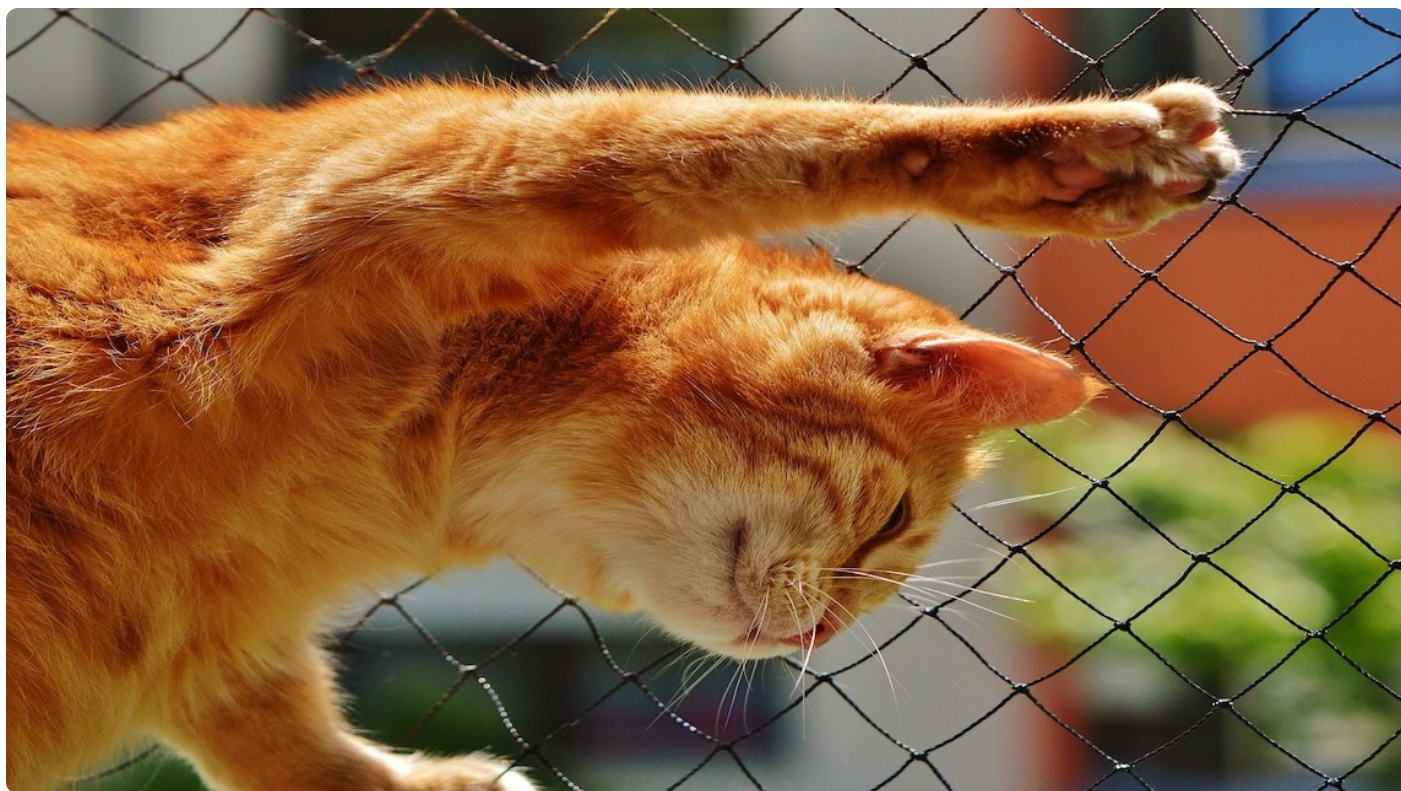


04 | State: 如何让页面“动”起来？

2022-04-04 蒋宏伟

《React Native 新架构实战课》

课程介绍 >



讲述：蒋宏伟

时长 20:59 大小 19.23M



你好，我是蒋宏伟。

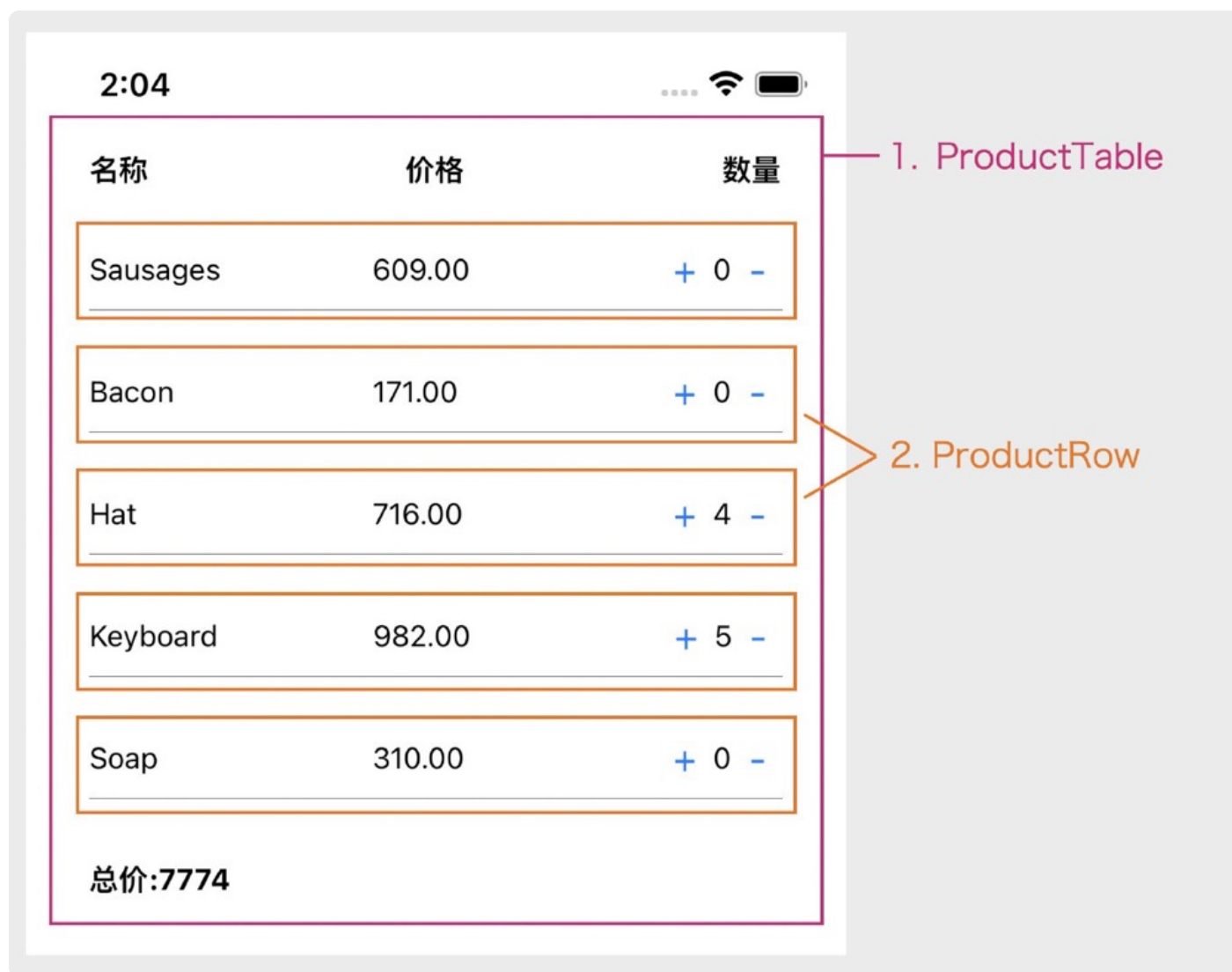
那么这一讲，我们来讲搭建页面的第二步，让页面“动”起来，这里的“动”说的是在不同场景下，让页面展示出不同的内容。

怎么让页面“动”起来呢？这就要用到状态 **State** 了。

一个页面也好，一个应用也好，只有把状态设计清楚了，程序才能写得好。讲到状态，有些人可能会说，状态不就是页面中那些会“动”的数据吗？这很简单，还有什么好讲的。

这没错，状态确实是页面中会“动”的数据，但是要把状态用好不容易，有时候容易把状态设计复杂了，不仅代码要写得更多，还容易导致程序维护起来更麻烦。

这次，我会以搭建一个会“动”的简易购物车页面为例，和你分享下我在这方面的经验。简易购物车页面是这样的：



它比上一讲的商品表单页多了一些交互，它的所有数据都是从网络请求过来的，这些数据包括商品名称、商品价格、商品数量，数据从网络请求回来后会展示在页面上。你可以点击页面中的加号或减号，来添加数量或减少商品数量，底部的结算总价会随着商品数量的变化而变化。

