

Usando o State do Hook

Hooks são uma nova adição ao React 16.8. Eles permitem que você use o state e outros recursos do React sem escrever uma classe.

A [página de introdução](#) usou este exemplo para familiarizar com Hooks:

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

function Example() {
  // Declarar uma nova variável de state, na qual chamaremos de "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Clique aqui
      </button>
    </div>
  );
}
```

Vamos começar a aprender sobre Hooks comparando este código com um exemplo equivalente utilizando classe.

Exemplo Equivalente com Classe

Se você já usou classes no React, este código deve parecer familiar:

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  },
```



```

    }

    render() {
      return (
        <div>
          <p>Você clicou {this.state.count} vezes</p>
          <button onClick={() => this.setState({ count: this.state.count + 1
        }}}>
            Clique aqui
          </button>
        </div>
      );
    }
  }
}

```

O state começa como `{ count: 0 }`, e aumentamos o `state.count` chamando `this.setState()` quando o usuário clica no botão. Vamos utilizar trechos dessa classe ao longo da página.

Nota

Você pode estar se perguntando porque estamos usando um counter aqui ao invés de um exemplo mais realista. Isto é pra nos ajudar a focar na API enquanto ainda damos os primeiros passos com Hooks.

Hooks e Componentes de Função

Para lembrar, componentes de função, no React, se parecem com isto:

```

const Example = (props) => {
  // Você pode usar Hooks aqui!
  return <div />;
}

```

ou isto:

```

function Example(props) {
  // Você pode usar Hooks aqui!
  return <div />.
}

```



```
    return null;
}
```

Você pode ter conhecido estes exemplos como “componentes sem estado”. Nós estamos introduzindo a habilidade de utilizar o state do React neles, portanto preferimos o nome “componentes de função”.

Hooks **não** funcionam dentro de classes. Mas você pode usá-los em vez de escrever classes.

O que é um Hook?

Nosso novo exemplo começa importando o `useState` Hook do React:

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

function Example() {
  // ...
}
```

O que é um Hook? Um Hook é uma função especial que te permite utilizar recursos do React. Por exemplo, `useState` é um Hook que te permite adicionar o state do React a um componente de função. Vamos aprender outros Hooks mais tarde.

Quando eu deveria usar um Hook? Se você escreve um componente de função e percebe que precisa adicionar algum state para ele, anteriormente você tinha que convertê-lo para uma classe. Agora você pode usar um Hook dentro de um componente de função existente. Vamos fazer isso agora mesmo!

Nota:

Existem algumas regras especiais sobre onde você pode ou não utilizar Hooks dentro de um componente. Vamos aprender elas nas [Regras dos Hooks](#).



Declarando uma Variável State

Em uma classe, inicializamos o state `count` para `0` definindo `this.state` para `{ count: 0 }` no construtor:

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

Em um componente de função, não temos `this`, portanto não podemos definir ou ler `this.state`. Em vez disso, chamamos o Hook `useState` dentro do nosso component:

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

function Example() {
  // Declarar uma nova variável de state, na qual chamaremos de "count"
  const [count, setCount] = useState(0);
}
```

O que o `useState` faz? Ele declara um variável state. Nossa variável é chamada de `count` mas poderíamos chamar de qualquer coisa, como `banana`. Esta é uma maneira de “preservar” alguns valores entre as chamadas de funções — `useState` é uma nova maneira de usar as mesmas capacidades que o `this.state` tem em uma classe. Normalmente, variáveis “desaparecem” quando a função sai mas variáveis de state são preservadas pelo React.

O que passamos para o `useState` como argumento? O único argumento para o Hook `useState()` é o state inicial. Diferente de classes, o state não tem que ser um objeto. Podemos manter um número ou uma string se for tudo que precisamos. No nosso exemplo, apenas queremos um número para quantas vezes o usuário clicou, então passamos `0` como state inicial para nossa variável. (Se quiséssemos guardar dois valores diferentes no state, chamaríamos `useState()` duas vezes.)

O que `useState` retorna? Ele retorna um par de valores: o state atual e uma função que atualiza o state. É por isso que escrevemos `const [count, setCount] = useState()`. Isto é similar ao `this.state.count` e `this.setState` em uma classe, exceto o fato de



pegá-los em par. Se você não está familiarizado com a sintaxe que usamos, vamos voltar nisso no final dessa página.

Agora que sabemos o que o Hook `useState` faz, nosso exemplo deve fazer mais sentido:

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

function Example() {
  // Declarar uma nova variável de state, na qual chamaremos de "count"
  const [count, setCount] = useState(0);
```

Nós declaramos uma variável state chamada `count` e definimos ela para 0. O React vai lembrar o valor atual entre cada re-renderização e fornecer o valor mais recente para nossa função. Se quisermos atualizar o `count` atual, podemos chamar `setCount`.

Nota

Você pode estar se perguntando: Por que é chamado `useState` ao invés de `createState`?

“Create” não seria muito preciso porque o state é criado apenas na primeira vez que nosso componente renderiza. Durante as próximas renderizações, `useState` nos dá o state atual. Caso contrário, não seria “state” de qualquer maneira! Também tem outro motivo pelo qual nomes de Hook sempre começam com `use`. Vamos aprender o porque depois, nas Regras dos Hooks.

Lendo o State

Quando queremos mostrar o `count` atual em classe, lemos `this.state.count`:

```
<p>Você clicou {this.state.count} vezes</p>
```

Em uma função, podemos usar `count` diretamente:



```
<p>Você clicou {count} vezes</p>
```

Atualizando o State

Em uma classe, podemos chamar `this.setState()` para atualizar o state `count`:

```
<button onClick={() => this.setState({ count: this.state.count + 1 })}>
  Clique aqui
</button>
```

Na função, já temos `setCount` e `count` como variáveis então não precisamos do `this`:

```
<button onClick={() => setCount(count + 1)}>
  Clique aqui
</button>
```

Recapitulação

Vamos **recapitular o que aprendemos linha por linha** e checar nosso entendimento.

