# 前端框架原理浅析（React）

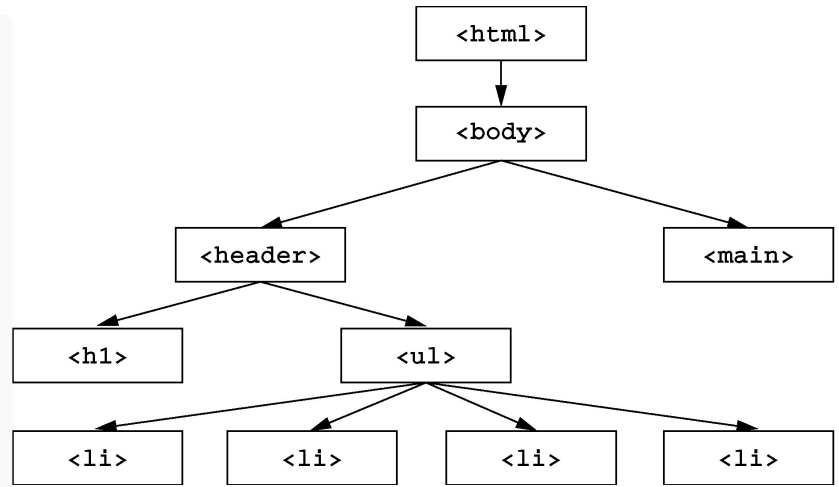
# 前置知识

1. DOM是一棵树
2. 浏览器单线程且操作dom很耗时

```
<html>

<body>
    <header>
        <h1></h1>
        <ul>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
        </ul>
    </header>
    <main></main>
</body>

</html>
```



# 传统的开发方式

## 直接操作dom

接口响应操作dom，用户行为操作dom，整个dom，东一榔头，西一棒槌的

整个页面显示逻辑不是很清晰，且操作比较原始，效率低下

```
// 获取dom再做操作
document.querySelector('.some-class').onClick = () => {
  document.querySelector('.other-class').style.color = 'red';
}
// jquery
$('.some-class').text('text');
```

## 存在的问题

- 代码结构复杂难以维护
- 代码难以复用
- 开发效率低下

# React怎么解决这些问题

期望数据和dom有一个映射关系，这样只要数据有变化，dom自动的进行变化，不用再手动操作dom

## 处理方案

声明式 和 组件化

```
1   const textComponent  = (text) => {
2     const [count, setCount] = useState(text);
3     return  <div onClick={() => setCount(count++)}>{count}</div>
4   }
```

## 简单理解下

就是你开发者只需要声明数据和dom的关系，然后具体的渲染页面什么的我React替你办了

## 对比理解下

之前开发，有数据变化，就要去找对应的dom，然后去做对应的改变

现在呢，开发者只需要提前定义好数据和dom的映射关系，完了修改数据，页面自动刷新，开发者只关心数据

# React做了什么

1. 接管了整个dom层，简单来说React相当于一个状态机，负责根据状态渲染页面

2. 当状态发生变化时，React会自动更新页面，这样开发者只需要关心状态和页面布局即可

## 渲染

页面初始化，将提前声明好的模板渲染成dom，这个没什么问题

## 主要是更新 怎么更新？

直接把整个页面清空了，再走一遍页面初始化, 在浏览器整个单线程，且操作dom很费时间

那就把整个模板和页面上的dom进行对比，看哪个需要更新，单独给处理下，操作dom很费时间

# 虚拟dom

React引入了虚拟dom来处理这个问题

## 什么是虚拟dom

是由普通的 JS 对象来描述DOM对象，因为不是真实的DOM对象，所以叫虚拟dom(Virtual DOM)。

既然操作dom慢，那就操作普通的js对象，这个比操作dom快

```
{
  "type": "div",
  "props": {
    "children": [{
        "type": "h1",
        "style": "color: red"
    }]
  }
}
```

# React的渲染和更新逻辑

1. 初始化渲染时，先将模板翻译成虚拟dom做一份缓存，然后用虚拟dom渲染页面
2. 当有状态更新时（dom需要更新），将新的虚拟dom和缓存的虚拟dom进行对比，然后在页面中只更新需要更新的dom

