React提供了在函数式组件中也可以使用状态，生命周期等的方法，称为Hook

**State Hook**

如下我们在函数式组件中声明一个状态count，而setCount为更新这个状态的方法

import React, { useState } from 'react';

function Example() {

  // 声明一个叫 “count” 的 state 变量。

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

  return (

    <div>

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

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

        Click me

      </button>

    </div>

  );

}

我们可以调用多次useState生命多个状态

function ExampleWithManyStates() {

    // 声明多个 state 变量！

    const [age, setAge] = useState(42);

    const [fruit, setFruit] = useState('banana');

    const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);

    // ...

}

**Effect Hook**

我们可以在函数是组件中声明生命周期方法

function FriendStatus(props) {

    const [isOnline, setIsOnline] = useState(null);

    // 组件componentDidMount、componentDidUpdate时调用此方法

    useEffect(() => {

        isOnline = true;

        // 组件componentWillUnmount时调用此方法，如果没有return，则不调用

        return () => {

            // ...

        };

    });

    if (isOnline === null) {

      return 'Loading...';

    }

    return isOnline ? 'Online' : 'Offline';

}

**Context Hook**

获取上下文

const themes = {

    light: {

        foreground: "#000000"

    },

    dark: {

        foreground: "#ffffff"

    }

};

const ThemeContext = React.createContext(themes.light);

function App() {

    return (

        <ThemeContext.Provider value={themes.dark}>

            <ThemedButton />

        </ThemeContext.Provider>

    );

}

function ThemedButton() {

    const theme = useContext(ThemeContext);

    return (

        <button style={{ color: theme.foreground }}>

            I am styled by theme context!

        </button>

    );

}

**Reducer Hook**

reducer是针对一个状态的多种操作，如下示例一个reducer

function todosReducer(state, action) {

    switch (action.type) {

        case 'add':

            return [...state, action.text];

        case 'remove':

            return state.fliter(e => e != action.text);

        case 'clear':

            return [];

        default:

            return state;

    }

}

react提供了一个useReducer，其实这是一个很简单的封装

// reducer: 和redux的reducer一个样

// initialState: state的初始数据

function useReducer(reducer: (state: any, action: any) => any, initialState: any) {

    const [state, setState] = useState(initialState);

    function dispatch(action) {

        const nextState = reducer(state, action);

        setState(nextState);

}

    return [state, dispatch];

}

**自定义 Hook**

如下就是一个自定义的Hook，一个很简单的函数

function useIndex(index) {

    const [i, setIndex] = useState(index + 1);

    return [i, setIndex];

}

使用自定义Hook

function FriendListItem(props) {

    const [i, setIndex] = useIndex(0);

    return (

      <li>{i}</li>

    );

}