DRUPALCAMP TOKYO 2026

# 40個のDrupal Viewsを Claude Codeでテーマして苦戦した話

一見、**Claude**くんは仕事ができる

Speaker：Kyotaro Kon・ANNAI株式会社

< ANNAI >

## 自己紹介

# Kyotaro Kon

**ANNAI株式会社**

Drupal と歩んで 20年　@タイ

X（旧Twitter）：/KonKyotaro15230

今日のひとこと

**プロンプトと罰**

## 今日のあらすじ

01 40個のViewsをClaude Codeに任せた

02 一見、うまくいったように見えた

03 見た目ほど、単純な話ではなかった

04 効いた工夫と、これからやりたいこと

## ことの始まり

40 Viewsを Theming

**ある案件**　Layout Builder で組む、自由度の高い Drupal サイト。**デザインHTMLはすでに上がっている。**

**やること**　Views の出力を、デザインの HTML に合わせる　（ **= Twig テンプレートを書く**）

**40**

対応すべき Views の件数

**想像したくない**

手作業でやった場合の時間

# Claude Code、君に任せた

こうなるはずだった（期待）

STEP 01

**デザインHTML を渡す**

ぽーん

▶

STEP 02

**Claude Code が生成**

がりがり

▶

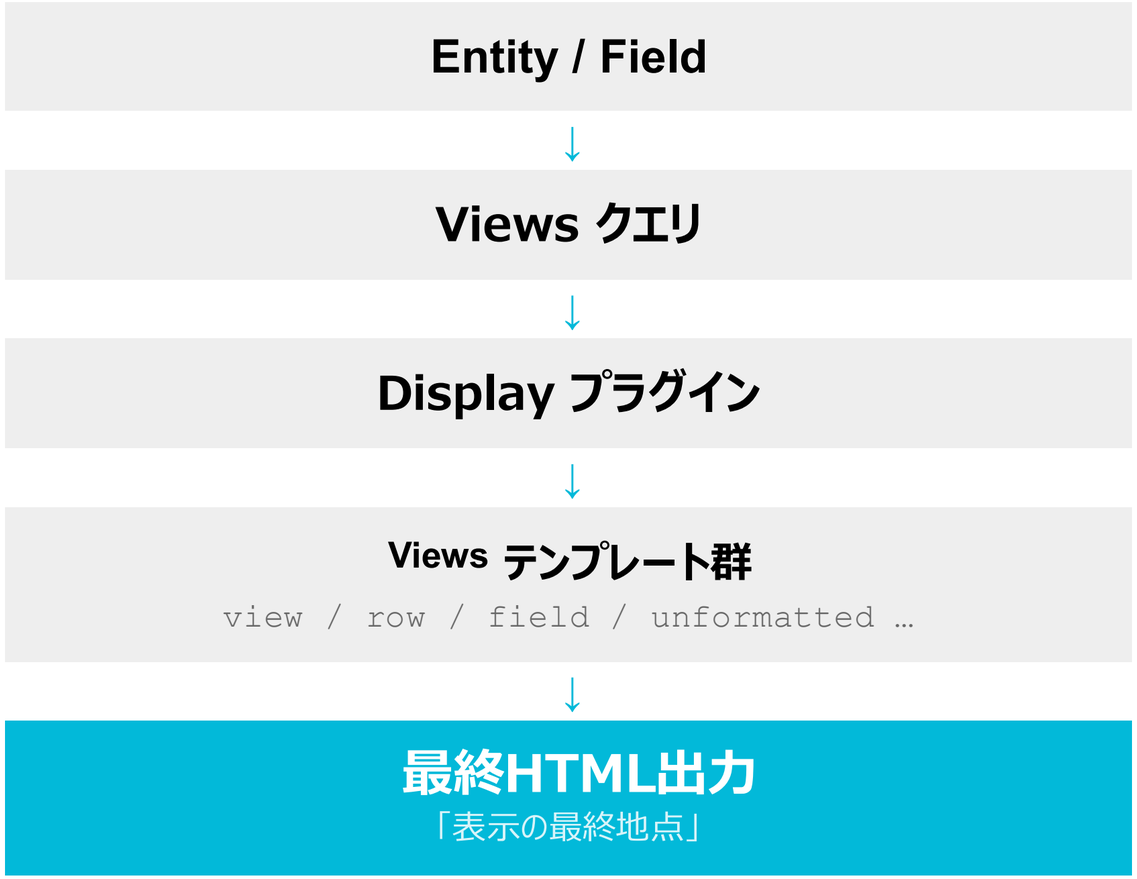
STEP 03

**40個の Twig テンプレート完成**

やったね

…と、思っていた時期がありました。

# そもそも、Views とは




- 複数層のテンプレートが合成されて出力になる
- Drupal の規約に依存する箇所が多い
- プレビュー = データを含む状態でしか確認できない

**ざっくり一括投入では、難しい。この複雑さが、後で効いてくる。**

FAILURE 01

# 出所が、わからない

## ◎ 一見

デザイン通りの見た目
テキストもちゃんと出ている

**「お、できてるじゃん」**

## ✕ 実は

例：時間フィールドの横に注釈表示

どのフィールド由来か特定不能

→ 静的テキストとして埋まる

→ Drupal で更新しても反映されない

**デザインからは、フィールドの「出所」までは読めない。**

FAILURE 02

# バリエーションが、見えない

## ◎ 一見

デザインモック通りに見える
テキストも表示できている

**「動いてますね」**

## ✖ 実は

例：複数カテゴリの繰り返し処理 ／ プレーンテキスト内の改行

1パターンで固定化される

→ データの多様さを感知できない

**デザインモックは、データの「1サンプル」でしかない。**

FAILURE 03

# HTML が、違う

## ◎ 一見

見た目はピクセル単位で同じ
クラス名も合ってる（ように見える）

**「完璧では？」**

## ✖ 実は

DOM 構造が地味に違う

div の階層、ラッパ要素

→ 後工程の CSS / JS に影響が及ぶ

**見えない違いほど、後で効いてくる。**

つまり

