React Hooks

FUTURE4

Definição

- React Hooks são uma **nova** adição ao React. Eles permitem que usemos estado e outras funcionalidades **sem escrever uma classe**.

- Lançados no início de 2019, ainda estão sendo adotados

- **Tudo que pode ser feito usando Hooks já podia ser feito antes, ele apenas introduz novas maneiras**

Motivação

1. É difícil reutilizar lógica de estado/lifecycle entre componentes

2. Componentes complexos ficam difíceis de entender

3. Classes são confusas

Motivação

1. **É difícil reutilizar lógica de estado/lifecycle entre componentes**

2. Componentes complexos ficam difíceis de entender

3. Classes são confusas

Exemplo

- Dois componentes precisam das informações de um usuário, vindo de uma requisição

1. Componente de informações do usuário

2. Componente de header, que mostra um botão condicionalmente se o usuário for administrador

Exemplo

```
1  class InfosUsuario extends Component {
2    constructor(props) {
3      super(props)
4
5      this.state = {
6        user: undefined
7      }
8    }
9
10   componentDidMount() {
11     fetchUser().then(user => this.setState({user}))
12   }
13
14   render() {
15     const {user} = this.state
16     return (
17       <div>
18         <p>Nome: {user.name}</p>
19         <p>Email: {user.email}</p>
20       </div>
21     )
22   }
23 }
```

```
1  class Header extends Component {
2    constructor(props) {
3      super(props)
4
5      this.state = {
6        user: {}
7      }
8    }
9
10   componentDidMount() {
11     fetchUser().then(user => this.setState({user}))
12   }
13
14   render() {
15     const {user} = this.state
16     return (
17       <div>
18         <p>Seja bem vindo, {user.name}</p>
19         {user.isAdmin && <button>Área do administrador</button>}
20       </div>
21     )
22   }
23 }
```

Como resolver hoje - HOC

- Higher Order Components
- Funções que recebem um componente e retornam um componente, adicionando lógica a ele
- Exemplo: *connect()(Component)*

Exemplo

```
1 const withUser = (Component) => {
2   return class extends React.Component {
3     constructor(props) {
4       super(props)
5       this.state = {
6         user: {}
7       }
8     }
9
10    componentDidMount() {
11      fetchUser().then(user => this.setState({user}))
12    }
13
14    render() {
15      const {user} = this.state
16      return <Component {...this.props} user={user}/>
17    }
18  }
19 }
```

Exemplo

```
class InfosUsuario extends Component {
  render() {
    const {user} = this.props
    return (
      <div>
        <p>Nome: {user.name}</p>
        <p>Email: {user.email}</p>
      </div>
    )
  }
}
export default withUser(InfosUsuario)
```

```
class Header extends Component {
  render() {
    const {user} = this.props
    return (
      <div>
        <p>Seja bem vindo, {user.name}</p>
        {user.isAdmin && <button>Área do administrador</button>}
      </div>
    )
  }
}
export default withUser(Header)
```

Como resolver hoje - Render Props

- Componente recebe uma prop “especial” chamada de Render Prop

- É uma função que renderiza um componente, passando quaisquer parâmetros relevantes

Exemplo

```
1 class User extends React.Component {
2   constructor(props) {
3     super(props)
4     this.state = {
5       user: {}
6     }
7   }
8
9   componentDidMount() {
10    fetchUser().then(user => this.setState({user}))
11  }
12
13  render() {
14    const {user} = this.state
15    return this.props.render(user)
16  }
17 }
```

Exemplo

```
class InfosUsuario extends Component {
  render() {
    return (
      <div>
        <User render={(user) => {
          return <div>
            <p>Nome: {user.name}</p>
            <p>Email: {user.email}</p>
          </div>
        }}/>

      </div>
    )
  }
}
```

```
class Header extends Component {
  render() {
    return (
      <div>
        <User render={(user) => {
          return <div>
            <p>Seja bem vindo, {user.name}</p>
            {user.isAdmin && <button>Área do administrador</button>}
          </div>
        }}/>
      </div>
    )
  }
}
```

Motivação

1. É difícil reutilizar lógica de estado/lifecycle entre componentes

2. **Componentes complexos ficam difíceis de entender**

3. Classes são confusas

Exemplo

- Requisição do usuário agora depende de um id, que pode mudar

- Precisamos tratar essa lógica no método *componentDidUpdate*, verificando se o id mudou

Exemplo

```
1 class User extends React.Component {
2   constructor(props) {
3     super(props)
4     this.state = {
5       user: {}
6     }
7   }
8
9   componentDidMount() {
10    fetchUser(this.props.userId).then(user => this.setState({user}))
11  }
12
13  componentDidUpdate(prevProps) {
14    if(prevProps.userId !== this.props.userId) {
15      fetchUser(this.props.userId).then(user => this.setState({user}))
16    }
17  }
18
19  render() {
20    const {user} = this.state
21    return this.props.render(user)
22  }
23 }
```

Motivação

1. É difícil reutilizar lógica de estado/lifecycle entre componentes

2. Componentes complexos ficam difíceis de entender

3. **Classes são confusas**

Classes são confusas...

