

Vivliostyle 活用ガイド

Table of Contents

活用ガイド	6
ガイド一覧	7
このセクションについて	7
脚注ガイド	8
概要	9
脚注認識メカニズムについて	9
脚注を実現する手段	10
CSS直書き方式（GCPMクラス方式）	10
DPUB-ARIAロール方式（#1700、PR#1703）	11
2つの認識メカニズムの比較	12
VFMによる変換（ <code>[^1]</code> 記法）	12
脚注のカスタマイズ	14
<code>footnote-display</code> プロパティ（#1825）	14
<code>::footnote-marker</code> の <code>list-style-position: outside</code> （#1702）	15
ページスコープリセットとクロススコープカウンタ	15
CSSプロパティ一覧	15
まとめ	15
参照	17

CMYK変換ガイド	18
この機能でできること・できないこと	19
前提知識：PDFのカラーオペレーター	21
CSS関数 <code>device-cmyk()</code>	22
仕組み	22
基本構文	22
使用できる場所の制約	22
アルファ値	23
プロセスカラーの実用例	23
CLIによるCMYK PDF出力	23
有効化	23
内部動作	23
SVGなど、CSS外で生まれるRGB色への対応： <code>reserveMap</code>	24
最後の手段： <code>overrideMap</code>	24
マッピング漏れの検出： <code>warnUnmapped</code>	24
実用上のワークフロー	25
画像の差し替え： <code>pdfPostprocess.replaceImage</code>	25
検証	25
参照	26
ページグループガイド	27
なぜ名前付きページが必要か	28
名前付きページの基本	28
<code>:nth()</code> ページセレクタ	29
ページグループという概念	29
<code>:nth(An+B of C)</code> —— ページグループ 内のセレクタ (v2.39新機能)	30
実践例:章構造のある書籍レイアウト	30
Vivliostyle Viewerでの確認方法	31
参照	32

CSS Nestingガイド	33
組版での使いどころ	34
1.疑似要素のネスト（Vivliostyle固有の組版疑似要素にも対応）	34
2.メディアクエリのインライン化	37
既存テーマへの取り込み	37
知っておきたい制限	37
関連ガイド	37
参考リンク	38

活用ガイド

Vivliostyleの最近の追加機能について、プロダクト別ではなく機能別に、仕様レベルの記述ではなく「**何ができるか**」「**どう使うか**」に焦点を当てた実践的なガイドをまとめています。

各ガイドには対応バージョンが冒頭に明記されているため、自分の環境で記事の内容が利用できるか一目で判断できます。

ガイド一覧

- **脚注（フットノート）** — DPUB-ARIA 脚注サポート、標準 `@page { @footnote { } }` ルール、`footnote-display`、VFMの新しい `footnote` モード（`pandoc` / `gcpm` / `dpub`）。
 - 対象バージョン: Vivliostyle.js v2.41 / CLI v10.5
- **CMYK変換** — CSSの `device-cmyk()` 関数と、CLIのPDF CMYK出力（`pdfPostprocess.cmyk`）。
 - 対象バージョン: Vivliostyle.js v2.40 / CLI v10.6
- **ページグループ** — 名前付きページ、`:nth(An+B of c)` ページセレクトタによる章ごとのレイアウト。
 - 対象バージョン: Vivliostyle.js v2.39
- **CSS Nesting ガイド** — CSS NestingによるテーマCSSの整理と、組版時の `::footnote-call` などへの応用。
 - 対象バージョン: Vivliostyle.js v2.42

このセクションについて

ここで扱う機能は、Vivliostyleの複数のプロダクト（コアCSS、CLI、VFM、テーマ）にまたがるものが多く、プロダクト別セクションには収まりにくいものです。「活用ガイド」では、プロダクト軸ではなく**機能軸**で整理しています。

仕様レベルのリファレンスはサポートするCSS機能とCore APIリファレンスを参照してください。

脚注ガイド

対象バージョン: `Vivliostyle.js v2.41` (2026-04-11)、`Vivliostyle CLI v10.5` **公開日:** 2026-05-05 **最終更新日:** 2026-05-14

このガイドでは、`Vivliostyle.js`（およびVFM）で脚注を扱うための手段と、脚注のカスタマイズ方法を解説します。

概要

`Vivliostyle`で脚注を実現する手段と、その設定・必要なCSSの対応:

やりたいこと	<code>vivliostyle.config.js</code>	テーマ・CSS	結果
後注（文末）	<code>vfm: { footnote: 'pandoc' }</code> (デフォルト)	不要	文末の後注セクション
HTMLで直接記述する脚注 (CSSクラス方式)	— (HTML直書き)	<code>theme-base / theme-techbook</code>	<code>float: footnote</code> による脚注
脚注 (VFM→GCPM変換)	<code>vfm: { footnote: 'gcpm' }</code>	<code>theme-base / theme-techbook</code>	CSSクラスベースの脚注
脚注 (VFM→DPUB変換)	<code>vfm: { footnote: 'dpub' }</code>	不要 (v2.41で自動対応)	DPUB-ARIAベースの脚注

