11-4 useReducer 源码解读

资源

1. useReducer

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;
```

函数组件挂载阶段

mountReducer

JavaScript

```javascript
function mountReducer<S, I, A>(
  reducer: (S, A) => S,
  initialArg: I,
  init?: I => S,
): [S, Dispatch<A>] {
  const hook = mountWorkInProgressHook();
  let initialState;
  if (init !== undefined) {
    initialState = init(initialArg);
  } else {
    initialState = ((initialArg: any): S);
  }
  hook.memoizedState = hook.baseState = initialState;
  const queue: UpdateQueue<S, A> = {
    pending: null,
    lanes: NoLanes,
    dispatch: null,
    lastRenderedReducer: reducer,
    lastRenderedState: (initialState: any),
  };
  hook.queue = queue;
  const dispatch: Dispatch<A> = (queue.dispatch = (dispatchReducerActi
    null,
    currentlyRenderingFiber,
    queue,
  ): any));
  return [hook.memoizedState, dispatch];
}
```

mountWorkInProgressHook

TypeScript

```
function mountWorkInProgressHook(): Hook {
  const hook: Hook = {
    memoizedState: null,

    baseState: null,
    baseQueue: null,
    queue: null,

    next: null,
  };

  // 构建单链表
  if (workInProgressHook === null) {
    // 头结点
    currentlyRenderingFiber.memoizedState = workInProgressHook = hook;
  } else {
    // 加入单链表尾部，同时更新workInProgressHook
    workInProgressHook = workInProgressHook.next = hook;
  }
  return workInProgressHook;
}
```

dispatchReducerAction

JavaScript

```
function dispatchReducerAction<S, A>(
  fiber: Fiber,
  queue: UpdateQueue<S, A>,
  action: A,
): void {
  const lane = requestUpdateLane(fiber);
  // ! 1. 创建update
  const update: Update<S, A> = {
    lane,
    revertLane: NoLane,
    action,
    hasEagerState: false,
    eagerState: null,
    next: (null: any),
  };

  if (isRenderPhaseUpdate(fiber)) {
    enqueueRenderPhaseUpdate(queue, update);
  } else {
    // ! 2. 把update暂存到concurrentQueues数组中
    const root = enqueueConcurrentHookUpdate(fiber, queue, update, lan
    if (root !== null) {
      // ! 3. 调度更新
      scheduleUpdateOnFiber(root, fiber, lane);
      entangleTransitionUpdate(root, queue, lane);
    }
  }
}
```

函数组件更新阶段

JavaScript

```
function updateReducer<S, I, A>(
  reducer: (S, A) => S,
  initialArg: I,
  init?: I => S,
): [S, Dispatch<A>] {

  const hook = updateWorkInProgressHook();
  return updateReducerImpl(hook, ((currentHook: any): Hook), reducer);
}
```

updateWorkInProgressHook

JavaScript

```javascript
function updateWorkInProgressHook(): Hook {
  let nextCurrentHook: null | Hook;
  if (currentHook === null) {
    const current = currentlyRenderingFiber.alternate;
    if (current !== null) {
      nextCurrentHook = current.memoizedState;
    } else {
      nextCurrentHook = null;
    }
  } else {
    nextCurrentHook = currentHook.next;
  }

  let nextWorkInProgressHook: null | Hook;
  if (workInProgressHook === null) {
    nextWorkInProgressHook = currentlyRenderingFiber.memoizedState;
  } else {
    nextWorkInProgressHook = workInProgressHook.next;
  }

  if (nextWorkInProgressHook !== null) {
    // There's already a work-in-progress. Reuse it.
    workInProgressHook = nextWorkInProgressHook;
    nextWorkInProgressHook = workInProgressHook.next;

    currentHook = nextCurrentHook;
  } else {
    // ? sy
    // Clone from the current hook.
    currentHook = nextCurrentHook;

    const newHook: Hook = {
      memoizedState: currentHook.memoizedState,

      baseState: currentHook.baseState,
      baseQueue: currentHook.baseQueue,
      queue: currentHook.queue,

      next: null,
    };

    if (workInProgressHook === null) {
      // This is the first hook in the list.
      currentlyRenderingFiber.memoizedState = workInProgressHook = new
    } else {
      // Append to the end of the list.
      workInProgressHook = workInProgressHook.next = newHook;
    }
  }
  return workInProgressHook;
}
```

updateReducer

```typescript
function updateReducer<S, I, A>(
  reducer: (S, A) => S,
  initialArg: I,
  init?: I => S,
): [S, Dispatch<A>] {
  const hook = updateWorkInProgressHook();
  return updateReducerImpl(hook, ((currentHook: any): Hook), reducer);
}
```

updateReducerImpl