

# Jev 完全版マニュアル（2026年9月版）

文章を書かない判定専用のAI「Jev」を、仕事の仕組みに組み込むための1冊

---

## このマニュアルの使い方

---

このマニュアルは、noteの記事「Jevとは？」に入りきらなかった内容まで、全部をまとめた完全版です。Jevの基本から、聞き方の設計、世界中で公開された使い方、Claude CodeとCodexでの組み込み方、効果の数字の読み方、私が自分の仕事に組み込んだ4つの実例まで入れました。

- 初めての方は、第1章と第2章から
- すでにつながっている方は、第3章（聞き方）と第4章（線の引き方）から。ここがいちばん差がつくところです
- 使い道を探している方は、第5章（アイデア集）から
- Claude CodeやCodexに入れたい方は、第6章・第7章、そのまま貼れる頼み方は第10章にあります

情報は2026年9月28日時点のものです。Jevは出てから日が浅く、料金や窓口の状況は変わりやすいので、使う前に公式の案内も確かめてください。

---

## もくじ

---

- 第1章 Jevの基本 — 何を返して、何ができないか
- 第2章 始め方 — 本家と3つの窓口、最初の1回
- 第3章 聞き方の設計 — 答えの質は、質問で決まる
- 第4章 確率と線の引き方 — どこまで任せるか
- 第5章 活用アイデア集 — 5つの型と、世界の作例
- 第6章 Claude Codeに組み込む
- 第7章 Codexに組み込む
- 第8章 効果の数字を正しく読む
- 第9章 私の組み込み4例（設計と実測）

- 第10章 そのまま貼れる頼み方集
- 第11章 困ったとき・よくある質問
- 第12章 用語集
- 出典・参考にした公開情報

## 第1章 Jevの基本 — 何を返して、何ができないか

### 1-1. ひとことで言うと

Jevは、TypeSafe社が2026年9月に公開した「判定だけを返すAI」です。文章もコードも作りません。渡した状況について、決められた形の答えと確率を返します。

生成AIが作文の答案を書くAIだとすると、Jevはマークシートに印をつけるAIです。答えの欄はこちらが用意して、Jevはどれに印をつけるか、どれくらい自信があるかだけを返します。

### 1-2. 答えの形は3つ

| 形           | 返すもの                | 向いている問い                |
|-------------|---------------------|------------------------|
| Choice（どれか） | 選択肢から1つ＋全選択肢の確率＋確信度 | 「担当はどれか」「どの手順書を使うか」    |
| Score（何段階か） | 段階の中の位置＋確率の分布＋確信度   | 「影響度はどの段階か」「満足度は」      |
| Noul（何割か）   | はいである確率（0～1）        | 「返金を求めているか」「途中で切れているか」 |

- Choiceの選択肢は最大255個、Scoreの段階は最大10個
- Noulは確率そのものが答えなので、確信度は付かない
- ChoiceとScoreの確信度は「答えがどれだけ1つに集中したか」を表す数字（正解率ではない。第4章）

1-3. 1回で何個でも聞ける

Jevには「状況（state）」を1つ渡し、それについての質問を何個でもまとめて聞けます。質問は同時に処理されるので、1個でも10個でも待ち時間はほとんど変わりません。

問い合わせ1通に対して、「種類はどれか」「急ぎか」「人と話したがっているか」「怒っているか」を1回で聞く。これがJevらしい使い方です。

ただし、質問どうしはお互いの答えを見られません。「種類が請求なら、次は請求の細かい区分を聞く」のように、前の答えで次の質問が変わるときは、2回に分けて呼びます。

1-4. できないこと

- 文章を書く、要約する、会話する
- 画像・音声・動画を直接読む（読めるのは文字だけ。動画なら先に文字起こしをしてから渡す）
- 最新の情報を自分で調べる（渡されたものだけで判定する）
- 長すぎる文を読む（状況と最も長い質問を合わせて約3万2,000トークン、1回全体で6万4,000トークンまで）

1-5. 数字で見るJev（公式の案内）

| 項目      | 値                                                                                                          |
|---------|------------------------------------------------------------------------------------------------------------|
| 返ってくるまで | 70～500ミリ秒（公式の主張。主に米国西海岸からの測定）                                                                              |
| 料金      | 入力100万トークンあたり0.042ドル。出力は無料                                                                                 |
| 回数の上限   | 公式表記は1分1,200回。ただし「需要に合わせて動的に調整中」と注記あり                                                                      |
| 言語      | 英語がいちばん得意。ほかの言語は自分のデータで確かめるよう案内されている                                                                       |
| モデル名    | 本家は <code>jev-latest</code> （2026年9月時点の実体は <code>jev-1.13.0</code> ）。Vercelでは <code>typesafe-ai/jev</code> |

## 1-6. なぜ文章を書かないのか

Jevを作ったTypeSafe社のCEO、ディオゴ・アルメイダ氏は、ChatGPTの土台になった学習法（RLHF）とInstructGPTの共同発明者として紹介されています。発表では、人の好みに合わせて鍛えるやり方が、会話を上手にした一方で3つの弱点を生んだと語られています。

- 答えが似たものに寄っていく
- 自信たっぷりに間違える
- 同じ問いでも答えが安定しない

そこで、確信の数字そのものが正しくなるように鍛える別の学習法（RLCD）を作り、最初の公開モデルとして出したのがJevです。TypeSafe社の呼び方では、じっくり考えて文章にする従来のAIが「システム2」、とっさに決めるJevのようなAIが「システム1（System One）モデル」。

決めた選択肢の中からしか答えないので、形が崩れる失敗は起きません。発表では、決めた形で答えられなかった割合がJevは0.00%、Claude Sonnet 5が12.60%、Claude Fable 5.1が7.99%と示されています（TypeSafe社の計測）。ただしこれは形の保証で、中身の正しさの保証ではありません。

---

## 第2章 始め方 — 本家と3つの窓口、最初の1回

### 2-1. 本家の新規登録は9月28日に再開

| 日付（日本時間） | 出来事                                |
|----------|------------------------------------|
| 9月16日    | TypeSafe社がJevを発表（順番待ち制）            |
| 9月21日    | 「誰でも使えるようにした。順番待ちなし」と発表            |
| 9月22日    | 申し込みが殺到し、新規登録を一時停止（登録済みの人は継続して使える） |
| 9月28日    | 受け入れられる量を増やし、新規登録を再開               |

