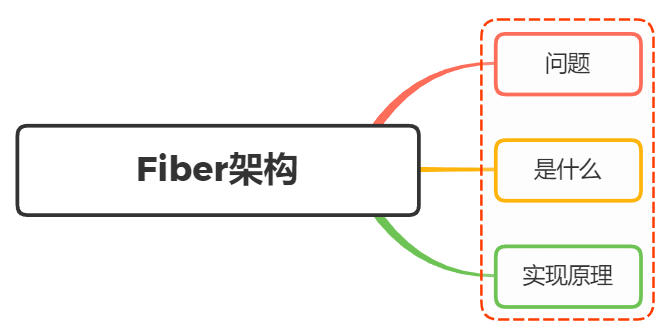
# 面试官：说说对Fiber架构的理解？解决了什么问题？



## 一、问题

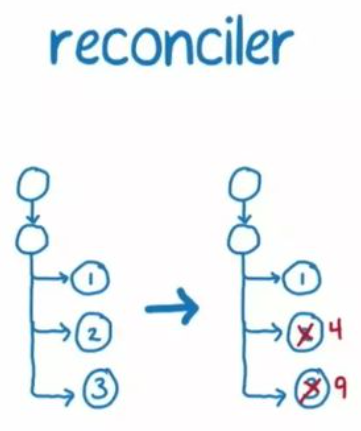
JavaScript引擎和页面渲染引擎两个线程是互斥的，当其中一个线程执行时，另一个线程只能挂起等待

如果 JavaScript 线程长时间地占用了主线程，那么渲染层面的更新就不得不长时间地等待，界面长时间不更新，会导致页面响应度变差，用户可能会感觉到卡顿

而这也正是 React 15 的 Stack Reconciler所面临的问题，当 React在渲染组件时，从开始到渲染完成整个过程是一气呵成的，无法中断

如果组件较大，那么js线程会一直执行，然后等到整棵VDOM树计算完成后，才会交给渲染的线程

这就会导致一些用户交互、动画等任务无法立即得到处理，导致卡顿的情况



## 二、是什么

React Fiber 是 Facebook 花费两年余时间对 React 做出的一个重大改变与优化，是对 React 核心算法的一次重新实现。从Facebook在 React Conf 2017 会议上确认，React Fiber 在React 16 版本发布

在react中，主要做了以下的操作：

* 为每个增加了优先级，优先级高的任务可以中断低优先级的任务。然后再重新，注意是重新执行优先级低的任务
* 增加了异步任务，调用requestIdleCallback api，浏览器空闲的时候执行
* dom diff树变成了链表，一个dom对应两个fiber（一个链表），对应两个队列，这都是为找到被中断的任务，重新执行

从架构角度来看，Fiber 是对 React核心算法（即调和过程）的重写

从编码角度来看，Fiber是 React内部所定义的一种数据结构，它是 Fiber树结构的节点单位，也就是 React 16 新架构下的虚拟DOM

一个 fiber就是一个 JavaScript对象，包含了元素的信息、该元素的更新操作队列、类型，其数据结构如下：

type Fiber = {  
 // 用于标记fiber的WorkTag类型，主要表示当前fiber代表的组件类型如FunctionComponent、ClassComponent等  
 tag: WorkTag,  
 // ReactElement里面的key  
 key: null | string,  
 // ReactElement.type，调用`createElement`的第一个参数  
 elementType: any,  
 // The resolved function/class/ associated with this fiber.  
 // 表示当前代表的节点类型  
 type: any,  
 // 表示当前FiberNode对应的element组件实例  
 stateNode: any,  
  
 // 指向他在Fiber节点树中的`parent`，用来在处理完这个节点之后向上返回  
 return: Fiber | null,  
 // 指向自己的第一个子节点  
 child: Fiber | null,  
 // 指向自己的兄弟结构，兄弟节点的return指向同一个父节点  
 sibling: Fiber | null,  
 index: number,  
  
 ref: null | (((handle: mixed) => void) & { \_stringRef: ?string }) | RefObject,  
  
 // 当前处理过程中的组件props对象  
 pendingProps: any,  
 // 上一次渲染完成之后的props  
 memoizedProps: any,  
  
 // 该Fiber对应的组件产生的Update会存放在这个队列里面  
 updateQueue: UpdateQueue<any> | null,  
  
 // 上一次渲染的时候的state  
 memoizedState: any,  
  
 // 一个列表，存放这个Fiber依赖的context  
 firstContextDependency: ContextDependency<mixed> | null,  
  
 mode: TypeOfMode,  
  
 // Effect  
 // 用来记录Side Effect  
 effectTag: SideEffectTag,  
  
 // 单链表用来快速查找下一个side effect  
 nextEffect: Fiber | null,  
  
 // 子树中第一个side effect  
 firstEffect: Fiber | null,  
 // 子树中最后一个side effect  
 lastEffect: Fiber | null,  
  
 // 代表任务在未来的哪个时间点应该被完成，之后版本改名为 lanes  
 expirationTime: ExpirationTime,  
  
 // 快速确定子树中是否有不在等待的变化  
 childExpirationTime: ExpirationTime,  
  
 // fiber的版本池，即记录fiber更新过程，便于恢复  
 alternate: Fiber | null,  
}

## 三、如何解决

Fiber把渲染更新过程拆分成多个子任务，每次只做一小部分，做完看是否还有剩余时间，如果有继续下一个任务；如果没有，挂起当前任务，将时间控制权交给主线程，等主线程不忙的时候在继续执行

即可以中断与恢复，恢复后也可以复用之前的中间状态，并给不同的任务赋予不同的优先级，其中每个任务更新单元为 React Element 对应的 Fiber节点

实现的上述方式的是requestIdleCallback方法

window.requestIdleCallback()方法将在浏览器的空闲时段内调用的函数排队。这使开发者能够在主事件循环上执行后台和低优先级工作，而不会影响延迟关键事件，如动画和输入响应

首先 React 中任务切割为多个步骤，分批完成。在完成一部分任务之后，将控制权交回给浏览器，让浏览器有时间再进行页面的渲染。等浏览器忙完之后有剩余时间，再继续之前 React 未完成的任务，是一种合作式调度。

该实现过程是基于 Fiber节点实现，作为静态的数据结构来说，每个 Fiber 节点对应一个 React element，保存了该组件的类型（函数组件/类组件/原生组件等等）、对应的 DOM 节点等信息。

作为动态的工作单元来说，每个 Fiber 节点保存了本次更新中该组件改变的状态、要执行的工作。

每个 Fiber 节点有个对应的 React element，多个 Fiber节点根据如下三个属性构建一颗树：

// 指向父级Fiber节点  
this.return = null  
// 指向子Fiber节点  
this.child = null  
// 指向右边第一个兄弟Fiber节点  
this.sibling = null

通过这些属性就能找到下一个执行目标

## 参考文献

* https://juejin.cn/post/6926432527980691470
* https://zhuanlan.zhihu.com/p/137234573
* https://vue3js.cn/interview