

React

为什么是 Hooks

- ② 组件之间复用状态逻辑很难
- ② 复杂组件变的难以理解
- ② 难以理解的*class*

为了解决这些问题，**Hook** 使你在非 **class** 的情况下可以使用更多的 **React** 特性。从概念上讲，**React** 组件一直更像是函数。而 **Hook** 则拥抱了函数，同时也没有牺牲 **React** 的精神原则。**Hook** 提供了问题的解决方案，无需学习复杂的函数式或响应式编程技术。

重要钩子:

- useState: 用于定义组件的 State，对标到类组件中this.state的功能
- useEffect: 通过依赖触发的钩子函数，常用于模拟类组件中的componentDidMount, componentDidUpdate, componentWillUnmount方法
- 其它内置钩子:useContext: 获取 context 对象
- useReducer: 类似于 Redux 思想的实现，但其并不足以替代 Redux，可以理解成一个组件内部的 redux，并不是持久化存储，会随着组件被销毁而销毁；属于组件内部，各个组件是相互隔离的，单纯用它并无法共享数据；配合useContext的全局性，可以完成一个轻量级的 Redux
- useCallback: 缓存回调函数，避免传入的回调每次都是新的函数实例而导致依赖组件重新渲染，具有性能优化的效果；
- useMemo: 用于缓存传入的 props，避免依赖的组件每次都重新渲染；
- useRef: 获取组件的真实节点；
- useLayoutEffect: DOM更新同步钩子。用法与useEffect类似，只是区别于执行时间点的不同。
useEffect属于异步执行，并不会等待 DOM 真正渲染后执行，而useLayoutEffect则会真正渲染后才触发；可以获取更新后的 state；
- 自定义钩子(useXXXX): 基于 Hooks 可以引用其它 Hooks 这个特性，我们可以编写自定义钩子。

useState

React hook: not magic, just arrays

```
1 function RenderFunctionComponent() {  
2     const [firstName, setFirstName] = useState("Rudi");  
3     const [lastName, setLastName] = useState("Yardley");  
4  
5     return (  
6         <Button onClick={() => setFirstName("Fred")}>Fred</Button>  
7     );  
8 }
```

hooks-state-example.js hosted with ❤ by GitHub

[view raw](#)

1. 初次渲染的时候，按照 useState, useEffect 的顺序，把 state, deps 等按顺序塞到 memoizedState 数组中。
2. 更新的时候，按照顺序，从 memoizedState 中把上次记录的值拿出来。
3. 如果还是不清楚，可以看下面的图。

