21 React 应用程序首次渲染时 prerender 阶段的工作

更新时间：2020-09-14 14:39:41

> 天才就是长期劳动的结果。——牛顿

前言

前面提到过，我们将 React Fiber 架构的应用程序运行过程分为 **prerender**，**render** 和 **commit** 三个阶段。其中只有在应用程序首次渲染时才会经历 **prerender** 阶段。React 应用程序的首次渲染和更新渲染内部运行所经历的阶段见下图 5.2.1。

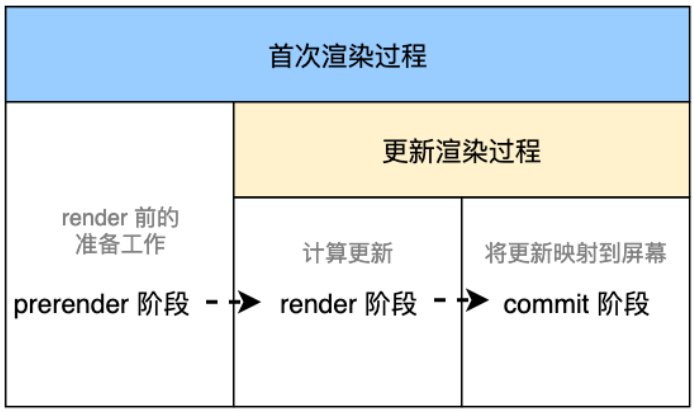

图 5.2.1 React应用程序首次渲染和更新渲染阶段划分

React 应用程序首次渲染时，在 prerender 阶段主要做一些 render 前的准备工作，在 render 阶段做最重要的更新计算，然后在 commit 阶段将上一阶段计算得到的更新内容映射到屏幕。本节将主要介绍 React 在 prerender 阶段做的具体工作。

1. 检查容器是否合法

当执行 `ReactDOM.render( element, container )` 时，React 要做的第一件就是检查 `container` 是否合法，见代码示例 5.2.1。

```
render: function (element, container, callback) {
  (function () {
    // 检查container是否是有效的容器
    if (!isValidContainer(container)) {
      {
        // 如果容器不合法则直接抛出异常
        throw ReactError(Error('Target container is not a DOM element.'));
      }
    }
  })();
  // 如果容器通过检查，则继续执行后续的逻辑
  return legacyRenderSubtreeIntoContainer(null, element, container, false, callback);
}
```

代码示例 5.2.1 render 函数检查容器是否合法

2. 初始化 fiberRoot 对象

前面提到过，fiberRoot 对象时整个 Fiber 架构的「根基」，它是整个 Fiber 树的「根」。React 是如何创建这个对象的呢？见代码示例 5.2.2。

```
// 源码位置：packages/react-reconciler/src/ReactFiberRoot.js
function createFiberRoot(containerInfo, tag, hydrate) {
  var root = new FiberRootNode(containerInfo, tag, hydrate);

  // 下面会形成一个闭环结构
  // createHostRootFiber函数用于创建Fiber树的第一个Fiber结点，其tag属性值为HostRoot（常量3）
  var uninitializedFiber = createHostRootFiber(tag);
  root.current = uninitializedFiber;
  uninitializedFiber.stateNode = root;

  return root;
}
```

代码示例 5.2.2 创建 fiberRoot 对象

fiberRoot 对象是一个「闭环」结构，该对象上面的 `current` 属性指向了 Fiber 树的第一个 Fiber 结点，而该结点的 `stateNode` 属性又指向了 fiberRoot 对象。fiberRoot 对象的结构见图 5.2.1。

