理解 React，但不理解 Redux，该如何通俗易懂的理解 Redux?

网上文章都看不太懂啊，什么action，reducer的，求解释

71 个回答 默认排序

Wang Namelos

953 人赞同了该回答

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：
1. 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。
2. 一般构建的React组件内部可能是一个完整的应用。它自制它。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。
3. 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。
4. 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。
5. 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：
a. 需要回调通知state (等同于回调参数) -> action
b. 需要根据回调处理 (等同于父级方法) -> reducer
c. 需要state (等同于总状态) -> store
对Redux来说只有这三个要素：
a. 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。
b. 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。
c. 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。
a. 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。
b. 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
mapDispatchToP

相关问题

为什么昨天阮一峰老师发布全栈工程师资料中主要学习react和node.js? 21 个回答

Redux有哪些最佳实践? 16 个回答

很迷茫，不知道自己现在是要继续学习React.js 还是系统地学习 JS? 32 个回答

初级前端er如何学习react.js? 8 个回答

哪里有比较成熟的 React.js 项目案例? 28 个回答

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：
1.顶层分发状态，让React组件被动地渲染。
2.监听事件，事件有权利回到所有状态顶层影响状态。

发布于 2016-03-14

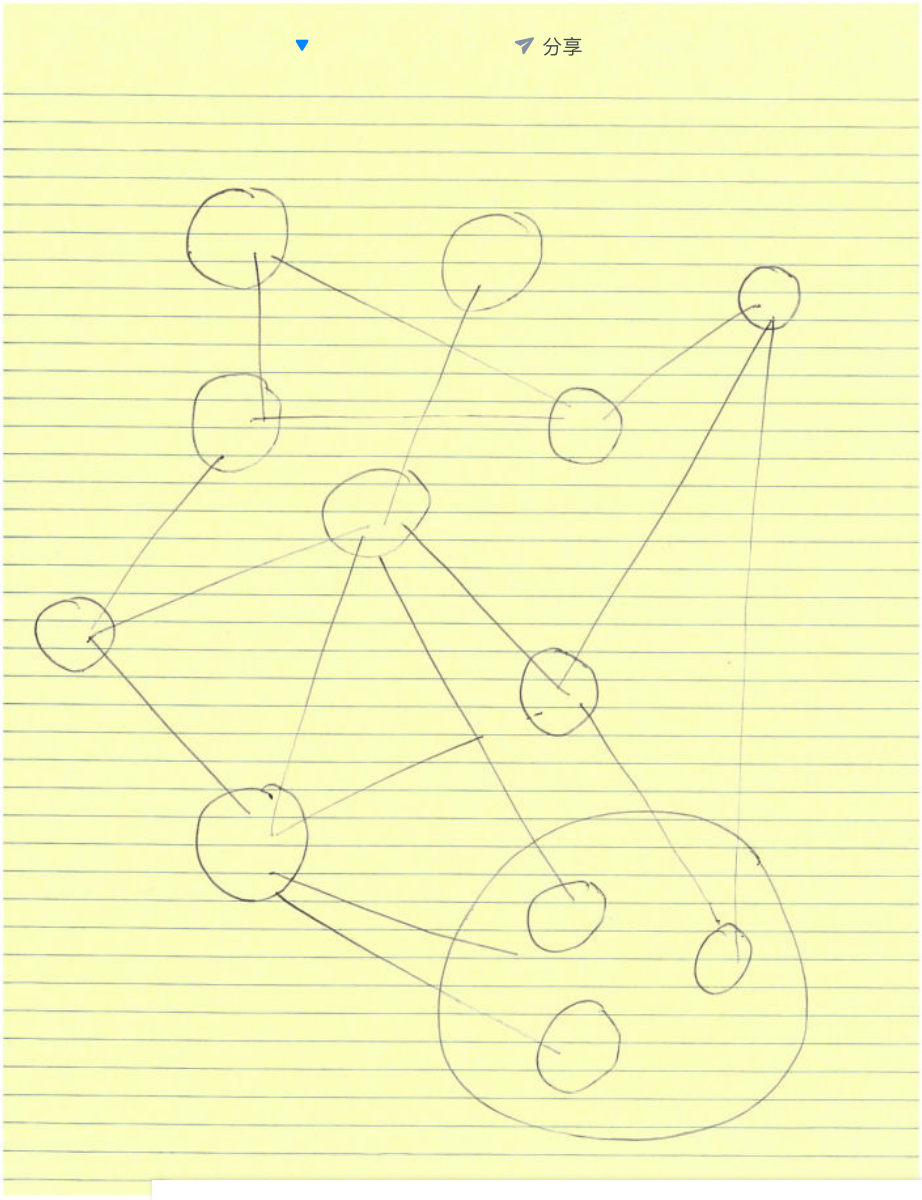
知乎用户
itlr.cc

44 人赞同了该回答

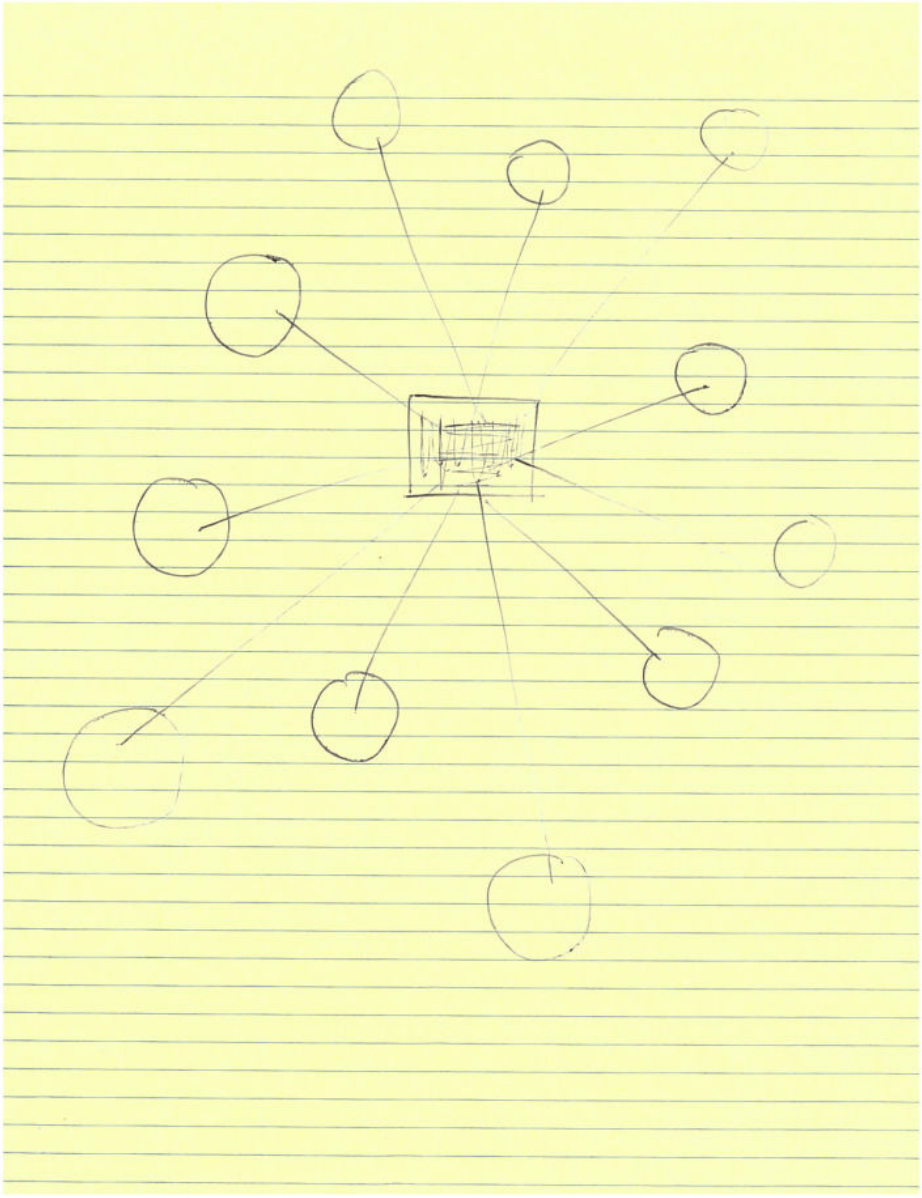
关键是Redux并不“通俗”。

要明白Redux，就要明白为什么需要Redux。

图一，这是一个模块化的项目，这是它的模块“间”通信



图二，这是另一个项目，去除了模块“间”通信，通信统一通过中间那个方块，模块只向这个方块发送通信请求，由方块决定这个请求如何处理，是改变方块内部的某个状态，还是让另一个模块来处理这个请求。



