

レイアウトがスキーマになる

型検査つきプレゼンツールPeitho

Hackers Champloo 2026 2026-10-12



宮下 剛輔

合同会社Serverspec Operations代表社員

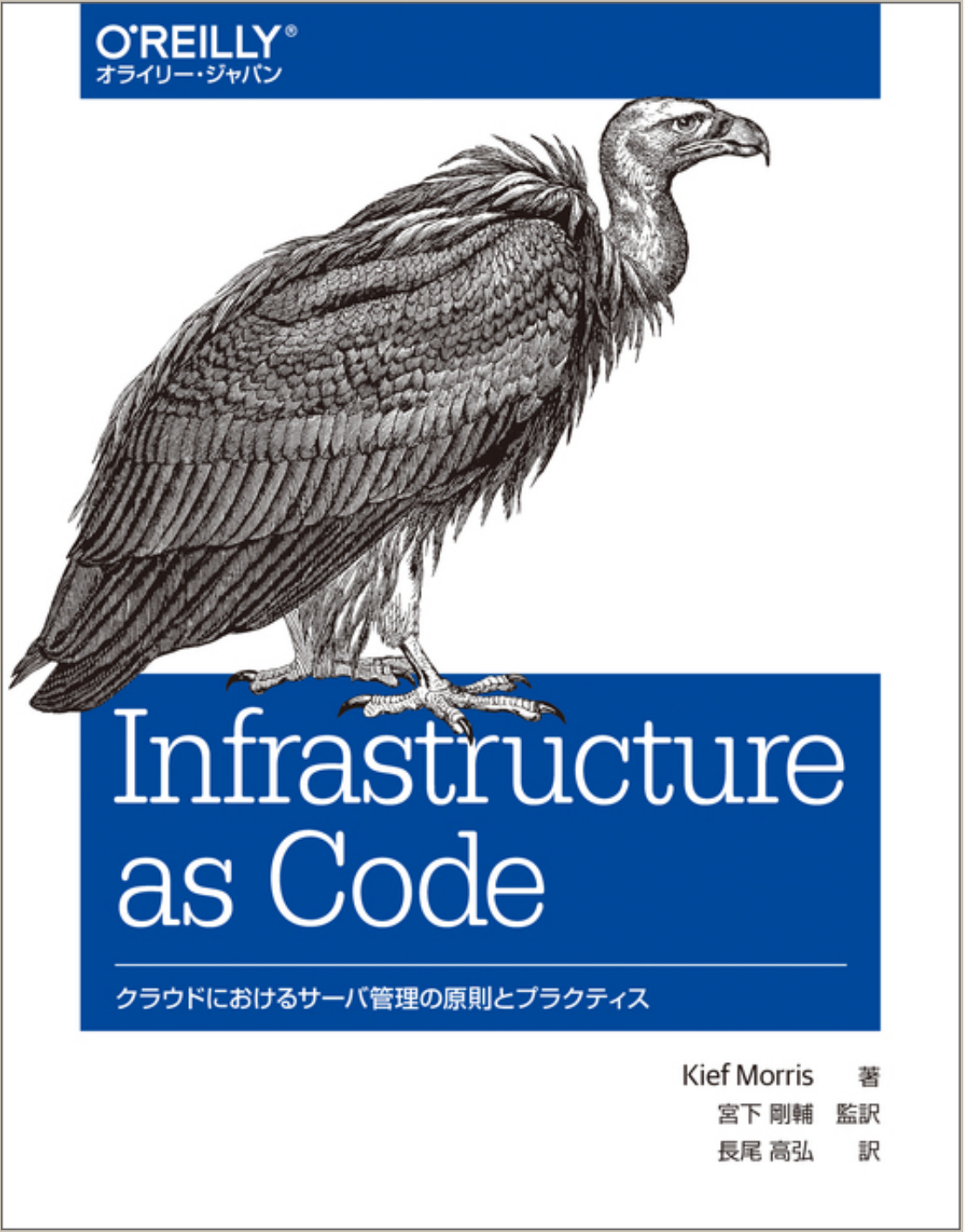
フリーランスのソフトウェアエンジニア

 mizzy.org /  mizzy /  gosukenator

北海道出身・神奈川県在住

沖縄は約4年ぶりです

Infrastructure as Code関連の主な著作



情報処理学会論文誌 Vol.61 No.3 677-686 (Mar. 2020)

