

link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=67 sentences=118, words=1548

□

1. React Hooks

- Hook可以让你在不编写 class组件 的情况下使用 state

2. 搭建项目

```
npx create-react-app zhufeng_hooks  
cd zhufeng_hooks  
yarn start
```

3. useState

- useState 会返回一对值: 当前状态和一个让你更新它的函数
- useState 唯一的参数就是初始 state

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

3.1 使用

```
import React from 'react';  
import ReactDOM from 'react-dom';  
function Counter() {  
  const [number, setNumber] = React.useState(0);  
  return (  
    <>  
      <p>{number}</p>  
      <button onClick={() => setNumber(number+1)}>+button</button>  
    </>  
  )  
}  
function render() {  
  ReactDOM.render(<Counter />, document.getElementById('root'));  
}  
render();
```

3.2 实现

```
import React from 'react';  
import ReactDOM from 'react-dom';  
+let lastState;  
+function useState(initialState) {  
+  lastState = lastState || initialState;  
+  function setState(newState) {  
+    lastState = newState;  
+    render();  
+  }  
+  return [lastState, setState];  
+}  
  
function Counter() {  
  const [number, setNumber] = useState(0);  
  return (  
    <>  
      {number}  
      setNumber(number+1)}>+  
    </>  
  )  
}  
function render() {  
  ReactDOM.render(, document.getElementById('root'));  
}  
render();
```

4. 多useState

4.1 使用

```
import React from 'react';  
import ReactDOM from 'react-dom';  
function Counter() {  
  const [number1, setNumber1] = React.useState(0);  
  const [number2, setNumber2] = React.useState(0);  
  return (  
    <>  
      <p>{number1}</p>  
      <button onClick={() => setNumber1(number1+1)}>+button</button>  
      <hr />  
      <p>{number2}</p>  
      <button onClick={() => setNumber2(number2+1)}>+button</button>  
    </>  
  )  
}  
function render() {  
  ReactDOM.render(<Counter />, document.getElementById('root'));  
}  
render();
```

4.2 实现

```

import React from 'react';
import ReactDOM from 'react-dom';
let hookStates = [];
let hookIndex = 0;
function useState(initialState) {
    hookStates[hookIndex]=hookStates[hookIndex] || initialState;

    let currentIndex = hookIndex;
    function setState(newState) {
        hookStates[currentIndex]=newState;
        render();
    }
    return [hookStates[hookIndex++],setState];
}
function Counter() {
    const [number1,setNumber1] = useState(0);
    const [number2,setNumber2] = useState(0);
    return (
        <>
            <p>{number1}</p>
            <button onClick={()=>setNumber1(number1+1)}>+button</button>
            <hr/>
            <p>{number2}</p>
            <button onClick={()=>setNumber2(number2+1)}>+button</button>
        </>
    )
}
function render() {
    hookIndex = 0;
    ReactDOM.render(<Counter/>,document.getElementById('root'));
}
render();

```

5. 优化

- 我们可以使用 `useMemo` 和 `useCallback` 来减少渲染

5.1 使用

```

import React from 'react';
import ReactDOM from 'react-dom';
let Child = ({ onClick, data }) => {
    console.log("Child render");
    return <button onClick={onClick}>{data.number}</button>;
}
Child = React.memo(Child);

function App() {
    const [number, setNumber] = React.useState(0);
    const [name, setName] = React.useState("zhufeng");
    const addClick = React.useCallback(() => setNumber(number + 1), [number]);
    const data = React.useMemo(() => ({ number }), [number]);
    return (
        <div>
            <input value={name} onChange={e => setName(e.target.value)} />
            <Child onClick={addClick} data={data} />
        </div>
    );
}
function render() {
    ReactDOM.render(<App />,document.getElementById('root'));
}
render();