要实现这个简易购物车的静态很简单，它只包括两个组件，商品表单组件 **ProductTable** 和商品组件 **ProductRow**。完成静态页面的搭建后，接下来就要让页面“动”起来了，我把这个过程分成了 4 步来实现，状态初选、状态确定、状态声明、状态更新。

第一步：状态初选

状态初选说的是，先看看页面那些数据是会变化的，这些会变化的数据都可能是状态，我们先把它们找出来。

程序本身的事件和用户操作都有可能导致页面发生变化，因此我们从这两个方向来进行初选。

程序本身的事件，比如网络请求、`setTimeout`，都可能导致页面发生变化。在购物车页面中，商品列表的数据是从服务端请求过来，在列表数据从服务端回来之前，页面是空白的，在请求过程中会有加载提示，请求成功后购物车页面就会展示出来，当然还要把请求失败的情况考虑进去。

所以，我圈选出来的第一条动态数据是请求过程，第二条是可能的请求失败情况；第三条是商品表单本身，第四条是所有商品的结算总价。

用户操作，比如点击、滑动、缩放，也可能导致页面发生变化。在购物车页面中，用户点击加号购物车中的商品数量会增加，点击减号数量会减少。反映商品数量的数据，就是第五条的动态数据。

现在，我已经把这 5 个初选的状态给你在图中标出来了：

2:04

....  

名称	价格	数量
Sausages	609.00	<div>⑤ + 0 -</div>
Bacon	171.00	<div>+ 0 -</div>
Hat	716.00	<div>+ 4 -</div>
Keyboard	982.00	<div>+ 5 -</div>
Soap	310.00	<div>+ 0 -</div>
总价:7774		

1. ProductTable

①请求中
②请求失败
③商品列表
④结算总价

2. ProductRow

⑤商品数量

第二步：状态确定

有些人完成状态初选这一步后，就直接写起代码来了，一口气声明了 5 个状态，然后想办法去操作这 5 个状态如何变化。

但我的经验是，状态初选完成后，不能急着写代码，要先确定一下这些初选状态中那些是真正的状态，把其中无用的状态剔除掉，然后再去写代码。这样代码写得少、写得快，代码逻辑也会更简单一些，也更难出 BUG 一些。

这些都是我们要在状态确定这一步要做的，我总结了三条经验：

首先，一件事情一个状态。我发现有些同学写代码的时候，在定义请求状态时，喜欢用布尔值 `isLoading` 来表示空闲状态或请求中的状态，用 `isError` 来表示成功状态或失败状态，明明就是网络请求这一件事，却用了两个状态来表示，这就有点多余了，甚至在一些不好测试的边界条件下可能还会留坑。

这时其实只需要定义一个状态，代码示例如下：

```
1  const requestStatus = {  
2    IDLE : 'IDLE',  
3    PENDING : 'PENDING',  
4    SUCCESS : 'SUCCESS',  
5    ERROR : 'ERROR',  
6  }
```

 复制代码

这里，我定义的是一个枚举对象 `requestStatus`，用它来表示请求状态。这个对象有 4 个值，包括请求空闲 `IDLE`、请求中 `PENDING`、请求成功 `SUCCESS`、请求失败 `ERROR`。你看，用一个状态是不是比用两个状态更加贴合请求的实际情况呢？

第二，重复状态不是状态。商品组件 `ProductRow` 中的这个商品数量确实是一个状态，但它却和从网络请求中回来的商品表单状态重复了。从代码层面上，我们确实有办法同时保留两个状态，但这样做就绕弯子了。

更好的做法是，把这两个在不同组件之间的重复状态进行合并，去掉底层组件的重复状态，只保留顶层组件中的商品数量作为唯一的状态。

最后，可计算出来的状态不是状态。一个状态必须不能通过其他状态、属性或变量直接计算出来，能通过其他值计算出来的状态，都不是状态。比如，在购物车页面中，结算总价这个动态数据，是可以通过对所有商品的单价和数量的积进行求和得出来的，所以它不是状态。