脚注認識メカニズムについて

`Vivliostyle`が脚注として認識するHTMLマークアップには、由来の異なる2つの方式があります。

CSS GCPM (CSS Generated Content for Paged Media) は、W3CがページメディアのCSS組版向けに策定した仕様です ([CSS GCPM 3 \(https://www.w3.org/TR/css-gcpm-3/\)](https://www.w3.org/TR/css-gcpm-3/))。この仕様で定義された `float: footnote` を含むCSSクラスをHTMLに適用することで、脚注フロートを実現します。`Vivliostyle`は`theme-base / theme-techbook`にこのスタイルを同梱しており、v2.41以前から対応しています。

DPUB-ARIA (Digital Publishing WAI-ARIA) は、W3CがEPUBなどの電子出版物のアクセシビリティ向けに策定したARIAロールの拡張仕様です ([DPUB-ARIA 1.1 \(https://www.w3.org/TR/dpub-aria-1.1/\)](https://www.w3.org/TR/dpub-aria-1.1/))。 `role="doc-noteref" / role="doc-footnote"` というHTMLの `role` 属性で注の参照元と注本文をマークアップします。`Vivliostyle.js v2.41`で対応し、テーマCSSなしで脚注が使えるようになりました。

メカニズム	由来	HTML パターン	テーマ CSS
CSS GCPM	CSS Generated Content for Paged Media	<code></code>	<code>.footnote { float: footnote }</code> を含むテーマが 必要 (theme-base / theme-techbook)
DPUB-ARIA (v2.41 新 規)	W3C DPUB-ARIA 1.1	<code> →</code> <code><aside role="doc-footnote"></code>	不要 (v2.41 で自動対応)

脚注を実現する手段

CSS 直書き方式 (GCPM クラス方式)

最も直接的な記法は、HTML でマークアップする方法です:

```
<p>本文<span class="footnote">脚注の本文。</span>はここに続く。</p>
```

`.footnote { float: footnote }` を含むテーマ (theme-base / theme-techbook など) を使うと、`` の内容がレイアウト時にページ下部の脚注エリアに移動されます。本文中の参照番号と脚注エリアのマーカは、`::footnote-call` および `::footnote-marker` 擬似要素を介して生成されます。

HTML 中に `` として記述した脚注¹は、テーマ CSS (theme-base / theme-techbook など) の `.footnote { float: footnote }` によってページ下部の脚注エリアに送られる。

本文中に複数の参照を入れた場合²でも、マーカは `::footnote-call` と `::footnote-marker` 擬似要素で連番つきに生成される。

-
1. 脚注の本文。float: footnote によりレイアウト時にページ下部に移動される。
 2. もう一つの注。連番は自動的に振られる。

GCPM クラス方式の脚注。本文中の上付き番号と、ページ下部の脚注エリアにマーカー番号付きで自動配置される

VFM Markdown ソース

VFMが生成するHTML (body部)

DPUB-ARIA ロール方式 (#1700 (<https://github.com/vivliostyle/vivliostyle.js/issues/1700>)、PR#1703 (<https://github.com/vivliostyle/vivliostyle.js/pull/1703>))

Vivliostyle.js v2.41 は、DPUB-ARIA ロールから直接脚注を認識します:

```
<p>本文<a role="doc-noteref" href="#fn1">1</a>はここに続く。</p>
<aside id="fn1" role="doc-footnote">注の本文。</aside>
```

Vivliostyle.js が `[role="doc-footnote"]` に対して `float: footnote` を適用するため、これは**テーマCSSなし**で動作します。注はページ下部に表示され、本文中の参照番号 (`::footnote-call`) も自動生成されません。

本文中に `[^1]` と書くと、VFM dpub モードは `1` と `<aside role="doc-footnote">` を出力する。

Vivliostyle.js v2.41.0+ のビルトイン UA スタイルが `float: footnote` を自動で適用するため、テーマCSSなしでもページ下部の脚注エリアに配置される²。

¹脚注の本文。CSSは何も書いていない。

²もう一つの注。連番は自動。

DPUB-ARIA 脚注。テーマCSSを一切書かなくてもページ下部の脚注エリアに自動配置される

VFM Markdown ソース

▶ Details

VFMが生成するHTML (body部)

2つの認識メカニズムの比較

メカニズム	由来	認識パターン	テーマCSS
CSS GCPM	CSS Generated Content for Paged Media	<code></code>	必要
DPUB- ARIA	W3C DPUB-ARIA 1.1	<code>role="doc-noteref" / role="doc-footnote"</code>	不要 (v2.41)

2つのメカニズムは同一ドキュメント内で共存できますが、実際にはソースに応じてどれか1つを採用するのが普通です。

VFMによる変換（`[^1]` 記法）

VFM v2.6は `footnote` オプションを導入しました (PR#226 (<https://github.com/vivliostyle/vfm/pull/226>))、PR#231 (<https://github.com/vivliostyle/vfm/pull/231>))。Markdownの `[^1]` 記法から生成されるHTMLが、モードによって異なります:

