草台班子・實戰系列

AI 管理學

用管理框架駕馭 AI 能力

版本 v1.3.0

2026-09-04

AI 管理學：用管理框架駕馭 AI 能力

版本 v1.3.0 • 2026-09-04 定版

本書內容以 chapters/ 目錄為唯一正本，版本與建置流程見 VERSION.md。

© 2026 草台班子 AI 管理學。保留一切權利。

目 錄

第 0 章：管理者必懂的 AI 基本概念

> 核心命題：你不需要會寫程式，但你需要聽得懂工程師、廠商、員工嘴裡的那十幾個名詞——因為每一個名詞背後，都是一個管理決策。

這本書談的是管理，不是技術。但管理 AI 有一個前提：當別人跟你說「這個 Agent 的幻覺率很高」「我們用 MCP 接內部系統」「API 計費比訂閱划算」，你不能點頭假裝聽懂。

這一章把後面會反覆用到的概念，用管理者聽得懂的方式講一遍。每個概念只回答三個問題：它是什麼、它像什麼、它對你的管理決策意味著什麼。

LLM：那個博學但沒有記憶的新進員工

LLM（大型語言模型）是 ChatGPT、Claude、Gemini 這些服務背後的引擎。把它想成一個剛報到的新進員工：讀過整個圖書館，知識廣博，反應極快，但對你的公司一無所知——不知道你的客戶是誰、產品怎麼賣、上個月發生什麼事。

每次你開新對話，這個員工就重新「第一天上班」。它不記得你昨天教過它的格式、不喜歡的用語、公司的禁忌。這跟缺陷無關，純粹是架構：LLM 本身沒有記憶，記憶要靠外部系統餵給它。

管理含義：你要像對待新進員工一樣對待 LLM——給背景、給範例、給明確的工作說明，而不是期待它「應該知道」。這就是第 8 章知識管理存在的理由。

Token：AI 的計量單位，也是溝通成本

Token 是 LLM 處理文字的基本單位，粗略理解成「字數的計費顆粒」。中文大約一個字一到兩個 token。你跟 AI 說的每一句話、它讀的每一份文件、它產出的每一段文字，都以 token 計量——計量的是用量，也是成本。

管理含義有兩個。一是成本意識：把整本手冊丟給 AI「參考」，等於讓員工讀完倉庫才准開工，錢和時間都在燒。二是溝通紀律：指令越精煉、上下文越聚焦，產出品質越穩定。給 AI 的資訊，準比多重要。

幻覺：自信的實習生，會編出不存在的法規

LLM 的產出是「最可能的下一段文字」，不是「查證過的事實」。所以它會一本正經地編造：不存在的法條、編造的引用來源、看起來合理但從未發生的數據。這叫幻覺（Hallucination）。

把它想成一個極度自信的實習生：你問他不知道的事，他不會說不知道，他會用所有學過的知識拼出一個「聽起來很對」的答案，而且表情毫無破綻。

管理含義：幻覺沒有「修好」的一天，你只能管理它。高風險產出（法務、財務、醫療、對外承諾）必須有人工查證節點；低風險產出（內部草稿、腦力激盪）可以放手。這是第 9 章品質管理和第 10 章風險管理的起點。

Chatbot 與 Agent：顧問跟員工的差別

Chatbot（聊天機器人）是顧問：你問，它答，它給建議，但動手的是你。Agent（AI 代理）是員工：你給目標，它自己去查資料、開工具、跑流程，把結果交出來。

差別在「有沒有動手的能力」。一個只能回答的 AI 再聰明，也只是坐在會議室裡的顧問；一個能操作系統的 AI，才是真正進了組織編制的員工。本書絕大部分的管理框架，是為後者準備的——因為只有會動手的員工，才需要被管理。

Harness：員工的辦公室、權限與門禁

Harness（代理框架／執行環境）是讓 LLM 變成 Agent 的那層基礎設施：它決定 AI 能看到什麼檔案、能用什麼工具、能執行什麼命令、出錯時怎麼停下來。Claude Code、OpenClaw、各種 Agent 平台，本質上都是 harness。

把它想成公司發給員工的辦公室和門禁卡：坐在哪裡、能進哪些區域、能動用哪些設備、下班後權限自動失效。同一個員工，給他全公司門禁和只給他一張桌子，風險完全不同。

管理含義：選 harness 這件事，本質是權限設計。你給 AI 的環境越大、權限越高，產能越高，爆炸半徑也越大。這個取捨是管理決策，不該外包給工程師單獨決定。

Tools 與 Skills：設備跟工作手冊是兩回事

Agent 會動手，靠的是兩樣東西，經常被混為一談。

Tools（工具）是設備：查資料庫、發 Email、操作試算表、呼叫外部系統。沒有工具的 Agent 像沒有電腦的員工，只能空談。

Skills（技能）是工作手冊：告訴 AI「遇到這類任務，照這個流程做」。工具讓 AI 能做，技能讓 AI 做得對、做得一致。一個有完整 SOP 的新員工，第一週就能交出八十分的成果；沒有 SOP 的老員工，做三年還是看心情。

管理含義：這本書反覆講的「模板化」「流程沉澱」「方法卡」，在 Agent 世界裡的對應物就是 skills。你過去為人類團隊寫的每一份 SOP，都是未來 AI 員工的教材。

API：系統之間的點餐窗口

API（應用程式介面）是系統跟系統之間溝通的標準方式。像餐廳的點餐窗口：你不需要進廚房，不需要認識廚師，只要照著菜單格式點餐，廚房就照單出菜。格式對了就有結果，格式錯了就被退單。

當廠商說「我們有 API」，意思是「你的 AI 員工可以直接跟我們的系統點餐」；當廠商說「沒有 API」，意思是「你的 AI 只能站在門外，要人類進去幫它點」。

管理含義：評估任何系統或工具時，「有沒有 API」直接決定它能不能進入你的自動化流程。這要列進採購條件，別留給工程師自己決定。

MCP：工具界的 USB 接頭

過去，每個 AI 平台接每個工具都要單獨開發，就像每個電器配一種插座。MCP（Model Context Protocol）是企圖統一這件事的共同規格：工具照 MCP 標準做一個接頭，任何支援 MCP 的 Agent 插上就能用。

管理含義：MCP 降低的是「整合成本」和「換供應商的成本」。今天用 A 家的 AI、明天換 B 家，工具接頭不用重做一次。評估生態系時，支援共同規格的工具，長期持有成本更低。

計費：月卡跟單點的取捨

AI 服務有兩種付費邏輯。訂閱制像健身房月卡：固定月費，用量內隨你用，適合個人日常使用。API 計費像單次購票：用多少 token 付多少錢，適合系統對系統的自動化流程——因為機器不會刷卡進健身房，它只認票。

管理含義：個人工作用訂閱，流程自動化用 API，兩者預算科目不同、管控方式不同。API 用量會隨流程規模成長，要有用量監控和上限，否則帳單會像沒人管的加班費。

資安三原則：爆炸半徑、最小權限、機敏不上雲

AI 員工跟其他員工一樣，會犯錯、會被騙、會被利用。差別在於它動作快、不知疲倦、而且永遠不會自己覺得「這樣好像不太對」。三個原則夠用：

爆炸半徑：假設它一定會出錯，問題是錯的時候波及多大。能發信給全客戶的 AI，跟只能產出草稿給人看的 AI，是兩種風險等級。設計流程時先問「最壞的情況是什麼」。

最小權限：工作需要什麼權限就給什麼，不多給。只需要讀資料的，不給寫入；只需要寫草稿的，不給發送。權限是事後很難收回的東西。

機敏不上雲：客戶個資、財務數據、未公開的商業資訊，進了雲端服務就離開了你的控制範圍。哪些資料可以給 AI 處理、哪些永遠不行，要白紙黑字寫清楚，而不是靠每個人的常識。

人在迴圈：所有概念的共同收尾

這一章的每個概念，最後都指向同一個管理原則：Human in the Loop，人在迴圈中。AI 負責產出，人負責決策；AI 負責草稿，人負責簽核；AI 負責執行，人負責承擔。

技術名詞會過時——明年的模型、後年的協定，一定跟今年不同。但「理解概念是為了做更好的管理決策」這件事不會過時。後面十三章，就是把這些概念放進管理框架裡，一個一個展開。

✅ 行動清單

1. ☐ 找一個技術背景的同事或廠商聊 15 分鐘，測試自己能不能聽懂 LLM、Token、幻覺、Agent、API、MCP 這六個詞的實際用法
2. ☐ 盤點你目前在用的 AI 服務：哪些是訂閱制、哪些是 API 計費？帳單各歸哪個科目？
3. ☐ 檢查一個你授權給 AI 的工具權限：它能不能讀到它不需要的東西？能不能發出你不想要的東西？
4. ☐ 寫下你的「機敏不上雲」清單：哪些資料永遠不貼給雲端 AI

第 1 章：為什麼 AI 需要管理

> 核心命題：AI 的能力足以接手工作，但可靠性還沒到可以放心交接——能力與可靠性之間的距離，就是管理必須存在的地方。

很多人第一次用 ChatGPT 的感覺是驚嘆。問它寫一封信、整理一段資料、解釋一個概念，幾秒鐘就給出看起來很不錯的答案。那個瞬間，你會覺得 AI 好用到不需要管理。

但那只是第一次。

從「問 AI」到「用 AI 做事」之間的斷層

「問 AI」是一次性的：丟一個問題，它回答，結束。不管答案好不好，你不會再追問第二次。這就像在路邊問人「最近的地鐵站怎麼走」——答案對了很好，錯了也只是多走幾步。

但「用 AI 做事」不同。你是要把一個有明確目標的工作流程交給它——整理客戶名單、寫週報初稿、分析競品動態、處理客服信件。這些任務有標準，有時限，有後果。做錯了的後果，是客戶收到錯誤報價、老闆看到荒謬數字。

斷層就在這裡：AI 的能力足以接手很多工作，但可靠性還沒到可以放心交接的程度。能力跟可靠性之間的這段距離，就是管理必須存在的地方。

不管理的 AI 會出什麼事

沒有管理的 AI 使用，問題不是「會不會出錯」，而是「出錯的方式你無法預測，也無法系統性地修正」。

不一致。同樣的任務，今天做得很漂亮，明天做得很糟。原因多半在你這邊：每次給的指令、上下文、格式要求都不一樣，但你沒有察覺。你以為在「跟 AI 溝通」，其實每次都是全新對話。

幻覺。AI 會編造看起來合理但完全錯誤的內容。一個不存在的法規條文、一個編造的數據來源、一個看起來有道理但從未被驗證的結論。把 AI 的產出直接丟進工作流程，幻覺就變成你背書的錯誤。

範圍蔓延。你讓 AI 整理會議紀錄，它自動加了行動項目。你讓它寫道歉信，它自動解釋了根本沒發生的事。AI 很熱心，但熱心沒有邊界意識。沒有明確的任務範圍管理，AI 會自作主張地擴大工作範圍。

人類疲勞。每次 AI 產出不確定，你就得人工檢查。一開始覺得「檢查一下很快」，但當 AI 接手的工作從一件變成十件，你就從「用 AI 省時間」變成「幫 AI 擦屁股」。管理 AI 的成本比自己做還高。

為什麼「更好的 Prompt」不是答案

面對這些問題，最常見的反應是學好 prompt engineering。

Prompt engineering 有用，但它解決的是單次互動的品質問題，不是系統性的管理問題。一個精心設計的 prompt 可以讓 AI 這一次表現更好，但不能解決：下次忘了怎麼寫、換一個人結果完全不同、任務變複雜後 prompt 越來越長但效果越來越差、AI 的產出需要整合進既有流程。

這些是管理層面的問題，不是 prompt 層面的問題。它們需要流程設計、品質標準、知識沉澱、風險控制——管理學研究了一百年的東西。

管理思維的切入點

管理學的核心假設：任何智能體都有能力邊界和可靠性邊界。管理的任務不是消除邊界，而是設計一套系統，讓邊界內的產出穩定可靠，邊界外的風險被及時攔截。

搬到 AI 上，切入點很清晰：

委派判斷——不是所有事都該交給 AI，你需要判斷框架。目標設定——AI 沒有主動性，你的目標就是它的方向。流程設計——AI 的產出不該直通終點，它需要檢查點和 review gate。品質管理——「看起來不錯」不是標準，你需要具體、可重複、可驗證的驗收條件。知識管理——AI 每次對話都是全新的，你必須主動沉澱經驗。風險管理——幻覺、偏見、資安、過度依賴，不會因為你忽略就消失。

這六個方向就是本書的結構。第 2 章把它們展開成管理框架，第 3 章開始逐一深入。

這本書的承諾

這本書不教你寫更好的 prompt，不教你哪個模型最強，不教你 AI 的技術原理。

它教你一套管理框架，讓你在用 AI 做事時，知道怎麼派任務、設目標、管流程、抓品質、存經驗、控風險。

你的 AI 用得好不好，取決於你管得好不好。從下一章開始，我們把「管好」變成具體可操作的東西。

✅ 行動清單

. ☐
盤點本週所有跟 AI 相關的工作，標記哪些是「問 AI」、哪些是「用 AI 做事」
. ☐

從「用 AI 做事」的項目中，找出一件曾經出過錯的，寫下當時的錯誤類型（不一致／幻覺／範圍蔓延）

. ☐
為那件任務寫下三條明確的品質標準，下次交辦時直接附上

. ☐
設定一個每週 15 分鐘的行事曆提醒，回顧本週 AI 產出品質

第 2 章：管理學基礎框架——六個核心武器

> 核心命題：管理 AI 不需要發明新理論，把管理學既有框架搬到 AI 場景，就足以解決八成的問題。

前一章講了為什麼 AI 需要管理。這一章把「管理」拆成六個方向，每個方向對應一套可操作的框架。後續章節逐一展開，這一章是你手上的地圖。

武器一：任務委派

管理的第一個動作是分工。哪些事交給 AI，哪些留給自己，哪些需要人機協作，這不是憑感覺，而是判斷框架。

核心判斷線是一條光譜。左端是資訊處理——整理、比較、格式化、翻譯——AI 幾乎可以完全接手。右端是判斷——決策、人際溝通、價值取捨——AI 提供分析支援，最終判斷權留在人類手上。中間地帶是初稿和創意發想，AI 做 70%，人類做最後 30% 的收斂。

委派還包含交出去之後的管理：設定第一個檢查點、定義卡住的回收條件、結果走 review gate。方法卡「AI 委派檢查卡」和「AI 任務指派分流卡」是操作工具。第 3 章完整展開。

武器二：目標設定

AI 不會主動理解你的意圖。你說「幫我寫一篇文章」，它會寫，但寫出來的東西可能完全不是你要的。問題出在你沒講清楚「什麼叫好」。

對 AI 來說，目標設定的意義比對人類更大。人類同事可以從上下文推斷意圖、可以問「這樣可以嗎」、可以根據你的表情調整。AI 不行。它只接收你給的文字，盡可能字面地執行。目標越精確，執行越貼近需要；目標越模糊，它就在模糊空間裡自由發揮。