```
1: import React, { useState } from 'react';
2:
3: function Example() {
4:   const [count, setCount] = useState(0);
5:
6:   return (
7:     <div>
8:       <p>Você clicou {count} vezes</p>
9:       <button onClick={() => setCount(count + 1)}>
10:        Clique aqui
11:      </button>
12:    </div>
13:  );
14: }
```



- **Linha 1:** Importamos o Hook `useState` do React. Ele nos permite manter o state local em um componente de função.
- **Linha 4:** Dentro do componente `Example`, declaramos uma nova variável de state chamando o Hook `useState`. Ele retorna um par de valores, no qual damos nomes. Estamos chamando nossa variável `count` porque ela mantém o número de cliques no botão. Inicializamos como zero passando `0` como o único argumento do `useState`. O segundo item retornado é a própria função. Ela nos permite atualizar o `count` então nomeamos para `setCount`.
- **Linha 9:** Quando o usuário clica, chamamos `setCount` com um novo valor. O React então vai re-renderizar o componente `Example`, passando o novo valor de `count` para ele.

À primeira vista pode parecer muita coisa. Não se apresse! Se você está perdido na explicação, olhe o código acima novamente e tente lê-lo de uma ponta a outra. Prometemos que a partir do momento que você “esquecer” como state funciona em classes e olhar este código com novos olhos irá fazer sentido.

Dica: O que os Colchetes Significam?

Você pode ter percebido os colchetes quando declaramos a variável state:

```
const [count, setCount] = useState(0);
```

Os nomes na esquerda não são parte da API do React. Você pode nomear suas próprias variáveis state:

```
const [fruit, setFruit] = useState('banana');
```

Esta sintaxe do JavaScript é chamada de “atribuição via desestruturação”. Significa que estamos fazendo duas novas variáveis `fruit` e `setFruit`, onde `fruit` é definido para o primeiro valor retornado por `useState`, e `setFruit` é o segundo. É equivalente a este código:

```
var fruitStateVariable = useState('banana'); // Retorna um par
var fruit = fruitStateVariable[0]; // Primeiro item do par
var setFruit = fruitStateVariable[1]; // Segundo item do par
```



Quando declaramos uma variável com `useState`, ela retorna um par — um array com dois itens. O primeiro item é o valor atual e o segundo é uma função que nos permite atualizá-la. Usar `[0]` e `[1]` para acessá-las é um pouco confuso porque elas tem um significado específico. É por isto que utilizamos atribuição via desestruturação no lugar.

Nota

Você pode estar curioso como o React sabe qual componente o `useState` corresponde já que não passamos nada como `this` para o React. Vamos responder [esta pergunta](#) e muitas outras na seção FAQ.

Dica: Usando Múltiplas Variáveis State

Declarar variáveis de state como par de `[something, setSomething]` também é útil porque nos permite dar *diferentes* nomes para diferentes variáveis de state se quiséssemos usar mais de uma:

```
function ExampleWithManyStates() {  
  // Declarar múltiplas variáveis de state!  
  const [age, setAge] = useState(42);  
  const [fruit, setFruit] = useState('banana');  
  const [todos, setTodos] = useState([{ text: 'Aprender Hooks' }]);  
}
```

No componente acima, temos `age`, `fruit` e `todos` como variáveis locais e podemos atualizá-las individualmente:

```
function handleOrangeClick() {  
  // Similar ao this.setState({ fruit: 'orange' })  
  setFruit('orange');  
}
```

Você **não tem que** usar muitas variáveis de state. Elas podem conter objetos e arrays muito bem. Portanto você ainda pode juntar dados relacionados. De qualquer maneira, diferentemente de `this.setState` em classe, ao atualizar uma variável de state, ela sempre é *substituída* em vez de incorporada.



Damos mais recomendações em separação de variáveis de state independentes [no FAQ](#).

Próximos Passos

Nesta página aprendemos sobre um dos Hooks fornecido pelo React, chamado `useState`. Também, em algumas vezes, vamos nos referir como o “State do Hook”. Ele nos permite adicionar um state local a um componente de função — o que fizemos pela primeira vez!

Também aprendemos um pouco mais sobre o que são Hooks. Hooks são funções que permitem que você utilize recursos do React em componentes de função. Seus nomes sempre começam com `use`, e existem mais Hooks que não vimos ainda.

Agora vamos continuar aprendendo o próximo Hook: `useEffect`. Ele permite que você execute efeitos colaterais em um componente e é similar ao método de ciclo de vida em classes.

Esta página é útil?  

[Edite esta página](#)

[Artigo anterior](#)

[Hooks de Forma Resumida](#)

[Artigo seguinte](#)

[Usando Effect Hook](#)

