

DBStart

React

Instrutor: Júlio Pereira Machado (julio.machado@pucrs.br)



React Hooks



React - Hooks

- React fornece uma coleção variada de hooks que são funções especiais disponíveis na fase de renderização de um componente
 - Fácil de identificar na API: funções que começam com a palavra *use*
 - Veja mais em <https://react.dev/reference/react>
- É usual surgir a necessidade de criar hooks customizados para encapsular comportamentos específicos de uma aplicação
- CUIDADO!
 - Hooks seguem regras explícitas de funcionamento e possuem restrições
 - Só podem ser chamados no nível mais alto dos componentes ou outros Hooks, ou seja, não podem ser chamados dentro de laços de repetição, comandos condicionais ou funções aninhadas

React - useState Hook

- Hook **useState** permite a um componente funcional possuir estado local e executar alterações internamente
- **useState** recebe um estado inicial e retorna um array contendo: estado atual e função de alteração de estado
- Para alterar o estado?
 - Chamar a função de alteração de estado passando um novo estado que substitui o estado anterior
 - React irá enfileirar uma solicitação de renderização no caso de mudança de estado

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

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

  const handleClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={handleClick}>Click me</button>
    </div>
  );
}

export default Counter;
```

React - useReducer Hook

- Componentes com muitas atualizações de estado espalhadas por diversos tratadores de eventos podem complicar o funcionamento dos componentes
- Possível solução:
 - Consolidar a lógica de atualização do estado de forma externa ao componente
 - Implementar alterações de estado em uma função chamada *reducer*

React - useReducer Hook

- A criação de *reducers* se dá através do hook **useReducer**
- useReducer recebe dois argumentos:
 - A função *reducer*
 - Um estado inicial
- useReducer retorna:
 - Um objeto de estado
 - Uma função *dispatch*, responsável por “despachar” ações para a função *reducer* decidir qual tipo de alteração de estado será realizada
- Ações são objetos que possuem os dados necessários para uma tomada de decisão sobre qual alteração de estado será realizada
 - Possuem usualmente um formato tal qual:
{ type: 'nome da ação', dado1: valor1, ... }

React - useReducer Hook

Taylor

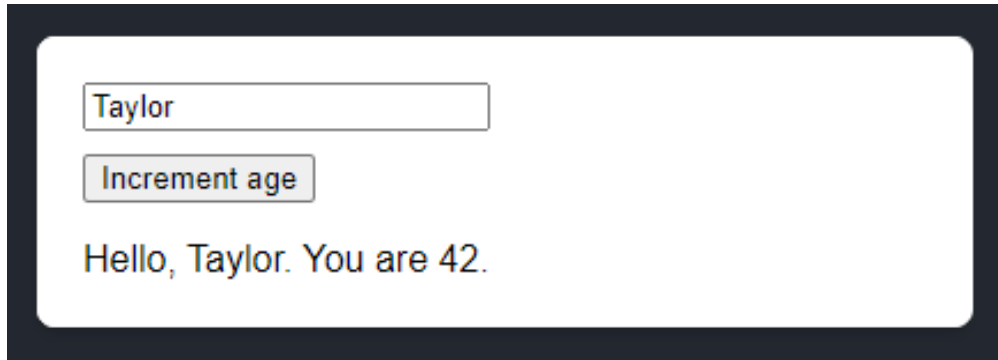
Increment age

Hello, Taylor. You are 42.

Tratador de evento
onChange

Tratador de evento
onClick

React - useReducer Hook



Taylor

Increment age

Hello, Taylor. You are 42.