9月28日に私が本家で登録したときは、登録直後の残高が0ドルで、5ドル分の無料クレジットは付きませんでした。残高が0のままだとキーを作れないので、Billingでカードを登録して入金してから、API Keysでキーを作ります。お金をかけずにまず試すなら、次のVercelが手軽です。

## 2-2. 本家のほかに使える3つの窓口

| 窓口                    | 特徴                                                                              | モデル名                             |
|-----------------------|---------------------------------------------------------------------------------|----------------------------------|
| Vercel AI Gateway     | 本家と同じ形と呼べる接続先がある。公式スキルがそのまま使える。無料期間は2026年9月25日で終了。カード登録が必須で、登録すると毎月5ドル分の無料クレジット | <code>typesafe-ai/jev</code>     |
| OpenRouter            | いろいろなAIを1つのキーで使えるサービス。すでに使っている人向け                                               | <code>typesafe/jev-latest</code> |
| Cloudflare Workers AI | 順番待ちなし。Cloudflareでツールを動かしている人向け                                                 | <code>typesafe/jev</code>        |

Claude CodeやCodexから使うなら、Vercelがいちばん簡単です。接続先 `https://ai-gateway.vercel.sh/typesafe` に替えるだけで、本家向けの公式スキルがそのまま動きます。

## 2-3. Vercelでの手順（要点）

1. vercel.comでアカウントを作る（Googleのアカウントで続けるのが早い）
2. 左のメニューの「AI Gateway」を開く
3. 「Create Key」でキーを作り、1回だけ表示されるキーを保存する
4. 「Add a Card」でカードを登録する（未登録だと「有効なクレジットカードが必要です」で止まる）
5. 「Budgets & Spend」で月の上限を決める（私は3ドル、75%でメール通知）

画面つきの詳しい手順は、1本目の記事と「Jevを今日から使う 導入手順書」にまとめています。

## 2-4. キーの置き方

キーは、人に見られると勝手に使われる合言葉です。会話には貼らないこと。ファイルに書いて、その場所をClaude CodeやCodexに伝えるのが安全です。

```
AI_GATEWAY_API_KEY=（ここにキー）
```

このファイルはgitなどで共有しない場所に置きます。Claude CodeやCodexには「キーは（ファイルの場所）の `AI_GATEWAY_API_KEY` を読んで。値は画面に出さないで」と頼みます。

2-5. 最初の1回で確かめること

- 答えが返ってくるか（403なら、カードの登録を確認）
- 請求の欄（Vercelの応答には cost と marketCost が付く。2026年9月25日までの無料期間中は cost が0だった）
- かかった時間

私の最初の1回は、問い合わせ1通に質問2つで、入力483トークン、通信込み0.79秒、費用0（本来は約0.00002ドル）でした。

第3章 聞き方の設計 — 答えの質は、質問で決まる

Jevは速いぶん、聞き方が雑だと、速くて的外れな答えが返ってきます。この章は、Jevを使ううえでいちばん大事なところ です。

3-1. 1問で1つのことだけ聞く

「この問い合わせに問題はあるか」では、料金の問題なのか、怒りなのか、規約違反なのかが決まりません。「返金を求めているか」「サービスが使えなくなっているか」のように分けます。

ただし、分けすぎて文脈を壊さないこと。「請求への不満」と「返金の依頼」を見分けたいなら、その違いが分かる前後の文は残して渡します。

3-2. 「何をもって、はいとするか」まで書く

同じ問い合わせで、聞き方だけを変えた私の実測です。

| 質問の文               | 急ぎである確率 |
|--------------------|---------|
| 急ぎかどうか             | 0.75    |
| すぐに対応が必要な急ぎの問い合わせか | 0.96    |

「急ぎ」の中身を足しただけで、答えがはっきりしました。Noulなら「はい」と「いいえ」の意味を、Choiceなら選択肢ごとの説明を書けます（criteria）。境目になりそうな例もここに書きます。

- よい例：「顧客は代金の返還を求めているか。料金への不満だけなら、いいえ」

- 弱い例：「返金？」

### 3-3. 質問の名前（ID）はJevに届かない

プログラムで使う質問の名前（例：refund\_requested）は、Jevには送られません。名前で意味を伝えつつも、質問の文が「該当しますか？」だけなら、Jevには何を聞かれているか分かりません。意味は全部、質問の文とcriteriaに書きます。

### 3-4. 状況（state）は、名前を付けて分けて渡す

顧客のメッセージと会社の規約を1つの文に混ぜると、どちらの話かが曖昧になる。これがよくある失敗です。

```
state:
  customer_message: （問い合わせの本文）
  order: （注文日・金額）
  policy: （返金の規約）
  質問: 「`customer_message` は返金を求めているか」
```

質問の中で `customer_message` のように名前を指すと、何を読んで何を決めるかがはっきりします。

### 3-5. 「どれも無い」を必ず用意する

選択肢に当てはまるものが無いとき、Jevは用意された中からいちばん近いものを選んでしまいます。「その他」「情報が足りない」「該当なし」を入れて、そこに入ったものは人が見る流れにします。

### 3-6. Scoreの段階は、それぞれを単独で読める文にする

「1～5点で」だけでは、3点が何なのかが決まりません。

- 作業が続けられる
- 代わりの方法はあるが、手間が増える
- 主な業務が止まり、代わりの方法がない

のように、段階ごとの状況を書きます。「前の段階より深刻」のような書き方は、各段階が単独で評価されるので向きません。

### 3-7. 2段で聞く（私の実測から）

自分のメモ827件を、「文章を書くどの場面で役立つか」の7つに仕分けたとき、場面ごとの質問だけを聞くと、販売の文章と関係のないメモでも0.5～0.6あたりの数字が並びました。

そこで、先に「そもそも販売の文章を書くのに役立つか」を聞き、それが0.6を超えたものだけ、場面の答えを採る形にしました。これで仕分けがはっきりしました。大きな問い→細かい問い、の2段にするのは、どんな仕分けにも使える工夫です。

