Claude Code を育てる　No.01

記憶編

Claude Code は、会話を新しくすると前の会話の中身を持ち越しません[1]。この編では、決めたことを1件ずつファイルに残し、目次だけを毎回読ませることで、前のことを覚えている相棒にする方法を説明します。読み終えたら、その日のうちに自分のフォルダで始められるように、手順ごとに「なぜ」「どうやる」「例」「つまずき」を書いています。

> **この編を読み終えるとできること**
>
> - 記憶の置き場と目次を作り、Claude Code に毎回読ませる
> - 決めたことを、あとで使われる形の1件として書く
> - 記憶が増えても、古い物や重なりで混乱しないように手入れする

1 なぜ「1件ずつ」と「目次」に分けるのか

覚えておいてほしいことを全部 CLAUDE.md に書き足していくと、毎回読ませる量が増え続けます。量が増えると、守られにくくなります。言語モデルは、長い文の真ん中にある情報ほど使いにくくなることが分かっています[2]。Anthropic も、文脈は限りのある資源で、量が増えるほど一つ一つへの注意が薄れると説明しています[3]。

そこで、記憶を次の3つの層に分けます。毎回読むのは小さな目次だけにし、中身は関係がある時だけ開きます。出来事を細かい記録として溜めておき、その時に関係のある物だけを取り出して使う形は、長い期間でも一貫した振る舞いにつながることが示されています[4]。「いつも見える小さな所」と「必要な時に取り出す大きな所」に分ける作りも、限られた文脈で長い仕事を続ける方法として提案されています[5]。

層1　毎回、自動で読む
CLAUDE.md（決まり）と、記憶の目次 目次.md
短く保つ。1行は呼び名だけ

↓

層2　関係がある時だけ開く
記憶の本体（1件1ファイル。[[リンク]] でつなぐ）
目次の呼び名か、言葉で探して、要る物だけ

↓

層3　別に置く物（資料と日誌）
資料/（調べた結果・経緯）、days/（毎日の記録）
記憶からは場所だけを指す

> **考え方の芯**
>
> 毎回読ませる物は小さく。要る物は要る時に探して読ませる。記憶にするのは「ファイルやコードを読んでも分からない事」だけ。

2 手順1　記憶の置き場と目次を作る

なぜ

置き場が決まっていないと、Claude は毎回ちがう所に書いたり、書いた物を見つけられなかったりします。置き場を1つにし、その入口として目次を置くと、書く所と探す所が同じになります。

どうやる

1. 作業用のフォルダに `記憶/` を作る
2. その中に目次 `目次.md` を置き、見出しを3〜4個だけ作る（自分・作業の決まり・仕事ごと）
3. 見出しの下は空のままでよい。書き足すのは手順2から

例

```
# 記憶の目次（1行は呼び名だけ）

## 自分
- [Python は初心者](user-python-beginner.md)

## 作業の決まり
- 資料：[見出しは事実だけ](feedback-plain-headings.md)・[表は横幅いっぱい](feedback-full-width-tables.md)
- [大事なファイルは直す前に控え](feedback-backup-before-edit.md)

## 仕事
- [請求は月末締め](project-billing-cycle.md)
- [売上のシートの場所](reference-sales-sheet.md)
```

つまずき

- **目次に中身まで書いてしまう。**目次が本体の写しになり、毎回読む量が増えます。1行は「何の時に使うか」が分かる短い呼び名だけにします。
- **見出しを細かく分けすぎる。**見出しが多いと、どこに書くかで迷います。同じ話題は「資料：」のように1行に束ねます。

3 手順2　1件を書く

なぜ

1つのファイルに1つの事実だけを書くと、古くなった時に、その1件だけを直したり消したりできます。いくつもの事を1ファイルに書くと、一部だけ古くなった時に、正しい所と古い所が混ざります。

どうやる

1. ファイルの頭に3つの項目を書く。`name`（短い呼び名）、`description`（1行の説明。探す時に使う）、`type`（下の4つの種類のどれか）
2. 本文に事実を1つ書く
3. やり方の指摘（feedback）と進行中の仕事（project）には、**Why:**（なぜ）と **How to apply:**（いつ使うか）を1行ずつ足す
4. 関係する記憶があれば、本文に `[[その記憶の name]]` と書いてつなぐ
5. 目次に1行足す

type	書く物	例
user	自分の役目・得意なこと・苦手なこと・好み	Python は初心者。説明は手順を細かく
feedback	やり方の指摘。直された事も、うまくいった事も	資料の見出しに評価の言葉を入れない
project	進行中の仕事の決まり・期限・理由	2026-12-01 から受付の形を変える
reference	外にある物の場所	売上のシートの URL

