Introdução a React Hooks

v16.8.0

Eles permitem que você use o state e outros recursos do React sem escrever uma classe.

Motivação

- **É difícil reutilizar lógica com estado entre componentes (eram usados HOC ou render props).** Hooks permitem reutilizar a lógica sem mudar a hierarquia de componentes.
- **Componentes complexos e difíceis de entender.** Hooks permitem a divisão de um componente em funções menores que estão relacionadas.
- **Classes confundem as pessoas e as máquinas (this, bind no handlers).** Hooks permitem usar as funcionalidades do React sem usar as classes.

useState

- Preserva o estado entre renderizações.
- Retorna um par: estado atual, função para atualizá-lo.
- É parecido com o this.setState em uma classe, exceto que não mescla o estado antigo com o novo.

useEffect

- Permite executar efeitos colaterais em componentes funcionais. “Efeitos colaterais” podem ser: fazer fetching de dados, configurar uma subscription etc, e são executados depois do render.
- Por padrão o effect é executado depois da primeira renderização e depois de toda atualização.
- Esse hook é a combinação do componentDidMount, componentDidUpdate e componentWillUnmount.

useEffect sem limpeza / useEffect com limpeza

- Quando configuramos alguma subscription. Evitamos o vazamento de memória fazendo um “cleanup”.

- A limpeza de efeitos também é realizada antes de rodar o próximo efeito. Diferente do componentWillUnmount que executa apenas uma vez durante a desmontagem.

useEffect

- A limpeza ou execução de um efeito em cada renderização pode criar um problema de performance.
- Em classes usamos uma comparação com o prevProps ou prevState para ter certeza que devemos executar um efeito.
- Com hooks passamos um array de dependência. Mas tenha certeza que o array inclua qualquer valor (props ou state) que mude com o tempo.

useContext

- Aceita um objeto de contexto (React.createContext) e retorna valor atual do contexto. O valor contexto atual é determinado pela prop value do <MyContext.Provider> mais próximo acima do componente na árvore.
- É equivalente ao contextType = MyContext em uma classe ou o uso de um <MyContext.Consumer>

Regras dos hooks

- **1° Use hooks apenas no nível superior.** Isso garante que os hooks sejam chamados na mesma ordem a cada renderização, o que permite o React preservar corretamente o estado dos hooks quando são chamados várias vezes na mesma função.
- **2° Use hooks apenas em funções do React.** Não use em funções comuns do javascript.
- **3° Para não se preocupar com isso use o plugin *eslint-plugin-react-hooks*.**

Crie seus próprios hooks!

HOOKS CUSTOMIZADOS

- Permite que extraia a lógica de um componente em funções reutilizáveis.

- Um hook customizado é uma função javascript cujo nome começa com “use” e pode utilizar outros hooks.

FIM