commit阶段流程概览

	流程	说明
1	结束render阶段工作	结果：rootFiber.firstEffect上保存了一条需要执行副作用的Fiber节点的单向链表effectList，Fiber节点的updateQueue中保存了变化的props。
2	commitRoot(fiberRootNode)	fiberRootNode会作为传参，一些生命周期钩子（比如componentDidXXX）、hook（比如useEffect）需要在commit阶段执行。
3	before mutation阶段之前	1、flushPassiveEffects() 2、重置优先级相关 3、清除已完成的discrete updates 4、重置全局变量 5、将rootFiber的effectTag添加到effectList尾部
4	before mutation阶段（执行Dom操作之前）	1、处理DOM节点渲染/删除后的 autoFocus、blur逻辑。 2、调用getSnapshotBeforeUpdate生命周期钩子。 3、异步调度useEffect。
5	mutation阶段（执行Dom操作）	1、根据ContentReset effectTag重置文字节点 2、更新ref 3、根据effectTag分别处理，其中effectTag包括(Placement | Update | Deletion | Hydrating)
6	layout阶段（执行Dom操作后）	1、DOM渲染完成（mutation阶段完成）后执行 2、该阶段触发的生命周期钩子和hook可以直接访问到已经改变后的DOM 执行如下： commitLayoutEffectOnFiber（调用生命周期钩子和hook相关操作） commitAttachRef（赋值 ref）
	layout阶段之后	1、useEffect相关的处理 2、性能追踪相关 3、在commit阶段会触发一些生命周期钩子（如componentDidXXX）和hook（如useLayoutEffect、useEffect）

3

before mutation阶段之前

- flushPassiveEffects()
  - 触发useEffect回调与其他同步任务。由于这些任务可能触发新的渲染，所以这里要一直遍历执行直到没有任务
    前提：rootWithPendingPassiveEffects !== null（layout阶段后赋值）
    rootWithPendingPassiveEffects 就是effectList
- 重置优先级相关变量
  - root指 fiberRootNode，
    root.finishedWork指当前应用 的rootFiber。
    // 凡是变量名带lane的都是优先级相关
    root.finishedWork = null;
    root.finishedLanes = NoLanes;
    // 重置Scheduler绑定的回调函数
    root.callbackNode = null;
    root.callbackId = NoLanes;
    // 重置优先级相关变量
    markRootFinished(root, remainingLanes);
- 清除已完成的discrete updates
  例如：用户鼠标点击触发的更新
  - 前提：rootWithPendingPassiveEffects !== null（layout阶段后赋值）
- 重置全局变量
  - workInProgressRoot = null;
    workInProgress = null;
    workInProgressRootRenderLanes = NoLanes;
- 将rootFiber的effectTag添加到effectList尾部
  - 将effectList赋值给firstEffect，由于每个fiber的effectList只包含他的子孙节点，所以根节点如果有effectTag则不会被包含进来，所以这里将有effectTag的根节点插入到effectList尾部，这样才能保证有effect的fiber都在effectList中

重点关注

```
if (finishedWork.effectTag > PerformedWork) {
  if (finishedWork.lastEffect !== null) {
    finishedWork.lastEffect.nextEffect = finishedWork;
    firstEffect = finishedWork.firstEffect;
  } else {
    firstEffect = finishedWork;
  }
} else {
  // 根节点没有effectTag
  firstEffect = finishedWork.firstEffect;
}
```