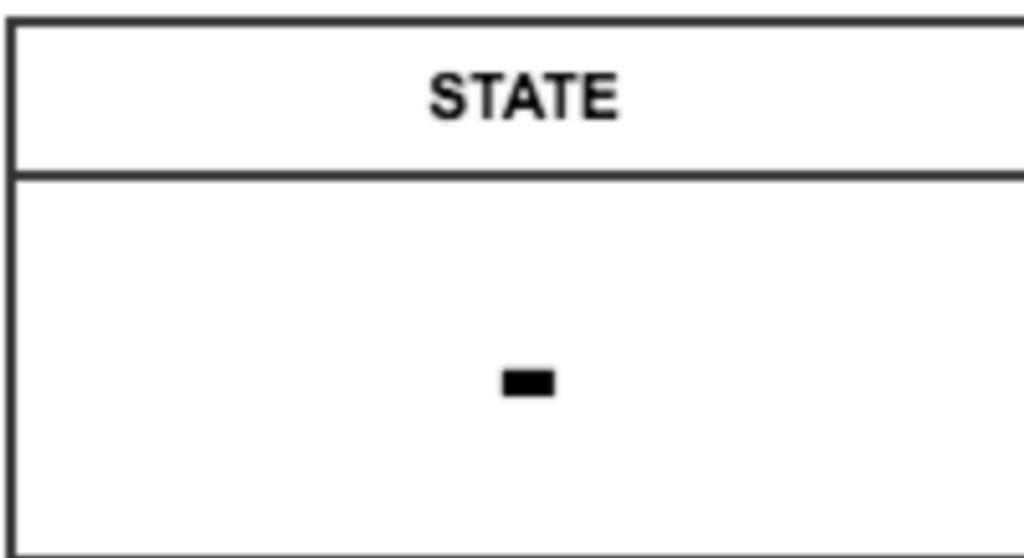
```
let memoizedState = [] // hooks 存放在这个数组
let cursor = 0 // 当前 memoizedState 下标

function useState(initialValue) {
  memoizedState[cursor] = memoizedState[cursor] || initialValue;
  const currentCursor = cursor;
  function setState(newState) {
    memoizedState[currentCursor] = newState;
    render();
  }
  return [memoizedState[cursor++], setState]; // 返回当前 state，并把 cursor 加 1
}

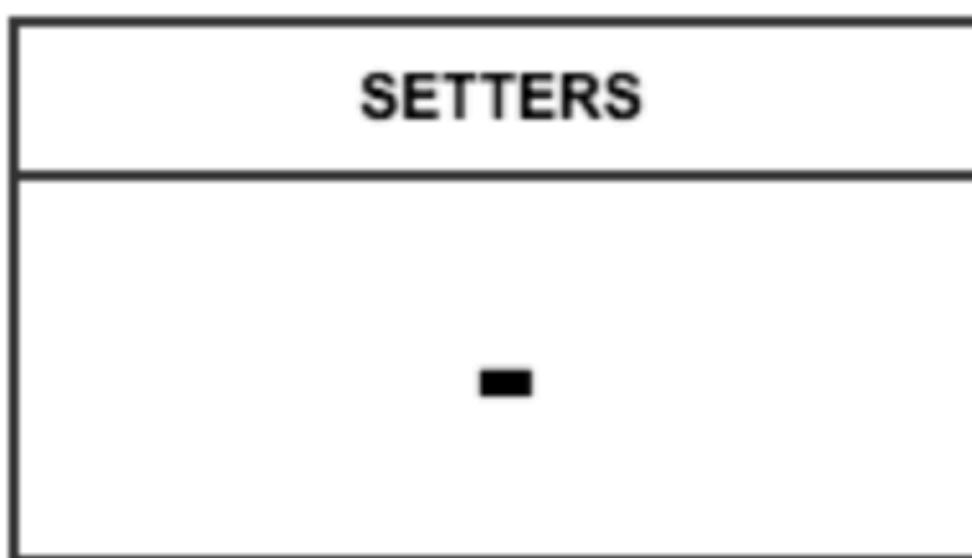
function useEffect(callback, depArray) {
  const hasNoDeps = !depArray;
  const deps = memoizedState[cursor];
  const hasChangedDeps = deps
    ? !depArray.every((el, i) => el === deps[i])
    : true;
  if (hasNoDeps || hasChangedDeps) {
    callback();
    memoizedState[cursor] = depArray;
  }
  cursor++;
}
```

创建两个空的array, state and
setters

Initialization



0



Initialisation: Two empty arrays, Cursor is 0

第一次渲染，循环所有useState，把state和
setter方法分别压入两个数组

First Render

(cursor = 0)

```
const [firstName, setFirstName] = useState("Rudi")
```

STATE

"Rudi"

SETTERS

setFirstName

0

CURSOR ++

```
const [lastName, setLastName] = useState("Yardley")
```

STATE

"Rudi"

SETTERS

setName

"Yardley"

1

setLastName

First render: Items written to the arrays as cursor increments.

更新state，触发render，cursor被重置，根据
useState的声明顺序依次拿出state值，更新UI

setFirstName("Fred")