這個框架以 OKR 形式展開——Objectives 定方向，Key Results 定驗收標準。第 4 章詳細說明。

武器三：流程設計

很多人用 AI 的方式是「產出 → 直接使用」。拿到答案，直接採用。工作量小的時候勉強可以，當 AI 接手的工作變多變複雜，直通車模式就會出事。

流程設計解決的問題：AI 的產出從產出到最終交付之間，需要經過哪些關卡？

最基本的流程是三段式：生成 → 檢查 → 交付。生成是 AI 的工作，檢查是人類的責任，交付是確認無誤後的動作。更複雜的場景需要更多關卡：初稿檢查結構和方向，細節驗證數據和事實，整合做最終審核。

流程存在的意義，是在錯誤擴大之前攔截它。初稿階段攔住一個問題，修正成本五分鐘；直接進入客戶簡報，修正成本是你的信譽。第 7 章展開。

武器四：品質管理

AI 的產出有一個特性：總是看起來不錯。語句通順、結構完整、用詞專業，第一眼很難發現問題。但「看起來不錯」和「實際上正確」之間的距離，可能非常大。

品質管理要建立一套超越「看起來不錯」的標準。具體——不是「寫得好」，而是「每個論點至少一個數據支撐」「不超過 800 字」「語氣是給 C-level 看的」。可重複——同一個標準今天用和下週用，判斷結果一致。可驗證——有人能根據你的標準獨立判斷合格與否，不需要你解釋。第 9 章展開完整框架。

武器五：知識與記憶管理

AI 有一個根本限制：每次新對話，不記得上次做了什麼。

這意味著如果不主動管理「AI 應該知道什麼」，你就會不斷重複同樣的說明、犯同樣的錯、得到不一致的結果。你跟 AI 的第三次合作，不會比第一次更熟練——除非你建立了記憶系統。

知識管理在 AI 語境下包含兩件事：把經過驗證的做法固化成可重複使用的指令或範本（經驗沉澱），以及把任務相關的背景知識在正確的時間提供給 AI（上下文管理）。做得好，越用越高效；做不好，永遠原地打轉。第 8 章展開。

武器六：風險管理

AI 的風險很實際：使用中你一定會遇到。

幻覺——AI 自信滿滿地給出錯誤資訊，而且錯得很有說服力。偏見——訓練數據帶有偏見，在某些議題上系統性地偏向特定立場。資安——你貼進 AI 的內容，是否包含不該外流的資訊？過度依賴——用 AI 用到失去自己的判斷力，最安靜也最危險的失控。

風險管理不是叫你不用 AI，而是知道哪些場景風險高、哪些低，在高風險場景加上防護。第 10 章展開辨識、監控和應變三個層次。

六個武器的關係

委派判斷決定什麼任務進入系統，目標設定決定 AI 往哪跑，流程設計決定產出走什麼路徑，品質管理決定終點門檻，知識管理決定系統能不能越用越好，風險管理決定哪裡需要設防線。

它們一起構成完整的管理體系。

這六個武器是核心管理維度。書中還有工具箱（第 6 章）、多 AI 協作升級（第 5 章）、組織設計（第 11 章）、持續改進（第 12 章）作為延伸主題，補充六個維度之外的實戰面向。

✅ 行動清單

1. ☐
   拿出本週的 AI 使用紀錄，按六個武器分類：哪幾項你有在做？哪幾項完全沒碰過？
2. ☐
   針對最弱的那個維度，寫下一個本週就能執行的改善動作
3. ☐
   畫出你自己目前 AI 工作的流程草圖，標出六個武器分別對應哪個環節
4. ☐
   把這張流程圖貼在工作區，每次用 AI 時快速對照：哪個環節漏了管理？

第 3 章：AI 任務委派原則——什麼該交給 AI，什麼不該

> 核心命題：委派不是把工作丟出去，而是從交出去那一刻起，設計檢查節奏、回收條件與驗收閘門。

前一章你拿到了管理框架，接下來進入第一個實戰問題：到底哪些工作該交給 AI，哪些不該？

框架如果不能幫你做出委派判斷，就還只是知識，不是能力。

一條判斷線：創意與判斷 vs 資訊與處理

每次遇到一個任務，先問自己：這件事最核心的價值是什麼？

如果答案是「我的判斷」「我的創意」「我對人的理解」，那 AI 可以幫你加速，但主體要你來。如果答案是「整理大量資訊」「產出初稿」「比較選項」，那 AI 可以幾乎完全接手。

這不是一個二選一的開關，而是一條光譜。越往左，AI 的自主空間越大；越往右，你越要把判斷權留在自己手上。

光譜的左端是這類任務：總結一份十頁的文件、翻譯一篇英文文章、比較十個競品的規格、整理會議紀錄的格式。AI 在這些事上幾乎不需要你介入，你只需要驗收。

中間地帶是初稿生成和創意發想。AI 可以寫出一封向國外客戶解釋延遲交貨的郵件，但前提是你給了夠清楚的框架——語氣要專業但誠懇、不超過 150 字、開頭直接說重點、結尾提出新的時間表。框架給得越精確，AI 的產出越可用。創意發想也一樣：讓 AI 發想 50 個標語方向，人類從中選擇和收斂。AI 是起點，你的判斷是終點。

光譜的右端是決策和人際互動。AI 可以幫你分析兩個工作機會的發展曲線、風險點、還需要收集什麼資訊，但最終選哪一個，是你的人生。AI 也可以幫你準備談判的情境分析、開場白版本、對手可能的底線，但它讀不到房間裡的氣氛，也不會替你承擔談判破局的後果。

方法卡「管理者管的是做什麼和為什麼」把這件事講得很直白：管觸發條件、管 AI 角色、管結果處理。技術細節不是你該管的。觸發條件是「什麼情況下啟動 AI，什麼情況下回到人工」，這是業務判斷。AI 角色是「它這次是草稿機、審核員還是分析師」，角色不同，出錯的代價完全不同。結果處理是「AI 輸出後誰看、看什麼、不合格怎麼辦」，這是管理流程。

委派不是丟出去就沒事了

很多人以為把任務交給 AI 就叫委派。但委派真正的管理動作，是從交出去那一刻才開始的。

方法卡「AI 委派檢查卡」提了一個很實際的問題：你有沒有在交辦時就說好第一次檢查的時間點？如果沒有，那只能算把工作丟出去，稱不上委派。

好的委派至少要有四件事。第一，一開始就定第一個檢查點，不要說「有進度再回」，因為沉默很容易被誤認成「還在做」。第二，講清楚持續檢查的節奏，多久重檢一次、什麼狀態下不用追、什麼狀態下要立刻介入。第三，先定卡住的回收條件——什麼叫卡住、卡住時先回什麼、你要補資料還是改路線還是直接回收。第四，結果回來後先停在 pending review，不要有產出就當完成。

方法卡「AI 任務指派分流卡」進一步把委派的前門管起來。任務來了，先判斷能不能派。判斷標準很簡單：這是執行問題還是最終判斷問題？成功標準講得清楚嗎？失敗了可回收嗎？如果答案偏向高判斷責任、高風險承擔、成功標準講不清楚，那就先留在自己手上。

能派的任務，還要選對入口。是直接交給 producer 做產出，還是先補成完整的任務包再派？是送去 review 和驗證，還是停在等待人類決策？這一步選錯，後面整條鏈都會變貴。

一個實際的例子：你讓 AI 幫你比較市面上十款 CRM 軟體的功能和價格。AI 可以快速給你完整的比較表，整理每個軟體的優劣勢。但價格資訊務必二次驗證——AI 可能用了過時的數字。最終的選擇，要根據你的實際需求，不是只看功能數量。

最後一句話

AI 可以幫你把問題看清楚，但不該替你承擔最後的判斷。你的工作是管好三件事：任務走對入口、中途有檢查節奏、結果有 review gate。

行動：拿出你這週的待辦清單，逐項問「核心價值是判斷還是處理？」把偏向處理的挑出來，挑一項今天試著交給 AI，並且在交辦時就設定第一個檢查點。

✅ 行動清單

1. ☐
   拿出本週待辦清單，每項標記「判斷」或「處理」，把偏向處理的任務列為 AI 委派候選
2. ☐
   挑一項候選任務，用「AI 任務指派分流卡」跑一次：成功標準清楚嗎？失敗可回收嗎？
3. ☐
   交辦時明確寫下：第一個檢查點在何時、卡住的回收條件是什麼、結果停在 pending review
4. ☐

任務完成後回顧：委派判斷對不對？檢查點有沒有攔到問題？寫成一條筆記

第 4 章：AI OKR——如何給 AI 下精準指令

> 核心命題：模糊指令等於模糊結果——用 OKR 邏輯把「做什麼事」升級成「我要什麼結果」，指令品質才會過關。

第 3 章解決了什麼任務該交給 AI。這章處理的是：決定交出去之後，怎麼給指令，才能讓它真的做對。

多數人給 AI 的指令，其實是無效的

「幫我寫一篇關於 ESG 的文章。」

這句話聽起來像在交代工作，其實什麼都沒說。給投資人看還是給員工看？五百字還是兩千字？論文體還是科普體？你以為自己講清楚了，AI 只拿到一個空殼。你拿到產出之後開始改，改完發現跟自己寫差不多，然後覺得 AI 沒用。問題不在 AI，在於你給的是「做什麼事」，而不是「我要什麼結果」。

這不是少數人的問題。多數人跟 AI 對話的方式，跟在通訊軟體交代同事差不多——一句話丟出去，期待對方自己腦補。差別在於，同事會問「所以你要給誰看？」，AI 不會問，它直接給你一個它覺得合理的版本。而它覺得合理的，通常是最平庸的那個。

把 OKR 的邏輯拿來寫指令

OKR 大家都熟——Objective 是你要什麼結果，Key Results 是怎麼判斷做到了。這套邏輯不需要重講，直接搬到 AI 指令上，正好能解決「模糊輸入→模糊輸出」的死循環。

O（Objective）問的是：這次 AI 交付的東西，要達成什麼目的？不是「寫一篇文章」，而是「讓董事會三分鐘內掌握 ESG 重點並支持綠能計畫」。目的不同，產出完全不同。同一個 ESG 主題，給內部員工看是溝通素材，給投資人看是信任憑證，給大眾看是品牌聲明。目的沒定清楚，AI 只能猜，而它猜的方向通常是你最不想要的那個。

KR（Key Results）問的是：交付物能不能被快速驗證？根據方法卡「AI 產出必須能被驗證」的原則——如果你沒辦法在三十秒內判斷 AI 的產出過不過關，問題不在模型，在於你給的指令本身就不夠具體。好的 KR 是「三大重點各附一個數據」「半頁以內」「結尾有明確行動呼籲」這種能直接比對的條件。不是「做得專業一點」「看起來高級」這種感覺詞——這些話對人都不夠清楚，對 AI 更沒用。

OKR 在這裡的角色，是一個指令品質的檢查工具。你寫完一段指令，回頭問自己兩件事：目的講清楚了嗎？產出能不能在三十秒內驗收？兩個都答是，指令才算過關。

兩個對比：看見差距

第一組——商業報告。

「幫我寫一個 ESG 報告。」

AI 會給你一份四平八穩、不痛不癢的通用報告。裡面什麼都有，但沒有一句是你需要的那句。你開始改，改著改著發現還不如自己寫。整趟下來，AI 變成一個需要你大量校稿的實習生。

換成這樣：

「我要向董事會匯報 ESG 進度，目標是讓他們支持綠能計畫。半頁以內，三大重點各附數據，語氣專業但有說服力。不要寫成學術論文，不要出現還沒有數據支撐的承諾。」

同樣是 ESG 報告，第二版的產出可以直接拿去用。差別不在模型，在於你把 O（目的）和 KR（可驗證標準）講清楚了。董事會要的是重點和判斷依據，不是知識普及。你的指令把這個意圖鎖住了，AI 才有辦法對準。

第二組——創意發想。

「幫我想一些健身 App 的標語。」

你會得到二十句差不多的話。因為 AI 不知道你跟誰溝通、走什麼調性、在什麼場景下曝光。它只能用最安全、最泛用的方式回應，結果每一句單獨看都沒問題，放在一起完全沒有方向感。

換成：

「目標讀者是 25-35 歲台北上班族，時間碎片化，想改變體態但不想去健身房。三個方向各十句：強調懶人也能做、強調省時、強調在家就能做。每句不超過十個字，要有記憶點，讓人想點進去了解。」

方向綁住了，產出才有得挑。你可以直接從三十句裡篩出三句進入測試，而不是從二十句泛泛之作裡一句都挑不出來。

管的是什麼，不是怎麼做

方法卡「管理者管的是做什麼和為什麼」講得很明白：觸發條件歸你定，AI 角色歸你定，結果處理歸你定，技術細節不歸你管。

搬到指令設計上也一樣。你要定的是三件事：什麼情境下啟動 AI，AI 在這個任務裡扮演什麼角色，產出來了誰看、看什麼、不合格怎麼退。

AI 的角色尤其值得想清楚。它是主筆，還是草稿機，還是審核員？角色不同，出錯的代價完全不同。如果是主筆，你的指令要包含風格、語氣、禁止事項；如果是草稿機，你只需要給方向和結構，細節留給自己修；如果是審核員，你要給它判準和參考範例。角色沒定就丟任務，等於叫一個不知道自己是廚師還是外場的人去炒菜。

至於 prompt 怎麼調、用哪個模型、API 怎麼串，那是工程師的事。你管到這一層，就變成瓶頸；你完全不管，方向就會跑掉。答案是管對邊界——管該管的，放手不該管的。

先想輸出，再反推輸入

很多人不知道怎麼下指令，是因為不確定自己需要什麼。這時候不要從「幫我做個簡報」開始猜，而是反過來想：我需要的最終產物長什麼樣？

假設你要一個五分鐘的投資人簡報。先想每一頁講什麼：第一頁拋問題，第二頁秀市場規模，第三頁講解決方案和差異化。再想讀者關心什麼：市場夠不夠大、團隊做不做得起來、商業模式說不說得通。這些條件都想清楚了，再把它們寫進指令。AI 拿到的就不是「做個簡報」這四個字，而是一張具體的藍圖。

這個方法在概念上很簡單，但在實務上很有效。原因是：多數人對「我需要什麼」的認知是模糊的。反推的過程逼你想清楚，而想清楚本身就是最好的指令優化。

幾個實用的提醒

不要問「能不能」。AI 不會說不能，它只會試圖回答。「能不能幫我翻譯」浪費了第一句的機會。直接把翻譯要求和格式寫出來，省掉一輪對話。

給 AI 一個角色能提升品質。「你是有十年經驗的內容編輯，請分析這篇文章的標題吸引力、結構邏輯、語氣一致性」就夠了，不需要幫它寫履歷。角色給的是思考框架，不是人設。你不需要告訴它「你曾在麥肯錫工作八年」，你需要的是讓它用特定視角看問題。

忘記設禁止事項，是很多人忽略的盲點。AI 不知道你們公司內部溝通不搞「以此類推」「本著」這些公文腔，也不知道你討厭罐頭結尾。這些偏好不講，它就用最安全的預設值，而預設值往往是你最受不了的那種。

