Claude Codeの動きを Langfuseで見てみよう

Langfuse Night #5

大坪 悠

Yu Otsubo

経歴

新潟県出身。2016年KDDI新卒入社。通信設備の運用、社内データ基盤構築のPM、Webチャットボット開発などを経験。スタートアップへの転職を経て2024年にKAG入社。
生成AIを活用したアプリケーション開発に従事。

著書

- **AIエージェント開発運用入門**（SBクリエイティブ）
- **やさしいMCP入門**（秀和システム）

SNS

- @tubone24
- tubone-project24.xyz

趣味: 犬と遊ぶこと

宣伝枠

Portfolio

突然ですが

Langfuse

使ってますか？

（会場でたくさん手が上がっている想定）

でもぶっちゃけ

「AIエージェント
開発運用者向けでしょ？」

その通りです

LLMOpsツールだから当然

- AIエージェントのコスト・レイテンシ監視
- AIエージェントのトレーシング
- プロンプトの評価・改善

普段AIエージェント作ってない人には遠い世界...？

でも

みなさん身近に
使っている
AIエージェント
ありますよね？

Claude Code

使ったこと
ありますよね？

あの中身

AIエージェント

どう動いているか
気になりませんか？

Langfuseで
可視化
やってみたくない？

Claude Codeの動きを Langfuseで 見てみよう

タイトル回収

実はClaude Code OTELに対応してる

Langfuseも
OTEL対応してる

じゃあOTELで
Langfuse連携
すればよくない？

実はLangfuseのOTEL連携だけでは足りない

OTLPの3シグナル型 Traces Metrics Logs

Claude CodeのOTELで送れること

OTEL_METRICS_EXPORTER / OTEL_LOGS_EXPORTER

- ✓ Metrics の送信（トークン数・コストなど）
  - 💼 エンタープライズではチーム全体の利用傾向把握・ガバナンスにも有用
- ✓ Logs の送信（APIリクエスト・ツールイベントなど）
- ⚠ Traces の送信（最近βで追加されたらしい）

LangfuseのOTELで受け取れないこと

/api/public/otel/v1/traces のみ実装

- ⊗ MetricsとLogsの受信（Tracesエンドポイントのみ）
- ⊗ LLM応答テキスト（OTELシグナルに含まれない）
- ⊗ ツール実行の詳細な入出力内容

Langfuseで
丸見えにできます

Claude Code Hooks + シェルスクリプト使用

用意するもの

たった3つだけ

1. Langfuseアカウント — Cloud版でOK
2. Claude Code — いつも使ってるやつ
3. Claude Code Hooks設定 — これがキモ

セットアップは10分で終わります

Langfuseの用語ざっくり解説

3層の階層構造

Session — 複数のTraceをまたぐグループ化単位

Trace — 1リクエスト（問いから答えまでの処理単位）

Observation — Trace内の個々のイベント

出典: Langfuse.com/docs/tracing-data-model

Observationの5タイプ

Generation

LLMコール

モデル名・トークン数・コスト情報

Tool

外部ツール呼び出し

API・ファイル操作・シェル実行など

Span

{ } 作業単位の時間幅

複数処理をまとめるコンテナ

今回の連携では未使用

Event

単一イベント

瞬間的な通知・ログポイント

今回の連携では未使用

Agent

サブエージェント実行

子エージェントをラップする単位

↔ Task tool で起動するサブエージェント

出典: Langfuse.com/docs/observability/features/observation-types

マッピング

Claude Code	Langfuse
Session	Session
ユーザープロンプト起点の1ターン処理	Trace
ツール実行	Tool Observation
サブエージェント	Agent Observation（親）＋ Tool Observation
LLMレスポンス	Generation

Claude Code Hooks

Claude Code Hooksとは

"Hooks allow you to execute arbitrary shell commands at specific points in Claude Code's lifecycle"

Claude Codeのライフサイクルの特定タイミングで任意のシェルコマンドを自動実行できる仕組み。
ツール実行の前後・セッション開始・応答完了など、各イベントにスクリプトを差し込める。

イベント駆動

特定タイミングで自動起動

任意シェル実行

スクリプトで何でも実行可

stdin経由でJSON

イベント情報をJSONで受信

出典: docs.anthropic.com/en/docs/claude-code/hooks

主要3つのライフサイクルイベント

PreToolUse　ツール実行の直前

PostToolUse　ツール実行の直後

Stop　エージェントの応答完了時

その他のイベント（全27種）

SessionStart　SessionEnd　UserPromptSubmit　UserPromptExpansion　Notification　PostToolUseFailure　SubagentStart　SubagentStop

PermissionRequest　PermissionDenied　PreCompact　PostCompact　InstructionsLoaded　ConfigChange　FileChanged　CwdChanged　Elicitation

ElicitationResult　TaskCreated　TaskCompleted　TeammateIdle　WorktreeCreate　WorktreeRemove　StopFailure

出典: docs.anthropic.com/en/docs/claude-code/hooks

Claude Code Hooks設定イメージ

~/.claude/settings.json

```
{
  "hooks": {
    "PreToolUse": [{ "matcher": "",
      "hooks": [{ "type": "command",
        "command": "bash Langfuse-logger.sh" }] }],
    "PostToolUse": [{ ... }],
    "Stop": [{ ... }]
  }
}
```

