day09-Hook

1. Hook

### 简介

Hooks是 React v16.8 的新特性，可以在不使用类组件的情况下，使用 state 以及其他的React特性.

Hooks是完全可选的，无需重写任何已有代码就可以在一些组件中尝试Hook

从React v16.8发布开始推出Hooks特性，100%向后兼容，Hooks不包含任何破坏性改动.

React也没有计划移除class类组件，而且Hooks不会影响对React的理解，它为已知的React概念提供了更直接的API.

### Hook解决的问题

**函数式组件不能使用state**：函数式组件比类组件更简洁好用，而Hooks让它更加丰富强大；

**副作用问题**：诸如数据获取、订阅、定时执行任务、手动修改ReactDOM这些行为都可以称为副作用；而Hooks的出现可以使用 useEffect 来处理这些副作用；

**有状态的逻辑重用组件**

**复杂的状态管理**：之前通常使用 redux、dva、mobx 这些第三方状态管理器来管理复杂的状态，而Hooks可以使用 useReducer、useContext 配合实现复杂的状态管理；

**开发效率和质量问题**：函数式组件比类组件简洁，效率高，性能也好。

### useState

useState用来管理组件的状态, 当下主要用来给函数式组件添加状态.

通过多次调用useState()，一个函数组件可以拥有多个状态。

function Bulbs() {

const [on, setOn] = useState(false);

const [count, setCount] = useState(1);

const lightSwitch = () => setOn(on => !on);

const addBulbs = () => setCount(count => count + 1);

return (

<>

<div className="bulbs">{bulbs}</div>

<button onClick={lightSwitch}>开/关</button>

<button onClick={addBulbs}>添加数量</button>

</>

);

}

### useEffect

useEffect用来给函数式组件添加生命周期方法.

函数组件中没有生命周期，那么可以使用 useEffect 来替代。如果你熟悉 React class 的生命周期函数，你可以把 useEffect Hook 看做 componentDidMount(空依赖)，componentDidUpdate(有依赖) 和 componentWillUnmount(有返回函数) 这三个函数的组合.

**默认情况下，它在第一次渲染之后和每次更新之后都会执行**。你可能会更容易接受 effect 发生在“渲染之后”这种概念，不用再去考虑“挂载”还是“更新”。React 保证了每次运行 effect 的同时，DOM 都已经更新完毕。

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

function Example() {

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

useEffect(() => {

document.title = `You clicked ${count} times`;

});

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

数据获取，设置订阅以及手动更改 React 组件中的 DOM 都属于副作用。**有些副作用可能需要清除，所以需要返回一个函数**，比如挂载时设置定时器，卸载时取消定时器

function Example() {

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

useEffect(() => {

const id = setInterval(() => {

setCount(c => c + 1);

}, 1000);

return () => clearInterval(id);

}, []);

return <h1>{count}</h1>

}

**通过跳过 Effect 进行性能优化**，在某些情况下，每次渲染后都执行清理或者执行 effect 可能会导致性能问题.

这是很常见的需求，所以它被内置到了 useEffect 的 Hook API 中。如果某些特定值在两次重渲染之间没有发生变化，你可以通知 React 跳过对 effect 的调用，只要传递数组作为 useEffect 的第二个可选参数即可.

useEffect(() => {

document.title = `You clicked ${count} times`;

}, [count]); // 仅在 count 更改时更新.

### useRef

创建ref，方便访问操作DOM节点

import React, { useState, useEffect, useMemo, useRef } from 'react';

export default function App(props){

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

const couterRef = useRef();

useEffect(() => {

console.log( couterRef.current );

}, [count]);

return (

<>

<button ref={couterRef} onClick={() => {setCount(count + 1)}}>Count: {count}</button>

</>

);

}

代码中用useRef创建了couterRef对象，并将其赋给了button的ref属性。这样，通过访问couterRef.current就可以访问到button对应的DOM对象.

如果获取子组件的引用，需先对子组件使用forwardRef(Child)高阶函数处理

### Hooks的使用规则

**只在顶层调用Hooks :** Hooks的调用尽量只在顶层作用域

不要在循环、条件或嵌套函数中调用Hook，否则可能会无法确保每次组件渲染时都以相同的顺序调用Hook

**只在函数组件调用Hooks**：目前只支持函数式组件，未来版本Hooks会扩展到class类组件；

**React Hooks的应用场景**：函数式组件、自定义Hooks;

1. 综合案例

<cdf-会员购> 详情页,左下角收藏功能

<cdf-会员购> 我的收藏页,取消收藏功能