- `'pandoc'` (デフォルト) — 後注を生成します。 `<section role="doc-endnotes">` としてドキュメント末尾に配置されます。 `float: footnote` の対象にはならず、 `@footnote` スタylingも影響しません。
- `'gcpm'` — `` を生成します。 `float: footnote` を含むテーマCSS (theme-base / theme-techbook など) が必要です。
- `'dpub'` — `<aside role="doc-footnote">` を生成します。 Vivliostyle.js v2.41が `float: footnote` を適用するため、**テーマCSS不要**です。将来デフォルト化予定 (#234 (<https://github.com/vivliostyle/vfm/issues/234>))。

`vivliostyle.config.js` での設定:

```
export default {
  vfm: {
    footnote: 'dpub',
  },
  // ...
};
```

pandoc モード (後注) の例

VFM v2.5.x以前では、Markdownの `[^1]` 記法はPandoc出力スタイルの後注を生成しました:

本文。[^1]

[^1]: 注の本文。

VFMはこれを `<section role="doc-endnotes">` に変換し、**ドキュメント本文の末尾**に配置します。後注は本文と同じフローに置かれるため、`float: footnote` の対象にはなりません。組み込みの `@footnote` スタイリングも後注には影響しません。

VFM のデフォルト設定 (`footnote: 'pandoc'`) では、`[^1]` 記法は `<section role="doc-endnotes">` として本文末尾に追加される¹。

本文と通常フローでつながるため、`float: footnote` や `@page { @footnote { } }` の対象にならない²点に注意。

-
1. 最初の後注の本文。↩
 2. 次の後注の本文。↩

VFM Pandoc形式の後注。本文末尾に`<section role="doc-endnotes">`がフローし、各注に戻るリンクが付く

VFM Markdown ソース

VFMが生成するHTML (body部)

footnote オプションのオブジェクト形式

`footnote` はオブジェクト `{ mode, body }` の形でも指定でき、生成HTMLをカスタマイズできます:

```
vmf: {
  footnote: {
    mode: 'gcpm',
    body: (h, props, children) =>
      h('span.footnote', props, h('span.footnote-wrap', ...children)),
  },
}
```

用途は、ぶら下げインデント用ラッパーのような**スタイリング目的のDOM変形をMarkdownソースから分離**することです。Markdownには素のままに脚注を書き、ビルド設定側でラッパーを適用します。

YAMLフロントマターによるファイル単位の設定

脚注モードはファイルごとにYAMLフロントマターで指定でき、グローバル設定を上書きできます:

```
---
vfm:
  footnote: dpub # | pandoc | gcpm
---
```

プロジェクト全体ではデフォルト設定を使いつつ、特定のファイルだけ別モードに切り替えたい場合に便利です。

脚注のカスタマイズ

footnote-display プロパティ ([#1825 \(https://github.com/vivliostyle/vivliostyle.js/issues/1825\)](https://github.com/vivliostyle/vivliostyle.js/issues/1825))

脚注本体のレイアウト方式を切り替えます。指定先は `@footnote` ルールの中ではなく**脚注要素自身**です:

- `block` (デフォルト) — 各脚注を改行
- `inline` — 複数の脚注を同じ行にフロー
- `compact` — 短い脚注はインライン、長い脚注は自動的にブロックヘフォルバック

```
.footnote { footnote-display: inline; }
```

VFM Markdown ソース (`footnote-display: inline`)

▶Details

VFMが生成するHTML (body部)

`compact` は、短い注はインライン、長い注は自動でブロックにフォールバックします。

`::footnote-marker` の `list-style-position: outside` (#1702 (<https://github.com/vivliostyle/vivliostyle.js/issues/1702>))

脚注マーカーが `list-style-position` を尊重するようになりました。マーカーを脚注本体の外側に配置することで、ぶら下げインデントの体裁が作れます:

```
.footnote { margin-inline-start: 1.5em; }
.footnote::footnote-marker {
  content: counter(footnote) ". ";
  list-style-position: outside;
}
```

VFM Markdown ソース

VFMが生成するHTML (body部)

ページスコープリセットとクロススコープカウンタ

脚注カウンタをページグループ単位でリセットできるようになりました。章ごとに脚注番号を新規に採番する書籍で便利です。 `counter-reset` を、[ページグループガイド](#)で扱う名前付きページの仕組みと組み合わせて使います。

CSS プロパティ一覧

プロパティ・at-rule	役割
<code>float: footnote</code>	ブロックレベル要素をページ下部の脚注エリアに移動
<code>footnote-display</code>	<code>block</code> / <code>inline</code> / <code>compact</code>
<code>footnote-policy</code>	注をページ間で分割するかを制御
<code>::footnote-call</code>	本文中の参照マーカーの擬似要素
<code>::footnote-marker</code>	注本体の横に表示されるマーカーの擬似要素

まとめ

VFMで利用できる注の種類と、その記法・設定・CSSの対応:

注の種類	VFM での記 法	<code>vivliostyle.config.js</code>	テーマ・CSS	備考
後注 (endnotes)	<code>[^1]</code>	<code>vfm: { footnote: 'pandoc' }</code> (デフォルト)	不要 (通常フローで <code><section role="doc-endnotes"></code> 出力)	<code>@footnote</code> 非対応
脚注・CSSク ラス方式 (footnotes)	<code>[^1]</code>	<code>vfm: { footnote: 'gcpm' }</code>	<code>.footnote { float: footnote }</code> を含むテーマが必要 (theme-base / theme-techbook)	<code></code> として出力。VFM v2.6新機能
脚注・DPUB- ARIA方式 (footnotes)	<code>[^1]</code>	<code>vfm: { footnote: 'dpub' }</code>	不要 (v2.41で自動対応)	<code><aside role="doc-footnote"></code> として出力。VFM v2.6新機能。将来デフォルト化予定 (#234 (https://github.com/vivliostyle/vfm/issues/234))
傍注 (sidenotes)	—	—	—	VFMの脚注モード (pandoc / gcpm / dpub) ではサポートされない。傍注を実現するにはカスタムHTML+CSSによる独自実装が必要

gcpm と dpub の選択指針

- **dpub 推奨** テーマCSS 不要で手軽に脚注が使える。将来デフォルト化予定
- **gcpm**: 既存テーマ (theme-base / theme-techbook) との互換性が必要な場合、またはオブジェクト形式の `body` コールバックでHTMLを変形したい場合

参照

- W3C CSS GCPM 3 §2 Footnotes (<https://www.w3.org/TR/css-gcpm-3/#footnotes>)
- W3C DPub-ARIA 1.1 (<https://www.w3.org/TR/dpub-aria-1.1/>) (`doc-noteref` / `doc-footnote` の定義)
- W3C 日本語組版処理の要件 §4.2 注の処理 (<https://www.w3.org/TR/jlreq/#processing-of-notes-in-japanese>)
- Qiita: @u1f992 「Vivliostyle.js v2.41 (CLI v10.5) の脚注機能を Markdown で利用する」 (<https://qiita.com/u1f992/items/6466c03aa4f569a39572>)
- テストケース: `vivliostyle.js/packages/core/test/files/footnotes/` (<https://github.com/vivliostyle/vivliostyle.js/tree/master/packages/core/test/files/footnotes/>)
- VFM ソース: `vfm/src/plugins/footnotes.ts` (<https://github.com/vivliostyle/vfm/blob/main/packages/vfm/src/plugins/footnotes.ts>)、PR#226 (<https://github.com/vivliostyle/vfm/pull/226>)、PR#231 (<https://github.com/vivliostyle/vfm/pull/231>)
- VFM Issue #234 (<https://github.com/vivliostyle/vfm/issues/234>) (dpub デフォルト化計画)
- テーマ: `theme-base/css/partial/footnote.css` (<https://github.com/vivliostyle/themes/blob/main/packages/%40vivliostyle/theme-base/css/partial/footnote.css>)、`theme-techbook/theme.css` (<https://github.com/vivliostyle/themes/blob/main/packages/%40vivliostyle/theme-techbook/theme.css>)

CMYK 変換ガイド

対象バージョン: *Vivliostyle.js v2.40 (2026-01-11)*、*Vivliostyle CLI v10.6* **公開日:** 2026-05-05 **最終更新日:** 2026-05-16

このガイドは、`device-cmyk()` CSS関数と `pdfPostprocess.cmyk` の仕組みと制約を説明します。

この機能でできること・できないこと

Vivliostyle CLI は Chromium を制御して PDF を生成します。Web ブラウザは CMYK カラー空間を原理的にサポートしないため、Chromium が出力する PDF の色はすべて DeviceRGB です。この機能はその後段で MuPDF を使った後処理を行い、PDF コンテンツストリーム内の DeviceRGB カラー演算子 (`rg / RG`) を DeviceCMYK カラー演算子 (`k / K`) に置換します。

変換される (DeviceRGB カラー演算子として出力される箇所) :

- テキストの色 (`color` プロパティ)
- 単色の背景色・ボーダー (`background-color`、`border-color`)
- SVG のベクター要素の単色塗り (`reserveMap` 経由)

変換されない (DeviceRGB カラー演算子以外の方法で PDF に書き込まれる箇所) :

- **ラスター画像** (JPEG・PNG 等) — あらかじめ CMYK 変換した画像素材を用意し `replaceImage` で差し替えてください
- **グラデーション** (`linear-gradient()` 等) — Chromium が PDF の Shading Object として出力するため置換対象外
- **フィルタ・ブレンドモード等** — 透明グループ等の別機構で表現されるため置換対象外

つまり本機能は、画像など既存のソリューションが確立している領域は扱わず、**文字・ベクター単色塗り**に限定した置換を行う、割り切った実装です。実用にあたっては素直な単色指定の CSS を心がけ、加工済みの画像素材を用意することが前提になります。

なお、印刷会社が RGB 入稿を受け付ける場合は、そちらを選択するほうが確実なケースも少なくありません。

前提知識：PDF のカラーオペレーター

この機能の動作を正しく理解するには、PDF のカラーオペレーターについての知識が必要です。

PDF のコンテンツストリームでは、色をオペレーター 1 語で指定するショートカット形式があります。オペレーター名自体がカラースペースを内包しているため、別途カラースペースの宣言を書く必要がありません。これを**暗黙型カラーオペレーター**と呼びます。

オペレーター	カラースペース	対象
<code>rg</code> (小文字)	DeviceRGB	塗り (fill)
<code>RG</code> (大文字)	DeviceRGB	線 (stroke)
<code>k</code> (小文字)	DeviceCMYK	塗り (fill)
<code>K</code> (大文字)	DeviceCMYK	線 (stroke)