Setters “remember” their index and set memory according to it.

```
let tag = true;

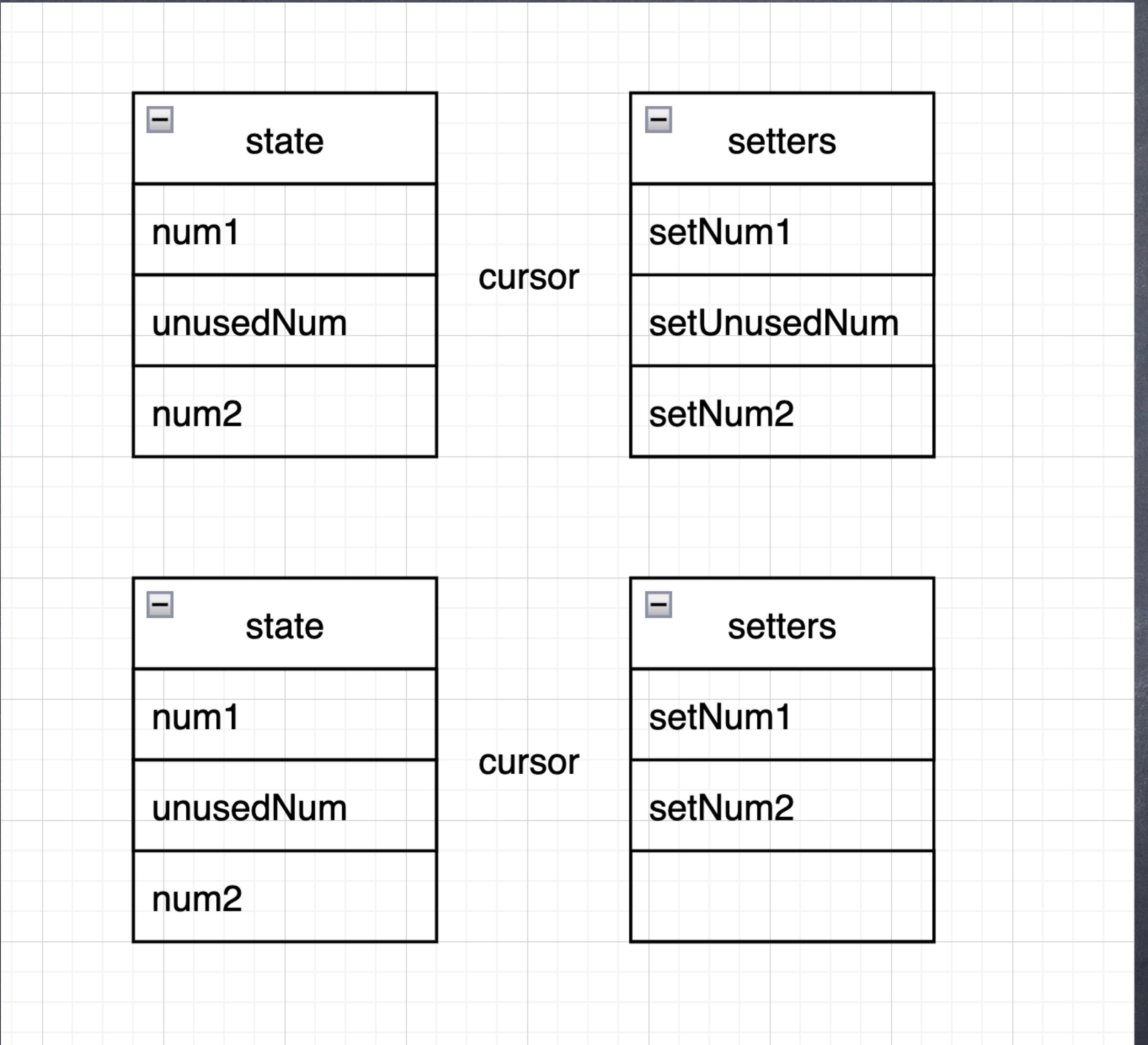
function App() {
  const [num, setNum] = useState < number > 0;

  // 只有初次渲染，才执行
  if (tag) {
    const [unusedNum, setUnusedNum] = useState < number > 1;
    tag = false;
  }

  const [num2, setNum2] = useState < number > 2;

  return (
    <div>
      <div>num: {num}</div>
      <div>
        <button onClick={() => setNum(num + 1)}>加 1</button>
        <button onClick={() => setNum(num - 1)}>减 1</button>
      </div>
      <hr />
      <div>num2: {num2}</div>
      <div>
        <button onClick={() => setNum2(num2 * 2)}>扩大一倍</button>
        <button onClick={() => setNum2(num2 / 2)}>缩小一倍</button>
      </div>
    </div>
  );
}
```

