データフェッチハンズオン 【Next.js App Router】

2025/10/17 平野大介

Next.js App Router

- React Server Component や Server Action といった機能をサポート
- データの READ はサーバーコンポーネントで行う。
- データの CREATE,UPDATE,DELETE は Server Action で行う。

作ったもの

- 一覧画面
  - 検索,ソート,ページネーション
  - 削除
- 作成画面
- 編集画面

レポジトリ：https://github.com/nebular-lab/word-book
デプロイ：https://word-book-two.vercel.app

一覧画面(と時間があれば作成画面)について説明します。

一覧画面の方針

- 検索・ソート・ページネーションの状態はクエリパラメータで管理する。
  - 例：/?q=test&order=author&offset=25
  - クエリパラメータを型安全に使うために、nuqs を使う
- 検索・ソート・ページネーションをする場合、そのクエリパラメータの URL へ遷移する。
- サーバーコンポーネントでクエリパラメータを読み取ってデータフェッチする。
- MicroCMS の API の取得結果はキャッシュする。(Data Cache を活用)
- ローディング中はテーブルの代わりにスピナーを表示する。

```tsx
//app/page.tsx
import { queryParamsCache } from "@/lib/query-params-cache";

export default async function Page({ searchParams }: PageProps<"/">) {
  // Point: nuqsの機能であるqueryParamsCacheにクエリパラメータを保存。
  // サーバーコンポーネントから参照できるようになる。
  const { q, orders, offset } = await queryParamsCache.parse(searchParams);

  // Point: ここではデータフェッチをせず、Suspenseで囲ったコンポーネント内でデータフェッチを行う。
  // データフェッチ中にSuspenseの外側のコンポーネントを先に描画できる。

  return (
    <main className="mx-auto flex h-full w-full max-w-5xl flex-col gap-6 px-6 py-12">
      <div className="flex justify-between">
        <H1>英単語帳一覧</H1>
        <div className="flex gap-4">
          <SearchForm />
          <ToCreateLinkButton />
        </div>
      </div>
      {/* Point: クエリパラメータが変わったときにfallbackを表示するためには、keyを指定する必要がある。*/}
      <Suspense key={`${q}${orders}${offset}`} fallback={<PageCenterSpinner />}>
        <WordsTableContainer />
      </Suspense>
    </main>
  );
}
```

```tsx
//app/(home)/components/words-table-container.tsx
import { listWord } from "@/api/words";
import { nullToUndefined } from "@/lib/nullToUndefined";
import { queryParamsCache } from "@/lib/query-params-cache";

export const WordsTableContainer = async () => {
  // Point: queryParamsCacheからクエリパラメータを取得
  const cachedQueryParams = queryParamsCache.all();
  const queryParams = nullToUndefined(cachedQueryParams);

  const { q, orders, offset } = queryParams;

  // データフェッチ
  const { contents, totalCount } = await listWord({
    q,
    offset,
    orders,
  });

  // 以下省略。取得したデータを表示する。
  return <></>;
};
```

```tsx
// src/api/words.ts
import { unstable_cache } from "next/cache";

export async function uncachedListWord(queries: {
  q?: string;
  offset?: number;
  orders?: string;
}) {
  return microcmsClient.getList<Word>({
    endpoint: "words",
    queries: {
      limit: PAGE_SIZE,
      ...queries,
    },
  });
}

// Point: fetchAPIを使わない場合は、unstable_cacheを使ってキャッシュする。
// Memo: 将来的に、"use cache"ディレクティブに置き換えられる予定。
export const listWord = unstable_cache(uncachedListWord);
```

```tsx
// src/app/(home)/components/search-form.tsx
// Point: ユーザー操作が必要のため、クライアントコンポーネントにする。
"use client";

import { parseAsString, useQueryStates } from "nuqs";
import { Input } from "@/components/ui/input";

export function SearchForm() {
  // Point: nuqsを使うことで、ReactのuseStateのようにクエリパラメータを扱える。
  const [{ q }, setQueryParams] = useQueryStates({ q: parseAsString, offset: parseAsString }, { shallow: false });

  const handleChange = (value: string) => {
    setQueryParams((previous) => ({
      ...previous,
      q: value ? value : null,
      offset: null,
    }));
  };
  return (
    <Input type="text" placeholder="Search..." value={q ?? ""} onChange={(event) => handleChange(event.target.value)} />
  );
}
```

作成画面の方針

- サーバーアクションから MicroCMS の作成 API を叩く
- サーバーアクションでのバリデーションエラーをフォームに表示するために、Conformを使う。
- 作成に成功したら一覧画面のデータキャッシュを削除する

```tsx
"use client";
import { useForm } from "@conform-to/react/future";
import Form from "next/form";
import { useActionState } from "react";
import { createWord } from "@/api/words"; // Point: サーバーアクション
import { createWordSchema } from "@/lib/schema/words";

export function CreateWordForm() {
  // Point: useActionStateを使うことで、サーバーアクションの状態を取得できる。
  //   lastResult サーバーアクションの戻り値。失敗時はエラー情報を返すようにしている。
  //   isPending サーバーアクションが実行中かどうか。
  const [lastResult, action, isPending] = useActionState(createWord, null);

  // Point: Conformライブラリのuseformを使うことで、サーバーアクションのバリデーション結果を
  // フォームに反映することができる。
  // react-hook-formにはこのような機能はない。
  const { form, fields } = useForm({
    lastResult,
    shouldRevalidate: "onInput",
    // Point: クライアントでバリデーションを行える。
    schema: createWordSchema,
  });

  return (
    // formとactionを渡す
    <Form {...form.props} action={action}>
      省略
    </Form>
  );
}
```

```tsx
import { parseSubmission, report } from "@conform-to/react/future";
import { revalidatePath } from "next/cache";
import { redirect } from "next/navigation";

export async function createWord(_: unknown, formData: FormData) {
  // フォームデータをバリデーションして、エラーであればエラー内容をreturnする。
  const submission = parseSubmission(formData);
  const result = createWordSchema.safeParse(submission.payload);
  if (!result.success) {
    return report(submission, {
      error: {
        issues: result.error.issues,
      },
    });
  }
  try {
    await microcmsClient.create({
      endpoint: "words",
      content: result.data,
    });
  } catch (error) {
    // Point: エラーをthrowせずに、エラー内容をreturnする。
    // サーバーアクション内でthrowされると、ErrorBoundaryが表示されてしまう。
    // フォームを表示したままエラーメッセージを表示するためには、エラー内容を戻り値として返す必要がある。
    if (error instanceof Error && error.message.includes("Please input unique value on 'title' field.")) {
      return report(submission, {
        error: {
          // Point: "title"のバリデーションエラーとしてフロントに返すことができる。
          fieldErrors: { title: ["すでに同じ英語が登録されています。"] },
        },
      });
    }
    return report(submission, {
      error: {
        formErrors: ["予期せぬ理由で作成に失敗しました。もう一度お試しください。"],
      },
    });
  }

  // Point: 一覧ページのキャッシュを削除する。
  revalidatePath("/");

  // 一覧ページにリダイレクト
  redirect("/");
}
```

感想

良かった点

- useState が一度も登場せず、とてもシンプルに書けた。
- nuqs と conform が使いやすい。

良くなかった点

- Router Cache と Data Cache の使い分け方がよくわからなかった。
- クエリパラメータの組み合わせごとに Data Cache していたら、キャッシュがたまり過ぎてしまって良くないことが起こりそう。
- Link コンポーネント による prefetch が並列で走るとパフォーマンスが悪くなる。

→ Dynamic IO (キャッシュをシンプルにするための機能)に期待