Chromium はテキスト・単色塗りを `rg / RG` オペレーターとして PDF に出力します。Vivliostyle CLI の後処理はこの `rg` を `k` に、`RG` を `K` に置換します。引数の数が RGB の 3 値から CMYK の 4 値に変わるため、

RGB→CMYKのマッピングが必要です。このマッピングを提供するのが `reserveMap` と `overrideMap` です。

CSS関数 `device-cmyk()`

`device-cmyk()` は [CSS Color 5 \(https://drafts.csswg.org/css-color-5/#device-cmyk\)](https://drafts.csswg.org/css-color-5/#device-cmyk) で定義されています。

仕組み

VivliostyleはWebブラウザ上で動作するため、CSS処理段階では `device-cmyk()` を一旦 `color(srgb r g b)` に変換し、そのRGB値とCMYK値の対応をCmykStoreに記録します。実際のCMYK化はCLIの後処理で行われます。

基本構文

```
color: device-cmyk(0 1 1 0);          /* プロセスレッド (M=Y=100%) */
color: device-cmyk(100% 0% 0% 0%);  /* プロセスシアン (パーセント表記) */
color: device-cmyk(0 0 0 1);        /* 純黒 (K=100%) */
```

引数の順序は**C, M, Y, K**。各チャンネルは `0..1` の数値またはパーセント値 (`0%..100%`) です。カンマ区切りのレガシー構文 (CSS3時代の互換用) も受け付けます。

使用できる場所の制約

`device-cmyk()` は `<color>` 値が使える任意のCSSプロパティに記述できますが、CMYK変換が有効なのは変換後のRGB値がPDFコンテンツストリームの `rg / RG` オペレーターとして直接露出する箇所に限られます。単色の `color` ・ `background-color` ・ `border-color` は概ねこれに該当しますが、グラデーション (`linear-gradient()` 等) は対象外です。グラデーション内での `device-cmyk()` 使用は避けてください。

```
/* OK: CMYK変換される */
h1 { color: device-cmyk(0 0 0 1); }
.box { background-color: device-cmyk(0 0.1 0.2 0); }
.box { border: 1pt solid device-cmyk(0 0.5 1 0.1); }

/* NG: CMYK変換されない (グラデーションはShading Objectになる) */
.box {
  background-image: linear-gradient(
    device-cmyk(1 0 0 0),
    device-cmyk(0 0 0 0)
  );
}
```

アルファ値

末尾に `/` でアルファを指定できますが、アルファはカラーオペレーターとは別の仕組みで表現されるため、後処理での置換においてアルファは無視されます。

プロセスカラーの実用例

用途	CMYK	備考
リッチブラック（大面積のベタ用）	<code>device-cmyk(0.4 0.3 0.3 1)</code>	K単色がやや灰色に見える問題を回避
プロセスレッド	<code>device-cmyk(0 1 1 0)</code>	M=Y=100%
プロセスグリーン	<code>device-cmyk(1 0 1 0)</code>	C=Y=100%
プロセスブルー	<code>device-cmyk(1 1 0 0)</code>	C=M=100%
淡い塗り（囲み・ハイライト用）	<code>device-cmyk(0 0.1 0.2 0)</code>	暖色系の淡いトーン

CLIによるCMYK PDF出力

有効化

```
// vivliostyle.config.js
export default {
  pdfPostprocess: {
    cmyk: true, // または cmyk: {}（等価）
  },
};
```

デフォルトは `false`。 `false` のときは `device-cmyk()` で指定した色のCMYK置換も行われません。

内部動作

後処理はMuPDFで実行されます。PDFのコンテンツストリームを走査し、DeviceRGBカラーオペレーター（`rg / RG`）をDeviceCMYKカラーオペレーター（`k / K`）に置換します。置換に使うRGB→CMYKのマッピングは、`device-cmyk()` からCmykStoreに記録された値と、`reserveMap`・`overrideMap`で指定した値から構成されます。

SVGなど、CSS外で生まれるRGB色への対応：

reserveMap

SVGのベクター要素の塗り色はChromiumが直接RGB値としてPDFに出力するため、`device-cmyk()`では指定できません。これらのRGB→CMYKの対応を `reserveMap` でCSS処理前にCmykStoreへ登録します。

```
pdfPostprocess: {
  cmyk: {
    reserveMap: [
      ['#FF6633', { c: 0, m: 7000, y: 9000, k: 0 }],
      ['#003366', { c: 10000, m: 8000, y: 2000, k: 4000 }],
    ],
  },
},
```

エントリ形式： `[rgb, { c, m, y, k }]` のタプル配列。

値のスケール： CMYK・RGB値ともに**0-10000の整数**（10000 = 100%、最小単位0.0001%）。JavaScriptの浮動小数点数の精度問題を避けるためこの表現が使われています。

RGBのキー： Hex文字列（`'#FF6633'`）も使用できます。Chromiumの内部実装をエミュレートした処理により0-10000のRGB値に正規化されます。