每次下指令前花三十秒想三件事：這個產出的目的是什麼？誰會看？怎麼判斷過關？想清楚再寫，比拿到產出之後改三輪快得多。

重點回顧

一、模糊指令等於模糊結果。用 OKR 邏輯拆：O 是你要什麼結果，KR 是能不能在三十秒內驗收。不能快速驗證的 AI 任務，不是執行問題，是管理定義問題。

二、管定義，不管執行。觸發條件、AI 角色、結果處理是你該定的。模型選擇和 prompt 細節交給工程師。管太細是瓶頸，管太鬆會歪。

三、先想輸出再反推輸入。定好交付物的樣子，再把條件寫進指令，比從任務名稱開始湊有效率得多。

行動：選一個你本週要交給 AI 的任務，用 O 加 KR 的結構重寫指令，跑一次，跟之前的產出比較。你會看見差距。

✅ 行動清單

. ☐
選一個本週要交給 AI 的任務，寫出 O（目的）和三條 KR（可驗證標準）
. ☐
依「三十秒內能否判斷過關」檢查每條 KR，不通過就改寫到具體可量化
. ☐
在指令裡加上一條禁止事項（AI 預設會犯的毛病），跑一次觀察差異
. ☐
把這次滿意的指令存成模板，下次同類任務直接套用

第 5 章：單一 AI 與多 AI 協作——從一個人到一支團隊

> 核心命題：多 Agent 協作的重點不是分工省錢，而是讓每個角色待在最穩定、最可控的工作範圍裡。

大多數人用 AI 的方式是：打開一個聊天窗口，所有問題都在那裡問。短期夠用，但任務一多、流程一長，問題就會浮出來。

這章要講的是：什麼時候該從「一個 AI」升級到「多個 AI 協作」，以及怎麼做。

為什麼不讓一個 AI 全包？

一個夠強的 AI，為什麼不能處理所有事？

短期看起來省事，但任務量和複雜度一拉高，會出現幾個問題。

第一，上下文是有限資源。你塞進去的規則、偏好、背景越多，真正重要的資訊越容易被淹沒。反應變慢，成本拉高，品質反而下降。

第二，流程記憶會混淆。如果同一個 AI 同時負責研究、寫作、審核、發布，它很容易把 A 流程的習慣帶到 B 流程。該短寫時寫太長，該收斂時還在發散，該驗收時直接跳過。

第三，專職角色就是比較穩。即使你全部都用最強的模型，只要每個角色有明確分工、上下文乾淨、驗收清楚，整體系統還是比單一萬能 Agent 穩定。

所以多 Agent 的核心理由不是「便宜分工」，而是讓每個角色待在自己最穩定、最可控的工作範圍裡。

> 多 Agent 是為了降低混淆、降低上下文負擔、提高流程穩定性，不只是為了省錢。

什麼時候該升級？

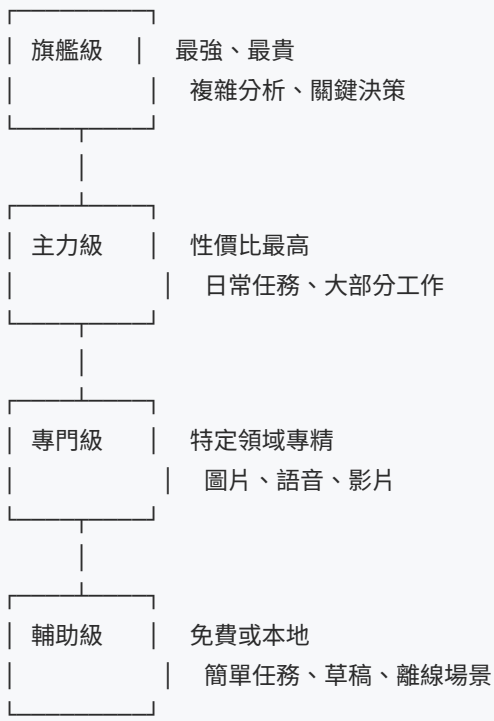
三個信號出現任何一個，就值得考慮拆分。

你在重複做相似的流程。例如每天「研究主題 → 寫草稿 → 配圖 → 發布」，每個環節都可以獨立成一個 Agent。

一個 AI 開始出現上下文混淆。讓同一個 AI 同時做研究和排版，通常兩個都做不好。分開後各自上下文更乾淨，成本也更容易控制。

你需要 24/7 運行，或者成本開始成為考量。一個聊天窗口不可能全天候開著，但 Agent 可以。如果你每天用 AI 超過兩小時，也值得想想：是不是有些任務可以用更經濟的模型處理？

AI 的用人策略：分層原則



核心法則：用主力級處理八成日常工作，旗艦級只用在錯誤代價高的那兩成。翻譯郵件用主力級就好，市場趨勢分析才需要旗艦級。法務或醫療相關的審查，當然也是旗艦級。

這和管人一樣：不是每個任務都需要最資深的人處理，但關鍵決策一定要讓最有經驗的人把關。

協作模式：串聯、並聯、階層

多 Agent 協作常見三種模式：串聯（管線，上游做完交下游）、並聯（分工，多個 Agent 同時跑獨立任務）、階層（主從，一個主控分配任務給執行 Agent）。三種可以混合使用。詳細的角色設計、流程圖與模板，見第 11 章。

Agent 之間怎麼溝通？

多 Agent 協作最大的挑戰不是技術，是通訊。

最簡單的做法是共享檔案：Agent 寫入 JSON 或 Markdown，其他 Agent 讀取。小團隊、低頻通訊很夠用，但要注意同時寫入的衝突問題。

再進一步是事件驅動：Agent A 寫入事件，Agent B 輪詢讀取，處理後寫結果。適合有明確上下游的非同步流程。

如果需要中央決策，就走主控中轉：所有結果回到主控，由主控統一分配和轉發。代價是主控本身成為單點，要考慮它的穩定性。

三個最容易出事的地方

多 Agent 系統上線後，問題通常出在這三個地方。

記憶不一致

Agent A 修改了共享資料，Agent B 還在讀舊版本。

解法是每次修改後立即寫入，讀取前檢查時間戳。重要資料用版本號或狀態欄位管理。每個任務加上狀態（待處理 → 處理中 → 完成），Agent 處理前先檢查狀態，也順便解決兩個 Agent 搶同一個任務的問題。

另外兩個常見問題——上游錯誤被逐級放大的「錯誤級聯」，以及所有 Agent 都搶著通知人類的「通知疲勞」——在第 11 章有完整的處理架構。

交接才是真正的設計重點

多 Agent 系統成熟後，不能只設計「誰做哪一步」，還要設計每一步的交付與交接規則：

- 上游交出什麼
- 下游怎麼接手
- 中途失敗時誰停下、誰報錯

委派出去不等於完成。第一個檢查點什麼時候、卡住時怎麼回收、結果回來後誰有 final review 的權力，這些都要事先約定。這也是為什麼方法卡〈AI 委派檢查卡〉把「已委派」重新定義成「有檢查、有回收、有 review gate 的受控委派」。

更完整的角色責任、備援路徑、升級制度，會在第 11 章再展開。

本章重點

多 Agent 協作的本質是交接設計與流程穩定性，不只是分工。三種協作模式（串聯、並聯、階層）按場景選擇，可以混合使用。模型分層用主力級處理八成工作，旗艦級只用在刀口上。三個最常出錯的地方：錯誤級聯、通知疲勞、記憶不一致，事先設計好就能大幅降低。

行動：選一個你每週都在重複做的流程，判斷它適不適合拆成兩個角色。如果適合，寫下每個角色的輸入、輸出與驗收標準。

✅ 行動清單

1. ☐
   盤點你目前用 AI 跑的重複流程，找出是否有上下文混淆或品質下降的信號
2. ☐
   選一個最成熟的流程，嘗試拆成兩個角色（如：研究員 + 寫手），寫清楚各自的輸入與輸出
3. ☐
   為每個角色選定對應的模型分層（旗艦級／主力級／輔助級），記下理由
4. ☐
   設計交接規則：上游交什麼格式、下游怎麼接手、中途失敗誰停下報錯

第 6 章：建立你的 AI 工具箱

> 核心命題：工具是為流程服務的——先定義問題和流程，再選工具；順序反了，採購單就是混亂放大器。

上一章談到任務變複雜時，單一 AI 會開始撐不住。但對大多數人來說，更實際的起點是：先把手邊一兩個工具用到熟，讓它們穩定支撐日常工作。

很多人一開始就掉進兩個極端。要嘛只用一個 AI，什麼都塞給它；要嘛一口氣註冊十幾個，每個都只碰過一次。兩種都走不遠。

這章不談哪個工具最紅。那些資訊每個月都在變。要回答的是：你現在的工作需要哪幾類 AI 能力？怎麼組合才不會散掉？

先想清楚你要解什麼問題

方法卡「工具選擇是最後一步」把這件事講得很直白：先買工具再想流程，通常只是把原本的混亂包成更貴的混亂。

正確的順序：先說清楚要解什麼問題；接著畫出流程，定好誰負責、誰驗收、出錯誰接手；流程跑順之後，再回頭看哪個環節值得用工具加速。如果連人工流程都跑不順，工具只會把問題放大、加快。

五類工具，五種問題

管理者不需要追幾百個工具的動態。認得五類就夠了——每一類對應一種管理現場會遇到的問題。

寫作類：日常文字的主力

處理郵件、摘要、初稿、潤稿、腦力激盪。投資報酬率最高的一類，幫你把「零碎念頭變成可讀文字」從三十分鐘壓到五分鐘。每天早上的郵件草稿、會議後的重點整理、週報的初稿，這些你本來就會做的事，工具只是幫你省下「從空白頁開始」那段最痛苦的時間。選寫作工具的重點是輸出穩定性——穩定可重複，比偶爾驚豔重要。

分析類：消化大量資訊的助手

處理長文本拆解、資料比對、趨勢整理。關鍵在於能不能處理你實際會遇到的文件長度和複雜度。選之前拿真正會用的文件試一遍。分析類工具的價值不只是幫你讀完東西，而是幫你「帶著問題去讀」——提問的能力，決定了它能發揮多少價值。

創意類：把想法變成看得到的東西

處理簡報設計、圖片生成、內容視覺化。對內容工作者和行銷來說，能大幅降低「想到了但做不出來」的門檻。以前這些事情要找設計師排隊，現在可以自己先出一版，再決定要不要交給專業處理。但產出通常需要人工把關——把它當「快速出草稿」的工具，而不是「直接交付」的工具。

協作類：讓團隊資訊不再散落

處理會議紀錄、訊息摘要、文件共編、知識沉澱。它的隱藏價值是建立「組織記憶」——把散落在腦袋裡、email　串裡、錄音檔裡的知識固定下來。協作類工具能不能落地，取決於團隊有沒有人願意當維護者。

自動化類：把重複流程交給機器跑

把「搜集資料、整理摘要、生初稿、排版輸出」這種每次都要手動接力的流程，交給平台自動跑。它是五類裡面最後才需要碰的——如果連手動跑這個流程都跑不順，自動化只是讓錯誤跑得更快。

怎麼選：三個實際問題

選工具的時候問自己三件事就夠了：

能不能在一天內學會基本操作？學不會的，通常也用不起來。

能不能接到現有的工作節奏裡？如果每次都要切換平台、複製貼上，遲早會荒廢。

公司允許用嗎？涉及客戶資料、財務數字、法務文件的任務，合規是底線。

至於免費還是付費，原則很簡單：先免費試，確認真的常用再為那一兩個付費。付費要解決的是你實際遇到的瓶頸，不是焦慮。

最常見的陷阱：工具收集癖

註冊了很多工具，每個都試過一點，但真正穩定在用的只有兩三個。問題不在工具不夠好，而在沒有形成工作組合。一個工具用到八十分，勝過五個工具各用二十分。

方法卡「工具選擇是最後一步」裡的操作建議可以直接照做：列出你實際要解決的三個問題（是「我們卡在哪」，不是「AI 能做什麼」）；每個問題找一個最對口的工具，試用兩週；兩週後只留真正被用起來的，其餘果斷淘汰。

從角色出發的組合建議

管理者最需要寫作類、分析類和協作類，重點是縮短從「收到資訊」到「做出判斷」的時間。內容工作者在寫作類基礎上加上創意類。知識工作者或研究者，寫作類加分析類通常就夠了，自動化類等發現自己每天都在做同樣的整理動作時再來。

不管哪種角色，先從一兩個工具用熟，確定能穩定支撐日常工作之後，再根據缺口補進來。

什麼時候該升級到自動化

三個訊號：你開始重複做同一串流程；你需要跨工具接力每天切來切去；你出現固定節奏（每天回顧、每週報告、每月整理）。出現這三個訊號，自動化才真正有價值。在那之前，先把手動流程跑順。

自動化還有一個隱藏成本：設定和維護。如果沒人定期檢查它是不是還在正確跑，自動化流程會慢慢變成另一種形式的技術債。

兩個重點，一個行動

工具是為流程服務的。先定義問題和流程，再選工具。順序反了，採購單就是混亂放大器。

選一兩個工具用到熟，比同時追十個有效得多。能被穩定用起來的，才是對的工具。

行動：今天就盤點你手上的 AI 工具。列出你實際在用的和只是「有裝」的，把後者收掉。然後問自己：我現在最卡的工作環節是什麼？用一句話寫下來。這句話就是你的問題定義，有了它，你才知道該從五類工具裡挑哪一個來試。

✅ 行動清單

1. ☐ 盤點手上的 AI 工具，分成「穩定在用」和「裝了沒用」兩堆，後者本週內退訂或移除
2. ☐ 用一句話寫下現在最卡的工作環節，對應到五類工具（寫作／分析／創意／協作／自動化）中的哪一類
3. ☐

為那個環節挑一個最對口的工具，排兩週試用期，試用結束只留真正被用起來的

. ☐

把「工具→解決的問題→使用頻率」寫成清單，貼在工作區當決策參考

第 7 章：AI Prompt 工作流設計——把流程變成可重複系統

本章核心

> 核心命題：Prompt　不是一句話的技巧，而是工作流的入口——把一次成功變成每次都能重複成功的系統，才是管理的目標。

多數人不是不會用 AI，而是沒有把一次成功，變成下一次也能重複成功的流程。

所以本章不談零散技巧，而談一個管理問題：**如何把 AI 的使用方式標準化、模組化、可迭代化。**

Prompt 不是一句話的技巧，而是工作流的入口。

把 Prompt 當成流程，不當成靈感

如果每次都重新想怎麼問，結果就會依賴當下狀態；品質不穩、耗時增加，也無法交接。

因此，Prompt 設計的目標不是「這次問得好」，而是：

1. 讓同類任務可以重複執行
2. 讓輸出品質更穩定
3. 讓流程可以被檢查與改善

這就是 SOP 化的起點。方法卡「Prompt 只佔 20%，剩下 80% 決定成敗」講的也是同一件事——把注意力從措辭轉向流程、review 機制和 KPI，才是管理者該用力的地方。

