Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Занятие 19

React-router: BrowserRouter | hashRouter | Routes | Route | Link

План занятия

ПЕРЕДОВАЯ
ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

1. React-router
2. BrowserRouter, NativeRouter, HashRouter, HistoryRouter, MemoryRouter, StaticRouter
3. Построение маршрутизации

React-Router

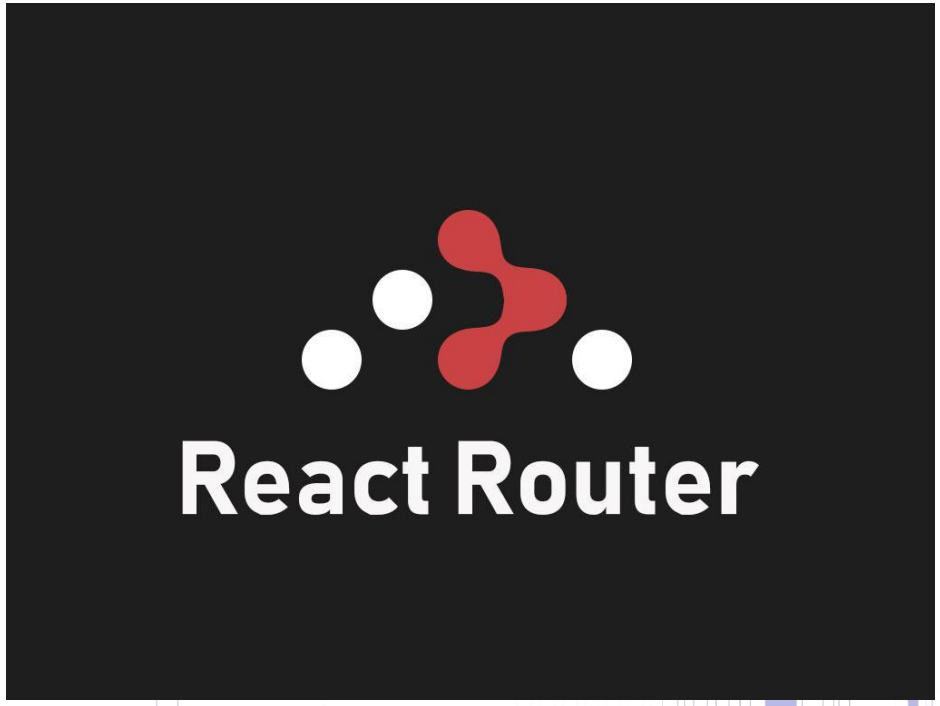
Самая популярная библиотека маршрутизации в React

Основы React Router

Для установки React Router, необходимо запустить npm i react-router-dom.

Библиотека устанавливает DOM версию React Router.

Если используется React Native, нужно будет установить react-router-native. За исключением этого небольшого отличия, библиотеки работают почти одинаково



React-Router

Чтобы использовать React Router, нужно сделать три вещи:

1. Настроить роутер
2. Прописать свои маршруты
3. Управлять навигацией

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Настройка роутера

Настройка роутера является самым простым шагом. Все, что нужно сделать, это импортировать конкретный роутер, который нужен (BrowserRouter для веба и NativeRouter для мобильных устройств) и обернуть все приложение в этот роутер

```
import React from "react"
import ReactDOM from "react-dom/client"
import App from "./App"
import { BrowserRouter } from "react-router-dom"

const root =
ReactDOM.createRoot(document.getElementById("root"))
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
)
```

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Как правило, импортируете свой маршрутизатор в файле index.js приложения и он будет оборачивать компонент App.

React Router работает так же, как контекст в React, и предоставляет всю необходимую информацию приложению, чтобы можно выполнять маршрутизацию и использовать все пользовательские хуки из React Router

React-Router

Рассмотрим все виды роутеров из библиотеки React Router v6:
BrowserRouter, NativeRouter, HashRouter, HistoryRouter, MemoryRouter, StaticRouter

BrowserRouter

Router по умолчанию, который должны использовать, если работаете над веб-приложением, и это роутер, который будете использовать в 99% всех приложений, поскольку он охватывает все обычные варианты использования маршрутизации

NativeRouter

NativeRouter по сути является эквивалентом BrowserRouter, но для React Native. Если используете React Native, то это роутер для него

