26 React 如何将副作用更新到屏幕

更新时间：2020-09-25 09:53:13

> 人要有毅力，否则将一事无成。——居里夫人

前言

本章前面几节介绍了应用程序首次渲染时在 render 阶段 React 如何构建 workInProgress 树以及收集副作用等工作。在 render 阶段结束时 fiberRoot 对象上面将会得到一个副作用列表（Effect List），这个副作用列表中携带的更新内容就是要更新到屏幕中的信息。本节将会介绍 React 如何将副作用列表中的信息更新到屏幕以及 React 在这个过程中主要做了哪些工作。

应用程序执行时进入 commit 阶段的时机

进入 commit 阶段的入口是 `commitRoot` 函数，该函数也是在 `renderRoot` 函数中调用，见代码示例 5.7.1。

```
function renderRoot(root, expirationTime, isSync) {
  // 此处形参root为fiberRoot对象
  ...
  // root.current.alternate指向的是workInProgress对象树
  root.finishedWork = root.current.alternate;
  ...
  switch (workInProgressRootExitStatus) {
    ...
    case RootCompleted:
     // 进入commit阶段
     return commitRoot.bind(null, root);
  }
}
```

代码示例 5.7.1 commitRoot 函数的执行时机

工作循环执行结束，标志着应用程序渲染过程中 render 阶段工作的完成。此时 fiberRoot 对象上面的 workInProgress 树存储在 root.current.alternate 指向的内存单元。React 会将 workInProgress 树赋值到 fiberRoot 对象上的 finishedWork 属性中。然后，检查 workInProgressRootExitStatus 的状态，如果是正常结束，则应用程序渲染任务进入 commit 阶段。

