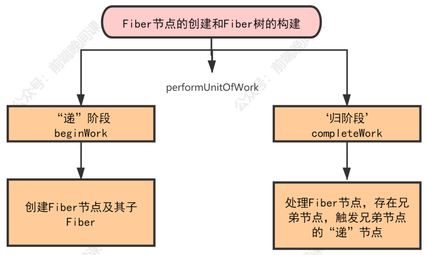
Fiber树结构



<App/>即为rootFiber

render阶段流程概览

render阶段开始

是否异步可中断更新

- 是 → performConcurrentWorkOnRoot方法 → workLoopConcurrent方法 shouldYield判断是否当前浏览器帧是否有剩余时间，没有会中断更新
- 否 → performSyncWorkOnRoot方法 → workLoopSync方法

performUnitOfWork(workInProgress)
递”和“归”阶段会交错执行直到“归”到rootFiber。至此，render阶段的工作就结束了

“递”阶段

beginWork(current, workInProcess, renderLanes)

current：当前组件对应的Fiber节点在上一次更新时的Fiber节点，即workInProgress.alternate

workInProgress：当前组件对应的Fiber节点

renderLanes：优先级相关，此处省略

update/mount

根据current是否为null判断

- update → 是否可以复用
  - 根据前后props和type判断 current.memoizedProps === workInProgress.pendingProps && workInProgress.type === current.type
  - 可以，didReceiveUpdate = false → bailoutOnAlreadyFinishedWork(current, workInProgress, renderLanes) → 判断子树是否需要更新
    - 需要 → cloneChildFiber(current, workInProgress) → return workinProcess.child
    - 不需要 → return null
  - 不可以，didReceiveUpdate =true
- mount

根据workInProgress.tag区别对待（ClassComponent/FunctionComponent/HostRoot...）

- update → reconcileChildren(current,workInProgress,nextChildren,renderLanes) → reconcileChildFibers(workInProgress, current.child, nextChildren) → diff算法 → 生成effectTag的Fiber节点
  - 工作结束后会通知Renderer在commit阶段需要执行的DOM操作，要执行DOM操作的具体类型就保存在fiber.effectTag中
- mount → mountChildFibers(workInProgress, null, nextChildren) → 生成新的子Fiber节点（workinProcess.child）
  - mount也是会有effectTag产生，但是只会在rootFiber存在Placement effectTag，避免所有子Fiber节点都需要执行Placement effectTag操作

“归”阶段

completeWork(current, workInProcess, renderLanes)

current：当前组件对应的Fiber节点在上一次更新时的Fiber节点，即workInProgress.alternate

workInProgress：当前组件对应的Fiber节点

renderLanes：优先级相关，此处省略

作用就是在上一个节点diff完成后，对它进行一些收尾工作，表现就是对已产生的effectTag的Fiber节点执行相应的操作

根据workInProgress.tag走不同的路

- 重点关注页面渲染所必须的**HostComponent**（即原生DOM组件对应的Fiber节点），其他的讲解Update的时候描述

update/mount

current !== null && workInProgress.stateNode != null来判断是否为update，需要判断workInProgress.stateNode来查看Fiber节点是否存在对应的DOM节点

- update → updateHostComponent(current, workInProgress, type, newProps, rootContainerInstance,);
  - 作用是diff props，返回一个需要更新的属性数组，处理完后的props会以数组形式存在，[key1, value1, key2, value2...]然后存储在workInProgress.updateQueue，并最终会在commit阶段被渲染在页面上
- mount → createInstance(type, newProps, rootContainerInstance, currentHostContext, workInProgress,);
  - 为fiber创建对应DOM节点，创建好的Dom节点会赋值给workInProgress.stateNode
  - appendAllChildren(instance, workInProgress, false, false);
    - 将子孙DOM节点插入刚生成的DOM节点（instance）中，验证了mount时只会在rootFiber存在Placement effectTag，只执行一次插入DOM操作
  - finalizeInitialChildren(instance, type, newProps, rootContainerInstance, currentHostContext,)
    - 与update逻辑中的updateHostComponent类似的处理props的过程，初始化Dom对象的事件监听和内部属性

将有effectTag的fiber挂载在父级Fiber节点的effectList末尾（fiber.nextEffect ），并返回下一个workInProcess fiber节点

每个执行完completeWork且存在effectTag的Fiber节点会被保存在一条被称为effectList的单向链表中,effectList中第一个fiber节点保存在fiber.firstEffect，最后一个元素保存在fiber.lastEffect,最终形成一条以rootFiber.firstEffect为起点的单向链表,这样，在commit阶段只需要遍历effectList就能执行所有effect了

```
                    nextEffect        nextEffect
rootFiber.firstEffect ----------> fiber ----------> fiber
```

render阶段全部工作完成

performSyncWorkOnRoot方法

fiberRootNode传入

commitRoot(root)
进入commit阶段

公众号：前端晚间课