前端开发核心知识进阶：50 讲从夯实基础到突破瓶颈
来自 Lucas ... · 盐选专栏

查看详情 >

从框架和类库，我们该学到什么

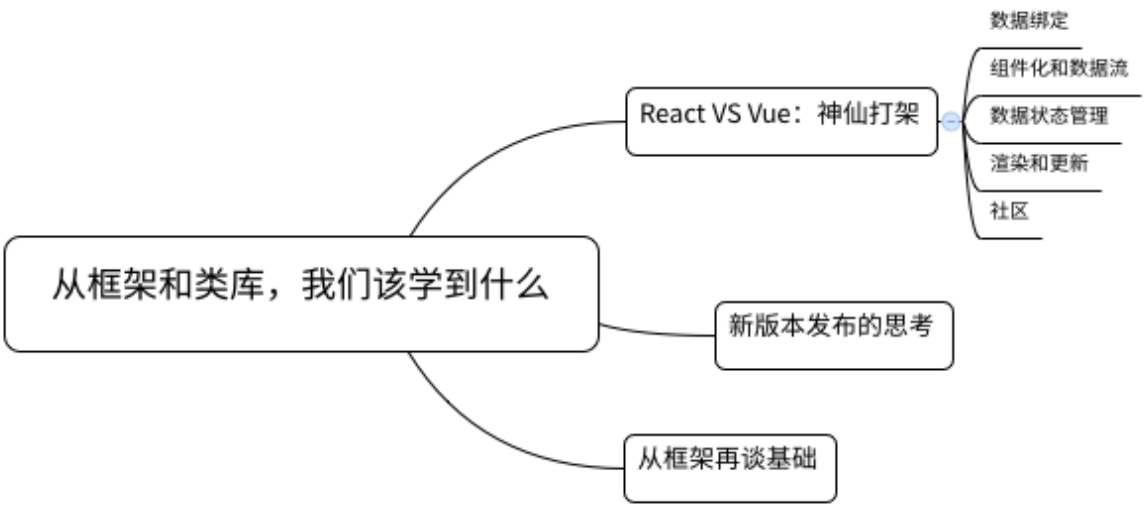
我记得作为实习生时，第一个独立前端项目是为欧洲天然气石油系统做一个计费应用：在地图上，用户可以和地图交互，从欧亚大陆任何一个石油气点为起点，到任何一个石油气点作为终点，计算出沿途所有的路径运输方案以及计费详情，还要支持各种终端的查询以及打印，这是一个纯 JavaScript 单页面应用 + Severless 的项目。

实习期满离职前我进行复盘：繁多的原生 JavaScript APIs 让我无语，兼容性让我崩溃，「如果我一开始就选用 jQuery 做该多好！」，too young too simple。这时候再让我复盘，「jQuery 也不适用那样一个单页应用，我为什么不用 React 或者 Vue？」果然 sometimes naive。

前端框架确实一直在演进、发展，那么框架除了简化我们的操作外，我们还能从中学到什么？这一讲，我就对这个开放性话题进行展开。本节课内容比较轻松，没有大量的代码案例，让我们来舒缓一下，但请读者在阅读时始终思考——「从框架和类库，我们该学到什么」这一问题。

Vue 和 React 加其生态方案使得 Vue 和 React 不再是单纯的视图层类库，因此下面我们将 Vue 和 React 统称「框架」，表示其全家桶。

相关知识点如下：



React VS Vue：神仙打架

React 和 Vue 这两个极其优秀的前端「框架」基本上占据了前端开发的半壁江山甚至更多。作为开发者，你可以首选 React 或者 Vue 其中一个，但是如若尝试将两者进行对比，那么一定会收获很多。

这里我们将从以下五大方面进行比较：

数据绑定

组件化和数据流

数据状态管理

渲染和更新

社区

数据绑定

Vue 在数据绑定上，采取了双向绑定策略，依靠 Object.defineProperty （Vue 3.0 已迁移到 Proxy）以及监听 DOM 事件实现。具体实现方法在先前课程中我们已经剖析过了，简单来说数据改变，依赖对数据进行拦截 / 代理；视图改变，依赖 DOM 事件（如 onInput、onChange 等）。Vue 实例中的 data 和 模版展现是一条线，无论谁被修改，另一方也会发生变动。

