12-2 实现 useReducer，掌握 Hooks 的底层结构实现与函数组件的状态

useReducer

TypeScript

```typescript
type Dispatch<A> = (action: A) => void;

export function useReducer<S, I, A>(
  reducer: (state: S, action: A) => S,
  initialArg: I,
  init?: (initialArg: I) => S
) {
  // ! 1.  构建hook链表
  const hook: Hook = updateWorkInProgressHook(); //{ memoizedState: nu

  let initialState: S;
  if (init !== undefined) {
    initialState = init(initialArg);
  } else {
    initialState = initialArg as any;
  }

  // ! 2. 区分函数组件是初次挂载还是更新
  if (!currentlyRenderingFiber!.alternate) {
    // 函数组件初次渲染
    hook.memoizedState = initialState;
  }

  // ! 3. dispatch
  const dispatch: Dispatch<A> = dispatchReducerAction.bind(
    null,
    currentlyRenderingFiber,
    hook,
    reducer
  );

  return [hook.memoizedState, dispatch];
}
```

updateWorkInProgressHook

```typescript
function updateWorkInProgressHook(): Hook {
  let hook: Hook;

  const current = currentlyRenderingFiber.alternate;
  if (current) {
    currentlyRenderingFiber.memoizedState = current.memoizedState;
    if (workInProgressHook) {
      workInProgressHook = hook = workInProgressHook.next;
      currentHook = currentHook.next;
    } else {
      hook = workInProgressHook = currentlyRenderingFiber.memoizedStat
      currentHook = current.memoizedState;
    }
    // 更新
  } else {
    // 初次渲染
    currentHook = null;
    hook = {
      memoizedState: null,
      next: null,
    };

    if (workInProgressHook) {
      workInProgressHook = workInProgressHook.next = hook;
    } else {
      // hook0
      workInProgressHook = currentlyRenderingFiber.memoizedState = hoo
    }
  }
  return hook;
}
```

`dispatchReducerAction`

```typescript
function dispatchReducerAction<S, I, A>(
  fiber: Fiber,
  hook: Hook,
  reducer: (state: S, action: A) => S,
  action: any
) {
  hook.memoizedState = reducer ? reducer(hook.memoizedState, action) :

  const root = getRootForUpdatedFiber(fiber);

  fiber.alternate = { ...fiber };
  if (fiber.sibling) {
    fiber.sibling.alternate = fiber.sibling;
  }

  scheduleUpdateOnFiber(root, fiber);
}
```

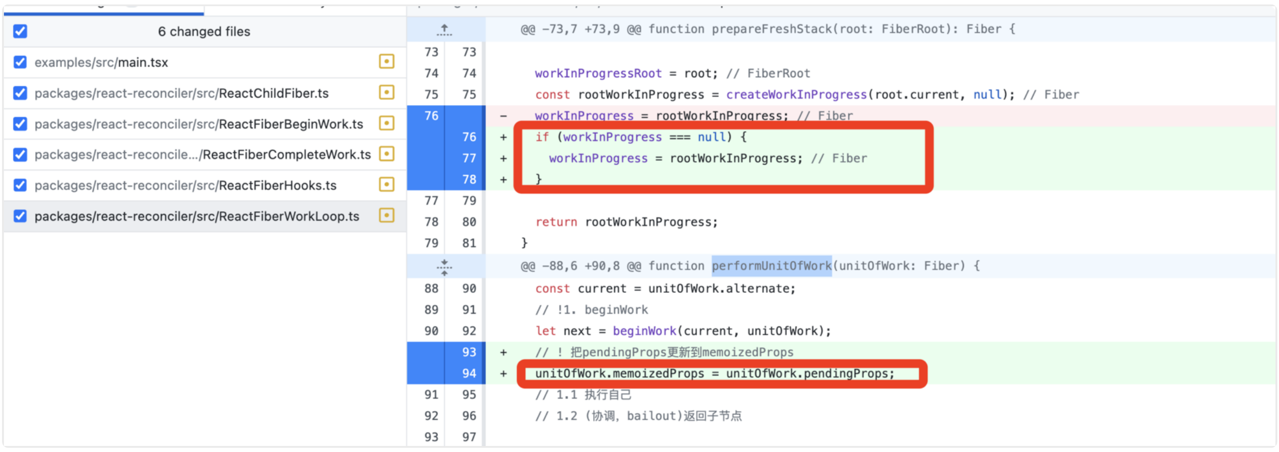
`getRootForUpdatedFiber`

```javascript
// 根据 sourceFiber 找根节点
function getRootForUpdatedFiber(sourceFiber: Fiber): FiberRoot {
  let node = sourceFiber;
  let parent = node.return;

  while (parent !== null) {
    node = parent;
    parent = node.return;
  }

  return node.tag === HostRoot ? node.stateNode : null;
}
```

render 阶段


6 changed files
examples/src/main.tsx
packages/react-reconciler/src/ReactChildFiber.ts
packages/react-reconciler/src/ReactFiberBeginWork.ts
packages/react-reconcile.../ReactFiberCompleteWork.ts
packages/react-reconciler/src/ReactFiberHooks.ts
packages/react-reconciler/src/ReactFiberWorkLoop.ts

```diff
@@ -73,7 +73,9 @@ function prepareFreshStack(root: FiberRoot): Fiber {

   workInProgressRoot = root; // FiberRoot
   const rootWorkInProgress = createWorkInProgress(root.current, null); // Fiber
-  workInProgress = rootWorkInProgress; // Fiber
+  if (workInProgress === null) {
+    workInProgress = rootWorkInProgress; // Fiber
+  }

   return rootWorkInProgress;
 }
@@ -88,6 +90,8 @@ function performUnitOfWork(unitOfWork: Fiber) {
   const current = unitOfWork.alternate;
   // !1. beginWork
   let next = beginWork(current, unitOfWork);
+  // ! 把pendingProps更新到memoizedProps
+  unitOfWork.memoizedProps = unitOfWork.pendingProps;
   // 1.1 执行自己
   // 1.2 (协调, bailout)返回子节点

