

Executar os processos de codificação, manutenção e documentação de aplicativos computacionais para desktop

SENAC PE

Thiago Nogueira

Instrutor de Educação Profissional

Hook useRef no React

useRef

O que é?

- No React, **useRef** é um Hook especial que ajuda você a "referenciar" ou rastrear um elemento DOM ou um valor em seus componentes funcionais sem causar novas renderizações.
- Mas o que isso significa?
- Imagine que você tem um caderno mágico onde você pode escrever uma nota e sempre consultá-la, não importa quantas vezes você mude as páginas. Nessa analogia, **useRef** é como aquele caderno mágico. Ele permite que você mantenha uma nota persistente (ou referência) sem alterar o comportamento do seu componente.

useRef

Necessidade

- Antes dos Hooks, se você quisesse acessar um elemento DOM ou manter algum valor mutável sem causar re-renderizações, você tinha que usar componentes de classe e lidar com código mais complexo. Com **useRef**, você pode fazer isso em componentes funcionais facilmente.
- O **useRef** é útil para:
 - Acessando um elemento DOM diretamente.
 - Armazenar um valor que não precisa acionar novas renderizações quando atualizado.
 - Manter o controle de um valor anterior para comparar com o atual.
 - Em resumo, **useRef** fornece uma maneira de interagir com elementos e valores diretamente, sem afetar a renderização do componente.

useRef

Uso básico

Primeiro, no topo do arquivo, **importamos** o useRef

Dentro do componente funcional **criamos o objeto ref.**

- **inputRef**: objeto ref. Usado para armazenar uma referência ao elemento de entrada.
- **useRef(null)**: criar um objeto ref que começa como null.

Dentro da função **handleClick**, acessamos o elemento de entrada usando **inputRef.current**.

A chamada **inputRef.current.focus()** focaliza o campo de entrada quando o botão é clicado.

Anexamos a referência ao elemento de entrada usando o atributo **ref**.

Ao definir **ref={inputRef}**, o React vincula o objeto ref a esse elemento de entrada, permitindo-nos acessá-lo por meio de **.inputRef.current**.

```
import React, { useRef } from 'react';

function ExampleComponent() {
  // 1. Create a ref object
  const inputRef = useRef(null);

  const handleClick = () => {
    // 2. Access the DOM element
    inputRef.current.focus();
  };

  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={handleClick}>Focus the input</button>
    </div>
  );
}

export default ExampleComponent;
```

useRef

O que useRef realmente faz?

- Vamos examinar o que está acontecendo:
- **Valor inicial:** quando o componente é carregado pela primeira vez, `useRef(null)` cria um objeto de referência com um valor inicial de `null`.
- **Acessando elementos:** Uma vez que o componente é renderizado, o React atribui o objeto ref ao elemento DOM. Você pode então usar `inputRef.current` para interagir com o elemento.
- **Referência estável:** O objeto ref não muda entre renderizações. Ele sempre aponta para o mesmo elemento ou valor DOM.
- É assim que `useRef` ajuda a controlar elementos ou valores sem causar novas renderizações desnecessárias.

useRef

Coisas importantes para saber sobre useRef

- **Ref não é para re-renderização:** Alterar o valor ref não faz com que seu componente seja re-renderizado. É puramente para acessar elementos ou armazenar valores sem afetar a UI.
- **O objeto ref persiste:** O objeto ref persiste por toda a vida útil do componente. Ele não é redefinido em re-renderizações.
- **Use com cautela:** Use refs somente quando precisar interagir com o DOM diretamente ou armazenar um valor mutável. Para outras necessidades de estado, prefira usar **useState**.

useEffect

Exemplos

Acessando um elemento DOM

inputRef contém uma referência ao elemento de entrada, permitindo que você o foque programaticamente.



```
const inputRef = useRef(null);  
  
<input ref={inputRef} type="text" />  
<button onClick={() => inputRef.current.focus()}>Focus Input</button>
```


useEffect

Exemplos

Armazenando valores mutáveis

`countRef` mantém o controle de um valor de contagem que pode ser atualizado sem causar novas renderizações.



```
const countRef = useRef(0);

const increment = () => {
  countRef.current += 1;
  console.log(countRef.current);
};
```

Hook useContext no React

useContext

O que é?

