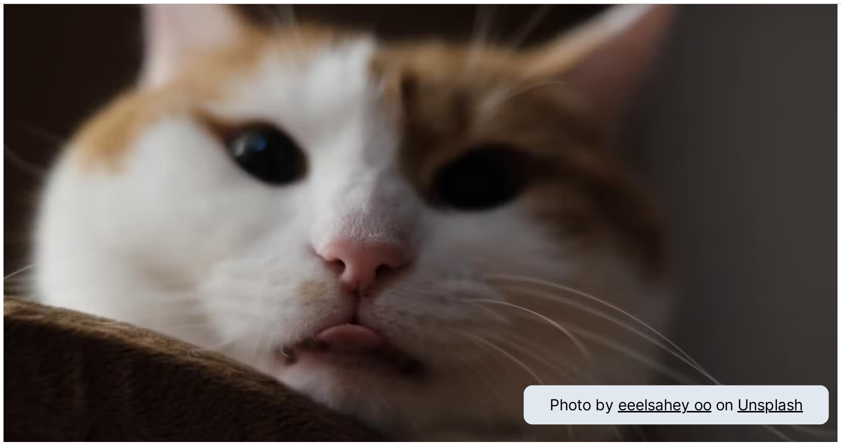


Photo by eeelsahey oo on Unsplash

Learning How React Router DOM works With ChatGPT

Mayur Patil · Jul 26, 2023 · 12 min read

you know in react-router dom, we can use routing and links which don't refresh full page while loading, behind the hood link also use a tag but why does the page not refresh?

ChatGPT

In React Router DOM, when you use the `<Link>` component, it internally uses the HTML `<a>` tag for navigation. However, the reason the page does not refresh like a traditional anchor link is due to React Router's mechanism for handling navigation.

When you use the `<Link>` component, React Router intercepts the click event on the link, preventing the default behavior of the anchor link, which is to reload the entire page. Instead, React Router updates the URL and renders the appropriate component based on the route specified in the `<Link>`.

Here's a simplified overview of what happens when you use `<Link>`:

1. When the user clicks on a `<Link>`, React Router captures the click event.
2. React Router then checks if the clicked link matches any defined routes in the application's routing configuration.
3. If there is a match, React Router updates the URL using the HTML5 History API (pushState or replaceState). This updates the displayed URL in the browser without causing a full page reload.
4. After updating the URL, React Router re-renders the appropriate component for the new route without a full page refresh. This is possible because React is designed to update only the parts of the DOM that have changed (virtual DOM diffing), resulting in a more efficient and faster update process.

By handling navigation internally and updating the URL without a full page refresh, React Router provides a smoother and more seamless user experience compared to traditional anchor links. This is a key advantage of using a client-side routing library like React Router in modern web applications.

ohh do you mean, react-router-dom uses HTML 5 history API for doing this? can you tell me how it works step by step with example and implementation

ChatGPT

Yes, that's correct! React Router DOM uses the HTML5 History API to handle client-side navigation without full page reloads. The History API allows JavaScript to manipulate the browser's session history, which includes adding, modifying, and changing the browser's URL without causing a full page refresh.

Here's a step-by-step explanation of how the React Router DOM handles navigation using the History API:

1. **Initial Routing Setup:** When you set up your React application with React Router DOM, you define your routing configuration using the `<BrowserRouter>` or `<HashRouter>` component. This is typically done in the root of your application, such as in the `index.js` or `App.js` file.
   For example, in `index.js`, you might have:

   ```
   jsxCopy codeimport React from 'react';
   import ReactDOM from 'react-dom';
   import { BrowserRouter as Router } from 'react-router-dom';
   import App from './App';

   ReactDOM.render(
     <Router>
       <App />
     </Router>,
     document.getElementById('root')
   );
   ```

2. **Navigation with `<Link>`:** In your application components, you use the `<Link>` component from React Router DOM to create navigation links.