- 第一次执行， state有三个元素
- 更新state触发render
- 此时tag=false， 不会执行if流程， 也就是没有useState
- 最后导致setNum2不起作用



结论：不要在循环、判断逻辑中使用
`useState`，最好是在函数顶部使用

useEffect

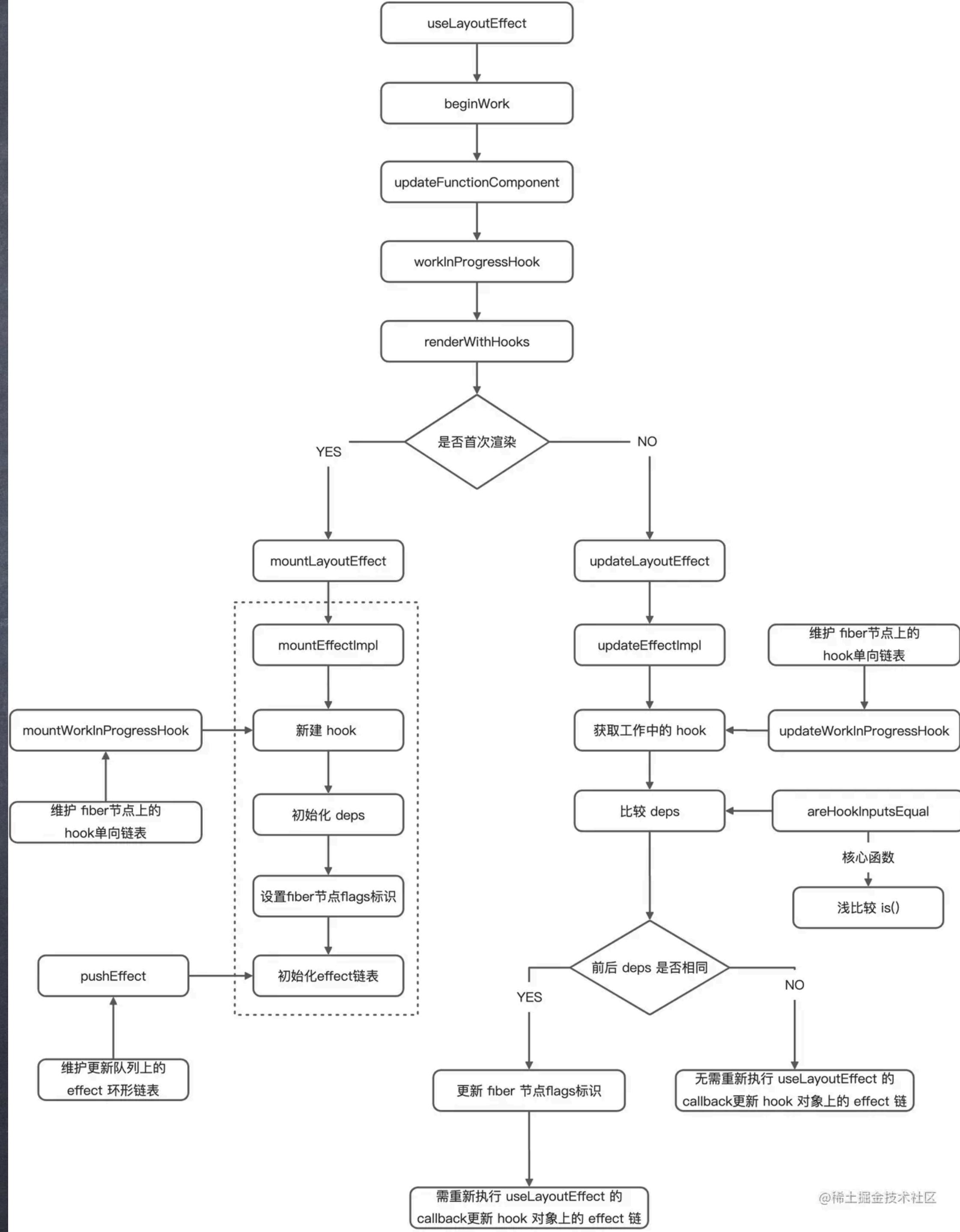
副作用函数？

- `useEffect` 函数第一个参数为 `callback` 函数，就是要执行的函数
- 第二个参数为一个数组，数组中是要监听的变量
- `[]` 空数组之所以会起到 `componentDidMount` 的作用，是因为每次更新，`[]` 空数组没有任何监听的变量，也就是不存在变化一说，所以只会被执行一次
- `useEffect` 函数返回的第一个函数作为销毁 `state` 使用

```
1 type Hooks = {
2   memoizedState: any, // 指向当前渲染节点 Fiber
3   baseState: any, // 初始化 initialState, 已经每次 dispatch 之后 newState
4   // 当前需要更新的 Update , 每次更新完之后, 会赋值上一个 update,
5   // 方便 react 在渲染错误的边缘, 数据回溯
6   baseUpdate: Update<any> | null,
7   queue: UpdateQueue<any> | null, // UpdateQueue 通过
8   next: Hook | null, // link 到下一个 hooks, 通过 next 串联每一 hooks
9 }
10
11 type Effect = {
12   tag: HookEffectTag, // effectTag 标记当前 hook 作用在 life-cycles 的哪一个阶段
13   create: () => mixed, // 初始化 callback
14   destroy: (() => mixed) | null, // 卸载 callback
15   deps: Array<mixed> | null,
16   next: Effect, // 同上
17 };
```

useLayoutEffect

`useLayoutEffect`主要在`useEffect`函数结果不满意时才被用到，一般的经验是当处理`dom`改变带的副作用才会被用到，该`Hook`执行时，浏览器并未对`dom`进行渲染，较`useEffect`执行要早



useEffect and useLayoutEffect

- ① useEffect和useLayoutEffect在Mount和Update阶段执行方法一样，传参不一样
- ② useEffect异步执行，而useLayoutEffect同步执行
- ③ Effect用HookPassive标记useEffect，用HookLayout标记useLayoutEffect

`useMemo`

传递一个函数和依赖项，当依赖项发生变化，执行函数，该函数需要返回一个值

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

const userInfo = {
    // ...
    age: count,
    name: 'Jace'
}

return <UserCard userInfo={userInfo}>

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

const userInfo = useMemo(() => {
