**组件之间的通信**

react**推崇的是单向数据流，自上而下进行数据的传递**，但是由下而上或者不在一条数据流上的组件之间的通信就会变的复杂。解决通信问题的方法很多，如果只是父子级关系，父级可以将一个回调函数当作属性传递给子级，子级可以直接调用函数从而和父级通信。

组件层级嵌套到比较深，可以使用上下文getChildContext来传递信息，这样在不需要将函数一层层往下传，任何一层的子级都可以通过this.context直接访问。

**兄弟关系的组件之间无法直接通信，它们只能利用同一层的上级作为中转站**。而如果兄弟组件都是最高层的组件，为了能够让它们进行通信，必须在它们外层再套一层组件，这个外层的组件起着保存数据，传递信息的作用，这其实就是redux所做的事情。

组件之间的信息还可以通过全局事件来传递。不同页面可以通过参数传递数据，下个页面可以用location.param来获取。**其实react本身很简单，难的在于如何优雅高效的实现组件之间数据的交流。**

## Redux

首先，redux并不是必须的，它的作用相当于在顶层组件之上又加了一个组件，作用是进行逻辑运算、储存数据和实现组件尤其是顶层组件的通信。如果组件之间的交流不多，逻辑不复杂，只是单纯的进行视图的渲染，这时候用回调，context就行，没必要用redux，用了反而影响开发速度。但是如果组件交流特别频繁，逻辑很复杂，那redux的优势就特别明显了。我第一次做react项目的时候并没有用redux，所有的逻辑都是在组件内部实现，当时为了实现一个逻辑比较复杂的购物车，洋洋洒洒居然写了800多行代码，回头一看我自己都不知道写的是啥，画面太感人。

先简单说一下redux和react是怎么配合的。react-redux提供了connect和Provider两个好基友，它们一个将组件与redux关联起来，一个将store传给组件。组件通过dispatch发出action，store根据action的type属性调用对应的reducer并传入state和这个action，reducer对state进行处理并返回一个新的state放入store，connect监听到store发生变化，调用setState更新组件，此时组件的props也就跟着变化。

关于redux的运作方式没看懂，具体请见

<https://github.com/bailicangdu/react-pxq>

看懂了再写到笔记里。

【prop】