Serverspec：宣言的記述でサーバの設定状態をテスト可能な汎用性の高いテストフレームワーク

宮下 剛輔^{1,2,a)} 栗林 健太郎³ 松本 亮介¹

受付日 2019年6月18日, 採録日 2019年11月29日

概要：システムの大規模・複雑化にともない、サーバの構築・運用を効率化するために、サーバの設定状態をコードで記述する手法が数多く提供されている。それらの手法を効率良く扱うプロセスとして、テスト駆動開発の手法をサーバ構築に応用した Provisioning Testing という手法が提案されている。この手法を支援するテストフレームワークもいくつか登場しているが、あるものは特定の構成管理ツールに依存、またあるものは OS ごとの違いを自ら吸収しなければならないなど、汎用性に難がある。シェルコマンドを直接記述する必要があり、可読性に難があるものも存在する。そこで本論文では、生産性や保守性を向上するために、サーバの設定状態を汎用的かつ可読性の高い宣言的なコードでテスト可能なテストフレームワークを提案する。提案手法では、汎用性を高めるために、OS や構成管理ツール固有の振舞いを整理して一般化し、運用業務で発生するコマンド群を体系化・抽象化した汎用コマンド実行フレームワークを定義する。続いて、テストコード記述の抽象度を高め可読性を上げるために宣言的な記法で汎用コマンド実行フレームワークを操作できる制約テストフレームワークを定義する。これにより、管理者が OS や構成管理ツールの違いを気にすることなく、可読性の高いコードでサーバの設定状態を容易にテストできるようになり、サーバの運用・管理コストを低減できる。提案するテストフレームワークを Serverspec と名付けた。提案手法の評価は、生産性、保守性に加え、拡張性、実装公開後の影響の 4 つの観点で行った。

キーワード：test-driven infrastructure, provisioning testing, serverspec, specinfra

Serverspec: A Versatile Test Framework for Testing States of Servers by Declarative Description

GOSUKE MIYASHITA^{1,2,a)} KENTARO KURIBAYASHI³ RYOSUKE MATSUMOTO¹

Received: June 18, 2019, Accepted: November 29, 2019

Abstract: As the increase of large and complex systems, many ways to describe states of servers as code are supplied. As an effective process to handle these ways, Provisioning Testing is proposed. It is the application of Test-Driven Development to the server configuration. Several test frameworks that support this process are appearing, but they have difficulty in versatility because some frameworks depend on specific configuration management tools, and others need the consideration of differences between OSes to write test code. Also, others need to write shell commands directly, which is difficult to read. In this paper, we propose a test framework that doesn't depend on specific configuration management tools or OSes for testing states of servers by versatile and readable code for productivity and maintainability. In our proposal, we arrange and generalize the behavior of OSes and configuration management tools and define the versatile command execution framework to increase versatility. Next, we define the test framework that controls the command execution framework by declarative and natural language to increase the level of abstraction and readability. By this test framework, system administrators can test states of servers easily without considering the differences between OSes or configuration management tools and can decrease the costs of server operations. Also, by defining the frameworks separated into purposes, we can replace the control test framework with another one that has original notation easily. We named the proposed test framework Serverspec. We evaluated this proposed method on four perspectives: productivity, maintainability, extendability, and impact after the publication of the implementation.

Keywords: test-driven infrastructure, provisioning testing, serverspec, specinfra

© 2020 Information Processing Society of Japan

677

2015年にハッカソンだけ参加してました



いわむーちょ

@takusamar · [Follow](#)



会社の後輩がgosukenator氏とnaoya_ito氏がどういう人か全く知らずに送迎をしていた件。
まあ知らないおかげでビビらず自然体で接することができて、むしろ良かったんじゃないかな。
でも立ち食いのキンタコ連れてったのは、さすがにどうかと思う。

4:49 AM · Jul 2, 2015



8



Reply



Copy link

[Read more on X](#)

naoyaさんと相部屋



naoya

@naoya_ito · [Follow](#)



mizzyさんと相部屋でこの距離感



6:54 AM · Jun 24, 2015



62



Reply



Copy link

[Read more on X](#)