Redux的实现，是很不“通俗”的。有比他更通俗的实现，一直就存在的pubsub，这两年说的要替代Redux的MobX（observable data）。这些东西的架构目的是类似的：更好维护的模块间通信。

action 这些是由模块发起的通信请求，其实就是一个数据对象。

reducer 就是方块内部对这些数据对象的筛选处理过程，为什么叫reducer，想象一下有多个模块依次请求改变方块内一个状态，效率起见，这个状态的最终值是最后一个请求的值，所以用一个reducer来“整理归总”这样的请求，状态改变一次就够了。

编辑于 2017-06-09

尤雨溪
前端开发、JavaScript、前端工程师 话题的优秀回答者

Redux 毕竟是从 Flux 进化而来的，所以不如先看看这个问题下面的回答：如何理解 Facebook 的 flux 应用架构？ - 前端开发

然后你就对 Flux/Redux 的意义大致有所了解了。要打比喻的话， @haochuan 的答案也已经解释的比较直白了。这里我想补充下 Redux 具体的 API 为什么是这个样子，它有哪些独特的设计上的考量：

1. 强调数据的 immutability，但是又不想强制使用 Immutable.js，因此采用了 reducer 这样的设计，当接收到 action 的时候，不是原地修改数据，而是返回一份全新的数据。这个设计巧妙的地方在于不依赖任何库就模拟了 immutable data，但是 reducer 写起来就比较蛋疼，很依赖目前尚在 stage 2 的 object spread 语法。这个设计也有较大一部分原因是因为 React 的渲染机制，immutable 的数据对于 React 的渲染优化非常有利。

2. Redux 的一大卖点就是几乎所有的部件都能热重载。为了能热重载，各个部件都需要尽可能的减少对 singleton 模块的依赖。举例来说你不会在 action creator 里面直接引用 store.dispatch，而是通过 redux-thunk 返回一个函数，这个函数会获得一个 dispatch 函数（有点像依赖注入）。这样 action creator 本身不和任意 store 有直接关系，而是在组件里实际使用时才用 bindActionCreators 和具体的 store 实例绑定。这一点也使得 Redux 对 server-side rendering 友好（因为每一个请求都需要一个全新的 store 实例，如果部件依赖 store singleton 就会很难做）

3. 各个部件最好都是 pure function，至少也得严格限制副作用。这可以使得各部件容易维护、移植、测试。为了在 action creators 里面减少副作用，可以将副作用按类别抽取到 middlewares 当中去管理。

4. Single state tree，方便对整个状态树打快照以及 time-travel。另一个卖点。

Redux 骨子里还是延续了 Flux 的思想，并且引入了大量函数式编程的影响。它的 API 有时候感觉很繁琐，有点绕圈子，一方面是因为这些设计上的取舍，一方面是因为过分强调 functional composition，恨不得所有的 API 都是一个套一个的函数。然而本质上来说，这些函数式的东西并不是 Flux 所必须的元素。相比之下如 Reflux 则是完全的实用主义，对以上这些没有没有考虑，API 怎么直接怎么来，然而也就没有 hot-reload / time-travel 这样的 fancy 功能了。

最后，还是不得不提一下 Vuex：GitHub - vuejs/vuex: Flux-inspired Application Architecture for Vue.js. 倒不是为了安利，只是因为从设计上来说，Vuex 还是受到了 Redux 不少的影响，但是在 Redux 的优点和 API 的直接简单之间做了一个平衡：不强求 immutability，不强求 functional API，但是依然保证部件的可测试、可热重载、time-travel，并且对数据获取默认优化（类似 redux + reselect）。

编辑于 2016-03-13

Cyandev
Independent Developer / Open-Source Contributor

29 人赞同了该回答

高度抽象后就是一个函数：

```
function Redux(action, state, reducer) {
    return reducer(action, state);
}
```