    return {
        // ...
        name: "Jace",
        age: count
    };
}, [count]);

return <UserCard userInfo={userInfo}>
```

javascript 复制代码

```
const [age, followUser] = useMemo(() => {
  return [
    new Date().getFullYear() - userInfo.birth, // 根据生日计算年龄
    async () => { // 关注用户
      await request('/follow', { uid: userInfo.id });
      // ...
    }
  ];
}, [userInfo]);

return (
  <div>
    <span>name: {userInfo.name}</span>
    <span>age: {age}</span>
    <Card followUser={followUser}/>
    {
      useMemo(() => (
        // 如果 Card1 组件内部没有使用 React.memo 函数，那还可以通过这种方式在父组件减少子组件的渲染
        <Card1 followUser={followUser}/>
      ), [followUser])
    }
  </div>
)
```

useCallback

返回一个函数，当监听的数据发生变化才会执行回调函数

```
1 import React, { useState, useCallback } from "react";
2 import Button from "./Button";
3
4 export default function App() {
5   const [count1, setCount1] = useState(0);
6   const [count2, setCount2] = useState(0);
7   const [count3, setCount3] = useState(0);
8
9   const handleClickButton1 = () => {
10     setCount1(count1 + 1);
11   };
12
13   const handleClickButton2 = useCallback(() => {
14     setCount2(count2 + 1);
15   }, [count2]);
16
17   return (
18     <div>
19       <div>
20         <Button onClickButton={handleClickButton1}>Button1</Button>
21       </div>
22       <div>
23         <Button onClickButton={handleClickButton2}>Button2</Button>
24       </div>
25       <div>
26         <Button
27           onClickButton={() => {
28             setCount3(count3 + 1);
29           }}
30         >
31           Button3
32         </Button>
33       </div>
34     </div>
35   );
}
```

useRef