最後の手段： overrideMap

`overrideMap` は、`reserveMap` もCSSの `device-cmyk()` も届かない色が残存する場合の回避手段です。内部で生成される不可視要素など、ソース上に現れないRGB色が後処理後も残る場合に使います。後処理時にCmykStoreの最終マップに上書きマージされます。

エントリ形式・値スケールは `reserveMap` と同じです。`reserveMap` で登録済みの色は警告にも現れないため、同じ色を `overrideMap` に追加する必要はありません。

マッピング漏れの検出： warnUnmapped

後処理後にPDFに残ったDeviceRGBオペレーターに対して警告します。PDFの段階ではCSS由来かSVG由来かを区別できないため、残存するすべてのRGB色が対象になります。

```
pdfPostprocess: {
  cmyk: {
    warnUnmapped: true,
  },
},
```

警告に現れたRGB色を `reserveMap` に追加していくことが、実際のワークフローになります。

実用上のワークフロー

1. CSSは単色指定（`color` ・ `background-color` ・ `border`）に留め、`device-cmyk()` で色を指定する
2. ラスター画像はあらかじめCMYK変換した素材を用意する（`replaceImage` で差し替え）
3. SVGのベクター色を `reserveMap` に登録する
4. `warnUnmapped: true` を有効にして出力し、警告のRGB色を `reserveMap` に追加する
5. Ghostscriptの `inkcov` で最終的なインクカバレッジを確認する

画像の差し替え：`pdfPostprocess.replaceImage`

`replaceImage` は `cmyk` とは独立した機能で、`cmyk: false` の場合でも動作します。

```
pdfPostprocess: {
  replaceImage: [
    { source: 'cover.jpg', replacement: 'cover-print.tiff' },
    { source: /^images\/web-(.+)\.jpg$/, replacement: 'images/print-$1.tiff' },
  ],
},
```

- `source` : 文字列またはJavaScript正規表現
- `replacement` : 差し替え先のパス
- 両者とも原稿ディレクトリ（`entryContextDir`、既定は `.`）を起点に解決されます

検証

生成PDFのインクカバレッジを確認するには、Ghostscriptの `inkcov` デバイスを使います:

```
gs -o - -sDEVICE=inkcov mybook.pdf
```

ページごとのインクカバレッジが4つの数値 (C M Y K) として `0..1` の範囲で出力されます。K以外の値が出ているページには意図しないインクが乗っています。

参照

- W3C CSS Color 5 — `device-cmyk()` (<https://drafts.csswg.org/css-color-5/#device-cmyk>)
- CLI例: `vivliostyle-cli/examples/cmyk/` (<https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/cmyk>)

ページグループガイド

対象バージョン: Vivliostyle.js v2.39 (2025-12-25) 公開日: 2026-05-05 最終更新日: 2026-05-05

このガイドでは、名前付きページと `:nth(A+B of C)` ページセレクタを解説します。これらを組み合わせることで、同一ファイル内のセクションごとにまったく異なるページレイアウトを与えられるようになります。

なぜ名前付きページが必要か

書籍では、1冊の中で複数のページデザインが必要になることがよくあります:

- 写真エッセイ章のための、図版に余裕のある幅広レイアウト
- 用語集のための、狭い2段組レイアウト
- 表紙や扉のための、フルブリードレイアウト

CSSは `@page :left / :right / :first` でページを区別できますが、これらは見開きの左右と文書全体の最初のページしか区別しません。今いる章を区別することはできません。名前付きページはその穴を埋めます。

名前付きページの基本

`page` プロパティで、ブロックレベルの要素 (section、`<div>` など) をそれが表示されるページの種類の名前を指定します:

```
section.glossary {
  page: narrow;
}

section.gallery {
  page: wide;
}
```

そのうえで、対応する `@page` ルールでスタイルを指定します:

```
@page narrow {
  size: A5;
  margin: 1.5cm 1cm;
}

@page wide {
  size: A4 landscape;
  margin: 1cm;
}
```

`section.glossary` をレイアウトしたページは `narrow`、`section.gallery` のページは `wide` を使います。

:nth() ページセレクタ

1つの `@page` 名（または無名のデフォルト）の中で、`:nth()` は位置でページを選択します:

```
@page :nth(1)      { /* 最初のページ */ }
@page :nth(odd)   { /* 奇数ページ */ }
@page :nth(even)  { /* 偶数ページ */ }
@page :nth(3n+2)  { /* 2ページ目から3ページごと */ }
@page :nth(-n+3)  { /* 最初の3ページ */ }
```

An+B 形式は `:nth-child` と同じ規則です。n は 0, 1, 2, ... を取り、結果が0以下になる場合は無視されます。

ページグループという概念

レイアウトエンジンがページにコンテンツを流し込む過程で、フローボックスに付いた `page` 名が変わる（例: `glossary` の後に `gallery` が来る）と、Vivliostyle は強制改ページを発行します。1つの名前付きページブロックから生成された、連続するページの塊が**ページグループ**です。

同じ `page` 名の要素が複数続く場合（例:複数の `section.chapter`）は、`break-before: page` を指定しないと改ページされず、新しいページグループも開始されません。

言い換えれば、ページグループは「同じ `@page` 名を使い、ソースの同じフローレベル領域から生成された、連続するページの並び」です。これが `:nth(... of <name>)` の選択単位です。

