React Hooks 详解

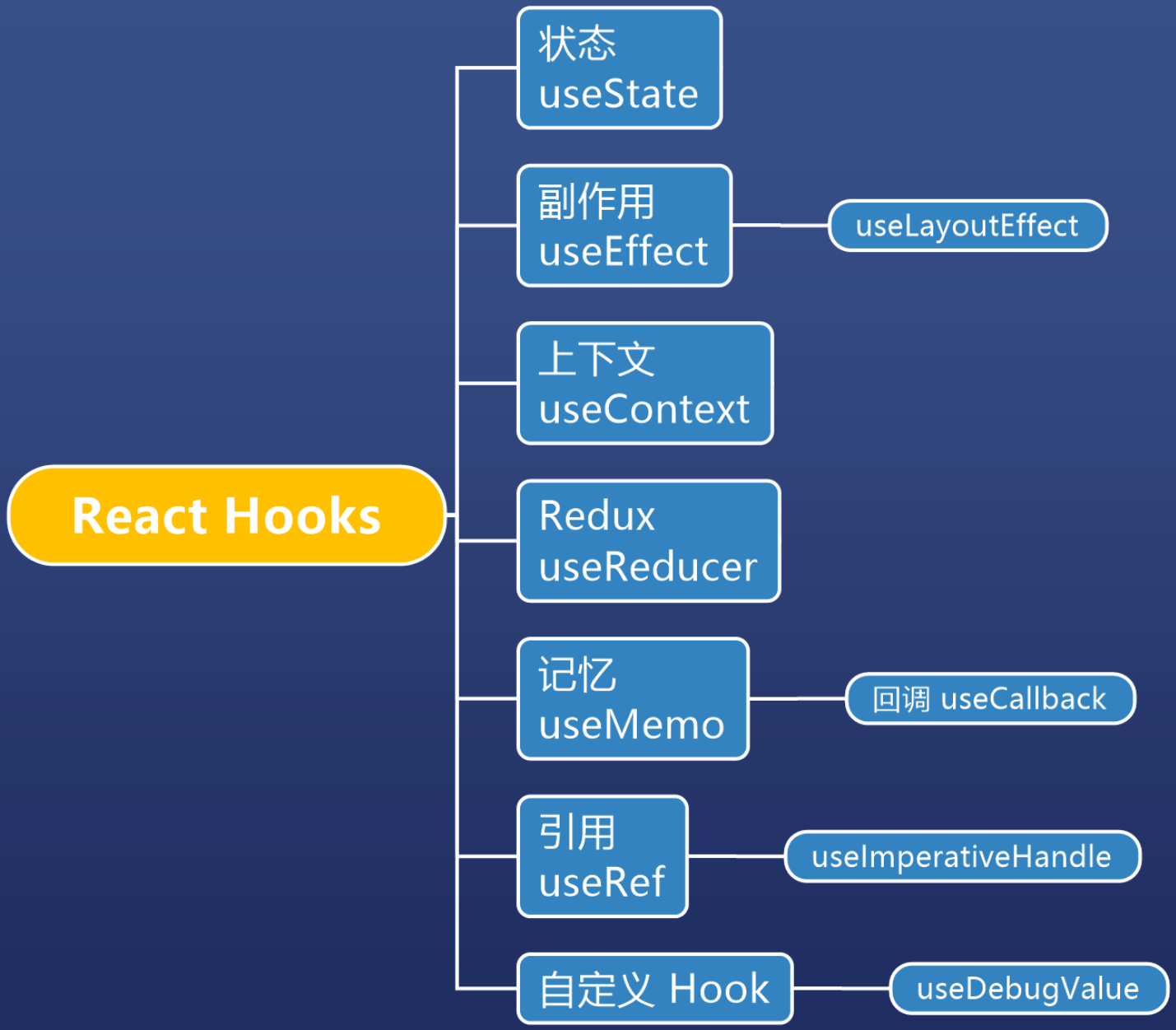
React 造轮子 - 理论课

版权声明

联系方式

如果你想要购买本课程
请微信联系 xiedaimala02 或 xiedaimala03

如果你发现有人盗用本课程
请微信联系 xiedaimala02 或 xiedaimala03


React Hooks
状态
useState
副作用
useEffect
useLayoutEffect
上下文
useContext
Redux
useReducer
记忆
useMemo
回调 useCallback
引用
useRef
useImperativeHandle
自定义 Hook
useDebugValue

useState

- 使用状态
  - const [n, setN] = React.useState(0)
  - const [user, setUser] = React.useState({name:'F'})
- 注意事项1：不可局部更新
  - 如果 state 是一个对象，能否部分 setState?
  - 答案是不行，请看示例代码
  - 因为 setState 不会帮我们合并属性
  - 那么 useReducer 会合并属性吗？妈的，也不会！
- 注意事项2：地址要变
  - setState(obj) 如果 obj 地址不变，那么 React 就认为数据没有变化

useState 续

- useState 接受函数

```
const [state, setState] = useState(()=>{
  return initialState
})
```

