发的功能编写功能文档。

- 需要解释它如何工作，明确重要情形，并让帮助客户的人能够正确回答问题。

同一天，我或许还想完成一篇公司博客文章。

- 这篇文章未必讨论同一个功能。
- 但它可能涉及同一个业务主题：产品所服务的行业、用户反复遇到的困难，或他们实践方式的变化。

于是，两份文档会使用同一部分知识，却讲述不同的内容。

- 面向客服团队的文档解释产品实际如何运作。
  - 它必须精确、可操作，并适合回答用户的问题。
- 文章则展开一段经历、一个业务问题或一种思考。
  - 它可以调用一部分相同的想法，但会为了另一类读者和另一个目标重新组织它们。

我不必把自己对这个主题的认识重建两次。

- 上下文保留一种中间材料（**原子笔记**与**主题笔记**），它既不是文档，也不是文章。
- AI 可以根据这些材料准备多种组合：一份围绕产品使用方式，另一份则围绕更广泛的业务议题。

当然，我会审阅两个结果。

- 支持文档影响人们对产品的理解。
- 文章代表我的观点。

但我会把时间用来核实内容与方向，而不是再次寻找相同材料，并从头重写一遍。

这一过程可以在同一天发生。新信息不断进入的同时，已经组织好的知识也让我们能够产出企业或我本人需要的文档。

一句话也可能成长为一个产品议题

早上听到的那句话，不一定会立即触发一项决定。

- 单独来看，它可能什么也证明不了。
- 几周后，另一位客户也许会表达相近的困难。
- 一次访谈会补充一项细微差别。

- 当天得到的白皮书也许会为观察到的变化提供一个名称。
- 一项支持数据可能显示，问题出现得比我们原先以为的更频繁。

这些内容会在不同时刻、以不同形式到来。

- 上下文让我们能把它们联系起来，同时不抹去各自的出处。
- 它可以指出
  - 一条新反馈与已经保存的信号相似，
  - 揭示一个矛盾，
  - 询问什么证据能在两种解释之间作出区分。

渐渐地，一句孤立的话可能变成值得认真处理的议题。到了那一刻，我已经拥有它的一部分历史：最初的信号、帮助理解它的文档、证实它的内容、反驳它的内容，以及仍未解决的问题。

上下文帮助这些材料成熟。

- 它不会判断问题是否重要，也不会替我选择解决方案。

产品押注的责任仍由我承担。

- 但我作出决定时，依据的不再只是模糊的记忆或一堆散落的文档。

随后，同一个系统继续陪伴工作：

- 加入访谈，
- 用事实检验假设，
- 准备多种方案，
- 为首席产品官准备论证，
- 编写规格说明，
- ……

一个在忙乱的一天中捕捉到的微弱信号，就这样可以延续足够长的时间，最终成为一项真正的决定。

被放弃的想法仍可继续成熟

并不是每一次探索都会走到上线。

我曾研究过一些看起来很有意思的人工智能用途，但当时的技术还不足以正确实现它们。

- 想法本身未必有问题。
- 不合适的只是时机。

在传统工作方式下，这次探索会产生几份文档，然后在决定放弃之后消失在某个文件夹里。

- 一两年后，我们还得重新找出这些文件，回忆最初的假设，并弄清当初为什么停下。

有了上下文，探索可以连同整个过程一起归档：最初的信号、设想过的方案、做过的试验、遇到的限制，以及停止的原因。

- 技术发展以后，我可以恢复这项工作，而不会把过去的负面结果误解成一个永远不好的想法。

我不仅能找回当初想做什么，也能知道它为什么行不通。于是，我可以检查障碍是否依然存在，保留仍然有效的内容，再从那里继续思考。

因此，一个产品议题的成熟过程并不总是连续向前的旅程。

- 它可以始于一句话，在几个月里逐渐丰富，中途停止，然后在产品、市场或技术发生变化后重新开始。

在不拖慢一天节奏的情况下，让秩序浮现

我的一天依然支离破碎。

- 客户不会让自己的反馈配合客服团队的请求。
- 开发人员不会特意选择我刚刚读完 PDF 的时刻来找我。
- 即使我正在编写文档或文章，需要准备的决定也不会暂停。

上下文不会把这种现实变成一串完美有序的任务。

它让每个片段都能保持连续性。

- 一句话和一份长文档可以进入系统，却不必接受相同的处理。
- 多个议题可以同时推进而不彼此混淆。
- 已经加工过的知识可以服务于多份交付物，而不会被封闭在第一份交付物里。
- 一个中断的想法可以等待，同时不失去整个历史。

这就是我在产品管理中使用它的方式：不是为了让一天减少变化，而是为了避免这些变化把我学到的东西冲散。

混乱仍会不断进入。

但它不再以损失的形式离开。

这三种用途有一个我尚未点明的共同点：每一种最终都会有某样东西离开——一份发送出去的答复、一篇发布的文章、一项作出的决定。

下一部分将关注交付物离开之后还剩下什么。

要点——多条线索并行

客户的一句话与一份五十页的 PDF，需要不同的处理，也会去往不同的地方。上下文给每个片段一个位置、一个出处，以及在恰当时刻被找回的机会。 被放弃的探索会保留停止的原因：当技术或市场发生变化时，这项探索便可以重新启动。

部分 4

交付物离开后留下了什么

前面三种用途，最终都以交出一份交付物结束。 这一部分要关注的是，交付物离开后还剩下什么。

同一个问题相隔六个月被问了两次，成为贯穿其中的线索。

第一次，答案只占电子表格中的一个单元格。 第二次，这个答案仍然存在，却已经不够用了：缺少的是当初产生答案的推理，以及支撑这段推理的文档之间的关系。

4.1

单元格容得下答案，却容不下推理

让我们回到上一部分的一份招标文件。

问题出现在电子表格的 D47 单元格中：

> 你们的备份在静态存储时是否加密？

最终答案只有三行。 但要写出这三行，我必须汇集多个要素：

- 我们的文档描述备份的一般运作方式；
- 一个配置文件显示实际启用的设置；
- 运维团队确认这项设置在生产环境中的应用方式；
- 法务人员明确哪些内容可以向潜在客户透露。

我从电子表格转到文档，从文档转到配置，然后从与运维团队的交流转到法务判断。 最后，我回到 D47 单元格，写下答案。

文档被发送出去。

一条链条末端的三行文字

单元格里装的是工作结果。 它没有显示产生结果的路径。

它不会说明：

- 我查阅了哪份文档；
- 我核实了哪项设置；
- 谁确认了设置的应用情况；
- 哪种表述获得了许可；
- 发给潜在客户的文字中省去了哪些保留意见。

但这些内容确实存在。 它们决定了这三行中的每一句；在我回答问题时，它们之间的关系在我脑中十分清楚。

文档一旦发出，这个整体便消失了。 电子表格保留答案，却没有保留让人能够信任答案的推理。

每个工具都完成了自己的工作

问题并不是整理不当。

- 文档确实放在文档空间中。
- 配置仍留在产品里。
- 团队的回答保留在相应的讨论串中。
- 法务判断有自己的载体。
- D47 单元格也处在潜在客户文档中应有的位置。

每个工具都保存了交给它的对象。 但没有一个工具负责保存整段推理。

要重建它，必须知道该寻找哪些要素、去哪里寻找，以及每个要素发挥了什么作用。 这些知识仍然只装在准备答案的人脑子里。

交付物压平了产生它的一切

交付物必须为接收者呈现合适的结果。

- 潜在客户需要的是清晰答案，而不是为了得到答案所进行的完整调查日志。

这种简化是必要的。但当交付物成为工作的唯一痕迹时，它就会带来代价。

最终结果总比通往结果的路径短——无论是发出的答复、发表的文章，还是作出的产品决定。如果只剩下结果，下一项工作就必须重新走过其中一部分路。

给推理一个地址

上下文既不取代电子表格，也不取代文档、配置或团队交流。

- 它为连接这些内容的工作提供一个共同地址。

D47 的问题、查阅过的要素、获得的验证和最终答案，都属于同一个议题。它们可以保持原有的形态和位置，同时在上下文中继续相互连接。

单元格继续履行自己的职责：回答潜在客户。上下文则履行另一项职责：保存将来理解、核实并继续处理这个答案所需的内容。

六个月后，这种差别变得至关重要。

要点——交付物不保存推理

一个答案可能只有三行，却依赖多个来源、验证和保留意见。每个工具保存自己的对象；它们之间的关系仍然要靠人来记住。 上下文为连接这些要素的工作提供一个地址，却不取代它们原本所在的工具。

4.2

六个月后，答案已经不够用了

D47 单元格出现六个月后，一位新的潜在客户提出了同一个备份加密问题。

- 我只用十秒钟就找到了过去的答案。
- 它清晰、行文得体，随时可以复制。

但我盯着它，却无法使用。

只需三个问题，就足以让我没法接着用下去：

- 这个答案来自哪里：文档、观察到的设置，还是运维团队当天对我的解释？
- 法务人员批准的是这一确切表述，还是一个更加谨慎的版本？
- 今天的设置是否仍然相同？

答案留了下来；让人能够信任它的依据却消失了。

找回并不等于能够恢复工作

过去的单元格还给我一段文字。 但它没有还给我：

- 证据强度，
- 适用限制，
- 再次使用前所需的核查。

如果没有其他记忆，我必须按相反顺序重做上一章的整场调查。我并非完全从零开始，因为手里已经有一个答案。但我仍然不知道过去的工作还有多少有效。

这就像把认证年份写成 2017 而不是 2018 的错误，只是这一次发生在我们自己的答案上：句子明确，也确实出自我们之手，但它的证据已经不可见。

找回每个要素承担的作用

在上下文中，过去的答案不会孤立存在。它会保留产生答案的推理所需的关系：

```
文档 ——— 描述 ———┐
配置 ——— 确认 ———┼——→ D47 答案
运维团队 — 补充限定 —┤
法务 ——— 批准 ———┘
```

这些动词改变了我可以如何使用每个要素。

- 文档描述一般运作方式；它无法单独证明实际启用的设置。
- 配置确认某个时刻的设置；它不决定哪些内容可以对外传达。
- 运维团队补充来自实际运营的限定。
- 法务批准面向潜在客户的表述。

我找回的不再只是与答案有关的四个对象。 我也找回了每个对象承担的功能。

这种精确性可以防止一个来源证明超出它实际能够确立的内容。它也说明了时间过去后，需要重新进行哪项核查。

核查可能发生变化的内容

上下文不会把旧答案变成永恒的真相： 它会为核查答案做好准备。

- 文档可能仍然有效，而配置已经改变。
- 配置可能保持相同，但获准使用的表述需要重新审视。
- 产品演进后，运维团队曾经补充的限定可能已经不复存在。

这些关系让差别清晰可见。我可以检查可能发生变化的要素，而不必毫无方向地重做整场调查。

最终答案也许与六个月前完全相同。差别在于，我知道自己为何仍能使用它。

保存真正服务于工作的关系

目的不是把所有文档彼此连接起来。

- 两个文件可能含有同一个词，却没有参与同一段推理。

当一种关系会改变我们理解、核实、决策或产出的方式时，它才值得保存。

- 对 D47 而言，四种关系已经足够：描述、确认、补充限定和批准。

它们可以回答具体问题：

- 这个答案依据什么？
- 今天需要重新核查哪一部分？
- 谁曾批准我们能够传达的内容？
- 什么变化会让这个答案失效？

因此，上下文并不试图绘制一张涵盖我全部知识的完整地图。它只保存某项具体工作所需的关系网络。

打开机器

第一册展示了上下文如何诞生、它怎样改变人与 AI 的关系，以及它如何服务于多种用途。 D47 的案例则揭示出仍需理解的问题。

- 如何保存工作的使命，而不必在每次会话中重复？
- 来源、想法、决定和交付物应放在哪里，才能继续承担彼此不同的作用？
- 如何只加载此刻有用的内容，而不是把所有材料都塞进对话？

第二册将打开这台机器。它会沿着材料的完整路径前进：从进入上下文，直到完成转化、产出，再到恢复工作。

要点——找回让人能够信任的依据

找到一个答案还不够：还要找回让它依然可用的依据。一种有名称的关系，会说明来源、验证或保留意见所承担的确切作用。上下文通过显示什么支撑答案、什么可能已经改变，为核查做好准备。

卷 II

构建上下文，让它保持活力

5 上下文的内部结构
6 生态系统中的上下文

部分 5

上下文的内部结构

到目前为止，我们一直通过上下文能够做到什么来描述它。 这一部分将把它打开。

首先会展示完整的目录树，随后从头到尾跟随一次工作会话：打开并找回当前状态，让材料进入，把材料转化为想法，产出一份交付物，作出取舍，最后保存，以便下一次会话无需重建任何内容就能重新开始。

