[React 源码] useRef [1.4k 字 - 阅读时长3min]

> 代码都来自 React 18 源码， 大家可以放心食用

读完收获

ref 怎么实现获取 Dom 引用的?

怎么实现 useRef 可以得到一个可变且不刷新页面的值?

如果 ref.current 改变后，会刷新页面，如果你是 React 开发者，如何设计?

useRef 使用

useRef ref 值作为 DOM 引用

案例 1：使用 ref focus input

```javascript
function App() {
  return <Counter label="Label" value="Value" isFocus />;
}

function Counter({ label, value, isFocus }) {
  const ref = React.useRef(); // (1)

  React.useEffect(() => {
    if (isFocus) {
      ref.current.focus(); // (3)
    }
  }, [isFocus]);

  return (
    <label>
      {/* (2) */}
      {label}: <input type="text" value={value} ref={ref} />
    </label>
```

```
  );
}
```

useRef 作为可变值：

案例 2： 判断组件是首次渲染还是重新渲染

javascript 复制代码

```
function Counter() {
  const [count, setCount] = React.useState(0);

  function onClick() {
    setCount(count + 1);
  }

  const isFirstRender = React.useRef(true);

  React.useEffect(() => {
    if (isFirstRender.current) {
      isFirstRender.current = false;
    } else {
      console.log(
        `
          I am a useEffect hook's logic
          which runs for a component's
          re-render.
        `
      );
    }
  });

  return (
    <div>
      <p>{count}</p>

      <button type="button" onClick={onClick}>
        Increase
      </button>
    </div>
  );
}
```

复制代码

useRef 原理

> 这里，我们直接进入到 reconciler 阶段，默认已经通过深度优先调度到了 Counter 函数组件的 Fiber节点

---

useRef mount 挂载阶段

第一：判断是函数节点的 tag 之后，调用 renderWithHooks.

```javascript
/*
workInProgress: 当前工作的 Fiber 节点
Componet: Counter 函数组件
_current: 老 Fiber 节点 也就是 workInProgress.alternate
*/
let value = renderWithHooks(_current,workInProgress,Component);
```

第二：在 renderWithHooks 当中调用 Counter 函数

```javascript
let children = Component();
```

第三： 调用 Counter 函数 的 useRef 函数

```javascript
export function useRef(reducer, initialArg) {
  return ReactCurrentDispatcher.current.useRef(reducer, initialArg);
}
```

第四：挂载阶段 `ReactCurrentDispatcher.current.useRef` 实则是调用了 `mountRef`，

```javascript
function mountRef<T>(initialValue: T): { current: T } {
  const hook = mountWorkInProgressHook();
  const ref = { current: initialValue };
  hook.memoizedState = ref;
  return ref;
}
```

第五：在 `mountRef` 中调用， `mountWorkInProgressHook` 函数，创建 `useRef` 的 `Hook` 对象，构建 `fiber.memoizedState` 也就是 Hook 链表, 创建 ref 对象，里面有 current 属性， `useRef`

的 `Hook.memoizedState` 属性 就是 该 `ref` 对象。 返回 `ref` 对象。组件当中可以通过 `ref.current` 拿到该初始值。

第六：reconciler 阶段执行之后,来到 render 阶段 中的 layout 阶段，在 `commitLayoutEffectOnFiber` 当中执行 `safelyAttachRef` 再执行 `commitAttachRef`

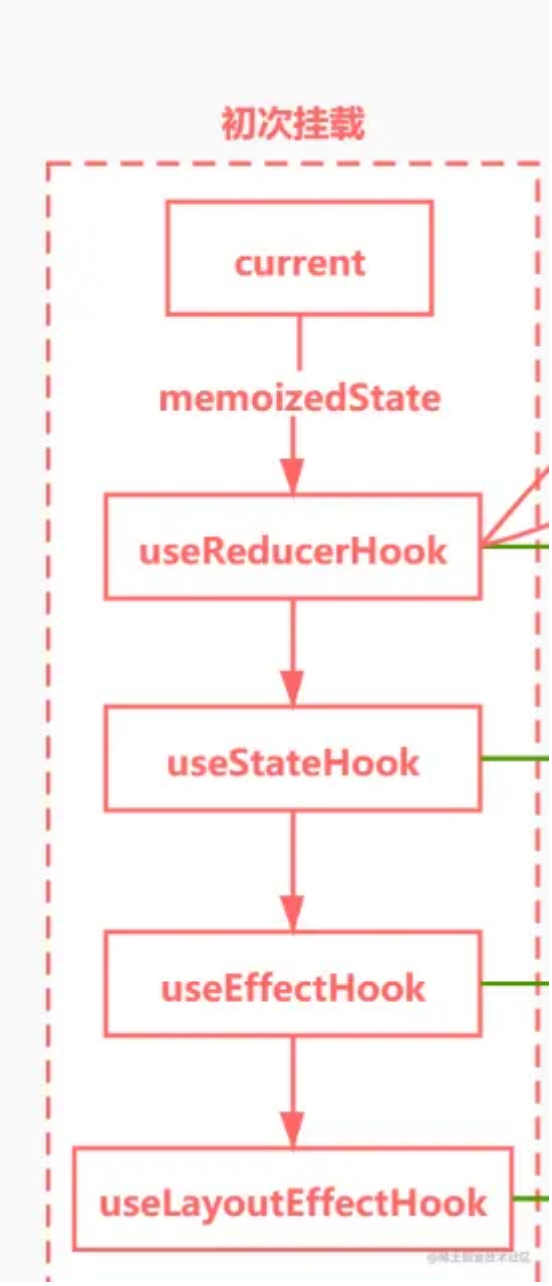
```javascript 复制代码
function commitLayoutEffectOnFiber(
  finishedRoot: FiberRoot,
  current: Fiber | null,
  finishedWork: Fiber,
  committedLanes: Lanes,
): void {
     safelyAttachRef(finishedWork, finishedWork.return);
}
```

