Introduzindo JSX

Considere esta declaração de variável:

```
const element = <h1>Hello, world!</h1>;
```

Esta sintaxe estranha de tags não é uma string, nem HTML.

É chamada JSX e é uma extensão de sintaxe para JavaScript. Recomendamos usar JSX com o React para descrever como a UI deveria parecer. JSX pode lembrar uma linguagem de template, mas que vem com todo o poder do JavaScript.

JSX produz "elementos" do React. Nós iremos explorar a renderização para o DOM na próxima seção. Abaixo você descobrirá o básico de JSX necessário para começar.

Por que JSX?

O React adota o fato de que a lógica de renderização é inerentemente acoplada com outras lógicas de UI: como eventos são manipulados, como o state muda com o tempo e como os dados são preparados para exibição.

Ao invés de separar *tecnologias* artificialmente colocando markup e lógica em arquivos separados, o React separa *conceitos* com unidades pouco acopladas chamadas "componentes" que contém ambos. Voltaremos aos componentes em outra seção. Mas, se você ainda não está confortável em usar markup em JS, esta palestra pode convencer você do contrário.

O React não requer o uso do JSX. Porém, a maioria das pessoas acha prático como uma ajuda visual quando se está trabalhando com uma UI dentro do código em JavaScript. Ele permite ao React mostrar mensagens mais úteis de erro e aviso.

Com isso fora do caminho, vamos começar!

Incorporando Expressões em JSX

No exemplo abaixo, declaramos uma variável chamada `name` e então a usamos dentro do JSX ao envolvê-la com chaves:

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Você pode inserir qualquer expressão JavaScript válida dentro das chaves em JSX. Por exemplo, `2 + 2`, `user.firstName`, ou `formatName(user)` são todas expressões JavaScript válidas.

No exemplo abaixo, incorporamos o resultado da chamada de uma função JavaScript, `formatName(user)`, dentro de um elemento `<h1>`.

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Try it on CodePen

Separamos o JSX em múltiplas linhas para melhorar a legibilidade. Mesmo que não seja obrigatório, quando fizer isso, também recomendamos colocar dentro de parênteses para evitar as armadilhas da inserção automática de ponto-e-vírgula.

JSX Também é uma Expressão

Depois da compilação, as expressões em JSX se transformam em chamadas normais de funções que retornam objetos JavaScript.

Isto significa que você pode usar JSX dentro de condições `if` e laços `for`, atribuí-lo a variáveis, aceitá-lo como argumentos e retorná-los de funções:

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

Especificando Atributos com JSX

Você pode usar aspas para especificar strings literais como atributos:

```
const element = <div tabIndex="0"></div>;
```

Você também pode usar chaves para incorporar uma expressão JavaScript em um atributo:

```
const element = <img src={user.avatarUrl}></img>;
```

Não envolva chaves com aspas quando estiver incorporando uma expressão JavaScript em um atributo. Você deveria ou usar aspas (para valores em string) ou chaves (para expressões), mas não ambos no mesmo atributo.

> **Atenção:**
> Como JSX é mais próximo de JavaScript que do HTML, o React DOM usa `camelCase` como convenção para nomes de propriedades ao invés dos nomes de atributos do HT
>
> Por exemplo, `class` se transforma em `className` em JSX, e `tabindex` se transforma em `tabIndex`.

Especificando Elementos Filhos com JSX

Se uma tag está vazia, você pode fechá-la imediatamente com `/>`, como XML:

```
const element = <img src={user.avatarUrl} />;
```

Tags JSX podem conter elementos filhos:

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

JSX Previne Ataques de Injeção

É seguro incorporar entradas de usuário em JSX:

```
const title = response.potentiallyMaliciousInput;
// This is safe:
const element = <h1>{title}</h1>;
```

Por padrão, o React DOM escapa quaisquer valores incorporados no JSX antes de renderizá-los. Assim, assegura que você nunca injete algo que não esteja explicitamente escrito na sua aplicação. Tudo é convertido para string antes de ser renderizado. Isso ajuda a prevenir ataques XSS (cross-site-scripting).

JSX Representa Objetos

O Babel compila JSX para chamadas `React.createElement()`.

Estes dois exemplos são idênticos:

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

`React.createElement()` realiza algumas verificações para ajudar você a criar um código sem bugs, mas, essencialmente, cria um objeto como este:

```
// Nota: esta estrutura está simplificada
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

Estes objetos são chamados “Elementos React”. Você pode imaginá-los como descrições do que você quer ver na tela. O React lê esses objetos e os usa para construir o DOM e deixá-lo atualizado.

Exploraremos a renderização de elementos React no DOM na próxima seção.

> **Dica:**
> Recomendamos o uso da definição de linguagem “Babel” no seu editor preferido para que ambos os códigos em ES6 e JSX sejam devidamente realçados. Este website usa o esquema de cores Oceanic Next, no qual é compatível com o Babel.

Edite esta página

Artigo anterior
Hello World

Artigo seguinte
Renderizando Elementos