   ```
   jsxCopy codeimport React from 'react';
   import { Link } from 'react-router-dom';

   function Home() {
     return (
       <div>
         <h1>Home Page</h1>
         <Link to="/about">Go to About Page</Link>
       </div>
     );
   }

   export default Home;
   ```

3. **Handling Navigation:** When a user clicks on a `<Link>` component, React Router intercepts the click event and prevents the default behavior of the anchor link, which is to cause a full page refresh.
4. **Updating URL:** After preventing the default behavior, React Router uses the `pushState` or `replaceState` method of the History API to update the URL in the browser's address bar with the new route. This change in the URL does not cause a full page reload.

For example, if the user clicks on the "Go to About Page" link, the URL will be updated to `http://yourdomain.com/about` without a full page refresh.

5. **Rendering New Component:** After updating the URL, React Router identifies the matching route based on the URL and renders the appropriate component associated with that route.
   In the example above, when the URL is updated to `/about`, the `<App>` component (or the component responsible for handling the `/about` route) will be rendered, and the content of the page will change accordingly.
6. **Virtual DOM Diffing:** React's virtual DOM diffing mechanism comes into play at this point. React compares the new component tree with the previous one, and it efficiently updates only the parts of the DOM that have changed, resulting in a fast and seamless update of the user interface.

This process allows React Router DOM to handle navigation within your React application without causing full page refreshes, providing a smooth and efficient user experience. The use of the HTML5 History API enables React Router to manage the browser's history and handle client-side navigation effectively.

Now i understood the basic of it, now tell me step by step with detailed explnation how it works

ChatGPT

Sure, let's dive into a more detailed step-by-step explanation of how React Router DOM works with the HTML5 History API to handle client-side navigation without full page refreshes:

1. **Initial Routing Setup:** In your React application, you set up the routing configuration using the `<BrowserRouter>` or `<HashRouter>` component. The choice between `<BrowserRouter>` and `<HashRouter>` depends on how your web server is configured. For most cases, `<BrowserRouter>` is preferred.
   For example, in your `index.js` file, you would have:

   COPY

   ```
   jsxCopy codeimport React from 'react';
   import ReactDOM from 'react-dom';
   import { BrowserRouter as Router } from 'react-router-dom';
   import App from './App';

   ReactDOM.render(
     <Router>
       <App />
     </Router>,
     document.getElementById('root')
   );
   ```

2. **Route Configuration:** You define the routes and their corresponding components using the `<Route>` component from React Router DOM. This is typically done in your main `App.js` or `Main.js` file. Each `<Route>` component represents a different page or view in your application.
   For example, in `App.js`, you might have:

   COPY

   ```
   jsxCopy codeimport React from 'react';
   import { Route, Switch } from 'react-router-dom';
   import Home from './components/Home';
   import About from './components/About';
   import Contact from './components/Contact';
   ```

```
function App() {
  return (
    <div>
      <Switch>
        <Route exact path="/" component={Home} />
        <Route path="/about" component={About} />
        <Route path="/contact" component={Contact} />
      </Switch>
    </div>
  );
}

export default App;
```

3. **Navigation with `<Link>`:** To create navigation links within your components, you use the `<Link>` component from React Router DOM. The `<Link>` component generates anchor tags (`<a>`) under the hood.
   For example, in your `Header.js` component, you might have:

   ```
   jsxCopy codeimport React from 'react';
   import { Link } from 'react-router-dom';

   function Header() {
     return (
       <header>
         <nav>
           <ul>
             <li>
               <Link to="/">Home</Link>
             </li>
             <li>
               <Link to="/about">About</Link>
             </li>
             <li>
               <Link to="/contact">Contact</Link>
             </li>
           </ul>
         </nav>
       </header>
     );
   }

   export default Header;
   ```

4. **Handling Navigation:** When a user clicks on a `<Link>` component, React Router intercepts the click event and prevents the default behavior of the anchor link, which is to cause a full page refresh.
5. **Updating URL:** After preventing the default behavior, React Router uses the `pushState` or `replaceState` method of the HTML5 History API to update the URL in the browser's address bar with the new route. This change in the URL does not cause a full page reload.
   For example, if the user clicks on the "About" link, the URL will be updated to `http://yourdomain.com/about` without a full page refresh.
6. **Rendering New Component:** After updating the URL, React Router identifies the matching route based on the URL and renders the appropriate component associated with that route.