贯穿这六章的示例完全是虚构的：在决定产品是否需要改变之前，先了解为什么一些现场技术人员提交的作业报告并不完整。

5.1

完整地图

在第 2 部分，我暂时放下了那个关注作业报告质量的上下文：

- 三个来源，
- 两则原子笔记，
- 一则主题笔记，
- 一份用于下一次访谈的指南。

当时，我关注的是它帮助我们理解了什么。 现在，我要打开文件夹本身。

下面是本书所用版本的完整目录树：

```
contexte-qualite-comptes-rendus/
├── README.md                 总体导航，自动生成
├── offline.md                会话之外的收件箱
│
├── 00-frame/                 提供持久方向的内容
│   ├── README.md             区域导航，自动生成
│   ├── mission.md            目标、问题与边界
│   ├── directives.md         使命特有的规则
│   └── loading.md            打开时要读取的内容
│
├── 01-state/                 工作的当前状态
│   ├── README.md             区域导航，自动生成
│   ├── state.md              已确定内容、不确定项与欠账
│   ├── synthesis.md          已经沉淀为持久知识的内容
│   ├── todo.md               要完成的行动
│   ├── decisions.md          构成上下文的选择
│   ├── contradictions.md     仍然存在张力的内容
│   ├── handoff.md            工作交接说明
│   ├── checkpoints.md        一系列简洁的状态记录
│   └── health.md             上一次检查的结果
│
├── 02-history/               带日期的事实记录
│   ├── README.md             这类记录的作用，自动生成
│   └── YYYY-MM-DD.md         当天发生的事件
│
├── 03-sources/               收到或表达出来的内容
│   ├── README.md             来源清单，自动生成
│   ├── ingestion-log.md      读取了什么、产生了什么
│   ├── documents/
│   │   ├── README.md         文档清单，自动生成
│   │   ├── raw/              原始文件
│   │   ├── fulltext/         可处理的完整文本
│   │   └── SRC00001-….md     身份、出处与状态
│   └── sessions/
│       ├── README.md         会话清单，自动生成
│       └── SESSION-….md      工作中表达出来的话
│
├── 04-notes/                 从来源中提取的知识
│   ├── README.md             笔记地图，自动生成
│   ├── atomic/               每则笔记一个想法——ATO00001-….md
│   ├── thematic/             连接多个想法的视角——THM00001-….md
│   ├── glossary/             一个稳定下来的术语——GLO00001-….md
│   └── decision-records/     一项取舍——DRE00001-….md
│
└── 05-outputs/               为某种用途产出的内容
    ├── README.md             结果清单，自动生成
    ├── _draft/               正在制作的交付物
    ├── questions-entretien.md    用于继续工作的访谈指南
    ├── note-de-cadrage.md        用于决策的概述
    └── presentation-equipe.md    为团队改写的同一批认识
```

编号规定的是阅读顺序，而非价值层级：

- 框架与状态为工作定向；

- 历史让我们能够核查工作的经过；
- 来源保存材料，笔记把材料转化为知识；
- 交付物则为特定接收者赋予知识相应的形态。

六个区域，六种功能

`00-frame/`回答："我为什么在这里，又要遵循哪些规则？"这个区域很少变化，原因很简单：修改使命，会改变我判断所有来源、任务和结果的方式。

```
contexte-qualite-comptes-rendus/
├── 00-frame/                提供持久方向的内容
│   ├── README.md            区域导航，自动生成
│   ├── mission.md           目标、问题与边界
│   ├── directives.md        使命特有的规则
│   └── loading.md           打开时要读取的内容
```

`01-state/`回答："我现在进展到哪里？"这个区域随工作变化，让我能够区分已经确定的内容与假设、未来行动与已经作出的决定，以及短暂困难与值得保持可见的矛盾。


```
contexte-qualite-comptes-rendus/
├── 01-state/                工作的当前状态
│   ├── README.md            区域导航，自动生成
│   ├── state.md             已确定内容、不确定项与欠账
│   ├── synthesis.md         已经沉淀为持久知识的内容
│   ├── todo.md              要完成的行动
│   ├── decisions.md         构成上下文的选择
│   ├── contradictions.md    仍然存在张力的内容
│   ├── handoff.md           工作交接说明
│   ├── checkpoints.md       一系列简洁的状态记录
│   └── health.md            上一次检查的结果
```

`02-history/`回答："实际发生了什么？"这个区域并不试图把会话讲成一个漂亮故事：它只保留有用事实，例如加入了一个来源、放弃了一条思路、作出了一项决定、创建了一则笔记。我很少查阅它，但当当前状态显得前后矛盾时，它就会变得至关重要。

```
contexte-qualite-comptes-rendus/
├── 02-history/              带日期的事实记录
│   ├── README.md            这类记录的作用，自动生成
│   └── YYYY-MM-DD.md        当天发生的事件
```

03-sources/回答："我收到了哪些材料？"外部文档以固定形态进入，我自己的话则随着会话逐步进入。无论哪一种，来源始终与上下文对它的理解分开。

```
contexte-qualite-comptes-rendus/
├── 03-sources/              收到或表达出来的内容
│   ├── README.md            来源清单，自动生成
│   ├── ingestion-log.md     读取了什么、产生了什么
│   ├── documents/
│   │   ├── README.md        文档清单，自动生成
│   │   ├── raw/             原始文件
│   │   ├── fulltext/        可处理的完整文本
│   │   └── SRC00001-….md    身份、出处与状态
│   └── sessions/
│       ├── README.md        会话清单，自动生成
│       └── SESSION-….md     工作中表达出来的话
```

SRC00001-….md 条目既不包含文档本身，也不包含文档摘要。它回答了我需要了解的所有相关问题：

- 文档来自哪里，我为什么加入它；
- 去哪里找到原件和完整文本；
- 文本提取是否成功；
- 应当给予它怎样的可信程度；
- 阅读它是否已经产生了笔记。

因此，它让来源的状态清晰可见，却不触碰材料本身。

04-notes/回答："我学到了什么？"

- 一则原子笔记稳定一个想法。
- 一则主题笔记连接多个想法。
- 术语表明确一个词语。
- 一则决策记录保存作出取舍的理由。

```
contexte-qualite-comptes-rendus/
├── 04-notes/                从来源中提取的知识
│   ├── README.md            笔记地图，自动生成
│   ├── atomic/              每则笔记一个想法——ATO00001-….md
│   ├── thematic/            连接多个想法的视角——THM00001-….md
│   ├── glossary/            一个稳定下来的术语——GLO00001-….md
│   └── decision-records/    一项取舍——DRE00001-….md
```

05-outputs/回答："我要如何使用这些知识？"草稿和结果都存放在这里，并采用受众期待的形式：

- 一份访谈指南；
- 一篇文章；
- 一份规格说明；
- 一次演示；
- 一项决定，以及其他可能的形式。

```
contexte-qualite-comptes-rendus/
└── 05-outputs/              为某种用途产出的内容
    ├── README.md            结果清单，自动生成
    ├── _draft/              正在制作的交付物
    ├── questions-entretien.md   用于继续工作的访谈指南
    ├── note-de-cadrage.md       用于决策的概述
    └── presentation-equipe.md   为团队改写的同一批认识
```

想法的世界：可用于多种构造的积木

04-notes/与 05-outputs/之间的分离，承载了上下文最重要的优势之一。笔记和术语表共同形成一个*想法的世界*：一组可以无限复用的知识积木。它们已经不再是原始材料，却还不是面向某个人的结果。


```
03-sources/
     │
     ▼
04-notes/                        想法的世界
├── ATO00001                     一个稳定下来的想法
├── ATO00002                     另一个稳定下来的想法
└── THM00001                     对二者关系的一种解读
     │
     ├──> questions-entretien.md
     ├──> note-de-cadrage.md
     └──> presentation-equipe.md
```

在这个示例中，填写时的阻力与结构化数据的需要，早在我知道要以什么形式呈现它们之前就已经稳定下来。

- 访谈指南从中提取问题，
- 概述从中提取需要关注的事项，
- 演示则从中提取面向团队的叙述。

这三份交付物并不拥有它们所使用的知识。

- 它们只是为了特定需要、受众和格式，选择并重新组合其中的一部分。
- 因此，一份交付物不会耗尽构成它的笔记：积木仍可用于另一种构造。

如果没有这个中间世界，每次产出新结果时，我都必须重新打开来源、重新建立联系，再次表达已经得出的认识。

上下文不仅保存已经产出的内容，也在构建一种能够持续产出新内容的能力。

并非所有文件都以相同方式演变

只有当我知道目录树中的每个要素如何演变时，它才真正变得可读。

有些文件会被重写，以呈现此刻最准确的状态：

- `state.md`；
- `handoff.md`；
- `05-outputs/_draft/`中的草稿。

另一些文件只会不断追加，以保存过去：

- 会话来源；
- 按日期记录的历史；
- 知识沉淀记录；
- 一系列检查点。

收到的文档保持不变，被另一则笔记取代的旧笔记也依然可以查阅。最后，`README.md` 文件根据实际存在的内容自动生成：手工修改它们，会制造现实的第二个版本，而这个版本最终必然出现偏差。

这种多样性解释了为什么一个大文件已不再够用。那样做，等于要求同一段文字同时：

- 显示现在，
- 保存过去，
- 接收原始材料，
- 稳定想法，
- 承载完成的结果。

一副骨架，而不是一份照抄的配方

这副骨架不是放在上下文旁边的一份说明文档，而是一个我会直接复制的真实文件夹。

- 每个文件一开始就带有应当包含的章节和一些提示，因此，相隔数月创建的两个上下文仍然共享同一组参照。

不过，结构依然可以调整。 一个上下文可以加入：

- 一条特殊指令，
- 一种其业务特有的交付物，
- 一种从未出现过的来源。

共同基础主要保留我的工作需要的三条边界：

- 稳定内容与当前状态，
- 原始材料与解释，
- 知识与知识的最终表达形式。

与上下文对话

到目前为止，我展示的都是文件夹和文件。这可能让人以为，我必须逐个打开文件，知道应在哪里写入，并亲自移动每一条信息。我的工作方式并非如此。

我会对 AI 说：

> 打开关注作业报告质量的上下文。

> 把这份文档添加为来源。

> 把这个来源转化为笔记。

> 保存工作。

每个请求背后，都有一套我只需编写一次、AI 可以在每次会话中重复执行的流程。

- 在 Claude Code 和 Codex 中，这种可重复使用的流程有一个名称：**技能（skill）**。

例如，打开技能会规定：

- 开始前需要吸收哪些内容；
- 哪些文件必须读取；
- 哪些内容应留在磁盘上，直到使命真正需要；
- 如何报告问题；
- 应当向我返回哪些信息，让我可以立即行动。

我不必每次重新说明这些步骤。

- 我只需要让它打开上下文。

对话承载我此刻的意图。

- 技能承载可靠执行重复动作的方式。

因此，我的系统拥有几个稳定的动词：

- 创建一个上下文；
- 打开它并找回进行中的工作；
- 添加一个来源或保存一次会话；
- 把来源转化为笔记；
- 保存工作状态；
- 检查一致性；
- 在目标实现后归档。

并不是我与 AI 的每次交流都会通过技能进行。我仍会使用自然语言思考、提问、纠正和作出取舍。

技能只约束那些始终需要遵循相同步骤的操作。

- 例如，它们会防止一个来源在没有保留出处的情况下被转化，也会避免会话在没有为恢复工作做好准备时结束。

流程保持稳定，但判断并不会自动化。两个模型可能以不同方式表述同一则笔记。但它们仍须查阅相同的要素，尊重相同的边界，并进行相同的检查。

因此，与他人分享技能，并不等于把我的上下文交给他。而是让他拥有同样的方法，去创建并持续运用自己的上下文。

关于技能的运作机制，我不会再展开。

技能既不是我的发明，也不是上下文独有的特点。Claude Code 和 Codex 提供它们，本就是为了描述可重复使用的流程；关于如何创建和使用技能，官方文档会比我在这里解释得更好。

真正属于上下文的，是我们选择赋予它哪些能力：创建、打开、输入材料、转化、保存、检查和归档。 也包括这些能力衔接的顺序，以及每一种能力必须尊重的边界。

文件保存上下文知道的内容。 技能组织它演变时所经过的动作。

地图已经展开。 现在，参观可以像一次真正的会话那样开始：打开上下文。

要点——上下文的骨架

一个新的上下文从一开始就配备齐全：六个区域，每个区域回答不同的问题。编号规定阅读顺序，而非价值层级。让目录树变得可读的，不是它的形状，而是我们知道每个文件如何演变。 Claude Code 和 Codex 提供技能机制；上下文则定义重复动作、动作顺序及其边界。

5.2

打开：找回使命与现状

距离我上次处理作业报告，已经过去两周——这类文档由技术人员在设备作业完成后，记录自己观察到了什么、做了什么。

- 访谈和笔记仍在原位。

但如果我随意重新打开它们，就要花半小时重建自己原本已经知道的内容，而且重建出来的结果还未必准确。

- 因此，打开时要从一个小得多的核心开始：框架、状态与工作交接。

我不必逐一提出这些要求。我调用打开技能，它便按流程执行。

首先找回我为何在这里

AI（通过它的上下文打开技能）读取的第一个文件是 `00-frame/mission.md`。它的模板提出五个问题：

```
# 使命

## 目标
这个`上下文`应当帮助实现什么？

## 核心问题
什么问题指导筛选与探索？

## 主题范围之内
哪些材料可以进入？

## 主题范围之外
哪些内容即使看起来有趣，也应拒绝？

## 预期交付物
如果已经知道，我们想产出什么？
```

