

i 補足: この記事は Typst Advent Calender 2025 の TODO 日目の記事です。

## 執筆環境

ソフト名	バージョン	補足
Typst	0.14	TYPST_FEATURES=html
Tinymist Typst	0.14.4	—

## 1. はじめに

Typst、とてもいいですね。Markdown のような書き心地で figure や footnote や参考文献への参照等が書けるためとても気に入っています。

その Typst ですがバージョン 0.14 で HTML エクスポート機能が大幅に強化されました。なんとほとんど<sup>1</sup>のセマンティック要素(Model カテゴリの要素)が HTML として適切に変換できるようになり[1]、また、`html.elem` を使用して任意の HTML タグを Typst から生成できるようになりました。これにより Typst は HTML の DOM ツリーを直接操作できるようになりました。

そんなわけで実際に Typst だけで記述可能なブログシステムを作ってみました。SSG(Static Site Generator)のようなものを構築したのですが、記事の記述等々を Typst 内部だけで完結させた点が面白かったのでその裏側を紹介します。

なお、スクリプトは WTFPL で公開しているものの、個人用に作ったものであるため他人が使うことを想定していません。現時点でもここまでできるんだよということを紹介するためのものとして受け取ってもらいたいです。

## 2. ファイル構成

ファイル構成はコード 1 のようになっています。

トップレベルには 5 つのファイルがあります。 `index.typ` ではこのサイト(<https://bibouroku.minimarimo3.jp>)にアクセスしたとき最初に表示されるページを、 `style.css` ではサイト全体のテーマを、 `posts.typ` では記事のメタデータ（公開日、更新日、概要など）の管理を、 `template.typ` では記事のテンプレートを、 `build.py` では `posts.typ` のメタデータをもとにディレクトリを走査し、ビルドや添付ファイルの移動を担当しています。

`public` ディレクトリはビルド後の出力先です。Web サイトとして公開するのはこのディレクトリになります。

その他ディレクトリは記事のディレクトリです。このディレクトリに記事を書き、 `posts.typ` にその記事のメタデータ（公開日、更新日、概要など）を記述することで `index.typ` と `build.py` が記事の存在を認識しビルドされます。

---

<sup>1</sup>例えばカスタム HTML 内での footnote などはまだ使えません

記事を記述するファイル名が `index.typ` になっていますが、これは Web サイトの慣習によるものです。 `index.typ` をビルドすると通常 `index.html` ができます。この `index.html` という名前は特別なもので、URL にアクセスするファイル名が指定されていないときにデフォルトでアクセスするファイルです。

#### BIBOUROKU.MINIMARIM03.JP

```
| index.typ      # トップページ
| style.css     # サイト全体のテーマを設定するファイル
| posts.typ     # 公開対象の記事のメタデータを記述するファイル
| template.typ  # 記事のテンプレート
| build.py      # posts.typのデータをもとにディレクトリを走査し、ビルドと添付ファイル
の移動を行うスクリプト
|
| └─public      # ビルド後の出力先
|   | index.html
|   | style.css
|   | feed.xml  # build.pyによって生成されます
|   | sitemap.xml # build.pyによって生成されます
|   |
|   └─Typstでブログを書く
|       | index.html
|       | index.pdf
|       |
|       └─テスト
|           | index.html
|           | index.pdf
|           | テスト用画像.png
|           |
|   └─Typstでブログを書く # 記事1
|       | index.pdf
|       | index.typ
|       | Typstでブログを書く.yaml
|       |
|   └─テスト          # 記事2
|       | index.pdf
|       | index.typ
|       | reference.bib
|       | テスト用画像.png
|       |
|   └─.github
|       └─workflows
|           └─deploy.yml
```

コード 1: 当ブログのファイル構成

## 3. 実装

### 3.1. `html.html` で出力される HTML の構造をカスタマイズ

`html.html` や `html.head` を使用せずに HTML エクスポートすると、文書の内容はすべて `body` タグ内に記述されることになります。これは `html.style` や `html.meta` であっても `body` タグ内に記述されるということです。 `head` に記述したい内容がある

場合は `html.html` を使用してエクスポートする HTML の構造を一から記述する必要があります。

今回のブログでは、`template.typ` 内に以下のような構造を定義しました。

```
html.html(lang: "ja", {
  html.head({
    html.meta(charset: "utf-8")
    html.meta(name: "viewport", content: "width=device-width, initial-scale=1")
    html.title(title)

    // OGP設定やGoogle Fontsの読み込み
    if description != "" {
      html.meta(name: "description", content: description)
    }
    html.elem("meta", attrs: (property: "og:title", content: title))
    html.link(rel: "stylesheet", href: "/style.css")
    html.link(rel: "preconnect", href: "https://fonts.googleapis.com")
    html.link(rel: "preconnect", href: "https://fonts.gstatic.com",
crossorigin: "anonymous")

    html.script(src: "/script.js")
    html.link(rel: "stylesheet", href: "/style.css")
  })

  html.body({
    html.div(class: "site-container", {
      // ヘッダー、記事本文、サイドバーなどを自由に配置
      html.main(class: "main-content", body)
      html.aside(class: "sidebar", { ... })
    })
  })
})
```