### 3-8. 日本語で使うとき

公式には英語がいちばん得意と書かれています。私が日本語の問い合わせ・日本語の記事で試した範囲では、仕分けに困ることはありませんでした。精度が気になる仕事では、同じ材料で次の3つを比べてみてください。

- 質問も材料も日本語
- 質問だけ英語、材料は日本語
- 日本語の材料に英訳を添える

遠回しな言い方（「できれば返金していただけると助かります」）を拾えるか、否定（「返金は不要です」）を取り違えないかも、試す材料に入れておきます。

---

## 第4章 確率と線の引き方 — どこまで任せるか

---

### 4-1. 数字の意味を取り違えない

- **Noulの0.8**：「はいである確率が0.8」。深刻さが80点、という意味ではない
- **ChoiceとScoreの確信度0.9**：答えの確率が1つに集中している、という意味。「10回中9回は正しい」という保証ではない
- **確信度が高くても、まるごと間違っていることはある**

## 4-2. 線は、自分のデータで引く

公開されている精度の検証（Kaggleでの分類の比較）では、何も学習させていないJevが、その分類のために専用に学習させたモデルにわずかに届かない程度、上位の生成AIと比べても見劣りしない結果が報告されています。同じ報告で、いちばん精度が出たのは「Noulの確率に、自分のデータで線を引き直す」やり方で、0.5で切ったままだと大きく劣りました。

線の引き方は地味ですが、次の手順が確実です。

1. 自分の仕事の材料を20件ほど集める（普段の例のほか、遠回しな言い方・否定・情報不足・複数の要望を混ぜる）
2. 自分で正解を付ける
3. Jevに当てて、確率と正解を並べる
4. 「ここより上なら自動」「ここより下なら人」の線を決める

## 4-3. 3段で運用する

| 確率・確信度       | 扱い                    |
|--------------|-----------------------|
| 高い（例：0.85以上） | 自動で進める                |
| 中くらい         | 今までどおりの処理（人・生成AI）に任せる |
| 低い・該当なし・エラー  | 人が確かめる                |

公開されている組み込みの中には、選んだ答えの確率と確信度の両方に0.85以上を求め、満たさないときは自動で進めない作りのものもあります。

## 4-4. いきなり本番に入れない —「影で動かす」

判定だけさせて、実際の処理には使わない期間を数日つくります。「もしJevに任せていたら何を選んでいたか」だけを記録して、違和感のある判定がないかを見てから本番に入れます。公開されているモデル振り分けの組み込みにも、この「影で動かす」モードを持つものがあります。

## 4-5. 最後の判断は人に残す

消す・送る・公開する。この3つのように、やり直しがきかない操作は、確率が何と出ても最後は人が見る。Jevは判断を安くする道具で、責任を肩代わりする道具ではありません。外から取り込んだ文に「指示を無視しろ」のような乗っ取りの文が紛れていることもあるので、キーの扱いや送信・削除の権限を、Jevの判

定だけに任せない作りしておくのが安全です。

---

## 第5章 活用アイデア集 — 5つの型と、世界の作例

---

公開から1週間で、世界中の開発者がJevを組み込んだ試作を公開しました。ここでは、それを5つの型に整理し、それぞれ作例と、仕事への応用案を並べます。作例の数字は、公開した人が示した条件での報告値です。

### 型① 仕分ける — 届いたものを、行き先ごとに分ける

**世界の作例** - 配信中のコメントを「質問・感想・要望・その他」に振り分けるデモ（日本の開発者 こぎぞさん） - 表の列に「Urgency（緊急度）」と書くだけで、各行の内容から緊急度を付ける表計算の試作（nader dabit さん） - データベースで「在宅勤務ができそうな人」のような、意味で絞り込む仕組み（Zachi さん） - 会議の発言1つずつに「業務の話か」「業務フロー図に足すか・直すか・消すか」を聞き、発言しながら業務フロー図ができていくデモ（岡村 正樹 さん。確率50%以上だけを生成AIに回し、低いものは「要確認」に残す作り）

**仕事への応用案** - 問い合わせフォームを「受講相談・制作依頼・取材・営業・その他」に分ける - アンケートの自由記述を、良い・普通・悪い + 理由の種類に分ける - 依頼一覧を「今日確認・通常・情報不足」に分ける

### 型② 採点する — 同じ物差しで、何度も測る

**世界の作例** - 投稿の伸びやすさを61の質問で約1秒・約0.0004ドルで採点する試作（Rob Hallam さん） - コードの変更を「秘密の情報が入っていないか」「テストを消していないか」など14項目で点検（Paolo Rosson さん。1件約0.00007ドル） - 書いた文章を行ごとにルールで点検し、引っかかった行だけ行番号つきで出す道具 slop-grader（lukstei さん） - LangChain社は、Jevを「採点役」にして、人の正解との一致と、同じものを何度採点してもぶれないかを、生成AIと比べて公開しています（本人たちは「有望だが初期段階」「狭い検証」と注記）

**仕事への応用案** - 記事の下書きを「誰向けか分かるか」「結論が先にあるか」「数字に出典があるか」で観点別に点検 - タイトル案10本を同じ物差しで比べる（採用は人が決める） - 納品前のチェックリストを、Jevに1項目ずつ当てる

採点のコツは、「良いか悪いか」を1問で聞かないこと。観点を分け、平均点で重大な問題を隠さない。「読みやすいが、重要な数字に根拠がない」ものは止める、のように必須条件を別に持ちます。

### 型③ 選ぶ — たくさんの候補から、当たりを選ぶ

**世界の作例** - 話している内容に合うスライドを自動で出すプレゼン補助（Rinte さん） - 「さっきダウンロードしたPDF」のような説明から、目当てのファイルを上位に出すランチャー（nader dabit さん） - 動画生成で、人物・背景・動画ごとに合うモデルを選ぶ（Higgsfieldとの組み合わせ。人物にSoul 2を82%で選ぶ、など） - 手順書（スキル）の一覧から依頼に合う1つを選ぶ。TypeSafe社の実験では、182種類の手順書・488件の依頼で、選び間違いが16.8%→7.3%、要らない手順書を使う割合が9.8%→4.0%に下がった（Hermesの手順書群・Claude Haiku 4.5・Jev 1.12での結果）