在我的文件中，开头是这样填写的：

```
## 目标
了解哪些因素降低了作业报告的质量。

## 核心问题
如何在不让人对填写望而却步的情况下，收集可以利用的信息？

## 主题范围之外
- 在对照多个来源之前选择界面。
- 根据一次访谈作出普遍结论。

## 预期交付物
为产品决策和后续访谈做好准备。
```

如果使命只写“改进作业报告”，它并不能为我提供多少方向。而核心问题已经能让某些文档显得比其他文档更相关；主题范围之外的内容，则能防止一个诱人的想法悄悄把探索变成解决方案项目。

- 交付物也可以暂时未知。
- 我会如实写下这种不确定，而不是凭空制造一个过早的结果：这样，上下文就知道自己仍处于探索阶段。

我设定的规则正在等我

两项相近的使命，可能需要不同的预防措施。`directives.md` 把这些规则与工作保存在一起：

- 保密要求，
- 证据标准，
- 词汇，
- 验证方式或预期形式
- ……

模板刻意保持简单：一条规则、一个日期，以及命令式表述。

```
# 指令

- (2026-09-11) 每次访谈在纳入系统前必须匿名化：用受访者的职能替换其姓名，但保留公司名称与国家。
- (2026-09-11) 区分来源实际显示的内容与我们由此作出的推断。
- (2026-09-11) 在记录真实用途之前，不提出任何解决方案。
```

一条指令只有能够指导未来行动时才有价值。

- “保持谨慎”仍让我不得不重新完成全部解释；
- “区分观察与推断”则在编写笔记或交付物时给我一项明确的检查标准。

但指令不只是一项我打开文件时重新看到的要求。它是一条持久规则，只要某项行动落入适用范围，上下文就必须执行。

例如，我加入一份新访谈的文字稿。我只是让上下文（通过专用技能）将它纳入系统，并没有想起要提醒其中含有受访者姓名。但匿名化指令仍会在添加过程中生效：技能先用受访者的职能替换姓名，同时保留公司名称和国家，然后才把文字稿保存为来源。

我不必恰好在规则变得必要的时刻想起它。上下文会替我记住。

因此，打开过程结束后，指令并不会停止生效。它们在整个会话中持续适用，陪伴上下文的各种动作：

- 添加来源时，匿名化规则保护受访者身份；
- 创建笔记时，证据标准规则将来源显示的内容与我们的推断分开；
- 准备交付物时，最后一条指令会防止我们过早地把某种解决方案说成既定选择。

执行某项行动的技能，必须先识别与行动有关的指令，再检查结果是否符合这些指令。

```
请求
    ↓
适用指令
    ↓
技能执行行动
    ↓
检查结果
```

因此，指令环绕着行动：执行前，它们约束行动；执行后，它们又成为检查结果的依据。它们无法独立行动，也并非无所不知。真正让它们在恰当时刻发挥作用的是技能。

使命指出去往哪里。指令则陪伴上下文的每个动作，让我无需在每个阶段重新想起已经决定的预防措施。

不是所有内容都值得重新打开

我的文件夹中有几十个来源和笔记。每次打开时全部加载，只会让优先事项淹没在大量内容中。 因此，`loading.md` 选择一个稳定的核心：

```
## 始终加载

- `00-frame/mission.md`
- `00-frame/directives.md`
- `01-state/state.md`
- `01-state/todo.md`
- `01-state/handoff.md`

## 按需加载

- 核查所需的来源
- 与当前议题相关的笔记
- 当前草稿
```

所有内容始终可以访问，但并非全部都加载到 AI 的上下文中。 `loading.md` 指定了理解使命、解释请求，并在不遗忘既有规则的情况下行动所必需的内容。

- 其余内容留在磁盘上，只在需要时查阅。
- 一个旧来源可能对核实某项论断必不可少，却无需在每次会话期间始终占用活动上下文的空间。

`loading.md` 的内容取决于具体上下文。

- 一本书会在每次打开时加载提纲，一次产品探索则会加载核心主题笔记。
- 原则始终相同：
  - 从足以采取行动的信息开始，
  - 然后在使命确有需要时再寻找其余内容。

我实际进展到了哪里

使命保持稳定；`01-state/state.md` 则生来就是为了变化。 它由三个部分组成：

```
# 状态

## 我们知道的内容
- 第一次访谈显示，延迟填写会导致细节丢失。
- 第二次访谈显示，某些必填字段会被规避。

## 仍不确定的内容
- 我们不知道填写后究竟会重复使用哪些字段。

## 欠账
- 白皮书已经加入，但其可信程度仍待评估。
```

状态不会讲述全部历史，也不会概括所有笔记：它呈现的是我对工作本身所拥有的最佳理解。

`state.md` 同时面向 AI 和人。

对 AI 而言，它是一种持久的工作记忆。

- 每次打开时加载，它会告诉 AI 哪些已确定内容可以使用、哪些不确定性必须保留，以及哪些欠账不能忘记。
- 如果没有它，AI 每次开始新会话时都必须根据来源和笔记重新构建状态。

对我而言，它是一块仪表板。

- 我可以用几行文字核查上下文将什么视为已确定、将什么仍视为不确定，以及哪些工作仍未完成。
- 如果状态与我的理解不符，我便可以修正它。

这样，几十行文字就足以让 AI 在不混淆确定内容与假设的情况下重新开始工作，也足以让我核查它所依据的理解。

`todo.md` 从中提取行动：

```
## 进行中
- T-001 — 准备一次访谈，了解必填字段的实际用途。

## 待办
- T-002 — 核实哪些数据会用于审计。

## 已完成
- T-003 — 比较前两次访谈。— 完成于 2026-09-10
```

我不一定亲自打开 `todo.md` 进行修改。 我可以直接说：

> 添加一项任务，核实哪些数据会用于审计。

上下文会创建一行，为它分配标识符，并放入正确的章节。

任务也可能在工作过程中出现。

- 如果分析某个来源时发现需要进行一项核查，AI 会建议把它加入任务，而不是任其消失在对话中。
- 它不会悄悄把每个想法都变成新的义务：当一项任务会改变优先级或带来额外工作时，作出取舍的责任仍由我承担。

工作完成后，我可以说：

> 关闭任务 *T-002*（也可以给出任务的大致描述，*AI* 会找到正确的引用）。

上下文不会删除这一行。它会把任务移到已完成，添加关闭日期，并在有用时注明任务结果或足以将其视为完成的要素。

- 完成的任务不会消失。
- 它的存在可以防止我重复同一项工作，也能解释为什么某条思路不再活跃。

因此，`todo.md` 并非一份写好后就不再变化的清单。 它是上下文的行动队列：AI 查阅它以了解还有什么要做，随着工作推进更新它，并保留已完成的任务以免重复进行。

从哪里重新开始

`handoff.md` 是核心中的最后一个文件。 每次保存时，它都会被重写，以回答一个非常具体的情形：此刻有某个人或某个 AI 到来，对之前的对话一无所知。 要怎样才能在不犯错的情况下重新出发？

```
# 工作交接

## 当前进展
三个来源揭示出填写便利性与数据运用之间的张力。

## 继续之前需要知道的事
两次访谈描述的是特定情形，白皮书则提出一种一般观点。
这批材料揭示出一种张力，却还不足以衡量其规模，也不能得出它存在于所有组织中的结论。

## 下一步行动
准备一组问题，用来识别哪些字段得到了实际使用。

## 仍未解决的事项
审计所需的结构化程度仍然未知。
```

下一步行动只有一项，而且可以立即执行。如果给我十种可能，就会把排定优先级的工作重新交还给我，而这个文件存在的目的恰恰是替我省去这项工作。

因此，打开命令会先吸收上下文沉睡期间存入的内容，读取这个核心，然后告诉我下一步行动。

- 它先找回目标，再找回细节；于是，我离开两周的代价只剩一分钟阅读。
- 现在，会话可以接收新材料，同时不失去方向。

要点——打开时的核心

重新打开一个上下文，不是重读其中的全部内容，而是加载一个小型核心：使命、指令、状态与工作交接。使命说明去往哪里，指令则规定旅程的条件。有用的工作交接只指定一项可以立即执行的行动。

5.3

输入：让材料进入

会话开始，三样东西同时到来。

- 一位运营负责人发给我一份访谈记录，
- 交流过程中，我产生了一个想法，
- 前一天留下的一则笔记还在收件箱里等待处理（笔记管理器、电子邮件等）。

这三类材料不会以相同方式进入上下文，每一类都有自己的技能。

文档以完整对象的形式到来

外部文档会连同原始文件、可处理文本，以及一份用于跟踪它的条目一起保存。

```
03-sources/documents/
├── SRC00001-entretien-exploitation.md
├── raw/
│   └── SRC00001-entretien-exploitation.pdf
└── fulltext/
    └── SRC00001-entretien-exploitation.md
```

`SRC00001-entretien-exploitation.md` 条目可能采用以下形式：

```
---
added: 2026-09-11
kind: pdf
intention: entretien
raw: raw/SRC00001-entretien-exploitation.pdf
fulltext: fulltext/SRC00001-entretien-exploitation.md
extraction: ok
reliability: medium
---

# 与运营负责人的访谈

## 为什么它在这里

了解是什么阻碍技术人员产出可以利用的作业报告。

## 解读—尚未分析

负责人描述了现场快速填写与分析所需结构化数据之间的对立。

## 涵盖内容

- 移动端填写
- 必填字段
- 数据复用

## 它产生了什么

- 分析后补充
```


这份条目不一定会概括文档。 它首先回答管理方面的问题：

- 文档来自哪里，
- 我为什么加入它，
- 我能否读取其内容，
- 应当给予它多大信任？

解读—尚未分析一节把最初的阅读印象隔离起来。 这可以避免导入时产生的一种印象，立即被当成已经确立的知识。

当来源是 PDF、网页或其他难以直接处理的格式时，提取过程会生成一份人和 AI 都可以阅读的文本版本。

- 这一步尚未产生任何知识：它只是让材料变得可访问，同时不修改原始文档。

转化随后才会发生。

- 提取的文本会接受分析，从中提炼出原子笔记、主题笔记、术语表条目或取舍决定。
- 这些笔记会进入*想法的世界*，让来源中的材料变成一组可重复使用的知识积木。

可信程度不是对所添加来源的价值评判。

- 它只说明应如何使用这份材料：是孤立证词、经过验证的数据、合同文档，还是有待确认的假设。

并非所有有用材料都已经存在于文档中。其中一部分存在于我的经验、记忆、观察，以及我在工作中作出的区分里。

当我在与 AI 的工作会话中表达它们时，它们也会进入上下文。

- 它们不会自动成为既定事实
- 它们会保留出处，日后可以与其他来源对照。

因此，添加材料既是纳入一份文档，也是保存上下文拥有者在对话中带来的内容。

人的知识通过会话进入

会话并不是一次性交付的文档。 它会随着交流逐步构成。 它保存在 `03-sources/sessions/SESSION-2026-09-12.md` 中，可能采用以下形式：

```
# 2026 年 9 月 12 日的会话

## 09:41 – AI

必填字段真的提高了质量，还是只把负担转移给技术人员？

## 09:42 – 作者

在访谈中，必填字段主要保证某个值确实得到了填写。
它无法保证这个值的准确性或用途。
如果技术人员不理解某个字段日后会如何使用，就会尽快填完，以便继续工作。
因此，约束提高的是表面上的完整性；它可能只转移了负担，却没有改善真正可用的信息。
……
```

AI 说的每句话都没有必要保存。但改变了工作的一个问题、一项反对意见或一次改写，则值得保留。

- 我的话与 AI 的话彼此分开。

- 这样，我可以找回想法的出处，并区分我的意图与系统提出的建议。

一次会话可以很混乱。

- 它的作用是捕捉思考真实的运动，而不是呈现一份完美总结。

上下文关闭时，收件箱替我们与它沟通

会话之外，我可能会在火车上、会议中途或无法联网时产生一个想法。

- 这时，它会进入收件箱。
- 这个收件箱就是放在上下文根目录下的 `offline.md` 文件。
- 我只需在这个 Markdown 文件中写下来。

它可能是这样：

```
询问技术人员是否会用笼统内容来规避必填字段。
```

下一次打开上下文时，这则笔记会被重新读取，与当前工作联系起来，然后被吸收到合适的位置。它可以变成一项任务、一个访谈问题、一则笔记；如果已经不再相关，也可以消失。

添加还不等于理解

这些材料进入以后，我的目录树包含了多个来源和当前会话的记录。

```
03-sources/
├── documents/
│   ├── SRC00001-entretien-exploitation.md
│   ├── SRC00002-entretien-exploitation.md
│   └── SRC00003-livre-blanc-donnees-intervention.md
└── sessions/
    └── SESSION-2026-09-11.md
```