例

```
---
name: feedback-plain-headings
description: 資料の見出しに評価の言葉を入れない
metadata:
  type: feedback
---
資料の見出しには事実だけを書く（「すごい」「画期的」などは入れない）。
**Why:** 読む人が自分で判断できるようにするため。
**How to apply:** 人に渡す資料の見出しを書く時。社内のメモには当てない。
関係：[[feedback-full-width-tables]]
```

つまずき

- **Why を書かない。**理由が無いと、似ているが違う場面にも当てはめ過ぎます。上の例なら、社内のメモにまで評価の言葉を禁じてしまいます。
- **How to apply を書かない。**「いつ使うか」が無いと、記憶が呼ばれても使われません。
- **「来週」「昨日」と書く。**読む日によって意味が変わります。必ず「2026-12-01」のように日付で書きます。

4 手順3　毎回読ませる

なぜ

目次を作っても、読まれなければ使われません。Claude Code は起動のたびに CLAUDE.md を読みます。CLAUDE.md には `@ファイルの場所` と書いて、別のファイルを一緒に読み込ませる書き方があります[1]。これで、目次だけが毎回読まれ、本体は読まれません。

どうやる

1. 作業用のフォルダの CLAUDE.md に、`@記憶/目次.md` の1行を足す
2. 同じ CLAUDE.md に、記憶の書き方の決まりを短く書く（下の例）
3. 新しい会話を始め、「記憶の目次に何がある？」と聞いて、読まれているかを確かめる

例

```
@記憶/目次.md

## 記憶の書き方
- 1つのファイルに1つの事実。頭に name・description・type
- feedback と project には Why: と How to apply: を1行ずつ
- 書いたら 目次.md に1行。似た記憶があれば、足さずにそれを直す
```

つまずき

- **本体のファイルまで @ で読み込ませる。**毎回読む量が増え、層に分けた意味がなくなります。@ で読むのは目次だけです。
- **同じ決まりを CLAUDE.md と記憶の両方に書く。**片方だけ直すと、食い違います。決まりの正本を1つに決め、もう一方からは場所だけを指します。

5 手順4　1日の終わりに書き足す

なぜ

記憶は、決めた時に書かないと残りません。会話の途中で毎回書かせると作業が止まるので、区切りのよい時にまとめて書かせます。

どうやる

場面	Claude への一言
1日の終わり	今日決まったことを、1件1事実で記憶にして。似た記憶があれば新しく足さずに直して。 目次にも1行足して
やり方を直した時	今のは覚えておいて。理由と、いつ使うかも書いて
うまくいった時	このやり方でよかった。次も同じにするよう記憶にして
前と違う事を言われた時	この件の記憶を探して、古い方を直して

つまずき

- **直された事だけを記憶にする。**うまくいった事も書かないと、次は別のやり方を試されてしまいます。
- **その日だけの事まで記憶にする。**「今日はこのファイルを直した」は日誌（days/）へ。記憶は、次の会話でも効く事だけです。

6 手順5　探して呼ぶ

なぜ

記憶が増えると、目次の呼び名だけでは見つからない事が出てきます。関係する記憶どうしをつないでおき、言葉で探せるようにすると、要る物にたどり着けます。

どうやる

1. 記憶の本文に `[[ほかの記憶の name]]` を書いてつなぐ。1件を開いたら、つながった物を1段だけたどるよう、CLAUDE.md に書いておく
2. 目次で見つからない時は、言葉で探すよう頼む。Claude Code の検索で十分です
3. 探す時は、呼び名と説明に当たった物を先に、本文だけに当たった物を後に見るよう頼む

例

```
資料の見出しのことで決めた記憶を、記憶/ から言葉で探して。
name と description に当たった物を先に読み、[[リンク]] を1段たどって。
```

