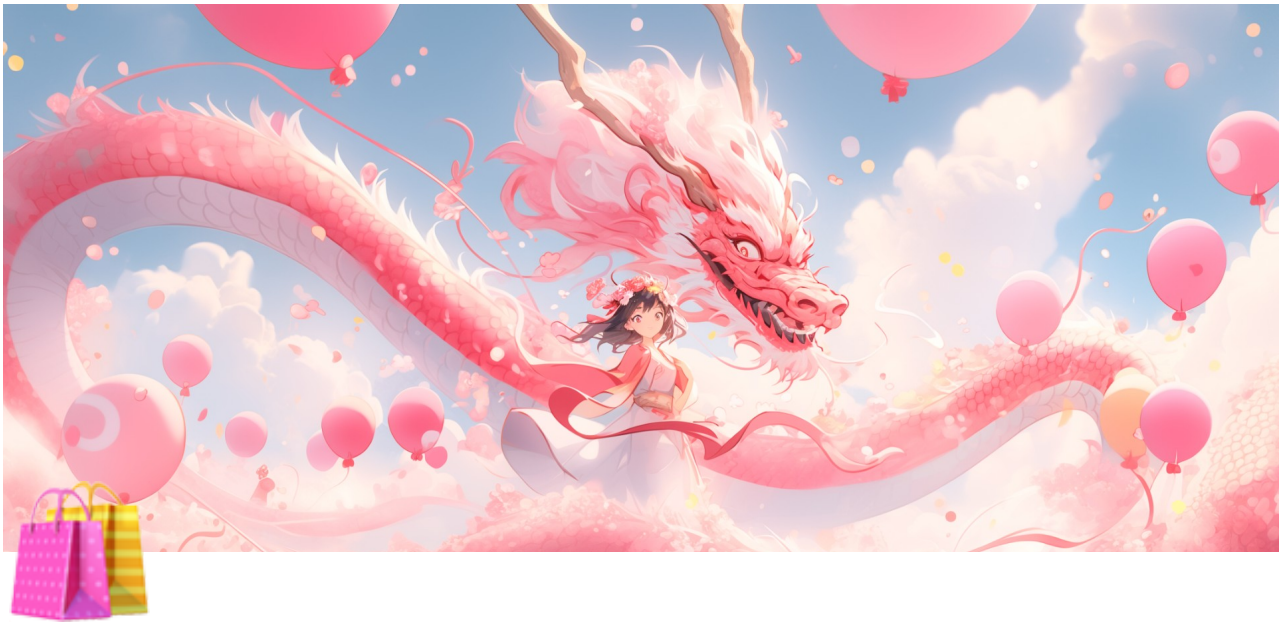

11-3 函数组件的 Hook 源码解读

如果以下代码没有特殊标记路径的源码，那么路径都是 react/packages/react-reconciler/src/ReactFiberHooks.js。

资源

1. useReducer

`useReducer` 是一个 React Hook，它允许你向组件里面添加一个 reducer。

JavaScript
```
const [state, dispatch] = useReducer(reducer, initialArg, init?)
```

FunctionComponent

React JSX
```
import { useReducer, useState } from "react";

export default function UseReducerPage(props) {
  const [count1, setCount1] = useReducer((x, n) => x + n, 0);
```

```
  const [count2, setCount2] = useState(0);

  return (
    <div>
      <h3>FunctionComponent</h3>
      <button
        onClick={() => {
          setCount1(1);
        }}
      >
        useReducer {count1}
      </button>
      <button
        onClick={() => {
          setCount2(count2 + 1);
        }}
      >
        useState {count2}
      </button>
    </div>
  );
}
```