这些文件证明材料已经可用。 但它们不能证明材料已经得到理解。

`03-sources/ingestion-log.md` 会区分这两个操作。 对于每个来源，它会指出分析是完整、部分完成，还是需要重做，并保留内容指纹和一些处理说明。

当文档被替换、提取结果发生变化或会话中途停止时，这种区分变得至关重要。 上下文可以继续分析，而不会假装自己已经知道一切。

材料已经进入。接下来，我还需要把它转化为足够精确的想法，使之能够彼此联系、接受讨论并重复使用。

要点——来源

文档、会话中的话和会话之外的笔记通过三扇不同的门进入，各自保留原有形态。来源保存材料与证据，而不保存日后从中得出的解释。添加不等于理解：纳入记录会区分已经存在的内容与已经吸收的内容。

5.4

转化：从来源走向想法

现在，我面前有三个来源。

- 两次访谈描述表单引起的阻力，
- 一份白皮书主张结构化数据的必要性。

我们很容易想立即作出结论，写下必填字段有损质量。 但这样太快了：上下文必须保留这些来源各自带来的内容，而不能把它们揉成一个结论。

因此，我调用用于创建和更新笔记的技能：它会读取尚未处理的内容，并从中提取想法。

提取一个可以独立成立的想法

一个来源很少会包含可以直接复用的现成想法。

- 访谈会混合事实、示例、观点、犹豫，以及所述情形特有的细节。
- 转化并不是概括所有内容，而是从中提取想法。

这个过程分为三个步骤：

```
证词                “表单太长时，有些技术人员会填入一个笼统的值，
        |                 只求能够完成。”
        ↓
观察                 有时会使用笼统的值来加快填写。
        |
        ↓
稳定的想法          过于繁琐的填写要求可能使作业报告
                                                  失去信息量。
```

有力的想法并不是耸人听闻的句子，也不是最终真理。

- 它拥有清晰而有力的核心，
- 离开产生它的原文也能得到理解
- 边界足够明确，能够被证实、限定或反驳。
- 它还会保留出处与局限，以免把一份特殊证词变成普遍规律。

AI 根据使命和指令阅读来源，再与已有笔记进行对照。想法是新的，就创建一则新笔记；否则，就用新出现的限定或局限丰富已有笔记。

它完成这些工作时，不会要求我逐一批准每种表述。之后，我可以重新阅读并修正笔记。

- 如果会话中出现歧义，它会向我提问；
- 如果日后发现自己无法解决的矛盾，它会保留矛盾，而不是替我作出判断。

```
04-notes/
└── atomic/
    └── ATO00001-la-contrainte-de-saisie-peut-appauvrir-les-comptes-rendus.md
```

```
---
id: ATO00001
status: active
sources: [SRC00001, SRC00002]
---

# 过于繁琐的填写要求可能使作业报告失去信息量

## 核心想法

当表单拖慢现场作业时，技术人员可能会填写笼统的值，或把描述缩减到最低要求。

## 为什么这很重要

形式上结构清晰的数据可能看起来很整洁，却无法准确描述现场现实。

## 限定与局限

两次访谈不足以衡量这种现象出现的频率。

## 出处／证据

- SRC00001：为尽快完成而使用笼统值的示例。
- SRC00002：增加必填字段后观察到准确度下降。

## 关联

- 与 ATO00002 形成对照。
```

标题表达的是一个命题，而不只是一个主题。无需重新打开产生它的文档，笔记本身仍然可以理解；与此同时，它保留了让我能够核实出处的链接。

原子笔记不会在初稿时就永远固定下来。

- 它可以增加限定、新证据或新关系，而不失去自身身份。

白皮书会产生第二则笔记：结构化数据是比较、汇总和自动处理的必要条件。

- 这个想法与第一则笔记分开，因为它提出的是另一件事。

拒绝过早作出判断

两个想法似乎相互对立，而我还没有足够材料在二者之间作出取舍。因此，我登记一项矛盾。

```
## C-001 – 在不损害填写质量的情况下实现结构化

- 一方面：强制字段可能使现场描述失去信息量。
- 另一方面：大规模运用数据需要数据具有可比性。
- 关键问题：确定哪些信息必须在填写时结构化，哪些可以事后处理。
- 开启日期：2026-09-11
```

这项矛盾帮助我避开两种对称的错误：过早选择一边，或者用一种谁都不会反对的模糊表述磨平分歧。

- 它会一直保持开放，直到某项决定、新来源或重新表述将它解决。

并非每种不确定性都值得这样处理。

- 我只把它用于真正影响后续推理或决策质量的张力。

连接这些想法共同表达的内容

在纳入过程中，AI 不只创建原子笔记。

- 它还会检查提取出的想法与已有笔记之间的关系。
- 当其中多个想法形成一个无法归结为单一命题的整体时，它会在同一操作中创建主题笔记，并建立连接它们的关系。

我不必再发出第二条命令。

- 创建原子笔记、将它们彼此联系，以及可能出现主题笔记，都属于同一次纳入过程。

```
---
id: THM00001
status: active
sources: [ATO00001, ATO00002]
---

# 质量同时取决于填写与运用

## 视角

不能只根据填写字段的数量来评价作业报告的质量。

## 知识沉淀

系统必须获得足以运用数据的结构，同时不能让填写变得生硬或代价高昂。

## 相关笔记

- ATO00001 — 填写约束可能使作业报告失去信息量。
- ATO00002 — 数据运用需要结构化数据。

## 张力

- C-001 — 在不损害填写质量的情况下实现结构化。
```


主题笔记不只是把多份摘要并排放在一起。它提出对各项关系的理解，并显示哪些问题仍未解决。

我意识到一个词对不同的人含义不同

有时，分歧的一部分来自词汇。这时，术语表会确定这个上下文所采用的含义。

```
## 结构化数据

按照明确形式记录的信息，可以进行比较或自动处理。

不要与以下概念混淆：要求用户在现场作业时必须填写的字段。

示例：故障类型、停机时长、相关设备。
```

这个本地定义并不打算成为普遍定义。

- 它只是防止两则笔记或两个人使用同一个词，却在谈论不同的东西。

面对交付物保持中立的积木

只要一则笔记的目的仍是稳定知识，而不是对某类受众产生效果，它就属于*想法的世界*。

- 因此，它不应过早变成文章的开场语、幻灯片、商业论据或规格说明的一部分。

这种中立并不意味着笔记要保持模糊。

- 恰恰相反，标题提出明确命题，正文阐明限定，链接则让人能够核查其基础。
- 保持中立的是它的形式：它不围绕某一个结果来组织。

因此，`ATO00001` 可以在访谈指南中变成一个关于规避填写的问题，在概述中变成一项风险，或在团队演示中变成一个示例。 `ATO00002` 可以支持一个关于数据用途的问题，证明某项产品要求的合理性，或向团队解释为什么自由文本并不总是足够。

主题笔记不会把这些积木合并成最终文本。

- 它说明把它们放在一起能理解什么，并保留那项需要根据不同交付物以不同方式处理的张力。

这个中间层避免了为每次产出重新进行分析。

- 来源仍然是证据，
- 笔记成为可以调用的知识，
- 每份交付物则根据自身需要使用这些知识。

我可以从每个想法追溯到证据

到了这个阶段，关系可以这样表示：

```
SRC00001 ┐
         ├──> ATO00001 ──┐
SRC00002 ┘               │
                         ├──> THM00001
SRC00003 ──> ATO00002 ───┘
```

标明关系类型的链接让我能够区分一个来源是在证实、限定还是反驳某个想法。

- 可信程度则让人看出，哪些内容基于多次观察，哪些仍然只是设想。

纳入记录会显示哪些来源真正为笔记作出了贡献。它防止我把一份仅仅存在于文件夹中的文档，当作已经得到吸收的内容。

这些笔记并非为某一份特定交付物而写。

- 它们形成一套共同知识，可以支持多种结果，而无需复制，也不会被封闭在其中任何一份结果里。

***想法的世界*正是这种可以复用的资产：一组足够稳定、能够被找回，又足够独立、能够重新组合的积木。**

要点——原子笔记

稳定的想法承载的是命题，而不只是主题；无需重新打开产生它的文档也能得到理解。只要矛盾仍会影响推理，就应保留它，而不是过早解决或将其磨平。每则笔记都保留指向来源的链接，让我们可以从想法追溯到证据。

5.5

产出与决策

主题笔记揭示出一个明确问题：现场作业结束时必须填写哪些信息，哪些可以稍后补充？

现在，有三个人在等待我提供某样东西。

- 一位需要我访谈的技术人员，
- 一个需要作出决定的委员会，
- 一个需要我进行说明的团队。

我不会做三次调查：我会三次取用同一批笔记。

这是整场会话中唯一没有技能帮助我的时刻：产出交付物是一项写作工作，而不是一种重复动作。

三次取用同一批储备

从笔记到交付物，并不是复制。每个结果都会选择对自己有用的积木，并根据接收者来排序和重新表述。

共同积木	访谈指南	概述	团队演示
ATO00001——过于繁琐的填写要求可能使作业报告失去信息量	询问哪些字段遭到规避，以及如何规避	描述质量损失风险	展示一个现场阻力的示例
ATO00002——数据运用需要结构化数据	询问哪些数据确实得到了复用	确立比较与自动处理的需要	解释团队期待获得的价值
THM00001——质量同时取决于填写与运用	围绕两方面要求组织访谈	表述关键问题与取舍标准	构建叙事线索

共同积木	访谈指南	概述	团队演示
C-001——在不损害填写质量的情况下实现结构化	把张力转化为需要核实的问题	把它保留为有待解决的问题	让它保持可见，但不提前宣布解决方案

这样，同一批储备产生三种组合；没有任何交付物包含全部内容，也不会取代为它提供材料的笔记。

这种分离最大的好处，是让我只需修正一次知识。

- 如果新来源为 ATO00001 增加了限定，今后的交付物会从这块经过补充的积木重新出发，而不会继续复制散落在多份文档里的旧表述。

从最紧急的事情开始：下一次访谈

我的第一个需要，是准备与技术人员的访谈。 交付物从草稿区域开始。

```
05-outputs/
└── _draft/
    └── questions-prochain-entretien.md
```

```
# 下一次访谈的问题

1. 现场作业期间，哪些字段最拖慢你？
2. 要求的信息无法获得时，你会填写什么？
3. 哪些信息可以在现场作业结束后补充？
4. 哪些数据必须立即准确，才能为运营所用？
```

笔记努力保存一个可以复用的想法。交付物则选定接收者、形式和预期行动。

完成审阅与确认后，文件会离开_draft/，直接进入 05-outputs/。 这种移动表示成熟程度的变化，而不是文档类型的区别。

到目前为止，我只是在产出：为三类受众，把已经知道的内容转化成合适形式。本章接下来的内容，属于一种完全不同的动作。到了某个时刻，我们必须停止复述并作出选择——而作出的选择，必须在六个月后仍能找到。

决策记录保存塑造结果的选择

“编写规格说明前先访谈一位技术人员”，并不是对最终交付内容的取舍。

- 如果这次访谈只是下一步行动，它属于 `01-state/todo.md`。
- 如果所有探索工作都必须遵循这条规则，它会成为 `00-frame/directives.md` 中的一条指令。

两种情况下，它都不需要一则决策记录。

当工作呈现出多个可能方向，而我们必须选择其中一个来塑造产品、文本、方法或上下文的另一项结果时，决策记录才会出现。它会保存我们决定设计或交付什么、放弃了哪些替代方案、作出选择的理由，以及接受了哪些后果。

因此，在我的示例中，取舍与如何组织访谈无关。 它涉及一项功能决定：

- 要求现场作业结束时填写所有信息，
- 只保留自由文本，
- 或者允许稍后再补充作业报告。

一则有用的决策记录，不会把选择的核心留给实施者自行决定。三个人阅读后，应当理解同样的范围、同样的预期行为与同样的边界。不影响结构的技术细节可以保持开放，但任何细节都不能悄悄改变决定本身。

```
04-notes/
└── decision-records/
    └── DRE00001-limiter-la-cloture-a-quatre-informations.md
```