Peitho と は

Markdownで書いてブラウザで発表する

- コンテンツはMarkdown、レイアウトはHTML、見た目はCSS
- ビルドすると静的なファイル式になる
- 他にあまりない特徴は2種類の検査
 - 静的検査（型検査）：レイアウトに書いたスロット定義がそのままスキーマになる
 - 動的検査（lint）：実際に描画して、はみ出しや小さすぎる文字を報せる

開発の動機

プレゼンツール利用遍歴

- 1997年 ~ 2014年
 - PowerPoint / Keynote
- 2014年 ~ 2025年
 - Deckset¹
- 2025年
 - k1LoW/deck²

1. <https://www.deckset.com/>

2. <https://github.com/k1LoW/deck>

Deckset

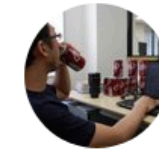
- Markdownで内容を書き、デザインはプリセットのテーマから選択
- 色やフォントはカスタマイズできるが、レイアウトは変えられない
- オリジナルテーマの作成はできない
- デザインのことを考えずに中身に集中できるのは利点
- レイアウト調整ができない、特にコードを見やすくするための調整ができないのが難点

k1LoW/deck

- Markdownで書いた内容をGoogle Slidesに流し込む
- デザインはGoogle Slidesのレイアウトとして自分でつくれる
 - HTML/CSSを書かなくてもGoogle SlidesのUIでデザインを調整できるのは自分にはうれしい
- Songmu/laminate¹と組み合わせることでコードも見やすくできる
 - シンタックスハイライト＋画像化が簡単にできる