Prompt 工作流的三層結構

第一層：模板化

把常見任務整理成固定欄位，至少包含四個元素：

- 任務情境
- 目標輸出
- 限制條件

- 禁止事項

模板化的價值，不是省打字，而是降低遺漏與歧義。

第二層：分步化

複雜任務不要一次完成，而要拆成數個可驗收的小步。

基本拆法是：

1. 先定義輸出框架
2. 再逐段生成內容
3. 最後統整與修正

分步化的目的，是把思考、生成、驗收分開，避免 AI 一次做太多決策。

第三層：迭代化

每次執行後，都要留下修正痕跡：

- 哪裡不符合需求
- 哪個欄位描述不夠清楚
- 哪一步最容易失真

這些修正不是補救，而是下一版流程的輸入。方法卡「連續 5 次一致才算穩定」給了明確門檻：流程要連續跑 5 次、每次 output 格式與內容都一致，才算通過穩定性測試。達不到就不要上線。

設計 Prompt 工作流的四個步驟

步驟 1：先定義交付物

不要先想怎麼問，要先想最後要收到什麼。

至少定義：

- 產出格式
- 讀者對象
- 使用場景
- 成功標準

如果交付物沒定義清楚，Prompt 再漂亮也只是模糊執行。

步驟 2：拆解任務節點

問自己：這個任務有哪些階段不能混在一起？

常見節點包括：

- 蒐集資訊
- 建立框架
- 撰寫初稿
- 審核修正
- 統一格式

原則是：每一步只處理一種主要認知工作。

步驟 3：為每個節點指定輸入與輸出

每一步都應明確回答兩個問題：

- 這一步吃進什麼資料？
- 這一步交付什麼結果？

一旦輸入輸出明確，流程才能銜接，品質才能檢查。

步驟 4：建立回饋機制

每次使用後，只記錄最重要的三件事：

- 哪個地方做對了
- 哪個地方出錯了
- 下次要修改哪條規則

這樣才能把偶然成功，慢慢變成穩定能力。

兩個最重要的原則

原則一：逆向設計

從最終交付物反推 Prompt，而不是從腦中想到什麼就丟什麼。

順序應該是：

交付標準 → 輸出格式 → 所需資訊 → Prompt 結構

原則二：分段驗收

不是等整份結果做完才檢查，而是在每個關鍵節點就驗收。

這樣可以更早發現：

- 任務理解錯誤
- 結構偏掉
- 內容深度不足
- 格式不符合要求

管理的本質，不是事後救火，而是中途校正。

常見失敗，不是 AI 笨，而是流程設計有問題

最常見的三種問題是：

1. **一步塞太多事**：讓 AI 同時思考、判斷、整理、表達，結果每一項都變淺。
2. **缺少限制條件**：沒有定義邊界，輸出就容易發散。
3. **沒有沉澱成功做法**：每次都從零開始，能力無法累積。

所以要修的，通常不是某一句 Prompt，而是整個流程結構。

把好結果沉澱成資產

Prompt 工作流真正的價值，不在單次產出，而在累積。

每完成一次高品質任務，都應留下三種資產：

- 可複用模板
- 可重跑流程
- 可迭代標準

這樣你累積的就不是聊天紀錄，而是工作系統。方法卡「AI　交辦閉環卡」的最後一步就是在做這件事——把一次好的產出，變成一個可管理、可驗收、可複用的閉環。

與全書其他章節的關係

本章處理的是「流程設計」。

- 第 4 章處理的是「如何把目標說清楚」
- 本章處理的是「如何把任務跑順」
- 第 9 章處理的是「如何驗收品質」
- 第 12 章處理的是「如何持續優化」

Prompt 工作流的角色，是把單次指令提升為可管理流程的中介層。

本章結論

AI 使用成熟的標誌，不是你會寫多少厲害 Prompt，而是你是否建立了：

- 可複用的模板
- 可拆解的步驟
- 可驗收的節點
- 可迭代的回饋

當 Prompt 從一句話升級成工作流，AI 才真正從工具變成可管理的生產力系統。

✅ 行動清單

1. ☐
   選一個本週要交給 AI 的任務，先定義交付物（格式、讀者、成功標準），再反推指令結構
2. ☐
   把任務拆成不超過三個節點，為每個節點寫清楚輸入與輸出
3. ☐
   執行完畢後記錄三件事：哪裡做對了、哪裡出錯了、下次要改哪條規則
4. ☐
   把這次跑順的流程沉澱成可複用模板，存入你的 Prompt 模板庫

第 8 章：知識管理的重點不是知識庫

> 核心命題：知識管理的目標不是「建立一個知識庫」，而是讓 AI 產出的好結果可以被重複使用。

先回答一個問題：你的 AI 產出過好結果嗎？

如果答案是肯定的——某次 prompt 產出了你很滿意的文案、某次流程跑出了可用的分析報告、某次 AI 把會議紀錄整理得比你手動做還好——那下一個問題才是：這個好結果，下次還能複製嗎？

多數團隊在這裡斷裂了。好的 AI 產出是一次性的煙火，看過就沒了。下一次，換個人、換個對話、換個情境，一切從零開始。這才是知識管理該解決的問題：不是「把知識存起來」，而是「把成功的 AI 使用經驗變成可複用的資產」。

我們在方法卡〈先做自動化再做知識管理〉裡講過這個順序：先讓 AI 幫你省時間，再用省下來的時間沉澱經驗。順序不能反。很多團隊同時推自動化和知識管理，結果兩邊都卡住——因為自動化需要穩定的 input，而知識管理的 input 天然不穩定。硬混在一起，只會讓團隊覺得 AI 導入是個無底洞。

所以這一章不談「怎麼建知識庫」。那個問題太早了。我們談的是：在 AI 已經開始產出價值的情況下，怎麼把好的產出固化下來，讓它不再是運氣。

AI 的記憶就是上下文，上下文就是成本

AI 的記憶不是大腦裡的神經連結，而是每次對話時你灌進去的文字。這件事決定了兩個現實：載入越多，每次請求越貴越慢；載入越多不相關的東西，AI 的注意力越分散，輸出越不精確。

方法卡〈AI 團隊的上下文就是成本〉把這件事講得很直白：上下文輕量化是成本控制，跟整理癖無關。一個 agent 啟動時讀了超過三個檔案，你就該問為什麼。工作目錄裡一堆三天沒動過的檔案，就是在燒管理者的錢。

這意味著知識管理不是「把所有東西都存起來」，而是「在正確的時刻載入正確的知識」。選擇性載入，不是全部記住。

三層知識架構

實際操作上，把知識分成三層，按需載入。

L1 是系統身份層，每次對話都帶上。內容是你的角色定位、溝通偏好、核心目標、絕對不做的事。控制在兩百字以內，每季更新一次。不要放公司歷史或產品細節，那些是 L3 的事。

L2 是任務模板層，執行特定任務時才載入。每種任務一個模板，寫清楚需要什麼輸入、走什麼流程、交付什麼格式、什麼算不合格。模板的價值在快速取用，不在分類精細。按任務類型建資料夾，每個放一個模板檔就好。

L3 是專案上下文層，處理特定專案時才載入。寫清楚專案目標、約束條件、已拍板的決策、還沒解決的開放問題。其中「已確認決策」最有價值——它能防止 AI 重複提出已經被否決的方案。每個專案一個檔案，專案結束就歸檔。

低層級穩定通用，高層級動態具體。這個原則跟人類團隊的管理邏輯一樣：新人不需要知道所有專案的細節，但他需要知道公司的基本規矩。

好結果怎麼變成可複用的資產

這是整章最關鍵的操作。方法卡〈Prompt 只佔 20%〉說得很清楚：把 AI 用好的人，花 20% 的時間調 prompt，80% 的時間設計流程、review 機制和 KPI。知識管理屬於那 80%。

具體做法有兩個。

一個是反向生成模板。當某次 AI 任務的產出讓你滿意，不要只是高興一下就過去了。花一分鐘請 AI 總結這次任務的 context，生成一份可重用模板。審核過後存入 L2 模板庫。下次同類任務，直接載入模板，省掉重新解釋背景的時間。

另一個是任務後萃取。每次完成一個有意義的任務，問自己：這次有哪些 context 可以沉澱成模板，或更新到專案文件？不需要寫長篇大論，一句話就夠。但這個習慣是知識系統持續成長的驅動力。

方法卡〈AI 上下文衛生管理〉把維護紀律講得更系統化：測量先於行動，區分三種結構性浪費（重複說明、閒置工具、過期內容），設定維護觸發條件，記錄可量化的改善結果。有興趣的讀者可以參考那張卡的完整四步法。這裡只強調一件事：知識管理的敵人不是技術，是遺忘。一個季度不做維護，你的模板庫就會變成垃圾場。

入職培訓：對待 AI 像對待新人

把一個新的 AI 應用導入團隊時，比照新人入職。不要一次把所有指令倒給它。

第一步，給身份卡和專案 context，讓它建立基本認知。第二步，跑一次完整流程，連同你期望的輸出一起展示，讓它對齊標準。第三步，從簡單的子任務開始，確認品質沒問題再逐步加難度。

這個順序不是為了照顧 AI 的感受。是因為一次性灌太多資訊，AI 的表現反而會下降——回到前面講的，上下文太重，注意力會稀釋。逐步放手，才能在每個階段確認方向對了沒有。

最常見的兩個錯誤

第一個是建了一個很大的知識庫，裡面什麼都有，但沒有人在維護。方法卡〈先做自動化再做知識管理〉把這個叫「反面信號」：知識庫建了但沒人更新，自動化和知識管理共用一個專案計畫，有人說「AI 可以順便幫我們建知識庫」。看到這些，就知道順序搞反了。

第二個是把所有知識塞進一份超長的 prompt。每次對話都帶著幾千字的背景資料，覺得這樣 AI 就能「理解全局」。事實上，上下文越重，AI 越容易迷失在細節裡。分層載入，按需讀取，才是正確的做法。

本章重點

1. 知識管理的目標是讓好結果可複製，不是建一個大知識庫。先確認 AI 已經能穩定產出價值，再談沉澱。
2. 三層架構處理不同粒度的知識：L1 身份、L2 任務模板、L3 專案上下文。分層載入，不要一次全塞。
3. 維護比建設重要。季度回顧加任務後萃取，是知識系統不會腐化的最低要求。

行動：找出你上週最滿意的一次 AI 產出。花一分鐘，讓 AI 把那次任務的 context 總結成一份可重用模板。存起來。這就是你知識管理系統的第一個種子。

✅ 行動清單

. ☐
找出最近一次滿意的 AI 產出，讓 AI 反向生成一份可重用模板，審核後存入 L2 模板庫

. ☐
檢查你慣用的對話開頭：身份、規則、目標是否濃縮在兩百字內（L1）？

. ☐
盤點一個進行中的專案，把「已確認決策」寫成清單放進 L3，防止 AI 重提已否決方案

. ☐
在日曆排一個季度 30 分鐘的知識庫維護：刪過期模板、補新萃取的經驗

第 9 章：AI 輸出品質管理

> 核心命題：系統性判斷「這個輸出可用嗎」——不是全盤相信，也不是全盤否定。

品質問題的本質

AI 不知道它不知道什麼。

這句話被講到快爛了，但多數團隊的做法還是停在「產出之後再看一眼有沒有錯」。看一眼只能算賭運氣，稱不上審核。尤其是當 AI 的輸出看起來通順、格式整齊、語氣專業的時候，人很容易放鬆警覺。但 AI 犯的錯有一個特徵：它往往看起來非常合理。不會拼錯字，不會文法不通；它用完美的句子，講一個完全錯誤的意思。

一個實際發生的例子：某個團隊讓 AI 整理客戶反饋摘要，AI 把一個關鍵的負面意見合併到另一條裡，用中性語氣包裝掉了。摘要讀起來流暢、結構完整，主管直接轉發給管理層。直到客戶經理在會議上被問到為什麼沒處理那個問題，才發現摘要漏掉了核心風險。

這種錯表示品質管理缺了一環，不能只當成單點失誤處理。你需要的是一套可重複的審核架構，由三件事構成：review 機制、驗收標準、review owner。三個都到位，品質才可控。缺任何一個，AI 產出就會在你沒注意的地方出問題。

Review 機制：把檢查變成流程的一部分

很多人把 review 當保險，覺得「有加比較好」。這想法要反過來：沒有 review 的 AI 自動化，不是在省時間，是在製造沒人負責的錯誤（方法卡：AI 產出的 review 不是選配）。

Review 要有效，得先分級。不是每個產出都需要同樣的審核深度，按任務風險和輸出形式來切：

- L1：內部參考、低風險。三十秒掃一眼，方向對、沒有明顯錯就好。適合草稿、腦力激盪、初步想法。
- L2：影響決策、中等風險。花三到五分鐘，對照原始需求檢查事實和邏輯。適合內部報告、策略分析、給同事的參考資料。
- L3：對外發布、高風險。完整審核加上交叉驗證，十五到三十分鐘跑不掉。適合客戶提案、公開內容、合約文件。

分級的好處不只是省時間。當團隊知道每類任務對應哪個等級，就不會出現重要文件隨便看兩眼、或者內部備忘錄搞三輪審核的浪費。每個團隊應該有自己的分級表，根據實際業務來定。

審核流程本身也可以用 AI 輔助。做法是讓生成者先自我檢查——要求它列出最可能出錯的三個地方、每個事實陳述的來源、如果自己是這領域的專家會挑戰哪些部分。這一步能抓到約三到四成的明顯錯誤。接著用另一個模型做交叉審核，只給最終輸出，不給原始 prompt，讓它專注找問題：事實有沒有錯、邏輯有沒有跳步、有沒有遺漏反面論點、語氣適不適合目標場景。

交叉審核的結果本身也要人類判斷，不能無條件採信。審核者有時候會過度挑剔，有時候會漏掉實際問題。這就是為什麼機制裡要有人類把關的環節。整個流程的設計重點是：讓 AI 暴露不確定性，而不是掩蓋它。

有一個常見的誤區是讓 AI-2 重新做一次，而不是找問題。審核者的角色是挑錯。讓它帶著「資深審核者」的視角去檢視，產出的回饋會實際得多。如果審核者收到的指示是「產生一份更好的版本」，它會把精力花在重寫而不是找問題，那就失去交叉審核的意義了。

驗收標準：三十秒內要能判 pass 或 fail

很多團隊抱怨 AI 產出不穩，實際上是不清楚自己要驗什麼。任務描述停在「做得專業一點」「看起來高級」「整理得清楚」，這些話對人都不夠清楚，對 AI 更沒用。結果是每輪 review 都在重講品味、重講方向、重講期待。返工看似來自 AI 能力不足，其實是管理者沒有把可驗收條件前置（方法卡：AI 產出必須能被驗證）。

好的驗收標準有幾個特徵：能在三十秒內判斷過或不過、有明確的正例和反例、退件理由能指向具體項目而不是只說「感覺不對」。舉幾個實際能用的例子：「摘要不得遺漏三個關鍵數字」「首頁三秒內看懂主標與行動呼籲」「翻譯不得出現任何未經確認的專有名詞」「每個建議都必須附上資源需求和時程估算」。這種標準一看就知道怎麼驗。