下面是写成后的样子。模板很长，因为它保存了让决定能够被理解和重新审视的完整推理——熟悉决策记录的读者可以直接跳到代码块之后的说明。

```
---
uuid: 0d5efbe2-1eac-4ad7-94c1-29f68adcf001
id: DRE00001
created: 2026-09-11
updated: 2026-09-11
status: accepted
decided_by: 产品团队
decided_on: 2026-09-11
```

```
supersedes: null
superseded_by: null
sources: [ATO00001, ATO00002, THM00001]
---

# 将现场作业结单所需的必填信息限制为四项

## 决定

一项现场作业要被视为已经结单，必须恰好具备以下四项信息：

1. 相关设备，已在现场作业单上注明，或在结单时填写。
2. 现场作业实际结束的日期和时间。
3. 现场作业结束时的设备状态：已恢复运行、维持降级运行，或仍然停用。
4. 一段非空的简要说明：说明已完成的工作，或说明设备未能恢复运行的原因。

任何其他信息都不应成为结单条件。

结单之后，可以继续补充故障性质、已识别原因、已完成工作性质、更换的零件和设备不可用时长，而不影响现场作业的结单状态。
技术人员可以填写这些补充信息，但运营负责人必须在二十四小时内补充或确认。

尚未完整的作业报告仍然可以查阅。
它会被标记为仍待补充；二十四小时后，系统会提醒运营负责人。

## 背景

技术人员必须能够在现场为作业结单，而不必先填写完整的分析表单。
不过，运营部门仍需要可比较的信息，以分析故障、故障原因、完成的工作、更换的零件和不可用时长。

## 假设

- 现场作业单通常会在工作开始前标明设备。
- 结单时，技术人员知道设备状态，也能简要描述完成的工作。
- 运营负责人拥有必要的时间和信息，可以在二十四小时内确认完整报告。

## 替代方案

- 要求在结单时填写九项信息。
- 只保留一段自由文本，之后再自动提取现场作业信息。
- 将所有信息设为选填，只在事后检查质量。

## 理由

四项即时信息足以识别现场作业、记录结束时间、了解设备状态，并理解已经完成的工作。
五项分析信息仍然必要，但不应阻碍技术人员结束现场工作。
第二阶段明确把补充或确认作业报告的责任交给运营负责人。

## 依据

- ATO00001 描述过度限制填写所导致的规避行为。
- ATO00002 确立可比较数据的需要。
- THM00001 连接这两方面要求。
```

```
## 后果

### 积极影响

- 结单只要求四项能够立即知道的信息。
- 作业报告补充完整后，现场作业分析所需的信息仍然可用。
- 缺失数据的责任清晰可见。

### 取舍

- 现场作业完成后的二十四小时内，相关指标可能仍不完整。
- 运营负责人将承担额外的确认工作。

### 风险

- 不完整的作业报告可能不断积累。
- 事后添加的信息，可能不如现场记录那样忠实反映实际情形。

## 合规标准

- 四项即时信息缺少任何一项，现场作业都不能结单。
- 即使五项补充信息一项都没有，现场作业仍可结单。
- 修改补充信息不会改变现场作业的结单状态。
- 在所有使用场景中，作业报告都显示为“仍待补充”或“完整且已确认”。
- 如果作业报告尚未完整并得到确认，结单二十四小时后提醒运营负责人。

## 相关决定

- 解决 C-001——在不损害填写质量的情况下实现结构化。

## 来源

- SRC00001
- SRC00002
- SRC00003
```

这里，accepted 状态表示取舍已经获得确认，而且足够精确，可以指导实施。

- 只要预期信息、责任或合规标准仍未确定，决策记录就应保持 proposed 状态。
- 一旦接受，它就作为历史记录固定下来，不再悄悄变化。

如果一项新决定取代了它，旧决策记录仍然可以查阅，两个文件通过 supersedes 与 superseded_by 彼此连接。

这种连续性保存了完整的取舍时间线：当时决定了什么、依据哪些信息，以及后来什么因素导致我们作出不同选择。

因此，六个月后，我找回的不只是当前规则。我还能理解为什么最初的方向曾经显得正确、什么让它变得不够，以及下一项决定是如何形成的。

没有这条时间线，一连串选择最终会显得前后矛盾。有了它，这些选择重新成为一个产品演进过程中可以理解的历史。

- 替代方案显示我们放弃了什么。
- 取舍让牺牲了什么变得可见。
- 依据则把决定与支持它的知识和证据连接起来。

产出不会耗尽我们知道的内容

访谈指南、主题笔记和决策记录都在处理同一个问题，却并不重复，因为三者承担不同功能。

- 笔记保存上下文认为自己知道的内容。
- 决策记录保存一项选择及其理由
- 指南则把知识转化为面向特定对象的行动。

另一份交付物可以用不同形式再次使用同一批共同知识。**因此，产出不会掏空上下文的内容；它只是揭示这些内容能帮助我们做什么。**

接下来，我还要记录已经达到的状态，让下一次会话不必重建走过的路径就能重新开始。

要点——决策记录

同一批笔记支持多份交付物，没有任何交付物会消耗它所使用的知识。决策记录保存塑造最终设计或交付内容的取舍，而不是一项待执行的行动。它保存被放弃的内容与接受的牺牲，使决定仍可重新审视。

5.6

保存与恢复工作

决策记录刚刚通过，三分钟后又有一场会议开始。我既没有完成研究，甚至也没有完成一个看起来像阶段终点的步骤。

这才是通常情况：另一项任务需要我处理，电脑必须重启，我要更换电脑，或者 AI 的上下文窗口变得过于拥挤。

- 因此，保存并不是为了回应工作本身的某种需要，例如“研究已经结束”。
- 它是一种技术保障：上下文的状态必须能够在产生它的会话结束后继续存在。

我必须能够现在停下来，日后重新打开，并准确找回离开时的工作状态。

一项技能会负责此事，而且必须在几秒内完成：如果保存代价很高，人们就不会在匆忙时执行它，而那恰恰是它最有用的时候。

AI 固定我们在这一刻知道的内容

`01-state/state.md` 文件概括那些已经足够稳定、可以指导后续工作的内容。

```
contexte-qualite-comptes-rendus/
└── 01-state/
    └── state.md
```

保存时，它可能采用以下形式：

```
# 当前状态

## 已稳定的知识

- 过于繁琐的填写要求可能使作业报告失去信息量。
- 但大规模运用数据仍然需要结构化数据。

## 已确定的决定

- 现场作业结单需要相关设备、结束日期和时间、设备状态，以及完成工作的简要说明。
- 故障性质与原因、工作性质、更换的零件和不可用时长会在结单后补充，并由运营负责人在二十四小时内确认。

## 主要不确定性

- 衡量这种分阶段填写，是否确实能缩短结单时间，同时不降低数据质量。

## 知识欠账

- 尚未通过任何试点衡量结单时间，也没有测量二十四小时后完整作业报告所占的比例。
```

- 已完成的任务可以从待办事项中消失。
- 尚未解决的矛盾则继续显示在状态中。

知识沉淀只包含工作已经充分稳定下来的内容。 状态会被重写以描述现在，知识沉淀则逐渐积累，以保存已经获得的理解。

为未来的自己准备一项行动

工作交接文件不会讲述全部历史：它只为下一个有用动作做好准备。

```
contexte-qualite-comptes-rendus/
└── 01-state/
    └── handoff.md
```

`01-state/handoff.md` 文件可能采用以下形式：

```
# 工作交接

## 当前状态

DRE00001 已接受，C-001 已解决。
四项结单信息和五项补充信息已经确定。

## 注意事项

不得在结单条件中增加新的必填信息，除非新建一则取代 DRE00001 的决策记录。

## 下一步行动

开始实施 DRE00001。

## 仍未解决

- 确定向技术人员展示的三种现场作业后设备状态标签。
- 准备衡量结单时间，以及二十四小时后完整作业报告所占的比例。
```

有用的工作交接只指定一项下一步行动。如果列出十种可能，下一次会话就不得不重做我刚刚完成的决策工作。

留下一份记录，用来比较每次恢复时的状态

`01-state/checkpoints.md` 文件保存一系列简洁的状态快照。

```
contexte-qualite-comptes-rendus/
└── 01-state/
    └── checkpoints.md
```

文件可能采用以下形式：

```
## 2026-09-11

- 来源：3
- 原子笔记：2
- 主题笔记：1
- 开放矛盾：0
- 已接受的决策记录：1
- 下一步行动：开始实施 DRE00001。

## 2026-09-04

- 来源：1
- 原子笔记：1
- 主题笔记：0
- 开放矛盾：0
- 下一步行动：了解运营部门的看法。
```

这些快照让我无需比较所有文件，就能看到上下文的演变。它们也帮助我发现停滞，或发现一项正在增长而我尚未察觉的欠账。

AI 记录发生了什么，而不是它如何看待这些事

历史会记录重要事件，却不试图重建每一次交流。在这个示例中，当天的事件被添加到 `02-history/2026-09-11.md`。


```
contexte-qualite-comptes-rendus/
└── 02-history/
    └── 2026-09-11.md
```

加入的内容块可能采用以下形式：

```
## 2026-09-11

- 添加 SRC00003-livre-blanc-donnees-intervention。
- 创建 ATO00002 和 THM00001。
- 打开 C-001。
- 确认`questions-prochain-entretien.md`。
- 接受 DRE00001，并解决 C-001。
```

- 状态说明我们进展到哪里。
- 历史说明发生了什么。
- 工作交接说明下一步做什么。

让机器检查我看不到的问题

保存时，一些机械检查可以减少遗漏。 检查结果写入 `01-state/health.md`。

```
contexte-qualite-comptes-rendus/
└── 01-state/
    └── health.md
```

- 所有被引用的文件都存在吗?
- 标识符都唯一吗?
- 每则笔记都注明来源了吗?
- 已添加但尚未分析的来源得到标记了吗?
- 被取代的决定与取代它的决定建立联系了吗?
- 下一步行动明确吗?

这些检查无法取代我的判断。 它们只是防止结构错误冒充思考问题。

完整循环，以及它允许我们做什么

现在，整个过程已经清晰可见。

```
打开
  ↓
读取收件箱
  ↓
加载使命、状态与工作交接
  ↓
添加来源
  ↓
创建或更新笔记
  ↓
产出草稿
  ↓
检查、确认或决策
  ↓
更新状态、知识沉淀与工作交接
  ↓
检查点与历史
  ↓
保存检查
  ↓
下一次打开
```

我不会等到这条链条走完才保存。

- 添加来源后、创建笔记途中、作出决定前或准备交付物时，只要发现上下文窗口快要爆满，我就会保存……

这正是保存的作用：**无论在哪种状态中断，都可以恢复工作，而不依赖当前对话。**

因此，我的会议可以开始了。

上下文不只是一棵文件目录树：它是一套支持长期思考的系统，让材料进入、保存张力、稳定想法、产出多种结果，并为恢复工作做好准备。

它的形态有意保持简单和便携——只需可读文件，就足以让工作可以观察、核实和传递。

理解这台引擎后，又会出现另一个问题：上下文如何与存在于自身之外的工具、知识库和真实产品对话？

要点——保存

保存并不标志一个阶段结束；它保障状态能在产生它的会话结束后继续存在。状态说明进展到哪里，历史说明发生了什么，工作交接说明下一步做什么。循环可以在任何位置中断，之后仍可从该位置恢复工作。

部分 6

生态系统中的上下文

一个维护良好的上下文，很容易让人想把所有可用知识都放进去。

这一部分关注它把什么留在外面，以及为什么要这样做。 它保持小巧，才能继续思考；那些只是出于谨慎而保留的内容，会在原件于别处继续变化时悄无声息地过时。 因此，它会与邻居协作：一些工具专门负责保存、测量或执行，还有一些上下文研究与它不同的问题。

6.1

查询结构化知识

作业报告先放一放：今天早上，我在处理同一产品的另一个问题——用户在注册过程中放弃。一家机构刚刚交付研究，我把它加入上下文：多位参与者面对密码规则时犹豫不决。

这个观察让我想起一个已经遇到过的想法：在用户尚未感受到产品价值之前，就要求付出很大努力，可能会让他们离开。

- 我几乎可以肯定，自己拥有相关材料。
- 我只是不再知道它在哪里，也不记得当时用了哪些词。

我当时说的是初始投入、阻力、激活，还是用户引导？ 这个想法也许

- 是从一本书中提取的，
- 是在一则产品管理笔记中展开的，
- 曾与另一条用户旅程建立联系，
- ……

我可以尝试几个关键词搜索，再逐一打开结果。 但我选择向上下文提问：

> 对于用户在感受到产品价值之前被要求付出的努力，我们已经学到了什么？

上下文查询一份更广阔的记忆

答案不在活动的上下文中。 但它可能存在于我的 LLM Wiki 里。这是一个共同知识库，其中已经整理了从书籍、文章、标准和其他文档中提取的数千则笔记。

LLM Wiki 与上下文性质不同。

Wiki 是一个知识库。

- 它根据他人写下的内容保存结构化笔记（在我的实现中，我复用了原子笔记的概念）：定义、原则、研究结果或标准要求。
- 每则笔记都说明知道了什么，以及这些内容来自哪里。

- 它可以包含数万则笔记，历经多年积累，涉及彼此毫无关联的主题。

任何一个上下文都无法容纳这样的数量，这也不是它的职责。

但 Wiki 的能力到此为止。

- 它不会说明这些知识为什么对我今天进行的工作重要，也不会说明我如何使用了它们。
- 它不会把一个想法与一项决定连接起来，也不会把一项决定与我为之付出的代价连接起来。

LLM Wiki 不是思考系统：它可以大规模保存，却不会推理。

推理存在于上下文中。 是它保存事情的来龙去脉：

- 使命是什么，
- 有哪些指令和已经确定的决定，
- 为什么为这项使命保留这个想法，
- 我在什么选项之间作出了取舍，
- 我接受失去什么，
- 哪个矛盾仍未解决。