**仕事への応用案** - 社内資料の検索結果から、質問に答える根拠として使える資料を選ぶ（「そもそも制度がない」という規約も大事な根拠として残す） - 請求書の金額・連絡先を、先にプログラムで候補を抜き出してから、Jevに「どれが請求総額か」を選ばせる。存在しない数字を作らせない形

### 型④ 止める — やっていいかを、手前で確かめる

**世界の作例** - 実行する前に「このコマンドは危ないか」を聞き、危ないものだけ止める組み込み（jev-use など） - AIが作った原稿の主張と、引用元の本文を並べて「引用がこの主張を支えているか」を確かめる（引用文が原文にあるかは文字の照合で、支えているかはJevで、の二段構え。TypeSafe社の作例にある形） - 会話の履歴に入る前に、危ない操作を止める「ゲート」と、依頼を振り分ける「ルーター」の2層にする設計（TypeSafe社のガイド「Jev Engineering」）

**仕事への応用案** - 送信前のメールに「添付を忘れていないか」「宛先の会社名と本文の会社名が合っているか」を聞く - 公開前の記事に「社外に出してはいけない名前が入っていないか」を聞く - 調べもので読んだ記事が「途中で切れていないか」を聞く（第9章の実例）

### 型⑤ 操作を決める — 画面やゲームの次の一手を選ぶ

**世界の作例** - ブラウザ操作の道具 Browser Use と組み合わせて、Googleフライトで便を検索。約7秒・約0.0039ドル（ページの部品の一覧からJevが次の操作を選び、文字の入力は小さな生成AIが担当） - 声で指示してブラウザを操作。1回の判断は約300ミリ秒・約0.0002ドル（Moritz Kremb さん） - 画面の文字を読み取って、クリック先をJevが選ぶコンピュータ操作。判断は約90ミリ秒（Milind S さん） - iPhoneのシミュレーター操作、Minecraft、スーパーマリオ、Slay the Spire 2、Subway Surfers（50ゲーム同時）など、ゲームでの作例多数 - TypeSafe社のDoomのデモ：毎秒およそ10回の判断で、1時間約7ドル

どの作例も、Jevが画面を見ているわけではありません。画面の状態を文字にする仕組みが前にあり、Jevは「用意された候補からどれを押すか」だけを決めています。「速く判断できる」と「安全に任せられる」は別で、売買や運転のような作例は、試作として見るのが正確です。

## Claude Code・生成AIの中で使う型（第6章へ）

- 会話の履歴から、要らない実行結果を捨てる
- 依頼の重さを見て、軽いモデルと重いモデルに振り分ける
- 見直しの指摘を「今直す」と「あとでいい」に分ける

---

## 第6章 Claude Codeに組み込む

---

### 6-1. 考え方

Claude Codeは、作業中にたくさんの小さな判断をしています。この古い結果はもう捨てていいか。この依頼は軽いモデルで足りるか。どの手順書を読むか。このコマンドは実行して大丈夫か。どれも「はい・いいえ」か「どれか」で答えが出るので、ここをJevに任せます。

分担は3つです。

- 作るのはClaude Code（文章・コード・仕組み）
- 決めるのはJev（意味を読んで、どれか・何割か）
- 数えるのはプログラム（計算・日付・件数・条件分岐）

Jevを入れて減るのは、小さな判断にかかるお金と時間です。Claude Codeが書く文章やコードの出来は、Claude Code自身のままです。

### 6-2. まず公式スキルを入れる

```
/plugin marketplace add typesafe-ai/skills
/reload-plugins
/plugin install typesafe@typesafe-ai
/reload-plugins
```

最後に「1 plugin・1 skill」と出れば完了です。公式スキルは「Jevの使い方の説明書」で、入れても既存の動きは変わりません。「この仕分けをJevでやる形にして」と頼んだときに、Claude Codeが説明書を読んで聞き方を組み立てます。Vercel経由なら、頼むときに接続先・モデル名・キーの場所を添えます（第10章）。

### 6-3. 公開されている組み込み一覧

どれも公式のものではなく、個人の開発者が公開しているものです。入れる前に、それぞれのページで最新の入れ方と注意書きを確かめてください。

| 何に効くか                                     | 名前（公開者）                                    | 入れ方                                                                                                                                                    | 向くとき                  | 気をつけること                              |
|-------------------------------------------|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|--------------------------------------|
| 会話を軽くする（要らない実行結果を捨てる。要約しないので、残った部分は原文のまま） | fast-jev-<br>compaction<br>（tamaratran さん） | <code>claude plugin marketplace add tamaratran/fast-jev-compaction</code> → <code>claude plugin install fast-jev-compaction@fast-jev-compaction</code> | 1つの作業を何時間も続けるとき       | 捨てる基準を強くしすぎると、あとで見たい結果まで消える。最初は既定のまま |
| 長い出力を読む前に削る                               | jev-pruner（同）                              | <code>claude plugin marketplace add tamaratran/jev-pruner</code> → <code>claude plugin install fast-jev-output@fast-jev-output</code>                  | 長いログを出すコマンドが多いとき      | 元の出力は手元に残る作り                         |
| 関係ない結果を見出しだけに畳む（あとで取り出せる）                 | winnow<br>（GhalebDweikat さん）               | 公開ページを参照                                                                                                                                               | 捨てるのが不安なとき            | —                                    |
| 手順書を必要なときだけ読む                             | jev-skill-suggestion<br>（Daniel San さん）    | <code>npx claude-code-templates@latest --mod productivity/jev-skill-suggestion</code>                                                                  | 手順書を10個以上持っているとき      | 似た名前の手順書は取り違えやすい。説明文を短く書き分ける         |
| 依頼の重さでモデルを振り分ける                           | jev-model-router<br>（hityyhz さん）           | <code>npx claude-code-templates@latest --mod productivity</code>                                                                                       | 1日に何十回も依頼し、軽い作業が混ざるとき | 依頼の内容が判定のため外に送られる。軽いと判定されて浅          |

