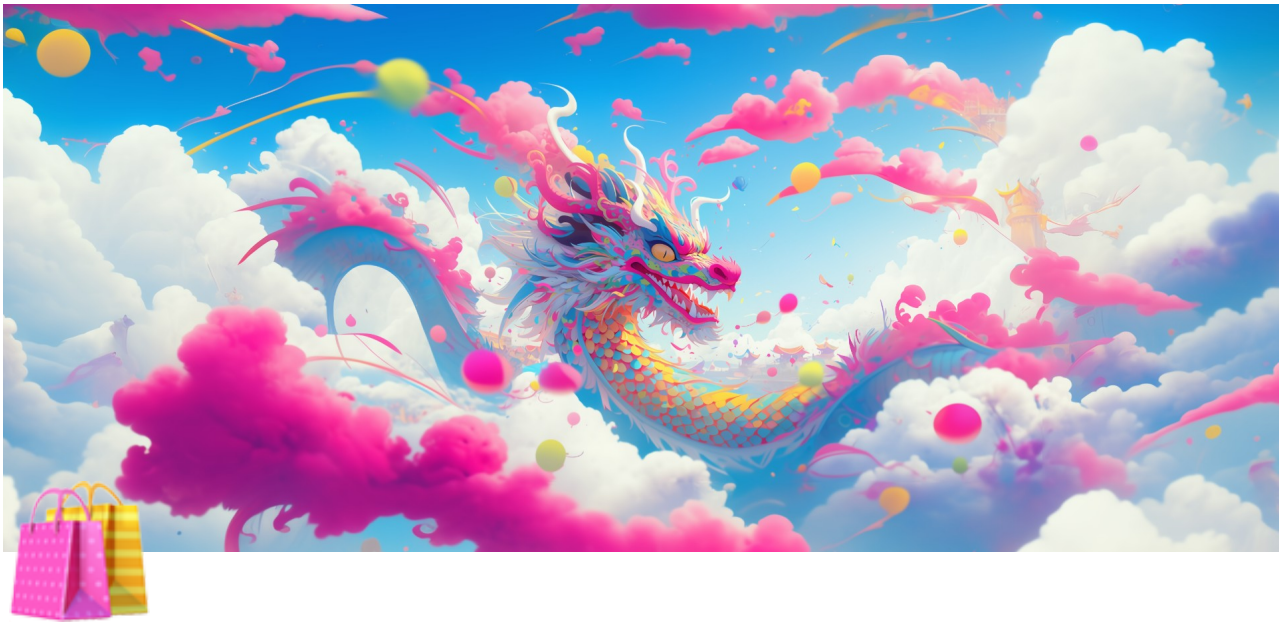

14–7 如何实现 useEffect 的 effect 执行

```TypeScript
useEffect(setup, dependencies?)
```

- `setup`：处理 Effect 的函数。setup 函数选择性返回一个 **清理（cleanup）** 函数。当组件被添加到 DOM 的时候，React 将运行 setup 函数。在每次依赖项变更重新渲染后，**React 将首先使用旧值运行 cleanup 函数（如果你提供了该函数），然后使用新值运行 setup 函数。在组件从 DOM 中移除后，React 将最后一次运行 cleanup 函数。**
- **可选** `dependencies`：`setup` 代码中引用的所有响应式值的列表。响应式值包括 props、state 以及所有直接在组件内部声明的变量和函数。如果你的代码检查工具 配置了 React，那么它将验证是否每个响应式值都被正确地指定为一个依赖项。依赖项列表的元素数量必须是固定的，并且必须像 `[dep1, dep2, dep3]` 这样内联编写。React 将使用 `Object.is` 来比较每个依赖项和它先前的值。如果省略此参数，则在每次重新渲染组件之后，将重新运行 Effect 函数。

useLayoutEffect 与 useEffect

函数签名相同。**effect 执行时机不同。**

但是 useLayoutEffect 会在所有的 DOM 变更之后同步调用 effect。

useEffect 会在组件渲染到屏幕之后延迟执行 effect。

例子

TypeScript

```
function FunctionComponent() {
  const [count1, setCount] = useReducer((x) => x + 1, 0);
  const [count2, setCount2] = useState(0);

  // layout effect
  useLayoutEffect(() => {
    console.log("useLayoutEffect"); //sy-log
  }, [count1]);

  // passive effect
  useEffect(() => {
    console.log("useEffect"); //sy-log
  }, [count2]);

  return (
    <div className="border">
      <h1>函数组件</h1>
      <button onClick={() => setCount()}>{count1}</button>
      <button onClick={() => setCount2(count2 + 1)}>{count2}</button>
      <Child count1={count1} count2={count2} />
    </div>
  );
}

function Child({ count1, count2 }: { count1: number; count2: number })
  // layout effect
  useLayoutEffect(() => {
    console.log("useLayoutEffect Child"); //sy-log
  }, [count1]);

  // passive effect
  useEffect(() => {
    console.log("useEffect Child"); //sy-log
```

```
  }, [count2]);

  return <div>Child</div>;
}
```

commit 阶段

TypeScript

```
function commitRoot(root: FiberRoot) {
  // !1. commit阶段开始
  const prevExecutionContext = executionContext;
  executionContext |= CommitContext;
  // !2.1 mutation阶段, 渲染DOM树
  commitMutationEffects(root, root.finishedWork as Fiber); //Fiber,Hos
  // !2.2 passive阶段, 执行passive effects
  flushPassiveEffects(root.finishedWork!);

  // !3. commit结束
  executionContext = prevExecutionContext;
  workInProgressRoot = null;
}
```

flushPassiveEffects 执行 passive effects

如果从 finishedWork 检查到其有 passive effects，那么这个时候要先遍历其子节点，然后提交每一层节点的 passive effects。然后提交自己的 passive effects。

TypeScript

```
export function flushPassiveEffects(finishedWork: Fiber) {
  // !1. 遍历子节点，检查子节点
  recursivelyTraversePassiveMountEffects(finishedWork);
  // !2. 如果有passive effects，执行~
  commitPassiveEffects(finishedWork);
}
```

recursivelyTraversePassiveMountEffects

TypeScript

```
function recursivelyTraversePassiveMountEffects(finishedWork: Fiber) {
  let child = finishedWork.child;
  while (child !== null) {
    recursivelyTraversePassiveMountEffects(child);
    commitPassiveEffects(child);
    child = child.sibling;
  }
}
```

commitPassiveEffects

TypeScript

```
function commitPassiveEffects(finishedWork: Fiber) {
  switch (finishedWork.tag) {
    case FunctionComponent: {
      if (finishedWork.flags & Passive) {
        commitHookEffectListMount(HookPassive, finishedWork);
        finishedWork.flags &= ~Passive;
      }
      break;
    }
  }
}
```