12 关于 React Fiber不得不说的事

更新时间：2020-08-21 10:07:56

> 每个人都是自己命运的主宰。——斯蒂尔斯

为什么叫 Fiber 呢？

在计算机科学中，纤程（英语：Fiber）是一种最轻量化的线程（lightweight threads），它是一种用户态线程（user thread），让应用程序可以独立决定自己的线程要如何运作。操作系统内核不能看见它，也不会为它进行调度。

就像一般的线程，纤程有自己的定址空间。但是纤程采取合作式多任务（Cooperative multitasking），而线程采取先占式多任务（Pre-emptive multitasking）。应用程序可以在一个线程环境中创建多个纤程，然后手动运行它。纤程不会被自动运行，必须要由应用程序自己指定让它运行，或换到下一个纤程。跟线程相比，纤程较不需要操作系统的支持。

> 注：关于纤程，大家了解即可。

React 将新的架构取名为 Fiber 是要表达新的架构会将（更像）任务拆分为最小的单元进行执行。

为什么要对 React v15 版本做重构

下面两个链接分别指向了使用 React v15 版本写的 demo1 和使用 React v16 版本写的 demo2。

点击查看demo1

点击查看 demo2

对比两个 demo 的效果后可以明显发现，如果应用程序在每次更新非常多 DOM 元素时，demo1 有明显的卡顿现象，demo2 却表现的非常流畅。这也说明了使用 React v16 版本开发的应用程序相对于 React v15 版本在用户体验上面有了很好的改善。

为了解决 React v15 版本在复杂交互场景下一次 setState 需要同时更新非常多 DOM 元素造成页面卡顿的问题（如用户的输入不能即时响应，动画不连续以及页面拖动迟缓），React 团队在 v16 版本对核心算法做了重构，研发了 **React Fiber **架构。

造成页面卡顿的原因

首先我们需要了解的是 FPS — 每秒传输帧数（Frames Per Second）决定页面流畅度，最优的帧率是 60（ 1 秒 60 帧），即 16.5ms 左右渲染一次。为了保证浏览器页面的流畅度，理想的情况下每次程序执行的时间最好控制在 16.5ms 左右。JavaScript 在浏览器的主线程上运行，与样式计算、布局以及许多其他工作一起执行。如果 JavaScript 运行时间过长，就会阻塞其他工作，进而会导致页面掉帧出现卡顿现象。

> 浏览器是多线程的，它包括 **GUI（渲染）线程，JS引擎线程，事件触发线程，定时触发线程** 和 **异步网络请求线程**。其中**GUI（渲染）线程，JS 引擎线程** 分别用于浏览器页面的渲染和 JS 的执行。如果 JS 引擎线程执行时间过长（超过 16.5ms），长期占用CPU，就到导致 GUI（渲染）线程无法及时工作，就会使页面更新产生视觉上的时间差，也就是卡顿现象。

React Fiber 之前架构卡顿的原因

有的动画执行需要浏览器对页面进行回流或者重绘，比如上文中 demo 中的动画就需要浏览器进行回流。有些动画如颜色或者透明度的变化则需要重绘。无论是回流还是重绘都需要渲染引擎去执行。如果 JS 引擎长期占有 CPU 资源导致渲染引擎一直无法工作就会引起页面卡顿现象的产生。

React v15 版本应用程序调用 `setState()` 和 `render()` 方法进行更新和渲染时主要包含两个阶段：

- **调度阶段（reconciler）**：React Fiber 之前的 reconciler（被称为 Stack reconciler）是自顶向下的递归算法，遍历新数据生成新的Virtual DOM。通过 diff 算法，找出需要更新的元素，放到更新队列中去。
- **渲染阶段（render）**： 根据所在的渲染环境，遍历更新队列，调用渲染宿主环境的 API, 将对应元素更新渲染。在浏览器中，就是更新对应的 DOM 元素，除浏览器外，渲染环境还可以是 Native、WebGL 等等。