1. <https://github.com/Songmu/laminate>

最高すぎてもう戻れない

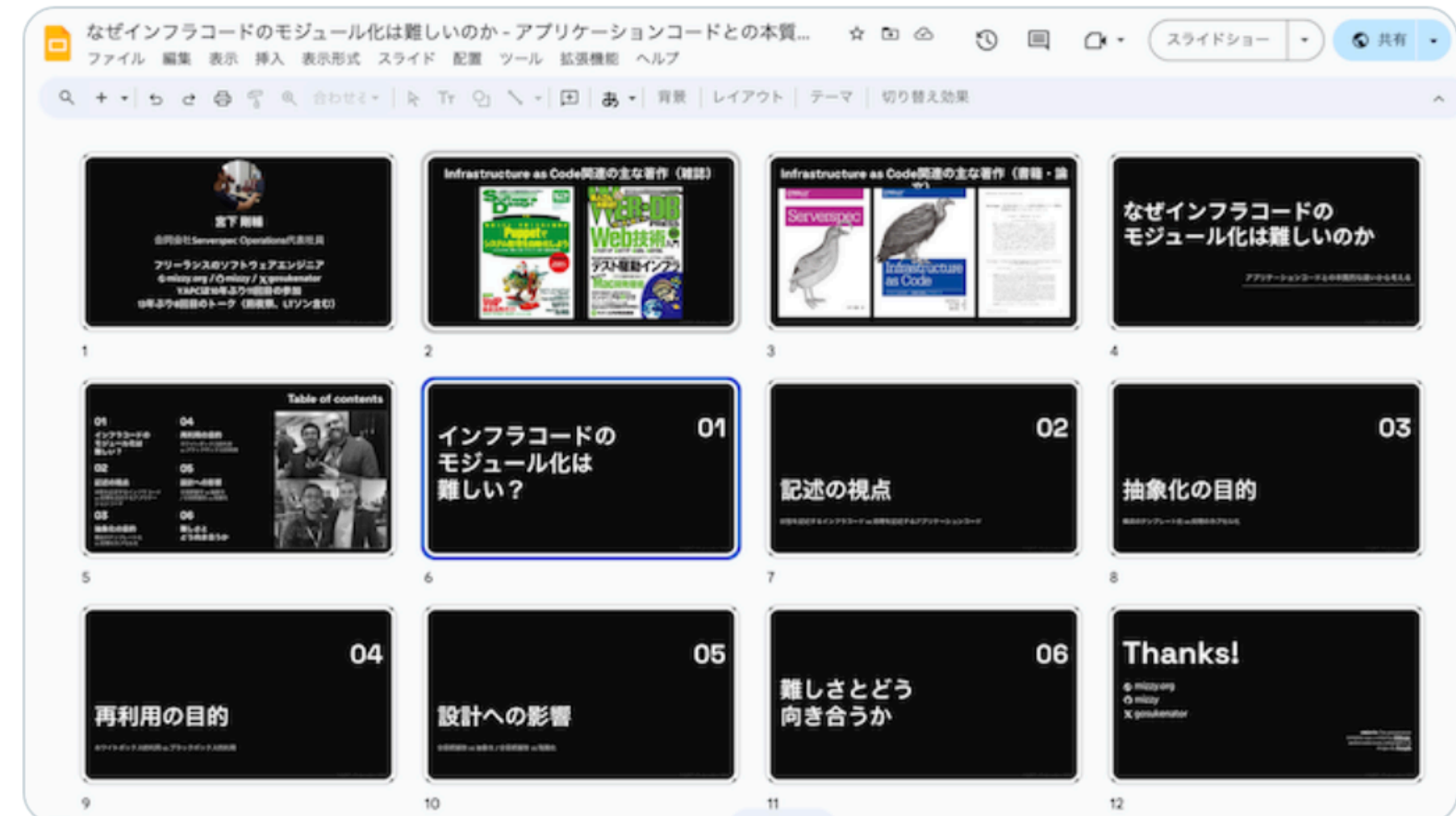


mizzy

@gosukenator · [Follow](#)



YAPC用スライドの進捗。全体の構成はこんな感じかな。あとは中身を埋めていく。Markdownでスライドつくるツールは長いことDeckset使っていたけど、k1LoW/deckが最高すぎてもう戻れない。



8:24 AM · Sep 21, 2025



11



Reply



Copy link

[Read 1 reply](#)

と思っていたが...

- AIの進歩がすさまじい
- 自分が苦手なデザインをAIに任せたいと思うようになった
- Google SlidesはAIにデザインを任せるには不便
- AIにデザインを任せやすいプレゼンツールが欲しくなった
- ただしコンテンツの主導権は自分が握りたい
- だから自分でつくることにした

設計の過程

デザインの記述はHTML/CSS

- 「Claude Designにスライドをデザインしてもらうなら、フォーマットは何かベスト？」
- 答えはHTML/CSS。Claude Designが生成するのがHTML/CSSだから
- 当初Speaker Deckへの入稿（PDF）も考慮
- Speaker Deckにはこだわらないことにした

コンテンツの記述はMarkdown

- コンテンツとデザインの分離
 - 書いている間、内容に集中できる
- バージョン管理したい
- 既存ツールはどれもMarkdown
 - Deckset・k1LoW/deck・Marp・Slidev...etc

成果物をどう生成するか

- 最初に考えたのは「Markdownを書く → Claude Designが完成したスライドのHTMLにする → Peithoで発表する」
- だがAIが完成品を吐くならPeithoの役割は発表だけになる
 - それならreveal.jsで十分ではないか、という話になる
- そもそもAIの生成は同じ入力から同じ出力になるとは限らない
- Markdown修正 → 生成 → 確認のループを回す時間も気になる
- そこでHTMLはPeithoが決まったルールで生成し、AIを通さない

役割の決定

- Markdown
 - 内容と構造を記述する
- Peitho
 - MarkdownをレイアウトHTMLに流し込み、発表まで担う
- Claude Design
 - スライドの生成には挟まず、テーマとレイアウト体系をつくる
 - それをClaude CodeがレイアウトHTMLとCSSに書き起こす

MarpやSlidevとの差別化

- HTMLネイティブでMarkdownから組み立てるツールは既にある
- ではPeithoの差別化をどこに置くか
- レイアウトに型を持たせて内容を検査する

型を持ち込むのはClaudeの提案

Carina¹で独自DSLと型システムをやってるmizzyさんなら、
スライド記法そのものに型や構造の保証を持ち込む（例: レイアウトの
制約を型で縛る、参照切れをコンパイル時に検出する）あたりが、
既存ツールにないPeithoらしさになりうる

1. <https://github.com/carina-rs/carina>

k1LoW/deckのプレースホルダを型の土台にした

- k1LoW/deckが使う Google Slidesのレイアウトは、名前付きプレースホルダ（TITLE / SUBTITLE / BODY）を持つテンプレート
- Markdownは決まったルールでプレースホルダに流し込まれる
- Peithoはこのプレースホルダに型を持たせることにした