React-Router

HashRouter

Этот роутер работает очень похоже на BrowserRouter, но основное отличие заключается в том, что вместо того, чтобы изменять URL-адрес на что-то вроде http:*//localhost:3000/books он будет хранить URL-адрес в хэше, как http://localhost:3000/#/books.*

URL-адрес имеет # после URL-адреса, который представляет собой хэш-часть URL-адреса. Все, что находится в хэш-части URL-адреса, является просто дополнительной информацией, которая обычно обозначает идентификатор на странице для целей прокрутки, поскольку страница будет автоматически прокручиваться до элемента с идентификатором, представленным хешем, при загрузке страницы

В React Router этот хэш на самом деле не используется для хранения идентификационной информации для прокрутки, а вместо этого он хранит информацию, связанную с текущим URL-адресом. Причина из-за которой React Router реализует данный функилнал - некоторые хостинг-провайдеры не позволяют фактически изменять URL-адрес страницы. В этих очень редких случаях нужно использовать HashRouter, поскольку HashRouter не изменит фактический URL-адрес вашей страницы, а изменит только хэш вашей страницы.

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

HistoryRouter

HistoryRouter — это роутер, который позволяет вручную управлять объектом истории, который React Router использует для хранения всей информации, связанной с историей маршрутизации приложения

Объект истории помогает убедиться, что такие вещи, как кнопки «Назад» и «Вперед» в браузере, работают правильно

Это роутер, никогда не должен использоваться, если нет очень конкретной причины, по которой нужно перезаписать или контролировать поведение истории по умолчанию React Router

React-Router

MemoryRouter

MemoryRouter немного отличается от остальных роутеров, поскольку вместо того, чтобы хранить информацию о текущем маршруте в URL-адресе браузера, этот роутер хранит информацию о роуте непосредственно в памяти

Это очень неподходящий роутер для обычных операций маршрутизации, но этот роутер невероятно полезен, когда пишутся тесты для приложения, у которого нет доступа к браузеру

В React Router компоненты упакованы в маршрутизатор, иначе весь код маршрутизации будет выдавать ошибки и ломаться. Это означает, что даже если нужно протестировать один компонент, нужно будет обернуть этот компонент внутрь маршрутизатора, иначе он будет выдавать ошибки.

Если тестируете код, что у него нет доступа к браузеру (например, модульное тестирование), то маршрутизаторы, будут выдавать ошибки, поскольку все они зависят от URL-адреса в браузере. MemoryRouter, с другой стороны, хранит всю свою информацию в памяти, что означает, что он никогда не обращается к браузеру и идеально подходит при модульном тестировании компонентов. Однако, за исключением этого конкретного случая использования, этот маршрутизатор не нужен

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

StaticRouter

Роутер StaticRouter очень специфический вариант использования.

Роутер специально предназначен для серверного рендеринга приложений React, поскольку он принимает один props location и визуализирует приложение, используя этот location в качестве URL-адреса.

Роутер на самом деле не может выполнять какую-либо маршрутизацию и будет просто отображать одну статическую страницу, но это идеально для рендеринга на сервере, поскольку просто отобразить HTML приложения на сервере, а затем клиент может настроить всю маршрутизацию и так далее

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Определение маршрутов

Следующим шагом в React Router является определение маршрутов

Обычно это делается на верхнем уровне приложения, например в компоненте App, но можно сделать в любом месте

```
import { Route, Routes } from "react-router-dom"
import { Home } from "./Home"
import { BookList } from "./BookList"

export function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/books" element={<BookList />} />
    </Routes>
  )
}
```

React-Router

Определить компонент Route для каждого маршрута в приложении, а затем поместить все эти компоненты Route в один компонент Routes.

