Next.js を使った静的ブログへ移行を検討した話

現在の個人ブログ（Nuxt.js＋AMP）をNext.jsを使った静的ブログへの移行を試した話です。

2022.09.01

この参考文献をもとにすすめました

ただし、JavaScriptだったのでTypeScriptの勉強のために、jsxやjsをtsx、tsへ書き換えながら進めました

https://reffect.co.jp/react/nextjs-markdown-blog

目次(1/2)

ハマったところ

2022.09.01

TypeScript で Next.js をつくるとき

```
npx create-next-app kght6123.page --ts
```

`--ts` を忘れると面倒、結局、作り直しました。

2022.09.01

TypeScript の型ガード

`gray-matter` はMarkdownのメタ情報の部分を取り込むライブラリです。
ここにブログの基本情報を書きたい。

```
---
title: "Next.jsでmarkdownブログを構築"
date: "2022-07-13"
description: "Next.jsでmarkdownファイルを利用したブログの構築手順を解説しています。"
image: nextjs.png
categories: ['react']
---

## 目次
```

`gray-matter`からは下記のような型で返ってくる。

```
const data: {
    [key: string]: any;
}
```

anyを使うときに毎回、型判定が必要で面倒なので、型ガードで
stringまたはstring[]にして、またオブジェクトに戻す。

最初につくったのがこれ

```
const frontMatter = Object.entries(data).filter(
  (entry): entry is [string, string] => typeof entry[1] === "string"
  ).map(entry => {
    return {[entry[0]]: entry[1]}
  });
return {
  frontMatter,
  slug,
};
```

型定義のあった方が明確に型がわかるかな・・・？

結果、dataをtypeとして定義したFrontMatterにするユーティリティを作りました

```
export type FrontMatter = {
  [key: string]: string | string[];
};
export const convertFrontMatter = (data: { [key: string]: unknown }) => {
  const frontMatter = Object.entries(data)
    .filter((entry): entry is [string, string|string[]] => typeof entry[1] === "string" || typeof entry[1] === "object")
    .reduce((p, cv) => {
      p[cv[0]] = cv[1];
      return p;
    }, {} as FrontMatter);
  return frontMatter;
};
```

工夫したところ

TypeScriptの型のインポート

2022.09.01

最初に作ったコード、propsの型を独自で定義していた

```
const CustomLink = ({
  children,
  href,
}: {
  children: string;
  href: string;
}): JSX.Element =>
  href.startsWith('/') || href === '' ? (
    <Link href={href}>
      <a>{children}</a>
    </Link>
  ) : (
    <a href={href} target="_blank" rel="noopener noreferrer">
      {children}
    </a>
  );
```

```tsx
const CustomImage = ({
  src,
  alt,
}: {
  src: string;
  alt?: string | undefined;
}): JSX.Element =>
  <Image src={src} alt={alt} width="1200" height="700" />;
```

毎回、ImageとかLinkカスタムタグのpropsの型を定義していくのはめんどい

元々用意されている、ImagePropsやLinkPropsの型を使うように

```
const CustomImage = ({
  src,
  alt,
}: ImageProps): JSX.Element =>
  <Image src={src} alt={alt} width="1200" height="700" />;
```

LinkのPropsは交差型を使うように

```
const CustomLink = ({
  children,
  href,
}: LinkProps & {
  children?: React.ReactNode; // next/linkのコードを見ると交差型で定義されているのでここも同じように
}): JSX.Element =>
  href.toString().startsWith('/') || href === '' ? (
    <Link href={href}>
      <a>{children}</a>
    </Link>
  ) : (
    <a href={href.toString()} target="_blank" rel="noopener noreferrer">
      {children}
    </a>
  );
```

ImagePropsやLinkPropsの型を使うことで、コードはスッキリ

↓

“ 明示的にどのPropsの属性をコンポーネントでつかってるか、コードを見ないとわからない ”

↓

業務での開発は元の方が良いと感じました。

Prettier でフォーマットしたい

```
npx prettier --write . --ignore-path \".gitignore\" #手軽に最新でフォーマット（npmのscriptsにいれる）
```

個人開発なので、まぁ常に最新でよいかなと。

気になったところ

2022.09.01

next.config.js

↓ js なのに型補完が IDE で効く

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true,
  swcMinify: true,
};

module.exports = nextConfig;
```

tailwind.config.js（`npx tailwindcss init -p`で生成される）

↓ jsなのに型補完がIDEで効く

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [],
  theme: {
    extend: {},
  },
  plugins: [],
};
```

jsですが、@typeでIDEに型を教えてあげている。

@typeが必要なのは、、、

Tailwind CSSの場合は、設定ファイルがTypeScriptに未対応のライブラリ（PostCSS）があるためtsではないようです。

使える場面では、効率が上がるので使っていきたいなと思いました。

※ Node.jsはtsをトランスパイルしないと使えないのも原因のひとつ？

まとめ

すんなり、Next.jsでブログが移行できそうな状態になりました。

- あとはデザインを整えて、
- Nuxt.jsのブログとURLをあわせたり
- gray-matterの属性をあわせたり

ただ、最新技術的にはあまり面白くなかったので、Solid.jsとかで試したいなと思いました。

ありがとうございました！！

2022.09.01