它之所以能够做到，恰恰因为自己保持小巧——只有几十则可供人重新阅读的笔记。

两者都为自己的优势付出代价。

- **Wiki 获得规模，却失去脉络。**
- **上下文保留脉络，却放弃规模。**

正因如此，二者应当相互沟通，而不是彼此融合。

描述我正在寻找的想法

我不会把整个 Wiki 复制到关注注册流程的上下文中。 我也不会先启动一次 RAG，再执行一次图搜索。

我只是向上下文提出问题：

> 对于用户在感受到产品价值之前被要求付出的努力，我们已经学到了什么？

我使用自己的语言来表达。

- 我无需知道笔记标题、存放位置，也无需记得写下它时所用的词汇。

上下文通过 API 把这个想法传给 LLM Wiki。 其余一切都在 Wiki 内部发生。

Wiki 先按含义搜索，再查看结果周围

LLM Wiki 首先找出含义接近我的问题的笔记。

- 一则笔记可能讨论初期阻力、激活，或用户首次感受到价值前的等待时间，却完全不使用“努力”这个词。

这正是 RAG 帮助它完成的工作：无需使用相同词汇，就能把我的想法与已经结构化的知识联系起来。

然后，Wiki 会查看最佳结果周围的内容。它的语义图会沿着笔记之间已有的关系继续前进，加入能够帮助理解主题的相邻笔记：

- 一个定义，
- 一项限制，
- 一个矛盾，
- 一个互补想法。

这些相邻笔记中的一则，也许永远不会直接回答我的问题。但它之所以变得宝贵，是因为它说明这个想法在什么条件下适用，或从哪里开始不再成立。


```
一个用我自己的语言表达的想法
                    │
                    ▼
Wiki 按含义找出相近笔记
                    │
                    ▼
加入有用的相邻内容
                    │
                    ▼
返回少量带有出处的参考
                    │
                    ▼
上下文保留服务于自身使命的内容
```

RAG 找回的是我不知道该如何命名的内容。 图会揭示出那些我没有想到要寻找的内容。

对我而言，这两个操作合而为一。 我请求能回应某个问题的参考；LLM Wiki 搜索这个想法及其相邻内容，再返回一份简短清单。

上下文收到的是参考，而不是现成答案

Wiki 返回的是已有笔记，连同笔记内容、出处和关系。它不会生成一份悄悄混合多个结果的综合答案。

因此，上下文可以阅读每项建议，与注册流程中的观察进行比较，并在需要核查证据时返回来源。

- 有些笔记确实讨论用户进入产品前被要求付出的努力。
- 另一些讨论的是更晚提出的要求，那时用户已经知道产品的价值。

与我的问题相近，并不足以让这些笔记彼此等同。

上下文根据自身使命进行筛选。 它可以：

- 保留一个能够解释注册过程中放弃现象的想法，
- 保存一个矛盾，
- 排除一则过于宽泛的笔记。

一则被保留的笔记会连同出处及其所依据的来源进入上下文。 其余内容则留在 Wiki 中。

这样，我可以利用一份广阔得多的记忆，却无需把每个上下文都变成整个知识库的副本。

- 我不必先学会这份记忆如何建立索引才能使用它：只需描述要寻找的想法，再核查它带回的参考。

这条链路让我们能够访问已经结构化的知识。

- 但它仍然没有告诉我们，产品今天实际上在做什么。

要回答另一个问题，上下文必须离开知识库，直接查询代码、测试和运行中的产品。

要点——两个系统，一扇门

LLM Wiki 获得规模，却失去脉络；上下文保留脉络，却放弃规模。 我用自己的语言描述一个想法；Wiki 按含义找出相近笔记，并加入有用的相邻内容。上下文收到少量带有出处的参考，只保留服务于自身使命的内容。

6.2

查询真实产品

我正在重读关于注册过程中放弃现象的研究，这时一个问题出现了：

> 受邀用户是否也必须像其他人一样创建密码？

问题看起来微不足道。 但一位客服人员正在等待答案，以便回复客户。

我可以：

- 中断阅读，
- 打开测试环境，
- 准备两个账户，
- 重新走一遍流程，
- 比较发生的情况。

我过去就是这样做的。

- 半小时后，我大概会得到答案。
- 但也会失去当前工作的脉络。

这一次，我把问题交给上下文。

上下文去查看产品

上下文正在使用用户研究产生的笔记，并把研究本身保存为相关论断的来源。

- 它也承载当前使命。
- 它理解这个问题为何重要，但这些内容无法告诉它产品现在的实际行为。

我不再维护一份试图长期描述这种行为的规格说明。

- 产品变化太频繁，单独放在旁边的文档不可能始终忠实地复制它。

唯一保持最新的事实依据，是已经交付并在生产环境中运行的东西：应用程序的源代码。

在我的系统中，上下文之外的技能懂得查询源代码并生成逆向文档。上下文会调用负责回答产品问题的技能，并把我的请求传给它。

```
产品经理的问题
        │
        ▼
      上下文
        │  调用
        ▼
  产品读取技能
        │  查询
        ▼
     真实产品
```

技能首先在代码中找出邀请流程。然后，它沿着这条流程追踪到账户创建规则。

代码显示程序设定的行为。一项测试确认，受邀用户可以通过收到的链接完成注册，而不必立即创建密码。配置则显示，这条流程没有启用任何客户专属变体。

技能返回一份功能层面的答案：

> 受邀用户可以通过邀请链接访问账户，在这一阶段无需创建密码。如果之后选择创建密码，则适用普通规则。测试确认了邀请流程与主动创建密码的流程。 忘记密码的找回流程没有检查。

我可以把前两句话转给客服团队。如果想与开发人员一起核查分析，也可以打开技能提供的参考。

最后，我还可以把这份答案连同证据、核查日期和注明的局限写入上下文。

- 回复客服团队，只解决一个案例，到此结束。
- 将内容沉淀进上下文，则会把一项经过核实的产品事实带入使命的*想法世界*：下一次关于邀请流程的问题不必从零开始；研究所产生的笔记也获得了单凭观察无法提供的信息——每条流程究竟在什么时候要求输入密码。

答案的价值来自随附的证据

技能并不满足于生成一个看似可信的答案。对于每项论断，它都会说明查阅了什么：

- 代码中的规则，
- 验证行为的测试，
- 决定当前启用哪种变体的配置。

这份记录可以防止一个自信却错误的回答。 它也会立即揭示出缺少的内容。

这里，技能没有检查忘记密码的找回流程。它会明确说明，而不是把结论延伸到自己没有读过的流程。

根据问题的不同，还可能需要其他要素。

- 代码包含各种可能规则，但功能开关会说明哪一条正在启用。
- 数据库结构显示产品可以记录什么，而在获得授权后读取的数据则可以帮助理解某个具体账户的情形。
- Git 历史与拉取请求可以帮助找回某条规则何时改变，以及与这次变化一起保留下来的解释。
- ……

技能只会汇集回答当前问题所需的内容。它不会吸入整个代码库，也不会把每个文件都变成上下文的来源。

答案仍然便于客服人员、销售人员或产品经理阅读。证据则留给需要核查答案或继续调查的人。

找到规则，并不意味着决定改变它

我回到关于注册的研究。它已经让我知道，两条流程并非在同一时刻要求创建密码。

技能补充了研究无法观察到的内容。受邀用户并没有绕过密码规则：他只是稍后才遇到它，而且届时适用的规则完全相同。

因此，阻力并没有为他消失，只是换了位置。 这个细节改变了我的问题。我不再寻找谁需要遵守规则，而是关注每个人在什么时候遇到它：感受到产品价值之前，还是之后。

代码告诉我产品在做什么。

- 但它不会告诉我这条规则是否合理、是否真正保护账户，也不会说明它给用户带来的代价是否可以接受。

要作出决定，我仍需要上下文中的其他要素：

- 研究中的观察及其适用范围；
- 安全目标；
- 当初选择这条规则的理由；
- 改变规则可能带来的后果。

技能确立一项关于产品的事实。 上下文把这项事实与已有笔记和决定连接起来，同时保留能够核查其出处的来源。 **作出取舍的责任仍由我承担。**

如果我决定改变密码创建方式，决策记录会保存选择、适用范围与支持理由。 实施以后，技能可以重新查看产品，核实新行为是否确实符合决定。

因此，这个过程可以双向运作。 产品帮助思考，决定再改变产品。

```
观察  —>  想法  —>  决定
 ▲                   |
 |                   ▼
已核实产品  <——  修改后的产品
```

6

保存“为什么”，重建“是什么”

以代码为中心的方法

代码不再只是开发的最终结果。它成为一种来源，我们可以据此重建对产品当前行为的描述。 上下文保存代码无法独自解释的内容：来源、推理、决定、放弃的替代方案与选择理由。

这种分离改变了文档的位置。

我会保存解释产品为何走向某个方向的内容：访谈、约束、假设和取舍。而描述产品当前行为的内容，我会从产品本身重新构建：

- 给客服团队的回答，

- 一份功能文档，
- 一份变更日志，
- ……

我不再手工维护同一种行为的两个版本：一个在代码中，另一个在文档里——而文档从发布那一刻起（其实早在发布之前）就开始老化。

不过，并非一切都来自代码。

- 用户研究不在代码中。
- 合规义务拥有自己的来源。
- 一项决定也不能被简化为它最终产生的行为。

代码承载已经实现的内容。上下文保存“为什么”，并在情形变化时让我们能够重新讨论。

六个月后再回来

提供给客服团队的答案，与今天检查的产品版本保持关联。它不会自动描述后续版本。

- 六个月后，邀请流程可能已经改变。
- 功能开关可能启用了另一种变体，或出现了新的登录方式。

上下文会保存问题、功能层面的答案、核查日期、所检查的版本、引用的证据与指出的局限。 这样，我可以找回当时进行推理时我们知道的内容。

要了解当前行为，我只需让技能重新进行核查。

- 它会重新从真实产品出发，而不是复制旧答案。

于是，我可以区分两个经常混在一起的问题：

- 当初为什么作出这项决定？
- 产品今天在做什么？

上下文通过保存推理回答第一个问题。 技能通过回到真实产品本身回答第二个问题。

因此，产品不会取代上下文，上下文也不声称自己包含产品。

- 一方提供可以核实的现实；

- 另一方保存赋予现实意义并推动其演变所需的知识。

这种流动并不止于外部工具。 当多个上下文分别处理同一产品的不同方面时，它们的知识也必须能够相互找到，而不必合并到同一个文件夹里。

要点——以代码为中心的方法

不再把任何规格说明当作产品的长期描述来维护：描述产品行为的内容从代码中重新构建。 一个答案的价值，来自随附的证据与它明确指出的局限。 代码承载已经实现的“是什么”；上下文保存“为什么”，并允许我们重新讨论。

6.3

我拥有三个上下文的那一天

一位技术人员告诉我，因为作业报告中缺少一项信息，他不得不再次打电话给客户。

我记下这句话。 然后停了下来，因为不知道该把它放在哪里。

- 我有一个关于作业报告质量的上下文——本书后半部分一直围绕它展开。
- 我还有第二个关于现场作业本身的上下文：它包含什么、受哪些规则约束，以及关于它已经作出过哪些决定。
- 几周前，我又打开了第三个范围更大的上下文：如何让操作人员承担起 1 级现场作业。

技术人员的这句话同时与三者有关：

- 它记录了一份有缺陷的作业报告，
- 它说明了现场作业的某些情况，
- 它也帮助理解操作人员的问题。

这一刻，问题的性质改变了。

- 只有一个上下文时，问题是：它属于这里，还是偏离主题？
- 有了三个以后，问题变成：它属于哪一个？

两种犯错方式

第一种诱惑，是把一切汇集起来。

- 一个巨大的空间，所有知识都放在同一个地方，再也不必考虑如何归类。

这恰恰是我一直试图逃离的东西。一个容纳一切的空间，会重新变成文档系统一直以来的样子：在这样一个地方，

- 信息确实存在，但每次查阅都要重新筛选；
- 各项目标彼此混合；
- 来源不断堆积；
- 再也没有哪则笔记拥有明确的归属者。

我在本书开头已经讲过这条死路；换一套新词重新制造它，并不会让它变得更好。

第二种诱惑正好相反：让每个上下文都只待在自己家中。每一个都整洁、有边界、前后一致——却对彼此视而不见。

后果很快就会显现。 我关于作业报告的上下文用了三次会话才稳定一条规则，而现场作业上下文早已拥有这条规则。这不是看得见的重复内容，而是在无人察觉的情况下重做了一遍工作。

因此，我们需要第三条路，而它始于弄清每个上下文承载什么。

名称前的一个数字

我的上下文分别叫作 `01-saisie-compte-rendu`、`02-intervention`、`03-autonomie-operateurs` 和 `04-exploitation-catch-all`。