| 何に効くか                              | 名前（公開者）                              | 入れ方                                                                                        | 向くとき                      | 気をつけること                |
|------------------------------------|--------------------------------------|--------------------------------------------------------------------------------------------|---------------------------|------------------------|
|                                    |                                      | <code>y/jev-model-router</code>                                                            |                           | い答えになることもある            |
| 普通の言葉でJevに聞く                       | jev<br>(sebastianbugalさん)            | <code>/plugin marketplace add sebastianbugal/jev</code> → <code>/plugin install jev</code> | 見直しの指摘を「今直す」と「あとで」に分けたいとき | 安全に関わる指摘は、確率が低くても自分で見る |
| 聞き方の設計を教える                         | building-with-jev-skill (dbreunigさん) | 公開ページを参照                                                                                   | 自分でJevの呼び出しを作るとき          | —                      |
| 文章が要らない工程だけJevに渡す（自信が足りないと生成AIに戻す） | jev-use<br>(shitianfangさん)           | <code>npx -y jev-use install</code>                                                        | 消す・書き換える作業を任せるとき          | 「危なくない」と出ても安全の保証ではない   |
| 書いた文章を行ごとに点検する                     | slop-grader<br>(luksteiさん)           | <code>npx @lukstei/slop-grader@latest -r no-ai-slop</code> 自分のファイル.md                      | 同じクセを何度も指摘されているとき         | 引かなかった行が全部悪いわけではない     |

Xなどからコピーしたコマンドは、`--` が長い横棒1本に化けていることがあります。動かないときは、半角のハイフン2本に打ち直してください。Claude Code向けの組み込みの一覧は、mrjev.com（非公式のまとめサイト）にもあります。

## 6-4. モデル振り分けの設計で知っておくこと

- **メインのモデルは、途中で切り替えない**：jev-model-routerは、メインのモデルを切り替えるのをセッションの始まりだけにしています。途中で替えると、それまでの会話をためておく仕組み（キャッシュ）が効かなくなり、かえって高くつくことがあるためです
- **判定は新しい依頼の最初の1回だけ**：ファイルを読むたびに聞き直すと、Jev自身の手間が増える
- **大事な作業は自分で指定する**：「強いほうでやって」と書けば、そちらが優先される作りが多い

## 6-5. 入れるかどうかの目安

いま困っていない場所には入れない。会話が重くて困っていないなら「会話を軽くする」は要らないし、手順書が数個しかないなら「選ぶ」仕組みも要りません。困っている場所が思い浮かばないなら、公式スキルだけ入れておき、使いたい場面が来たときに頼めば十分です。

---

## 第7章 Codexに組み込む

---

### 7-1. 入れ方

```
npx skills add typesafe-ai/skills --skill typesafe-ai
```

インストール先を聞かれたらCodexを選びます。標準はプロジェクト単位で、全体に入れるなら `-g` を付けます。入ったかどうかはCodexの `/skills` で確かめられ、`$typesafe-ai` で明示的に呼べます。表示されないときは、インストール先を確かめてCodexを開き直します。

### 7-2. 入れただけでは、Codexは賢くならない

公式スキルは、JevのAPIや聞き方をCodexに教える手順書です。「Jevを使う処理」を作ってはじめて連携になります。

たとえば返金の問い合わせなら、分け方は次の4つ。

- **Jev**：返金を求めているか
- **プログラム**：購入からの日数、返金できる上限
- **Codex**：返信文の下書き
- **人**：実際の返金の承認

### 7-3. スキルとSDKは別物

スキルはCodex向けの手順書、SDKは作ったプログラムからJevを呼ぶための部品です（Pythonなら `pip install typesafe-sdk`）。「スキルが入った」と「プログラムからJevが呼べた」は別なので、架空の問い合わせ1件で、実際に答えが返るところまで確かめます。

### 7-4. Codexへの頼み方の要点

- 「精度を上げて」ではなく、何を正解とし、何を自動にしないかを渡す
- 「古い記憶でAPIを書かない。最新の説明書を確認し、確認できない項目は作らない」と書く（公式スキル自体もそう求めている）

- 最初は架空のデータで試す。遠回しな依頼・否定文・情報不足・「指示を無視しろ」を含む文を混ぜる
- 調整に使うデータと、最後の確認に使うデータを分ける
- 「未実行のテストを成功と書かない」「キーが使えないときは未検証と書く」を入れる

そのまま貼れる文は第10章にあります。

## 第8章 効果の数字を正しく読む

Jevの話題には、「〇〇%削減」という数字がたくさん出てきます。ただ、何を測った数字かを取り違えると、入れたあとで「思ったほど減らない」となりがちです。

### 8-1. 同じ「削減」でも、測っているものが違う

| 公開されている数字                      | 何の数字か                                                     |
|--------------------------------|-----------------------------------------------------------|
| 約82%減（187,635→33,447トークン、1.4秒） | 会話を軽くする組み込みで、1回の整理のあとに次へ持ち越す会話の量（fast-jev-compaction の実例） |
| 約91%減（約100万→約8万6,000トークン、1秒）   | TypeSafe社のガイドに載っている、Alex Volkov 氏の整理の実例                   |
| 約60%減                          | Codex向けのモデル振り分けを、7日間・237回ぶんの履歴で再計算した推定コスト                 |
| 26.3%減                         | Claude向けのモデル振り分けの推定コスト（100件の評価で、力不足のモデルを選んだのは0件）          |
| 100万トークンあたり0.042ドル             | Jev自体の入力の単価                                               |

82%は会話の量、60%と26.3%はお金です。足し算はできません。

### 8-2. 会話の量が82%減っても、全体が82%減るわけではない

全体の入力のうち、過去の履歴が占める割合で効き方が変わります。

- 履歴が全体の6割なら： $60 \times 0.82 \div 49$ 、全体で約半分
- 短い作業で履歴が4割なら： $40 \times 0.82 \div 33$ 、全体で約3分の1
- 長い作業で履歴が7割なら： $70 \times 0.82 \div 57$ 、全体で6割近く

