12-3 节点删除

1. vdom 删除(不用做) 2. dom 删除

```
JavaScript
```

```
function FunctionComponent() {
  const [count1, setCount1] = useReducer((x) => x + 1, 0);

  return (
    <div className="border">
      {/* <h3>函数组件</h3> */}
      {count1 % 2 === 0 ? (
        <button
          onClick={() => {
            setCount1();
          }}
        >
          {count1}
        </button>
      ) : (
        <span>react</span>
      )}
    </div>
  );
}
```

当新节点是单个节点的时候

packages/react-reconciler/src/ReactChildFiber.ts

TypeScript

```
function reconcileSingleElement(
    returnFiber: Fiber,
    currentFirstChild: Fiber | null,
    element: ReactElement
  ) {
    // 节点复用的条件
    // ! 1. 同一层级下 2. key相同 3. 类型相同
    const key = element.key;
    let child = currentFirstChild;

    while (child !== null) {
      if (child.key === key) {
```

```
        const elementType = element.type;
        if (child.elementType === elementType) {
          // todo 后面其它fiber可以删除了
          deleteRemainingChildren(returnFiber, child.sibling);
          const existing = useFiber(child, element.props);
          existing.return = returnFiber;
          return existing;
        } else {
          deleteRemainingChildren(returnFiber, child);
          // 前提：React不认为同一层级下有两个相同的key值
          break;
        }
      } else {
        // 删除单个节点
        deleteChild(returnFiber, child);
      }
      // 老fiber节点是单链表
      child = child.sibling;
    }

    let createdFiber = createFiberFromElement(element);
    createdFiber.return = returnFiber;
    return createdFiber;
  }
```

删除单个节点

packages/react-reconciler/src/ReactChildFiber.ts

TypeScript

```
function deleteChild(returnFiber: Fiber, childToDelete: Fiber): void
  if (!shouldTrackSideEffects) {
    // Noop.
    return;
  }
  const deletions = returnFiber.deletions;
  if (deletions === null) {
    returnFiber.deletions = [childToDelete];
    returnFiber.flags |= ChildDeletion;
```

```
    } else {
      deletions.push(childToDelete);
    }
  }
```

删除多个节点

packages/react-reconciler/src/ReactChildFiber.ts

TypeScript

```
function deleteRemainingChildren(
  returnFiber: Fiber,
  currentFirstChild: Fiber | null
): null {
  if (!shouldTrackSideEffects) {
    return null;
  }

  let childToDelete = currentFirstChild;
  while (childToDelete !== null) {
    deleteChild(returnFiber, childToDelete);
    childToDelete = childToDelete.sibling;
  }
  return null;
}
```

commit

packages/react-reconciler/src/ReactFiberCommitWork.ts

TypeScript

```
// 提交协调的产生的effects，比如flags，Placement、Update、ChildDeletion
function commitReconciliationEffects(finishedWork: Fiber) {
  const flags = finishedWork.flags;
```

```
  if (flags & Placement) {
    // 页面初次渲染 新增插入 appendChild
    // todo 页面更新，修改位置 appendChild || insertBefore
    commitPlacement(finishedWork);
    finishedWork.flags &= ~Placement;
  }
  if (flags & ChildDeletion) {
    // parentFiber 是 deletions 的父dom节点对应的fiber
    const parentFiber = isHostParent(finishedWork)
      ? finishedWork
      : getHostParentFiber(finishedWork);
    const parent = parentFiber.stateNode;
    commitDeletions(finishedWork.deletions!, parent);
    finishedWork.deletions = null;
  }
}
```

删除多个 DOM 节点

packages/react-reconciler/src/ReactFiberCommitWork.ts

TypeScript

```
function commitDeletions(deletions: Array<Fiber>, parent: Element) {
  deletions.forEach((deletion) => {
    // 找到deletion的dom节点
    parent.removeChild(getStateNode(deletion));
  });
}

function getStateNode(fiber: Fiber) {
  let node = fiber;

  while (1) {
    if (isHost(node) && node.stateNode) {
      return node.stateNode;
    }
    node = node.child as Fiber;
  }
}
```