02 研究 React 应用程序的内部运行机制要找到切入点

更新时间：2020-08-12 11:11:03

> 读书而不思考，等于吃饭而不消化。——波尔克

前言

做为一名 React 开发者，你是否思考过为什么我们在开发 React 程序时并未直接（使用 JavaScript API）操作 DOM 却能更新 DOM，以及当组件中的 this.setState(...) 操作被触发后 React 内部到底做了哪些工作？本节将会介绍研究 React 内部运行机制的切入点以及整体研究思路。

如何研究 React 内部运行机制

我们想要知道 React 的内部运行机制，实际上就是要探索 **React 如何将组件映射屏幕，以及组件中的 state 和 prop 发生了变化之后 React 如何将这些「变化」更新到屏幕**。

想要搞清楚上面的两个问题，我们需要对 React 应用程序的首次渲染过程和更新渲染过程有整体的认知。那么，应用程序的首次渲染和更新渲染两个过程中 React 内部要做的工作有什么相同点与不同点呢？

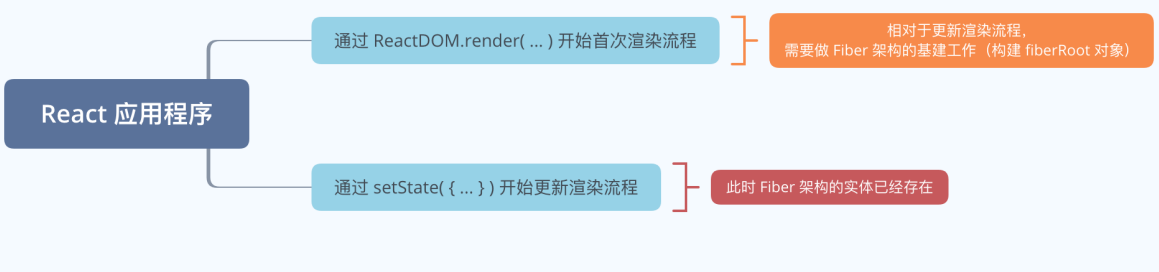

图 1.1.1 React 应用程序的两种渲染

应用程序首次渲染时 React 会做一些基建工作，比如 **将组件转化为元素，构建更新队列，构建 workInProgress 对象树以及构建 Effect list（副作用列表）** 等。这些基建工作是构建 React Fiber 架构的关键环节。而在应用程序更新渲染时，React Fiber 架构的实体—fiberRoot 对象已经存在于内存中，此时，React 更加关心的是 **计算出 Fiber 架构中各个结点的前后变化，并将【变化部分】更新到屏幕**。

现在，我们对上面图中的两个渲染过程中的关键环节用流程图表示，见图 1.1.2。

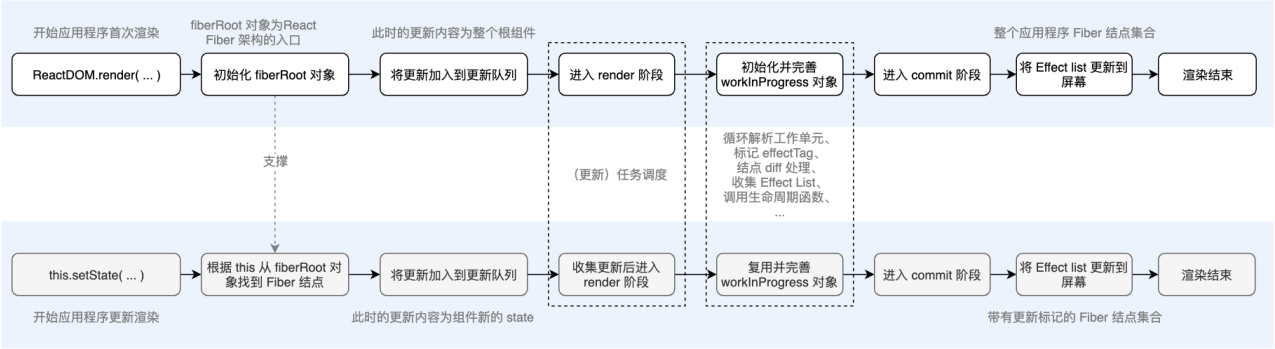

图 1.1.2 React 应用程序的两种渲染流程

React 应用程序首次渲染时的关键环节解析

1. 构建 fiberRoot 对象（FiberRootNode 构造函数的实例），fiberRoot 对象是整个 Fiber 架构的根结点对象。
2. 将更新加入到更新队列，此时的更新内容为应用程序的根组件。
3. 应用程序进入 render 阶段，在该阶段 React 的主要工作是构建 workInProgress 树（一颗 Fiber 树）。
4. 构建 workInProgress 树的过程中会做一些重要的工作，如为结点标记 effectTag，对结点进行 diff 处理，收集 Effect List（副作用列表），调用生命周期函数等。
5. 当收集好 Effect List 后则进入 commit 阶段，在该阶段 React 主要工作就是将 Effect List 更新到屏幕，然后渲染结束。

