Introdução ao React

Prof. Karan Luciano

Desenvolvimento Web I - ADS

18 de novembro de 2024

Sumário

Resumo do DOM

- O DOM é a representação da estrutura de um documento HTML.
- Elementos HTML são nós em uma árvore.
- JavaScript permite manipular o DOM (ex.: `getElementById`).
- Mudanças no DOM afetam a exibição da página.
- Eventos (ex.: cliques) podem ser gerenciados com `addEventListener`.

O que é React?

- Biblioteca JavaScript para construção de interfaces de usuário.
- Desenvolvida e mantida pelo Facebook.
- Foco em criar componentes reutilizáveis e gerenciamento eficiente do DOM.

Por que usar React?

- **Componentização**: Permite criar componentes reutilizáveis.
- **Performance**: Uso do Virtual DOM para atualizações eficientes.
- **Comunidade Ativa**: Grande suporte e vasta quantidade de recursos.

Como o React Funciona

- Utiliza uma abordagem declarativa para construir UIs.
- Mantém um "Virtual DOM"que otimiza as atualizações do DOM real.
- Atualiza eficientemente apenas os componentes que necessitam ser re-renderizados.

Analogia do Virtual DOM

- Pense no Virtual DOM como uma lista de compras.
- Em vez de ir ao mercado várias vezes, você anota tudo e vai uma única vez.
- O React faz mudanças no Virtual DOM e depois atualiza o DOM real de forma eficiente.

Pré-requisitos

- **Node.js** instalado (inclui o npm).
- Editor de código (ex.: Visual Studio Code).
- Conhecimento básico em JavaScript.

Instalando Node.js e npm no Windows

1. Baixe o instalador em https://nodejs.org/.
2. Execute o instalador e siga as instruções.
3. Verifique a instalação no Prompt de Comando:
   - `node -v`
   - `npm -v`

Instalando Node.js e npm no macOS

1. Baixe o instalador em https://nodejs.org/.
2. Alternativamente, use o Homebrew:
   - `brew install node`
3. Verifique a instalação no Terminal:
   - `node -v`
   - `npm -v`

Instalando Node.js e npm no Linux

1. Use o gerenciador de pacotes da sua distribuição:
   - Debian/Ubuntu:
     - `sudo apt update`
     - `sudo apt install nodejs npm`
   - Fedora:
     - `sudo dnf install nodejs npm`
2. Verifique a instalação no Terminal:
   - `node -v`
   - `npm -v`

Criando um Novo Projeto React

1. Abra o terminal.
2. Use o comando `npx create-react-app meu-app`.
3. Entre na pasta do projeto: `cd meu-app`.
4. Inicie o servidor de desenvolvimento: `npm start`.
5. Acesse `http://localhost:3000` no navegador.

Componentes

- Blocos de construção da interface.
- Podem ser funcionais ou de classe.
- Reutilizáveis e independentes.

Analogia de Componentes

- Pense em componentes como peças de LEGO.
- Você pode combinar diferentes peças para construir estruturas complexas.
- Cada peça (componente) é independente e reutilizável.

Props

- Abreviação de "Properties".
- Dados passados de um componente pai para um componente filho.
- Imutáveis dentro do componente filho.

Analogia para Props

- Imagine que um chefe (componente pai) dá ingredientes (props) para um cozinheiro (componente filho).
- O cozinheiro usa esses ingredientes para preparar um prato.
- Os ingredientes não devem ser alterados pelo cozinheiro.

Exemplo de Uso de Props

```
const Saudacao = (props) => {
    return <h1>Ol\'a, {props.nome}!</h1>;
};

// Uso do componente
const App = () => {
    return (
        <div>
            <Saudacao nome="Maria" />
            <Saudacao nome="Jo\~ao" />
        </div>
    );
};
```

Estado (State)

- Armazena informações internas do componente.
- Pode ser alterado dentro do componente.
- Alterações no estado causam re-renderização do componente.

