全4回で学ぶNext.js入門講座
2回目

2025/11/11

Kazuma SEKIGUCHI

前回のアジェンダ

- Next.jsの概要
- プロジェクト作成
- JSXの基礎、コンポーネント作成、props
- ページ作成とルーティング
- フォームの基本
- useStateで入力管理

今回のアジェンダ

- CSS Modules または Tailwind でのスタイル
- レイアウト（layout.js）の紹介と共通デザイン
- ルーティングの拡張
- 動的ルーティング
- use client の利用

CSSの扱い

- CSSファイルを作成し、各ページのフォルダー内に保存しておくことが可能
  - 各ページのpage.jsの先頭でCSSファイルを読み込むことでCSSを利用することが可能

```
import './style.css'
```

  - CSSファイルに記載のないものは、global.cssが利用される
  - CSSファイルでglobal.cssは上書きすることが可能

CSS Moduleを利用してスタイル適用

- ○○.module.cssという名前で作成し保存したCSSファイルのスタイルをJSのオブジェクトとして利用可能にしたもの

```
import styles from './style.module.css'
//省略
<h1 className={styles.title}>Other Page</h1>
```

- 各コンポーネントに対してスタイルをローカルスコープで適用する
- Next.jsはCSS Modulesをネイティブでサポートしており、設定なしで簡単に利用できる
- SSR（サーバーサイドレンダリング）にも対応しているため、CSS Modulesを使ってもページの初期表示が高速に行われ、SEOにも有利

Styled JSX

- コンポーネント内にはstyleタグを記述できないため、代替手段として提供されている機能
  - 各コンポーネント内にスタイルを記述でき、スタイルとコンポーネントが一体化する
  - Next.jsに組み込まれており、追加の設定やパッケージなしで利用可能
  - ページの初期表示が高速でSEOにも優れている
  - JSXの中でスタイルを直接記述できるため、スタイルとロジックをコンポーネント内にまとめられる

Styled JSX

- importした上で、タグ内にimport名のタグを作成してCSSを記述する
- 先頭に"use client";が必要

```
"use client";
import JSXStyle from 'styled-jsx/style'
//省略
return(
<main>
<JSXStyle>
{`.jsx-msg{
margin:10px;
text-align:center;
color:#f90;
}`)
</JSXStyle>
<p className="jsx-msg">ここに適用される</p>
```

Tailwind CSS

- 今回はCSSでTailwind CSSを使えるようにしているので、Tailwind CSSで使えるクラスは全部指定することが可能
  - https://tailwindcss.com/docs/installation にクラス名が載っているが、数が多い
- 複数クラスも当然可能なので、組み合わせて見せたいスタイルにすればOK

tailwindcss

```
<h1 class="text-2xl font-bold text-center text-blue-600">
  Hello Tailwind!
</h1>
```

layout.js

- すべてのページで共通して使う枠組み（レイアウト）を定義するファイル
- 例：ヘッダー・フッター・ナビゲーションなど
- ページごとに同じ要素を毎回書かなくてよくなる
- app ディレクトリのルートにlayout.jsを配置する
- 各フォルダー内にもlayout.jsを配置することが可能
  - この場合は、app/layout.jsは使われない
  - それ以下のフォルダーに継承される

layout.js

- 共通している部分のレイアウトをapp/layout.jsに記述することで、全ページに適用することが可能になる

```
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body
        className={`${geistSans.variable} ${geistMono.variable} antialiased`}
      >
        {children}
      </body>
    </html>
  );
}
```