In the example above, when the URL is updated to `/about`, the `<About>` component will be rendered, and the content of the page will change accordingly.

7. **Virtual DOM Diffing:** React's virtual DOM diffing mechanism comes into play at this point. React compares the new component tree with the previous one, and it efficiently updates only the parts of the DOM that have changed, resulting in a fast and seamless update of the user interface.
8. **Handling Back/Forward Navigation:** React Router also handles back and forward navigation using the browser's back and forward buttons. When the user clicks the back or forward button, React Router intercepts the event, updates the URL, and renders the corresponding component, just like it does with `<Link>` navigation.
9. **404 Not Found Page:** If the user tries to access a route that doesn't exist, React Router can render a 404 Not Found page by using a `<Route>` component with no specified `path`. This component will be rendered when no other routes match the current URL.

That's a detailed explanation of how React Router DOM works with the HTML5 History API to provide seamless client-side navigation in a React application. It allows you to build single-page applications (SPAs) where the user can navigate between different views without the need for full page refreshes, resulting in a smoother and more responsive user experience.

i understood upto 5th step, explain 6 th step more detailed, means react router identifies new url which is added by pushState and then how it loads that new component according to new url ?

ChatGPT

Sure! Let's dive deeper into step 6, where React Router identifies the new URL and loads the corresponding component according to that URL:

1. **Rendering New Component:**
   When the user clicks on a `<Link>` component or interacts with the browser's back or forward buttons, React Router identifies the new URL based on the current browser location. It does this by examining the updated URL after the `pushState` or `replaceState` method from the HTML5 History API is called.
   Once React Router has the new URL, it starts the process of rendering the corresponding component associated with that URL. To achieve this, React Router uses a core concept called the "Route Matching Algorithm."

   ## Route Matching Algorithm:

   The Route Matching Algorithm is the mechanism by which React Router determines which component to render based on the URL. It involves matching the current URL against the list of defined routes in your application.
   React Router keeps track of all the defined routes in your application, along with the components associated with each route. When the URL changes, React Router compares the current URL against the defined routes in order, until it finds a route that matches the URL.
   The matching process is performed using the `path` prop of each `<Route>` component. The `path` prop defines the pattern of the URL that the route should match. When a route's `path` matches the current URL, React Router knows which component to render.
   Example:

Let's consider the following simplified example:

COPY

```
jsxCopy code// In App.js
import React from 'react';
import { Route, Switch } from 'react-router-dom';
import Home from './components/Home';
import About from './components/About';
import Contact from './components/Contact';

function App() {
  return (
    <div>
      <Switch>
        <Route exact path="/" component={Home} />
        <Route path="/about" component={About} />
        <Route path="/contact" component={Contact} />
      </Switch>
    </div>
  );
}

export default App;
```

In this example, we have three routes defined: `/`, `/about`, and `/contact`, and each route is associated with a specific component.

- If the current URL is `/`, React Router matches the first route (`/`) and renders the `<Home>` component.
- If the current URL is `/about`, React Router matches the second route (`/about`) and renders the `<About>` component.
- If the current URL is `/contact`, React Router matches the third route (`/contact`) and renders the `<Contact>` component.

If the URL doesn't match any of the defined routes, React Router will not render any component. This is when you might use a `<Route>` component with no specified `path` as a catch-all for 404 Not Found pages.

Rendering Process:

Once React Router has identified the matching route, it proceeds to render the corresponding component. During this process, React will create a new virtual DOM tree for the component and perform a diffing mechanism to update the actual DOM. The result is that the new component is rendered on the page, and the content of the page changes accordingly.