:nth(An+B of C) — ページグループ内のセレクトタ (v2.39 新機能)

Vivliostyle.js v2.39 は `:nth(An+B of <name>)` 形式に対応しました。インデックスを名前付きページグループ内に限定します:

```
@page :nth(1 of body) {
  /* `page: body` を使う各章の最初のページ */
  @top-center { content: none; }
  margin-top: 4cm;
}

@page :nth(odd of body) {
  /* 各 `body` グループ内の奇数位置のページ */
  @bottom-right { content: counter(page); }
}
```

これにより、「各章の最初のページだけ別デザイン」が簡潔に表現できます。`of <name>` がないと `:nth(1)` は文書全体の最初のページしか選べませんが、`of <name>` を付ければ各グループの最初のページをそれぞれ別個に選択できます。

実践例:章構造のある書籍レイアウト

名前付きページと `:nth(... of <name>)` を組み合わせた例:

```

/* ソース構造
  <section class="cover">      ... </section>
  <section class="chapter">   ... </section>
  <section class="chapter">   ... </section>
  <section class="glossary">  ... </section>
*/

section.cover    { page: cover; }
section.chapter  { page: body; break-before: page; }
section.glossary { page: narrow; }

/* 各章の扉ページ */
@page :nth(1 of body) {
  @top-center { content: none; }
  margin-top: 4cm;
}

/* 章の2ページ目以降にはランニングヘッダーを表示 */
@page :nth(n+2 of body) {
  @top-center { content: string(chapter-title); }
}

/* 表紙 */
@page cover {
  size: A4;
  margin: 0;
  background: black;
}

```

このパターンはCSS GCPM 3仕様の Example 13・14で示されており、Vivliostyle.js v2.39でネイティブにサポートされるようになりました。

Vivliostyle Viewerでの確認方法

@page ルールはページ分割描画でしか効果が見えません。ページグループ出力を確認するには:

1. `vivliostyle preview` (または `npm run preview`) でビルドする
2. 生成HTMLを Vivliostyle Viewer (<https://vivliostyle.org/viewer/>) で開く
3. 「見開き」表示を切り替えて左右ページを確認
4. 各章が扉ページ用のレイアウトで始まっていることを確認

静的に検証したい場合は、`vivliostyle build` を実行して出力PDFを点検します。

参照

- W3C CSS GCPM 3 §3 Selecting Pages (<https://www.w3.org/TR/css-gcpm-3/#the-first-page-pseudo-element>)
- テストケース:
 - `vivliostyle.js/packages/core/test/files/named-pages/page-groups.html`
 - `vivliostyle.js/packages/core/test/files/nth-page/nth-of-page.html`

CSS Nesting ガイド

対象バージョン: Vivliostyle.js v2.42 (2026-04-25) Vivliostyle CLI v10.6 は Viewer 2.42 を同梱しているため、追加設定なしに利用できます。

公開日: 2026-05-14 **最終更新日:** 2026-05-14

CSS Nesting Module (<https://www.w3.org/TR/css-nesting-1/>)は、CSSのルールを入れ子にして記述するしくみです。従来のCSSでは `.chapter { ... } .chapter h2 { ... }` のように親セレクタを繰り返し書く必要がありましたが、ネスト記法では `.chapter { }` の中に子ルールをまとめて書けます。

```
.chapter {
  font-family: 'Source Serif Pro', serif;

  & h2 {
    font-size: 1.4em;
    border-bottom: 1px solid currentColor;
  }

  & p:first-of-type::first-letter {
    font-size: 3em;
    float: left;
    margin-right: 0.1em;
  }
}
```

これは展開すると以下と等価です。

```
.chapter { font-family: 'Source Serif Pro', serif; }
.chapter h2 { font-size: 1.4em; border-bottom: 1px solid currentColor; }
.chapter p:first-of-type::first-letter { font-size: 3em; float: left; margin-right: 0.1em; }
```

子ルール先頭の `&` は省略できます (`h2 { ... }` と書いても同じ意味です)。本ガイドでは、ネストであることを明示するため `&` を付けています。

