

React 常考进阶知识点

这一章节我们将来学习 React 的一些经常考到的进阶知识点，并且这章节还需要配合第十九章阅读，其中的内容经常会考到。

HOC 是什么？相比 mixins 有什么优点？

很多人看到高阶组件（HOC）这个概念就被吓到了，认为这东西很难，其实这东西概念真的很简单，我们先来看一个例子。

```
function add(a, b) {  
  return a + b  
}
```

现在如果我想给这个 add 函数添加一个输出结果的功能，那么你可能会考虑我直接使用 `console.log` 不就实现了么。说的没错，但是如果我們想做的更加优雅并且容易复用和扩展，我们可以这样做：

```
function withLog (fn) {  
  function wrapper(a, b) {  
    const result = fn(a, b)  
    console.log(result)  
    return result  
  }  
  return wrapper  
}  
const withLogAdd = withLog(add)  
withLogAdd(1, 2)
```

其实这个做法在函数式编程里称之为高阶函数，大家都知道 React 的思想中是存在函数式编程的，高阶组件和高阶函数就是同一个东西。我们实现一个函数，传入一个组件，然后在函数内部再实现一个函数去扩展传入的组件，最后返回一个新的组件，这就是高阶组件的概念，作用就是为了更好的复用代码。

其实 HOC 和 Vue 中的 mixins 作用是一致的，并且在早期 React 也是使用 mixins 的方式。但是在使用 class 的方式创建组件以后，mixins 的方式就不能使用了，并且其实 mixins 也是存在一些问题的，比如：

- 隐含了一些依赖，比如我在组件中写了某个 state 并且在 mixin 中使用了，就这存在了一个依赖关系。万一下次别人要移除它，就得去 mixin 中查找依赖
- 多个 mixin 中可能存在相同命名的函数，同时代码组件中也不能出现相同命名的函数，否则就是重写了，其实我一直觉得命名真的是一件麻烦事。。
- 雪球效应，虽然我一个组件还是使用着同一个 mixin，但是一个 mixin 会被多个组件使用，可能会存在需求使得 mixin 修改原本的函数或者新增更多的函数，这样可能就会产生一个维护成本

HOC 解决了这些问题，并且它们达成的效果也是一致的，同时也更加的政治正确（毕竟更加函数式了）。

事件机制

React 其实自己实现了一套事件机制，首先我们考虑一下以下代码：

```
const Test = ({ list, handleClick }) => ({
  list.map((item, index) => (
    <span onClick={handleClick} key={index}>
{index}</span>
    ))
  })
})
```

以上类似代码想必大家经常会写到，但是你是否考虑过点击事件是否绑定在了每一个标签上？事实当然不是，JSX 上写的事件并没有绑定在对应的真实 DOM 上，而是通过事件代理的方式，将所有的事件都统一绑定在了 document 上。这样的方式不仅减少了内存消耗，还能在组件挂载销毁时统一订阅和移除事件。

另外冒泡到 document 上的事件也不是原生浏览器事件，而是 React 自己实现的合成事件（SyntheticEvent）。因此我们如果不要事件冒泡的话，调用 `event.stopPropagation` 是无效的，而应该调用 `event.preventDefault`。

那么实现合成事件的目的是什么呢？总的来说在我看来好处有两点，分别是：

- 合成事件首先抹平了浏览器之间的兼容问题，另外这是一个跨浏览器原生事件包装器，赋予了跨浏览器开发的能力
- 对于原生浏览器事件来说，浏览器会给监听器创建一个事件对象。如果你有很多的事件监听，那么就需要分配很多的事件对象，造成高额的内存分配问题。但是对于合成事件来说，有一个事件池专门来管理它们的创建和销毁，当事件需要被使用时，就会从池子中复用对象，事件回调结束后，就会销毁事件对象上的属性，从而便于下次复用事件对象。

小结

你可能会惊讶于这一章节的内容并不多的情况，其实你如果将两章 React 以及第十九章的内容全部学习完后，基本上 React 的大部分面试问题都可以解决。

当然你可能会觉得看的还不过瘾，这不需要担心。我已经决定写一个免费专栏「React 进阶」，专门讲解有难度的问题。比如组件的设计模式、新特性、部分源码解析等等内容。当然这些内容都是需要好好打磨的，所以更新的不会很快，有兴趣的可以持续关注，都会更新链接在这一章节中。