答疑解惑01-如何转换应用ReactHooks的思路?

你好，我是王沛。今天我们来对第一章的内容进行集中答疑。

不知不觉我们的基础篇已经讲完了，也很高兴看到你在交流区积极留言，提出了很多有意义的问题。所以这节课呢，我会针对一些具有代表性的问题，进行集中答疑。一方面算是对课程内容作一个有针对性的补充，另外一方面也希望能对更多的同学有所帮助。

我会先对评论区的提出的几个具有代表性的问题进行讲解，然后再对课程中的几个思考题，做一个回答参考，并给出代码示例。

留言区的问题

02讲

问题：文中的例子里说，窗口大小发生变化，组件就会更新。对于这一点我不太理解，Class封装的还可以理解为，state发生改变了，导致重新render了，但Hooks感觉这么理解并不通顺，Hook哪里写得就类似一个纯函数调用，是怎么驱动组件重新更新的呢?

讲解：一个很好的问题，这是很多初学 Hooks 的同学都会有的困惑：一个普通函数怎么就让组件刷新了呢？其实答案也特别简单，那就是**自定义 Hooks 中也是通过 useState 这样的内置 Hook 来完成组件的更新的**。

可能你会觉得，自定义 Hooks 中一定会用到 state 吗？如果你写多了就会发现，自定义 Hooks 要实现的逻辑，要么用到 state，要么用到副作用，是一定会用到内置 Hooks 或者其它自定义 Hooks 的。

如果对比 Class 组件，state 发生变化，导致重新 render。这个在 Hooks 中是完全一样的，也是 state 发生变化，导致重新 render，只是我们可以在 Hooks 这样额外的函数里去 set state，而 Class 组件只能在当前 Class 中 set state。

03讲

问题1：函数体也是每次render都会执行，那么，需要每次都会render执行的语句是放在 无依赖的useEffect中呢，还是直接放在函数体中比较好呢?

讲解：这两种情况的语义是不一样的。useEffect 代表副作用，是在函数render 完后执行。而函数体中的代码，是直接影响当次 render 的结果。

所以在写代码的时候，我们一定要理解每个 API 的语义，副作用一定是和当前 render 的结果没关系的，而只是 render 完之后做的一些额外的事情。

问题2：老师你好，我看到 redux 官网实现 useActions 函数，让我很困惑：

地址：https://react-redux.js.org/api/hooks#recipe-useactions

摘录源码，它的依赖数组是动态的，这肯定是不对的，但是如何在 eslint-plugin-react-hooks 规则下写这个函数的呢？：

```react
import { bindActionCreators } from 'redux'
import { useDispatch } from 'react-redux'
import { useMemo } from 'react'

export function useActions(actions, deps) {
const dispatch = useDispatch()
return useMemo(
() => {
if (Array.isArray(actions)) {
return actions.map(a => bindActionCreators(a, dispatch))
}
return bindActionCreators(actions, dispatch)
},
// 这个依赖数组不是常量的
deps ? [dispatch, ...deps] : [dispatch]
)
}
```

讲解：需要注意的是，ESLint 的作用是帮助你发现可能存在的错误，而Hooks 本身并不需要依赖数组是常量，只要你确定写法没有问题，那么这种可以忽略 eslint 的配置，比如加上 //eslint-disable-line 这样的注释。类似的，如果要用 Hooks 实现一个 componentDidMount 这样的功能，我们看到是需要传递一个空的数组作为 useEffect 的依赖项，那么这时候即使副作用内部使用了某些变量，那么只要你确定它只有第一次需要用到，后面无需再关心，那么也可以在这一行禁用 ESLint 的检查。所以 ESLint 主要是一个辅助的作用，代码的正确性是可以完全由自己来判断的。

04讲

问题1：老师，有两种写法，请问在性能方面是否后者优于前者？写法如下：

```
const handleIncrement = useCallback(() => setCount(count + 1), [count]);
```

```
const handleIncrement = useCallback(() => setCount(q => q + 1), []);
```

我的理解是这样的：后者只创建了一次函数，但是又调用了多次在setCount的回调函数。前者只会在count变化的时候创建新的回调函数。这样分析下来我又觉得两者没什么差异。我不是太清楚这两者的优缺点，希望得到老师的解答。

讲解：确实后者是更好的写法，因为 handleIncrement 不会每次在 count 变化时都使用新的。从而接收这个函数的组件 props 就认为没有变化，避免可能的性能问题。

但是有时候如果 DOM 结构很简单，其实怎么写都没什么影响。但两种代码实际上都是每次创建函数的，只是第二种写法后面创建的函数是被 useCallback 忽略的。

所以这里也看到了 setState 这个 API 的另外一种用法，就是可以接收一个函数作为参数：setSomeState(previousState => {})。这样在这个函数中通过参数就可以直接获取上一次的 state 的值了，而无需将其作为一个依赖项。这样做可以减少一些不必要的回调函数的创建。

问题2：是任何场景，函数都用useCallback包裹吗？那种轻量的函数是不是不需要？

讲解：对于简单的 DOM 结构，或者函数不被作为属性传递到依赖或者组件的属性，那么用不用 useCallback 都可以。和函数是否轻量无关，主要和组件的复杂度有关。但是始终使用 useCallback 是个比较好的习惯。

课后思考题

接下来主要是针对第6、7讲的思考题进行回答。前几节课的思考题，我看到评论区已经有非常优秀的回答，而且有的同学也贴出了自己的代码示例，这都是非常不错的学习方法。而且我也把这些优秀的回答进行了置顶，方便大家交流讨论。

06讲

题目：在 useCounter 这个例子中，我们是固定让数字每次加一。假如要做一个改进，允许灵活配置点击加号时应该加几，比如说每次加10，那么应该如何实现？

讲解：这里要考察的是你有没有意识到 Hooks 就是普通函数，是可以给它传递任意参数的。所以我们只要由调用这决定加几就可以了：

```
function useCounter(n) {
  // 定义 count 这个 state 用于保存当前数值
  const [count, setCount] = useState(0);
  // 实现加 n 的操作
  const increment = useCallback(() => setCount(count + n), [count]);
  // 实现减 n 的操作
  const decrement = useCallback(() => setCount(count - n), [count]);
  // 重置计数器
  const reset = useCallback(() => setCount(0), []);

  // 将业务逻辑的操作 export 出去供调用者使用
  return { count, increment, decrement, reset };
}
```

07讲

题目：只考虑 Redux 部分，对于计数器应用，目前每次是固定加减1，如果要能够在每次调用时增加或减少指定的变量值，应该如何实现？

讲解：这和上一讲的思考题几乎一样，只是这里考察的是，有没有注意到 Redux 的 action 就是一个普通的 object，我们可以在其中加入任何需要的参数，只要 reducer 能处理就可以了。

代码如下：

```
const incrementAction = {
  type: 'counter/incremented',
  n: 5, // 实现每次加5
};

function counterReducer(state = initialState, action) {
  switch (action.type) {
    case 'counter/incremented':
      // 从 action 中去拿每次加几
      return { value: state.value + action.n }
    default:
      return state
  }
```

好了，这次的答疑课就是这些内容。欢迎同学们更积极的留言互动，确保每一个知识点都能准确掌握。

精选留言：

- Geeker 2021-06-12 12:21:17
  我是假期学习第一人