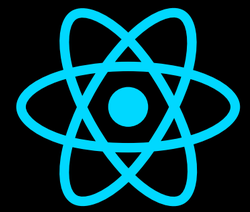
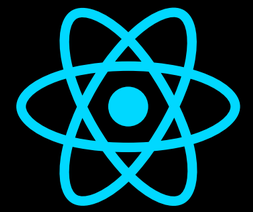


useEffect no React

NATHALIA MACEDO
DESENVOLVEDORA FULL STACK

O que iremos aprender?



- O que é o useEffect?
- Como funciona o useEffect?
- Quando usar o useEffect?
- Exemplo de uso
- O que é desmontar um componente?
- Limpeza de Efeitos
- Por que limpar o efeito?
- O que é setInterval?
- O que é clearInterval?
- Revisando conceitos
- Colocando em prática

O que é useEffect ?

Quando você está construindo um aplicativo React, você frequentemente precisa realizar ações que não estão diretamente relacionadas à renderização de componentes, tais como:

- Buscar dados de uma API
- Atualizar o DOM de acordo com certas condições
- Assinar eventos do navegador (como eventos de rolagem, redimensionamento, etc.)
- Executar operações de limpeza antes ou após o componente ser removido da tela

O que é useEffect ?

Estas são chamadas de "efeitos colaterais" porque estão fora do fluxo normal de renderização e atualização de componentes. O useEffect é uma ferramenta fornecida pelo React para lidar com esses efeitos colaterais em componentes funcionais.

Como Funciona o useEffect?

O useEffect é uma função que aceita dois argumentos:

Função de efeito: Esta é a função que será executada como um efeito colateral. Ela pode realizar qualquer ação necessária, como buscar dados, atualizar o DOM ou subscrever a eventos.

Como Funciona o useEffect?

Array de dependências (opcional): Este é um array opcional que especifica as dependências para o efeito. Se alguma das dependências mudar entre renderizações, a função de efeito será reexecutada. Se esse array estiver vazio, o efeito será executado apenas uma vez, após a montagem inicial do componente.

Suponha que você tenha um componente de contador e você quer exibir uma mensagem toda vez que o contador for atualizado. Vamos usar o useEffect para isso:

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

function Counter() {
  const [count, setCount] = useState(0);

  // Este efeito será executado após cada atualização de 'count'
  useEffect(() => {
    console.log("O contador foi atualizado:", count);
  }, [count]); // 'count' é especificado como uma dependência

  return (
    <div>
      <p>Contador: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Incrementar
      </button>
    </div>
  );
}

export default Counter;
```

Como Funciona o useEffect?

Agora, vamos explicar o código linha por linha:

1. Importamos **useState** e **useEffect** do React para podermos usá-los.
2. Definimos um componente funcional chamado **Counter**.
3. Criamos um estado **count** usando **useState**, que será o valor do contador.
4. Dentro do componente, usamos o **useEffect**. Esta função será executada após cada renderização do componente, porque não especificamos nenhuma dependência. Então, sempre que o componente for renderizado novamente, a função dentro do **useEffect** será executada.

Como Funciona o useEffect?

5.Dentro da função de efeito, usamos console.log para exibir uma mensagem indicando que o contador foi atualizado, e exibimos o valor atual do contador.

6. Passamos [count] como segundo argumento para o useEffect. Isso indica ao React que este efeito deve ser reexecutado sempre que o valor de count mudar entre renderizações. Se o valor de count não mudar, o efeito não será reexecutado.

7.No retorno do componente, renderizamos o valor do contador e um botão que permite incrementar o contador quando clicado.

Como Funciona o useEffect?

Então, basicamente, estamos usando o **useEffect** para realizar uma ação (no caso, exibir uma mensagem) sempre que o valor do contador for atualizado. E estamos especificando **count** como uma dependência, para que o efeito seja reexecutado apenas quando o valor de **count** mudar. Isso evita execuções desnecessárias da função de efeito.

Quando especificar as dependências?

As dependências devem ser especificadas quando você quer que o useEffect reaja a mudanças específicas em variáveis ou estados, e não a todas as renderizações do componente. Aqui estão algumas diretrizes para determinar quando usar dependências:

Quando especificar as dependências?

Quando o efeito depende de valores de estado ou propriedades: Se o efeito precisa ser reexecutado sempre que determinadas variáveis de estado ou propriedades mudarem, você deve incluir essas variáveis no array de dependências.

Evitar execuções desnecessárias: Se o efeito não precisa ser reexecutado sempre que o componente for renderizado novamente, especificar as dependências ajuda a evitar execuções desnecessárias da função de efeito, melhorando o desempenho do aplicativo.

Quando especificar as dependências?

Quando o efeito realiza operações assíncronas: Se o efeito realiza operações assíncronas (como buscar dados de uma API), é importante especificar quaisquer variáveis de estado ou propriedades usadas dentro do efeito como dependências. Isso garante que o efeito seja reexecutado quando essas variáveis mudarem, evitando que ele use valores desatualizados.

Evitar loops infinitos: Se você omitir as dependências ou incluir dependências incorretas, isso pode resultar em loops infinitos, onde o efeito é reexecutado continuamente. Especificar as dependências corretamente ajuda a evitar esse problema.

Quando especificar as dependências?

Em resumo, as dependências devem ser especificadas sempre que você precisar que o efeito seja reexecutado de forma seletiva, com base em mudanças específicas em variáveis ou estados do componente. Isso ajuda a garantir um comportamento previsível e eficiente do **useEffect** em seu aplicativo React

Quando usar o useEffect?

Você deve usar o useEffect sempre que precisar realizar uma ação que não é diretamente relacionada à renderização de um componente. Alguns exemplos comuns de quando usar useEffect incluem:

- Buscar dados de uma API assim que o componente é montado.
- Atualizar o DOM em resposta a mudanças de estado ou de propriedade.
- Subscrever a eventos do navegador, como cliques de mouse ou pressionamentos de tecla.
- Limpar recursos, como intervalos de tempo ou conexões de WebSocket, quando o componente é desmontado.

Exemplo de uso

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

function ExampleComponent() {
  // Definindo o estado inicial e uma função para atualizá-lo
  const [count, setCount] = useState(0);

  // Definindo um estado para armazenar os dados da API
  const [data, setData] = useState(null);
```