- ✓ 该函数返回初始 state，且只执行一次

- setState 接受函数

- ✓ `setN(i => i + 1)}`
- ✓ 什么时候用这种方式？看代码
- ✓ 如果你能接受这种形式，应该优先使用这种形式

useReducer

- 用来践行 Flux/Redux 的思想
  - ✓ 看代码，共分 4 步走
  - ✓ 一、创建初始值 initialState
  - ✓ 二、创建所有操作 reducer(state, action)
  - ✓ 三、传给 useReducer，得到读和写 API
  - ✓ 四、调用写({type:'操作类型'})
  - ✓ 总得来说 useReducer 是 useState 的复杂版
  - ✓ 示例代码
- 一个用 useReducer 的表单例子
  - ✓ 链接

如何代替 Redux

- 步骤
  - 一、将数据集中在一个 store 对象
  - 二、将所有操作集中在 reducer
  - 三、创建一个 Context
  - 四、创建对数据的读写 API
  - 五、将第四步的内容放到第三步的 Context
  - 六、用 Context.Provider 将 Context 提供给所有组件
  - 七、各个组件用 useContext 获取读写 API
- 例子
  - 代码
  - 如何模块化？其实很简单，代码

useContext

- 上下文
  - 全局变量是全局的上下文
  - 上下文是局部的全局变量
- 使用方法
  - 之前的例子已经介绍过了
  - 一、使用 C = createContext(initial) 创建上下文
  - 二、使用 <C.provider> 圈定作用域
  - 三、在作用域内使用 useContext(C) 来使用上下文
  - 代码示例

useContext 注意事项

- 不是响应式的
  - ✓ 你在一个模块将 C 里面的值改变
  - ✓ 另一个模块不会感知到这个变化

useEffect

- 副作用（API 名字叫得不好）
  - ✓ 对环境的改变即为副作用，如修改 document.title
  - ✓ 但我们不一定非要把副作用放在 useEffect 里
  - ✓ 实际上叫做 afterRender 更好，每次 render 后运行
- 用途
  - ✓ 作为 componentDidMount 使用，[ ] 作第二个参数
  - ✓ 作为 componentDidUpdate 使用，可指定依赖
  - ✓ 作为 componentWillUnmount 使用，通过 return
  - ✓ 以上三种用途可同时存在
- 特点
  - ✓ 如果同时存在多个 useEffect，会按照出现次序执行

useLayoutEffect

- 布局副作用（啥玩意）
  - ✓ useEffect 在浏览器渲染完成后执行
  - ✓ 例子
  - ✓ useLayoutEffect 在浏览器渲染前执行
  - ✓ 通过时间点来侧面证明（此处要画图说明）
- 特点
  - ✓ useLayoutEffect 总是比 useEffect 先执行
  - ✓ useLayoutEffect 里的任务最好影响了 Layout
- 经验
  - ✓ 为了用户体验，优先使用 useEffect（优先渲染）

useMemo

- 要理解 React.useMemo
  - ✓ 需要先讲 React.memo
  - ✓ React 默认有多余的 render
  - ✓ 讲代码中的 Child 用 React.memo(Child) 代替
  - ✓ 如果 props 不变，就没有必要再次执行一个函数组件
  - ✓ 最终效果：代码
- 但是
  - ✓ 这玩意有一个 bug：代码
  - ✓ 添加了监听函数之后，一秒破功
  - ✓ 因为 App 运行时会在次执行第 12 行，生成新的函数
  - ✓ 新旧函数虽然功能一样，但是地址不一样！
  - ✓ 怎么办？用 useMemo：代码

useMemo

- 特点
  - ✓ 第一个参数是 ()=> value，见定义
  - ✓ 第二个参数是依赖 [m,n]
  - ✓ 只有当依赖变化时，才会计算出新的 value
  - ✓ 如果依赖不变，那么就重用之前的 value
  - ✓ 这不就是 Vue 2 的 computed 吗?
- 注意
  - ✓ 如果你的 value 是个函数，那么你就要写成 `useMemo(()=> (x) => console.log(x))`
  - ✓ 这是一个返回函数的函数
  - ✓ 是不是很难用？于是就有了 useCallback

useCallback

- 用法
  - ✓ `useCallback(x => log(x), [m])` 等价于
  - ✓ `useMemo(() => x => log(x), [m])`