作業が長いほど効く、ということです。

### 8-3. 逆効果になる3つの場面

1. **キャッシュを壊すと、かえって高くなる**：履歴を頻繁に書き換えると、ためておいた分が使えなくなり、読み直しが増える。トークンの数だけでなく、請求の額で確かめる
2. **Jevを外しても同じだった実験がある**：あるモデル振り分けの実験では、Jevを外して検索（retrieval）だけにしても、ほぼ同じ結果だったと作者自身が書いています。既存のルールで足りているなら、Jevを挟む意味がない場合もある
3. **Jev自身も読む量のぶんだけお金と時間がかかる**：1回に読める量には上限があるので、長い会話を丸ごと渡すのではなく、1件ずつ・かたまりごとに判定する

### 8-4. 自分の環境で測る

よその数字は、よその環境の数字です。同じ作業を、Jevなしで1回、Jevありで1回。請求と時間を並べるのが、いちばん確かな測り方です。

---

## 第9章 私の組み込み4例（設計と実測）

---

私は、自分の仕事の仕組みの中で「毎回同じ物差しで決めている判断」を4つ選び、Jevに置き換えました。どれもVercel経由で、1回0.3～0.8秒。試したのはVercelの無料期間中だったため、請求は0円でした。

### 9-1. 依頼の重さを判定して、手伝い役のモデルを選ぶ

それまで：依頼文に「修正」「確認」などの言葉がいくつあるかを数えて、5段階の重さを決めていた。短い文はいつも「挨拶なみ」になるなど、言葉に引っぱられて外れることがあった。

**Jevへの聞き方**：Choiceで、次の5つから選ばせる。

- 挨拶・お礼・相づち。作業の中身が無い
- 読む・探す・一覧を出す・質問に答えるだけ。何も変更しない
- 1～2ファイルの小さな修正、既存の手順どおりの作業
- 新しい機能やスクリプトを作る、複数ファイルの変更、成果物を一から作る
- 仕組み全体の設計、大きな作り替え、本番に関わる判断、原因不明の不具合の調査

**結果：**「このフォルダに何が入ってる？」は、言葉を数える方式では「挨拶なみ」、Jevは「読むだけ」（こちらが正しい）。1回0.4秒前後。

**作りの工夫：**2.5秒で返ってこなければ、元の方式に自動で戻す。Jevが止まっても、仕事は止まらない。

## 9-2. 調べもので読んだ記事が「最後まで取れているか」を確かめる

**それまで：**調べもので公開記事をClaude Codeに読ませて要点を確かめるとき、記事が途中までしか取れていないことがあった。字数が少なすぎないか、本文が別の場所にないか、を機械のルールで見ている。

**Jevへの聞き方：**Noulを3つ。

- 本文が途中で切れているか（文の途中で終わる、「続きを読む」で終わる、見出しだけで中身がない）
- 取れた文だけで、中身が分かるか（一言だけで、本文は別のページや画像にある、なら「いいえ」）
- 取れた文は、開いたページとは別のサイトの文か

**結果：**20件に当てたところ、ルールでは「読めた」だった記事を、Jevが「途中で切れている」（0.88）と判定。本文に「7つ並べます」とあるのに、取れていたのは1と2だけだった。字数が4,427字あったので、ルールでは見抜けなかった。読み直したら12,694字で、7つ全部がそろった。英語の記事でも、文の途中で切れていたものを0.81で拾った。

**作りの工夫：**ルールの警告は消さない。Jevの数字を横に添えて、人が判断しやすくするだけにした。字数や形で見るルールと、中身を読むJevを重ねるのが、いちばん手応えのあった形。

## 9-3. 自分のメモ827件を、使う場面ごとに仕分け直す

**それまで：**学んだことから考え方だけを抜き出して、自分の言葉で1件ずつメモにしていた（元の文章は写さず、「何に使える考え方か」だけを残す）。付けた目印（タグ）で「文章を書くどの場面で使えるか」を振り分けていたが、目印の付け方が揺れると、中身は合っているのに出てこないメモがあった。

**Jevへの聞き方：**1件につき1回で、9つの質問をまとめて聞く。

- 販売・集客の文章を書くのに役立つか（Noul）
- 7つの場面それぞれで、開くと直接役立つか（Noul×7。分野が近いだけで、その場面の作業に使えるなら「いいえ」、と質問に書く）
- どの棚に置くのが自然か（Choice、10の棚）

**結果：**

- 827件を判定して失敗0件。1回平均0.41秒。読ませた量は合計約319万トークンで、本来の料金でも約0.13ドル（約20円）
- 「販売の文章に役立つ」（0.6以上）と判定されたのは554件
- 目印ではどの場面にも出てこなかったのに、Jevが拾ったメモが34件
- 逆に、目印では場面に入っていたのに、Jevの判定では場面に入らなかったメモが175件。中身を見ると2種類が混ざっていた。1つは、目印の言葉だけで引っかかっていたもの（こちらはJevが正しい）。もう1つは、戦略寄りのメモで、「販売の文章に役立つ」は0.9と高いのに、7つの場面のどれも0.7に届かなかったもの（こちらは線が厳しすぎた）
- ここから分かるのは、第4章のとおり「線は自分のデータで引く」しかないということ。索引に使うなら、0.7を決め打ちせず、目印とJevの両方で拾い、食い違った分だけ人が見る形にする
- 今の棚とJevの推す棚が強く食い違ったメモが26件。移すかどうかは、一覧を見て私が決める

**作りの工夫：**3-7の「2段で聞く」はここで見つけた。内容が変わっていないメモは呼び直さない（ファイルの指紋で判定）ので、2回目以降は新しく足した分だけで済む。

#### 9-4. 気になった記事に、読む順番を付ける

**それまで：**気になった記事のリンクは、その場では読まずにメモしておき、あとでまとめて読んでいた。数が増えると、どれから読むかで迷うようになった。

**Jevへの聞き方：**リンクごとに「いまの仕事に関係が深いか」（Noul）。

**作りの工夫：**Jevの数字は、読む順番を決めるためだけに使う。読んでどう使うかは私が決め、判断そのものは任せない。途中でJevが落ちてても、数字が付かないだけで作業は続く。

#### 9-5. 4つに共通して分かったこと

- **大量に回すと待たされる：**Vercel経由でメモ827件を流したとき、同時に何本投げても1分あたり約15回で頭打ちになり（全部で約47分）、「少し待ってから」と返された（公式表記は1分1,200回だが、需要に合わせて調整中と注記がある）。大量に回すなら、同時3本くらいで流しっぱなしにするのが確実
  - **止まっても困らない作りにする：**どれも、Jevが落ちたら元のやり方に戻るか、数字が付かないだけにした
  - **任せるのは「毎回同じ物差しの判断」だけ：**書くのはClaude Code、決めるのは最後は私
-