React 应用程序更新渲染时的关键环节解析

1. 相对于应用程序的首次渲染，更新渲染流程的主要区别有，不再重新构建 fiberRoot 对象，因为该对象已经存在于内存中。
2. 此时的更新内容一般为组件内部发生变化的 state 和 props 。
3. 在进入 render 阶段前要进行任务调度，申请过的更新执行权后才能进行后续渲染工作。
4. 此时构建 workInProgress 树时也会尽可能的复用上一次创建的 Fiber 结点，同时对需要更新的结点标记对应的 effectTag。
5. 在 commit 阶段得到的 Effect List 是被标记了 effectTag 的 Fiber 结点集合（一个链表），其一般是 workInProgress 树的子集。

> 注：上面提到的 React 在渲染过程中的工作多是抽象内容，后面我们会对其中每一项工作通过获取真实运行状态进行详细分析。

研究 React 内部运行机制 — 思维导图

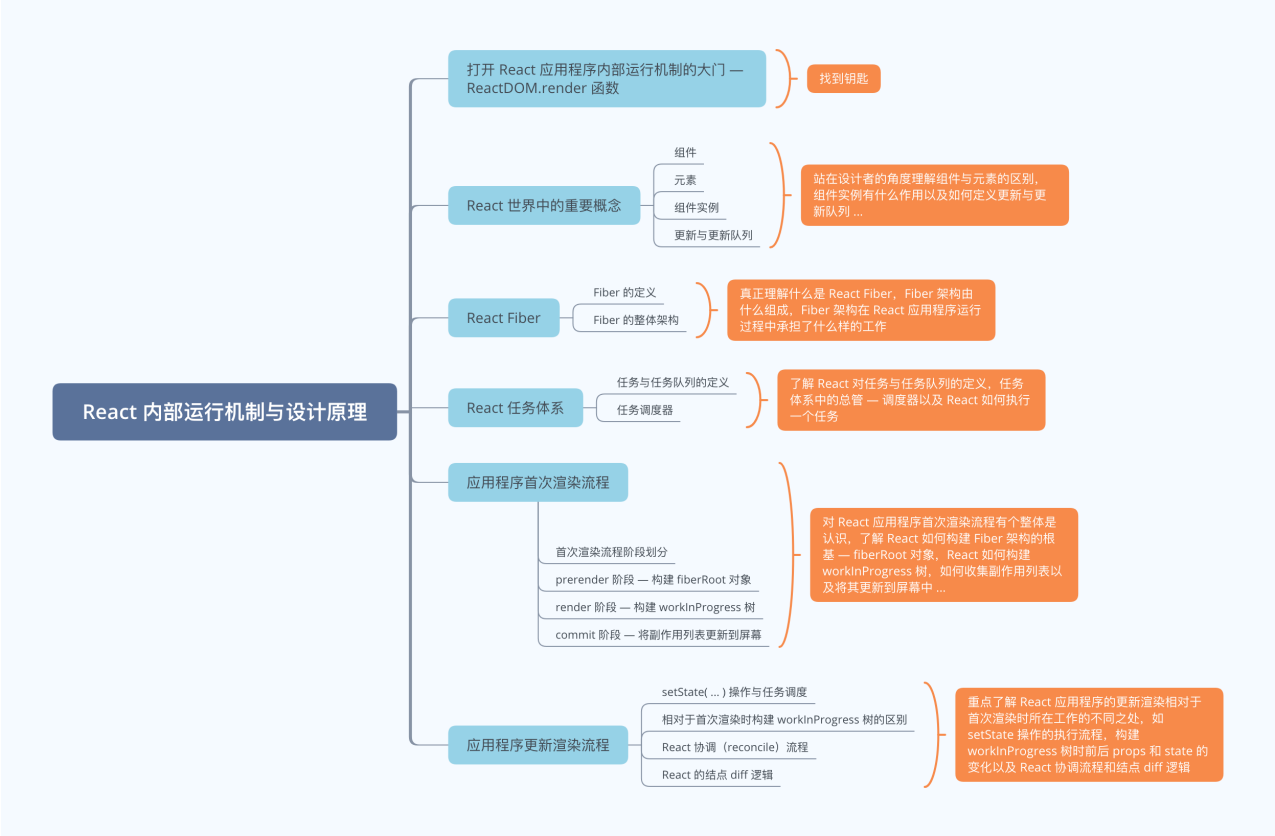

图 1.1.3 专栏学习思维导图

小结

本文提到的有关 React Fiber 架构的相关内容以及构建 workInProgress 树、Effect List 等内容在后面章节中会有详细介绍。现在我们知道，**要想打开 React 应用程序内部运行机制的大门，解锁 ReactDOM.render 函数是关键，因为 ReactDOM.render 函数是 React 应用程序首次渲染时的入口函数**，我们需要对它有更多的了解，ReactDOM.render 函数的更多介绍见本章第二节。

}

01 开篇词：为什么要研究 React 内部运行机制与设计原理

03 你对 ReactDOM.render(...) 操作了解多少