4

before mutation阶段（执行Dom操作之前）

- 遍历EffectList
  - 每次遍历经过以下流程
  - commitBeforeMutationEffects(finishedWork)
    - finishedWork即rootFiber
  - 内部调起
  - getSnapshotBeforeUpdate()
    返回的值会输送给ComponentUpdate
    - componentWillXXX钩子前会被增加了UNSAFE_前缀，因为改用Fiber结构，变成可中断，类似钩子会被多次调起，而commit阶段为同步，只执行一次，因此用此替代
    - 遍历了，为啥getSnapshotBeforeUpdate()还不会执行多次？详见状态更新原理图
  - 调度useEffect
  - 调度步骤
    - 1、before mutation阶段在scheduleCallback中调度flushPassiveEffects()
      2、layout阶段之后将effectList赋值给rootWithPendingPassiveEffects
      3、scheduleCallback触发flushPassiveEffects，flushPassiveEffects内部遍历rootWithPendingPassiveEffects
      - 调度：指的是就是1帧16ms处理不完，留给下一帧去继续执行的意思。
        此处flushPassiveEffects()会调起useEffect回调，但是前提rootWithPendingPassiveEffects(effectList)被赋值，effectList是effectTag Fiber链表，就是更新/删除等的标记，FunctionCompoent使用useEffect就会被打上effectTag标记，rootWithPendingPassiveEffects是在layout才赋值，所以flushPassiveEffects()只有layout后才被触发，这叫异步调度，所以useEffect异步执行的。

useEffect异步执行的原因主要是防止同步执行时阻塞浏览器渲染

5

mutation阶段（执行Dom操作）

- 遍历EffectList
  - commitMutationEffects（root: FiberRoot）
  - 内部调起
  - commitResetTextContent(nextEffect)；
    - 根据 ContentReset effectTag重置文字节点
  - commitDetachRef(current)
    - 更新ref
  - 接下来对根据effectTag进行处理
    下面只列举四种情况
    - 1 Placement（插入Dom，调起commitPlacement(nextEffect)方法）
      - 2 Update（更新Dom，调起commitWork(current, nextEffect)方法）
    - PlacementAndUpdate（插入DOM 并 更新DOM，调起commitPlacement(nextEffect)与commitWork(current, nextEffect)方法）
      - 3 Deletion（删除Dom，调起commitDeletion(current, nextEffect)方法）

1

commitPlacement(nextEffect)

- 获取父级DOM节点
- getHostParentFiber(finishedWork)
  finishedWork为传入的Fiber节点
- getHostSibling(finishedWork)
  获取Fiber节点的DOM兄弟节点
- 根据DOM兄弟节点是否存在决定调用parentNode.insertBefore或parentNode.appendChild执行DOM插入操作

2

commitWork(current, nextEffect)
current为nextEffect.alternate

- 判断fiber.tag
  - HostComponent 原生组件
    - commitUpdate
      - 将render阶段 completeWork（opens new window）中为Fiber节点赋值的updateQueue对应的内容渲染在页面上
    - hostComponent的updateQueue，即props变化更新
  - FunctionComponent
    - commitHookEffectListUnmount
      - 遍历effectList
        - mutation阶段会执行useLayoutEffect的销毁函数（更新Dom操作时）

3

commitDeletion(current, nextEffect)

- fiber.tag为ClassComponent的componentWillUnmount（opens new window）生命周期钩子调起
- 解绑ref
- 调度useEffect的销毁函数

6

layout阶段（执行Dom操作后）

- 遍历EffectList
  - commitLayoutEffects()
  - 内部调起
    - commitAttachRef(nextEffect)；
      赋值 ref
      就是执行ref方法，就是我们正常使用的React.createRef()赋值ref属性
  - 内部调起
    - commitLayoutEffectOnFiber(root, current, nextEffect, committedLanes)
      - fiber.tag判断
        - classComponent
          - current === null
            此时的生命周期是能拿到Dom
            - componentDidUpdate
            - mount
              - componentDidMount
        - FunctionCompoent
          - commitHookEffectListMount
            （执行useLayoutEffect的回调函数）
            - schedulePassiveEffect
              （调度useEffect的销毁函数与回调函数）
    - this.setState的第二参数回调，会在此执行
      ReactDOM.render的第三个参数回调会在此执行

为啥useLayoutEffect为同步?useEffect为异步?
mutation阶段执行了上一次更新时的useLayoutEffect注销函数，layout阶段调度useLayoutEffect回调，同步执行。useEffect在mutation之前阶段被调度，但是rootWithPendingPassiveEffects(effectList)未被赋值，所以等到layout阶段赋值完后回调和注销被调度，所以是异步调度

- root.current = finishedWork;

双缓存机制，workInProcess Fiber树会变成current Fiber树，通过修改fiberRootNode.cuurent指针