Когда URL-адрес изменяется, React Router будет просматривать маршруты, определенные в компоненте Routes, и будет отображать содержимое в props element Route, который с path, соответствующий URL-адресу

Преимущество React Router заключается в том, что при переходе между страницами он обновляет только содержимое внутри компонента Routes

Весь остальной контент на странице останется прежним, что повысит производительность и удобство использования

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Управление навигацией

В React Router производится обработка навигации

Обычно в приложении перемещаются с помощью тегов <a>, но React Router использует свой собственный кастомный компонент Link для обработки навигации

Link представляет собой просто оболочку вокруг тега <a>, которая помогает обеспечить правильную обработку всей маршрутизации и условного повторного рендеринга, чтобы использовать его так же, как обычный тег <a>

React-Router


```
import { Route, Routes, Link } from "react-router-dom"
import { Home } from "./Home"
import { BookList } from "./BookList"

export function App() {
  return (
    <>
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to="/books">Books</Link></li>
        </ul>
      </nav>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/books" element={<BookList />} />
      </Routes>
    </>
  )
}
```

React-Router

В примере добавлены две ссылки на главную страницу и страницу с каталогом книг

Использовали prop to для установки URL-адреса вместо props href, который привыкли использовать с тегом <a>

Это единственное различие между компонентом Link и тегом <a>, и это то, что нужно помнить, так как легко ошибочно случайно использовать prop href вместо to

<nav>, который создан в верхней части страницы, находится за пределами компонента Routes, что означает, что при смене страниц этот раздел навигации не будет повторно рендериться, так при изменении URL-адреса изменится только содержимое компонента Routes

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Динамическая маршрутизация
Самая простая и распространенная расширенная функция в React Router — это обработка динамических маршрутов.

Предположим, что хотим визуализировать компонент для отдельных книг в приложении. Можно жестко закодировать каждый из этих маршрутов, но если есть сотни книг или возможность для пользователей создавать книги, то невозможно жестко закодировать все эти маршруты. Вместо этого нужен динамический маршрут

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/books" element={<BookList />} />
  <Route path="/books/:id" element={<Book />} />
</Routes>
```

React-Router

Последний маршрут в приведенном примере — это динамический маршрут с динамическим параметром :id

Для определения динамического маршрута в React Router нужно поставить двоеточие перед тем, что должно изменяться

Динамический маршрут будет соответствовать любому URL-адресу, который начинается с /book и заканчивается каким-либо значением. Например, /books/1, /books/bookName и /books/literally-anything будут соответствовать динамическому маршруту

Почти всегда, когда есть такой динамический маршрут, получить доступ к динамическому значению в пользовательском компоненте можно через хук useParams

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

export function Book() {
  const { id } = useParams()

  return (
    <h1>Book {id}</h1>
  )
```

React-Router

Хук useParams не принимает параметры и возвращает объект с ключами, соответствующими динамическим параметрам в маршруте.

Динамическим параметром является :id, поэтому хук useParams вернет объект, имеющий ключ id, и значение этого ключа будет фактическим идентификатором в URL

Например, если бы URL-адрес был /books/3, страница отображала бы Book 3

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Приоритет маршрутизации

Жестко закодированными маршрутами, было довольно легко узнать, какой маршрут будет отображаться, но при работе с динамическими маршрутами это немного сложнее

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/books" element={<BookList />} />
  <Route path="/books/:id" element={<Book />} />
  <Route path="/books/new" element={<NewBook />} />
</Routes>
```

Если есть URL-адрес /books/new, какой маршрут будет задействован?

Технически есть два маршрута, которые совпадают. И /books/:id, и /books/new. Они будут совпадать, так как динамический маршрут будет просто предполагать, что new является частью :id URL-адреса, поэтому React Router нужен другой способ определить /books/:id какой маршрут отренедрить

React-Router

В старых версиях React Router маршрут, который был определен первым, будет визуализирован, поэтому будет отображаться маршрут /books/:id, что, очевидно, не то, что нужно

Версия 6 React Router изменила это подход, поэтому теперь React Router будет использовать алгоритм, чтобы определить, какой маршрут, скорее всего, является тем, который нужен.

Хотим отобразить /books/new, поэтому React Router выберет этот маршрут

Фактический принцип работы этого алгоритма очень похож на специфику CSS, поскольку он попытается определить, какой маршрут, соответствующий URL-адресу, является наиболее конкретным (имеет наименьшее количество динамических элементов), и выберет этот маршрут

React-Router


```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/books" element={<BookList />} />
  <Route path="/books/:id" element={<Book />} />
  <Route path="/books/new" element={<NewBook />} />
  <Route path="*" element={<NotFound />} />
</Routes>
```

* соответствует чему угодно, что делает его идеальным для таких вещей, как страница 404

Маршрут, содержащий *, также будет менее конкретным, чем все остальное

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Вложенные маршруты

Есть три маршрута, которые начинаются с /books, поэтому можем вложить эти маршруты друг в друга, чтобы очистить основные маршруты

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/books">
    <Route index element={<BookList />} />
    <Route path=":id" element={<Book />} />
    <Route path="new" element={<NewBook />} />
  </Route>
  <Route path="*" element={<NotFound />} />
</Routes>
```

React-Router

Вложение довольно просто сделать - это создать родительский Route, в котором свойство path будет установлено на общий путь для всех ваших дочерних компонентов Route

Внутри родительского маршрута можно поместить все дочерние компоненты Route.