## 新的问题

DOM 是树形结构，目前已知的完整树形结构对比算法复杂度为 O(n^3)，时间复杂度 O(n^3) 太高了。

## 怎么解决

所以Facebook工程师考虑到dom的特殊情况，做了大胆的假设和取舍，然后将复杂度降低到了 O(n)。

# 客观存在的问题

虽然复杂度降低了，但是每次对比的时间还是客观存在的，超级复杂的页面交互中，会存在页面卡顿的情况。

费了那么大劲，又接管了dom，又大胆假设和取舍搞算法研究，现在竟然嫌弃慢？

facebook 团队使用两年多的时间去重构 React 的核心算法，在React16以后引入了 Fiber 架构，也叫时间分片。

简单理解一下，React每次执行的时候看浏览器闲着没，闲着的话就执行对比计算等一些列操作，只要浏览器一忙，立马暂停，等！

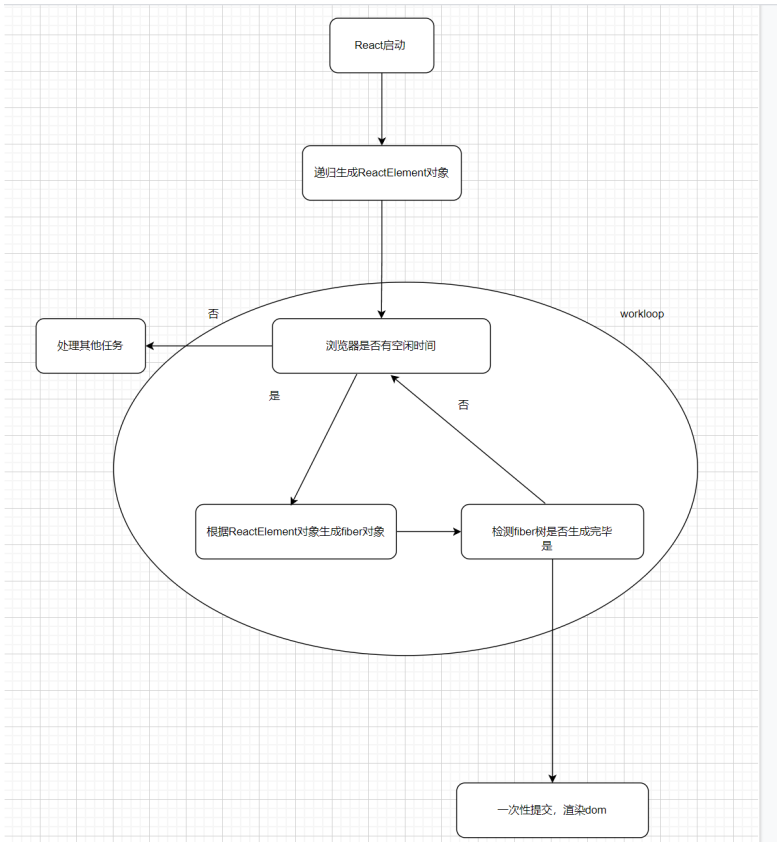
这样的话之前的虚拟dom的数据结构就无法满足要求了，之前的嵌套结构只能一次递归处理，没办法满足要求，现在需要一个方案来处理这种可以被随时停止也能随时恢复的操作，这个就是Fiber架构。

# 理解下Fiber架构

主要目的是为了将渲染工作进行分解，分解成一个个小单元，这样浏览器每次有空的时候执行下当前的小单元，执行完后指向下一个小单元，继续等浏览器的空闲时间，继续执行，直到所有的小单元执行完毕，一次刷新页面。

1. 拿到虚拟dom后，将虚拟dom处理成fiber树 这个在React中称为调和
2. 调和过程可中断，你浏览器但凡有点其他事情，您就忙去吧，闲了就过来给我继续调和，直到fiber树完成

# Fiber架构流程图


React启动
递归生成ReactElement对象
否
workloop
处理其他任务
浏览器是否有空闲时间
是
否
根据ReactElement对象生成fiber对象
检测fiber树是否生成完毕
是
一次性提交，渲染dom

