App Routerの紹介

Press Space for next page →

自己紹介

- 📝 飯野陽平（wheatandcat）
- 🏢 法人設立（合同会社UNICORN 代表社員）
- 💻 Work: シェアフル株式会社CTO
- 📚 Blog: https://www.wheatandcat.me/
- 🛠️ 今までに作ったもの
  - memoir
  - ペペロミア
  - MarkyLinky

App Routerとは？

- Next.js 13.4で追加された新しいルーティング機構
- 従来のNext.jsのルーティングは**Page Router**
- **RSC**（React Server Components）の使用を前提と設計

RSC（React Server Components） とは？

- Reactでサーバーサイドでレンダリングできるコンポーネントを宣言できる仕組み
- 元々、Reactにはクライアントサイドでレンダリングする**クライアントコンポーネント**しかなかった
- SSRは、**クライアントコンポーネントをサーバーサイドでレンダリングする仕組み**
- RSCを使用することでサーバーサイド固有の機能を実装できる
  - 例えばReact内で直接データベースにアクセスすることなどが可能
- 逆にRSCではクライアントサイド固有の機能は使用できない
  - 例えば、useStateやuseEffectなどのHooksは使用できない
  - 宣言した時点でエラーになる
- RSCで宣言されたコンポーネントはブラウザ側で読み込みbundleされたJavaScriptには含まれないため、読み込むファイルサイズは小さくなる

App Routerのルーティング基本機構

- 従来のPage Routerのルーティング機構

```
src
 └ pages
    ├ index.tsx → /
    └ items
        ├ [id].tsx → /items/[id]
        └ [id]
            └ share.tsx → /items/[id]/share
```

- App Routerのルーティング機構

```
src
 └ app
    ├ page.tsx → /
    └ items
        └ [id]
            ├ page.tsx → /items/[id]
            └ share
                └ page.tsx → /items/[id]/share
```

ルーティング毎に動的にfavicon、OG画像を変更する

- App Routerのルーティング機構だと以下のように設定が可能

```
src
└ app
    ├ layout.tsx  → 共通設定
    └ items
        └ [id]
            ├ icon.tsx → 動的faviconを変更
            └ opengraph-image.tsx  → 動的OG画像を変更
```

- 以下、書き方の参考
    - src/app/icon.tsx

データアクセス & レンダリングの構造①

App Routerはデフォルトは**RSC**でレンダリングされる、以下、コードの例

```
export default function Home() {
  return (
    <div className="container">
      <h1 className="text-5xl">
        Sample App
      </h1>
      <CrudShowcase />
    </div>
  );
}

async function CrudShowcase() {
  const session = await getServerAuthSession();
  if (session?.user) {
    const url = await api.url.existsByUserId.query({userId: String(session?.user.id)});
    if (url) redirect(`/schedule/${String(url?.id)}`);
  }
  return <CreateUrl />;
}
```

データアクセス & レンダリングの構造①

App Routerはデフォルトは**RSC**でレンダリングされる、以下、コードの例

```
export default function Home() {
  return (
    <div className="container">
      <h1 className="text-5xl">
        Sample App
      </h1>
      <CrudShowcase />
    </div>
  );
}

async function CrudShowcase() {
  const session = await getServerAuthSession();
  if (session?.user) {
    const url = await api.url.existsByUserId.query({userId: String(session?.user.id)});
    if (url) redirect(`/schedule/${String(url?.id)}`);
  }
  return <CreateUrl />;
}
```

データアクセス & レンダリングの構造①

App Routerはデフォルトは**RSC**でレンダリングされる、以下、コードの例

```
export default function Home() {
  return (
    <div className="container">
      <h1 className="text-5xl">
        Sample App
      </h1>
      <CrudShowcase />
    </div>
  );
}

async function CrudShowcase() {
  const session = await getServerAuthSession();
  if (session?.user) {
    const url = await api.url.existsByUserId.query({userId: String(session?.user.id)});
    if (url) redirect(`/schedule/${String(url?.id)}`);
  }
  return <CreateUrl />;
}
```

データアクセス & レンダリングの構造①

App Routerはデフォルトは**RSC**でレンダリングされる、以下、コードの例

```
export default function Home() {
  return (
    <div className="container">
      <h1 className="text-5xl">
        Sample App
      </h1>
      <CrudShowcase />
    </div>
  );
}

async function CrudShowcase() {
  const session = await getServerAuthSession();
  if (session?.user) {
    const url = await api.url.existsByUserId.query({userId: String(session?.user.id)});
    if (url) redirect(`/schedule/${String(url?.id)}`);
  }
  return <CreateUrl />;
}
```