```
export function useRef<T>(initialValue: T): {current: T} {
  currentlyRenderingFiber = resolveCurrentlyRenderingFiber();
  workInProgressHook = createWorkInProgressHook();
  let ref;

  if (workInProgressHook.memoizedState === null) {
    ref = {current: initialValue};
    workInProgressHook.memoizedState = ref;
  } else {
    ref = workInProgressHook.memoizedState;
  }
  return ref;
}
```

useReducer

- 聚合参数，以达到开发效率的提升以及代码的简洁
- `useReducer`在`re-render`时候不会改变存储位置，`state`作为`props`传给子`component`不会产生`diff`的效率问题 (`useMemo`优化)

全部分類 ▾ 搜尋書名、分類

書名	價錢 ◀	分類	出版日期
遊牧人生	332	社會科學	2019/10/31
韓國人不想讓你知道的事	234	社會科學	2021/03/30

書名	價錢 ◀	分類	出版日期
遊牧人生	332	社會科學	2019/10/31
韓國人不想讓你知道的事	234	社會科學	2021/03/30

```
const App = () => {
  // 搜索用关键字
  const [keyword, setKeyword] = useState("");
  // 目录
  const [category, setCategory] = useState("0");
  // 翻页页数
  const [pagination, setPagination] = useState(DEFAULT_PAGINATION);
  // 排序
  const [sortBy, setSortBy] = useState(null);
  // 聚合
  const [variables, setVariables] = useState({
    filter: { keyword, category },
    pagination
  });
  const { data, metadata } = useQuery("/api/books", { variables });

  const handleChangeKeyword = (e) => {
    setKeyword(e.target.value);
    setPagination({ ...pagination, current: 1 });
  };

  const handleChangeCategory = (value) => {
    setCategory(value);
    setPagination({ ...pagination, current: 1 });
  };
};
```

```
const reducer = (state, action) => {
  switch (action.type) {
    case "CHANGE_KEYWORD":
      return {
        ...state,
        filter: { ...state.filter, keyword: action.payload.keyword },
        pagination: { ...state.pagination, current: 1 }
      };
    case "CHANGE_CATEGORY":
      return {
        ...state,
        filter: { ...state.filter, category: action.payload.category },
        pagination: { ...state.pagination, current: 1 }
      };
    case "CHANGE_PAGINATION":
      return {
        ...state,
        pagination: { ...state.pagination, ...action.payload.pagination }
      };
    case "CHANGE_SORT":
      const newVariable = { ...state, sort: action.payload.sort };
      if (isUndefined(newVariable.sort.order)) delete newVariable["sort"];
      return newVariable;
    default:
      throw new Error(`不存在的 action type: ${action.type}`);
  }
};
```

```
const App = () => {
  const [variables, reducerVariables] = useReducer(reducer, INITIAL_STATE);

  const handleChangeKeyword = (e) => {
    reducerVariables({ type: "CHANGE_KEYWORD", payload: { keyword: e.target.value } });
  };

  const handleChangeCategory = (category) => {
    reducerVariables({ type: "CHANGE_CATEGORY", payload: { category } });
  };

  const handleTableChange = (_pagination, _filters, _sorter, extra) => {
    const { action } = extra;
    if (action === "paginate") {
      reducerVariables({
        type: "CHANGE_PAGINATION",
        payload: { pagination: _pagination }
      });
    } else if (action === "sort") {
      reducerVariables({
        type: "CHANGE_SORT",
        payload: { sort: _sorter }
      });
    }
  };

  return // UI layout
}
```

useXXX 自定义

与 React 组件不同的是，自定义 Hook 不需要具有特殊的标识。我们可以自由的决定它的参数是什么，以及它应该返回什么（如果需要的话）。换句话说，它就像一个正常的函数。但是它的名字应该始终以 `use` 开头，这样可以一眼看出其符合 Hook 的规则。

```
import { useState, useEffect } from 'react';
// 查询好友状态
function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    // 修改好友状态
    ChatAPI.findFriendStatus(friendID, (status)=>{
      setIsOnline(status.isOnline);
    });
  });

  return isOnline;
}
```

Class

代码逻辑清晰（构造函数、componentDidMount 等）

不容易内存泄漏

Hooks

需要配合变量名和注释

容易发生内存泄漏

谢谢

「狗哥」