值得一提的是 Vue 也支持计算属性，即 computed 属性，这个策略和 React-redux 当中的 mapStateToProps 有异曲同工之妙，都是通过计算，将所需要的数据注入给组件使用。

需要区分清楚的是：双向绑定和单向数据流并没有直接关联，双向绑定是指数据和视图之间的绑定关系，而单向数据流是指组件之间数据的传递。

React 并没有数据和视图之间的双向绑定，它的策略是「局部刷新」。当数据发生变化时，直接重新渲染组件，以得到最新的视图。这种「无脑」刷新的做法看似粗暴，但是换来的简单直观，并且 React 本身在性能上也提供了一定保障。

组件化和数据流

Vue 中组件不像 React 组件，它不是完全以组件功能和 UI 为维度划分的，而 Vue 组件本质是一个 Vue 实例。每个 Vue 实例在创建时都需要经过：设置数据监听、编译模版、应用模版到 DOM，在更新时根据数据变化更新 DOM 的过程。在这个过程中，类似 React 也提供了生命周期方法。

Vue 组件间通信或者说组件间数据流如同 React，也是单向的。数据流向也很类似：props 实现父组件向下传递数据，events 实现子组件向上发送消息给父组件，React 中是基于 props 的回调实现子组件向父组件传递数据（Vue 也支持）。

当然，这两种框架也分别通过 context 和 provide/inject 实现了跨层级数据通信，它们的实现也是非常类似的。

数据状态管理

对于较为复杂的数据状态，Redux 是 React 应用最常用的解决方案。这里需要说明的是：Redux 和视图无关，它只是提供了数据管理的流程，因此 Vue 使用 Redux 也是完全没有问题的。

当然，Vue 中更常用的是 Vuex，其借鉴了 Redux（Flux），也具有和 Redux 相同的 store 概念，组件不允许直接修改 store state，而是需要 dispatch action 来通知 store 的变化。但是这个过程不同于 Redux 的函数式思想，Vuex 改变 store 的方法支持提交一个 mutation。mutation 类似于事件发布订阅系统：每

个 mutation 都有一个字符串来表示事件类型（type）和一个回调函数（handler）以进行对应的修改。

另一个显著区别是：在 Vuex 中，store 是被直接注入到组件实例中的，因此用起来更加方便。而 Redux 需要 connect 方法，把 props 和 dispatch 注入给组件。

造成这些不同的本质原因是：

Redux 提倡不可变性，而 Vuex 的数据是可变的，Redux 中 reducer 每次都会生成新的 state 以替代旧的 state，而 Vuex 是直接修改；

Redux 在检测数据变化的时候，是通过浅比较的方式比较差异的，而 Vuex 其实和 Vue 的原理一样，是通过遍历数据的 getter / setter 来比较。

渲染和更新

就像上面所提到的，React 和 Redux 倡导不可变性，更新需要维持不可变原则；而 Vue 对数据进行了拦截/代理，因此它不要求不可变性，而允许开发者修改数据，以引起响应式更新。

React 更像 MVC 或者 MVVM 模式中的 view 层，但是搭配 Redux 等，它也是一个完整的 MVVM 类库。Vue 直接是一个典型 MVVM 模式的体现，虽然它一直标榜自己也只是 View 层，但是毫无疑问它本身包含了对数据的操作。比如，Vue 文档中经常会使用 VM（ViewModel 简称），这个变量名表示 Vue 实例，其命名让人想到 MVVM，这是 MVVM 模式的体现。

React 所有组件的渲染都依靠灵活而强大的 JSX。JSX 并不是一种模版语言，而是 JavaScript 表达式和函数调用的语法糖。在编译之后，JSX 被转化为普通的 JavaScript 对象，用来表示虚拟 DOM。

Vue templates 是典型的模版，这相比于 JSX，表达更加自然。在底层实现上，Vue 模版被编译成 DOM 渲染函数，结合响应系统，进行数据依赖的收集。Vue 渲染的过程如下：

new Vue，进行实例化

挂载 $mount 方法，通过自定义 Render 方法、template、el 等生成 Render 函数，准备渲染内容

通过 Watcher 进行依赖收集

当数据发生变化时，Render 函数执行生成 VNode 对象

通过 patch 方法，对比新旧 VNode 对象，通过 DOM Diff 算法，添加、修改、删除真正的 DOM 元素

当然 Vue 也可以支持 JSX。

关于更新性能问题，简单来说，在 React 应用中，当某个组件的状态发生变化时，它会以该组件为根，重新渲染整个组件子树。当然我们可以使用 PureComponent，或是手动实现 shouldComponentUpdate 方法，来规避不必要的渲染。但是这个实现过程要知悉数据状态结构，也需要一定的额外负担。