空のmatcherで全イベントをキャッチ → stdin経由でJSONコンテキストを受信

ただし…

HooksからはLLM出力が見えない

（ライフサイクルイベントのフックなんだから、当たり前といえば当たり前なんですが...）

問題

Hooksのイベントデータにはツールの I/Oが含まれるが、**LLMが生成したテキスト自体は含まれない**

→ GenerationなしではLangfuseのトレースでトークン数・コスト・応答内容がすべて空欄に

だから

Stopイベント時にJSONLトランスクリプトを解析して

アシスタントメッセージを抽出し、Generationとして記録する

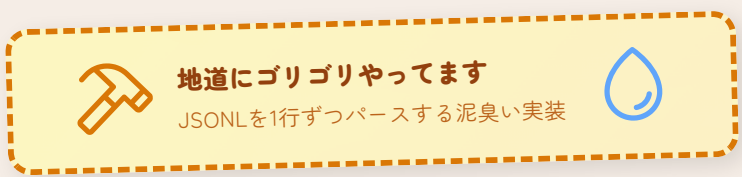
JSONLトランスクリプトの中身

```
~/.claude/projects/<project>/<session_id>.jsonl（各行が1イベント）

// ユーザープロンプト
{"type":"user","message":{"role":"user","content":"docsを調べて実装して"},"timestamp":"..."}

// Extended Thinking（stop_reason: null → 次のGenerationにマージ）
{"type":"assistant","message":{"content":[{"type":"thinking","thinking":"","signature":"EtEGCl..."}],
  "stop_reason":null,"usage":{"input_tokens":5201,"output_tokens":42}}}

// ツール呼び出し判断（stop_reason: "tool_use" → Generation記録）
{"type":"assistant","message":{"content":[{"type":"tool_use","name":"Read","input":{"file_path":"src/index.ts"}}],
  "stop_reason":"tool_use","usage":{"input_tokens":5243,"output_tokens":87}}}

// 最終回答（stop_reason: "end_turn" → Generation記録）
{"type":"assistant","message":{"content":[{"type":"text","text":"実装しました！"}],
  "stop_reason":"end_turn","usage":{"input_tokens":8102,"output_tokens":203}}}
```

stop_reason で記録内容が変わる

stop_reason	意味	Langfuseに記録するもの
null	Extended Thinkingブロック	次のGenerationのinputにマージ
"tool_use"	ツール呼び出し判断	ツール実行指示 Generation
"end_turn"	最終応答完了	ユーザーへの最終回答 Generation

transcriptのJSONLを解析して stop_reason ごとに Generation を生成



フック間の状態共有

Claude Code Hooksは別プロセスで動く → ファイルで状態共有

```
/tmp/claude-Langfuse/{session_id}/

├── current-trace-id  # TraceID
├── turn-count  # ターン番号
├── model  # 使用モデル名
├── transcript-offset  # 読み込み位置
└── subagent-{id}  # サブエージェント
```