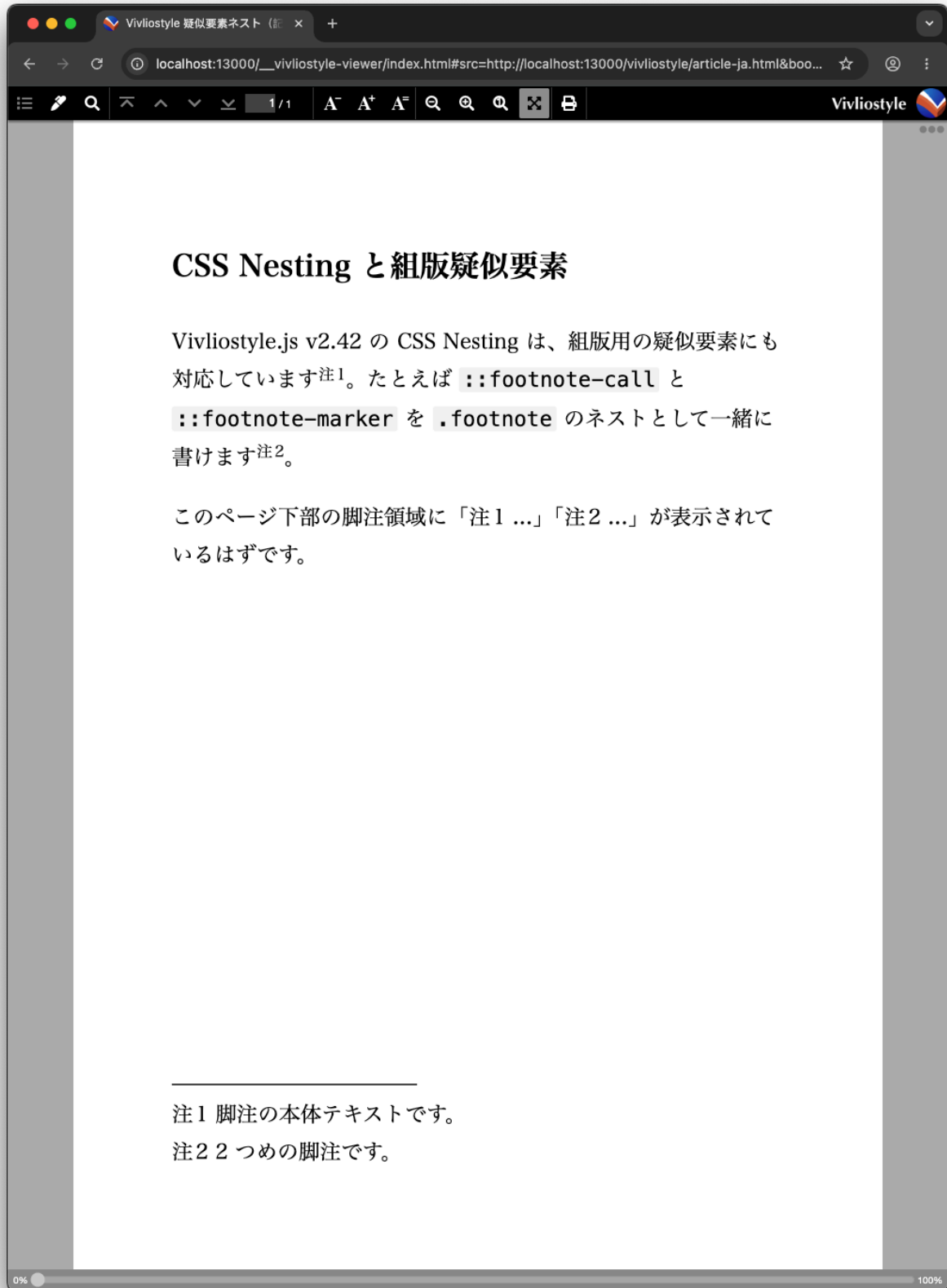
組版での使いどころ

1. 疑似要素のネスト (Vivliostyle 固有の組版疑似要素にも対応)

Vivliostyle.js v2.42 の CSS Nesting は、Vivliostyle が組版時に処理する CSS GCPM の疑似要素 `::footnote-call` / `::footnote-marker` などに対しても動作します。これにより、脚注スタイルを1ブロックにまとめて記述できます。

```
.footnote {  
  float: footnote;  
  
  &::footnote-call {  
    content: "注" counter(footnote);  
  }  
  
  &::footnote-marker {  
    content: "注" counter(footnote) " ";  
  }  
}
```

`.footnote` クラスを脚注として使用しつつ、本文中の呼び出し記号と脚注領域のマーカ―を同じスコープで定義できます。



疑似要素`::footnote-call`と`::footnote-marker`のネストが動作する例。本文中に「注1」「注2」が、ページ下部の脚注領域に「注1 ...」「注2 ...」が表示される

HTMLソース

▶ Details

参考: [メイユール \(@u1f992\) 氏の投稿 \(https://x.com/u1f992/status/2053013531600257211\)](https://x.com/u1f992/status/2053013531600257211)

2. メディアクエリのインライン化

`@media print` を必要なルールの内側に書けます。プレビュー時にだけ強調する装飾を切り分けやすくなります。

```
.cover {
  background: var(--cover-bg);

  @media screen {
    box-shadow: 0 4px 16px rgba(0,0,0,0.2);
  }

  @media print {
    box-shadow: none;
    break-after: page;
  }
}
```

既存テーマへの取り込み

- **既存のCSSファイルへの追加:** 今あるテーマのCSSファイルにネスト記法をそのまま追記できます。特別な設定変更は不要です。

知っておきたい制限

- ネストできるのはルール（`@page` / `@media` / クラス・要素セレクタ）の中身です。プロパティの中にネストはできません。

関連ガイド

- [脚注（フットノート）](#) — v2.42 で同時に脚注機能も拡張されました
- [CMYK変換](#)
- [ページグループ](#)

参考リンク

- [CSS Nesting Module — W3C](https://www.w3.org/TR/css-nesting-1/) (https://www.w3.org/TR/css-nesting-1/)
- [Vivliostyle.js #1032 — Add CSS Nesting support](https://github.com/vivliostyle/vivliostyle.js/issues/1032) (https://github.com/vivliostyle/vivliostyle.js/issues/1032)