```
<>
  <input
    value={state.name}
    onChange={handleInputChange}
  />
  <button onClick={handleButtonClick}>
    Increment age
  </button>
  <p>Hello, {state.name}. You are {state.age}</p>
</>
```

```
import { useReducer } from 'react';
const initialState = { name: 'Taylor', age: 42 };
function reducer(state, action) {...}
export default function Form() {
  const [state, dispatch] = useReducer(reducer, initialState);
  function handleButtonClick() {
    dispatch({ type: 'incremented_age' });
  }
  function handleInputChange(e) {
    dispatch({
      type: 'changed_name',
      nextName: e.target.value
    });
  }
  ...
}
```


React - useReducer Hook

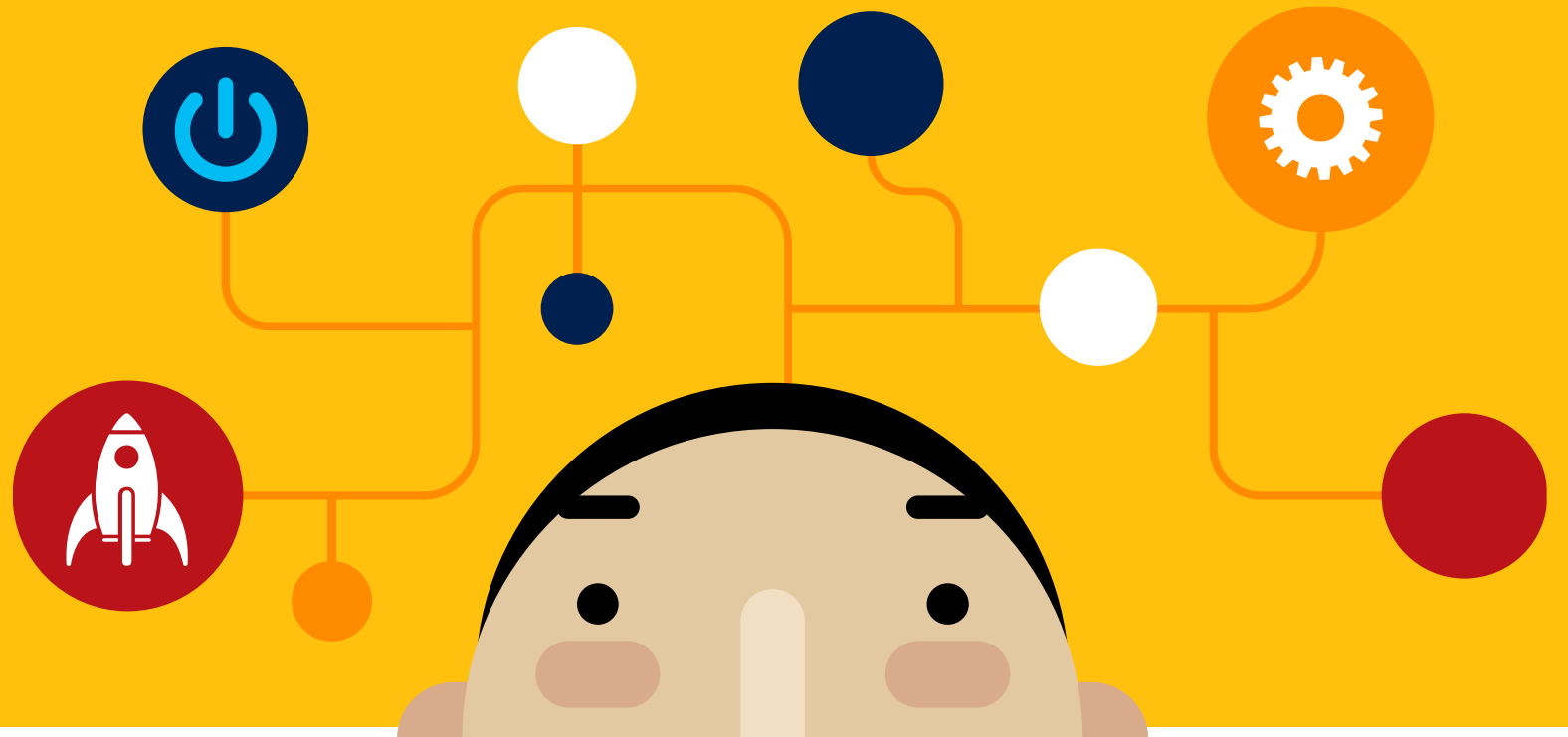
```
import { useReducer } from 'react';
const initialState = { name: 'Taylor', age: 42 };
function reducer(state, action) {
  switch (action.type) {
    case 'incremented_age': {
      return { name: state.name, age: state.age + 1 };
    }
    case 'changed_name': {
      return { name: action.nextName, age: state.age };
    }
    default: {
      throw Error('Unknown action: ' + action.type);
    }
  }
}
```