This rendering process is what allows React Router to provide the illusion of navigating between different views or pages without the need for full page refreshes.

By using the Route Matching Algorithm, React Router efficiently handles the rendering of components based on the URL, providing a seamless user experience when navigating within your React application. It's important to define your routes accurately and ensure that each route corresponds to the correct component to achieve the desired behavior in your single-page application.

Route Matching Algorithm ? how it works ?

ChatGPT

The Route Matching Algorithm is a crucial part of React Router's functionality. It determines which component to render based on the current URL and the defined routes in your application. When the user navigates to a new URL, React Router goes through the following steps to find the matching route and render the corresponding component:

1. **Parsing the URL:** When the user navigates to a new URL or interacts with the browser's back/forward buttons, React Router first parses the current URL to extract its path and any associated parameters or query strings.
2. **Comparing with Defined Routes:** React Router then compares the parsed URL against the list of defined routes in your application. Each route is represented by a `<Route>` component with a specific `path` prop.
3. **Matching Algorithm:** The matching process involves iterating through the defined routes one by one to find the first route that matches the current URL. The comparison is typically based on the `path` prop of each `<Route>` component.
4. **Path Matching:** The `path` prop in a `<Route>` component defines the URL pattern that the route should match. It can include placeholders for dynamic segments of the URL, such as parameters. React Router uses path-to-regexp library under the hood to transform the path pattern into a regular expression for efficient matching.
   For example, the `path` prop `/users/:id` would match URLs like `/users/123` or `/users/abc`, where `:id` is a parameter that can take any value.
5. **Partial and Exact Matches:** By default, React Router performs partial matches. It means that if the current URL is `/users/123/settings`, and you have a route with `path="/users/:id"`, it will match this route. This behavior allows you to nest routes and create complex nested UI structures.
   If you want to enforce exact matches, you can use the `exact` prop on a `<Route>` component. With `exact`, the route will only match if the current URL exactly matches the `path`.
6. **Rendering the Component:** Once React Router finds a matching route, it knows which component to render. It will create a new virtual DOM tree for that component and update the actual DOM to display the content associated with that route.
   If no route matches the current URL, React Router can optionally render a "Not Found" component, which serves as a catch-all for URLs that don't correspond to any defined route.
7. **Nested Routes:** React Router also supports nested routes, where components can have child routes with their own `path` and components. When navigating to a nested route, React Router performs a recursive search for the matching route, starting from the root level and going down the component tree.

By using the Route Matching Algorithm, React Router enables single-page applications to have multiple views that change dynamically based on the URL. It allows developers to build complex navigation systems, handle dynamic URLs, and provide a seamless user experience without the need for full page refreshes.

Subscribe to my newsletter

Read articles from directly inside your inbox.
Subscribe to the newsletter, and don't miss out.

Enter your email address **SUBSCRIBE**

WRITTEN BY

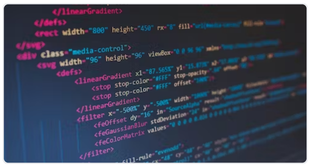
Mayur Patil

Follow

Frontend Developer || UI Developer || JavaScript || React

MORE ARTICLES

Mayur Patil



Why you should use textContent instead of innerHTML ?

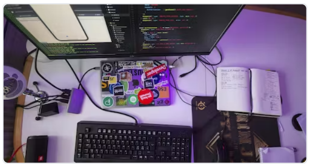
Note : This is Copied Content From ChatGPT for my own Reference :) ChatGPT Using textContent instea...

Mayur Patil

Best Blogs For Programmers (From Great Programmers)

Today, Out of Curiosity, I was exploring How Browsers Works and I landed on web.dev blog post and fo...

Mayur Patil



Top 30 React Native Interview Questions For All Levels

Junior Level: What is React Native and how does it differ from React? Explain the concept of JSX i...

©2023 Mayur Patil's Blog