React Fiber 之前的调度策略 Stack Reconciler，这个策略像函数调用栈一样，递归遍历所有的 Virtual DOM 结点进行 diff，**一旦开始无法被中断**，要等整棵 Virtual DOM 树计算完成之后，才将任务出栈释放主线程。而浏览器中的渲染引擎是单线程的，除了网络操作，几乎所有的操作都在这个单线程中执行，此时如果主线程上的用户交互、动画等周期性任务无法立即得到处理，就会影响体验。

> 注：React v15 版本中使用的架构，有兴趣的同学可自行详细了解。

React Fiber 架构如何解决上面问题

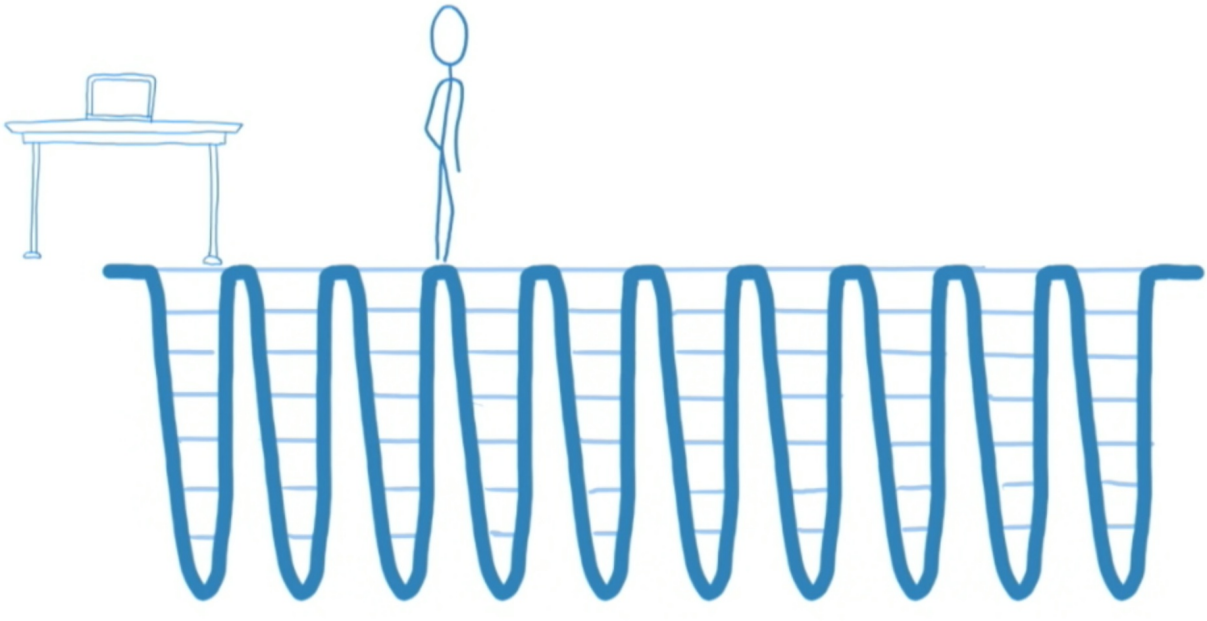

图 3.2.1 React Fiber 任务拆分机制

图 3.2.1 的描述的是 React 在执行整个任务时，会将整个任务拆分成若干子任务来执行，每次执行自认为都要检查是否拥有执行权，具体的逻辑如下：

1. 将任务按照单个 Fiber 结点为单位，拆分成细小的单元。
2. 重写 render 和 commit 两阶段的逻辑， render 阶段的任务每次在执行前先请求任务体系获得执行权。
3. 对任务建立优先级体系，高优先级任务优于低优先级工作执行。

小结

本节主要介绍的是 React 团队在研发 Fiber 架构时的背景，以及 Fiber 架构解决问题的思路，整体内容偏理论。在后续文章中会陆续介绍基于 React Fiber 架构的应用程序的内部详细执行流程，下一节会从宏观层面详细介绍基于 React Fiber 架构的应用程序整体渲染流程。

}

11 什么是 React Fiber

13 了解基于 React Fiber 架构的应用程序整体渲染流程