副作用 commit 过程

副作用 commit 时程序的整体执行流程见下图 5.7.1。

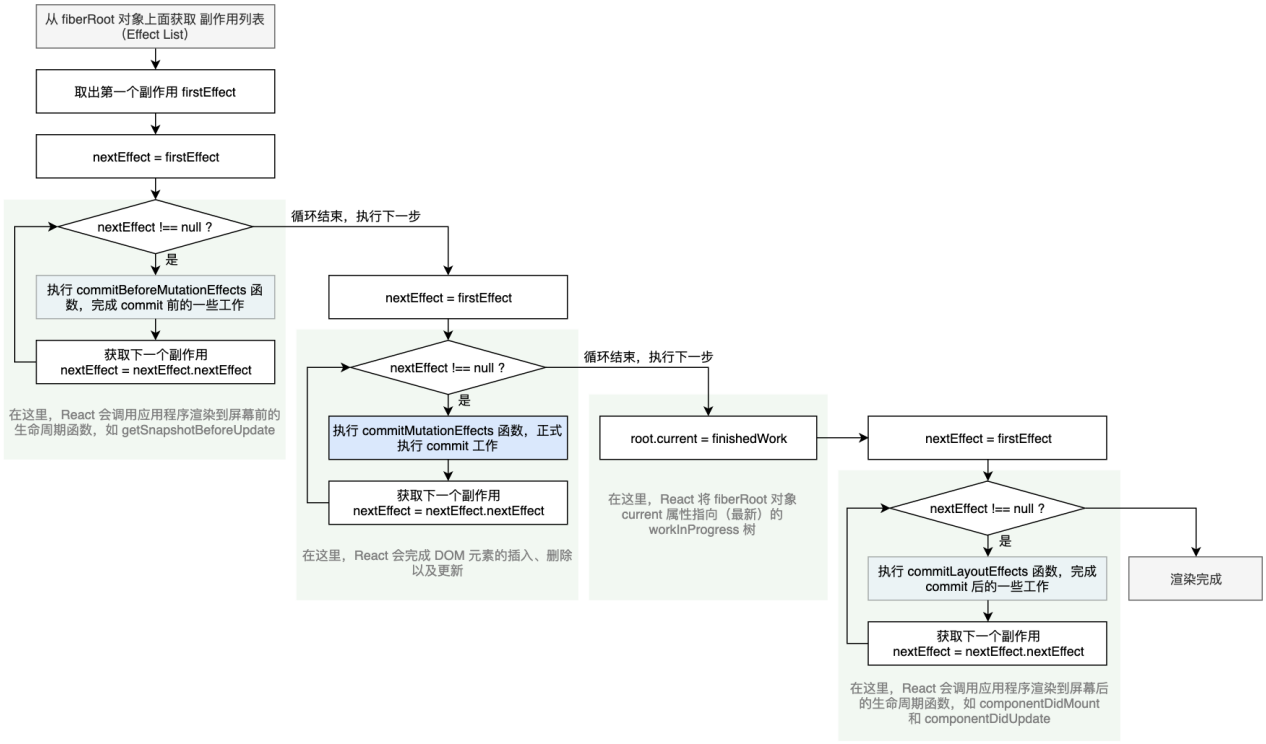

图 5.7.1 副作用 commit 时程序执行流程

应用程序执行在 commit 阶段时，React 将这个过程又分为三个步，分别是 **before commit、commit 和 after commit**。在这三个步分别调用 commitBeforeMutationEffects，commitMutationEffects 和 commitLayoutEffects 三个函数来执行具体的工作。commitRoot 函数内部调用 commitRootImpl 函数实现 commit 阶段的具体逻辑，见代码示例 5.7.2。

```
// 源码位置：packages/react-reconciler/src/ReactFiberWorkLoop.js
function commitRootImpl(root, renderPriorityLevel) {
  // ...
  // 获取workInProgress对象树
  var finishedWork = root.finishedWork;
  // ...
  var firstEffect = void 0;
  if (finishedWork.effectTag > PerformedWork) {
    // Fiber树的副作用列表只能由它的孩子组成，不包括它自己。如果根节点有一个副作用，那么需要将这个副作用放到
    // 副作用列表的尾部。最终的副作用列表将是不包括根节点的父节点的集合，如果只有一个副作用，那么就是根节点。
    if (finishedWork.lastEffect !== null) {
      finishedWork.lastEffect.nextEffect = finishedWork;
      firstEffect = finishedWork.firstEffect;
    } else {
      // 此时的副作用列表就是根节点（一般是首次渲染）
      firstEffect = finishedWork;
    }
  } else {
    // 如果在根节点没有副作用，则取finishedWork.firstEffect
    firstEffect = finishedWork.firstEffect;
  }

  // 接下来开始遍历副作用列表，先获取副作用列表的引用
  // 1. commit副作用之前处理生命周期函数
  nextEffect = firstEffect;
  do {
      invokeGuardedCallback(null, commitBeforeMutationEffects, null);
    // ...
  } while (nextEffect !== null);

  // 2. 正式commit副作用
  nextEffect = firstEffect;
  do {
   invokeGuardedCallback(null, commitMutationEffects, null, renderPriorityLevel);
    // ...
  } while (nextEffect !== null);

  // 3. commit布局副作用，意思是更新完成后的逻辑，如处理生命周期函数，重置fiber树等
  nextEffect = firstEffect;
  do {
    invokeGuardedCallback(null, commitLayoutEffects, null, root, expirationTime);
    // ...
  } while (nextEffect !== null);
}
```

代码示例 5.7.2 commit 阶段的三个步骤

代码示例 5.7.2 中，`finishedWork.effectTag > PerformedWork` 判断的是当前副作用是否需要更新，因为 `NoEffect` 和 `PerformedWork` 的值分别为 `0` 和 `1`，只有大于这两个值就说明有更新需要执行。

commitBeforeMutationEffects 函数（执行于副作用 commit 前）

commit 副作用之前会先在 `commitBeforeMutationEffects` 函数中处理相应的生命周期函数，见代码示例 5.7.3。

```
// 源码位置：packages/react-reconciler/src/ReactFiberWorkLoop.js
function commitBeforeMutationEffects() {
  while (nextEffect !== null) {
    ...
    // 调用 getSnapshotBeforeUpdate
    commitBeforeMutationLifeCycles(current, nextEffect);
    ...
    nextEffect = nextEffect.nextEffect;
  }
}
```

