

DOM e Javascript

Fonte: Documentação oficial [React](#) e [Next JS](#)

Atualizando a IU com métodos JavaScript e DOM

Vamos ver como você pode usar métodos JavaScript e DOM adicionando uma h1tag ao seu projeto.

Abra seu editor de código e crie um novo index.htmlarquivo. Dentro do arquivo HTML, adicione o seguinte código:

```
<!-- index.html -->  
<html>  
  <body>  
    <div></div>  
  </body>  
</html>
```

Em seguida, dê ao div um id exclusivo para que você possa direcioná-lo mais tarde.

```
<!-- index.html -->  
<html>  
  <body>  
    <div id="app"></div>  
  </body>  
</html>
```

Para escrever JavaScript dentro do seu arquivo HTML, adicione uma scripttag:

```
<!-- index.html -->  
<html>  
  <body>  
    <div id="app"></div>  
    <script type="text/javascript"></script>  
  </body>  
</html>
```

Agora, dentro da scripttag, você pode usar um método DOM, `getElementById()`, para selecionar o elemento `<div>` pelo seu id:

```
<!-- index.html -->
<html>
  <body>
    <div id="app"></div>

    <script type="text/javascript">
      const app = document.getElementById('app');
    </script>
  </body>
</html>
```

Você pode continuar usando métodos DOM para criar um novo <h1>elemento:

```
<!-- index.html -->
<html>
  <body>
    <div id="app"></div>


    <script type="text/javascript">
      // Select the div element with 'app' id
      const app = document.getElementById('app');

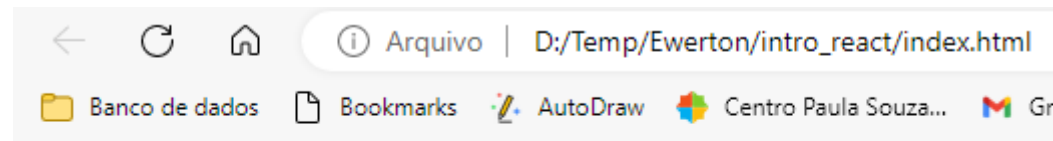
      // Create a new H1 element
      const header = document.createElement('h1');
```

```
      // Create a new text node for the H1 element
      const headerContent = document.createTextNode(
        'Develop. Preview. Ship. 🚀',
      );

      // Append the text to the H1 element
      header.appendChild(headerContent);

      // Place the H1 element inside the div
      app.appendChild(header);
    </script>
  </body>
</html>
```

Para garantir que tudo esteja funcionando, abra seu arquivo HTML no navegador de sua preferência. Você deve ver uma tag h1 que diz: 'Develop. Preview. Ship. '.



Develop. Preview. Ship. 

HTML x DOM

Se você observar os elementos DOM nas ferramentas de desenvolvedor do navegador, notará que o DOM inclui o elemento `<h1>`. O DOM da página é diferente do código-fonte - ou seja, do arquivo HTML original que você criou.

DOM

Elements

```
1 <html>
2   <head></head>
3   <body>
4     <div id="app">
5       <h1>Develop. Preview.
6 Ship. 🚀</h1>
7     </div>
8     <script type="text/
9 javascript">...</script>
10   </body>
11 </html>
```

SOURCE CODE (HTML)

index.html

```
1 <html>
2   <head></head>
3   <body>
4     <div id="app"></div>
5     <script type="text/
6 javascript">...</script>
7   </body>
8 </html>
9
10
11
```


Isso ocorre porque o HTML representa o conteúdo da página inicial , enquanto o DOM representa o conteúdo da página atualizado que foi alterado pelo código JavaScript que você escreveu.

Atualizar o DOM com JavaScript simples é muito poderoso, mas detalhado. Você escreveu todo esse código para adicionar um `<h1>` elemento com algum texto:

```
<!-- index.html -->
<script type="text/javascript">
  const app = document.getElementById('app');
  const header = document.createElement('h1');
  const headerContent = document.createTextNode('Develop. Preview. Ship. 🚀');
  header.appendChild(headerContent);
  app.appendChild(header);
</script>
```

À medida que o tamanho de um aplicativo ou equipe cresce, pode se tornar cada vez mais desafiador criar aplicativos dessa maneira.