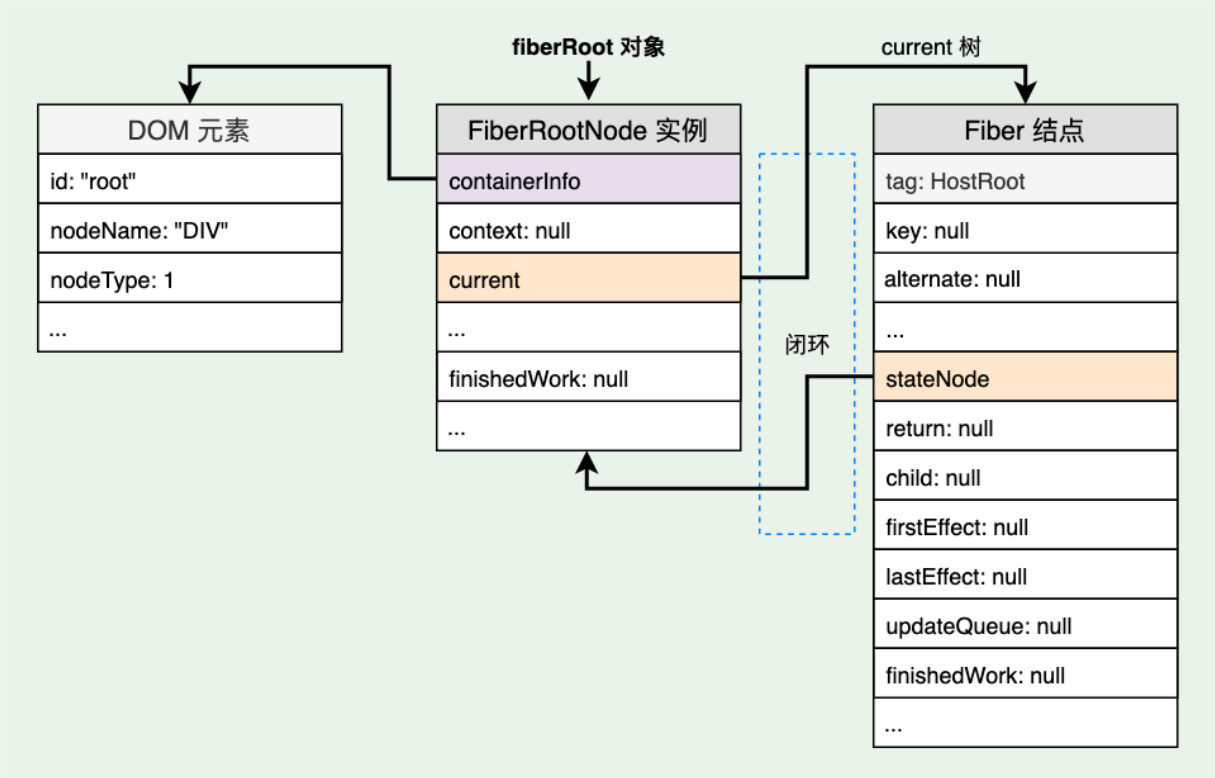

图 5.2.2 fiberRoot 对象初始化完成后的结构

在 fiberRoot 对象中，通过 current 属性指向了第一个 Fiber 结点，这个结点也就是 Fiber 树的根结点，同时该结点的 stateNode 属性又指向了 fiberRoot 对象，因此 fiberRoot 对象上面就形成了一个闭环。React 为什么要这样设计呢？个人认为主要是方便找到 Fiber 树根结点的 stateNode 属性中的值。

> Fiber 结点的 stateNode 属性存储的当前结点的最终产物，如果是 ClassComponent 类型的结点则该属性指向的是当前 class 组件的实例，如果是 HostComponent 类型的结点则该属性指向的是当前结点的 DOM 实例，如果是 HostRoot 类型的结点则该属性指向的是 fiberRoot 对象。

3. 创建更新对象并将其加入到更新队列

fiberRoot 对象初始化完成后，React 需要为应用程序的首次渲染创建更新对象，见代码示例 5.2.2。

```
// 源码位置：packages/react-reconciler/src/ReactFiberReconciler.js
function scheduleRootUpdate(current, element, expirationTime, suspenseConfig, callback) {
  // ...
  // 创建更新对象
  var update = createUpdate(expirationTime, suspenseConfig);
  // 更新对象的内容为整个根组件元素
  update.payload = { element: element };

  callback = callback === undefined ? null : callback;
  if (callback !== null) {
    // 如果有回调函数则为更新对象添加回调
    update.callback = callback;
  }
  // 将更新加入到更新队列，current指向的是Fiber树的第一个结点
  enqueueUpdate(current, update);
  // 调度更新任务
  scheduleWork(current, expirationTime);
}
```

应用程序首次渲染过程中，fiberRoot 对象初始化完成后，React 要将创建更新并将其加入到更新队列，此时的更新队列中只有一个更新对象，该更新对象携带的内容就是应用程序根组件元素，此时的 fiberRoot 内部结构见下图 5.2.3。

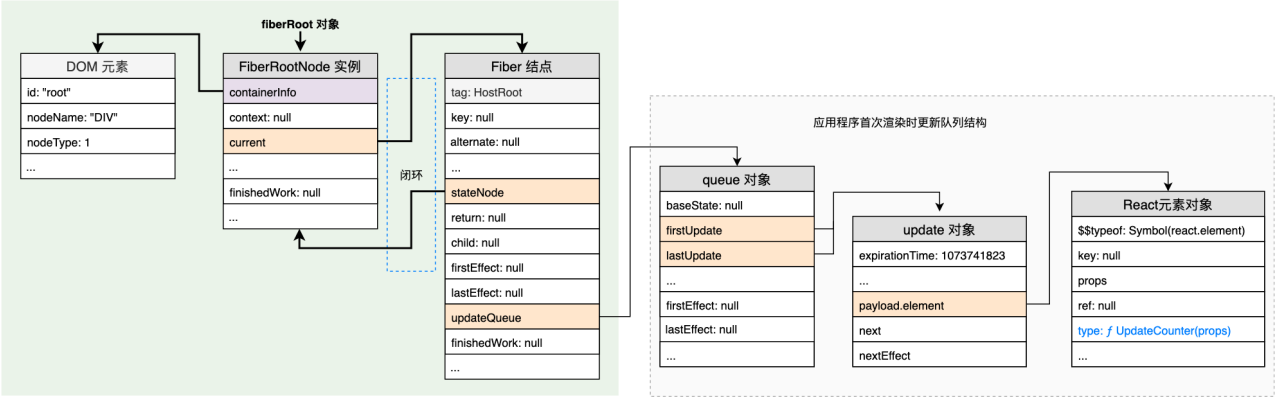

图 5.2.3 fiberRoot 对象添加更新队列后的结构

完成更新对象加入更新队列后，应用程序将会执行 scheduleWork(current, expirationTime) 进入任务调度过程。事实上，React 将应用程序首次渲染的任务设置为了同步任务，向任务调度器申请立即执行。任务调度器则立即安排执行该任务，随后应用程序首次渲染就进入了 render 阶段。

小结

React 在应用程序首次渲染时 prerender 阶段做的主要工作有检查容器是否合法，初始化 fiberRoot 对象，创建更新对象并将其加入到更新队列。随后，React 向任务调度器申请立即执行，任务调度器安排该任务立即执行，进而应用程序首次渲染就进入了 render 阶段。从下一节开始将会介绍 React 在应用程序首次渲染时 render 阶段所做的具体工作，首先介绍的就是 current 树与 workInProgress 树。

本文以及后续文章的所有对象结构图或者程序流程图均非 React 官方提供，这些图表仅供参考。图中的信息难免会有错误之处，请以程序执行结果为准。

}

20 React 应用程序首次渲染时内部运行流程概述

22 什么是 current 树和 workInProgress 树