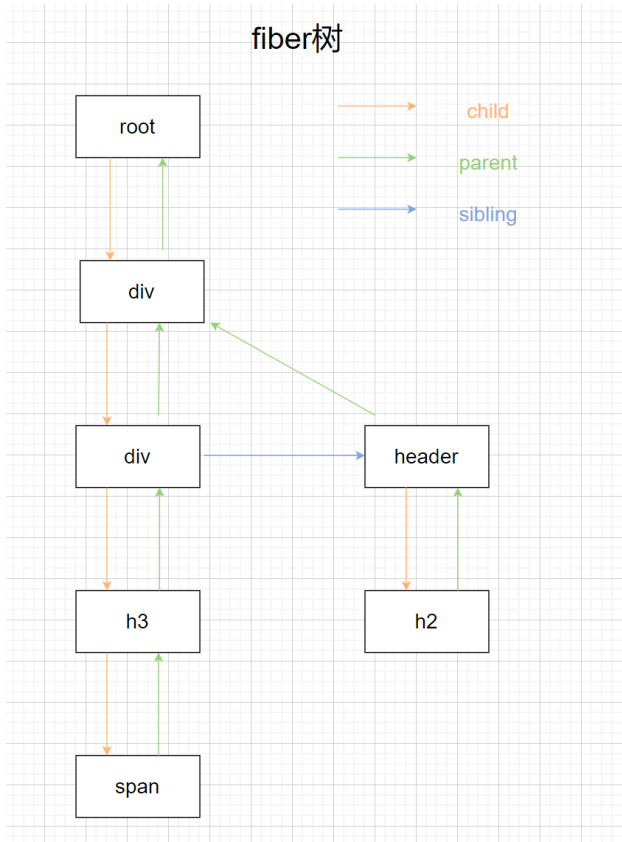
# Fiber树深入理解

1. `fiber` 树的目标是非常容易找到下一个单元工作，这也是为什么每一个 `fiber` 节点都有指向第一个节点和相邻节点以及父节点的链接。当我们完成在 `fiber` 上面的工作后，`fiber` 拥有 `child` 属性可以直接指向下一个需要进行工作的 `fiber` 节点。

2. 当 `fiber` 节点没有 `child` 也没兄弟节点时，我们去他们的*叔叔*（父节点的兄弟节点）节点，如果 `fiber` 的父节点也没有兄弟节点，我们继续往上找父节点的兄弟节点直到到根节点。当我们到根节点的时候，也意味着在这一次 `render` 我们完成了所有的工作。

深度优先遍历

# Fiber树遍历流程图



# 有了Fiber之后的渲染流程

1. 在浏览器空闲的时候，在内存中按照小的单元一份一份的生成dom，挂在fiber树上，这个过程可中断，完了后一次性更新到页面上，并将本次的fiber树缓存。
2. 更新的时候边生成新的fiber树，边和老的fiber树进行对比，fiber树处理完后直接渲染, 这个动作在react中叫提交。

这种技术有个学名叫双缓存技术，简单来说 就是 内存中绘制当前帧动画，绘制完毕后直接用当前帧替换上一帧画面，由于省去了两帧替换间的计算时间，不会出现从白屏到出现画面的闪烁情况

# 最后一个问题

React快吗？又是虚拟dom，又是Fiber架构，还整了双缓存技术


**尤雨溪**

前端开发、JavaScript、前端工程师 话题的优秀回答者

1,666 人赞同了该回答

这里面有好几个方面的问题。

**1. 原生 DOM 操作 vs. 通过框架封装操作。**

这是一个性能 vs. 可维护性的取舍。框架的意义在于为你掩盖底层的 DOM 操作，让你用更声明式的方式来描述你的目的，从而让你的代码更容易维护。没有任何框架可以比纯手动的优化 DOM 操作更快，因为框架的 DOM 操作层需要应对任何上层 API 可能产生的操作，它的实现必须是普适的。针对任何一个 benchmark，我都可以写出比任何框架更快的手动优化，但是那有什么意义呢？在构建一个实际应用的时候，你难道为每一个地方都去做手动优化吗？出于可维护性的考虑，这显然不可能。框架给你的保证是，你在不需要手动优化的情况下，我依然可以给你提供过得去的性能。

@稀土掘金技术社区