```

初始化

ReactFiberWorkLoop.ts

```typescript
function prepareFreshStack(root: FiberRoot): Fiber {
  root.finishedWork = null;

  workInProgressRoot = root; // FiberRoot
  const rootWorkInProgress = createWorkInProgress(root.current, null);
  if (workInProgress === null) {
    workInProgress = rootWorkInProgress; // Fiber
  }

  return rootWorkInProgress;
}
```

beginWork 阶段之后

ReactFiberWorkLoop.ts

```TypeScript
function performUnitOfWork(unitOfWork: Fiber) {
  const current = unitOfWork.alternate;
  // !1. beginWork
  let next = beginWork(current, unitOfWork);
  // ! 把pendingProps更新到memoizedProps
  unitOfWork.memoizedProps = unitOfWork.pendingProps;
  // 1.1 执行自己
  // 1.2 (协调，bailout)返回子节点

  if (next === null) {
    // 没有产生新的work
    // !2. completeWork
    completeUnitOfWork(unitOfWork);
  } else {
    workInProgress = next;
  }
}
```

beginWork 阶段

packages/react-reconciler/src/ReactFiberBeginWork.ts

```TypeScript
function updateHostRoot(current: Fiber | null, workInProgress: Fiber)
  const nextChildren = workInProgress.memoizedState.element;

  reconcileChildren(current, workInProgress, nextChildren);
  if (current) {
    current.child = workInProgress.child;
  }

  return workInProgress.child;
}
```

packages/react-reconciler/src/ReactChildFiber.ts

```TypeScript
function useFiber(fiber: Fiber, pendingProps: any): Fiber {
  const clone = createWorkInProgress(fiber, pendingProps);
  clone.index = 0;
  clone.sibling = null;
  return clone;
}
// 协调单个节点，对于页面初次渲染，创建fiber，不涉及对比复用老节点
function reconcileSingleElement(
  returnFiber: Fiber,
  currentFirstChild: Fiber | null,
  element: ReactElement
) {
  const key = element.key;
  let child = currentFirstChild;
  while (child !== null) {
    if (child.key === key) {
      const elementType = element.type;

      if (child.elementType === elementType) {
        // deleteRemainingChildren(returnFiber, child.sibling);
        const existing = useFiber(child, element.props);
        existing.return = returnFiber;
        return existing;
      }
      // 不匹配
      // deleteRemainingChildren(returnFiber, child);
      break;
    } else {
      // deleteChild(returnFiber, child);
    }
    child = child.sibling;
  }
  const created = createFiberFromElement(element);
  created.return = returnFiber;
  return created;
}
```

completeWork

packages/react-reconciler/src/ReactFiberCompleteWork.ts

```TypeScript
export function completeWork(
  current: Fiber | null,
  workInProgress: Fiber
): Fiber | null {
  const newProps = workInProgress.pendingProps;
  switch (workInProgress.tag) {
    case Fragment:
    case ClassComponent:
    case FunctionComponent:
    case HostRoot: {
      return null;
    }
    case HostComponent: {
      // 原生标签,type是标签名
      const { type } = workInProgress;
      if (current !== null && workInProgress.stateNode != null) {
        updateHostComponent(current, workInProgress, type, newProps);
      } else {
        // 1. 创建真实DOM
        const instance = document.createElement(type);
        // 2. 初始化DOM属性
        finalizeInitialChildren(instance, null, newProps);
        // 3. 把子dom挂载到父dom上
        appendAllChildren(instance, workInProgress);
        workInProgress.stateNode = instance;
      }
      return null;
    }
```

```
      case HostText: {
        workInProgress.stateNode = document.createTextNode(newProps);
        return null;
      }
      // todo
    }

    throw new Error(
      `Unknown unit of work tag (${workInProgress.tag}). This error is l
        "React. Please file an issue."
    );
  }

  function updateHostComponent(
    current: Fiber,
    workInProgress: Fiber,
    type: string,
    newProps: any
  ) {
    if (current.memoizedProps === newProps) {
      return;
    }
    finalizeInitialChildren(
      workInProgress.stateNode as Element,
      current.memoizedProps,
      newProps
    );
  }
  // 初始化属性
  function finalizeInitialChildren(
    domElement: Element,
    prevProps: any,
    nextProps: any
  ) {
    for (const propKey in prevProps) {
      const prevProp = prevProps[propKey];
      if (propKey === "children") {
        if (isStr(prevProp) || isNum(prevProp)) {
          // 属性
          domElement.textContent = "";
        }
      } else {
        // 3. 设置属性
        if (propKey === "onClick") {
          domElement.removeEventListener("click", prevProp);
        } else {
          if (!(prevProp in nextProps)) {
            (domElement as any)[propKey] = "";
          }
        }
      }
    }

    for (const propKey in nextProps) {
      const nextProp = nextProps[propKey];
      if (propKey === "children") {
        if (isStr(nextProp) || isNum(nextProp)) {
          // 属性
          domElement.textContent = nextProp + "";
        }
      } else {
        // 3. 设置属性
        if (propKey === "onClick") {
          domElement.addEventListener("click", nextProp);
        } else {
          (domElement as any)[propKey] = nextProp;
        }
      }
    }
  }
```