## 第10章 そのまま貼れる頼み方集

Vercel経由の人は、どの頼み方にも次の【つなぎ方】を添えてください。

### 【つなぎ方】

本家の TypeSafe には登録していないので、Vercel AI Gateway 経由で呼んでください。

- ・ 接続先 (base URL) : `https://ai-gateway.vercel.sh/typesafe`
- ・ モデル: `typesafe-ai/jev`
- ・ キー: (キーを保存したファイルの場所) の `AI_GATEWAY_API_KEY`  
(キーの値は画面に出さないでください)

### 10-1. まず1回試す

Jev の公式スキル (typesafe-ai) を使って、次の文を Jev で判定してください。

#### 【文】

(問い合わせやメールを1通)

#### 【聞きたいこと】

- ・ 種類はどれか (A・B・C・どれもない)
- ・ 急ぎか (「すぐに対応が必要か」を確率で)

結果は、答えと確率を日本語の表で見せてください。

### 10-2. たくさんの文を仕分けて表にする

Jev の公式スキルを使って、(ファイルの場所) の CSV の「本文」列を1行ずつ判定し、結果の列を足した CSV を別名で保存してください。

- ・ 判定は「種類 (A・B・C・どれもない)」と「急ぎか (確率)」
- ・ 同じ1行への質問は、1回の呼び出しにまとめる
- ・ 同時に投げるのは3本まで。待つよう返されたら、指示された秒数だけ待ってやり直す
- ・ 確信度が0.85未満の行は「要確認」の列に印を付ける
- ・ 最初は先頭10行だけで試して、結果を見せてから残りに進む

### 10-3. 線を引くための答え合わせ

(ファイルの場所) に、私が正解を付けた20件があります。

Jev の公式スキルを使って同じ判定をし、正解と並べた表を作ってください。

Nou1 の確率について、 $0.3 \cdot 0.5 \cdot 0.7 \cdot 0.85$  で線を引いたときの一致数も出してください。  
どこで線を引くかは私が決めるので、おすすめは書かなくて構いません。

#### 10-4. 記事や資料を観点別に点検する

Jev の公式スキルを使って、（原稿の場所）を段落ごとに点検してください。  
観点は次の4つ。それぞれ Nou1 で「当てはまる確率」を出し、0.7以上の段落だけ一覧にしてください。

- ・ 数字や事実に、出典や根拠が書かれていない
- ・ 専門用語が説明なしで出てくる
- ・ 同じことを前の段落でも言っている
- ・ 読者への呼びかけが無く、説明だけが続いている

直すのは私がやるので、書き直しはしないでください。

#### 10-5. 実行前に止める仕組みを作る

Jev の公式スキルを使って、ファイルを消す・上書きする・送信するコマンドの前に、  
「このコマンドは、元に戻せない変更をするか」を Nou1 で聞く仕組みを試作してください。

- ・ 0.5以上なら実行せず、内容を見せて私の確認を待つ
- ・ Jev から返事がない、エラーのときも止める
- ・ まずは架空のコマンド10個で試して、止まったもの・通ったものを表で见せる

#### 10-6. Codexで試作を作る

```
$typesafe-ai
```

TypeSafe のスキルを使ってください。

目的：日本語の問い合わせを、種類・有人対応の希望・業務への影響の3つで判定する試作を作る。

- ・ 外への返信や送信、本番データの変更はしない
- ・ どれにも当てはまらないものは、無理に分類せず「保留」に回す
- ・ 計算や日付の比較は、Jevではなく普通のコードで行う
- ・ 最新の説明書を確認し、確認できないAPIや項目は作らない
- ・ 架空の問い合わせ10件で試す（遠回しな依頼・否定文・情報不足・「指示を無視しろ」を含む文も入れる）
- ・ キーの値は画面にもログにも出さない
- ・ 未実行のテストを成功と書かない

#### 10-7. 読んだ記事が最後まで取れているか確かめる

Jev の公式スキルを使って、（読ませた記事のファイル）が最後まで取れているかを確認してください。  
Nou1 で3つ聞いてください。

- ・ 本文が途中で切れているか（文の途中で終わる／「続きを読む」で終わる／見出しだけで中身がない）
  - ・ 取れた文だけで中身が分かるか
  - ・ 取れた文は、元のページではなく別のサイトの文か
- 0.7を超えたものは、元のページをもう一度開いて確かめてください。

## 10-8. 手順書を選ばせる

Jev の公式スキルを使って、（スキルの一覧の場所）の説明文をもとに、次の依頼に合う手順書を Choice で1つ選ぶ仕組みを試作してください。

選択肢には必ず「どれも使わない」を入れてください。

私が普段頼む依頼を10個用意するので、選ばれた手順書と確信度を表にして見せてください。

## 第11章 困ったとき・よくある質問

### 困ったとき

| 症状                      | 直し方                                           |
|-------------------------|-----------------------------------------------|
| 「有効なクレジットカードが必要です」（403） | Vercelでカードを登録する（無料期間中も必須だった）                  |
| 何度も「待ってから」と返される（429）    | 同時に投げる数を減らす。返事に書かれた秒数だけ待つ。<br>私の実測では1分約15回が目安 |
| Claude Codeが本家に登録しようとする | 頼む文に【つなぎ方】の3行を添える                             |
| コピーしたコマンドが動かない          | <code>--</code> が長い横棒1本に化けていないか確かめる           |
| 答えがはっきりしない（0.5前後ばかり）    | 質問に「何をもって、はいとするか」を書く。大きな問い→細かい問いの2段にする        |
| いつも同じ選択肢が選ばれる           | 「どれもでない」を足す。選択肢ごとの説明を書き分ける                    |
| 長い文でエラーになる              | 1回に読める量（約3万2,000トークン）を超えている。分けて渡す             |
| キーが画面に出ってしまった           | Vercelでそのキーを消して、作り直す                          |