如果你發現自己寫不出這種標準，那通常代表任務本身還沒定義清楚。標準還是抽象詞的時候，先別把責任丟給 AI。任務定義沒完成，AI 做再多次都不會對。

驗收標準也不是只在最後才用。輸出品質的上限在 prompt 階段就決定了。在開工前把邊界講清楚——範圍、格式、長度、語氣的具體限制。資訊要完整，提供推理所需的全部背景，不讓它猜。一個實用的做法是在 prompt 裡要求 AI 先說明自己對任務的理解，確認對齊了再往下走。這一步能大幅減少後面的返工。

另一個容易被忽略的面向是確定性分級。要求 AI 在回答時標註每一步的確定性——高、中、低。高是公認事實或邏輯必然，中是有數據但附帶條件，低是推測或類比。人類優先驗證低確定性的部分，審核成本立刻降下來。這不是什麼複雜的方法，但在實務上非常有效。

四維審核是一個可以內化的框架：準確性（事實和數字正確嗎）、相關性（回應了真正的問題嗎）、完整性（涵蓋必要的觀點嗎）、可執行性（建議能直接落地嗎）。不同類型的任務，重點維度不同。分析類優先看準確性與完整性，決策類優先看相關性與可執行性，創意類優先看完整性與可執行性。不必每次都四個維度都跑滿，但要先想清楚這次應該驗什麼。

Review Owner：確認的人要有能力確認

有了機制和標準，還缺最後一塊：誰來把關。

這不是隨便指派一個人有空看一下就好。Review owner 必須懂業務、有時間、有權力退回產出。三個條件缺一不可。讓不懂業務的人做確認，等於沒有確認。讓沒有權力退件的人做確認，也等於沒有確認。這聽起來是常識，但在實際運作中，很多團隊的 review owner 是「剛好有空的那個人」，而不是「最適合判斷的那個人」。

更危險的情況是，隨著 AI 表現越來越穩定，團隊會開始「優化」掉人工確認。先是跳過一次，沒出事；再跳過一次，還是沒事；然後就變常態。直到出事。AI 的錯誤率不是零，而且它的錯誤分布不均勻——可能連續一百次都對，第一百零一次錯得離譜（方法卡：人工是最後一道關卡）。

人工確認是剎車，不是裝飾。你可以九十九次不踩，但那一次踩不下去的時候，代價是整台車。

如果確認步驟太重、太慢，解法不是移除它，而是讓它輕量化。用 checklist、用模板、用差異標記，讓確認的人能在幾分鐘內完成有效的把關。確認步驟的設計不好，是流程問題，不是「不需要確認」。每個 AI 產出流程都有一個終點——發給客戶、上線到產品、提交給主管、公開發布。在終點之前，必須有一個明確的人工確認步驟。不是「大概看一下」，而是「確認內容正確、語氣恰當、沒有遺漏」。

還有一個實務建議：每次審核後記錄發現了什麼類型的問題、在哪個階段被發現、回到 prompt 階段什麼設計可以避免。累積十幾次之後，你會看到自己的品質模式——哪類任務最易出錯、哪個審核階段最有效、prompt 該怎麼系統性調整。這個回饋迴路是長期提升品質的關鍵。

從審產出到審過程：把 AI 組成專案組

前面講的 review，對象都是產出——報告、文案、程式碼成品。但當 AI 開始跑比較長的專案，問題會變形：產出本身品質很好，方向卻已經歪了。

目標漂移是最貴的失敗模式。AI 很擅長把「順手看到可以改進的地方」做成正事：commit 很勤、報告很漂亮、每個單點產出都過得了審，拼起來卻不是原本要的東西。品質審核抓不到這種錯，因為它驗的是成品，不是軌道。時程落後是顯性問題，一眼就看得出來；目標漂移是隱性的，等你發現的時候，代價是好幾天的高品質白工。

這也暴露了只談品質的盲點。專案管理談的是時程、範疇、成本、品質四個維度，品質審核只管其中一個。AI 在不必要的地方死磕，品質維度完全無感——東西做得越好越無感——但另外三個維度同時在失血：時程被吃掉、計畫外的功能長出來、token 和工時燒在沒人要的東西上。AI 時代成本這一維特別容易被忽略，因為邊際成本看起來很低，「讓它順手做一下」感覺不用錢。但對一個按量計費、還會自我擴張範圍的執行者，成本失控不是花太多，是花在計畫外。

管理學對這個問題早有答案：專案組。一般企業裡的結構是——業務扮演甲方代表，提需求、定目標，並且持續監督 PM 的時程與功能實現；PM 把需求翻成計畫、盯執行；執行團隊負責交付。三個角色分開，誰都不能自己兼：甲方不能自己宣布需求達成，PM 不能自己核准自己的進度，生產者不能當自己的審核者。

這個結構可以整組搬到 AI 上。一個 AI 扮演業務兼 PM：負責商業規劃、維護時程表、驗收功能實現；一個 AI 扮演工程師，負責執行；監督稽核再交給另一個 AI 或固定流程；人只做定期抽檢。特別是甲方代表這一層——它是目標的來源，也是時程的第一個把關者。AI 專案組最常出問題的地方不是執行不夠快，是沒有人扮演這個角色：目標講完一次就沒人管了，執行者自然開始自己決定什麼值得做。

但 AI 專案組要跑順，有一個傳統專案管理沒強調的前提：對帳基準（方法卡：監督的是軌道，不是成品）。人類團隊開會對進度，靠的是共同理解和追問；AI 之間沒有這種默契，你給兩個 AI 各自「理解」任務，它們會理解成兩件事。所以監督者和執行者必須共享一個不用語意反推的帳本：一份寫死的計畫清單，加上執行痕跡——程式碼提交、文件變更、產出紀錄——每一筆都標明它對應計畫的哪一條。對得上，進度成立；對不上任何一條的產出，就是漂移訊號，直接浮出來，不用人猜。

這個原則跟前面的驗收標準一脈相承：把判斷題變成是非題。驗收標準讓「產出好不好」三十秒可判；對帳基準讓「方向偏沒偏」也三十秒可判。凡是需要監督者「看懂」執行者在幹嘛才能判斷的設計，都會漏，因為那個看懂本身又是 AI 在做，幻覺會在監督層複利。

人的角色也要跟著調整。當監督已經由 AI 在做，人的定期檢查就不該是重新對帳一遍，而是抽查對帳的品質：監督者有沒有漏抓、有沒有橡皮圖章化、計畫清單本身是不是還反映真實目標。監督層也會漂，所以人力用在監督監督者。這個抽檢不用久，但必須有——它是整套機制裡唯一不由 AI 擔任的職位，拿掉了，專案組就變成三個 AI 互相確認彼此都同意。

三件事記住

1. 每條 AI 產出流程都要有 review，沒有例外。Review 是流程的必要組件，拿掉它只是把犯錯的速度加快。
2. 驗收標準要在開工前定好。不能被快速驗證的任務，問題不在 AI，在管理定義。
3. 人工是最後一道關卡，確認的人要懂業務。剎車可以不踩，但不能沒有。

下一步：今天就列出團隊裡所有 AI 產出流程，確認每一條都有明確的 review owner、驗收標準和分級機制。缺的補上，本週完成。

✅ 行動清單

1. ☐
   列出團隊所有 AI 產出流程，逐條標記 review owner 是誰——沒有的立刻補上
2. ☐
   挑一條流程，把驗收標準改寫成「三十秒內能判 pass 或 fail」的形式
3. ☐
   用四維審核（準確／相關／完整／可執行）跑一次最近的 AI 產出，記下哪一維最常出問題
4. ☐
   為高風險產出線設定分級：哪些快速掃描、哪些四維審核、哪些多階審核
5. ☐
   挑一個正在跑的長專案，確認它有對帳基準：計畫清單 + 每筆產出可對應到清單條目，對不上的有沒有人處理

第 10 章：風險管理——AI 的邊界在哪裡

> 核心命題：AI　能做很多事，但不是所有事都該讓它做。風險管理的目標是劃清邊界，不是追求零風險。

為什麼需要風險邊界

AI 的錯誤有兩種特性： 1. **自信地犯錯**——語氣堅定但內容錯誤，讓人容易误判 2. **錯誤會放大**——一個環節的錯誤會沿流程傳播，越傳越遠

風險管理不是「不讓 AI 做事」，而是明確定義**什麼情況下必須有人類介入**。

風險分類框架

AI 的風險可以歸為四類：

風險類型	特徵	典型場景
事實風險	錯誤資訊被當成真的	數據引用、法規解讀、技術細節
決策風險	AI 建議導致錯誤行動	投資建議、人事決策、策略判斷
外洩風險	敏感資訊流出	客戶資料、商業機密、API Key
聲譽風險	AI 產出直接面對外部受眾	自動發文、客服回覆、公開報告

風險等級判定

用三個問題快速判定風險等級：

1. 錯誤的代價是什麼？（時間/金錢/法律/聲譽）
2. 輸出的使用場景？（個人參考/內部決策/對外發布）
3. 錯誤被發現的難度與修正成本？

等級對照表

等級	判定條件	管控措施
低風險	僅個人參考、可即時修正	輸出後快速掃描即可
中風險	影響決策或需與他人分享	必須經過四維審核（第 9 章）
高風險	對外發布或涉及法律財務	多階審核 + 人類最終把關

四道防線設計

第一道：輸入隔離

原則：不該碰的資料，一開始就不要給 AI。

建立「不可交給 AI」清單： - 客戶個資、身分證字號、帳號密碼 - 未公開的財務數據、商業機密 - 具法律效力的文件原稿

第二道：流程攔截

原則：高風險動作必須有人類確認點。

在流程中設置「閘門」： - 對外發送 → 必須人工批准 - 財務操作 → 必須人工確認金額與對象 - 客戶互動 → 必須經過審核才能發出

第三道：輸出審核

原則：根據風險等級匹配審核強度。

- 低風險：快速掃描
- 中風險：四維審核（準確性/相關性/完整性/可執行性）
- 高風險：多階審核 + 人類最終決策

第四道：事後監控

原則：已發布的內容仍需追蹤。

- 對外發布的內容建立回查機制
- 定期抽檢 AI 自動產出的內容品質
- 發現問題時要有快速撤回流程

常見風險場景與應對

場景一：AI 自動發文

風險：內容錯誤或不當直接公開。　**應對**：所有自動發文必須經過人類批准閘門；保留隨時撤回的能力。

場景二：AI 處理敏感資料

風險：資料外洩或不當使用。 **應對**：輸入前過濾敏感欄位；使用本地模型處理機密內容；API 調用確認無記錄政策。

場景三：AI 給出投資或決策建議

風險：建議基於錯誤前提，導致實際損失。 **應對**：所有決策類輸出標註確定性等級；人類必須獨立驗證關鍵前提；AI 只當參考，不當決策者。

場景四：AI 產出被當成最終版本

風險：未經審核的內容直接進入正式流程。 **應對**：所有 AI 產出預設標記為「草稿」；正式用途必須經過審核流程解鎖。

風險管理檢查表

每個新任務上線前，過一遍這張表：

- □ 這個任務涉及哪些風險類型？
- □ 風險等級是低/中/高？
- □ 哪些資料不能交給 AI？（輸入隔離）
- □ 流程中哪裡需要人工確認？（流程攔截）
- □ 審核強度是 L1/L2/L3？（輸出審核）
- □ 出問題時怎麼快速修正？（事後監控）

本章重點

1. AI 的錯誤會「自信地犯、放大地傳」——必須主動設防
2. 四類風險：事實、決策、外洩、聲譽
3. 四道防線：輸入隔離 → 流程攔截 → 輸出審核 → 事後監控
4. 每個新任務上線前，過風險檢查表

✅ 行動清單

. ☐
建立你的「不可交給 AI」清單

. ☐
為現有 AI 任務判定風險等級

. ☐
在高風險流程中加入人工確認閘門

. ☐
設計一個快速撤回機制

第 11 章：組織設計——多 Agent 架構怎麼分工

> 核心命題：多 Agent 組織設計只有三件事——角色切得清、交接接得住、卡住有路走；其餘都是工程細節。

第 5 章談了何時從單一 AI 升級到多 Agent 協作。這章要把鏡頭拉近：角色怎麼切、交接怎麼做、卡住時往哪裡分流。

多 Agent 不是為了看起來比較進階。分工的真正目的只有一個——讓每個 Agent 的上下文精簡、角色單一、出錯時不會拖垮整條線。

三個問題決定你要不要拆

不是所有任務都需要多 Agent。三個判斷：

任務是不是超過三個性質不同的步驟？單一 Agent 的 prompt 是不是已經破千字？流程裡是不是需要「生成」和「審核」兩種互相對抗的角色？

三個都沒有，用單一 Agent 就好。有一個成立，就值得想想怎麼拆。

實務上的起手式是：先用一個 Agent 把整條流程跑順，穩了以後再根據瓶頸拆。別一開始就搞架構。

拆完之後的三種協作模式

串聯：A 產出，B 加工，C 審核，人類批准

最直覺的模式。研究→寫作→審核→發布，一路往下傳。

串聯的關鍵在介面：每個 Agent 的輸出格式，必須是下一個 Agent 的預期輸入格式。格式亂了，下游不是猜就是漏。這時候〈AI 接力交接卡〉裡的五格法就派上用場——交接時先講「做到哪裡」，再講「哪些結論可以直接沿用」，最後講「下一手要交什麼」。不靠貼整包背景，而是切清楚交接點。

並聯：多個 Agent 同時跑，一個整合

需要同時查三個主題、比對五份資料時用這個。每個 Agent 獨立處理一個子任務，最後交給一個整合 Agent 合併。

並聯的坑在整合。合併規則沒講清楚，整合 Agent 要嘛照單全收產出矛盾內容，要嘛自作主張丟掉重要資訊。整合 Agent 的 prompt 裡必須寫明：碰到資訊衝突時怎麼處理、哪些來源優先、哪些欄位必須保留。

階層：一個指揮 Agent 拆任務、派工、監控

複雜、動態、需要即時調度的任務適合這種模式。指揮 Agent 不做具體執行，只負責拆分和判斷。執行 Agent 只做自己被分配的那一塊。

階層模式最容易出的問題是：指揮 Agent 介入太深，自己跳下去做，角色又混成一團。設定上要硬性限制指揮 Agent 的輸出只能是「任務描述 + 分配對象 + 完成條件」，不能直接給答案。

角色設計的四條底線

一個 Agent 一個角色。又做內容生成又做品質審核，等於自己審自己，校驗毫無意義。需要對抗結構的時候，生成和審核必須分開。

每個 Agent 的 context 只載必要的知識。研究 Agent 不需要發布流程，發布 Agent 不需要研究方法。塞越多不代表越強，反而會稀釋注意力。

Agent 之間的溝通格式要固定。JSON 或固定區塊的 Markdown 都行，但不能是自由文字。自由文字的交接，每一次都在賭下游的理解能力。

失敗要隔離。一個 Agent 超時就跳過或重試，不拖垮整條管線。關鍵 Agent 掛了要能通知人類。整體流程要有一個緊急停止開關。這不是工程問題，是管理問題——〈AI 委派檢查卡〉裡說得很明白：卡住的判準要先定好，不能等沉默太久才發現已經停了。