```

5.2 实现

```

import React from 'react';
import ReactDOM from 'react-dom';

+let hookStates = []; //放着此组件的所有的hooks数据
+let hookIndex = 0; //代表当前的hooks的索引
+function useState(initialState){
+  //如果有老值取老值,没有取默认值
+  hookStates[hookIndex]=hookStates[hookIndex]||initialState;
+  //暂存索引
+  let currentIndex = hookIndex;
+  function setState(newState){
+    hookStates[currentIndex]=newState;
+    render();
+  }
+  return [hookStates[hookIndex++],setState];
+}

+function useCallback(callback,dependencies){
+  if(hookStates[hookIndex]){
+    let [lastCallback,lastCallbackDeps] = hookStates[hookIndex];
+    let same = dependencies.every((item,index)=>item === lastCallbackDeps[index]);
+    if (same) //如果老依赖和新的依赖都相同,则直接返回老的,如果不一相同,则返回新的
+      hookIndex++;
+    return lastCallback;
+  }else{
+    hookStates[hookIndex++]=[callback,dependencies];
+    return callback;
+  }
+}

+function useMemo(factory,dependencies){
+  if(hookStates[hookIndex]){
+    let [memo,lastDeps] = hookStates[hookIndex];
+    let same = dependencies.every((item,index)=>item === lastDeps[index]);
+    if (same) //如果老依赖和新的依赖都相同,则直接返回老的,如果不一相同,则返回新的
+      hookIndex++;
+    return memo;
+  }else{
+    let newMemo = factory();
+    hookStates[hookIndex++]=[newMemo,dependencies];
+    return newMemo;
+  }
+}

let Child = ({ onClick, data }) => {
  console.log("Child render");
  return {data.number};
}

Child = React.memo(Child);

function App() {
  const [number, setNumber] = useState(0);
  const [name, setName] = useState("zhufeng");
  const addClick = useCallback(() => setNumber(number + 1), [number]);
  const data = useMemo(() => ({ number }, [number]);
  return (
    <div>
      {name} {number}
      <button onClick={addClick}>add</button>
      <div>
        {data.number}
      </div>
    </div>
  );
}

function render(){
  hookIndex = 0;
  ReactDOM.render(,document.getElementById('root'));
}

render();

```

6.1 使用

```
import React from 'react';
import ReactDOM from 'react-dom';

function Counter() {
  const [name, setName] = React.useState('珠峰');
  const [number, setNumber] = React.useState(0);
  React.useEffect(() => {
    console.log(number);
  }, [number]);
  return (
    <>
      <p>{name}:{number}</p>
      <button onClick={()=>setName('架构')}>修改名称</button>
      <button onClick={()=>setNumber(number+1)}>+</button>
    </>
  )
}

function render() {
  ReactDOM.render(<Counter/>, document.getElementById('root'));
}

render();
```

6.2 实现useEffect

```
import React from 'react';
import ReactDOM from 'react-dom';

+let hookStates = [];
+let hookIndex = 0;
+function useState(initialState) {
+  hookStates[hookIndex]=hookStates[hookIndex]||initialState;
+  let currentIndex = hookIndex;
+  function setState(newState) {
+    hookStates[currentIndex]=newState;
+    render();
+  }
+  return [hookStates[hookIndex++],setState];
+}

+function useEffect(callback,dependencies) {
+  if(hookStates[hookIndex]){
+    let lastDeps = hookStates[hookIndex];
+    let same = dependencies.every((item,index)=>item === lastDeps[index]);
+    if(same){
+      hookIndex++;
+    }else{
+      hookStates[hookIndex++]=dependencies;
+      setTimeout(callback);
+    }
+  }else{
+    hookStates[hookIndex++]=dependencies;
+    setTimeout(callback);
+  }
+}

function Counter() {
  const [name, setName] = useState('珠峰');
  const [number, setNumber] = useState(0);
  useEffect(() => {
    console.log(number);
  }, [number]);
  return (
    <>
      {name}:{number}
      setName('架构')>修改名称
      setNumber(number+1)>+
    </>
  )
}

function render() {
  hookIndex=0;
  ReactDOM.render(, document.getElementById('root'));
}

render();
```

7. useLayoutEffect

- 其函数签名与 useEffect 相同，但它会在所有的 DOM 变更之后同步调用 effect
- useEffect 不会阻塞浏览器渲染，而 useLayoutEffect 会阻塞浏览器渲染
- useEffect 会在浏览器渲染结束后执行，useLayoutEffect 则是在 DOM 更新完成后，浏览器绘制之前执行

7.1 使用

```

import React from 'react';
import ReactDOM from 'react-dom';

const Animate = () => {
  const red = React.useRef();
  const green = React.useRef();
  React.useLayoutEffect(() => {
    red.current.style.transform = `translate(500px)`;
    red.current.style.transition = `all 500ms`;
  });
  React.useEffect(() => {
    green.current.style.transform = `translate(500px)`;
    green.current.style.transition = `all 500ms`;
  });
  let style = { width: '100px', height: '100px' }
  return (
    <div>
      <div style={{ ...style, backgroundColor: 'red' }} ref={red}>div</div>
      <div style={{ ...style, backgroundColor: 'green' }} ref={green}>div</div>
    </div>
  )
}

function render() {
  ReactDOM.render(<Animate />, document.getElementById('root'));
}

render();