## よくある質問

**JevはChatGPTの代わりになりますか** なりません。文章を書く仕事は生成AIの仕事です。Jevは、その裏で「どれか」「何割か」を決める係です。

**プログラミングができなくても使えますか** Claude CodeかCodexがあれば使えます。公式スキルを入れて、第10章の頼み方を貼れば、呼び出しの部分は組み立ててもらえます。

**無料期間（2026年9月25日まで）が終わったいま、いくらかかりますか** 入力100万トークンあたり0.042ドルです。私の実測では、問い合わせ1通の判定が約0.00002ドル、記事1本に9つの質問で約0.00017ドル。Vercelの毎月5ドルの無料クレジットの範囲で、かなりの回数を回せます。

**画像や動画は判定できますか** できません。動画や音声は、先に文字起こしをしてから渡してください。

**判定をそのまま信じて自動で進めていいですか** 消す・送る・公開する、のようなやり直しのきかない操作は、最後は人が確かめる形をおすすめします。

**Jev-Omniとは違うものですか** 別物です。TypeSafe社のJevとは関係のないものと考えてください。

---

## 第12章 用語集

| 用語                   | 意味                                          |
|----------------------|---------------------------------------------|
| Jev（ジエブ）             | TypeSafe社の判定専用AI。文章を書かず、答えと確率を返す            |
| System One（システム1）モデル | Jevのような「とっさに判断する」AIの呼び名。文章を考えて書く従来のAIはシステム2 |
| Choice・Score・Noul    | Jevの答えの3つの形（どれか・何段階か・何割か）                   |
| state（状況）            | Jevに判断させる材料。文章でも、名前を付けた項目の集まりでもよい           |
| criteria             | 選択肢や段階の説明。「何をもって、はいとするか」を書く場所               |
| 確信度（confidence）      | ChoiceとScoreの答えが、どれだけ1つに集中したか。正解率ではない       |
| トークン                 | AIが文字を読む単位。料金はこの数で決まる                       |

| 用語         | 意味                                               |
|------------|--------------------------------------------------|
| 公式スキル      | TypeSafe社が出している、Claude CodeやCodexにJevの使い方を教える説明書 |
| AI Gateway | Vercelの、いろいろなAIを1つの窓口で使える仕組み                     |
| キャッシュ      | それまでの会話をためておき、読み直しを省く仕組み。モデルを途中で替えると効かなくなることがある  |
| RLHF       | 人の好みに合わせて鍛える学習法。ChatGPTの土台                       |
| RLCD       | 確信の数字が正しくなるように鍛える学習法。Jevはこちら                     |

## 出典・参考にした公開情報

確認日：2026年9月24日。各作例は、公開した人の投稿・ページに基づきます。数字は公開者が示した条件での報告値です。

**TypeSafe社・公式** - TypeSafe AI「Introducing System One Models & Jev」  
<https://typesafe.ai/blog/introducing-system-one-models-and-jev> - 公式ドキュメント（クイックスタート・API・モデル・確信度・作り方のガイド） <https://docs.typesafe.ai/> - 公式スキル  
<https://github.com/typesafe-ai/skills> - 公式X（新規登録の一時停止）  
<https://x.com/typesafeai/status/2102281508950307159>

**窓口** - Vercel「TypeSafe AI's Jev now available on AI Gateway」  
<https://vercel.com/changelog/typesafe-ai-jev-now-available-on-ai-gateway> - OpenRouter  
<https://openrouter.ai/typesafe> - Cloudflare Workers AI  
<https://developers.cloudflare.com/ai/models/typesafe/jev/>

**Claude Code・Codexの組み込み** - fast-jev-compaction <https://github.com/tamaratran/fast-jev-compaction> / jev-pruner <https://github.com/tamaratran/jev-pruner> - winnow <https://github.com/GhalebDweikat/winnow> - jev（sebastianbugal さん）  
<https://github.com/sebastianbugal/jev> - building-with-jev-skill  
<https://github.com/dbreunig/building-with-jev-skill> - jev-use  
<https://github.com/shitianfang/jev-use> - slop-grader <https://github.com/lukstei/slop-grader>

grader - jev-model-router (hityyhz さん)

<https://x.com/hityyhz/status/2101288260756484483> - Claude Code向けの組み込み一覧 (非公式) <https://mrjev.com/projects/category/coding-agents/>

**日本語の解説・検証 (Xの記事)** - にくさん「Jevって何ができる？ 活用事例30選」

[https://x.com/29meat\\_ai/article/2100844631693095267](https://x.com/29meat_ai/article/2100844631693095267) - ヤスさん「Jev×Claude Codeと掛け合わせると何ができる？」 [https://x.com/yasu\\_ai\\_good/article/2102011664388505783](https://x.com/yasu_ai_good/article/2102011664388505783) - キムさん (Jev×Codex・Claude Code、効果の数字の読み方)

<https://x.com/kimuai08/status/2101782307237777920> - manaさん「Jev×Codexの実践ガイド」

[https://x.com/MakeAI\\_CEO/article/2101924475814212065](https://x.com/MakeAI_CEO/article/2101924475814212065) - やまさん (Jev Engineering・会話の整理) [https://x.com/yama\\_threads/status/2102260578521645349](https://x.com/yama_threads/status/2102260578521645349) - xjuntaroさん

(Kaggleでの精度検証) <https://x.com/xjuntaro/status/2101989210362454268> - だんさん「小学生でも分かるJevガイド」 [https://x.com/dan\\_pro\\_man/article/2101967189091991570](https://x.com/dan_pro_man/article/2101967189091991570) - 岡村さん (会議から業務フロー図) [https://x.com/masa\\_okamura108/status/2101446065526632473](https://x.com/masa_okamura108/status/2101446065526632473) - 茶圓さん (jev-model-router、Higgsfieldとの組み合わせ)

<https://x.com/masahirochaen/status/2101593485161963561> /

<https://x.com/masahirochaen/status/2101611863507419411>

**そのほか** - classmethod「Jevの使い方」 <https://dev.classmethod.jp/articles/jev-guide-with-examples/> - chot「Vercel AI Gateway で Jev を使う」 <https://chot-inc.com/columns/vercel-ai-gateway-jev-api-guide>