つまずき

- **リンクをたどり続ける。**何段もたどると、関係の薄い物まで読み込みます。1段で止めます。

7 手順6　手入れ：忘れるは、消すではない

なぜ

記憶は放っておくと、重なり、古くなり、長くなります。古い記憶と新しい記憶が両方残ると、Claude は前と逆の事をします。週に1回ほど手入れをすると、記憶が増えても混乱しません。

どうやる

見つけ方	すること
同じ事を書いた記憶が2件ある	1件にまとめる。元のファイルは `記憶/archive/` に移して残す
1件が長くなった（目安 3KB 超）	事実ごとに割る。経緯は 資料/ へ移し、記憶からは場所だけを指す
しばらく使われていない（目安 30日）	目次の行だけを外す。ファイルは残す
事実が変わった	古い記憶を直す。新しい記憶を足して重ねない

例

```
記憶の手入れをして。重なっている物・3KB を超えた物・30日使われていない物を探し、
直し方の案を表にして。消す前に私に見せて。
```

つまずき

- **使っていない記憶を消してしまう。**あとで要る事があります。目次から外すだけにし、ファイルは残します。
- **手入れを全部まかせる。**まとめる・割る案は Claude に出させ、消す・まとめる前に自分で見ます。

8 記憶にする・しないの見分け

出来事	行き先	わけ
「見出しに評価の言葉を入れないで」と直した	記憶（feedback）	次の会話でも効く。ファイルを読んでも分からない
2026-12-01 から受付の形を変えると決めた	記憶（project）	理由と日付を書く
道具を3つ比べて1つに決めた	資料/ と記憶	比べた表は 資料/、決めた事と理由だけを記憶に
今日直したファイルと結果	days/	その日だけの事
関数の使い方・フォルダの作り	書かない	コードを読めば分かる
パスワード・鍵	書かない	ファイルに残さない

> **迷った時の3つの問い**
>
> 1. 次の会話でも、この事を知らないと困るか
> 2. ファイルやコードを読めば分かるか
> 3. もう同じ事を書いた記憶がないか
>
> 1が「はい」、2と3が「いいえ」の時だけ記憶にします。

9 よくある失敗と直し方

起きたこと	原因	直し
記憶があるのに使われない	目次に載っていない。呼び名が中身と合っていない	書いたら目次に1行。呼び名は「何の時に使うか」が分かる言葉にする
前と逆のことをされる	古い記憶と新しい記憶が両方ある	足す前に似た記憶を探して直す。古い方は archive/ へ
毎回読む量が増え続ける	目次に中身や経緯を書いている	目次は呼び名だけ。経緯は 資料/、毎日の事は days/
決まりが食い違う	同じ決まりを2か所に書いた	正本を1つにし、もう一方は場所だけを指す
記憶が日記になる	その日だけの事まで記憶にした	記憶は次の会話でも効く事だけ
当てはめ過ぎる	Why と How to apply が無い	理由と、使う場面・使わない場面を書く

10 1週間後に見る所

- 新しい会話で、同じ説明をし直す回数が減ったか
- 前と逆のことをされた回が無いか。あれば、その件の記憶を探して直す
- 目次の1行が長くなっていないか。長い行は呼び名に縮める
- 記憶にした物の中に、日誌に行くべき物が混ざっていないか

11 md の一式との対応

この編の手順	一式のファイル	使い方
手順1 置き場と目次	`記憶/目次.md`	見出しを自分に合わせて書き換える
手順2 1件を書く	`記憶/example-feedback.md`	写して名前と中身を変える。例は消してよい
手順3 毎回読ませる	`CLAUDE.md` ・ `AGENTS.md`	`@記憶/目次.md` と記憶の書き方の決まりが入っている
層3 置き場	`資料/README.md` ・ `days/_型.md`	経緯と毎日の事の行き先

出典　[1] Anthropic「Claude Code のドキュメント（Memory）」docs.claude.com　[2] Liu ほか「Lost in the Middle: How Language Models Use Long Contexts」TACL, 2024　[3] Anthropic「Effective context engineering for AI agents」2025　[4] Park ほか「Generative Agents: Interactive Simulacra of Human Behavior」UIST, 2023　[5] Packer ほか「MemGPT: Towards LLMs as Operating Systems」2023

次の編　引き継ぎ編（会話を新しくしても続きから始める）