この中身を書き換える

layout.js

- {children}内に他のpage.jsの内容が入ってくる

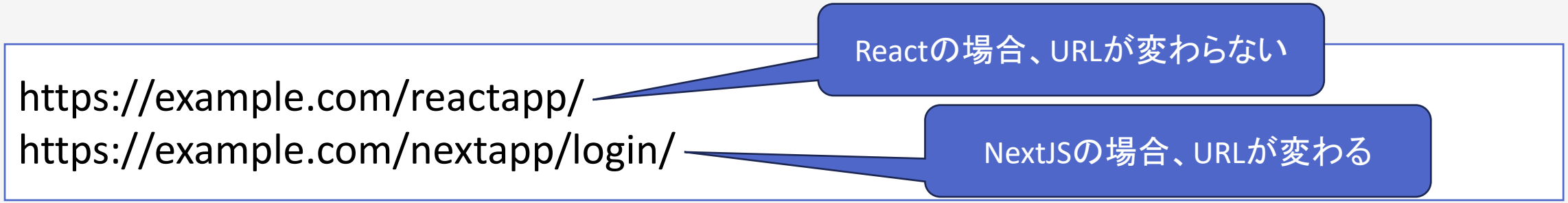
```
export default function RootLayout({ children }) {
  return (
    <html lang="ja">
      <body
        className={`${geistSans.variable} ${geistMono.variable} antialiased`}
      >
        <header>ヘッダー内容</header>
        {children}
        <footer>フッター内容</footer>
      </body>
    </html>
  );
}
```

Appルーティング

- Reactではページを表示している間機能する
  - 別のページに移動すると情報は全て失われる
  - 1ページで完結するようなコンテンツの作成に向いている（SinglePageApplication）
- Webアプリの場合、1ページで完結できる方が珍しい
  - 複数のページを用意し、必要に応じて移動したりして機能を実現する
  - 1ページで作成すると、非常に肥大化して保守性が悪い

Appルーティング

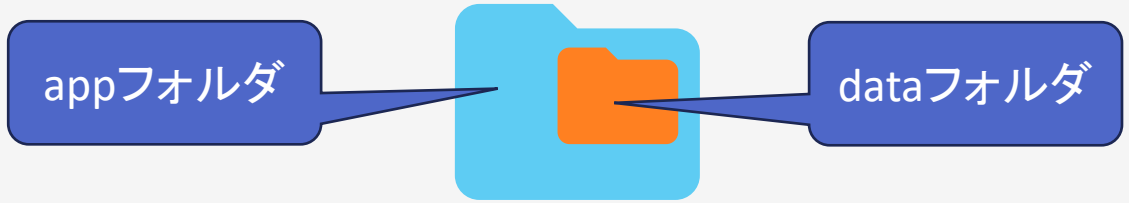
- Next.jsの場合、複数のページのアプリケーションを作成することが可能
  - それぞれのページに対して決まったURLを振ってアクセスすることが可能



  - URLに応じて、どの処理を行なうのか制御する仕組み＝ルーティング
  - Next.jsにはいくつかのルーティングの仕組みが搭載されている

Appルーティング

- Next.jsをインストールするときにappルーティングを使うか、という選択肢が出てくるが、使うとしたときに利用されるのがAppルーティング
  - ファイルシステムをベースとしたルーティング
  - appフォルダー内に別のフォルダーを作成し、そこにpage.jsファイルを作成して、コンポーネントを記述
    - フォルダーにアクセスすると動作するようになっている



https://example.com/data/
でアクセス可能になる

Appルーティング

- ページ間のリンクには、Linkコンポーネントを利用する
  - aタグでもリンクは貼れるが、ページ遷移をしてしまうため、Linkコンポーネントで擬似的なリンクとして機能させる
- Linkコンポーネント自体はaタグとほぼ同じように利用可能

```
import Link from 'next/link';
//省略
<div>
      <Link href="/">戻る</Link>
</div>
```