Com essa abordagem, os desenvolvedores gastam muito tempo escrevendo instruções para dizer ao computador como ele deve fazer as coisas. Mas não seria legal descrever o que você quer mostrar e deixar o computador descobrir como atualizar o DOM?

Programação Imperativa x Programação Declarativa

O código acima é um bom exemplo de programação imperativa . Você está escrevendo as etapas de como a interface do usuário deve ser atualizada. Mas quando se trata de construir interfaces de usuário, uma abordagem declarativa é frequentemente preferida porque pode acelerar o processo de desenvolvimento. Em vez de escrever métodos DOM, seria útil se os desenvolvedores pudessem declarar o que desejam mostrar (neste caso, uma tag h1 com algum texto).

Programação Imperativa x Programação Declarativa

Em outras palavras, a programação imperativa é como dar a um chef instruções passo a passo sobre como fazer uma pizza. A programação declarativa é como pedir uma pizza sem se preocupar com as etapas necessárias para prepará-la.

Uma biblioteca declarativa popular que ajuda os desenvolvedores a criar interfaces de usuário é o React .

React: uma biblioteca de interface do usuário declarativa

Como desenvolvedor, você pode dizer ao React o que deseja que aconteça com a interface do usuário, e o React descobrirá as etapas de como atualizar o DOM em seu nome.

Qual das seguintes declarações é mais declarativa?

- a) "Tricotar a massa, enrolar a massa, adicionar molho de tomate, adicionar queijo, adicionar presunto, adicionar abacaxi, assar a 200 graus Celsius em forno de pedra por..."
- b) "Uma pizza havaiana, por favor."

Qual das seguintes declarações é mais declarativa?

- a) "Tricotar a massa, enrolar a massa, adicionar molho de tomate, adicionar queijo, adicionar presunto, adicionar abacaxi, assar a 200 graus Celsius em forno de pedra por..."
- b) "Uma pizza havaiana, por favor."

Aprenda mais em:

[HTML X DOM](#)

[Como a IU declarativa se compara à imperativa](#)

Introdução ao React

Para usar React em seu projeto, você pode carregar dois scripts React de um site externo chamado unpkg.com :

- react é a biblioteca principal do React.
- react-dom fornece métodos específicos do DOM que permitem que você use React com o DOM.


```
<!-- index.html -->
<html>
  <body>
    <div id="app"></div>

    <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>

    <script type="text/javascript">
      const app = document.getElementById('app');
    </script>
  </body>
</html>
```

Em vez de manipular diretamente o DOM com JavaScript simples, você pode usar o método ReactDOM.render() do react-dom para dizer ao React para renderizar nosso título <h1> dentro de nosso elemento #app.

```
<!-- index.html -->
<html>
  <body>
    <div id="app"></div>

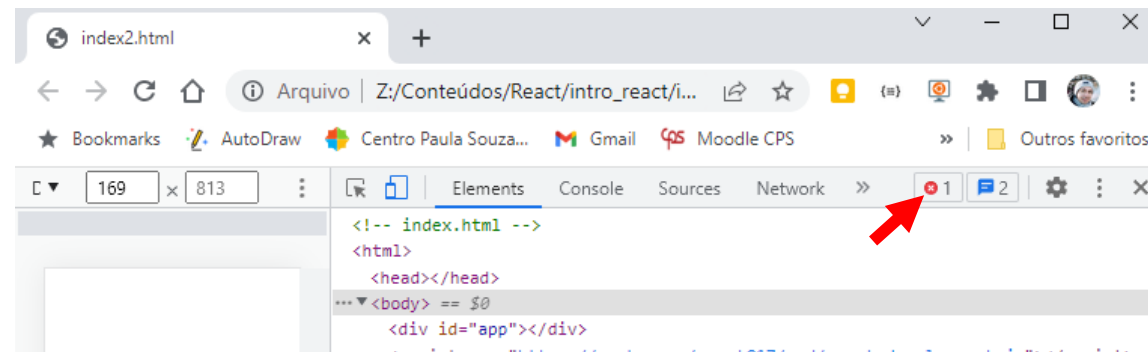
    <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>

    <script type="text/javascript">
      const app = document.getElementById('app');
      ReactDOM.render(<h1>Develop. Preview. Ship. 🚀 </h1>, app);
    </script>
  </body>
</html>
```

