

Π Ε Ι Θ Ω

Peitho



なぜつくったか

スライドをMarkdownで書くのはずっとやってきた。[Deckset](#)はデザインの自由度がなく、[k1LoW/deck](#)は思想は最高だけど、デザイン調整はGoogle Slidesでの手作業になる。

デザイン成果物がHTML/CSSというただのコードなら、生成AIに書かせられるし、gitでdiffもレビューもできる。

設計の三本柱

- **コンテンツとデザインの分離** — コンテンツはMarkdown、デザインはレイアウトHTMLとCSS。
両者を混ぜない
- **バージョン管理できるデザイン** — 成果物はただのHTML/CSS。diffが取れ、レビューできる
- **型検査されるスロット契約** — 内容がレイアウトの宣言に合わないときは、黙って捨てずにビルドエラー

デッキはただのMarkdown

`---` で区切ると1枚のスライド。最も浅い見出しがタイトル、コードブロックはcodeスロット、残りはbodyスロットへ自動で割り当てられる。

1枚目のスライド

本文の段落。

- リストも
- 使える

2枚目のスライド

レイアウト自身がスキーマ

デザインの単位はレイアウトで、ただのHTMLファイル。スロットの名前・受け入れる型・個数を宣言し、ビルド時にPeithoがここから契約を取り出す。

```
<section class="peitho-slide">
  <h1><slot name="title" accepts="inline" arity="1"></slot></h1>
  <div class="body">
    <slot name="body" accepts="blocks" arity="0..*"></slot>
  </div>
</section>
```


合わなければビルドが止まる

スロットの過不足も、型の不整合も、行番号とヒント付きのビルドエラーになる。「黙って何かが起きる」経路を残さない。

```
error: slide 2 ('code-slide'), line 7: slot 'code' got 2 item(s),  
      but layout 'title-body-code' allows 0..1  
= help: use a layout with more code capacity or remove one code block
```

キー付きper-slide調整

スライドに安定キーを振ると `data-slide-key` 属性が付いて、CSSからそのスライドだけを狙える。
デッキに存在しないキーを指定したCSSはビルドエラーになるので、参照切れが静かに残らない。

```
[data-slide-key="payoff"] .slot-code {  
  background: #0d2822;  
}
```

型駆動ディスパッチ

レイアウトは複数持てて、スライドの内容の形にスロット契約が一致するレイアウトへ自動で振り分けられる。

- ページ設定コメントでの明示指定が最優先
- 自動で決まるのは、一致がちょうど1枚のときだけ
- 複数一致も0枚一致もビルドエラー。黙って解決しない

デモサイト `peitho.gosu.ke`

リポジトリの `examples/` を `peitho build` してそのまま配信したもの。同じツール、同じMarkdown規約で、まったく違うデッキになる。

Peitho

Markdownをsource of truthにする、HTMLネイティブな
プレゼンテーションツール。以下はサンプルデッキを
そのまま`peitho build`して配信したデモです。

Keynote セリフ体・中央寄せのキーノート。タイトルだけのスライドはcoverレイアウトへ、本文のあるスライドはstatementレイアウトへ、内容の形から自動でディスパッチされる。

Code Walkthrough ターミナル風2カラムのコード解説。毎スライドにコード必須の契約 (`arity="1"`)、`syntect`によるビルド時シンタックスハイライト、キー付きCSS `override`の実例。

Lightning Talk ダーク+大型タイポのLT向けデザイン。レイアウトにcodeスロットが無いので、コードを書くとビルドエラーになる。

Minimal バイナリ内蔵のデフォルトレイアウトとテーマだけで、`deck.md`単体からビルドした最小構成。

github.com/mizzy/peitho

Keynote — クリーム地とセリフ体

cover(タイトルのみ)とstatement(タイトル+本文)の2レイアウト構成。内容の形から型駆動ディスプレイで振り分けられる。

Peitho

Code Walkthrough — コード必須のレイアウト

codeスロットが `arity="1"` なので、毎スライドにコードが必須。シンタックスハイライトはsyntectでビルド時にかけていて、実行時JSはない。

// Phases live in the type

```
pub struct Deck<Phase> {  
    slides: Vec<Slide>,  
    _phase: PhantomData<Phase>,  
}
```

```
pub enum Parsed {}  
pub enum Checked {}  
pub enum Rendered {}
```