这个数字不是排名，而是一项命名约定。甚至不用打开，它就能说明上下文承载什么。

- **`02-`表示一个产品实体**。它是生产环境中运行的软件组成部分，拥有自己的规则：ERP 中的 `02-facturation`，或我的系统中的 `02-intervention`。其中的内容已经确立。这个上下文不会关闭。
- **`01-`表示关于某个实体的一项假设**。`01-saisie-compte-rendu`：我认为填写方式应该改变，但还不知道怎样改变。它包含的内容没有一项已经确定，因此我创建一个 `01-`类型的上下文，以便明确区分稳定内容（生产环境中的内容）与假设。
- **`03-`表示一个问题**。`03-autonomie-operateurs`：让操作人员承担 1 级现场作业。这不是一项功能，它跨越多个实体，也会催生若干 `01-`。
- **`04-`表示综合收集区**。听到的一句话、一份 PDF、一个直觉：这些材料所属的主题尚未拥有上下文。如果没有它，每个想法片段都得创建一个上下文——我最初就是这样做的，结果短短几周内就有了五十多个缺乏内容的上下文。

`01-`最终总会离开活动工作

新填写方式上线的那一天，它不再是一项假设，而成为现场作业的一部分。

现在能够描述产品的内容会被提交给 02-intervention。其余内容仍留在探索上下文中，随后整体归档。不会再有两个活动位置都声称描述作业报告的实际行为。

一项没有实现的假设，会连同阻止它实现的原因一起归档。这不是需要抹去的失败，而是防止我们两年后重走同一条路的依据。

04-则以相反方式清空。当一个问题真正浮现时，我会创建对应的上下文，然后回到综合收集区，查看它过去保存的相关内容。没有用的东西会被删除。

所有这些都只是为了解决一个问题——我把它放在哪里——方法是把正在研究的内容与已经确定的内容分开。

- 如果没有这种分离，每项被放弃的假设都会在产品描述中留下痕迹。
- 而在一个一切都理应为真的地方，人们也不敢再提出任何假设。

我用了几个月才看见的错误

我最初创建的是 03-tarification。当时看起来理所当然：这是一个大主题，涉及多个实体，而且长期存在。

六个月后，其中包含：

- 价格表，
- 两份竞争对手基准分析，
- 一场关于折扣的讨论，
- 一张涉及错误账单的支持工单，
- 一则关于比利时增值税的笔记，
- 三张竞争对手界面的截图，
- ……

其中没有任何内容偏离主题：它们确实都在讨论定价。

问题恰恰在这里。要拒绝一份材料，必须能够说“这对我没有帮助”。但这个上下文没有在寻找任何东西——**它没有要回答的问题，只有一个标签。**

而 03-autonomie-operateurs 有一个问题：如何让操作人员承担起 1 级现场作业。

因此，03-不是一个宏大主题，而是一个问题。定价并没有因此消失：它横跨我的多个上下文——02-facturation 承载相关规则，某个 01-也可能依赖它——但它不会统辖任何一个上下文。

所以，我的约定并没有为领域设置分类，也没有为许多其他事物设置分类。它只负责安置那些位置显而易见的内容，仅此而已。其余内容在综合收集区等待；日后寻找时，描述要找的东西，而不是努力回忆当初把它放在哪里。

事实上，我有两个综合收集区，这是犯错后学到的结果。只有一个时，来自产品工作的内容与来自工业运营领域的内容混在一起——两套词汇、两类受众、两种命名同一对象的方式。它变得难以阅读的速度，比被清空的速度更快。

- 04-exploitation-catch-all 接收客户业务领域的内容，即运营；
- 04-productops-catch-all 接收与我的职业及其组织方式有关的内容，即产品运营。

两者都保持可读，因此也都能逐渐清空。

技术人员的那句话则不需要经过综合收集区。它进入了 02-intervention，因为无论我此刻在思考什么问题，那里存放的都是关于产品的确定事实。

剩下的问题是，我的三个上下文如何得知这件事。

要点——四种前缀，一个问题

名称前的数字说明上下文承载什么：02-表示一个正在运行的实体，01-表示关于该实体的一项假设，03-表示跨越多个实体的问题，04-则是存放尚不知如何归类内容的缓冲区。01-的存在，是为了向相应的 02-提交现在能够描述产品的内容，然后归档。如果没有实现，也会连同阻止它的原因一起归档。一个领域不是一个上下文：它可以横跨多个上下文，却不会统辖任何一个。没有明确问题的上下文，永远无法判断什么偏离主题。

6.4

去邻居家阅读，但不搬进去

我重新打开 01-saisie-compte-rendu，这个上下文关注作业报告的质量。 要继续推进，我需要知道现场作业本身的规则：技术人员可以填写什么、什么是必填项，以及哪些问题已经有了决定。

这些规则并不存放在我的上下文中。 它们稳定地存在于邻近的 02-intervention 中；后者描述产品中实际存在的内容。

我可以把这些规则复制过来，放在手边。但六个月后，我很可能会得到同一条规则的两个版本，却没有简单办法知道该以哪一个为准。

因此，我先做一个更轻量的动作：我的上下文声明自己读取 02-intervention。

读取让知识留在其归属者那里

这项关系写在我的上下文的稳定框架中，与使命和指令放在一起。它开放一项明确权限：查阅 02-intervention 已经知道的内容。

材料仍留在其归属者那里。它继续在那里演变，我的探索可以使用它，却不会声称拥有它。

这样，我可以核查当前的必填字段规则，而无需在每项涉及作业报告的研究中都复制一份。 我也可以回到建立这条规则的来源与决定。

读取并不授权修改。如果一则笔记在我看来不完整或错误，我的上下文可以提出建议，却永远不会直接修改邻居。

第一种关系回应一种常见需要：使用外部知识，同时让它继续在负责它的位置发展。

但它无法涵盖所有情况。

吸收让知识变得自主

有时，只是查阅还不够。

我可能需要固定一项决定所依据的确切规则。我可能想根据探索的特定范围重新表述它。如果原始上下文被归档、移动或重新组织，我也可能仍需这份知识继续可用。

这时，我会选择吸收它。

吸收会让一份知识进入我自己的上下文。我可以原样接收一则笔记，也可以使用自己的词汇重新表述。根据需要，我也会复制让笔记能够得到核实的来源，或保留指向来源的引用。

但副本不会因此变成一个没有过去的想法。它会保留原来的上下文、吸收日期，以及重新表述过的内容。

因此，差别与其说在技术动作，不如说在动作带来的责任：

- 当我读取时，知识继续属于邻居；
- 当我吸收时，一个自主版本进入我这里，其使用责任也转交给我。

读取产生依赖。 吸收产生新的责任。

流动可以转向相反方向

我的探索不只使用 `02-intervention` 已经知道的内容。它也会产生观察、假设，有时还会产生持久知识。

研究作业报告质量时，我可能发现，某项填写规则其实属于现场作业本身。这份知识不应继续封闭在一个最终可能被归档的探索上下文中。

我不会直接把它写进 `02-intervention`。 我会向后者提出建议。

流动方向发生了变化：

```
01-saisie-compte-rendu ── 读取 ──▶ 02-intervention

01-saisie-compte-rendu ── 提交建议 ──▶ 02-intervention
                                        │
                                        └── 决定是否吸收
```

产生知识的一方，不会决定知识最终归属哪里。接收知识的上下文会根据自己的使命进行评估。

它可以：

- 原样接收笔记；
- 使用自己的词汇重新表述；
- 当两种理解都依然有效时，保留差异；

- 如果对探索有用的内容并不描述现场作业本身，则拒绝吸收。

拒绝并不表示知识流动失败。 它防止局部结果悄悄成为普遍真理。

当探索变成产品现实

几周后，新的作业报告填写方式上线。 `01-saisie-compte-rendu` 完成了自己的使命：原本只是假设的内容，已经成为产品现实。

但我不会因此把它的全部内容都移进 `02-intervention`。 我会回顾探索产生的内容，并分成两组。

第一组包含现在能够描述或解释产品的内容：

- 确立新填写规则的笔记；
- 用来命名这些规则的稳定词汇；
- 结构性决定与作出这些选择的理由；
- 上线后仍未解决的矛盾；
- 核实这份知识所需的来源或引用。

这些要素会被提交给 `02-intervention`。 稳定上下文不会盲目堆积它们：它会丰富已有笔记，重新表述必要内容，并且只有当想法确实全新时才创建新笔记。

在我们的示例中，把结单条件限制为四项信息的规则，会连同解释它的决定一起进入 `02-intervention`。 它现在描述的是产品现实。

第二组讲述探索过程，却不描述当前产品：

- 被放弃的假设；
- 原型与被排除的方向；
- 中间草稿；
- 探索特有的使命；
- 帮助理解所经路径的讨论。

这些内容留在 `01-saisie-compte-rendu`。 它们并非已经无用：它们解释尝试过什么、为什么拒绝某些方向，以及如果没有新理由就不应重做什么。

完成筛选后，探索上下文会整体归档。它仍然可以阅读，日后也可以恢复其中的工作，但不再声称描述当前正在进行的工作。

如果探索没有实现，它同样会连同放弃原因一起归档，而不会向 `02-intervention` 强加任何内容。

因此，上线并不会合并两个上下文。它会触发一次回顾：现在能够描述产品的内容进入稳定上下文；讲述探索过程的内容则留在归档的上下文中。

这种筛选同时保护两边：

- 探索仍可自由犯错；
- 稳定上下文只接收它自己认可的内容。

关系是框架的一部分

读取、吸收和提交建议，不能只是隐含动作。

上下文的稳定框架会说明：

- 它可以查阅哪些其他上下文；
- 这种查阅允许做什么；
- 一份知识可以根据哪些规则被吸收；
- 如何保存出处和可能发生的重新表述。

这种治理不需要沉重的基础设施。 它只是为关系提供一个地址，让出处、责任与边界保持可见。

因此，网络并不是通过让一切彼此相连而建立的。每当一项工作确实需要一个邻居，它才会一条关系一条关系地形成。

- 去邻居家读取，可以避免复制它知道的内容。
- 吸收可以让那些需要在我这里生存的内容变得自主。
- 提交建议则让邻居自己决定，是否接纳我的探索所产生的内容。

这种流动在纸面上看起来十分整洁。但它也有代价：引用可能在没有通知的情况下老化，副本可能失去出处，两个上下文也可能保留不同理解。

下一章将探讨这些问题。

要点——三种动作，三种责任

读取是查阅仍属于归属者的知识；吸收则在接收知识的上下文中创建一个自主且带有出处的版本。 探索会提交自己学到的内容，但是否接纳，由目标上下文的归属者决定。 上线不会合并文件夹：稳定上下文吸收现在能够描述产品的内容，探索则在归档前保留其余一切。

6.5

知识流动的代价

六周过去了。

我对作业报告的研究已经准备送审，这时有人发现一处差异。我所依据的规则——规定结单时哪些字段必填的那条规则——已经不再是产品采用的规则。

它发生了变化。在我工作期间，另一项探索取得结果，现场作业上下文吸收了它。

没有人告诉我。我也没有重新读取，因为六周前已经读过，而没有任何信号提醒我应该回去查看。

上一章提出了一条看起来很干净的规则：沿读取方向建立引用，沿吸收方向进行复制。 我把它当成一项已经解决的取舍。

但它并没有消除风险。它只是转移了风险，而我们最好知道风险落在了哪一边。

看不见的依赖

刚刚发生的事情有一个名称：盲目依赖。

链接是动态的，因此我读取的内容可以在我不知情时变化。这本身不是问题——通常正是我想要的结果。一条规则在归属者那里演变，说明系统在正常运作。

问题在于，我不知道它已经变化。

缺少的东西说起来很简单：没有任何机制监视我读取的内容。我的上下文知道自己依赖谁，却不知道所依赖的内容何时改变。这套机制只完成了一半，而代价恰恰来自缺失的另一半。

复制可以保护我免受这种影响。接收内容的上下文会变得自主：它不再依赖来源是否存在，也不再受来源演变影响。

但作为交换，它会失去别的东西。如果没有说明副本来自哪里，出处就会消失——一年后，我会拥有一则正确的笔记，却再也不知道它依据什么。这正是本书从头到尾建立可追溯性所要避免的情形。

所以，结论不是选择其中一边。 而是明确写出每种动作必须携带什么：

- **引用**要求知道其来源何时发生变化；
- **副本**要求说明它来自哪里——哪个上下文、什么日期，以及过程中重新表述了什么。

可能的演进

另一种尚待实现的答案来自技术：让上下文能够在自身状态变化时通知观察者，尤其是另一个上下文所观察范围发生变化时。上下文仍是一个不断发展的主题，还有许多可能值得探索。