第七： `commitAttachRef` 判断 ref 挂载的类型， 原生节点，通过 fiber 节点 的 stateNode 属性获取到原生节点，将 原生 DOM 节点 赋值给 `ref.current` 。

> 通过第七点可以解决问题1：如何绑定 Dom 节点?
>
> render 阶段中的 mutation 阶段 Dom 已经挂载了， layout 阶段可以 通过 Fiebr.stateNode 获取到 新的 原生 Dom 节点，useEffect 是在 render 阶段之后执行的，所以 这时候通过 ref.current 可以 获取到 原生 Dom 节点。

```javascript 复制代码
function commitAttachRef(finishedWork: Fiber) {
  const ref = finishedWork.ref;
  if (ref !== null) {
    const instance = finishedWork.stateNode;
    let instanceToUse;
    switch (finishedWork.tag) {
      case HostResource:
      case HostSingleton:
      case HostComponent:
        instanceToUse = getPublicInstance(instance);
        break;
      default:
        instanceToUse = instance;
    }
     ref.current = instanceToUse;
  }
}
```

如果之后再有 useState useReducer，最终mout阶段的成果是



自此 useRef 挂载阶段执行完毕

---

useRef update 更新阶段

第一：我们直接进入到 `reconciler` 阶段，默认已经通过深度优先更新调度到了 `Counter` 函数组件的 `Fiber` 节点

第二：判断是函数节点的 tag 之后，调用 renderWithHooks.

```javascript 复制代码
/*
workInProgress: 当前工作的 Fiber 节点
Componet: Counter 函数组件
_current: 老 Fiber 节点 也就是 workInProgress.alternate
*/
let value = renderWithHooks(_current,workInProgress,Component);
```

第三：在 renderWithHooks 当中调用 Counter 函数

```javascript 复制代码
let children = Component();
```

第四： 调用 Counter 函数 的 useRef 函数

```javascript 复制代码
export function useRef(reducer, initialArg) {
  return ReactCurrentDispatcher.current.useRef(reducer, initialArg);
}
```

第五: 更新阶段 `ReactCurrentDispatcher.current.useReducer` 实则是调用了 `updateRef`

```javascript 复制代码
function updateRef<T>(initialValue: T): {current: T} {
  const hook = updateWorkInProgressHook();
  return hook.memoizedState;
}
```

第六：在 `updateRef` 中调用 `updateWorkInProgressHook` 函数，在此函数中最重要的就是通过 alternate 指针复用 `currentFiber` （老 Fiber） 的 memorizedState, 也就是 Hook 链表，并且按照严格的对应顺序来复用 `currentFiber` （老 Fiber） Hook 链表当中的 Hook（通过 currentHook 指针结合链表来实现），通过尽可能的复用来创建新的 Hook 对象，构建 `fiber.memoizedState` 也就是新的 Hook 链表。

第七： `updateRef` 直接将 hook.memoizedState 返回，也就是 ref 对象 `ref = {current: }`。

> 通过第七点可以解决问题2：useRef 如何作为可变值?
>
> 组件当中 直接通过 `ref.current = "true"` 修改 ref 对象的 current 属性，current 通过对象引用的修改， 已经最新值了, 但是 React 没有像 useReducer 中 dispatch 的时候去 调用 scheduleFiberOnRoot 从根节点开始遍历，刷新页面，所以 ref.current 的值

> 在 UI 当中不会刷新，直至下一次 dispatch 或者 setState 才会刷新。因为函数重新执行了，又调用了 useRef -> updateRef 拿到最新值。

如果我们想让 `ref.current` 修改的时候页面刷新，该怎么办，笔者的想法是 通过 `Object.definedProperty` 中定义 `settter getter` ，修改的时候 调用 setter 函数， 在 setter 函数中 调用 `scheduleFiebrOnRoot` 再次从 根节点开始深度遍历，重新经过 scheduler reconciler render 三个大阶段就可以实现刷新页面。

自此 useRef 更新完毕
