React実習

05. ルーティング

株式会社ジードライブ

今回学ぶこと

- React Routerを使い、複数ページのアプリケーションを実装する方法

React Routerとルーティング

React Routerとは

- 複数ページで構成されるReactアプリを作る際に利用されるライブラリ(Next.jsのようなフレームワークとしても利用可能)
  - https://reactrouter.com/home
  - ルーティング(URLとコンポーネントの対応づけ)を担当する
  - URLの書き換えをイベントの一つと見なし、そのURLに対応するコンポーネントをレンダリングする
    ⇒ サーバーに対して、新たにHTMLを要求するわけではない

- 利用に際しては、npmを使い、パッケージをインストールする必要がある

```
npm install react-router-dom
```

Routerの種類

- Routerには、いくつかの種類があるが、基本的には**BrowserRouter**を使用する

種類	説明
BrowserRouter	一般的なWebブラウザ用ルーター
HashRouter	「/#/members」のような形式でURLが扱われる **どうしても必要な場合以外は使わない**
MemoryRouter	メモリ内にルーティング情報を格納するルーター URLにページのパスの情報が現れない
StaticRouter	Node.js環境で動作するアプリで使用するルーター

Browser Routerの使い方

- BrowerRouter, Routes, Routeの３種類のコンポーネントを、この順番で階層にして配置する
  - これらの階層の間には、他のコンポーネントが入ってもよい
  - 具体的なURLは、Routeコンポーネントにpath属性として指定し、element属性で対応するコンポーネントを指定する

例：main.jsx

```
<StrictMode>
  <BrowserRouter>
    <App />
  </BrowserRouter>
</StrictMode>
```

例：App.jsx（Appコンポーネント）

```
<div className="container">
    <h1>MySystem</h1>
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
    </Routes>
</div>
```

各ページ共通の見出し

ルートのURLでは、
Homeコンポーネントを表示

/about というURLでは、
Aboutコンポーネントを表示

path属性の設定

- 基本的な設定方法
  - ルートのURLを表す場合は「/」のみ記す
  - URLがどのRouteにもマッチしない場合のページ(404ページ等)を用意する場合は、path属性値を「*」にする

例：基本的な設定

```
<Route path="/" element={<Home />} />
<Route path="/about" element={<About />}>
<Route path="*" element={<PageNotFound />} />
```

path属性の設定

- 階層をもつパスの設定
  - Routeコンポーネントを階層化することで、階層化するURLに対応することができる

例：/about以下のURLに対応するページの設定

```
<Route path="/" element={<Home />} />
<Route path="/about">
  <Route index element={<About />} />
  <Route path="products" element={<Products />} />
  <Route path="*" element={<AboutPageNotFound />} />
</Route>
<Route path="*" element={<PageNotFound />} />
```

/aboutに対応
path=""と記述することも可能

/about/productsに対応
属性値を「/」で始めない点に注意

/about以下のURLでマッチする
Routeが無い場合の対応

練習

- 練習05-1

ページリンクの作成

Linkコンポーネント

- 各ページへのリンクは、Linkコンポーネントで記述する
  - to属性でリンク先のパスを指定する
  - クリックすると、JavaScriptのhistory.pushState()が実行され、ブラウザの履歴エントリーに追加される
  - URLは変わるが、サーバーに対して新たなリクエストが送信されるわけではなく、すでにブラウザ上に存在しているJavaScriptを使って新しい画面の描画が行われる
  - 最終的にはa要素として<a href="〇〇">のような形でDOMに追加されるが、通常のa要素とは挙動が異なるので注意する

例：ナビゲーション

```
<ul>
<li><Link to="/" className="nav-link">ホーム</Link></li>
<li><Link to="/about" className="nav-link">会社概要</Link></li>
<li><Link to="/about/products" className="nav-link">製品一覧</Link></li>
</ul>
```

NavLinkコンポーネント

- NavLinkコンポーネントを使うことで、現在のブラウザ上のURLとリンク先のURLがマッチしているか否かを判断して利用することができる
  - マッチしている場合、activeクラスが付与される

例：ナビゲーション

URLがマッチする場合、
class="nav-link **active**" に変換される

```
<li>
  <NavLink to="/" className="nav-link">ホーム</NavLink>
</li>
<li>
  <NavLink to="/about" className="nav-link">会社概要</NavLink>
</li>
<li>
  <NavLink to="/about/products" className="nav-link">製品一覧</NavLink>
</li>
```

NavLinkコンポーネント

- active以外のクラス名を付与する場合、className属性の値を関数にする
  - この関数の引数はオブジェクトになっており、isActiveプロパティとisPendingプロパティを持っている