读取不会赋予的权利

一个经常读取的上下文，渐渐会让人产生自己也拥有它一部分的错觉。我有好几次想直接修改邻居的一则笔记——它并不完整，我知道应该补什么，而且只需三十秒。

但规则并不会因此改变：我要告诉它的内容只是一项建议，决定权属于接收方。

删除，但不破坏可追溯性

删除需要一条单独规则，而我是清空一个收集区时学到这一点的。

当一则笔记转移到另一个上下文时，产生它的来源仍留在原处。它们可能因此变得孤立，不再有任何笔记与之关联。

人的本能反应是进行清理。 这个反应只对了一半。

一个已经被阅读并转化为笔记的来源可以删除：它所承载的内容已经在别处继续存在。 一个从未被任何人读过的来源则不可以。

- 它承载着尚未被任何人提取的材料；丢弃它，就等于失去一份甚至从未打开过的内容。

因此，判断标准既不是文件年龄，也不是占用空间。

- 而是读取状态，再结合还有什么内容在引用它。

此外还有一个不可妥协的条件。删除前必须确认，没有任何笔记、词汇条目、草稿或日志行仍在引用这个来源。缺少这项检查，删除规则就会破坏它声称要服务的可追溯性。

差异并不等于错误

最后还有一种情况：两个上下文对同一个主题说法不同。

人的本能反应是把它视为不一致，于是迅速作出判断以恢复秩序。但这种判断往往是错的。

差异可能来自适用范围不同：双方在各自范围内都正确。它也可能是一项仍未解决的假设，一个上下文正在研究，另一个并不知道。还可能是两种同时成立的理解，因为它们从两个不同位置观察同一个主题。

这时，应当让差异保持原样，而是否如此由人决定。系统必须保证的是，我们知道差异存在。

被遗忘的差异，总会在最糟糕的时刻提醒人们：某个人依据其中一个版本行动，却不知道另一个版本的存在。我在结单规则上遇到的就是这种情况，只不过那里并非两种理解，而是同一种理解已经过时。

因此，网络需要付出代价，而我宁愿提前说出它，也不愿在审阅时才发现。连接两个上下文不会消除保持警觉的工作：它只是让这项工作成为可能，并把责任转移到人身上。

归根结底，这正是本书从一开始就在讲述的事情。

要点——网络的代价

引用可能在没有通知的情况下老化，副本可能失去出处：每种动作都有自己的欠账，必须明确写出。读取不授权写入；想对邻居说的内容只能是一项建议，绝不是直接修改。已转化为笔记的来源可以消失，从未读取的来源绝不可以——而删除任何内容之前，都必须核查还有谁在引用它。两个相互矛盾的上下文未必有谁错误：真正严重的是不知道矛盾存在。

合上书之前

收尾

收尾

本书到这里结束，但主题并不会随之结束。

后记会说明这项工作目前进展到哪里、哪些内容只是浅浅触及，以及后续会在哪里发表。 概念渊源则把上下文放在先于它出现或与它相邻的各种方法之间，让你了解它借用了什么，又没有沿用什么。

7.1

后记

谢谢你一路读到这里。

我努力让这本书尽可能易读，但并没有完全做到。其中仍有一些段落比我希望的更加枯燥，有些地方机制压过了故事。你还是坚持读了下来：这份馈赠，我绝不会轻视。

本书只是浅浅触及了这个主题

尽管篇幅不短，本书对上下文的讨论仍然只是表层。

有几个部分在概念层面仍显单薄，我在发表时也清楚这一点。例如原子笔记：根据实现方式不同，我得到的结果差异很大。在上下文中有效的做法，一旦转向 LLM Wiki 就需要调整，因为那里的材料无论数量还是用途都完全不同。

这些部分值得进一步加强。本书没有这样做，是因为一本书终究需要在某处结束，而不是因为这个主题已经完结。

一项将始终留在本职工作之外的项目

上下文会继续发展，但它永远只会是一项业余项目。

这是我在本职工作之外进行的活动，而我已经为它投入了太多个夜晚。我还会回来——我总会回来——但现在该放慢脚步，好好享受生活了。

接下来我想探索的内容

我已经完成了前人研究检索。它找出了许多与我的概念非常接近的思想，有些来自远离产品和知识管理的领域，也常常来自其他语言——尤其是日语和中文；从法语与英语世界望去，这些研究在很大程度上仍不可见。我写了一篇介绍这项工作的文章；之后会发表在博客上。

缺少的是下一步：分析这些成果。

相关人物已经找出，论文也摆在那里，但我还没有阅读。这是数百人用几十年时间完成的研究，探讨的却是我独自在一个个夜晚面对的问题。认真梳理这些成果，比继续独自发明，更能可靠地加强这套框架。

后续不会是第二卷

本书帮助这个概念结晶成形。这就是它的职责：汇集散落的内容，让它可以传递，并赋予它一种足够稳定、能够接受讨论的形态。

在接下来一段时间里，新的发展会发表在 `malorean.net` 上，而不是写成第二卷或本书的新版本。文章可以修正、补充，也可以自我反驳。书籍会把内容固定下来——而这里描述的事物还没有成熟到足以被固定两次。

如果你想继续关注后续，新的内容会写在那里。

7.2

概念渊源与相邻方法

上下文并非出现在思想真空之中。

构成它的大多数机制，都能在知识管理、笔记方法、软件工程、决策支持系统、认知科学或组织记忆等领域找到思想渊源，有些甚至非常古老。

其中一些影响从一开始就是有意识的。另一些则是在方法设计完成后才被发现；当时本书已接近完成，我进行了一次前人研究检索。

因此，需要区分两件事。

影响，是指我原本就知道，并有意识地借用或调整了部分原则的方法或思想。

思想渊源或**相邻方法**，是指更早出现、解决相近问题，有时甚至采用惊人相似机制，却没有直接影响上下文的系统。

这种区分很重要。两个想法可以十分接近，却不表示一个源于另一个。

Zettelkasten：把知识视为可复用单元组成的网络

与 Zettelkasten 的相似之处可能最为明显。

在 Niklas Luhmann 的系统中，一则笔记不会只被归入与当前项目对应的文件夹。它会进入一个由笔记彼此连接而成的持久系统，并可用于新的推理。

上下文非常认同这个想法：一项工作中产生的知识，不应随交付物一起消失。

上下文中的原子笔记显然属于这一传统。

主要差别在于笔记的地位。原子笔记必须保留出处，并进入一套明确区分来源、矛盾、决定、当前状态、知识沉淀与交付物的架构。

因此，在**逐步构建知识资产**这件事上，Zettelkasten 与它十分相似，但无法涵盖整套系统。

PARA：根据信息服务的行动来组织信息

Tiago Forte 提出的 PARA，是上下文有意识接受的影响之一。

PARA 把信息分成四类：Projects、Areas、Resources 和 Archives。它的重要洞见之一是，信息组织应服务于行动，而不是服务于对世界的抽象分类。

上下文保留了这一洞见的一部分。

一个上下文的存在，不是因为某个主题存在，而是因为一个人正在履行一项使命。

所以，它的边界不是："我所知道的关于数据库的一切"。

它可以是："为产品 X 选择一种数据架构"。

差别在于，每个上下文本身都会成为自主的工作单元，拥有自己的状态、来源、知识、决定和生命周期。

领域驱动设计：先有边界，后有内容

上下文这个名称，自然会让人想到领域驱动设计中的限界上下文（**Bounded Context**）。

两者的相似之处主要来自一个想法：信息或概念只有在某个边界内才拥有意义。

同一个词可以在两个不同上下文中拥有不同含义，而我们没有必要把它们合并。

上下文把这一洞见移植到了自己的系统中。

但它不让边界取决于一致的业务模型，而是让边界取决于一项**人的使命**。

因此，两个上下文可以对同一主题保留不同理解，因为它们追求不同目标。

Git：分歧、自主与重新整合

Git 带来的影响更加直接，也更加技术化。

当一项探索变得足够重要时，最好让它在一个单独的上下文中演变，而不是扰动稳定空间。

这种逻辑很像一条分支。

探索可以产生分歧，建立自己的来源和知识，然后把部分结论重新整合进原始空间。

不过，就像 Git 一样，整合并不一定意味着全部合并。

这个洞见也影响了上下文之间的关系：局部自主、允许分歧，以及选择性整合。

项目记忆：保存项目为何如此发展

有关**项目记忆**的研究，是我在设计上下文时尚未意识到的一项思想渊源。

它提出的问题，几乎与一项工作持续数月后出现的问题完全相同：组织很容易保存最终文档，却很难保存解释项目如何、又为何得到这个结果的背景。

为什么作出这项决定？

此前尝试过什么？

当时存在哪些约束？

一个新加入的人继续工作前，需要理解什么？

上下文使用多种相互区分的记忆来回答同样的问题：当前状态、历史、决定、知识沉淀与工作交接。

因此，两者在功能上十分接近。

Experience Factory：交付物并不是资产

Experience Factory 围绕 Victor Basili 的研究发展，并在 NASA 等机构得到实践检验。它带来了更重要的相似之处。

它把执行项目的组织，与将项目经验转化为可供后续项目复用知识的组织区分开来。

这个想法直接对应上下文的一项根本区分：

> 交付物并不是资产。

一个项目可以产出极其优秀的结果，却几乎什么也没有留下。

反过来，项目经验也可以得到分析、转化和保存，从而提高解决后续问题的能力。

上下文把这种沉淀直接整合进工作发生的空间。

Knowledge-Centered Service：在工作中产生知识

Knowledge-Centered Service（KCS）可能是上下文最有意思的相邻方法之一。

它的原则是，编写文档不应成为工作结束后另行进行的活动。

解决问题时，先寻找并复用已有知识。如果知识不完善，就加以改进；如果不存在，就创建它。

这样，工作本身便成为创造和维护知识的机制。

这也正是上下文的一项愿景：

> 每一次使用系统，都应有可能增加它的信息资产。

不过，KCS 主要面向服务与支持环境，而上下文希望为多种使命提供通用结构。

综合集成研讨厅体系：让人、机器与知识共处一个空间

前人研究检索中发现的最令人意外的思想渊源，可能来自中国。

综合集成研讨厅体系（Hall for Workshop of Meta-Synthetic Engineering，HWMSE）围绕钱学森自 20 世纪 90 年代初开始的研究发展起来。

它旨在通过明确结合以下要素来处理复杂问题：

- 人类专家；
- 机器系统；
- 知识系统。

更有意思的是，HWMSE 研究讨论的不只是使用已有知识。它还讨论**知识创造**，以及一个针对所研究问题建立、并在使用过程中不断丰富的知识系统。

因此，它与上下文有很多接近之处：

- 一项面向问题的活动；
- 人机协作；
- 工作中调用的资源与模型；
- 活动产生的知识；
- 不断积累的资产。

不过，HWMSE 并不拥有与上下文相同的操作语法。在那里，我们找不到这种形式的来源、原子笔记、状态、知识沉淀、矛盾、工作交接和交付物之间的分离。

但在所研究的系统中，它可能是架构上最接近的相邻方法之一。

ScholOnto 与 SWAN：命题并不等于来源

另一个十分相似的方向，出现在研究科学论述的系统中。

ScholOnto，以及后来出现的 SWAN，把科学文档与从这些文档中产生的命题区分开来。

一项资源可以成为某项论断的出处。

这项论断会保留其来源。

其他要素可以支持、讨论或反驳它。

这与上下文原子笔记背后的原则极其接近：

> 来源仍是来源；根据来源构建的知识则是另一个对象。

这些系统也说明，知识系统中记录的命题，不应自动被视为真相。

它在推理中拥有出处与状态。

这与上下文的认识论架构尤其相似。

联邦系统：读取并不等于复制

最后，关于上下文之间关系的一些想法，可以在联邦数据库系统中找到出人意料的古老渊源。

这类架构必须允许多个自主数据库协作，同时不放弃各自的自主性。

一个系统可以公开部分信息。

另一个系统可以远程查阅，也可以决定保存一份本地副本。

而一旦创建副本，就会出现一个经典问题：原件改变后会发生什么？

这正是上下文中的一句话所概括的内容：

> 引用 = 依赖；副本 = 责任。

因此，这些计算机机制早已存在。

上下文把它们转化成了人机空间之间的知识治理规则。

一种整合，而不是凭空发明

这些相邻方法让我们很难把上下文说成一组全新机制的发明。

- 历史并不支持这种说法。
- 出处早已存在。
- 原子笔记早已存在。
- 项目记忆早已存在。
- 跨项目经验沉淀早已存在。
- 在工作过程中创造知识早已存在。
- 结合人、机器和知识系统的环境早已存在。
- 带有出处并可被质疑的论断早已存在。
- 能够共享或复制信息的自主架构早已存在。

更能体现上下文特点的，是它把这些机制**选择、组合并规范为同一种方法**的方式。

这套方法为它们提供一种共同语法：

- 来源；
- 原子笔记；
- 矛盾；
- 决定；
- 状态；
- 历史；
- 知识沉淀；
- 行动；
- 工作交接；
- 交付物；
- 上下文之间的关系。

更重要的是，它把这套组合应用到一个非常当代的问题上：如何在人与 AI 之间维持持久协作，而不让每场对话都成为孤立而失忆的片段。

因此，上下文拥有许多思想渊源和相似之处。

但就像常见的情况一样，了解这些渊源与相似之处，主要是为了更好地理解什么真正属于它自己。