Единственное отличие состоит в том, что prop path дочерних компонентов Route больше не включает общий маршрут /books.

Маршрут для /books заменяется компонентом Route, который не имеет props path, но вместо этого имеет index. Путь индексного маршрута совпадает с родительским Route

Сила вложенных маршрутов заключается в том, как они обрабатывают общие макеты……..

React-Router

Общие макеты (layout)

Раздел nav со ссылками на каждую книгу, а также форму для добавления книг с любой из страниц.

Нужно создать общий компонент для хранения навигации, а затем импортировать его в каждый отдельный компонент, связанный с книгой

Однако это создает дублирование, поэтому React Router создал собственный вариант решения этой проблемы

Передаем props element родительскому маршруту, он отобразит этот компонент для каждого дочернего Route, поэтому легко разместить общую навигацию или другие общие компоненты на каждой дочерней странице

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/books" element={<BooksLayout />}>
    <Route index element={<BookList />} />
    <Route path=":id" element={<Book />} />
    <Route path="new" element={<NewBook />} />
  </Route>
  <Route path="*" element={<NotFound />} />
</Routes>
```

```
import { Link, Outlet } from "react-router-dom"

export function BooksLayout() {
  return (
    <>
      <nav>
        <ul>
          <li><Link to="/books/1">Book 1</Link></li>
          <li><Link to="/books/2">Book 2</Link></li>
          <li><Link to="/books/new">New Book</Link></li>
        </ul>
      </nav>

      <Outlet />
    </>
  )
}
```

React-Router

Код будет работать следующим образом: всякий раз, когда сопоставляется маршрут внутри родительского Route /book, он будет отображать компонент BooksLayout, который содержит общую навигацию.

Какой бы дочерний маршрут ни был сопоставлен, он будет отображаться везде, т.к. компонент Outlet находится внутри компонента макета

Компонент Outlet — это, по сути, компонент-заполнитель, который будет отображать любое содержимое текущей страницы

Структура невероятно полезна и делает обмен общими кусками между маршрутами невероятно простым

React-Router


```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/books" element={<BooksLayout />}>
    <Route index element={<BookList />} />
    <Route path=":id" element={<Book />} />
    <Route path="new" element={<NewBook />} />
  </Route>
  <Route element={<OtherLayout />}>
    <Route path="/contact" element={<Contact />} />
    <Route path="/about" element={<About />} />
  </Route>
  <Route path="*" element={<NotFound />} />
</Routes>
```

Два маршрута, /contact и /about, которые отображаются внутри компонента OtherLayout. Метод оборачивания нескольких компонентов Route в родительский компонент Route без props path полезен, чтобы эти маршруты использовали один макет, даже если у них нет похожего path.

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Контекст Outlet

Компоненты Outlet могут принимать prop context, который будет работать так же, как контекст React

```
import { Link, Outlet } from "react-router-dom"

export function BooksLayout() {
  return (
    <>
      <nav>
        <ul>
          <li><Link to="/books/1">Book 1</Link></li>
          <li><Link to="/books/2">Book 2</Link></li>
          <li><Link to="/books/new">New Book</Link></li>
        </ul>
      </nav>

      <Outlet context={{ hello: "world" }} />
    </>
  )
}
```

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

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

export function Book() {
  const { id } = useParams()
  const context = useOutletContext()

  return (
    <h1>Book {id} {context.hello}</h1>
  )
}
```

Передаем значение контекста { hello: "world" }, а затем в дочернем компоненте используем хук useOutletContext для доступа к значению контекста

Распространенный шаблон для использования, поскольку часто нужно передавать общие данные между всеми дочерними компонентами, что является идеальным вариантом использования для этого контекста

React-Router

Множественные маршруты

Использовать несколько компонентов Routes одновременно можно сделать в виде двух отдельных компонентов Routes, либо в виде вложенных Routes.