代码示例 5.7.3 应用程序 commit 前处理生命周期函数

正式 commit 副作用前，React 会调用结点的 getSnapshotBeforeUpdate 生命周期函数。

commitMutationEffects 函数（正式 commit 副作用）

在 commitMutationEffects 函数中正式 commit 副作用，将副作用更新到屏幕，见代码示例 5.7.4。

```
// 源码位置：packages/react-reconciler/src/ReactFiberWorkLoop.js
function commitMutationEffects(renderPriorityLevel) {
  while (nextEffect !== null) {
    ...
    // 获取副作用类型
    var effectTag = nextEffect.effectTag;
    ...
    var primaryEffectTag = effectTag & (Placement | Update | Deletion);
    // 根据 effectTAg 执行对应的操作
    switch (primaryEffectTag) {
      case PlacementAndUpdate:
        // 首次渲染时插入到屏幕
        commitPlacement(nextEffect);
        nextEffect.effectTag &= ~Placement;
        var _current = nextEffect.alternate;
        // 开始commit工作
        commitWork(_current, nextEffect);
        break;
    }
    ...
    nextEffect = nextEffect.nextEffect;
  }
}
```

代码示例 5.7.4 应用程序的副作用正式 commit

应用程序 commit 副作用时，会根据 EffectTag 的类型执行对应的操作（插入，删除，更新等）。应用首次渲染时副作用类型会匹配到 PlacementAndUpdate，具体处理逻辑在 commitPlacement 函数中，该函数主要负责插入 DOM 元素到屏幕，见代码示例 5.7.5。

```
// 源码位置：packages/react-reconciler/src/ReactFiberCommitWork.js
function commitPlacement(finishedWork) {
  // 形参finishedWork指的的当前需要执行的副作用（Fiber）结点
  // ...
  // 获取容器
  var parent = parentStateNode.containerInfo;
  var node = finishedWork;
  var isHost = node.tag === HostComponent || node.tag === HostText;
  var stateNode = isHost ? node.stateNode : node.stateNode.instance;
  // 操作DOM，将更新内容绘制到屏幕
  appendChildToContainer(parent, stateNode);
}
```

代码示例 5.7.5 将根据副作用生成的 DOM 元素插入到页面中对应位置

前面已经介绍过，HostComponent 类型的 Fiber 结点在完成工作单元时会创建当前结点对应的 DOM 元素实例，并将其赋值到该结点的 stateNode 属性上。commitPlacement 函数中的 node.stateNode 指向的就是当前结点对应的 DOM 元素实例。执行完 appendChildToContainer(parent, stateNode) 后应用程序中的副作用就会渲染到屏幕上。

应用程序中的副作用就会渲染到屏幕上后 React 还需要做一些收尾工作，如将当前的 workInProgress 树赋值给 current 树，调用更新完成后的生命周期函数等。

workInProgress 树赋值给 current 树

commitMutationEffects 函数执行完成后，需要更新的内容已经绘制到了屏幕上，此时的 workInProgress 树也就成了应用程序最新的状态树。因此 React 将 fiberRoot.current 指向（最新）的 workInProgress 树。

```
// 这里的finishedWork引用的是workInProgress树
root.current = finishedWork;
```