在 Vue 应用中，组件的依赖是在渲染过程中自动追踪的，因此系统能精确知晓哪个组件需要被重渲染。从理论上看，Vue 的渲染更新机制更加细粒度，也更加精确。

社区

这两个框架都具有非常强大的社区，但是对于社区的理念，Vue 和 React 稍有不同。举个例子：路由系统的实现。

Vue 的路由库和状态管理库都是由官方维护的，并且与核心库是同步更新的。而 React 把这件事情交给了社区，比如 React 应用中，需要引入 react-router 库来实现路由系统。

新版本发布的思考

我一直认为开发者可以从框架新版本的迭代 changelog 里汲取非常多的养分。因为每次发版，都是经过开源框架的维护团队精心设计的，更新点或涉及 bug fix 或涉及新的 features，我们便可以思考「为什么会有这些变动」?，「为什么这样解决 bug」？

除此之外，我还建议开发者从更高的层次「开启上帝视角」，抓着某一话题，某一次变更进行研究、学习。这里我就举个例子：Vue3.0 带来的一些思考。

本讲在编写之时，恰逢尤雨溪在上海 Vue Conf 进行分享，演讲主题：State of Vue，其中涉及到新版本发布。从这次新版本发布对 Vue 的改动，结合我们的课程，我能找到很多切入点进行分析，比如虚拟 DOM、Proxy 实现数据代理。

其中尤雨溪表示 Vue3.0 相比以往版本更加快速。那么「更加迅速」是如何做到更快的呢？我们在《第 4-1 课：触类旁通多种框架》一课中对比过 Object.defineProperty 和 Proxy 实现的数据拦截和代理。那么在 Vue3.0，Vue 团队就是用 Proxy 来代替 Object.defineProperty，达到了更好的性能保障。

除此之外，Vue 新版本还重构了虚拟 DOM，正好在《第 4-1 课：触类旁通多种框架》一课中也简单实现了一个粗糙的虚拟 DOM，我们可以展开谈谈：传统虚拟 DOM 的性能瓶颈在于虚拟 DOM 的 diff，也许某次更新只有一个很小的变动，但还是需要对比整棵虚拟 DOM 树，这显然是非常不划算的。

Vue 新版本将虚拟 DOM 的节点分为动态节点和静态节点。静态节点是指不会发生改变的节点，这些节点在进行 diff 时是应该进行规避的，而我们只需要对比动态节点。

比如这样的内容：

最理想的情况是只需要对比可能会发生变化的 p 标签。再看这种情况：

最理想的情况是只需要对比 \

以及 \{{data.foo}}\，因为前者可能会根据判断条件消失 / 出现，后者直接取决于模版变量的值，都属于动态节点。

这样一来，我们便可以根据模版，将动态节点切割为区块，在进行 diff 操作时，递归进行区块中的动态节点比对即可。因此，**新的 diff 策略更新性能不再取决于模版整体节点数量的多少，而和动态内容的数量正相关。**

当然，这次 Vue 新版本的发布还有很多有意思的点，将其和 React 相比，和 React hooks 相比也非常有趣。这里不再展开。

从框架再谈基础

这一环节我们不需要详细展开，每一个开发者都应该认识到基础的重要性。从框架上来看，如果基础薄弱，你可能就不会明白为什么：

> 「React 事件处理函数还需要手动绑定 this」，「而 React 生命周期函数中却不需要手动绑定 this」
>
> 「为什么 Vue 可以实现双向绑定」等问题。

研究框架也不一定非要等到基础很扎实的时候。因为我们在学习框架之时，也是对自己基础查漏补缺的很好时机。

总结

这一讲比较轻松，我们重点分析了框架对比、框架发展。从中每一个开发者都应该能体会到「从框架和类库，我们该学到什么」。也许通过这种形式，我们不仅能够从更高层面理解框架，更能对框架的学习、今后的学习有所启发。

到此，框架大章节到此结束了，我们调整一下心情，进入下一大章节的学习。

点击查看下一节

深入浅出模块化（含 tree shaking）（上）