 复制代码

```
1  结算总价 = Σ(商品的单价 * 数量)
```

初选的状态经过以上三步筛选之后，合并了①请求中和②请求失败，避免了重复数据③商品列表和⑤商品数量，根据已有状态推导出了④结算总价。初选状态一共 5 个，最终确定下来就只剩下网络请求状态和商品列表这两个状态了。

只有确定状态这一步做好了，你声明状态和改变状态，才会变得容易。

第三步：状态声明

React 提供了 `useState` 状态声明函数，你可以用它来管理函数组件的状态。

使用 `useState` 声明状态时，状态和组件是绑定的关系，`useState` 在哪个组件中使用，它生成的状态就属于那个组件。由于单向数据流的原因，**React** 把父组件的状态传给子组件只需要传一次，传给子子组件需要传递两次。

因此，你在定义状态的时候，一定要先考虑好把状态绑定到哪个组件上。我建议你用**就近原则**来绑定状态，就近原则的意思是哪个组件用上了状态，就优先考虑将状态绑定到该组件上，如果有多个组件使用了同一个状态，则将其绑定到最近的父组件上。这样做能让使用 `props` 传递状态的次数最少。

在我们的购物车案例里，你使用就近原则判断一下，你就可以确定购物车页面的两个状态，其实放在 `ProductTable` 组件中比较合适。

那接下来的问题是，状态声明代码应该如何写？

首先，我需要和你强调的是，在 **React/React Native** 中，所有使用 `use` 开头的函数，比如接下来要用到 `useState` 函数，它们都叫**钩子函数（hook function）**。和普通函数不同，你不能把钩子函数写在 `if` 条件判断中、事件循环中、嵌套的函数中，这些都会导致报错。

钩子函数类似于 JavaScript 的 `import`，你最好在函数组件的顶部使用它们。

具体声明商品表单状态 `products` 和请求状态 `requestStatus` 代码是这样的：

 复制代码

```
1 import React, {useState} from 'react';
2
3 export default function ProductTable() {
4   const [products, setProducts] = useState([]);
5   const [requestStatus, setRequestStatus] = useState('IDLE');
6   // ...
7 }
```

在文件的第一行代码中，我们从 `React` 中引入 `useState` 函数，然后在函数组件 `ProductTable` 的顶部使用了 `useState` 声明了两个状态。

`useState` 函数的入参是状态的默认值，函数的返回值是状态和更新该状态的函数。第一次调用 `useState` 函数后，就生成了默认值是空数组 `[]` 的商品表单状态 `products`，以及设置该状态的函数 `setProducts`。第二次调用 `useState` 函数后生成了默认值是字符串 `'IDLE'` 的请求状态 `requestStatus`，以及设置该状态的函数叫做 `setRequestStatus`。

如果在 `if` 中使用了任何的钩子函数，就会报错：

 复制代码

```
1 import React, {useState, useEffect} from 'react';
2
3 // 错误
4 export default function ProductTable() {
5   const [requestStatus, setRequestStatus] = useState('IDLE');
6   // ...
7   if(requestStatus === 'ERROR') return <Text>网络出错了</Text>
8
9   // 在 else 分支中，使用任何 use 开头的钩子函数，都会报错
10  const [products, setProducts] = useState([]);
11  useEffect(() => {})
12
13  return <Text>购物车页面</Text>
14 }
```

在这个错误示例中，我们先使用了 `if(requestStatus === 'ERROR')` 判断了网络请求状态。如果请求失败，则提示用户“网络出错了”，否则就返回真正的购物车页面。但 `if` `return` 后面的代码，就相当于 `else` 分支，在分支中使用了钩子函数，比如 `useState`、`useEffect`，代码就会报错。

出现这种报错，是因为 `if` 破坏了 React 的 [Hook 规则](#)。在 React 的 Hook 机制中，是把 Hook 的调用顺序作为索引，用它把 React 框架内部 `state` 和其函数组件的 `useState` 返回值中的 `state` 给关联起来了。当你使用了 `if` 的时候，就容易破坏 Hook 的调用顺序，导致 React 不能正确地将框架内部 `state` 与函数组件 `useState` 的返回值关联起来，因此 React 在执行的时候就会报错。