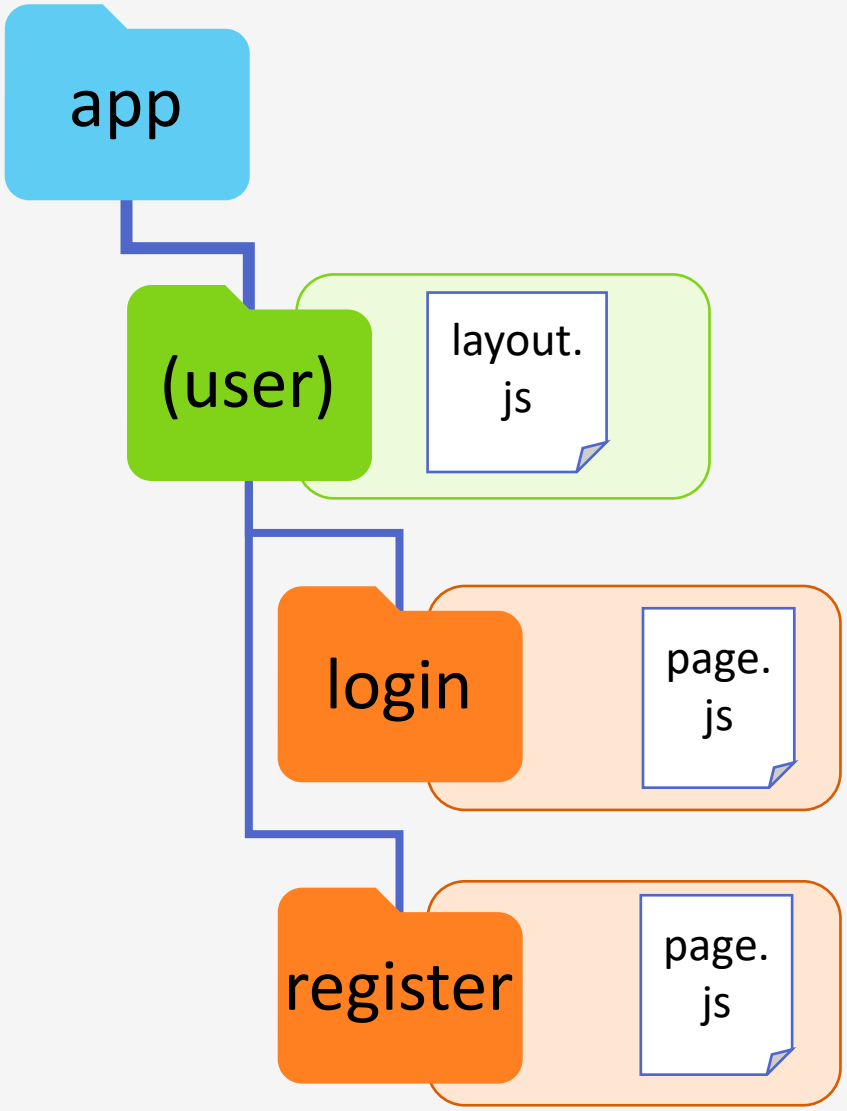
publicフォルダーの利用

- publicフォルダー内には画像など共通で使うものを格納しておく
  - srcと同じ階層にpublicフォルダーを作成する
  - 画像を表示するときは、Imageコンポーネントを利用する
  - 利用方法はほぼimgタグと同じ
    - Imageコンポーネントを使うことで、WebP形式の画像として表示される
  - サイズなどは{}で括っておく

```
import Image from 'next/image'
//省略
<p><Image src="/autumn.jpg" width={1000} height={664} /></p>
```

ルーティングの拡張

- (group)フォルダーでURLに含めない構成が可能
  - (名前)というフォルダーを作成し、内部にフォルダーを作成すると名前はURLに含まれないが、配下にフォルダーを配置することが可能
  - (名前)というフォルダー内にlayout.jsを置くことで、配下のフォルダーに同じレイアウトを適用することが可能



ダイナミックルーティング

- データから特定の項目だけ取り出したいときにパラメータをURLに付与するケースがある

https://example.com/item/125

125は状況で変化する

  - ダイナミックルーティング機能を使うことで、パラメータ部分を上手く処理することが可能
- 指定の形式でフォルダーを作成することで対応できる
  - 上記の場合、itemというフォルダーを作成し、中に[item]という名前のフォルダーを作成する

ダイナミックルーティング

- 複数のパラメータも受入れることが可能
  - [category]フォルダーを作成し、中に[id]フォルダーを作成する
    - フォルダー名は他の名前でも問題無い
  - category部分も動的に取得でき、idの値も取得することが可能

https://example.com/book/125

125は状況で変化する

ダイナミックルーティングの受け取り

```
export default function Item({ params }) {
//省略
{params.item}
```

- {params}として受入れる引数を用意し、格納をして貰う
  - itemというフォルダーにpage.jsを置いたので、パスの一部をitemという名前で取得することが可能
  - {params.item}という名前で送られてきたパラメータを受け取ることができる
- [item]内に更に別のフォルダーを作成すれば複数のパラメータを受け取ることが可能