- Imagine que você tem uma família grande e quer contar a todos o que tem para o jantar. Em vez de ir até cada membro da família um por um, não seria mais fácil fazer um anúncio que todos pudessem ouvir? É isso que **useContext** faz no React!
- **useContext** é como um alto-falante que permite que você compartilhe informações com muitas partes do seu aplicativo React ao mesmo tempo.

useContext

Necessidade

- Digamos que você esteja criando um site para sua escola. Você quer mostrar o nome da escola em todas as páginas. Sem **useContext**, você teria que passar o nome da escola para cada parte do seu site. Com **useContext**, você pode anunciar o nome da escola uma vez, e cada parte do seu site pode ouvi-lo!

useContext

Uso básico

- Primeiro, precisamos criar nosso alto-falante. No React, chamamos isso de "Contexto".
 - Isso cria um locutor especial chamado **SchoolContext** que conterá o nome da nossa escola.
- Agora precisamos anunciar o nome da escola para nosso aplicativo. Fazemos isso com um "Provider".
 - Aqui, estamos anunciando o **schoolName** para todo o aplicativo. Cada parte do nosso aplicativo agora pode ouvir esse anúncio.
- Agora, em qualquer parte do nosso aplicativo, podemos ouvir este anúncio e obter o nome da escola.
 - Neste componente Header, estamos usando **useContext** para ouvir o nome da escola. Podemos então usar isso para mostrar o nome da escola em nosso cabeçalho.

```
import React from 'react';

// Create our loudspeaker
const SchoolContext = React.createContext();

function App() {
  // Our school name
  const schoolName = "Friendly Neighborhood School";

  return (
    // Announce our school name to the whole app
    <SchoolContext.Provider value={schoolName}>
      <div className="App">
        <Header />
        <MainContent />
        <Footer />
      </div>
    </SchoolContext.Provider>
  );
}

function Header() {
  // Listen to the school name announcement
  const schoolName =
    React.useContext(SchoolContext);
  return (
    <header>
      <h1>{schoolName}</h1>
    </header>
  );
}

function MainContent() {
  // We can also listen to the school name here
  const schoolName =
    React.useContext(SchoolContext);
  return (
    <main>
      <h2>Welcome to {schoolName}!</h2>
      <p>We're glad you're here.</p>
    </main>
  );
}

function Footer() {
  // And we can listen to it here too!
  const schoolName =
    React.useContext(SchoolContext);
  return (
    <footer>
      <p>© 2023 {schoolName}. All rights reserved.</p>
    </footer>
  );
}

export default App;
```

useRef

O que useContext realmente faz?

- Neste exemplo:
- Criamos o **SchoolContext** (alto-falante).
- Em nosso App, anunciamos o nome da nossa escola usando **SchoolContext.Provider**.
- Em nossos Header, MainContent, e Footer, usamos **useContext** para ouvir esse anúncio e obter o nome da escola.
- Lembre-se, **useContext** é ótimo para informações que muitas partes do seu aplicativo precisam saber. Ajuda a manter seu código simples e fácil de entender.

Exercícios

Exercícios

Instruções


- Realizar o exercício da aula 21
- Documentar o que foi feito no README.md
- A documentação deve ter a explicação de cada bloco de código do novo componente criado bem como a explicação do que é o novo componente
- Enviar o exercício roteiro da aula 21 para um repositório no GitHub
- Anexar link do repo na atividade do Teams


*“Ensinar é impregnar
de sentido o que
fazemos a cada
instante”*


Paulo Freire

Obrigado!

Thiago Nogueira

 [linkedin.com/tdn](https://www.linkedin.com/tdn)

 thiago.nogueira@pe.senac.br

 (81) 9 9627-0419