Mas se você tentar executar este código no navegador, receberá um erro de sintaxe:

✖ Uncaught SyntaxError: Unexpected token '<' index4.html:15

Isso ocorre porque `<h1>...</h1>` não é um Javascript válido. Este pedaço de código é JSX.



O que é JSX?

JSX é uma extensão de sintaxe para JavaScript que permite que você descreva sua IU em uma sintaxe semelhante a HTML . O bom do JSX é que, além de seguir as três regras do JSX , você não precisa aprender nenhum novo símbolo ou sintaxe fora do HTML e do JavaScript.

Observe que os navegadores não entendem JSX imediatamente, então você precisará de um compilador JavaScript, como um Babel, para transformar seu código JSX em JavaScript normal.

Adicionando o Babel ao seu projeto

Para adicionar Babel ao seu projeto, copie e cole o seguinte script em seu arquivo index.html:

```
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

Além disso, você precisará informar ao Babel qual código transformar alterando o tipo de script para type=text/jsx.

```
<html>
  <body>
    <div id="app"></div>
    <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>

    <!-- Babel Script -->
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/jsx">
      const app = document.getElementById('app');
      ReactDOM.render(<h1>Develop. Preview. Ship. 🚀 </h1>, app);
    </script>
  </body>
</html>
```

Você pode executar seu código no navegador para confirmar se está funcionando corretamente.

Comparando o código React declarativo que você acabou de escrever:

```
<script type="text/jsx">
  const app = document.getElementById('app');
  ReactDOM.render(<h1>Develop. Preview. Ship. 🚀</h1>, app);
</script>
```

ao código JavaScript imperativo que você escreveu na seção anterior:

```
<script type="text/javascript">
  const app = document.getElementById('app');
  const header = document.createElement('h1');
  const headerContent = document.createTextNode('Develop. Preview. Ship. 🚀');
  header.appendChild(headerContent);
  app.appendChild(header);
</script>
```

Você pode começar a ver como, usando o React, pode reduzir muito código repetitivo.

E é exatamente isso que o React faz, é uma biblioteca que contém trechos de código reutilizáveis que executam tarefas em seu nome - neste caso, atualizando a interface do usuário.

Nota: Você não precisa saber exatamente como o React atualiza a interface do usuário para começar a usá-lo, mas se quiser saber mais, dê uma olhada nas [árvores da interface](#) do usuário e nas seções [react-dom/server](#) na documentação do React.

Por que você precisa compilar seu código React?

- a) O React usa uma nova versão do HTML que é muito avançada para os navegadores atuais.
- b) React usa JSX que precisa ser compilado em JavaScript.
- c) O React não sabe como atualizar o DOM, então precisa de um compilador para fazer isso.

Por que você precisa compilar seu código React?

- a) O React usa uma nova versão do HTML que é muito avançada para os navegadores atuais.
- b) React usa JSX que precisa ser compilado em JavaScript.
- c) O React não sabe como atualizar o DOM, então precisa de um compilador para fazer isso.

Aprenda mais em:

[Escrevendo marcação com JSX](#)

JavaScript essencial para React

Embora você possa aprender JavaScript e React ao mesmo tempo, estar familiarizado com JavaScript pode facilitar o processo de aprendizagem do React. A seguir seguem os conceitos mais utilizados:

- [Functions](#) and [Arrow Functions](#)
- [Objects](#)
- [Arrays and array methods](#)
- [Destructuring](#)
- [Template literals](#)
- [Ternary Operators](#)
- [ES Modules and Import / Export Syntax](#)