ダイナミックルーティングのパラメータ省略

- パラメータを省略してアクセスするとエラーになる
  - itemフォルダー内にpage.jsが存在しないため
  - パラメータが必要なのに記入されずにアクセスされたときのためにエラー画面を置いておくべき

use client

- 利用しているApp RouterではNext.jsだとデフォルトはサーバーで実行される（サーバーコンポーネント）
- Next.jsはサーバーコンポーネントとクライアントコンポーネントの2種類で構成されている
- イベント処理や状態管理（useStateやuseEffect）などはクライアント側で動作させる必要がある
  - 明示的にクライアント側で動作させることを区別する必要がある

```
use client;
```

ファイルの先頭に記述

- サーバーで実行できるものはサーバーで実行する、というNext.jsの指向による

組み合わせて利用可能

- サーバーとクライアントを分けて組み合わせる
- サーバーコンポーネントの中にクライアントコンポーネントを組み込める
- これにより「初期表示はサーバーで高速」「動的操作はクライアントで柔軟」に実現できる

サーバーとクライアントの分離

```
// app/page.js（サーバーコンポーネント）
import Counter from "./Counter";

export default function Home() {
  return (
    <main>
      <h1>トップページ</h1>
      <Counter />
    </main>
  );
}
```

```
// app/Counter.js（クライアントコンポーネント）
"use client";
import { useState } from "react";

export default function Counter() {
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(count +
1)}>カウント：{count}</button>;
}
```

- appフォルダー内にどちらも配置することが可能
  - 先頭にuse clientがあるかどうかで自動的に判断される

前回参加されていない方

- デスクップにフォルダーを作成してください（nextなどの名前を推奨）
- その後は、スライドに従ってインストール作業をしてください

NPXを使ってNext.jsでのプロジェクト作成

- NPXを使う場合、利用するフォルダーを指定する
- GUIは無いため、ターミナルなどから利用する
  - （Windows11の場合）「スタート」→「すべてのアプリ」→「ターミナル」を選択
  - （Windows10の場合）「スタート」→「ターミナル」を選択。「ターミナル」が見つからない場合、「Windowsシステムツール」から「コマンドプロンプト」を選択
  - （macの場合）Finderから「アプリケーション」→「ユーティリティ」→「ターミナル」を選択
- Windows環境ではWindows PowerShellは利用しないこと

NPXを使ってNext.jsをインストール

- フォルダーを移動する
  - cdと入力し半角スペースを空けて、フォルダーへのパスを記入して、Enter（Return）キー
    - フォルダーをドラッグ＆ドロップする方が最初は分かりやすいかも
    - mac環境の場合、ドラッグ＆ドロップしたら半角スペースが空くので削除した方が無難

NPXを使ってNext.jsをインストール

- フォルダーを移動したらnpx create-next-appと入力

```
C:¥Users¥user¥desktop¥next>npx create-next-app@15 sample_next_app
```

- 最初の実行時に「create-next-app@15.5.6」Ok？的なことを聞かれるので、Enterキーで進める
  - この場合、Next.jsの15.5.6バージョンで進めることになる
- 後のsample_next_appの名前は自由
  - ここに作業環境が作成される
  - 環境構築まで結構時間が掛かる（インターネット回線速度に依存するが、2～3分程度）

NPXを使ってNext.jsをインストール

- ある程度進むと質問が出てくる

Would you like to use TypeScript? >> No / Yes

今回はNoを選択
カーソルキーで切替が可能

Which linter would you like to use? >> ESLint

今回はESLintを選択

Would you like to use Tailwind CSS? >> No / Yes

今回はYesを選択

Would you like to use 'src/' directory? >> No / Yes

今回はYesを選択
カーソルキーで切替が可能

Would you like to use App Router?(recommended) >> No / Yes

今回はYesを選択

Would you like to use Turbopack? (recommended) >> No / Yes

今回はYesを選択

Would you like to customize the default import･･･ >> No / Yes

今回はNoを選択

NPXを使ってNext.jsをインストール

- 質問が終わってからインストールと設定が行なわれる
- Success!という表示が出ればインストールは完了

- インストールはsample_next_appフォルダー内に行なわれる
  - このフォルダーをVSCodeで開いて編集を行なっていく
- 一度表示を確認してみる

```
cd ./sample_next_app
npm run dev
```

ディレクトリを移動

開発サーバーを起動

ありがとうございました。

また次回。