交接：分工真正出事的地方

分工架構畫起來很漂亮，但真正決定品質的是交接。

最常見的交接失敗有兩種：一是上一手把整包背景丟過去，下一手不知道從哪裡接；二是已經做過的結論被下一手重新判斷一次，等於白做。

〈AI 接力交接卡〉給了一個很實用的框架。交接時至少要交代三件事：現在做到哪裡、哪些結論可以直接沿用不用重開、下一手的責任邊界到哪裡。素材也只要附最小必要的那包——必讀的、參考的、追溯來源的，分三層就好。讓下一手先能跑，再決定要不要深讀。

交接完成的判準也要先約好。下一手回來時要回什麼格式、什麼狀態算接手成功、卡住時要回報什麼而不是硬做。交接本身也需要被驗收。

升級分流：卡住時往哪裡走

多 Agent 系統一定會卡。問題不是會不會卡，而是卡住以後有沒有分流規則。

很多人的反應是直接重試。重試不動就換個說法再試。再不行就自己接手。

〈AI 升級分流卡〉建議的做法是：先判斷卡點類型，再選一條路。卡點通常只有幾種——任務不清、資料不足、能力不匹配、風險過高、拆工錯誤。不同的卡點，對應不同的路徑：補指令、補資料、換角色、或直接升級給人處理。

一個關鍵原則：一次只做一個主要修正。同時補指令又換角色又改流程，你根本看不出哪個動作有效。

還要問一個冷問題：這個任務值不值得繼續救？低影響又低重用的事情，直接人工收尾比較划算。高風險的事情，一開始就不該全交給 AI。高重用又可教化的，才值得花時間修到好，因為修完能沉澱成模板。

分流之後要有明確的交接語句。改派誰做、要補什麼、哪些限制不能破，都要寫清楚。不是口頭說「你接一下」，而是有結構的交接。累積幾次經驗以後，把卡點和對應路徑記下來，慢慢就能形成自己的分流規則。

錯誤級聯與通知疲勞

上游 Agent 給了錯誤輸出，下游基於錯誤輸入繼續處理，錯誤逐級放大。這在串聯模式裡特別常見。

解法不是加一個總檢查員。每個 Agent 都要獨立驗證輸入，不盲目信任上游。產出的人先自檢，接手的人再驗收，交接失敗時系統明確告警。這和〈AI 交辦閉環卡〉的核心邏輯一致——結果回來先停在 pending review，reviewer 確認後才算 accepted。

通知也一樣。如果每個 Agent 出問題都通知你，你會被淹沒。建立分級：能自動處理的不通知，只記錄在日誌；需要人類判斷的通知但不急；止損觸發或系統異常才立即通知。

從單 Agent 到多 Agent 的擴展路徑

1. 單一 Agent 跑完整條流程，人類全程盯著
2. 穩定後，把「生成」和「審核」拆成兩個 Agent
3. 加入「研究」Agent，形成研究→生成→審核的管線
4. 根據負載需要，加入並聯或指揮結構

每一步都等前一步穩了再推。急不得。

本章重點

角色分工、交接機制、升級路徑——多 Agent 組織設計就這三件事。角色切不清，Agent 之間會互相踩腳；交接沒做好，分工等於白分；卡住時沒有分流規則，最後一定是你自己跳出來收爛攤子。

先用單 Agent 跑順整條流程，再根據瓶頸拆。拆的時候把介面格式定死，把交接模板準備好，把升級分流的路徑想清楚。

行動：找一個你目前已經穩定運作的單 Agent 流程，試著把「生成」和「審核」拆成兩個 Agent。用〈AI 接力交接卡〉的模板寫交接內容，跑一輪看看交接品質。如果卡住了，拿〈AI 升級分流卡〉判斷卡點類型，選一條路走。

✅ 行動清單

1. ☐
   用三個問題檢查現有流程：步驟性質是否不同？prompt 是否破千字？是否需要生成與審核對抗？
2. ☐
   選一個穩定的單 Agent 流程，把「生成」與「審核」拆成兩個角色
3. ☐
   用〈AI 接力交接卡〉寫下交接內容：做到哪裡、哪些結論沿用、下一手責任邊界
4. ☐
   為這條管線寫下分流規則：卡住時的卡點類型，各自對應哪條路

第 12 章：PDCA 在 AI 的應用——持續改進的方法

> 核心命題：AI 用得好不好是管理紀律問題，不是技術問題——雙軌 PDCA 同時改 Prompt 和改策略，能力才會累積。

第 11 章談的是組織怎麼設計、角色怎麼分工。組織搭起來之後，真正決定它能不能長期變好的，不是分工本身，而是有沒有持續改進機制。這就是 PDCA 要解決的問題。

多數人的 AI 能力停在第一天

觀察一個現象：很多人用 AI 一個月之後，跟第一天比起來幾乎沒有進步。原因很簡單——他們沒有建立反饋迴圈。

PDCA 在管理學裡是老東西：Plan（規劃）、Do（執行）、Check（檢視）、Act（修正）。把它搬到 AI 使用上，對應關係很直接：Plan 是決定這個任務該不該交給 AI、用什麼指令；Do 是把指令丟進去、拿到回答；Check 是判斷這個回答品質如何、符不符合標準；Act 是不滿意就改指令再來，滿意就採用。

但問題出在大部分人只做了 Do，偶爾做 Check，Plan 和 Act 幾乎跳過。AI 用得好不好，本質上是管理紀律問題，不是技術問題。

雙軌 PDCA：改 Prompt，也要改策略

只改 Prompt 不改策略，就像減肥只改運動方式不改飲食——遲早撞到天花板。AI 的 PDCA 要跑在兩個層次上。

第一軌：Prompt 的 PDCA

這是最多人熟悉的層次。你給 AI 一個指令，它給你回答，你判斷好壞，然後決定是修正指令還是收下。

改進方向通常有三條：

指令越來越精準。一開始你說「幫我寫一封信」，後來你學會說「幫我寫一封延後交付的商業郵件，語氣專業但不要太硬，總長不超過 150 字，要提到延後原因並提出新的時間點」。這不是因為你變聰明了，而是你跑過幾次 PDCA 之後，知道哪些條件會影響產出品質。

學會切割複雜任務。與其一次要 AI「分析這個產業」，不如每次只問一個維度，累積起來反而更快、更準。這跟方法卡「每個執行週期只推一個主產物」（見方法卡索引）講的是同一件事——同時鋪三條線看起來像高產，實際上只是把注意力切碎。

把好結果沉澱成模板。每次拿到滿意的回答，追問一句：「這次的指令結構能不能變成下次直接套用的模板？」方法卡「AI 交辦閉環卡」的第六步就是在做這件事——把一次好的產出，變成一個可重複使用的管理資產。

很多人會想把這個過程記錄成「Prompt 版本迭代表」，從 v1.0 寫到 v2.0。想法沒錯，但實務上你不需要一整頁空白表格。只要每次修改 Prompt 時，花十秒記一件事：這次改了什麼、為什麼改、效果有沒有變好。一行就夠。累積幾次之後，你會自然看出自己的改進模式。

第二軌：AI 策略的 PDCA

AI 模型在進化，工具在更新，能力邊界在移動。如果你的 PDCA 只改 Prompt，不改整體策略，很快就會發現自己還在用去年的做法處理今年的問題。

策略層的 PDCA 建議每季跑一次：

Plan 階段問自己三個問題：我目前用哪些 AI？成本和表現的性價比如何？市場上有沒有更適合我需求的新工具或新模型？

Do 階段小規模測試。找一個任務，用新工具跑一週，跟舊做法比較。不要一上來就全面替換。

Check 階段看兩個維度：輸出品質有沒有提升？成本和效率有沒有改善？

Act 階段做決定：好就切換，普通就繼續觀察，差就退回舊方案。

這聽起來像常識，但方法卡「工具選擇是最後一步」提醒了一個關鍵順序問題：先畫流程再選工具。如果你連自己的流程 trigger、input、output、review owner 都還沒定清楚，換再多的工具也只是把原本的混亂包成更貴的混亂。

Act 階段最容易被忽略的事：檢查整體流程是否真的在運作

PDCA 不只檢查輸出品質，還要檢查流程本身。

第 5 章和第 11 章談過交付、交接與責任配置。到了 PDCA 這裡，重點不是把那些規則重講一次，而是定期檢查：上游有沒有真的完成交付？下游有沒有真的接住？失敗時有沒有被及時暴露和修正？

方法卡「AI 試點失敗是數據，不是藉口」講得很白：失敗不是壞事，真正危險的是把失敗含糊歸因成「模型還不夠好」，於是什麼流程問題都沒被修出來。一次失敗通常暴露的是 trigger 不清、input 缺欄、review owner 缺席、驗收標準靠感覺。管理者要追問的是：這次失敗能不能定位到具體步驟？如果明天重做一次，哪些條件已經比昨天更清楚？答不出來，代表這次失敗還沒被轉成有用的數據。

同樣道理，方法卡「不要自動化不穩定流程」也跟 Act 有關。流程還沒穩就不要接自動化——把每次做法都不同的流程丟進工具，只會把人為混亂放大成系統性混亂。Act 階段要做的是先收斂例外、砍少分支、釘清責任點，再談自動化。

標準比工具重要

公司裡來了一個明星員工，做事又快又好。三年後這個人離職了，公司好像倒退三年。管理學的核心不是找到最好的人，而是建立不依賴任何人的系統。AI 也一樣。

你的 SOP、工作流程、品質標準如果建立在正確的邏輯上，那麼當新的 AI 出現時，把它接進既有流程就好，不需要整套重來。這才是真正的 AI-Ready：有流程知道什麼適合交給 AI，有標準評估產出品質，有機制讓 AI 融入工作流。工具變得再快，你的系統不需要跟著重啟。

三個核心重點

一、AI 的 PDCA 要跑兩軌：Prompt 層次的持續改進，加上策略層次的每季評估。只改 Prompt 不改策略，會撞天花板。

二、Act 階段不只改產出，還要檢查流程本身——交付有沒有到位、接手有沒有完成、失敗有沒有被正確歸因。這才是修整體系統，不是修單一動作。

三、標準比工具重要。建立一套不依賴特定 AI 的流程和品質標準，工具怎麼換都能快速適應。

本週就能做的事

打開你最近一次用 AI 完成的任務。問自己兩個問題：這次的指令能不能變成模板？這次的過程有沒有一個可以修正的流程斷點？把答案寫成一行筆記。下週再做一次。六週之後，你會看到累積出來的東西比任何 AI 工具都實在。

✅ 行動清單

. ☐
打開最近一次 AI 產出，用十秒寫下一行筆記：改了什麼、為什麼改、效果有沒有變好
. ☐

檢查那次的指令能不能變成模板——可以就存起來，不可以就標記缺什麼條件

. ☐ 日曆上排一個每季 30 分鐘的策略 PDCA：目前用哪些 AI、性價比如何、有沒有更適合的新工具

. ☐ 找一個還在手動跑的重複流程，確認它已經穩定三週以上，再決定是否導入自動化

第 13 章：AI 管理學實踐路線圖

> 核心命題：框架再多，不落地就是零。這章回答：一個人或團隊，怎麼把 AI 管理學真正落地。

落地順序：先穩定，再擴張

三個常見錯誤：太早追求自動化、太早追求多 Agent、太早追求完整系統。結果是表面複雜，實際沒有穩定產出。

> **先把單點任務做穩，再把多個穩定節點串成系統。**

五步順序

步驟	動作	產出
1	選任務	高頻、規則清楚、風險可控的任務清單
2	定標準	目標、輸出格式、驗收條件、禁止事項
3	做流程	固定步驟的 SOP 或 Prompt 工作流
4	設檢查	品質檢查點與人類覆核點
5	做迭代	記錄結果與問題，修正模板流程

前四步沒穩定，第五步不會有效；第一步選錯任務，後面只放大錯誤。

三階段路線圖

階段一：建立基礎

目標：學會判斷、委派與驗收。

完成三件事： 1. **任務盤點**：哪些適合 AI，哪些不適合 2. **標準建立**：至少一個任務有清楚的輸入輸出要求 3. **驗收習慣**：用固定標準檢查，不靠感覺

檢查點：你有可重複的指令模板、不會把 AI 輸出直接當最終答案。沒完成就不要進階段二。

階段二：流程化與系統化

目標：同類任務可反覆產出接近品質的結果。

建立四個元素：

元素	說明
流程模板	高頻任務的固定步驟（收集→大綱→初稿→品質檢查）
知識載體	背景資訊與規則放在固定位置，不每次重講
驗收節點	定義哪些必須人工確認，哪些可自動進下一步
風險邊界	哪些資料不能交 AI、哪些不能直接對外、哪些必須人工決策

檢查點：有數個穩定 SOP、AI 輸出品質可預測、系統知道何時該停。

階段三：持續運營與優化

目標：從「能不能做」升級到「能不能持續做好」。

能力	做什麼
監控	追蹤任務是否按時執行、輸出是否完整
回顧	定期檢查模板有效性、流程出錯率
分工	任務變多時，分給不同角色或 Agent
成本控制	比較模型費、重試成本、修正成本與人工時間

檢查點：系統週期性運作、錯誤能被發現修正、成本品質速度可平衡。

五個落地原則

1. **以任務為起點**——先問「什麼任務、什麼標準、風險多高」，不是先問「用哪個 AI」
2. **以交付為中心**——每個流程定義交付內容、完成標準、誰來驗收
3. **以邊界保安全**——高風險場景先定義：什麼資料不能碰、什麼決策不能自動做
4. **以檢查提品質**——至少檢查：正確、相關、完整、可執行
5. **以迭代促成長**——持續回答：哪步最常出錯？哪個模板最有效？哪裡該人工接手？

常見誤區

誤區	正確理解
部署 = 完成	沒有驗收的自動化只是更快製造錯誤
記憶越多越好	沒分層沒更新的記憶只增加干擾
強模型 = 萬能解	任務風險、穩定性與成本要一起考慮
多 Agent = 成熟	分工是為了穩定可控，不是炫技
只看模型費	要算：模型費 + 重試 + 修正 + 監控 + 你的時間

從個人到團隊

兩者差別不在原理，在治理強度：

- **個人**：找出最值得交的任務、建立模板與檢查表、養成回顧習慣
- **團隊**：制定共同標準與邊界、明確分工與驗收責任、管理知識與權限、建立定期回顧節奏

個人版核心是**自我管理**；團隊版核心是**制度管理**。

本書總結

這本書表面上談 AI，實際上談一種新的管理能力：建立可重複、可驗收、可優化的工作系統。

> **AI 的價值，不來自它會回答，而來自你能不能把它放進一個被管理的系統。**

到這裡，AI 不再只是工具，而是你工作系統中的可管理單位。

✅ 行動清單

1. ☐
   挑一個高頻、低風險、規則清楚的任務
2. ☐
   寫下目標、輸出格式、驗收條件與禁止事項
3. ☐
   拆成固定步驟，形成第一版 SOP
4. ☐
   設至少一個人工檢查點

