記事を PDF でも読めるようにした

著者：関 勝寿　公開日：2026 年 9 月 13 日　ソース：https://sekika.github.io/2026/09/13/minitype-pdf/

このブログの記事に、ブラウザで読む本文とは別に、PDF で表示するリンクを付けた。記事のヘッダにある「PDF で表示」を選ぶと、A4 縦・二段組の PDF を新しいタブで開ける。

PDF は印刷したいときや、あとで手元に保存して読みたいときに便利である。画面幅やブラウザの表示倍率に左右されず、見出し、本文、画像、表を一定の版面で読めるのもよい点である。文字は埋め込みフォントで出力しているので、PDF ビューアで選択、検索、コピーできる。PDF 内の Web リンクも青字のリンクとして残してあるので、参考資料や元の記事へ移動できる。

すべての記事を無条件に PDF 化しているわけではない。ブラウザで操作することが本体である 15 パズル、Pyodide、Canvas を使うページなどは PDF にしても意味が変わってしまうため、PDF リンクを表示しない。通常の解説記事は、日本語・英語を問わず PDF 化する。

1.minitype を使う

PDF の作成には minitype を使った。minitype は、TypeScript ライブラリとして利用するヘッドレス組版エンジンである。日本語の段組、禁則処理、縦組、ルビなどの組版機能を持つ。

このブログの記事は Markdown で書いているので、minitype の Markdown プラグインで記事ファイルを読み込む。見出し、段落、リスト、コードブロック、表、画像、リンクなどを minitype の組版要素に変換できる。PDF を書き出す処理は Node.js で実行する。

記事の記述には Markdown だけでなく生 HTML も残っている。このブログでは、リポジトリ内のファイルを参照する img タグを Markdown 画像に変換する。PNG や JPEG などはそのまま配置し、SVG 画像は PDF に埋め込める PNG へ変換してから配置する。また、`[ リンク文字列 ](...)` も Markdown リンクへ変換するので、PDF では青字で表示され、クリックして移動できる。コードブロックや Jekyll の highlight ブロック内の HTML は、説明用のソースコードなので変換しない。

数式を使う既存記事には `layout: math` と `layout: katex` がある。これらのレイアウトでは、`$$` … `$$` または `[[` … `]]` で囲んだ独立した式と、`$` … `$` で囲んだ行内式を検出し、minitype の数式要素として組版する。数式の中で重ねて書かれたバックスラッシュも TeX の命令として正規化する。コードブロック、インラインコード、Jekyll の highlight ブロック内にある同じ記法は、説明用のソースコードとして変換しない。

この方式は、公開済みの HTML を画面キャプチャのように PDF 化するものではない。Markdown を読み、PDF 用の版面として改めて組版する。そのため、Web ページのボタンや JavaScript の実行結果を PDF に写すのではなく、記事本文を読みやすい文書にすることを目的としている。

2. このブログの組版設定

生成処理は generate-pdfs.mjs に置いた。用紙は A4 縦、横組で、上下左右の余白を指定し、本文を二段組にしている。タイトルはページ中央、著者・公開日・ソース URL は右寄せとし、ページ番号は本文の下余白を保ったままページ下部に置く。

Markdown 中のリンクには PDF のリンク注釈を付け、文字色も青にしている。minitype が生成する四角い注釈は PDF ビューアによっては赤枠や選択状態として見えるため、出力後にその注釈だけを取り除き、通常のリンク注釈は残すようにした。文字はアウトライン化せず、必要な文字を含むフォントのサブセットを PDF へ埋め込む設定にした。したがって、文字を選択して検索やコピーができる。

記事の PDF は、原則として記事 URL に対応する次の場所に保存する。

```
/pdf/YYYY/MM/DD/slug.pdf
```

たとえば `/2024/05/12/Bootstrap5/` の PDF は `/pdf/2024/05/12/Bootstrap5.pdf` となる。記事ヘッダは日本語用と英語用でこの規則から PDF の URL を組み立てるため、記事ごとにリンク先を書く必要はない。英語記事ではリンクの文言を View PDF に切り替えている。

3.PDF 化しない記事の判定

通常の記事はデフォルトで PDF 生成対象である。明示的に除外したいときは、フロントマターに次のように書く。

```
pdf: false
```

現在の実行型ページには、この指定を付けている。生成スクリプトも念のため記事本文を調べ、コード例の外にある script、canvas、form、input、textarea、select、button、iframe などを検出した記事を除外する。JavaScript のコードを解説としてコードブロックに掲載しているだけなら、PDF 化の対象のままである。

この判定により、15 パズルのように Canvas とボタンを使う記事や、ブラウザ内で Python を実行する Pyodide の記事は PDF を作らない。一方、JavaScript や Pyodide について説明する通常の記事は、本文を読むための PDF を作れる。

4.pre-commit で更新する

PDF を手作業で作ると、記事本文を更新した後に PDF だけ古くなる。そこで、記事をコミットするときに動く Git の pre-commit フックへ PDF 生成を組み込んだ。

初回だけ、PDF 生成に必要な Node.js パッケージをインストールし、フックを登録する。

```
cd tools/minitype-pdf
npm install
cd ../..
make -C setup install-hook
```

_posts/ 内の記事を追加・変更してコミットすると、フックは検索用の js/index.js を更新した後、未生成の PDF を一括生成する。コミット対象の記事に対応する PDF がすでにある場合も、本文と PDF が食い違わないよう再生成する。最後に生成物を自動でステージするため、通常は pdf/ を個別に git add しなくてよい。

手動で未生成 PDF を作る場合は、リポジトリのルートで次を実行する。

```
node tools/minitype-pdf/generate-pdfs.mjs --missing
```

詳細な前提条件と対象外判定は setup/Readme.md にまとめた。

5.GitHub Pages との役割分担

GitHub Pages の標準的な Jekyll ビルドは、リポジトリにある HTML、CSS、JavaScript、画像、PDF などの静的ファイルを公開する。一方、minitype の PDF 生成には Node.js の実行が必要であり、GitHub Pages の Jekyll ビルドの中で行うものではない。

このブログでは、ローカルの pre-commit フックで PDF を作り、生成した PDF を記事と同じ Git コミットに含める。GitHub Pages は、その PDF をほかの静的ファイルと同じように配信するだけである。この分担により、GitHub Pages の設定を複雑にせず、記事と対応する PDF を同じ履歴で管理できる。

今後は、数式や複雑な表、画像を多く含む記事についても、PDF での見やすさを確認しながら組版を調整していく予定である。