### 3.2. 関連記事のランダム生成

外部スクリプトで計算した結果を渡すのではなく、Typst 内部で乱数を生成して記事を選んでいきます。ただし、ビルドのたびに関連記事が変わるのは困るので記事タイトルのハッシュ地をシードにして乱数を固定しています。

```
// template.typより抜粋
#import "@preview/suiji:0.5.0": *

// 記事タイトルを数値化してシードにする
let seed = int(title.clusters().map(str.to-unicode).map(str).join().slice(0,
14))
let rng = gen-rng(seed)

// 記事リストをシャッフル
let (_, indices) = shuffle-f(rng, range(other-posts.len()))
```

```
// 上位3件を取得
let picks = indices.slice(0, 3).map(i => other-posts.at(i))
```

### 3.3. 記事の情報を Typst で管理する

ブログのトップページや RSS フィードを作るには全記事のリストとそのデータ（更新日等）が必要です。このために各記事ファイルのメタデータをまとめたファイル (posts.typ) で管理する仕組みを採用しました。これにより Typst ファイルからは下のような形で、

```
#import "../template.typ": project
#import "../posts.typ": post-data
#let meta = post-data.at("Typstでブログを書く")
#show: project.with(..meta)
```

ビルドスクリプト(Python)からは下のような形で同じ情報を取得することができます。

```
result = subprocess.run(
    ["typst", "query", "posts.typ", "<post-list>"],
    capture_output=True,
    text=True,
    check=True,
    encoding="utf-8"
)
data = json.loads(result.stdout)
```

### 3.4. 未実装機能への対処

#### 3.4.1. 数式(Math)を SVG 化して埋め込む

現状、数式の HTML エクスポートは未実装（または不完全）です。そこで、数式を html.frame で一度フレーム（画像的な扱い）にし、SVG として HTML 内に展開することで表示させました。

```
show math.equation.where(block: false): it => {
    html.elem("span", attrs: (role: "math"), html.frame(it))
}
show math.equation.where(block: true): it => {
    html.elem("figure", attrs: (role: "math"), html.frame(it))
}
```

#### 3.4.2. カスタム HTML 構造での注釈(Footnote)

html.html を使ってカスタム構造を作ると、標準の footnote がエラーになります。

```
error: footnotes are not currently supported in combination with a custom
`<html>` or `<body>` element
```

```
  \\\G:\マイドライブ\bibouroku.minimarimo3.jp\テスト\index.typ:97:16
```

```
97 |   これがノートをつけられる対象1#footnote[footnoteの中身1]
```

```
                                     ~~~~~
```

```
= hint: you can still use footnotes with a custom footnote show rule
```

幸い counter は使えるので自分で実装することで対処することができます。

```
let note-counter = counter("my-footnote")
show footnote: it => {
  note-counter.step()
  let num = note-counter.get().first()
  // CSSでツールチップ表示するためのHTML構造を出力
  html.span(class: "footnote-wrapper", {
    html.span(class: "footnote-marker", "※" + str(num))
    html.span(class: "footnote-content", it.body)
  })
}
```

## 4. まとめ

Typst での HTML 生成はまだ実験的な機能ですが、個人のブログや小規模なドキュメントサイトなら十分実用できるレベルにあると感じました。何より、慣れ親しんだ Typst 記法で記事が書けて、それがそのまま Web サイトになるのは非常に快適です。

## 参考文献

- [1] L. Mädje と M. Haug, 「Typst: Typst 0.14: Now accessible – Typst Blog」, Typst. 参照: 2025 年 12 月 19 日. [Online]. 入手先: [https://typst.app/blog/2025/typst-0.14#richer-html-export:~:text=Most%20semantic%20elements%20\(those%20from%20the%20Model%20category\)%20are%20now%20properly%20mapped%20to%20semantic%20HTML.%20We%27ve%20also%20improved%20handling%20of%20textual%20content%20in%20HTML%20export.](https://typst.app/blog/2025/typst-0.14#richer-html-export:~:text=Most%20semantic%20elements%20(those%20from%20the%20Model%20category)%20are%20now%20properly%20mapped%20to%20semantic%20HTML.%20We%27ve%20also%20improved%20handling%20of%20textual%20content%20in%20HTML%20export.)