然后这么用：

```
Redux("INCR", 1, function (action, state) {
    if (action === "INCR") {
        return state + 1;
    }
    return state; // Unknown action.
});
```

action 就是你要干的事；state 是当前的状态；reducer 是个纯函数，给定一定的参数会得出一定的结果，这个结果就是新的 state。

编辑于 2017-04-02

知乎用户
Javascripter | Node JSer | Metal党

166 人赞同了该回答

首先感谢楼主这个问题让我在欢乐中写了这篇文章：关于Redux到底是个什么鬼 - _haochuan的RandomNotes - 知乎专栏

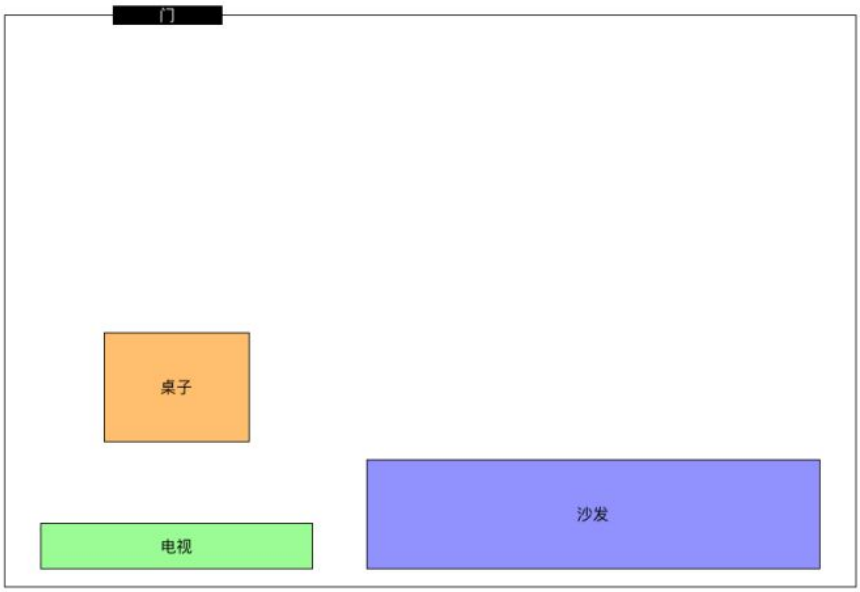
为了本着回答问题的原则，特地把文章内容贴过来供参考。

========== 我是分割线 ==========
========== 我是分割线 ==========
========== 我是分割线 ==========

我们故事的主人公，小明。

小明大学刚毕业，摆脱了宿舍的集体生活，自己在外面租了个一室一厅的小公寓住。

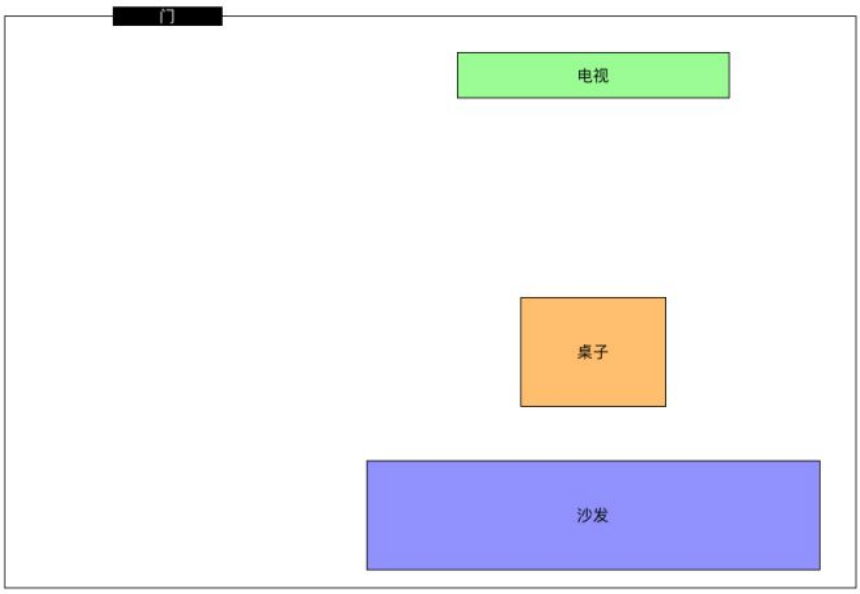
这是客厅的平面图：



一天小明邀请小马来家里做客。小马说：呀你家的家具摆放位置好奇特！这种通过眼睛看到的视觉效果，就是通过React来实现的。每一个家具都是一个component，各种不同的components组成了一个我们的web的‘页面’，或者说是所谓的‘view'。