useRef

- 目的
  - 如果你需要一个值，在组件不断 render 时保持不变
  - 初始化：const count = useRef(0)
  - 读取： count.current
  - 为什么需要 current?
  - 为了保证两次 useRef 是同一个值（只有引用能做到）
  - 此处需要画图解释
- 延伸
  - 看看 Vue 3 的 ref
  - 初始化：const count = ref(0)
  - 读取：count.value
  - 不同点：当 count.value 变化时，Vue 3 会自动 render

useRef

- 能做到变化时自动 render 吗?
  - ✓ 不能!
  - ✓ 为什么不能? 因为这不符合 React 的理念
  - ✓ React 的理念是 UI = f(data)
  - ✓ 你如果想要这个功能，完全可以自己加
  - ✓ 监听 ref，当 ref.current 变化时，调用 setX 即可
- 不想自己加?
  - ✓ 那你就用 Vue 3 吧，Vue 3 帮你加好了

forwardRef

- 讲了 useRef 就不得不讲一下它了
  - 代码1：props 无法传递 ref 属性
  - 代码2：实现 ref 的传递
  - 代码3：两次 ref 传递得到 button 的引用
- useRef
  - 可以用来引用 DOM 对象
  - 也可以用来引用普通对象
- forwardRef
  - 由于 props 不包含 ref，所以需要 forwardRef
  - 为什么 props 不包含 ref 呢？因为大部分时候不需要

useImperativeHandle

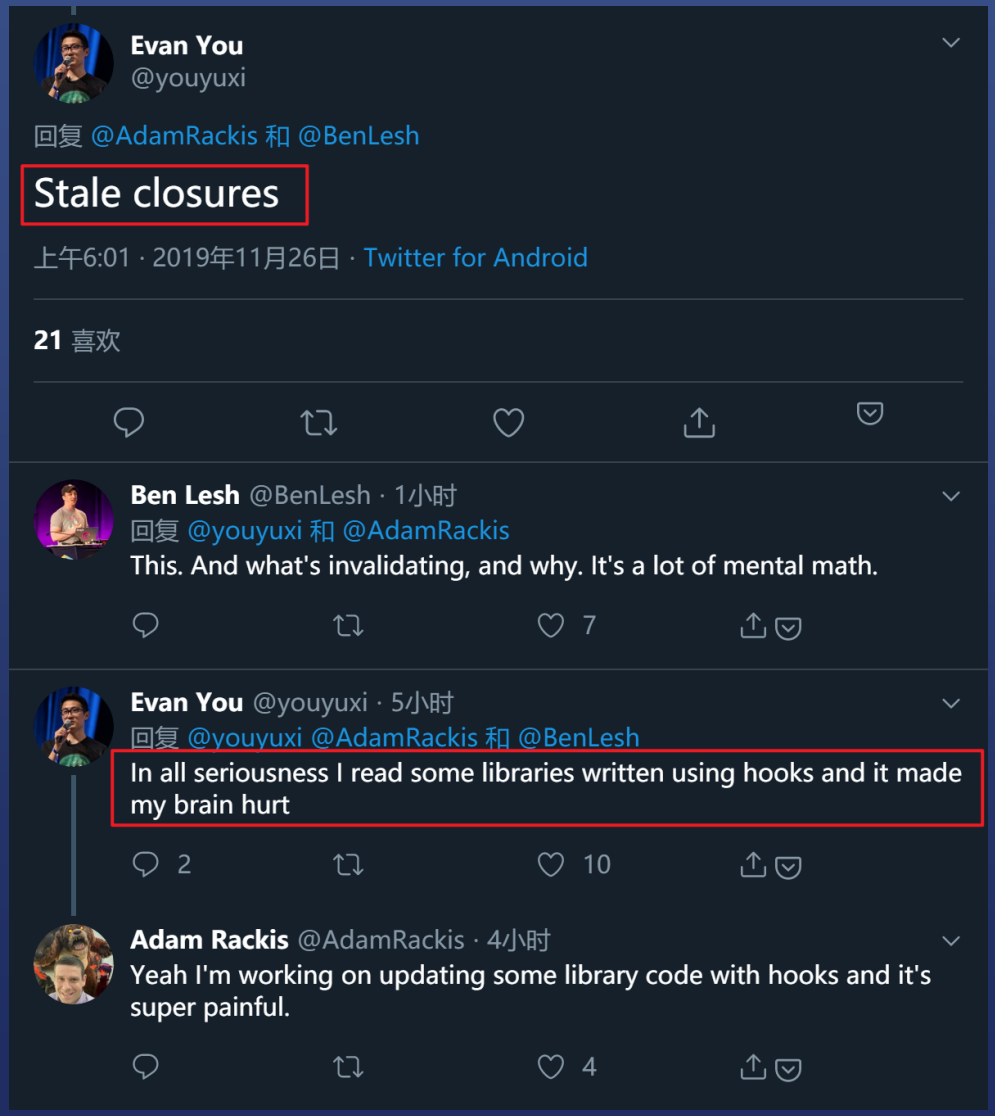
- 名字起得稀烂
  - ✓ 应该叫做 setRef
  - ✓ 不用 useImperativeHandle 的代码
  - ✓ 用了 useImperativeHandle 的代码
- 分析
  - ✓ 用于自定义 ref 的属性