Analogia para State

- Pense no estado como o humor de uma pessoa.
- O humor pode mudar ao longo do dia (estado muda).
- Quando o humor muda, a pessoa pode agir de forma diferente (componente re-renderiza).

Exemplo de Estado sem useState

```
class Contador extends React.Component {
    constructor(props) {
        super(props);
        this.state = { contagem: 0 };
    }

    incrementar = () => {
        this.setState({ contagem: this.state.contagem + 1 });
    }

    render() {
        return (
            <div>
                <p>Voc\^e clicou {this.state.contagem} vezes</p>
                <button onClick={this.incrementar}>
                    Clique aqui
                </button>
            </div>
        );
    }
}
```

useState Hook

- Hook que permite adicionar estado a componentes funcionais.
- Retorna um par: o valor atual do estado e uma função para atualizá-lo.

Analogia para useState

- Imagine uma caixa (estado) com um valor dentro.
- Você tem uma chave (função) que permite abrir a caixa e mudar o valor.
- `useState` fornece tanto a caixa quanto a chave.

Exemplo de useState

```
1  import React, { useState } from 'react';
2
3  const Contador = () => {
4      const [contagem, setContagem] = useState(0);
5
6      return (
7          <div>
8              <p>Voc\^e clicou {contagem} vezes</p>
9              <button onClick={() => setContagem(contagem + 1)}>
10                 Clique aqui
11             </button>
12         </div>
13     );
14 };
15
16 export default Contador;
```

useEffect Hook

- Permite realizar efeitos colaterais em componentes funcionais.
- Substitui métodos de ciclo de vida como `componentDidMount` e `componentDidUpdate`.
- Pode retornar uma função de limpeza para evitar vazamentos de memória.

Analogia para useEffect

- Pense em `useEffect` como um alarme que dispara após um evento.
- Quando algo muda (como o estado), o alarme toca e executa uma ação.
- Você pode definir o que acontece quando o alarme dispara.

Exemplo de useEffect

```
import React, { useState, useEffect } from 'react';

const ExemploUseEffect = () => {
    const [contador, setContador] = useState(0);

    useEffect(() => {
        document.title = `Voc\^e clicou ${contador} vezes`;
    }, [contador]); // Executa quando 'contador' muda

    return (
        <div>
            <p>Voc\^e clicou {contador} vezes</p>
            <button onClick={() => setContador(contador + 1)}>
                Clique aqui
            </button>
        </div>
    );
};

export default ExemploUseEffect;
```

Quando usar useEffect

- Fetch de dados de uma API.
- Subscrever a eventos.
- Manipular o DOM diretamente.
- Limpar subscrições ou temporizadores.

Exemplo de Fetch com useEffect

```jsx
import React, { useState, useEffect } from 'react';

const ListaUsuarios = () => {
    const [usuarios, setUsuarios] = useState([]);

    useEffect(() => {
        fetch('https://api.example.com/usuarios')
            .then(response => response.json())
            .then(data => setUsuarios(data));
    }, []); // Executa apenas uma vez

    return (
        <ul>
            {usuarios.map(usuario => (
                <li key={usuario.id}>{usuario.nome}</li>
            ))}
        </ul>
    );
};

export default ListaUsuarios;
```

Resumo

- React facilita a criação de UIs com componentes reutilizáveis.
- **Props** são como ingredientes passados para componentes filhos.
- **State** representa dados internos que podem mudar.
- **useState** permite adicionar estado a componentes funcionais.
- **useEffect** permite realizar efeitos colaterais em componentes funcionais.

Próximos Passos

- Explorar mais hooks: `useContext`, `useReducer`, etc.
- Aprender sobre React Router para navegação entre páginas.
- Integrar APIs externas e gerenciar estado com Redux.
- Praticar construindo seus próprios projetos.

Obrigado!

Dúvidas? Pergunte-me!