Отдельные Routes

Два разных раздела контента, которые зависят от URL-адреса приложения, вам потребуется несколько компонентов Routes. Это очень распространенный случай, если, например, у вас есть боковая панель, на которой вы хотите отображать определенный контент для определенных URL-адресов, а также главная страница, которая должна отображать определенный контент на основе URL-адреса.

React-Router


```jsx
import { Route, Routes, Link } from "react-router-dom"
import { Home } from "./Home"
import { BookList } from "./BookList"
import { BookSidebar } from "./BookSidebar"

export function App() {
  return (
    <>
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to="/books">Books</Link></li>
        </ul>
      </nav>

      <aside>
        <Routes>
          <Route path="/books" element={<BookSidebar />}>
        </Routes>
      </aside>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/books" element={<BookList />} />
      </Routes>
    </>
  )
}
```

React-Router

Основные маршруты Routes определяют все основные компоненты для страницы и есть вторичные Routes внутри aside, которые будут отображать боковую панель для страницы книг, когда находимся в /books

Если URL-адрес /books, то оба компонента Routes будут отображать контент, поскольку они оба имеют уникальное совпадение для /books в своих Routes.

Можно сделать с несколькими компонентами Routes, — это жестко закодировать prop location

```
<Routes location="/books">
  <Route path="/books" element={<BookSidebar />}>
</Routes>
```

Жестко кодируя prop location, переопределяем поведение по умолчанию React Router, поэтому независимо от того, какой URL-адрес страницы, этот компонент Routes будет соответствовать своему Route, как если бы URL-адрес был /books.

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Вложенные Routes

Использование нескольких компонентов Routes заключается в том, чтобы вложить их друг в друга.

Для большого количества маршрутов можно переместив похожие маршруты в свои собственные файлы

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/books/*" element={<BookRoutes />} />
  <Route path="*" element={<NotFound />} />
</Routes>
```

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

```
import { Routes, Route } from "react-router-dom"
import { BookList } from "./pages/BookList"
import { Book } from "./pages/Book"
import { NewBook } from "./pages/NewBook"
import { BookLayout } from "./BookLayout"

export function BookRoutes() {
  return (
    <Routes>
      <Route element={<BookLayout />}>
        <Route index element={<BookList />} />
        <Route path=":id" element={<Book />} />
        <Route path="new" element={<NewBook />} />
        <Route path="*" element={<NotFound />} />
      </Route>
    </Routes>
  )
}
```

React-Router

Нужно создать новый компонент для хранения вложенных Routes

Компонент должен иметь компонент Routes, а внутри этого компонента Routes должны быть все компоненты Route, которые сопоставляются с родительским Route

Маршруты /books в этот компонент BookRoute.
Затем в родительских Routes нужно определить Route, который равный пути, разделяемому всеми вложенными Routes - это /books

Нужно в последнем path родительского маршрута создать маршрут *, иначе он не будет правильно соответствовать дочерним роутам

Маршрут начинается с /book/, он ищет внутри компонента BookRoutes, есть ли соответствующий Route

Для этого еще один маршрут * в BookRoutes, чтобы точно гарантировать, что если URL-адрес не совпадает ни с одним из BookRoutes, он правильно отобразит компонент NotFound

React-Router

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Хук useRoutes

Последнее, что нужно знать об определении маршрутов в React Router, что можно использовать объект JavaScript для определения маршрутов вместо JSX

```
import { Route, Routes } from "react-router-dom"
import { Home } from "./Home"
import { BookList } from "./BookList"
import { Book } from "./Book"

export function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/books">
        <Route index element={<BookList />} />
        <Route path=":id" element={<Book />} />
      </Route>
    </Routes>
  )
}
```

React-Router


```
import { Route, Routes } from "react-router-dom"
import { Home } from "./Home"
import { BookList } from "./BookList"
import { Book } from "./Book"

export function App() {
  const element = useRoutes([
    {
      path: "/",
      element: <Home />
    },
    {
      path: "/books",
      children: [
        { index: true, element: <BookList /> },
        { path: ":id", element: <Book /> }
      ]
    }
  ])

  return element
}
```

Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Спасибо за внимание