- > A deck carries its pipeline phase as a type parameter.
- > The phase markers are empty enums: they exist only at compile time.

Lightning Talk — コードを書けないデッキ

レイアウトにcodeスロットがないので、コードブロックを書くとビルドエラーになる。レイアウトが「このスライドに何を書けるか」を決めて、ビルドが守らせる。

LIGHTNING TALK

スライドはMarkdownで書きたい

5分でわかる、発表資料をコードとして扱う話。

peitho presentで発表する

ローカルサーバが立ち上がり、外部ディスプレイにスライドをフルスクリーン、手元に発表者ツールを自動配置。Escで全部まとめて終了する。

SLIDE 02 OF 05 · SECTION — "PROBLEM"

スライドはMarkdownで書きたい

LIGHTNING TALK

デザインをいじりだすと準備が終わらない

- 内容よりフォントサイズを触っている時間のほうが長い
- 「あとで整える」は発表当日まで残り続ける
- 前回のスライドから使い回せるのは見た目だけ

02 / 05

← → navigate Space start / pause S swap Esc close

NOTES

Slide 02

No notes for this slide.

NEXT

03 / 05

LIGHTNING TALK

コンテンツとデザインを分ける

- 内容はMarkdown、見た目はテンプレートとCSS
- 書いている間、デザインのことは考えなくていい
- デザインは一度作れば次の発表でも使える

01:20 / 05:00

RUNNING

SLIDE PROGRESS

TIME

0:001:152:303:455:00

AGENDA

ACTUAL / PLANNED

● Setup 01

0:57 / 1:00 -0:03

● Problem 02

0:24 / 1:00

○ Approach 03-04

- / 2:00

○ Wrap-up 05

- / 1:00

PAUSE Space

PREV ←

NEXT →

RESET

SWAP S

CLOSE Esc

残り時間で色が変わるタイマー

frontmatterの `time: 15m` が計画時間になる。経過・計画・超過が並び、残り3分でオレンジ、残り1分で赤、超過すると `+MM:SS` が追加で出る。

01:20 / 05:00

● RUNNING

02:31 / 05:00

● RUNNING

04:12 / 05:00

● RUNNING

05:37 / 05:00 +00:38

● RUNNING

かめ🐢とうさぎ🐰のトラッカー

かめが経過時間、うさぎがスライドの進捗。位置関係だけで「今のペースで間に合うか」が読み取れる。数字の暗算は発表中にやりたくない。



超過したセクションが赤くなるアジェンダ

セクションごとに計画・実績・差分が並び、発表中のセクションも計画を超えた瞬間に赤くなる。セクション合計とデッキ全体の `time` がズレていたらビルドエラー。

AGENDA	ACTUAL / PLANNED		
● Setup 01	0:57	/ 1:00	-0:03
● Problem 02	0:55	/ 1:00	-0:05
● Approach 03-04	2:20	/ 2:00	.
○ Wrap-up 05	-	/ 1:00	.

検査を通らないとレンダリングできない

ビルドコアはRust。デッキの型がパイプラインの段階ごとに分かれていて、レンダラは検査済みの型しか受け取れない。検査をすっ飛ばすコードはコンパイルが通らない。

```
// Parsed → Mapped → Checked → Rendered  
fn render_deck(deck: Deck<Checked>) -> Result<Deck<Rendered>>
```

Π Ε Ι Θ Ω

言葉で、人の心を動かす

—