有时候一个函数组件很长，写到后面了，前面的一些逻辑就记不那么清楚了，如果代码写到哪就在哪儿声明一个新状态，一不小心就可能会踩坑。因此，你应该把 `use` 开头的钩子函数都写在组件的顶部，把 `JSX` 都写在函数组件的最后面，并使用 [eslint-plugin-react-hooks](#) 插件来保障 Hook 规则的会被正确执行。

第四步：状态更新

现在，到了最后一步了。不过，这一步中涉及购物车页面业务实现逻辑的部分，我就不一一介绍了，具体实现代码我放到了附加材料中，这里我想重点和你强调的是如何更新对象类型的状态。

在 JavaScript 中的数据类型可以分为两类，对象数据类型（`Objects`）和原始数据类型（`Primitive values`），对象数据类型包括对象（`Object`）、数组（`Array`），原始数据类型有 7 种，比如数字（`number`）、字符串（`string`）等等。

在 `React/React Native` 中，使用这两类数据类型作为状态都是可以的，但是更新这两类状态的方法不一样，如果你没有理解清楚二者的区别，就容易出现一些低级的 `BUG`。

我们先来看原始数据类型的状态如何更新。

在购物车页面中，商品数量可以通过点击加号进行加一，通过点击减号减一。我们用原始数据类型数字来表示商品数量状态，其代码实现如下：

```

1 export default function Count() {
2   const [count, setCount] = useState(0);
3
4
5   return (
6     <View>
7       <Text>{count}</Text>
8       <Button title="+" onPress={() => setCount(count + 1)} />
9       <Button title="-" onPress={() => setCount(count - 1 >= 0 ? count - 1 : 0)} />
10    </View>
  );
};

```

你可以看到，我们使用 `useState` 声明了商品数量状态 `count` 和更新状态的函数 `setCount`。

组件初始化时，也就是组件函数第一次调用时，商品数量状态 `count` 的默认值是 `0`，页面展示的数字就是 `0`。

当你点击页面中的加号（“+”）时，就会触发加号（“+”）点按组件（`Button`）的点击事件（`onPress`），此时会调用 `setCount` 函数更新状态。

`setCount` 函数的入参是 `count + 1`，其中 `count` 取的是组件第一次调用的默认值 `0`， $0 + 1 = 1$ ，因此新状态就是 `1`，之后 `React/ React Native` 会再调用一次组件函数，这一次调用时 `useState` 声明的状态 `count` 的值就是新状态的值 `1`，此时 `Text` 组件收到的值也是 `1`，最后页面刷新展示新状态 `1`。

第二次点击加号时，也是先更新 `React/ React Native` 内部的新状态，将它更新到 `2`，然后再执行一次组件函数，将内部的新状态同步给 `count`，最后刷新页面展示新状态 `2`。

以此类推，**对于原始数据类型而言，调用 `setCount` 更新原始数据类型状态的值，页面就会发生更新。**

那对象和数组类型的状态如何更新呢？

我们先声明一个对象状态和一个数组状态，代码如下：

 复制代码

```

1 const [countObject, setCountObject] = useState({num: 0});
2 const [countArray, setCountArray] = useState([0]);

```


这段代码中，调用了两次 `useState`，声明一个对象状态 `countObject` 和一个数组状态 `countArray`，以及对应的状态更新函数。

理论上，你可以直接改变对象状态或数组状态的值，再调用状态更新函数，代码如下：

 复制代码

```
1 countObject.num++;
2 setCountObject(countObject)
3
4 countArray[0]++;
5 setCountArray(countArray)
```

但是你试过后会发现，调用状态更新函数后，页面什么变化都没有，这是为什么呢？

弄清楚了对象数据类型（**Objects**）和原始数据类型的区别后，你就明白了。我给你举个例子：

 复制代码

```
1 const countObject = {num: 0}
2 countObject.num++;
3
4 countObject.num === countObject.num // false
5 countObject === countObject // true
6 setCountObject(countObject) // 不更新
```

你看，当你更新 `countObject.num` 时，`countObject.num` 确实更新了，但是 `countObject` 的引用并没有更新，所以调用 `setCountObject` 更新状态时，页面没有任何变化。