- ... para nós e para o computador!
- Podem gerar algumas dificuldades no desenvolvimento e falhas de otimização
- Confundem o desenvolvedor quanto ao uso do *this*

Pausa

FUTURE4

Hooks

FUTURE4

Como funcionam

- Hooks são funções do React que são chamadas em **componentes funcionais**

- Eles **não funcionam** em componentes de classe, mas você pode substituir seus componentes de classe por componentes funcionais que usam hooks

- Dois hooks principais:
  - useState
  - useEffect

useState

- Recebe estado inicial e retorna um array com duas posições
  - A primeira é uma variável que recebe o estado atual
  - A segunda é uma função que define o estado atual
- Você pode ter vários *useState* por componente. Cada um deles pode guardar uma parte do seu estado

Exemplo

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

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

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

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

function Contador() {
  // Aqui, declaramos uma nova variável de estado, que chamamos de count
  // A função useState recebe o estado inicial, que nesse caso é 0
  const [count, setCount] = useState(0);

  // Declaração de uma função onClickButton, atribuída a uma constante
  const onClickButton = () => {
    // Chamada de função setCount (que foi retornada pelo useState)
    // Ela recebe o novo estado, e atualiza o componente
    setCount(count + 1)
  }

  return (
    <div>
      {/*Aqui, usamos a variável de estado count*/}
      {/*Ela contém o valor atual da variável, mesmo entre renderizações*/}
      <p>Você clicou {count} vezes</p>

      {/*Passamos a referência da função onClickButton*/}
      {/*para ser acionada no clique do botão*/}
      <button onClick={onClickButton}>
        Clique
      </button>
    </div>
  );
}
```

Exemplo

```
1 import React, { useState } from 'react';
2
3 function Exemplo() {
4   const [age, setAge] = useState(42);
5   const [fruit, setFruit] = useState('banana');
6   const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
7   ...
8 }
```

useEffect

- Recebe uma função que será executada após a renderização do componente

- Parecido com *componentDidMount* e *componentDidUpdate*

- O segundo parâmetro é um array de **dependências**. Quando alguma das dependências mudar, a função é executada novamente

Exemplo

```
function InfosUsuario(props) {
  const [user, setUser] = useState({})

  useEffect(() => {
    // useEffect recebe função que será
    // executada após a renderização do componente
    fetchUser().then(user => setUser(user))
    // Como o segundo parâmetro é um array vazio,
    // a função só será executada uma vez
    // Análogo ao componentDidMount
  }, [])

  return <div>
    <p>Nome: {user.name}</p>
    <p>Email: {user.email}</p>
  </div>
}
```

Exemplo

```
1 function InfosUsuario(props) {
2   const [user, setUser] = useState({})
3
4   useEffect(() => {
5     // useEffect recebe função que será
6     // executada após a renderização do componente
7     fetchUser(props.id).then(user => setUser(user))
8     // Agora, props.id é uma dependência.
9     // Quando esse valor mudar, a função será executada novamente
10  }, [props.id])
11
12  return <div>
13    <p>Nome: {user.name}</p>
14    <p>Email: {user.email}</p>
15  </div>
16 }
```

Regras do Hooks

1. Só usar hooks no **corpo principal** da função
   - Não pode estar dentro de ifs, whiles ou funções

2. Só usar hooks em funções do React
   - Componentes funcionais
   - Hooks customizados

3. (Extra) Todas as variáveis do escopo da própria função que estiverem sendo usadas dentro de um useEffect devem ser declaradas como dependência

Pausa - revisão

- **useState** - Guarda uma variável de estado. Recebe o estado inicial e retorna uma variável com o conteúdo do estado e uma função para definir o estado.

- **useEffect** - Recebe uma função que será executada depois da renderização e um array de dependências. Quando uma dependência muda, a função é executada novamente. Parece com *didMount* e *didUpdate*.

Hooks customizados

FUTURE4

Pra que servem?

- Agrupar lógicas de estado e lifecycle em uma função
- Ela pode ser reutilizada em vários componentes
- Podem agrupar quaisquer tipos de hooks, qualquer número de vezes

Como criar

- Basta criar uma função que chame os hooks necessários
- Pode receber quaisquer parâmetros e retornar quaisquer valores
- Seu nome deve começar com **use...**

Exemplo

```
1 function useUser(userId) {
2   const [user, setUser] = useState({})
3
4   useEffect(() => {
5     fetchUser(userId).then(user => setUser(user))
6   }, [userId])
7
8   return user
9 }
```

Exemplo

```
function InfosUsuario(props) {
  const user = useUser(props.id)

  return <div>
    <p>Nome: {user.name}</p>
    <p>Email: {user.email}</p>
  </div>
}
```

```
function Header(props) {
  const user = useUser(props.id)

  return <div>
    <p>Seja bem vindo, {user.name}</p>
    {user.isAdmin && <button>Área do administrador</button>}
  </div>
}
```

Antes

```
1 class User extends React.Component {
2   constructor(props) {
3     super(props)
4     this.state = {
5       user: {}
6     }
7   }
8
9   componentDidMount() {
10    fetchUser(this.props.userId).then(user => this.setState({user}))
11  }
12
13  componentDidUpdate(prevProps) {
14    if(prevProps.userId !== this.props.userId) {
15      fetchUser(this.props.userId).then(user => this.setState({user}))
16    }
17  }
18
19  render() {
20    const {user} = this.state
21    return this.props.render(user)
22  }
23 }
```

Depois

```
1 function useUser(userId) {
2   const [user, setUser] = useState({})
3
4   useEffect(() => {
5     fetchUser(userId).then(user => setUser(user))
6   }, [userId])
7
8   return user
9 }
```

Coding Together

FUTURE4

Coding Together

- Hook customizado de forms: **useForm**

Dúvidas?

FUTURE4

Obrigado!

FUTURE4