day10-hooks高级应用

1. hooks

### hooks的其他应用

#### useContext

用来解决同一个父组件的后代组件之间的数据共享问题, 同一个父组件的所有后代组件都可以用useContext() 从最近的 context 中获取.

const context = useContext( context );

接受一个 context（上下文）对象（从React.createContext返回的值）并返回距离当前组件最近 context 的值。当父组件context数据更新时，此 Hook 将使用最新的 context 值触发重新渲染。

父组件

const App = () => {

return (

<div className="App">

<context.Provider value={{ state }}>

<Counter />

<CounterTest />

</context.Provider>

</div>

);

};

内部组件

import React, { useContext } from "react";

import { myContext } from "./reducer";

function Counter() {

const { state } = useContext(myContext);

return (

<div>

Counter Count: {state.count}

</div>

);

}

#### useReducer

useState的一个增强体，用于处理复杂的状态管理，灵感来源于Redux的reducer

useState 内部就是基于 useReducer 实现的，只是对于简单的状态管理，useState()比较好用.

function reducer(state, action) {

switch (action.type) {

case "increment":

return { count: state.count + 1 };

case "decrement":

return { count: state.count - 1 };

default:

return state;

}

}

const [state, dispatch] = useReducer(reducer, { count: 0 });

任何组件拿到 useReducer 返回的state,dispatch都可以方便的操作,管理数据.

#### useMemo

useMemo(callback, array)：性能优化，利用了闭包的特性，会返回一个记忆值, 通过记忆值来避免在每个渲染上都执行高开销的计算（计算缓存）；适用于复杂的计算场景，如复杂的列表渲染，对象深拷贝...

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

function MemoTest(props) {

let [count, setCount] = useState(0)

let [max, setMax] = useState(100)

// useMemo将依赖state数据计算的函数缓存起来,并将结果计算后返回,

// 如果依赖的state数据没有变化,函数不会执行,达到性能优化的目的

var getNum=useMemo(function(){

console.log("getNum函数执行了")

let num=0;

for (let i = 0; i < max; i++) {

num+=i

}

return num

},[max])

return (

<div>

<hr/>

<div>num的值:{getNum}</div>

<div>count的值:{count}</div>

<button onClick={()=>{

setMax(++max)

}}>修改max</button>

<button onClick={()=>{

setCount(++count)

}}>修改count</button>

</div>

);

}

export default MemoTest;

useMemo() 接收2个参数：一个计算函数 和 依赖数组。

计算函数: 返回一个函数

依赖数组: 放置依赖的数据

**使用useMemo() 提升计算阶乘的性能.**

useMemo(() => factorialOf(number), [number])

memo的目标是速度，他通过内存来提升速度, 记住这个是以空间换速度，所以最好确定你是否值得那么做，有些场景很有必要使用.

在处理递归函数时，Memon最有效，递归函数用于执行诸如GUI渲染，Sprite和动画物理等繁重操作.

#### useCallback

useCallback(callback, array)：也是用于性能优化，与useMemo()不同的是，返回值是callback本身；

import React,{useState,useCallback} from 'react'

// set集合具有去重属性

var set=new Set();

export default function CallBackTest() {

let [num, setNum] = useState(0)

let [count, setCount] = useState(0)

// useCallback优化返回后的函数

// 当被处理的函数所依赖的状态数据值不变,则不会重复创建

var getCount=useCallback(() => {return count},[count])

return (

<div>

<hr/>

<div>num的值:{num}</div>

<div>count的值:{getCount()}</div>

<button onClick={()=>{

setNum(++num)

}}>修改num</button>

<button onClick={()=>{

setCount(++count)

}}>修改count</button>

<ChildComponent getCount={getCount}></ChildComponent>

</div>

)

}

function ChildComponent({getCount}){

set.add(getCount)

return (

<div>

<hr/>

<div>我是callback子组件中的count:{getCount()}</div>

<div>set集合中有{set.size}个元素</div>

</div>

)

}

**主要优化场景:**

减少render次数

减少计算量

### 路由库中的Hooks

import {useHistory, useLocation, useParams, useRouteMatch } from "react-router-dom"

#### useHistory

这个钩子可以访问history，可以更好的进行导航

let history = useHistory();

#### useLocation

此钩子可以返回location表示当前URL对象

let location = useLocation();

#### useParams

useParams返回URL参数的键值对的对象

let {id} = useParams();

#### useRouteMatch

此钩子可以获取当前组件的路由地址,用于动态匹配路由

// useRouteMatch获取当前组件的url

var match = useRouteMatch();

return (

<div>

<ul style={{ listStyle: "none", display: "flex", justifyContent: "space-evenly" }}>

<li><NavLink to={`${match.url}/hotFilm`}>正在热映</NavLink></li>

<li><NavLink to={`${match.url}/newFilm`}>最新上映</NavLink></li>

</ul>

<Switch>

<Route path={`${match.url}/hotFilm`} component={HotFilm}></Route>

<Route path={`${match.url}/newFilm`} component={NewFilm}></Route>

</Switch>

</div>

);

### 自定义Hooks

通过自定义 Hook，可以将组件逻辑提取到可重用的函数中, 当然自定义Hook中也可以使用其他已有的Hook.

Hooks本质上就是封装好的勾子函数，下面封装一个；

const useRequest = () => {

const [list, setList] = useState([]);

//初始化时请求接口获取数据,空依赖的useEffect只调用一次

useEffect(() => {

req('/list').then(res => {

setList(res.data)

}).catch(e => {

console.error(e)

})

}, []);

return {

list,

setList,

}

};

使用自定义hooks

const useCustomHookDemo = () => {

const { list } = useList();

return (

<div>

<h3>自定义list hooks </h3>

{

list ?

(

<ol>

{

list.map(item => (

<li key={item.id}>姓名：{item.name}, 年龄：{item.age}</li>

))

}

</ol>

) :

('loading...')

}

</div>

)

};

1. 综合案例

<cdf-会员购> 收货地址管理.