```
dispatch({ type: 'incremented_age' })
```

```
dispatch({
  type: 'changed_name',
  nextName: e.target.value
})
```

Laboratório

- Siga as instruções do arquivo Lab02_React_Componentes



React - useEffect Hook

**USAR SOMENTE QUANDO
EXTREMAMENTE
NECESSÁRIO!**

React - useEffect Hook

- Lógica de renderização do componente não pode possuir efeitos colaterais
 - Componente deve ser produzido por uma função pura
- Componentes podem executar código com efeitos colaterais após a etapa de renderização através do hook **useEffect**
- Por padrão, useEffect é executado após cada renderização

```
function EffectExample() {  
  const [count, setCount] = useState(0);  
  
  // The useEffect hook takes a callback function,  
  // which may contain side effects  
  useEffect(() => {  
    // Updating the document title is a side effect  
    // (updating the mutable contents of the page)  
    // Can reference a captured variable inside here  
    document.title = `You clicked ${count} times`;  
  });  
  
  return (  
    <div>  
      <p>You clicked {count} times</p>  
      <button onClick={() => setCount(count + 1)}>Click  
me</button>  
    </div>  
  );  
}
```

React - useEffect Hook

- Callback do hook pode retornar uma função de “limpeza”
 - Executada antes do componente ser desmontado da árvore e se o efeito for reexecutado
- Hook aceita um argumento adicional que é o “array de dependências” que limita quando o efeito é executado
 - `useEffect(func)`: todos os estados (executa o callback após cada renderização)
 - `useEffect(func, [])`: todos os estados (executa o callback somente na montagem do componente)
 - `useEffect(func, [val1, val2])`: depende dos valores (executa o callback na montagem do componente e quando qualquer valor indicado for alterado)

```
function FriendStatus(props) {  
  const [isOnline, setIsOnline] = useState(null);  
  
  const friendId = props.friend.id;  
  useEffect(() => {  
    function handleStatusChange(status) {  
      setIsOnline(status.isOnline);  
    }  
  
    FriendsAPI.subscribe(friendId, handleStatusChange);  
  
    // Effect hook callbacks can return a cleanup function  
    return () => {  
      FriendsAPI.unsubscribe(friendId, handleStatusChange);  
    };  
  
    // Optional dependencies array:  
    // only runs callback when deps values change  
    // Think of this as "syncing effects to state"  
  }, [friendId]);  
  
  // omit rendering  
}
```

React - Hooks Customizados

- Não esqueça!
- Você pode criar seus próprios Hooks!

JS useOnlineStatus.js x

```
1  import { useState, useEffect } from 'react';
2
3  export function useOnlineStatus() {
4    const [isOnline, setIsOnline] = useState(true);
5    useEffect(() => {
6      function handleOnline() {
7        setIsOnline(true);
8      }
9      function handleOffline() {
10       setIsOnline(false);
11     }
12     window.addEventListener('online', handleOnline);
13     window.addEventListener('offline', handleOffline);
14     return () => {
15       window.removeEventListener('online', handleOnline);
16       window.removeEventListener('offline', handleOffline);
17     };
18   }, []);
19   return isOnline;
20 }
```

React - Hooks Customizados

```
JS App.js x
1 import { useOnlineStatus } from "../useOnlineStatus.js";
2 function StatusBar() {
3   const isOnline = useOnlineStatus();
4   return <h1>{isOnline ? "✅ Online" : "❌ Disconnected"}</h1>;
5 }
6 function SaveButton() {
7   const isOnline = useOnlineStatus();
8   function handleSaveClick() {
9     console.log("✅ Progress saved");
10  }
11  return (
12    <button disabled={!isOnline} onClick={handleSaveClick}>
13      {isOnline ? "Save progress" : "Reconnecting..."}
14    </button>
15  );
16 }
17 export default function App() {
18   return (
19     <>
20       <SaveButton />
21       <StatusBar />
22     </>
23   );
24 }
```