データアクセス & レンダリングの構造②

```
"use client";

export function CreateUrl() {
  const router = useRouter();
  const createMutation = api.url.create.useMutation({
    onSuccess: (data) => {
      router.push(`/items/${data.id}`);
    },
  });

  const onCreate = useCallback(() => {
    createMutation.mutate();
  }, [createMutation]);

  return (
    <button
      className="button"
      onClick={onCreate}
    >
      新しいデータを作る
    </button>
  );
}
```

データアクセス & レンダリングの構造②

```
"use client";

export function CreateUrl() {
  const router = useRouter();
  const createMutation = api.url.create.useMutation({
    onSuccess: (data) => {
      router.push(`/items/${data.id}`);
    },
  });

  const onCreate = useCallback(() => {
    createMutation.mutate();
  }, [createMutation]);

  return (
    <button
      className="button"
      onClick={onCreate}
    >
      新しいデータを作る
    </button>
  );
}
```

データアクセス & レンダリングの構造②

```tsx
"use client";

export function CreateUrl() {
  const router = useRouter();
  const createMutation = api.url.create.useMutation({
    onSuccess: (data) => {
      router.push(`/items/${data.id}`);
    },
  });

  const onCreate = useCallback(() => {
    createMutation.mutate();
  }, [createMutation]);

  return (
    <button
      className="button"
      onClick={onCreate}
    >
      新しいデータを作る
    </button>
  );
}
```

データアクセス & レンダリングの構造③

- **RSC**ではasync/awaitが使用できる
- RSCではhooksは使用できないのでデータ更新で使用する**useMutation**は**クライアントコンポーネントで実装する必要がある**
- RSCでデータ取得する場合は、直接データベースにアクセスが可能なので以下のように実装が可能

```
export default async function Page({ params }: { params: { id: string } }) {
  const session = await getServerAuthSession();

  const url = await api.url.exists.query({ id: params.id });
  if (url === null) {
    // 存在しないURLの場合はトップページに戻す
    redirect("/");
  }

  ... 省略
```

データアクセス & レンダリングの構造③

- **RSC**ではasync/awaitが使用できる
- RSCではhooksは使用できないのでデータ更新で使用する**useMutation**は**クライアントコンポーネントで実装する必要がある**
- RSCでデータ取得する場合は、直接データベースにアクセスが可能なので以下のように実装が可能

```
export default async function Page({ params }: { params: { id: string } }) {
  const session = await getServerAuthSession();

  const url = await api.url.exists.query({ id: params.id });
  if (url === null) {
    // 存在しないURLの場合はトップページに戻す
    redirect("/");
  }

  ... 省略
```

データアクセス & レンダリングの構造③

- **RSC**ではasync/awaitが使用できる
- RSCではhooksは使用できないのでデータ更新で使用する**useMutation**は**クライアントコンポーネントで実装する必要がある**
- RSCでデータ取得する場合は、直接データベースにアクセスが可能なので以下のように実装が可能

```
export default async function Page({ params }: { params: { id: string } }) {
  const session = await getServerAuthSession();

  const url = await api.url.exists.query({ id: params.id });
  if (url === null) {
    // 存在しないURLの場合はトップページに戻す
    redirect("/");
  }

  ... 省略
```

実装してみる

- PR: App Routerに変更する
- 以下、デモしながら説明
  - App Routerの移行で以下の処理はシンプルに実装できた例1
    - 旧コード
    - 新コード
  - App Routerの移行で以下の処理はシンプルに実装できた例2
    - 旧コード
    - 新コード

まとめ

- App Routerは**RSC**を前提としたルーティング機構になっている
- **RSC**を使用するとデータ取得周りはシンプルに実装できる
- ただ、App Routerはフロントエンド界隈では賛否両論で、当面Web文脈では議論されていきそう
  - 一休レストランで Next.js App Router から Remix に乗り換えた話
  - 元々Next.jsはWeb標準のAPIに準拠していないことが問題視されていて、App Routerでより、その点が強まった
  - 逆にRemixはWeb標準のAPIに準拠した思想で実装されているので対象的になっている

ご清聴ありがとうございました🎉