思考和扩展

React

1. Vdom
2. fiber
3. hooks

上面这些东西，能给我们什么启发 React VS Vue 设计的理念

1. Vdom 为什么有存在的必要 React和vue的vdom 区别

   0.1: innerHTML $(xx).html dom全量重写 性能灾难

   原则就是 少操作dom

   1. 响应式： vue1.0 精确追踪数据，变化后最小化操作dom 数据量打了之后，响应式数据过多，浏览器卡顿 option api { data(){},methods} 主动追踪
   2. React15- vdom 没有追踪的概念
      每次变化，生成新的数据 新老diff，计算出最小需要操作的dom 数据量大了之后，diff的过程是同步的，经常大于16ms，就会导致偶尔卡顿 class XX extend Component 被动计算 上面俩，都有瓶颈
      vue2: 响应式+vdom 这俩看起来是冲突的 以组件为单位，控制颗粒度 响应式走到组件级别，组件内部vdom option api React16 fiber+hooks 函数组件主推 函数组件有了内部状态，比class优秀很多 let [count ,setCount] = useXX() fiber重点理解 时间切片？ 给我们的启发

   现在 Vue3 整体的架构和vue2类似多了几个东西

   1. composition api （和hooks长的一样）
   2. Proxy取代defineProperty 利用浏览器新特性
   3. vdom的静态标记 静态标签 diff忽略 block 任何对dom结构产生影响的标签v-if v-for 内部都是一个block 无论嵌套多少层，动态节点都是在一个数组里维护，diff的时候不用递归 4 cusom renderer api

   React 17？ 期待

   横向的扩展

fiber 两个大件，1. vdom从树=》链表 2. 利用浏览器渲染的间隔时间 requestIdleCallback 1. 前端架构的更迭，本质上是数据结构和计算机基础逐渐深入的发展 以前的vdom{type：，children，props} 新的vdom {type，child，return，slibing， props} 2. 16.6ms 一帧 任何占用主进程超过这个时间，都可能会卡顿 这种任务，我们基本都可以用fiber这个理念来解决 md5的计算

block

性能优化 1. 减小计算量 2. 空间换时间 刷leetcode第一题 3. 缓存 4. fiber 时间切片

react父组件有更新，所有子孙组件默认都会执行render函数，这个不太理解为什么要这样 其实完全可以用pureComponent取代所有的component function 就用memo 为啥不这么干，而是提供了两种方式

React团队认为给你自由 Component手动挡 定制性更强

react设计将节点 时间复杂度O(n3)改成了O(n)其实没太理解 1。 考虑到web的场景 web大概率都是不会发生结构性的变化 很少又一个完整的树变成别的子节点 基本都是平级修改，删除，替换 两个树完整对比

vite 新一代的开发工具，利用浏览器自带的import

1. node_modules
2. css less sas