Google Slidesのレイアウトとプレースホルダの例

クリックしてテーマタイトル
の書式設定を編集

- クリックしてテーマ サブタイトル
の書式設定を編集



k1LoW/deckのMarkdown記法

前ページのレイアウトに流し込むMarkdown

```
<!-- {"layout": "title-subtitle-image"} -->
```

```
# スライドのタイトル
```

```
## サブタイトル
```

```

```

黙って捨てずにビルドエラーにする

プレースホルダが足りない場合、残りのコンテンツは描画されない

— k1LoW/deckのREADME¹

- k1LoW/deckではプレースホルダに対する検査がない
- Peithoではプレースホルダをスロットと呼び、受け取る型と個数を宣言する
- 合わなければ流し込む前にビルドエラーにする

1. <https://github.com/k1LoW/deck#how-markdown-maps-to-slide-placeholders>

静的検査（型検査）

レイアウトがスキーマになる

スロットが受け取る型と個数をレイアウトHTMLの中で宣言する。これが契約になる

```
<section>
  <h1><slot name="title"
    accepts="inline" arity="1"></slot></h1>
  <slot name="code"
    accepts="code" arity="0..1"></slot>
</section>
```

満たす例

タイトルだけ

```
# タイトル
```

コードが1つ

```
# タイトル
```

```
```rust  
let x = 1;
```
```

満たさない例

本文がある

```
# タイトル
```

```
本文。
```

コードが2つ

```
# タイトル
```

```
```rust  
let x = 1;
```
```

```
```rust  
let y = 2;
```
```

受け取るスロットがないとビルドエラーになる

本文を書くようになる

```
error: slide 1 ('slide-1'), line 7:  
  no slot in layout 'simple' accepts this paragraph  
help: add a 'body' slot to the layout or remove the paragraph
```

個数が合わなければビルドエラーになる

コードブロックを2つ書くようになる

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

契約に合うレイアウトが自動で選ばれる

- レイアウト名の指定は省略できる
- Peithoがレイアウトを決める手順
 - 明示指定 `{"layout": "..."}` があればそれを使う。未知の名前はビルドエラー
 - 明示指定がなければスライドの内容を各レイアウトの契約に当てる
 - 適合がちょうど1つならそれに決まる
 - 複数でも0個でもビルドエラー。黙ってどれかを選ぶことはしない

合うレイアウトがなければビルドエラーになる

合うレイアウトが存在しないようになる

```
error: slide 1 ('slide-1'), line 5:
```

```
no layout matches this slide
```

```
shot: no slot in layout 'shot' accepts this code block
```

```
simple: no slot in layout 'simple' accepts this image
```

```
help: adjust the slide content,  
or add a layout that accepts it
```

曖昧ならビルドエラーになる

契約に合うレイアウトが3つある場合、こうなる

```
error: slide 1 ('slide-1'), line 5:  
  slide matches multiple layouts:  
  chapter, cover, statement  
help: pick one explicitly  
  with <!-- {"layout":"..."} -->
```

名前を間違えたら候補が出る

レイアウト名を `code` ではなく `kode` と書くようになる

```
error: slide 1 ('slide-1'), line 5:
```

```
unknown layout 'kode'
```

```
help: use one of: books, chapter, code-stack,  
code, compare, cover, photos, portrait,  
profile, shot, statement, topic
```

検査はCSSにも適用される

Markdown: スライドにキーを付ける

```
<!-- {"key": "whoami"} -->  
# 宮下 剛輔
```

CSS: data-slide-keyでキーを指す

```
[data-slide-key="whoami"] .slot-body a::before {  
  width: 26px;  
  height: 26px;  
}
```

存在しないキーを指すとビルドエラーになる

キーを付けたスライドを消してCSSだけ残すようになる

```
error: line 7: overrides.css:  
  unknown slide key 'whoami'  
  in override selector  
help: add "key":"whoami" to the page  
  settings comment of the slide it  
  styles (<!-- {"key":"whoami"} --> if  
  it has none), or delete this rule  
  from overrides.css
```

動的検査 (lint)