. □
一週後回顧，修正模板或流程

附錄：實用模板與檢核清單

一、常用 Prompt 模板庫

以下是可以直接複製使用的模板：

模板一：商業郵件

```
【情境】
我是_____（職位），要寫一封給_____（對象）的郵件。

【目標】
_____（這封郵件的主要目的）

【語氣】
_____（正式/專業/友善/簡潔）

【禁止】
_____（特別不想出現的內容）

【原稿】
_____（有的話貼上原文）
```

模板二：市場分析

```
【目標】
分析_____（產業/市場）是否值得進入

【需要的維度】
1. 市場規模和成長率
2. 主要競爭者和市佔率
3. 進入門檻
4. 風險和機會
5. 建議策略

【公司背景】
_____（我們公司的優勢和劣勢）

【禁止】
不要編造數字，所有數字要標注來源
```

模板三：創意發想

```
【任務】
幫我想_____個_____（類型）的點子

【目標受眾】
_____（誰會看到/使用這些點子）

【方向要求】
_____（重點強調什麼）

【禁止】
_____（特別不想出現的方向）
```

使用說明：把這些模板複製到你常用的筆記或知識管理工具裡，每次用的時候拿出來填空就好。

二、AI 任務委派一頁清單

```
【第一步】這個任務的核心價值是什麼？
→ 資訊蒐集 / 初稿生成 / 創意發想 / 人類判斷

【第二步】用 OKR 方式下指令
→ O：目標（給誰看、做什麼用）
→ KR：具體標準（格式、長度、風格）
→ Don'ts：禁止的事項

【第三步】收到 AI 輸出，四維度檢查
→ □ 準確性 □ 相關性 □ 完整性 □ 可執行性

【第四步】滿意？
→ 是 → 問「這個能當模板嗎？」→ 存入模板庫
→ 否 → 修正 Prompt → 回到第二步

【第五步】每週回顧，持續改善
```

三、方法卡：AI 任務指派分流卡

很多團隊不是沒有 delegation、handoff、review，而是更前面就已經走錯入口：該留在 owner 手上的判斷被丟出去、該先補 packet 的任務直接開工、該走 reviewer 的送去 producer。

這張方法卡的用途，就是把「看到一個任務」先變成正確 assignment routing，再進後面的 delegation 鏈條。

五步法

1. **先判斷能不能派**：這是執行問題，還是最終判斷 / 風險承擔問題
2. **只選一個入口**：keep with owner / delegate to producer / route to reviewer / packet first
3. **指定本輪角色**：這一棒是 producer、reviewer、verifier、integrator，還是 owner
4. **先講邊界**：哪些只能 draft、不能直接 accepted；哪些一碰到就要回交 owner
5. **留 routing decision**：把 chosen route、原因、下一個 artifact 寫成一句可追溯決策

一頁版檢核

```
AI 任務指派分流卡

1. 先判斷能不能派
□ 這是執行問題，還是最終判斷問題？
□ 成功標準講得清楚嗎？
□ 失敗了可回收、可 review、可重做嗎？

2. 只選一個入口
□ keep with owner
□ delegate to producer
□ route to reviewer / verifier
□ packet / prep first

3. 指定本輪角色
□ 這一棒是 producer、reviewer、verifier、integrator，還是 owner？
□ 本輪交付物是什麼？
□ 下一站預期去哪裡？

4. 先講邊界
□ 哪些不能越界決定？
□ 哪些只能 draft、不能直接 accepted？
□ 什麼情況必須回交 owner？

5. 留 routing decision
□ 這次為什麼走這條路？
□ 下一個應看到的 artifact 是什麼？
```

一句話總結：你不是只在分配任務，你是在管理 delegation 鏈條的前門路由。

四、方法卡：AI 委派封包卡

很多人不是不會用 AI，而是一開始就沒有把任務交成「可接手的工作包」。結果是背景漏一半、限制晚一輪才補、換人接手又要全部重講。

這張方法卡的用途，就是把一句模糊需求，包成一個可接手、可續做、可驗收的最小委派封包。

五格法

1. **任務本體**：說清楚要完成什麼、交付物是什麼、做到什麼算完成
2. **工作場景**：交代對象、用途、以及它在整體流程中的位置
3. **限制條件**：先講不能做什麼、必須包含什麼、責任邊界在哪裡
4. **素材與缺口**：把可用資料、參考樣本、不可自行補完區一起交代
5. **回報方式**：先約定要回正文、大綱還是問題清單，以及卡點何時停下回報

一頁版檢核

```
AI 委派封包卡

1. 任務本體
□ 要完成什麼？
□ 交付物是什麼？
□ 完成定義是什麼？

2. 工作場景
□ 給誰看？
□ 拿來做什麼？
□ 在流程中的位置是什麼？

3. 限制條件
□ 不能做什麼？
□ 必須包含什麼？
□ 不確定時要怎麼處理？
□ 有哪些責任邊界？

4. 素材與缺口
□ 已提供哪些資料？
□ 可參考哪些樣本？
□ 哪些地方不能自行補完？

5. 回報方式
□ 先回正文、摘要還是大綱？
□ 遇到什麼卡點要先停？
□ 下一步要接閉環、分流還是審核？
```

一句話總結：你不是只在下 prompt，你是在把任務打包成可管理的最小委派封包。

五、方法卡：AI 委派啟動卡

很多團隊不是沒有 packet，而是 packet ready 之後，沒有人把它正式轉成 active work。結果是 manager 以為已 dispatch，實際上只是「可開始」而不是「已開跑」。

這張方法卡的用途，就是把 ready packet 轉成有 active owner、有啟動條件、也有第一個可觀察證據的正式 delegation run。

五步法

1. **先分狀態**：把 ready、dispatched、active 切清楚，不要都叫已交辦

2. **指定 active owner**：講清楚這一輪誰正式接棒、誰只保留 review / accept authority
3. **寫出啟動條件**：明確定義什麼條件滿足後才正式 dispatch
4. **定第一個啟動證據**：先講第一個可觀察工作證據與最晚出現時點
5. **留啟動記錄**：把 packet、dispatch 對象、active owner 與當前狀態寫下來

一頁版檢核

```
AI 委派啟動卡

1. 先分狀態
□ 現在是 ready、dispatched, 還是 active？
□ 還缺什麼才算升到下一狀態？

2. 指定 active owner
□ 這一輪誰正式接棒？
□ 誰保留最終 review / accept authority？

3. 寫出啟動條件
□ 什麼條件滿足後才正式 dispatch？
□ 條件未滿時應維持 ready 還是 parked？

4. 定第一個啟動證據
□ 啟動後第一個可觀察工作證據是什麼？
□ 若未出現，回到哪個狀態？

5. 留啟動記錄
□ packet、dispatch 對象、active owner、當前狀態有沒有寫？
```

一句話總結：委派不是 packet 寫完就算開始，而是有人正式接棒，且第一個工作證據已經出現。

六、方法卡：AI 交辦閉環卡

很多人會把任務丟給 AI，也會抱怨 AI 做得不好，但中間少了一個最關鍵的能力：**把任務管到結束**。

這張方法卡的用途，就是把一次 AI 對話變成一個可交辦、可驗收、可沉澱的閉環。

六步法

1. **定義任務**：說清楚對象、用途、成功標準
2. **先定驗收**：至少檢查準確性、相關性、完整性、可用性
3. **先拿初稿**：第一版只確認方向與結構，不求完美

4. **結構化回饋**：用「保留／刪除／補強」三分法修正
5. **做方向判斷**：問題是方向錯了，還是細節不夠？
6. **沉澱模板**：把成功做法整理成可重用 Prompt

一頁版檢核

```
AI 交辦閉環卡

1. 任務定義
□ 給誰？
□ 做什麼用？
□ 成功標準是什麼？

2. 驗收標準
□ 準確性
□ 相關性
□ 完整性
□ 可用性

3. 初稿判斷
□ 方向對嗎？
□ 結構能用嗎？

4. 修正回饋
□ 保留什麼？
□ 刪除什麼？
□ 補強什麼？

5. 驗收決策
□ 現在能用嗎？
□ 問題在方向還是細節？

6. 模板沉澱
□ 這次成功做法有沒有存下來？
```

可直接複製的回饋模板

```
請根據以下回饋修正：

【保留】
- _____

【刪除】
- _____

【補強】
- _____
```

一句話總結：你不是在反覆試 prompt，你是在管理一個數位助理的交付品質。

七、方法卡：AI 升級分流卡

很多人會管理 AI 的交辦開始，卻不會管理 AI 卡住之後的下一步。結果不是一直重試，就是過早人工接手。

這張方法卡的用途，就是把 AI 任務卡住時的處理，從憑感覺變成可分流的管理動作。

五步法

1. **先判卡點**：任務不清、資料不足、能力不匹配、風險過高、拆工錯誤
2. **只選一路**：補指令、補資料、換角色／換工具、升級人工
3. **算值不值得救**：時間成本、重用價值、出錯代價
4. **清楚交接**：留下明確的下一步與限制條件
5. **沉澱規則**：把卡點轉成下次可直接套用的路由規則

一頁版檢核

```
AI 升級分流卡

1. 先判斷卡點
□ 任務不清
□ 資料不足
□ 能力不匹配
□ 風險過高
□ 拆工錯誤

2. 只選一條主路
□ 補指令
□ 補資料
□ 換角色／換工具
□ 升級人工

3. 判斷值不值得再救
□ 再修一輪比自己做更便宜嗎？
□ 這個任務未來會重複出現嗎？
□ 出錯代價高不高？

4. 清楚交接
□ 下一步由誰做？
□ 要補什麼？
□ 哪些限制不能破？

5. 沉澱規則
□ 下次遇到同類任務，預設該怎麼分流？
```

一句話總結：你不是在判斷 AI 行不行，你是在判斷這個任務下一步該走哪條管理路徑。

八、方法卡：AI 接力交接卡

很多團隊不是沒有產出，而是產出一旦換手，就又從頭講一次。研究者、起草者、reviewer、整合者之間，只要 handoff 不清楚，就會出現重讀、重判、重做。

這張方法卡的用途，就是把「換手」變成可續跑的最小交接。

五步法

1. **先切交接階段**：現在做到哪裡、已完成什麼、下一手要直接接什麼
2. **保留可依賴結論**：哪些可沿用、哪些待判斷、哪些不要重開

3. **講清責任邊界**：下一手是續寫、審核、整合還是驗證；本輪交什麼；不能越界做什麼
4. **只附最小素材包**：必讀、參考、追溯來源分開交
5. **先定接手判準**：什麼算接住、要怎麼回報、卡住時先回什麼

一頁版檢核

```
AI 接力交接卡

1. 交接階段
□ 現在做到哪裡？
□ 已完成什麼？
□ 下一手要從哪裡接？

2. 可依賴結論
□ 哪些已確認可沿用？
□ 哪些仍待判斷？
□ 哪些不要重開？

3. 責任邊界
□ 下一手是什麼角色？
□ 這輪要交什麼？
□ 哪些不能越界決定？

4. 最小素材包
□ 必讀是什麼？
□ 參考是什麼？
□ 追溯來源在哪裡？

5. 接手完成判準
□ 什麼叫接住？
□ 要怎麼回報？
□ 卡住時先回什麼？
```

一句話總結：你不是只在換人接手，你是在管理任務如何被下一手穩定接住。

九、方法卡：AI 委派檢查卡

這張方法卡的用途，就是把已經送出去的 delegation，變成有第一檢查點、有重檢節奏、有 stuck trigger、且結果要先停在 review gate 的受控委派。

1. **先定第一檢查點**：不要只說有進度再回
2. **設持續重檢節奏**：不要靠 manager 想到才問

3. **定卡住回收條件**：避免沉默被誤認成還在做
4. **結果先停在 pending review**：有產出不等於已完成
5. **沉澱檢查規則**：把有效 checkpoint / cadence / authority 留成下次模板

```
AI 委派檢查卡

1. 第一檢查點
□ 何時第一次檢查？
□ 到時至少要看到什麼？

2. 持續檢查節奏
□ 多久重檢一次？
□ 什麼狀態可暫不追？
□ 什麼情況需立刻介入？

3. 卡住回收條件
□ 什麼叫卡住？
□ 卡住時先回什麼？
□ manager 要補什麼或回收什麼？

4. 結果 review gate
□ 回來先停在哪個狀態？
□ 誰有 final review / accept authority？
□ 什麼條件下才算真的完成？

5. 規則沉澱
□ 哪些 checkpoint / cadence 下次可直接重用？
□ 哪些 review gate 不能再省略？
```

十、方法卡：AI 審核整合卡

很多團隊會交辦、會修正、也會做 review，但最後常卡在同一個地方： **內容看過了，卻沒有真正進入正式系統**。

這張方法卡的用途，就是把「看過了」變成「正式接住了」。

五步法

1. **先分清卡點**：現在卡的是品質問題，還是整合問題
2. **指定 canonical target**：先講清楚要進哪個正式檔案 / 系統 / 頁面
3. **留雙層結論**：內容審核與整合審核分開寫

4. **做三種整合動作**：納入、連結、淘汰
5. **留下整合記錄**：讓下次同類任務有固定起跑面

一頁版檢核

```
AI 審核整合卡

1. 先判斷卡點
□ 問題在品質
□ 問題在整合

2. 指定 canonical target
□ 要進哪裡？
□ 它是主文、附錄、模板還是索引？
□ 整合後誰以它為準？

3. 留雙層結論
□ 內容審核：通過 / 需修 / 不採用
□ 整合審核：已整合 / 待整合 / 暫不整合

4. 做三種整合動作
□ 納入正式內容
□ 連結相關入口
□ 淘汰舊版本或降級為參考

5. 留整合記錄
□ 這次補了什麼空缺？
□ 下次同類任務應從哪裡開始？
```

一句話總結：你不是只在審核 AI 產出，你是在管理 AI 成果如何進入正式系統。

十一、方法卡：AI 變更控管卡

很多團隊不是不會開始，也不是不會　review，而是任務一旦開始跑，中途就有人一路加需求、補限制、改判準，最後讓原本可完成的 delegation 逐漸變形。

這張方法卡的用途，就是把「邊做邊改」變成有變更等級、有變更負責人、有正式回寫位置的受控變更。

五步法

1. **先判變更等級**：這是小修、重包，還是重開

2. **指定變更負責人**：不要大家一起順手改
3. **更新正式位置**：packet、routing、review gate、正式主檔哪些地方要回寫
4. **重設下游狀態**：必要時回到 in_progress、重跑 review、重做 handoff
5. **留下變更記錄**：把這次 change-control 沉澱成下次可重用規則

一頁版檢核

```
AI 變更控管卡

1. 先判變更等級
□ 這是小修、重包，還是重開？
□ 它改的是內容細節、packet、routing，還是完成定義？

2. 指定變更負責人
□ 誰有權接受這次變更？
□ 誰負責更新正式描述？
□ 誰不能自行擴 scope？

3. 更新正式位置
□ packet 要不要改？
□ routing / role 要不要改？
□ review gate 要不要改？
□ 正式主檔要不要改？

4. 重設下游狀態
□ 要不要回到 in_progress？
□ 要不要重跑 review？
□ 要不要重做 handoff / integration？

5. 留變更記錄
□ 這次是小修、重包還是重開？
□ 下次同類變更預設怎麼處理？
```