此时，fiberRoot 对象的内部结构如图 5.7.1。

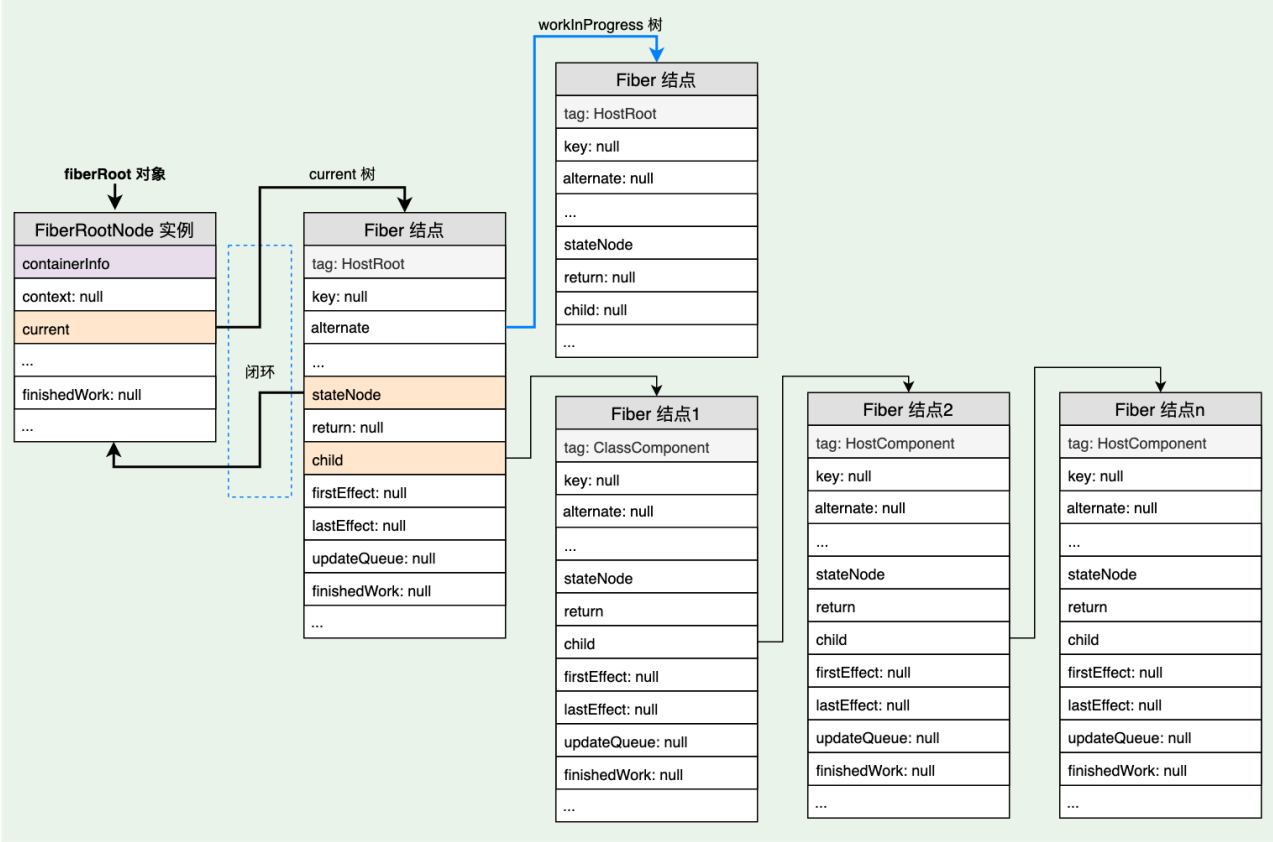

图 5.7.2 应用程序首次渲染完成后的 fiberRoot 对象内部结构

commitLayoutEffects 函数（执行于副作用 commit 后）

commitLayoutEffects 函数在更新内容 commit 到屏幕之后执行，主要负责 commit 阶段的一些收尾工作，见代码示例 5.7.6。

```
// 源码位置：packages/react-reconciler/src/ReactFiberCommitWork.js
function commitLayoutEffects(root, committedExpirationTime) {
  while (nextEffect !== null) {
    // ...
    var current = nextEffect.alternate;
    // class类型的组件，会调用instance.componentDidMount()和
    // instance.componentDidUpdate(prevProps, prevState, ...);
    commitLifeCycles(root, current, nextEffect, committedExpirationTime);
    // ...
    nextEffect = nextEffect.nextEffect;
  }
}
```

代码示例 5.7.6 commit 完成后调用相关生命周期函数

应用程序的副作用更新到屏幕后，对于 `ClassComponent` 类型的结点，React 会调用该结点的生命周期函数，如 `componentDidMount` 和 `componentDidUpdate`。

小结

本节主要介绍了在应用程序首次渲染时 React 如何将副作用列表的中的每一个副作用携带的更新信息渲染到屏幕，这个过程的主要工作就是生命周期函数的调用以及 DOM 元素的插入（删除或者更新）等。

本章主要介绍应用程序首次渲染时在不同的阶段 React 所做的一些重点工作，包括构建 fiberRoot 对象，构建 workInProgress 树，收集副作用列表以及将副作用渲染到屏幕等。事实上还有一些重要的内容并没有介绍到，就是 React Fiber 架构中结点的「协调」与「diff」逻辑，这部分内容将会在下一章应用程序更新渲染时 React 所做的工作内容中进行详细介绍。

看到这里，我们应该对 React 的内部执行机制有了一个整体的认识。在这里建议读者重新读一遍前面的文章加以深对应用程序执行过程中 React 内部执行机制的理解。

}

25 完成工作单元：创建 DOM 元素实例与收集副作用

27 React 应用程序更新渲染时内部运行流程概述