# 如何通俗易懂的理解 Redux?

> 链接：https://www.zhihu.com/question/41312576/answer/90782136

解答这个问题并不困难：唯一的要求是你熟悉React。 不要光听别人描述名词，理解起来是很困难的。 从需求出发，看看使用React需要什么：

- React有props和state: props意味着父级分发下来的属性，state意味着组件内部可以自行管理的状态，并且整个React没有数据向上回溯的能力，也就是说数据只能单向向下分发，或者自行内部消化。 理解这个是理解React和Redux的前提。
- 一般构建的React组件内部可能是一个完整的应用，它自己工作良好，你可以通过属性作为API控制它。但是更多的时候发现React根本无法让两个组件互相交流，使用对方的数据。 然后这时候不通过DOM沟通（也就是React体制内）解决的唯一办法就是提升state，将state放到共有的父组件中来管理，再作为props分发回子组件。
- 子组件改变父组件state的办法只能是通过onClick触发父组件声明好的回调，也就是父组件提前声明好函数或方法作为契约描述自己的state将如何变化，再将它同样作为属性交给子组件使用。 这样就出现了一个模式：数据总是单向从顶层向下分发的，但是只有子组件回调在概念上可以回到state顶层影响数据。这样state一定程度上是响应式的。
- 为了面临所有可能的扩展问题，最容易想到的办法就是把所有state集中放到所有组件顶层，然后分发给所有组件。
- 为了有更好的state管理，就需要一个库来作为更专业的顶层state分发给所有React应用，这就是Redux。让我们回来看看重现上面结构的需求：
    - 需要回调通知state (等同于回调参数) -> action
    - 需要根据回调处理 (等同于父级方法) -> reducer
    - 需要state (等同于总状态) -> store
- 对Redux来说只有这三个要素：
    - action是纯声明式的数据结构，只提供事件的所有要素，不提供逻辑。
    - reducer是一个匹配函数，action的发送是全局的：所有的reducer都可以捕捉到并匹配与自己相关与否，相关就拿走action中的要素进行逻辑处理，修改store中的状态，不相关就不对state做处理原样返回。
    - store负责存储状态并可以被react api回调，发布action.
- 当然一般不会直接把两个库拿来用，还有一个binding叫react-redux, 提供一个Provider和connect。很多人其实看懂了redux卡在这里。
    - Provider是一个普通组件，可以作为顶层app的分发点，它只需要store属性就可以了。它会将state分发给所有被connect的组件，不管它在哪里，被嵌套多少层。
    - connect是真正的重点，它是一个科里化函数，意思是先接受两个参数（数据绑定mapStateToProps和事件绑定mapDispatchToProps），再接受一个参数（将要绑定的组件本身）
    - mapStateToProps：构建好Redux系统的时候，它会被自动初始化，但是你的React组件并不知道它的存在，因此你需要分拣出你需要的Redux状态，所以你需要绑定一个函数，它的参数是state，简单返回你关心的几个值。
    - mapDispatchToProps：声明好的action作为回调，也可以被注入到组件里，就是通过这个函数，它的参数是dispatch，通过redux的辅助方法bindActionCreator绑定所有action以及参数的dispatch，就可

以作为属性在组件里面作为函数简单使用了，不需要手动dispatch。这个mapDispatchToProps是可选的，如果不传这个参数redux会简单把dispatch作为属性注入给组件，可以手动当做store.dispatch使用。这也是为什么要科里化的原因。

- 做好以上流程Redux和React就可以工作了。简单地说就是：
    - 顶层分发状态，让React组件被动地渲染。
    - 监听事件，事件有权利回到所有状态顶层影响状态。