一句話總結：你不是只在改需求，你是在管理任務如何在不失控的情況下持續演進。

十二、方法卡：AI 停泊再啟動卡

很多團隊不是不知道某件事現在不該做，而是不會把「先不要動」管理成正式狀態。結果 ready packet 沒 dispatch 就消失、任務暫時卡住就模糊擱置、之後想接回又得重猜一次。

這張方法卡的用途，就是把「先不要動」變成有停泊原因、有再啟動條件、有 recheck 節奏的受控等待狀態。

五步法

1. **先判狀態**：這是暫停、停泊，還是終止
2. **寫清不能推進的主因**：事件未到、資料未齊、authority 未對齊、packet 過時，還是優先序不對
3. **寫明再啟動條件**：什麼出現時才能重新啟動，且第一步要開哪份資料
4. **設定 recheck 節奏**：event-driven、time-driven，還是 owner-triggered
5. **留停泊記錄**：任務 / packet、停泊原因、再啟動條件、recheck 類型

一頁版檢核

```
AI 停泊再啟動卡

1. 先判狀態
□ 這是暫停、停泊，還是終止？
□ 之後是直接續跑、等條件再啟動，還是不再接回？

2. 寫清不能推進的主因
□ 是事件未到、資料未齊、authority 未對齊、packet 過時，還是優先序不對？

3. 寫明再啟動條件
□ 什麼出現時才能重新啟動？
□ 再啟動後第一步要開哪份資料？

4. 設定 recheck 節奏
□ event-driven、time-driven, 還是 owner-triggered？
□ 下次檢查時點或條件是什麼？

5. 留停泊記錄
□ 任務 / packet 名稱有沒有寫？
□ 停泊原因、再啟動條件、recheck 類型有沒有寫？
```

一句話總結：你不是只在管理等待，你是在管理哪些工作現在不做，才能在對的時候重新做對。

十三、方法卡：AI 結案重開卡

很多任務不是做不完，而是其實這一輪已經該正式結案，卻因為沒有寫清 closeout 邊界與 reopen 條件，結果一直掛在 ongoing；反過來，也常把其實該 reopen 的事誤當成新任務重開一條。

這張方法卡的用途，就是把「這一輪完成了」與「未來若有 trigger 要怎麼重開」拆清楚，讓 closeout 與 reopen 都有正式入口。

五步法

1. **先判這輪是否真的可結案**：目標、交付、review / integration 是否已到位
2. **寫清 closeout 邊界**：這輪到哪裡為止算完成，哪些不再繼續擴 scope
3. **列明 reopen trigger**：哪些新事實、外部變化、明確請求出現時才重開
4. **指定 reopen entry**：若重開，第一步該回哪張卡、哪份資料，而不是直接亂接續做
5. **留 closeout / reopen record**：任務名、closeout date、邊界、trigger、reopen entry

一頁版檢核

```
AI 結案重開卡

1. 先判是否可結案
□ 這輪目標已完成嗎？
□ review / integration 已完成嗎？
□ 現在不做的部分是刻意留到未來，不是遺漏嗎？

2. 寫清 closeout 邊界
□ 這輪完成到哪裡為止？
□ 哪些不再繼續擴 scope？
□ 哪些後續需求不應算在本輪內？

3. 列明 reopen trigger
□ 哪些事件出現時才重開？
□ 哪些變化不足以重開，只能當新雜訊？

4. 指定 reopen entry
□ 若要重開，第一步該回哪張卡或哪份資料？
□ 誰有權決定 reopen？

5. 留 closeout / reopen record
□ 任務名、closeout date、邊界、trigger、reopen entry 有沒有寫？
```

一句話總結：你不是把任務丟進已完成而已，你是在定義這一輪怎麼正式收口，以及未來何時才該重開。

方法卡前門快查圖

如果你現在不是要從頭讀完整條鏈，而是只想先判斷「這一刻先開哪一張卡最快」， 先看：`docs/ai-management-book/method-card-chain-quickstart-map.md`

它專門處理： - 只記得症狀，不記得檔名 - 分不清 assignment / activation / parked / closeout 這些相鄰入口 - 想先找到最短入口，再回到完整方法卡

方法卡鏈條建議順序

若要把這幾張卡串成一條最小管理路徑，建議順序如下：

1. `AI 任務指派分流卡`：先判斷任務該留在 owner、交給 producer、送 reviewer，還是先補 packet
2. `AI 委派封包卡`：再把任務包成可接手工作包
3. `AI 委派啟動卡`：把 ready packet 正式轉成 dispatched / active delegation run
4. `AI 交辦閉環卡`：把交辦、驗收、修正跑成閉環
5. `AI 升級分流卡`：任務卡住時決定下一條路
6. `AI 接力交接卡`：需要換角色、換 agent、換班次時，讓下一手可直接續跑
7. `AI 委派檢查卡`：任務送出後，用 checkpoint / cadence / review gate 管住 delegation 中段
8. `AI 審核整合卡`：最後把通過的內容正式納入主檔
9. `AI 變更控管卡`：中途需求變動時，判斷哪些可直接吸收、哪些要重包、哪些應重開，避免整條鏈悄悄漂移
10. `AI 停泊再啟動卡`：當任務現在不該 dispatch / 不該硬推，但未來仍需接回時，把等待狀態正式標成 parked 並寫好 reactivation entry
11. `AI 結案重開卡`：當任務這一輪其實已完成，不該再掛在 ongoing，但未來仍可能因明確 trigger 重開時，把 closeout 邊界與 reopen 入口正式寫清楚
12. `AI 回顧萃取卡`：當一輪工作已結束並完成整合，不是要重開，而是要把真正值得留下的管理增量抽成可重用內容時，先判斷 delta、落點形狀、主要落點與必要補充資料
13. `AI 雙面同步卡`：當一個改善同時改變對內起跑點與對外教學落點時，先定對內/對外兩個主檔，再做最小必要同步，避免兩面漂移
14. `AI Canonical Owner Routing 卡`：當對內/對外形狀大致已清楚，但仍不確定哪個對內主檔應成為唯一負責入口時，先列相鄰候選，再定唯一主入口與必要補充資料
15. `AI Surface Refresh 卡`：當可重用內容的落點與主檔都已清楚，下一個同類小增量出現時，先判斷是否只需刷新既有主檔與必要補充資料，而不是再展開一輪新卡 / 新索引 / 新說明
16. `AI Navigation vs Doctrine Split 卡`：當小型管理增量看起來像導航修補、卻又可能改到正式規則時，先拆成 `從哪裡開始` vs `以什麼為準`，避免把 findability 修補誤升成新 doctrine，或把真變更只藏在導航文字裡

17. `AI Symptom-First Routing 卡`：當方法卡鏈已經很多、你只記得目前卡住的症狀卻想不起文件名時，先用 symptom 映射到最短入口，優先修導航與檢索，不要把 findability 問題重寫成新 doctrine
18. `AI 任務三軌 Heartbeat × Delivery × Validation 卡`：當需要確認「任務完成」不只是狀態正常、而是真正完成驗證時，先追蹤三軌獨立信號——heartbeat（程序活著）、delivery（內容送達）、validation（結果符合標準）——避免把 heartbeat 綠誤當成任務已完成的證據

十四、方法卡：AI 回顧萃取卡

很多工作完成了，卻只留下結果，沒有留下可重用的管理增量。

這張卡的用途：把一輪已完成工作的經驗，萃成一個可重用的最小增量，先判形狀、再定落點、最後只補必要補充資料。

五步法

1. **找最值得留下的 delta**：只選一個——新判準、邊界、起跑入口或固定動作
2. **判斷落點形狀**：對外可重用內容 / 對內正式做法 / 補充筆記
3. **指定唯一主要落點**：未來同類工作從哪裡起跑，就落在哪裡
4. **只補必要補充資料**：只補會產生漂移風險的地方
5. **留萃取記錄**：任務名、delta、形狀、落點、補充資料、不納入面

一頁版檢核

- □ 這次只留一個最值得重用的 `delta` 嗎？
- □ 落點形狀有沒有誤把一次性背景升成主入口？
- □ 主要落點為什麼是這裡？
- □ 補充資料只補會漂移的地方嗎？
- □ 萃取記錄能讓幾週後的自己看懂嗎？

一句話總結：結案收口，回顧萃取把真正值得重用的管理增量留下。

十五、方法卡：AI 雙面同步卡

一個管理改善若同時改變了對內做法與對外教法，只修單邊會讓兩邊逐漸漂移。

這張卡的用途：當 delta 已確認值得保留，先定對內/對外各自主檔，再做最小必要同步。

五步法

1. **先判這是不是雙面變更**：對內做法受影響面 + 對外教學受影響面
2. **各指定一個主檔**：對內主檔管實際做法，對外主檔管對外教學
3. **只補必要補充資料**：只補會產生漂移的地方
4. **切清正式依據**：實際執行時從哪裡開始、正式做法以什麼為準；教學與分類又以什麼為準
5. **留同步記錄**：delta、對內主檔、對外主檔、補充資料、正式依據分工

一頁版檢核

□ 這次改善同時改了怎麼做與怎麼教嗎？
□ 對內主檔與對外主檔各只有一個嗎？
□ 實際執行的正式依據與教學分類的正式依據有沒有切清？
□ 補充資料只補會漂移的地方嗎？
□ `sync record` 能讓幾週後的自己看懂這次同步了哪兩面？

一句話總結：一個改善若改了做法與教法，先定雙 primary，再做最小必要同步。

十六、方法卡：AI Canonical Owner Routing 卡

當一個 delta 已確定值得留成對內正式做法，但兩三個相鄰入口看起來都能接手，容易造成重複入口。

這張卡的用途：先列候選入口、再定唯一主入口，避免同一 delta 分散寫進多個看起來都合理的入口。

五步法

1. **寫清這次 delta 要解的唯一問題**：起跑入口 / 路由規則 / 封包索引 / 正式索引
2. **列出最像的 2–3 個 owner 候選**：強迫比較各自管什麼
3. **指定唯一主入口**：未來同類工作第一個應該從哪裡重新起跑
4. **只補必要補充資料**：不讓補充資料像第二主檔
5. **留入口分流記錄**：delta、主入口、補充資料、不是主入口的相鄰候選

一頁版檢核

- □ 這次要固定的是起跑入口、路由規則、封包索引，還是正式索引？
- □ 候選入口只有 2–3 個嗎？
- □ 未來同類工作第一個該開哪裡有沒有只選一個？
- □ 補充資料只補會漂移的地方嗎？
- □ 入口分流記錄能讓幾週後的自己看懂為何是它當主入口嗎？

一句話總結：對內落點已清楚，真正要解的是先定唯一主入口。

十七、方法卡：AI Surface Refresh 卡

當可重用內容的落點與主入口都已清楚，下一個同類小增量出現時，若每次都展開新卡或新索引，會讓累積的碎片越來越多。

這張卡的用途：先判斷是否只需刷新既有主檔與必要補充資料，而不是再展開一輪新卡。

五步法

1. **先判增量形狀**：全新領域 → 新卡；現有領域的小填充 → 刷新
2. **指定刷新 primary**：這個增量最適合刷新哪個已有主檔
3. **只補必要補充資料**：index / chain map / playbook 中需要跟著刷新的條目
4. **寫清刷新範圍**：只改這一次真正改變的部分，不擴 scope
5. **留刷新記錄**：時間、刷新主檔、範圍、補充資料

一頁版檢核

- □ 這次真的是全新領域，還是現有領域的小填充？
- □ 刷新主檔為什麼是這裡而不是開新卡？
- □ 補充資料只補需要跟著變的條目嗎？
- □ 這次刷新有沒有悄悄擴 scope？
- □ 刷新記錄能讓幾週後的自己看懂這次改了什麼嗎？

一句話總結：當可重用內容與主入口都已清楚，下一個同類增量先問能不能刷新，而不是再開新卡。

十八、方法卡：AI Navigation vs Doctrine Split 卡

當方法卡鏈已經很多，小型管理增量看起來像導航修補，卻又可能改到正式規則，容易把 findability 修補誤升成新 doctrine，或把真變更只藏在導航文字裡。

這張卡的用途：先拆成「從哪裡開始」vs「以什麼為準」，再決定這次該動哪一個。

五步法

1. **先拆症狀**：是 findability 問題（找不到入口），還是 doctrine 問題（規則本身不對）
2. **導航修補**：只改 description / link / chain order / quickstart map，不動 rule 本身
3. **規則變更**：先走 Canonical Owner Routing 或雙面同步，再動陳述文字
4. **確認哪個 layer**：這次只動導航層，還是規則層也需要跟著動
5. **留 split record**：症狀分類、處置層、修改面

一頁版檢核

```
□ 這次是 findability 問題，還是 rule 本身有問題？
□ 如果只動導航層，這次改完有沒有影響 rule 邏輯？
□ 如果需要動 rule，相關的主入口 / 補充資料有沒有跟著更新？
□ 這次有沒有把 findability 修補偷偷升成新 doctrine？
□ split record 能讓幾週後的自己看懂這次為何這樣處理嗎？
```

一句話總結：小型增量先拆導航層與規則層，避免把 findability 修補誤升成 doctrine，或把真變更只藏在導航文字裡。

十九、方法卡：AI Symptom-First Routing 卡

很多時候，問題不是方法不存在，而是你只記得眼前的症狀，卻想不起正確文件名，結果為了找入口又新寫一份 note。

這張卡的用途：先從症狀回推最短入口，讓 retrieval / routing 問題先被解決，而不是把 findability 問題誤升成新 doctrine。

五步法

1. **先寫症狀，不寫解法**：只描述你眼前卡在哪裡
2. **先分段落**：這是 assignment、delegation、review、handoff、closeout，還是 navigation 問題
3. **找最短入口**：先開最有可能接住它的那一張卡，而不是重寫背景
4. **若找不到，再補導航**：只有在現有入口真的失效時，才補 quick map / symptom-first card
5. **留 symptom-routing record**：症狀、入口、為什麼不是新 doctrine

一頁版檢核

□ 我現在寫的是症狀，不是提前下結論嗎？
□ 這次卡點比較像哪一段管理動作？
□ 我先開的是既有入口，而不是重寫新 `note` 嗎？
□ 真正缺的是導航，還是規則？
□ 幾週後回來，我還看得懂這次是 `retrieval` 問題嗎？

一句話總結：先解入口，再解內容；不要把找不到檔案誤寫成方法不存在。

二十、模板檔案索引

以下範本以獨立檔案形式存在於 `templates/` 目錄：

- `templates/template-kevin-decision-packet.md` — Kevin 決策封包（需由 Kevin 拍板的決策節點用）
- `templates/template-prompt-pack-canonicalization.md` — Prompt Pack 典範化範本（用於收斂多版本 prompt 為統一最優版本）