例：ナビゲーション

```
// 現在のURLとマッチする場合、currentクラスを付与する
function getLinkClass({ isActive }) {
    return isActive ? "nav-link current" : "nav-link";
}
```

```
<li>
  <NavLink to="/about" className={getLinkClass} end>会社概要</NavLink>
</li>
<li className="nav-item">
  <NavLink to="/about/products" className={getLinkClass}>製品リスト
</NavLink>
</li>
```

end属性がないと、/about/productsの場合にもcurrentクラスが付与されてしまう

パラメーターを含むURL

- Routeのpath属性値の中で、コロンで始まる部分は、プレースホルダーとして扱われ、URLに含めるパラメーターとして利用できる
  - プレースホルダーの名前は自由に決めることができる
  - 1つのpath属性に対し、複数個所にプレースホルダーがあっても構わない

例：プレースホルダーを利用したルーティング

```
<Routes>
    <Route path="/" element={<Home />} />
    <Route path="/products" element={<Products />} />
    <Route path="/product/:id" element={<Product />} />
</Routes>
```

以下のようなURLに対し、柔軟に対応することができる
例１：/product/10
例２：/product/apple

パラメーターの取得

- URLに含まれるパラメーターは、useParamsフックで取得することができる
  - 戻り値はオブジェクトになっており、プレースホルダー名がオブジェクトのプロパティ名になっている

例：プレースホルダーを利用したルーティング

```
import { useParams } from "react-router-dom";

export default function Product() {
  const params = useParams();
  const id = params.id;

  return <h1>製品ID:{id}</h1>;
}
```

プレースホルダー名がプロパティ名になっている
取得できるデータは文字列なので、必要に応じて型変換する

リダイレクト

- Navigateコンポーネントを使うことで、リダイレクトすることができる

例：idが整数でない場合は、製品一覧ページへリダイレクト

```
import { Navigate, useParams } from "react-router-dom";

export default function Product() {
  const { id } = useParams(); // 分割代入

  const regex = /^¥d+$/; // 正規表現
  if (!regex.test(id)) {
    // idが整数でない場合は、リダイレクト
    return <Navigate to="/products" replace />;
  }

  return <h1>製品ID:{id}</h1>;
}
```

リダイレクト前の不正なURL
を履歴を残さないようにする

リダイレクト先のURL

リダイレクト

- useNavigateフックで作成したオブジェクトを使い、リダイレクトを実装することもできる

例：useNavigateの使用

```
import { useNavigate } from "react-router-dom";

export default function MemberOnly() {
  const navigate = useNavigate();

  useEffect(() => {
     … ログインのチェックなどの処理
     navigate("/home", { replace: true });
  }, [])

  return …
}
```

練習

- 練習05-2

共通コンポーネントの作成

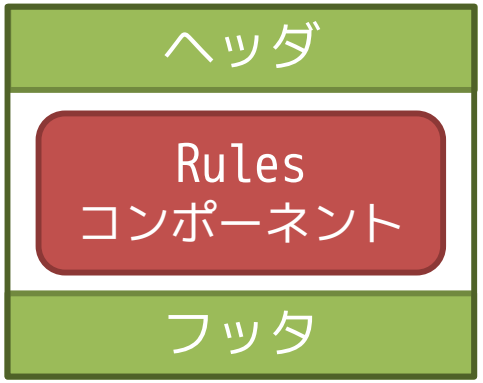
共通コンポーネントの作成

- 例として、以下のようなURLと、それに対応するページの作成手順について考える

/members
会員トップページ

/members/news
お知らせページ

/members/rules
会員規約ページ



これらのページは共通のヘッダ・フッタをもっている

各ページのコンポーネント作成

手順１：

- 各ページごとのコンポーネントを作成する
  - 通常のコンポーネントとして作成する

/members	/members/news	/members/rules
会員トップページ	お知らせページ	会員規約ページ
MemberHome コンポーネント	News コンポーネント	Rules コンポーネント

例：Newsコンポーネント

```
export default function News() {
  return(<>
    <h2>会員向けお知らせ</h2>
    <p>会員限定の…</p>
  </>);
}
```

共通部分の作成

手順２：

- 共通部分のコンポーネントを作成する
  - 各ページごとのコンポーネントが収まる部分に、**Outletコンポーネント**を挿入する

例：MembersCommonコンポーネント

```
import { Outlet } from "react-router-dom";

export default function MembersCommon() {
  return(<>
    <header>ヘッダ</header>
    <Outlet />
    <footer>フッタ</footer>
  </>);
}
```

ルーティング

手順３：

- Routeコンポーネントをネストしてルーティングをする
  - 親のRouteコンポーネントのelement属性として、共通部分のコンポーネントを設定する

例：ルーティング

```
<Routes>
    <Route path="/" element={<Home />} />
    <Route path="/members" element={<MembersCommon />}>
        <Route index element={<MembersHome />} />
        <Route path="news" element={<News />} />
        <Route path="rules" element={<Rules />} />
    </Route>
</Routes>
```

共通部分のコンポーネント

各ページごとのコンポーネントを指定

練習

- 練習05-3