Exemplo de uso

```
// Este efeito será executado após a montagem inicial do componente
useEffect(() => {
  // Simula uma requisição assíncrona para uma API
  setTimeout(() => {
    // Atualiza o estado 'data' com os dados obtidos da API
    setData({ id: 1, name: 'Exemplo de Dados' });
  }, 1000);

  // Retorno da função de limpeza - será executada quando o componente for desmontado
  return () => {
    console.log('Componente desmontado'); // Mensagem de log para indicar a execução da limpeza
  };
}, []); // Array vazio indica que este efeito só será executado após a montagem inicial do componente
```

Exemplo de uso

```
// Este efeito será executado sempre que o estado 'count' for alterado
 useEffect(() => {
   console.log('O valor de count foi atualizado:', count); // Mensagem de log com o valor atual de 'count'
 }, [count]); // 'count' é especificado
 //como uma dependência

 // Função para manipular o estado
 // 'count' quando o botão é clicado
 const incrementCount = () => {
   setCount(prevCount => prevCount + 1);
 };
```

Exemplo de uso

```
// Renderização do componente
  return (
    <div>
      <h2>Exemplo de useEffect</h2>
      <p>Contador: {count}</p>
      <button onClick={incrementCount}>Incrementar</button>
      {/* Condicionalmente renderiza
       os dados obtidos da API */}
      {data && (
        <div>
          <h3>Dados da API:</h3>
          <p>ID: {data.id}</p>
          <p>Nome: {data.name}</p>
        </div>
      )}
    </div>
  );
}

export default ExampleComponent;
```

O que é desmontar um componente?

Em React, "desmontar o componente" se refere ao processo de remover um componente da árvore de elementos DOM e interromper qualquer atividade associada a ele. Isso geralmente ocorre quando um componente é removido da visualização, seja por meio de uma mudança de rota, mudança de estado que condiciona sua renderização ou destruição da aplicação.

Quando um componente é desmontado, algumas ações podem ocorrer, dependendo da implementação:

O que é desmontar um componente?

1. Limpeza de recursos: O componente pode ter recursos que precisam ser liberados, como event listeners, timers ou subscriptions. Desmontar o componente é um momento ideal para limpar esses recursos, evitando vazamentos de memória ou comportamentos inesperados.
2. **Cancelamento de tarefas assíncronas**: Se o componente estiver realizando operações assíncronas, como buscar dados de uma API, é importante cancelar ou limpar essas tarefas para evitar que elas continuem rodando desnecessariamente após o componente ser removido da tela.
3. **Interromper animações ou transições**: Se o componente estiver atualmente em processo de animação ou transição, desmontá-lo interromperá esses efeitos visuais para evitar problemas de renderização ou desempenho.

O que é desmontar um componente?

Em resumo, desmontar o componente é o processo de remover o componente do DOM e realizar quaisquer ações necessárias para liberar recursos e interromper atividades associadas a ele.

Limpeza de Efeitos

Limpar o efeito em React é uma prática importante para garantir que não haja vazamentos de memória ou comportamentos inesperados quando o componente é desmontado ou quando ocorrem mudanças nas dependências do efeito.