render 阶段

beginWork

DebugReact > src > react > packages > react-reconciler > src > JS ReactFiberBeginWork.js > updateFunctionComponent

```
1023  function updateFunctionComponent(
1024    current: null | Fiber,
1025    workInProgress: Fiber,
1026    Component: any,
1027    nextProps: any,
1028    renderLanes: Lanes,
1029  ) {
1030    let context;
1031    if (!disableLegacyContext) {…
1034    }
1035
1036    let nextChildren;
1037    let hasId;
1038    prepareToReadContext(workInProgress, renderLanes);
1039    if (enableSchedulingProfiler) {…
1041    }
1042    if (__DEV__) {…
1055    } else {
1056      nextChildren = renderWithHooks(…
1063      );
1064      hasId = checkDidRenderIdHook();
1065    }
1066    if (enableSchedulingProfiler) {…
1068    }
1069
1070    if (current !== null && !didReceiveUpdate) {
1071      bailoutHooks(current, workInProgress, renderLanes);
1072      return bailoutOnAlreadyFinishedWork(current, workInProgress, renderLanes);
1073    }
1074
1075    if (getIsHydrating() && hasId) {…
1077    }
1078    // React DevTools reads this flag.
1079    workInProgress.flags |= PerformedWork;
1080    reconcileChildren(current, workInProgress, nextChildren, renderLanes);
1081    return workInProgress.child;
1082  }
```

renderWithHooks

JavaScript

```
export function renderWithHooks<Props, SecondArg>(
  current: Fiber | null,
  workInProgress: Fiber,
  Component: (p: Props, arg: SecondArg) => any,
  props: Props,
  secondArg: SecondArg,
  nextRenderLanes: Lanes,
): any {
  renderLanes = nextRenderLanes;
  currentlyRenderingFiber = workInProgress;

  workInProgress.memoizedState = null;
```

```
  workInProgress.updateQueue = null;
  workInProgress.lanes = NoLanes;

  ReactCurrentDispatcher.current =
    current === null || current.memoizedState === null
      ? HooksDispatcherOnMount
      : HooksDispatcherOnUpdate;

  let children = Component(props, secondArg);

  finishRenderingHooks(current, workInProgress, Component);

  return children;
}

function finishRenderingHooks<Props, SecondArg>(
  current: Fiber | null,
  workInProgress: Fiber,
  Component: (p: Props, arg: SecondArg) => any,
): void {
  ReactCurrentDispatcher.current = ContextOnlyDispatcher;

  renderLanes = NoLanes;
  currentlyRenderingFiber = (null: any);

  currentHook = null;
  workInProgressHook = null;
}
```

函数组件初次挂载

JavaScript

```
const HooksDispatcherOnMount: Dispatcher = {
  readContext,

  use,
  useCallback: mountCallback,
  useContext: readContext,
  useEffect: mountEffect,
  useImperativeHandle: mountImperativeHandle,
```

```
  useLayoutEffect: mountLayoutEffect,
  useInsertionEffect: mountInsertionEffect,
  useMemo: mountMemo,
  useReducer: mountReducer,
  useRef: mountRef,
  useState: mountState,
  useDebugValue: mountDebugValue,
  useDeferredValue: mountDeferredValue,
  useTransition: mountTransition,
  useSyncExternalStore: mountSyncExternalStore,
  useId: mountId,
};
```

函数组件更新阶段

JavaScript

```
const HooksDispatcherOnUpdate: Dispatcher = {
  readContext,

  use,
  useCallback: updateCallback,
  useContext: readContext,
  useEffect: updateEffect,
  useImperativeHandle: updateImperativeHandle,
  useInsertionEffect: updateInsertionEffect,
  useLayoutEffect: updateLayoutEffect,
  useMemo: updateMemo,
  useReducer: updateReducer,
  useRef: updateRef,
  useState: updateState,
  useDebugValue: updateDebugValue,
  useDeferredValue: updateDeferredValue,
  useTransition: updateTransition,
  useSyncExternalStore: updateSyncExternalStore,
  useId: updateId,
};
```

Hook 相关类型定义与初始值

JavaScript

```
export type Update<S, A> = {
  lane: Lane,
  revertLane: Lane,
  action: A,
  hasEagerState: boolean,
  eagerState: S | null,
  next: Update<S, A>,
};

export type UpdateQueue<S, A> = {
  pending: Update<S, A> | null,
  lanes: Lanes,
  dispatch: (A => mixed) | null,
  lastRenderedReducer: ((S, A) => S) | null,
  lastRenderedState: S | null,
};

export type Hook = {
  memoizedState: any,
  baseState: any,
  baseQueue: Update<any, any> | null,
  queue: any,
  next: Hook | null,
};

type Dispatch<A> = A => void;

let currentHook: Hook | null = null;
let workInProgressHook: Hook | null = null;
```