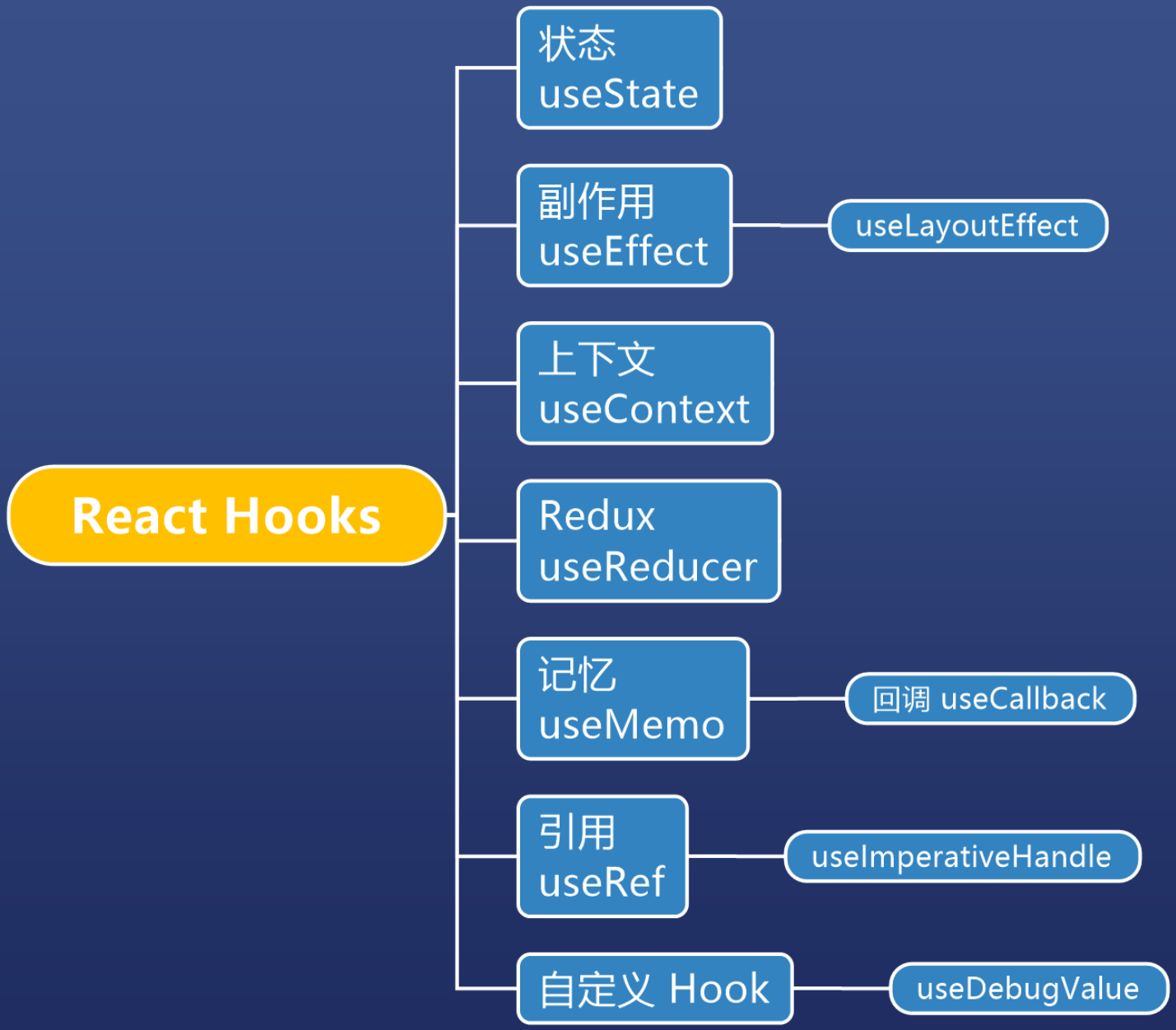
自定义 Hook

- 封装数据操作
  - 简单例子
  - 贴心例子
- 分析
  - 你还可以在自定义 Hook 里使用 Context
  - useState 只说了不能在 if 里，没说不能在函数里运行，只要这个函数在函数组件里运行即可

Stale Closure

- 过时闭包
  - ✓ 参考文章链接




React Hooks
状态
useState
副作用
useEffect
useLayoutEffect
上下文
useContext
Redux
useReducer
记忆
useMemo
回调 useCallback
引用
useRef
useImperativeHandle
自定义 Hook
useDebugValue