UserPromptSubmit

ターン番号++ → 新TraceIDを生成・保存

PreToolUse / PostToolUse

TraceIDを読み込んでTool Observationを送信

Stop

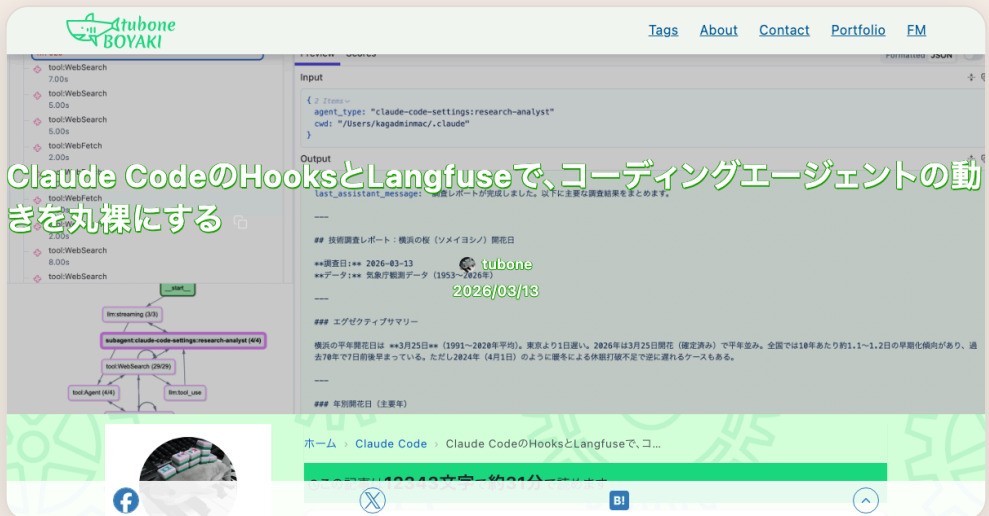
transcriptを解析してGenerationを一括送信

地道にゴリゴリやってます

tmpファイルで状態を受け渡す泥臭い実装

詳しくは
ブログで

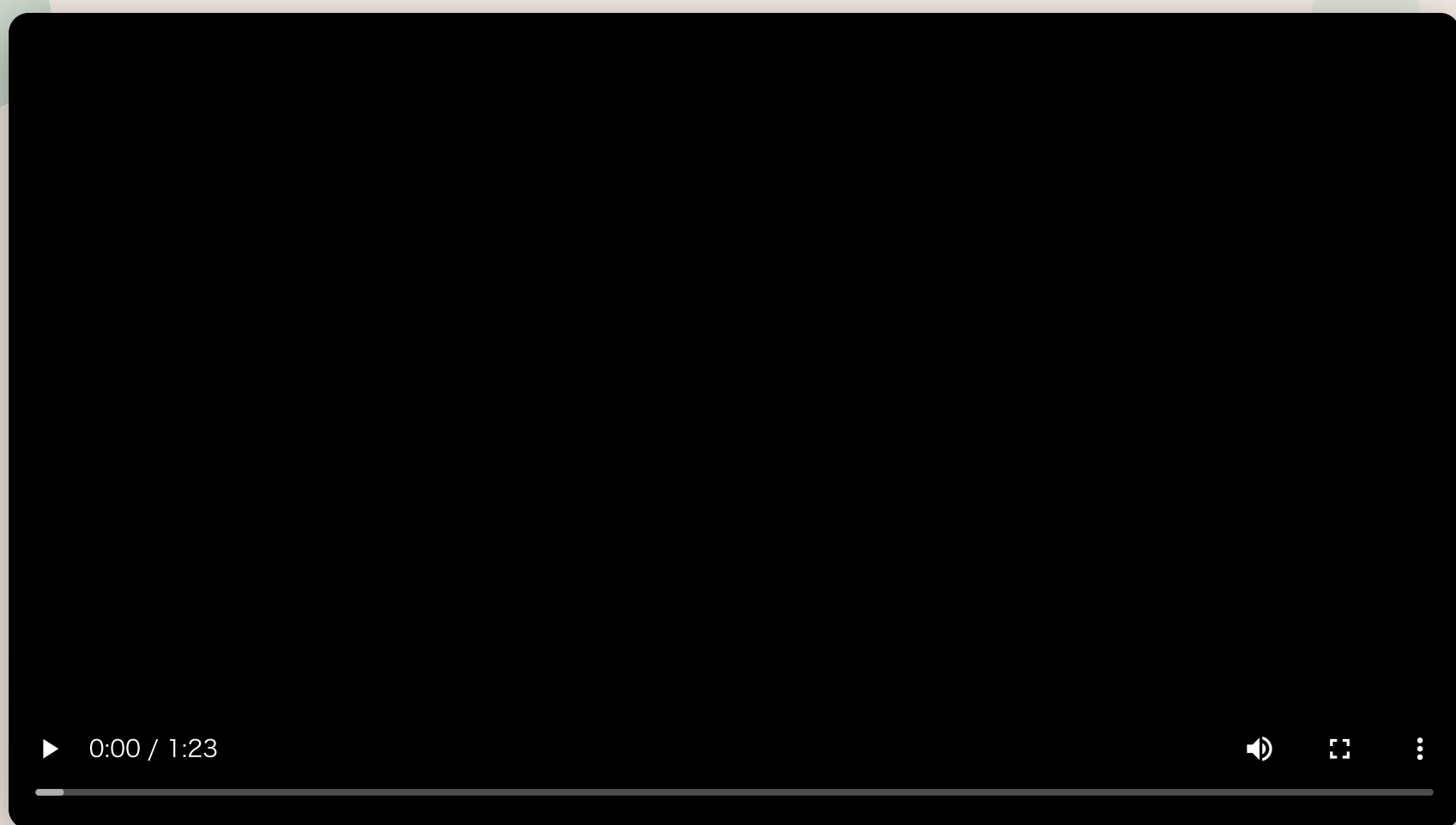
ブログ書きました



tubone-project24.xyz

セットアップ手順・スクリプト全文
トラブルシューティングまで

実際に動かしてみた


0:00 / 1:23

まとめ

Claude Codeの中身のぞいてみたらめちゃ面白かった

Langfuseで見えてきたこと

サブエージェントの戦略が洗練されている

目的に応じてサブエージェントを使い分け、役割と責任範囲が明確。親子の階層構造も整理されている

thinking → tool_use → end_turn の意思決定が可視化できる

「何を考えて」「何を呼び出して」「どう答えたか」がGenerationの連鎖として追える

これ、そのまま自前のAIエージェント設計の教科書になる

サブエージェント分割の粒度・並列化・フォールバック設計など、参考にできるパターンが詰まっている

自前の
AIエージェントにも
活かそう

おまけ

なんと

LLM as a Judge

もできます！！


Simple Criteria (ev...
Action
1.0000
contain the identical prompt: "@slides.md ありがとうございました。のあとにおまけと入れてほしい". There is a perfect 1:1 match between the defined criteria and the provided input.
🤫 続きは懇親会で

この話は
懇親会で。

ありがとうございました