```

7.2 实现

```

import React from 'react';
import ReactDOM from 'react-dom';

+let hookStates = [];
+let hookIndex = 0;
+function useEffect(callback,dependencies){
+  if(hookStates[hookIndex]){
+    let lastDeps = hookStates[hookIndex];
+    let same = dependencies.every((item,index)=>item === lastDeps[index]);
+    if(same){
+      hookIndex++;
+    }else{
+      hookStates[hookIndex++]=dependencies;
+      setTimeout(callback);
+    }
+  }else{
+    hookStates[hookIndex++]=dependencies;
+    setTimeout(callback);
+  }
+}
+function useLayoutEffect(callback,dependencies){
+  if(hookStates[hookIndex]){
+    let lastDeps = hookStates[hookIndex];
+    let same = dependencies.every((item,index)=>item === lastDeps[index]);
+    if(same){
+      hookIndex++;
+    }else{
+      hookStates[hookIndex++]=dependencies;
+      queueMicrotask(callback);
+    }
+  }else{
+    hookStates[hookIndex++]=dependencies;
+    queueMicrotask(callback);
+  }
+}
const Animate = () => {
  const red = React.useRef();
  const green = React.useRef();
  useLayoutEffect(() => {
    red.current.style.transform = `translate(500px)`;
    red.current.style.transition = `all 500ms`;
  });
  useEffect(() => {
    green.current.style.transform = `translate(500px)`;
    green.current.style.transition = `all 500ms`;
  });
  let style = { width: '100px', height: '100px' }
  return (
    <div>
      <div style={{ ...style, backgroundColor: 'red' }} ref={red}>div</div>
      <div style={{ ...style, backgroundColor: 'green' }} ref={green}>div</div>
    </div>
  )
}

function render() {
  ReactDOM.render(, document.getElementById('root'));
}

render();

```

8. useContext

- 接收一个 context 对象并返回该 context 的当前值

8.1 使用

```
import React from 'react';
import ReactDOM from 'react-dom';
const CounterContext = React.createContext();
function Counter() {
  let [state, setState] = React.useContext(CounterContext);
  return (
    <>
      <p>{state.number}</p>
      <button onClick={() => setState({number:state.number+1})}>+button</button>
      <button onClick={() => setState({number:state.number-1})}>-button</button>
    </>
  )
}
function App() {
  const [state, setState] = React.useState({number:0});
  return (
    <CounterContext.Provider value={{state,setState}}>
      <Counter/>
    <CounterContext.Provider>
  )
}
function render() {
  ReactDOM.render(<App/>, document.getElementById('root'));
}
render();
```

8.2 实现

```
import React from 'react';
import ReactDOM from 'react-dom';
const CounterContext = React.createContext();
+function useContext(context){
+  return context._currentValue;
+}
function Counter() {
  let [state, setState] = useContext(CounterContext);
  return (
    <>
      {state.number}
      <button onClick={() => setState({number:state.number+1})}>+
      <button onClick={() => setState({number:state.number-1})}>-
    </>
  )
}
function App() {
  const [state, setState] = React.useState({number:0});
  return (
    <Counter/>
  )
}
function render() {
  ReactDOM.render(, document.getElementById('root'));
}
render();
```

9. useReducer

- 它接收一个形如 (state, action) => newState 的 reducer，并返回当前的 state 以及与其配套的 dispatch 方法

9.1 使用

```
import React from 'react';
import ReactDOM from 'react-dom';

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return state+1;
    case 'decrement':
      return state-1;
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = React.useReducer(reducer, 0);
  return (
    <>
      Count: {state}
      <button onClick={() => dispatch({type: 'increment'})}>+button</button>
      <button onClick={() => dispatch({type: 'decrement'})}>-button</button>
    </>
  )
}

function render() {
  ReactDOM.render(<Counter/>, document.getElementById('root'));
}
render();
```

9.2 实现

```

import React from 'react';
import ReactDOM from 'react-dom';

+let hookStates = [];
+let hookIndex = 0;
+function useState(initialState) {
+  hookStates[hookIndex]=hookStates[hookIndex]||initialState;
+  let currentIndex = hookIndex;
+  function setState(newState) {
+    hookStates[currentIndex]=newState;
+    render();
+  }
+  return [hookStates[hookIndex++],setState];
+}
+function useReducer(reducer, initialState) {
+  hookStates[hookIndex]=hookStates[hookIndex]||initialState;
+  let currentIndex = hookIndex;
+  function dispatch(action) {
+    hookStates[currentIndex]=reducer(hookStates[currentIndex],action);
+    render();
+  }
+  return [hookStates[hookIndex++], dispatch];
+}
const reducer = (state=0,action)=>{
  switch(action.type){
    case 'add':
      return state+1;
    default:
      return state;
  }
}
function Counter() {
  const [number1,setNumber1] = useState(0);
  const [number2,dispatch] = useReducer(reducer,0);
  return (
    <>
      {number1}
      setNumber1(number1+1)}>+
      {number2}
      dispatch({type: 'add'})}>+
    </>
  )
}
function render() {
  hookIndex=0;
  ReactDOM.render(,document.getElementById('root'));
}
render();

```