35 React16题眼：理解Fiber思想

更新时间：2020-06-01 10:09:58

> 梦想只要能持久，就能成为现实。我们不就是生活在梦想中的吗？——丁尼生

React16 值得探讨的新特性有很多，这一点我完全认同。不过如果你时间有限，那么在你珍贵的备考时间里，我需要你非常着重掌握的就是大部分面试官最关心的、同时也是 React16 最大的一个改动——Fiber 架构。

Fiber 是个什么东西？

> Fiber is the new reconciliation engine in React 16. Its main goal is to enable incremental rendering of the virtual DOM

Fiber 是 React16 引入的一种**新的调和引擎**。

经过前面两节的学习，大家不难感觉出来，调和过程对应着 React 最核心的一套算法。“新的调和算法”，意味着 React 把自己最核心的东西给重写了。到底是什么原因，驱使 React 团队这么大费周章做这样一件事呢？

如果你知道 Fiber 这个英文单词的含义，事情会好理解很多：

“fiber”的翻译

名词

纤维
fiber, fibre

“fiber"的中文翻译是“纤维”。在现实世界中，纤维是一种比线还要纤细的东西。对应到计算机领域中，**fiber 就是比线程更为细微的流程控制机制：**

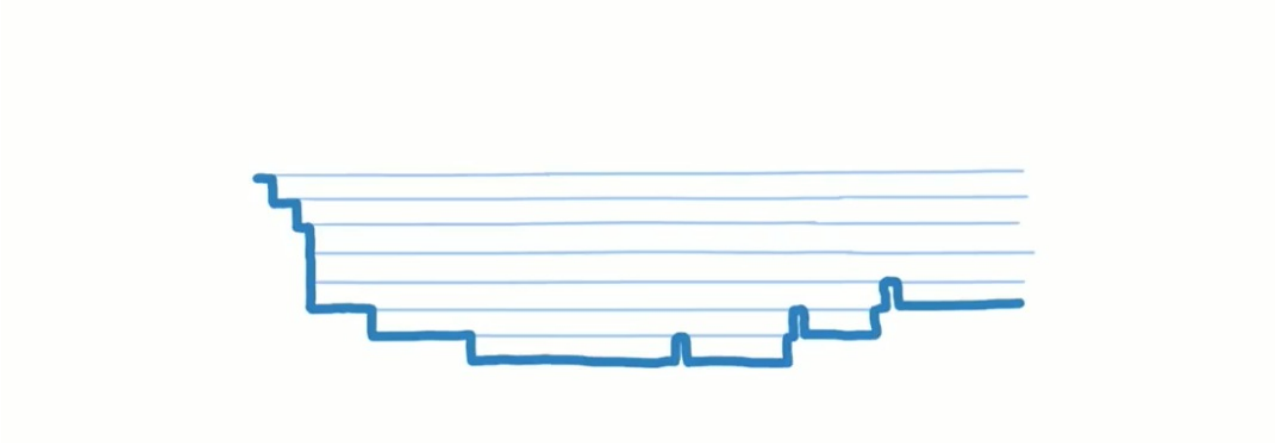
> a fiber is a particularly lightweight thread of execution。

React Fiber顾名思义，便是希望对整个渲染流程进行更加精细的控制。

Fiber 解决了什么问题

在 Fiber 架构前，当 React 决定要加载或者更新组件树时，会做一个“大动作”。

这个动作包括生命周期的调用、diff过程的计算、DOM 树的更新等等。这个动作很大，耗时不短，却竟然是**同步**进行的，**一旦开始就不能中断**。这意味着你在挂载/更新结束前，啥也不能干。



我们来看这张图。起点意味着调和开始了，React 会从根节点开始，深度遍历这棵树。

如果遇到了 React 子组件更新的情况，则会去调用这个子组件的更新流程：

```
componentWillReceiveProps->shouldComponentUpdate->render.....
```

注意在 render 这一步，又进入了子组件的调和过程，即重复上面的步骤。

就这样子子孙孙无穷尽也，直到图中所示的最低点最低点——叶子节点对应的子组件完成了自己的调和过程、执行到了 componentDidUpdate，调用才会一点一点地再返回到上一级、最后一直返回到根节点去。