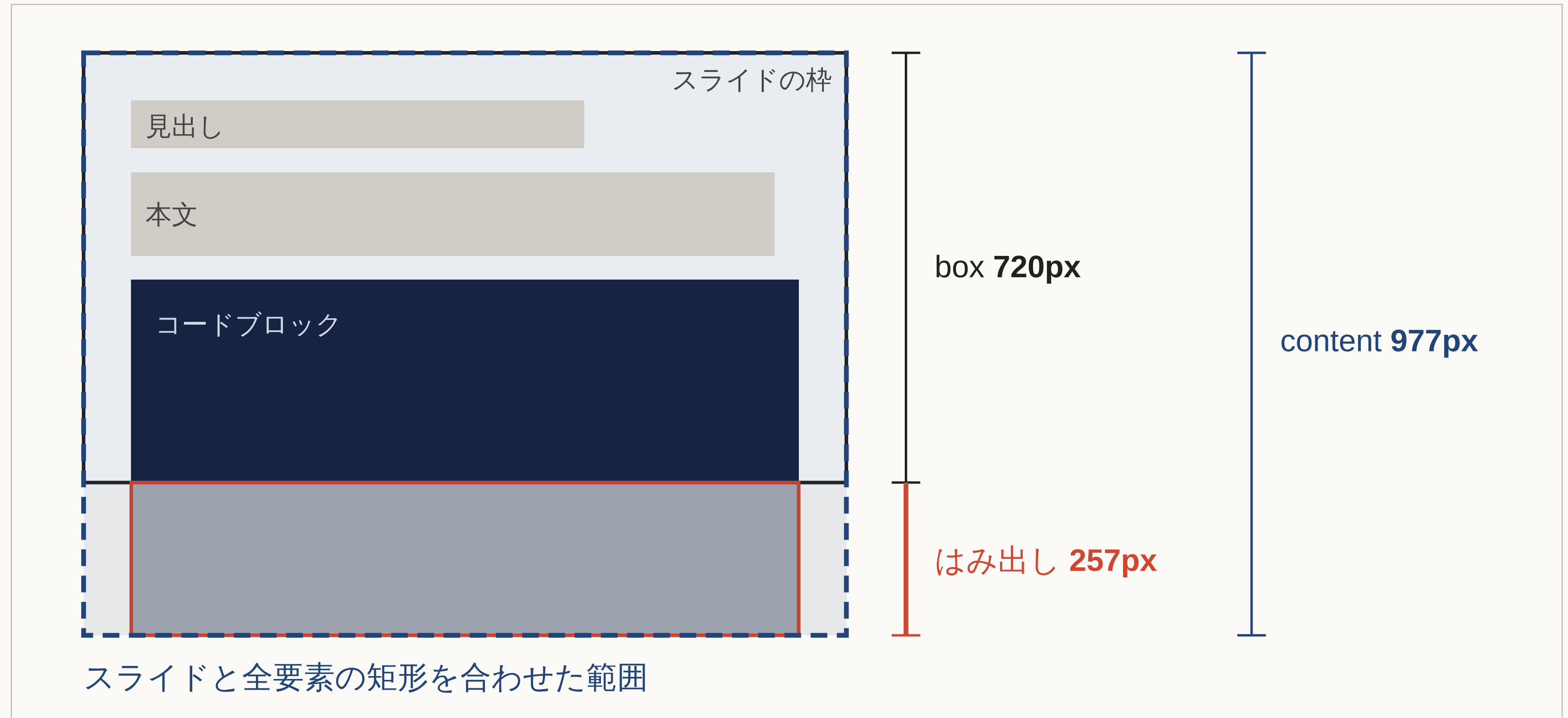
動的検査は型検査のあとで思いついた

- 型検査は最初の設計段階からあった
- はみ出しの検査を2週間後に追加
 - 入れてみたらPeithoのサンプルスライドで2枚はみ出していた
- さらに2週間後、文字サイズの検査を追加
 - はみ出しを文字を縮めてごまかせないようにする

動的検査がやっていること

- ビルドした全スライドをheadless Chromeで描画する
- はみ出し
 - スライドの枠: 全要素の `getBoundingClientRect()` と比べる
 - 要素の枠: 中身を切る要素（`overflow: hidden` など）の `scrollHeight - clientHeight`
- 文字サイズ: `getComputedStyle(要素).fontSize` の最小値を見る