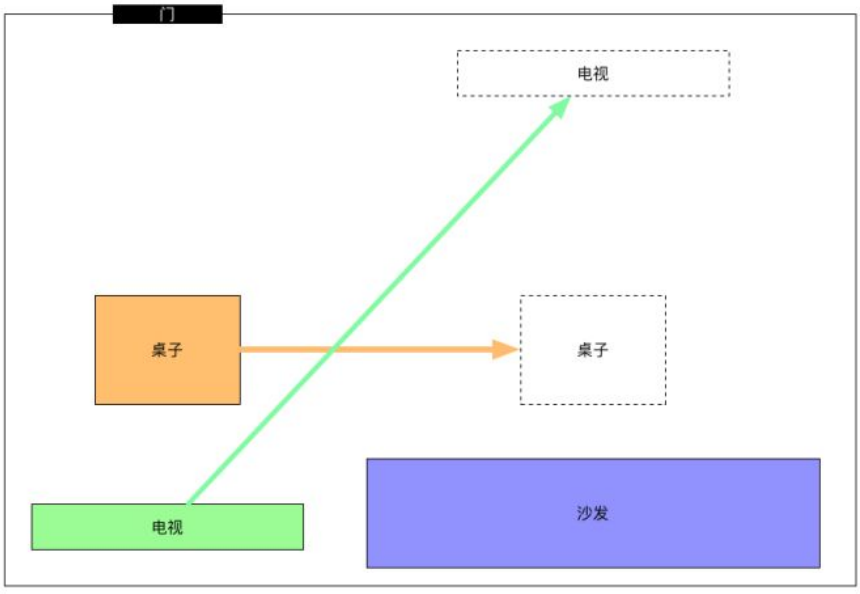
又有一天小马又来家里做客，街边买了50个肉串和10个大腰子，准备和小明一起边撸串儿边看人类和电脑下棋的电视节目。但是小马突然发现，莫名其妙的你妹为什么在这老子坐下来边吃东西边看电视貌似是无法满足的需求啊。这种想法来源于小马与各种家具(components)的一些交互。之后小马跟小明说，我们能不能想点办法来解决这个问题呢?

通过激烈的讨论和商议，小明和小马决定重新摆放家具的位置，然后画出了图纸如下：



有了图纸就要准备干活了。第二天，小明叫来了李雷和韩梅梅来帮忙，小明说：

“李雷，帮我把电视搬到沙发正对面然后靠墙的地方”
“韩梅梅，帮我把桌子向沙发方向平移5米”



十分钟之后，房间内家具的位置变成了像图纸的那样。

问题解决了，第三天。小明，小马，李雷，韩梅梅四个小伙伴在家里快乐的吃起了火锅。

完。

========== 我是分割线 ==========

先让我们回顾下

小马发现家具的摆放不合理 ---> 画出规划图纸 ---> 小明给李雷和韩梅梅分配任务 ---> 李雷和韩梅梅动手搬家具 ---> 家具布局改变

再来说下Redux里的几个核心概念,这里我们把React也加进来：

- view(React)
- store(state)
- action
- reducer

接下来看看Redux/React与这个故事的联系：

- view(React) = 家具的摆放在视觉的效果上
- store(state) = 每个家具在空间内的坐标(如：电视的位置是x:10, y: 400)
- action = 小明分配任务(谁应该干什么)
- reducer = 具体任务都干些什么(把电视搬到沙发正对面然后靠墙的地方)

所以这个过程应该是这样的：

view ---> action ---> reducer ---> store(state) ---> view

如果放入一个web app中，首先store(state)决定了view，然后用户与view的交互会产生action，这些action会触发reducer因而改变state，然后state的改变又造成了view的变化。

发布于 2016-03-12