Quando você define um efeito usando useEffect, é possível retornar uma função dentro do corpo do efeito. Essa função será executada quando:

1. O componente é desmontado.
2. Antes de executar o próximo efeito, se houver uma alteração nas dependências.

Por que limpar o efeito?

Por que limpar um efeito? Bem, às vezes, os efeitos que você cria podem estar manipulando recursos externos, como assinaturas de eventos, temporizadores ou solicitações de rede. Se você não limpar esses recursos quando o componente é desmontado, eles podem continuar em execução desnecessariamente, consumindo recursos e potencialmente causando vazamentos de memória.

Vamos ver um exemplo simples:

Por que limpar o efeito?

```
//Fazendo a importação dos hooks useState e useEffect
import React, { useState, useEffect } from 'react';
//Criando um componente chamado MeuComponente
function MeuComponente() {
  //criando a variável contador, e a função de
  // atualização do valor dessa variável.
  // Inicialmente começa como 0
  const [contador, setContador] = useState(0);
```

Por que limpar o efeito?

```
useEffect(() => {
    //Mostrando uma mensagem no console
    console.log('Efeito executado');
    // criando um temporizador usando setInterval.
    //A função será chamada a cada 1000 milisegundos ou 1 segundo
    const intervalId = setInterval(() => {
      //a cada segundo, o valor de prevContador é aumentado em 1
      setContador((prevContador) => prevContador + 1);
      //o 1000 é para definir quanto tempo levará
    }, 1000);
```

Mas o que é setInterval?

- setInterval é uma função que executa um determinado bloco de código repetidamente em intervalos regulares de tempo especificados, até que seja explicitamente cancelado.
- A sintaxe básica é setInterval(função, intervalo), onde:
  - função: é a função que você deseja executar repetidamente.
  - intervalo: é o tempo, em milissegundos, entre cada execução da função.
- Por exemplo:

Mas o que é setInterval?

```
const intervalId = setInterval(() => {
      console.log('Este log será impresso a cada segundo');
    }, 1000);
```

```
 useEffect(() => {
    //Mostrando uma mensagem no console
    console.log('Efeito executado');
    // criando um temporizador usando setInterval.
    //A função será chamada a cada 1000 milisegundos ou 1 segundo
    const intervalId = setInterval(() => {
      //a cada segundo, o valor de prevContador é aumentado em 1
      setContador((prevContador) => prevContador + 1);
      //o 1000 é para definir quanto tempo levará
    }, 1000);

    return () => {
      console.log('Limpeza do efeito');
      clearInterval(intervalId);
    };
  }, []); // A dependência está vazia, o efeito só é executado uma vez

  return (
    <div>
      Contador: {contador}
    </div>
  );
}

export default MeuComponente;
```

Mas o que é clearInterval?

- clearInterval é uma função que cancela um temporizador configurado com setInterval.
- A sintaxe é simples: clearInterval(identificador), onde identificador é o valor retornado por setInterval.
- Quando você chama clearInterval com o identificador do intervalo que deseja cancelar, esse intervalo deixa de ser executado.
- Por exemplo:

Mas o que é clearInterval?

```
1  const intervalId = setInterval(() => {
2    console.log('Este log será impresso a cada segundo');
3  }, 1000);
4
5  // Depois de 5 segundos, cancelamos o intervalo
6  setTimeout(() => {
7    clearInterval(intervalId);
8    console.log('Intervalo cancelado após 5 segundos');
9  }, 5000);
```

setInterval e clearInterval

Em resumo, setInterval é usado para executar uma função repetidamente em intervalos regulares de tempo, enquanto clearInterval é usado para cancelar essas execuções periódicas, impedindo que a função continue sendo executada

Sobre a limpeza de efeito...

Então, em resumo, limpar o efeito é uma maneira de garantir que quaisquer recursos externos ou efeitos secundários criados pelo efeito sejam adequadamente desativados ou limpos quando o componente for desmontado ou quando as dependências do efeito mudarem. Isso ajuda a manter o aplicativo eficiente e livre de vazamentos de memória.

Revisando Conceitos

O que é useEffect?

O useEffect é um gancho (hook) do React que permite executar código em componentes funcionais em resposta a mudanças no ciclo de vida do componente, como após a renderização inicial, após atualizações ou antes da remoção do componente. Ele é usado para lidar com efeitos colaterais em componentes funcionais, como chamadas de API, manipulação do DOM ou configuração e limpeza de temporizadores.

Revisando Conceitos

Como funciona o useEffect?