スライドの枠からのみ出しの測り方



スライドの枠からはみ出した場合

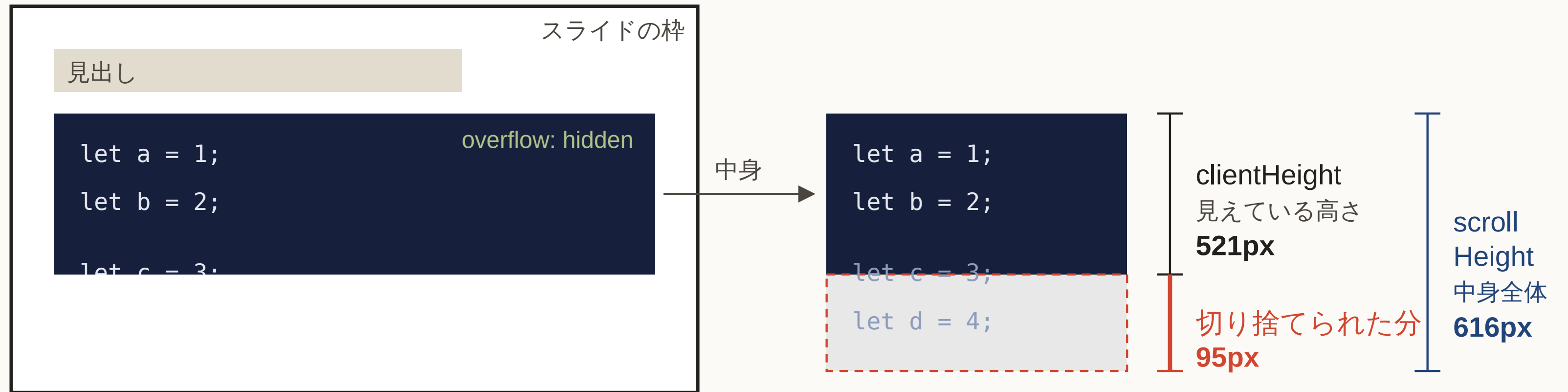
中身がスライドの枠を超えてこうなる

```
warning: slide 1 content overflows  
the slide box vertically by 257px  
(content 977px, box 720px)
```

```
help: shrink or split the slide content,  
or adjust the layout CSS
```

```
checked 1 slide(s): 1 warning(s)
```

要素の枠からはみ出しの測り方



パネルからははみ出している
スライドの枠からははみ出していない

切り捨てられた分 = scrollHeight - clientHeight

要素の枠からはみ出した場合

コードがパネルに収まらないようになる

```
warning: slide 1 content overflows  
the `code` slot vertically by 95px  
help: shrink or split the slide content,  
or adjust the layout CSS  
checked 1 slide(s): 1 warning(s)
```

文字が小さい場合

文字が24ptより小さいとこうなる

warning: slide 1 has text at 12pt,
below the recommended 24pt: "行1 ダミー..."
help: raise the font size in the layout CSS,
or move content to another slide
instead of shrinking it

わざと小さくするならCSSで宣言する

CSS: 下限を16ptに下げる

```
[data-slide-key="sources"] .slot-body {  
  font-size: 18pt;  
  --peitho-lint-min-font-size: 16pt;  
}
```

peitho lintの出力: 警告ではなくnoteになる

```
note: slide 1 has text at 18pt,  
allowed by --peitho-lint-min-font-size: 16pt: "出典"
```

なぜ描画して測るのか

- CSSを静的に読んでも実際の大きさは分からない
 - カスケード、継承、`em`や`%`のような相対単位がある
- 何行になるかは描いてみないと分からない
 - 折り返す位置はフォントの字幅によって変わる
- 描画してもすぐには大きさが確定しない
 - 画像とWebフォントは読み込みが終わると再描画される
 - 動的検査は読み込みを待ってから測る

ま と め

Peithoは型のあるプレゼンツール

- 静的検査（型検査）：内容がレイアウトの型に合うかを確認める
 - 合わなければ黙って捨てずにビルドエラーにする
- 動的検査（lint）：型では見つからない崩れを補う
 - はみ出しや小さすぎる文字を描画して測る

Peithoの柱は2つのプロダクトの掛け合わせ

- コンテンツとデザインを分離する
 - k1LoW/deckから
- Markdownをレイアウトのプレースホルダに流し込む
 - k1LoW/deckから
- 型で契約を縛り、合わなければエラーにする
 - Carinaから

ありがとうございました

github.com/mizzy/peitho

peitho.gosu.ke (ドキュメントとデモ)

```
brew install mizzy/tap/peitho
```