# 手放しでは、いけない。

# 「一見正しい」は、「検証できている」ではない。

# でも、これは効いた

BEFORE **いきなり投げる**

「このデザインで Twig 書いて」

全体をざっくり渡す

Claude くん「はい喜んで」

→ ①②③ の落とし穴を踏みやすい

AFTER **先に「設計」させる**

まずデザイン HTML を分析

リージョン単位で対応箇所を確認

「ここがフィールド X だね」を合意

→ 出力の質が劇的に向上

**コードを書かせる前に、分析させ、設計を書かせる。**

# 改めて、改善した（2026 / 5）

## やったこと

エージェントをスキルに変更し、プロセス化

デザイン側 / 生成側の HTML を、リージョン単位で AST 化

タグ階層・必須属性のみを比較（整形やクラス順は無視）

差分が出たら View ごとにフラグ立て → 人間が見にいく

**× テキスト Diff**

空白・属性順・整形でノイズ → **差分が読めない**

**○ リージョン単位の DOM 走査**

構造として「同じ」かを判定 → **ノイズに惑わされない**

# AI にも、聞いてみた

「お前ならどう改善する？」— Claude の答え

01 → 失敗 ①

## Twig 変数を AI に渡す

Devel の kint() 等で、レンダリング時に使える変数の構造を先に共有する。

02 → 失敗 ②

## テストデータを 多様化する

複数カテゴリ、改行、空値、長文 … バリエーションを揃えて検証する。

03 → 失敗 ③

## Drupal 標準を お手本に

Drupal core / Olivero の既定出力を「正しい構造」のリファレンスとして渡す。

**AI 自身の自己診断 — ぼちぼち**

# スキル（コマンド）の概要

使い方: /theming https://example.com/about

スキルの中身は「**Claudeへの作業指示書**」。以下のような指示で構成されています。

- 環境自動検出 — drush実行方法（Makefile/ddev/lando/docker compose）、サイトURL、テーマディレクトリを自動判定
- HTML取得 — agent-browserで参照元とDrupal側の両方のHTMLを取得
- 差分分析・報告 — ページの部品単位で突き合わせ、一致/ラッパー差分/欠落/過剰/相違に分類してテーブルで報告
- 対応範囲の確認 — ラッパー差分は自動で進め、欠落や相違はユーザーに判断を仰ぐ。確認後は最後まで自律進行
- Drupal固有ノウハウ — Viewsラッパー削除、空コンテンツ判定、LB編集画面との両立など、定型テクニックをリファレンスとして内蔵

## まとめと、今後

# 「一見正しい」を、信用しない。

| | | |
|---|---|---|
| 01 | AI に「設計」を先にさせる。 | コード生成より前に、対応関係を合意する。 |
| 02 | AI は、手を抜く。 | 大事なところが要約でうしなわれてしまう。 |
| 03 | AI は、手を抜く。。 | 指摘するといつもの「正直にいいますと、、、」のフレーズが開始する。 |
| 04 | AI に「設計」を先にさせる。 | 設計との連携、HTMLへデータ属性の追加、バリエーションの対応。 |

# ありがとうございました

Kyotaro Kon・ANNAI株式会社

< ANNAI >