O useEffect é uma função do React que permite realizar efeitos colaterais em componentes funcionais. Ele recebe dois argumentos:

1. Função de Efeito: O primeiro argumento é uma função que contém o código que você deseja executar como efeito colateral. Este código será executado após cada renderização do componente.
2. Array de Dependências: O segundo argumento é um array opcional de dependências. Se fornecido, o efeito só será reexecutado se uma ou mais das dependências listadas no array tiverem mudado desde a última renderização. Se o array estiver vazio, o efeito será executado apenas uma vez após a montagem do componente e não será reexecutado novamente.

Revisando Conceitos

Quando devemos especificar as dependências?

Você deve especificar as dependências no array de dependências do useEffect quando o efeito depende de valores que podem mudar ao longo do tempo e você deseja que o efeito seja reexecutado apenas quando esses valores mudarem. Isso ajuda a evitar execuções desnecessárias do efeito e otimiza o desempenho do seu aplicativo, garantindo que o código seja executado apenas quando necessário.

Revisando Conceitos

O que é desmontar um componente?

Desmontar um componente em React significa remover o componente da árvore de elementos do DOM. Isso acontece quando o componente não é mais necessário e deve ser removido da interface do usuário. Geralmente, isso ocorre quando o usuário navega para outra parte da aplicação ou quando uma condição específica é atendida para ocultar o componente. Quando um componente é desmontado, todos os recursos associados a ele, como temporizadores ou assinaturas de eventos, são limpos para evitar vazamentos de memória e garantir um comportamento adequado do aplicativo. Em resumo, desmontar um componente é o processo de removê-lo da interface do usuário e limpar qualquer estado ou recursos associados a ele.

Revisando Conceitos

O que é limpar um efeito?

Limpar um efeito em React é garantir que quaisquer ações ou recursos iniciados pelo efeito sejam desfeitos quando o componente é desmontado ou quando ocorrem mudanças específicas nas dependências do efeito. Isso é feito retornando uma função de limpeza dentro do próprio efeito, que é executada antes do próximo efeito ou quando o componente é removido da interface do usuário. Essa limpeza é importante para evitar vazamentos de memória e manter o comportamento correto do aplicativo. Em poucas palavras, limpar um efeito é garantir que tudo seja deixado em ordem quando o efeito não for mais necessário.

Revisando Conceitos

Por que devemos limpar um efeito?

Devemos limpar o efeito em React para garantir que quaisquer recursos alocados ou efeitos colaterais iniciados pelo efeito sejam desfeitos quando o componente é desmontado ou quando ocorrem mudanças específicas nas dependências do efeito. Isso ajuda a evitar vazamentos de memória e garante um comportamento consistente e eficiente do aplicativo, mantendo-o em um estado controlado e previsível. Em resumo, limpar o efeito é uma prática importante para manter a integridade do aplicativo e evitar problemas de desempenho ou comportamento inesperado.

Revisando Conceitos

Como funciona a função setInterval?

setInterval é uma função que executa um determinado bloco de código repetidamente em intervalos regulares de tempo especificados, até que seja explicitamente cancelado.

Recebe dois parâmetros: Função e intervalo. A função é o que desejamos executar de forma repetida. O intervalo é de quanto em quanto tempo o que foi determinado na função, deve se repetir!

Revisando Conceitos

Como funciona a função clearInterval?

clearInterval é uma função que cancela um temporizador configurado com setInterval.

Recebe apenas um parâmetro, que é o identificador! Quando você chama clearInterval com o identificador do intervalo que deseja cancelar, esse intervalo deixa de ser executado.

Colocando em prática

Imagine que você está criando um jogo de cronômetro simples. Você precisa criar um componente de cronômetro que conte o tempo em segundos desde que o usuário começou a contagem. Este componente deve ter um botão "Iniciar" que inicia a contagem, e um botão "Parar" que interrompe a contagem.
Implemente este componente usando useState, useEffect, setInterval e clearInterval. Certifique-se de limpar o temporizador quando o componente for desmontado.
Dica: Use um estado para rastrear se o cronômetro está ativo ou não.

Colocando em prática

1. Faça a importação do useEffect e do useState from react
2. Crie um componente funcional Cronômetro
3. Crie a variável segundos e a função setSegundos, começa com o valor 0
4. Crie uma variável cronometroAtivo e a função setCronometroAtivo, inciando com o valor false.
5. Faça uso do Hook useEffect e crie a condição: Se cronometroAtivo for verdadeiro, use a função setInterval para adicionar mais um ao valor de segundos.
6. Crie a função de limpeza do efeito usando o clearInterval. Isso é o que fica no return do useEffect. E lembre-se, o useEffect recebe um array de dependências nesse caso.

Colocando em prática

7. Em seguida crie duas funções, uma para iniciar o cronometro: ela muda o valor de cronometroAtivo, de false, para true. E uma para parar o cronômetro, que precisa mudar o valor de cronometroAtivo de true, para false.
8. Renderize seu