像这样层层递归的调和方式，我们叫它 **Stack Reconciler**。

这个过程中有两个非常明显的问题：

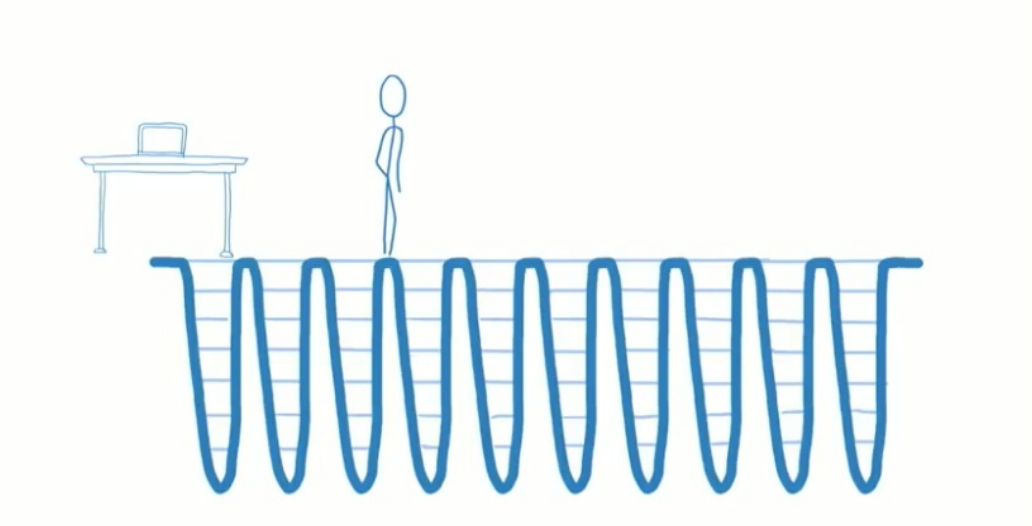
它是**一个同步的动作**。假如说我有一个组件，它比较大，包含了很多个子组件，这导致它的渲染过程可能会很长（比如四五百毫秒这样）。但是因为这个渲染过程是同步的，在这个过程里如果用户想进行其它的操作，是得不到任何交互的。那么这四五百毫秒从效果上来说就是阻塞了。

渲染动作没有优先级概念。假如一瞬间冲进来七八个组件都嚷嚷着要渲染，这七八个组件并没有优先级可言，React逮着谁渲染谁，最后可能会导致不符合预期的结果出现。

如果你平时没有被这样的问题折磨过，多半意味着你还没有接触到复杂度较高的 React 应用，并不代表这种问题不存在。事实上，随着 React 应用复杂度的提高，Facebook 团队也感知到了这“一往无前”的操作带来的体验问题，Fiber 就是为了解决这个问题而生的——它可以帮我们实现异步渲染。

Fiber 思想

面对“单个任务耗时过长”这个问题，工程师们的思路是，把一个庞大的任务成 N 多个微小的任务（如下图）**拆分**成 N 多个微小的任务（如下图）



这个拆分的结果，就叫 Fiber。它代表着一个单位的工作，也是接受调度的最小单元。

图中我们每一个波峰和波峰之间，就意味着是一个工作单元、一个 Fiber。每次到达波峰时，意味着该任务交出了对主线程的占用。我们可以看出，每完成一个小任务，都会**暂停一下**对主线程的占用。这“暂停一下”意义重大，React 会在这段时间里探出头去，看看外面有没有优先级更高的事情需要处理。以此来确保主线程总在做它当下最应该做的事情。

这种新的调和方式，叫做 **Fiber Reconciler**。

Fiber 与生命周期

首先大家需要知道的是，不管是新 React 还是 旧 React，它们在 re-render 时都会有以下两个工作阶段：

render/reconcilition：调和阶段——“找不同”的过程。这个过程里，React在内存中做计算，确认所有的更新行为。

这个过程涉及的生命周期有：

1. [UNSAFE_]componentWillMount (已废弃)；

2. [UNSAFE_]componentWillReceiveProps (已废弃)；
3. getDerivedStateFromProps；
4. shouldComponentUpdate；
5. [UNSAFE_]componentWillUpdate (已废弃)；
6. render。

commit： 执行调和阶段的计算结果，真正地去更新 DOM ，这个过程涉及的生命周期有：

1. getSnapshotBeforeUpdate；
2. componentDidMount；
3. componentDidUpdate；
4. componentWillUnmount。

调和阶段的生命周期，在 Fiber 架构引入之前都是不允许被打断的，就像咱们前面说的一样，它“一往无前”。

但是引入 Fiber 后，由于“切片”和“暂停”两个关键特性的实现，调和阶段变成了一个可以被打断的过程。

注意“打断”并不意味着放弃。等主线程处理完优先级更高的事情之后，会掉头再来执行一遍这个之前被打断的任务。也就是说调和阶段的生命周期，是可能被重复执行的。

由于调和阶段的生命周期逻辑是单纯的 js 计算，所有的工作都在内存里进行，**不涉及真实 DOM 操作**，也就是说你打断执行也好、重复执行也罢，用户都是不感知的，只要你最后能 commit 出正确的 DOM 更新就好。这里硬要说的话，有一个编码层面的坑需要注意一下：在调和阶段的生命周期里，不要尝试写入一些你期望它只执行一次的逻辑，它可不保证到底会给你执行多少次。

拓展：更深入的 Fiber

本节用到两张图，出自 Lin Clark 在 React Conf 2017 上一场名为《A Cartoon Intro to Fiber》的演讲。非常推荐大家都去看一下这个演讲，时间不长，半个小时。如果你对 Fiber 感兴趣，相信你会收获满满。

B站传送门：https://www.bilibili.com/video/av40427580?from=search&seid=4572863336877890139

另：听力不好的同学推荐去油管搜索同名视频，打开字幕即可。这里就不放链接了，和谐为贵哈。

}

34 真正理解虚拟DOM（二）——Diff算法 & 面试考点解析

36 Vue核心——响应式原理源码级解析