这是因为，对象它是一种复合数据类型，它内部的值是可变的（**mutable**），但它的引用是不可变了（**immutable**），你更新了对象的内部值后，它的引用并没有发生变化。

那状态是对象或数组时，应该怎么更新呢？

业内也有形似 **mutable** 的更新方案 `useImmer`，可以通过直接修改变量的值来更新状态。但其底层原理也是，新建一个对象或数组传给状态更新函数，让状态更新函数知道对象或数组确实发生了变化，这时 **React/React Native** 框架才会帮你更新页面。

这里，我用的也是直接新建对象、新建数组的方式，代码如下：

 复制代码

```
1 setCountObject({...countObject, num: countObject.num+1});
2
3 const newCountArray = [...newCountArray]
4 newCountArray[0]++;
5 setCountArray(newCountArray)
```

你可以看到，对于对象状态的更新我是这么处理的，我先创建了一个新对象`{}`，然后用`...`的解构的方式将老对象 `countObject` 的内部值重新赋值给了新对象`{}`，再指定`num`属性进行了复写。对于数组状态的更新也是类似的，你可以自己试试。

总结

这一讲，我们完成搭建页面的第二步：让页面“动”起来。让页面“动”起来，就要用到状态，我们这一讲的具体实现分为 4 个步骤，状态初选、状态确定、状态声明、状态更新。

- 状态初选，就是把设计稿中的那些会“动”的数据先选出来；
- 状态确定，就是合并同类状态、删除无用状态和衍生状态；
- 状态声明，在当前的初学阶段，只需要学会使用 `useState` 来声明组件状态即可；
- 状态更新是最后一步。交互事件和程序事件会触发状态的更新，但状态更新函数并不会帮我们自动合并上一个状态，因此在处理对象状态和数组状态时，每次更新时必须新建一个完整的对象或数组。

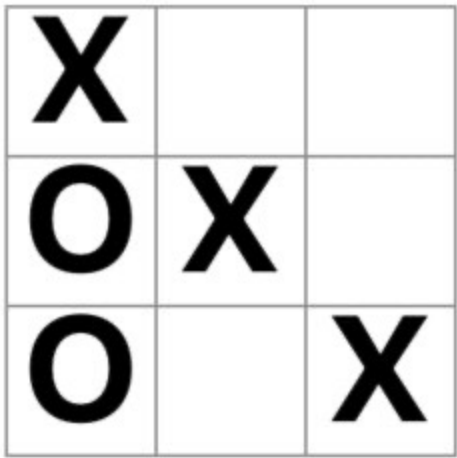
行军作战是兵马未动粮草先行，讲究的是谋而后动。搭建页面、开发组件也是如此，我们也要**代码未动构思先行**，先把组件状态设计好了，简单即美，要是没想清楚弄复杂了，后面填坑成本会很高。

附加材料

1. 再次强调，学习 `React` 最好的材料就是 `React` 新官网，我推荐你去读一读 [🔗 《如何使用状态响应用户的输入时间》](#)、[🔗 《更新状态对象》](#) 和 [🔗 《更新状态数组》](#)。
2. 实现购物车页面的完整代码，我放在了 [🔗 GitHub](#) 上。

作业

1. 请你实现一个井字棋。井字棋的规则和五子棋类似，两人在 $3 * 3$ 格子上进行连珠游戏，任意 3 个标记形成一条直线，则为获胜。在写之前，推荐你先玩一下这个井字棋，了解一下 [井字棋的最终效果](#)。
2. 请你思考一下实现一个井字棋，最少需要声明几个状态？




恭喜 X 获胜！
点击重玩

欢迎在留言区分享你的想法。下一节课我们将来讨论 **React Native** 中图片组件的用法和最佳实践，你可以做些准备。我是蒋宏伟，咱们下节课见。

分享给需要的人，Ta 订阅超级会员，你最高得 50 元

Ta 单独购买本课程，你将得 20 元

 生成海报并分享

 赞 2  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 03 | Style: 关于样式你需要知道的三件事

下一篇 05 | Image: 选择适合你的图片加载方式

精选留言 (1)

写留言



Sunny

2022-04-05

```
export default function ProductTable() {  
  // ...  
}
```

请教一下，这也是声明组件么？为什么不用 `class` 声明？

共 2 条评论 >

👍 1