

React父组件刷新导致子组件重新渲染问题



TORyTANG

27 • 1 • 2 • 9

发布于

2021-10-05

我在研究react时，碰到一个关于父组件重新渲染导致子组件重新渲染的问题：

当前代码结构如下：

```
function Clicker({ children }) {
  const [count, setCount] = useState(0);

  return (
    <div className="clicker">
      <h2>You clicked {count} times!</h2>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      {children}
    </div>
  );
}

function ComponentToRender() {
  const count = React.useRef(0);

  React.useEffect(() => {
    console.log("component rendered", count.current++);
  });

  return (
    <div className="ComponentToRender">
      <p>sub</p>
    </div>
  );
}
```

当点击按钮时，count增加，<Clicker />组件重新渲染，同时子组件ComponentToRender并不会重新渲染；

但是，当我把代码结构改成如下这样：

```
return (
  <div>
    <h2>You clicked {count} times!</h2>
    <button onClick={() => setCount(count + 1)}>Increment</button>
    <ComponentToRender />
  </div>
);
}

function ComponentToRender() {
  const count = React.useRef(0);

  React.useEffect(() => {
    console.log("component rendered", count.current++);
  });

  return (
    <div className="ComponentToRender">
      <p>sub</p>
    </div>
  );
}
```

此时，count增加，<App />组件state发生变化导致重新渲染，同时子组件ComponentToRender会重新渲染；

既然都是父组件状态发生变化后重新渲染，子组件也都没有使用React.memo，所以子组件本应该重新渲染。但是第一种情况<ComponentToRender />却没有重新渲染呢？

源代码地址

关注 4

收藏 4

赞 3



回复 · 阅读 11.6k

3 个回答

得票

最新



jsdeferred

3.1k • 2 • 14 • 16

发布于

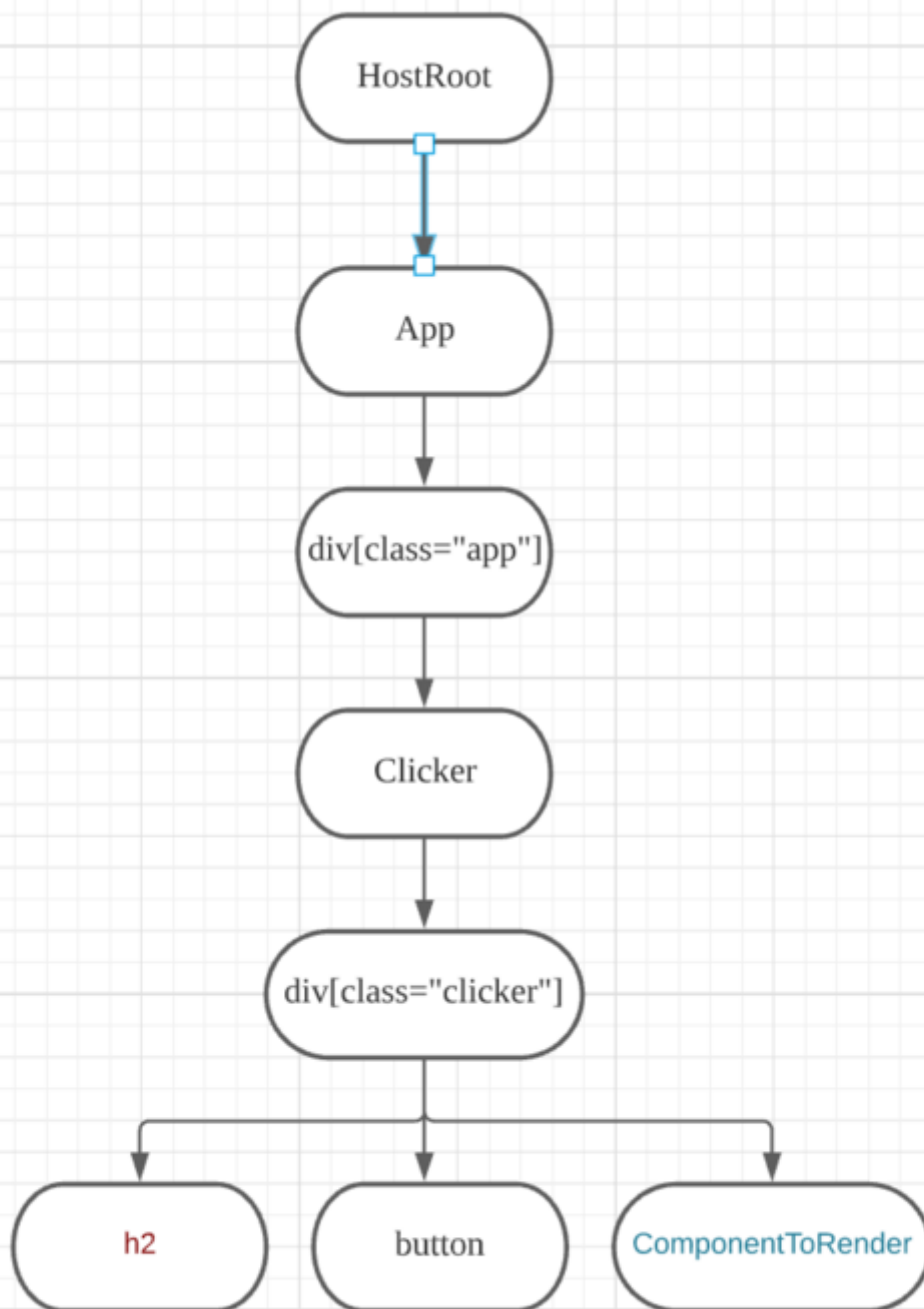
2021-10-08

更新于

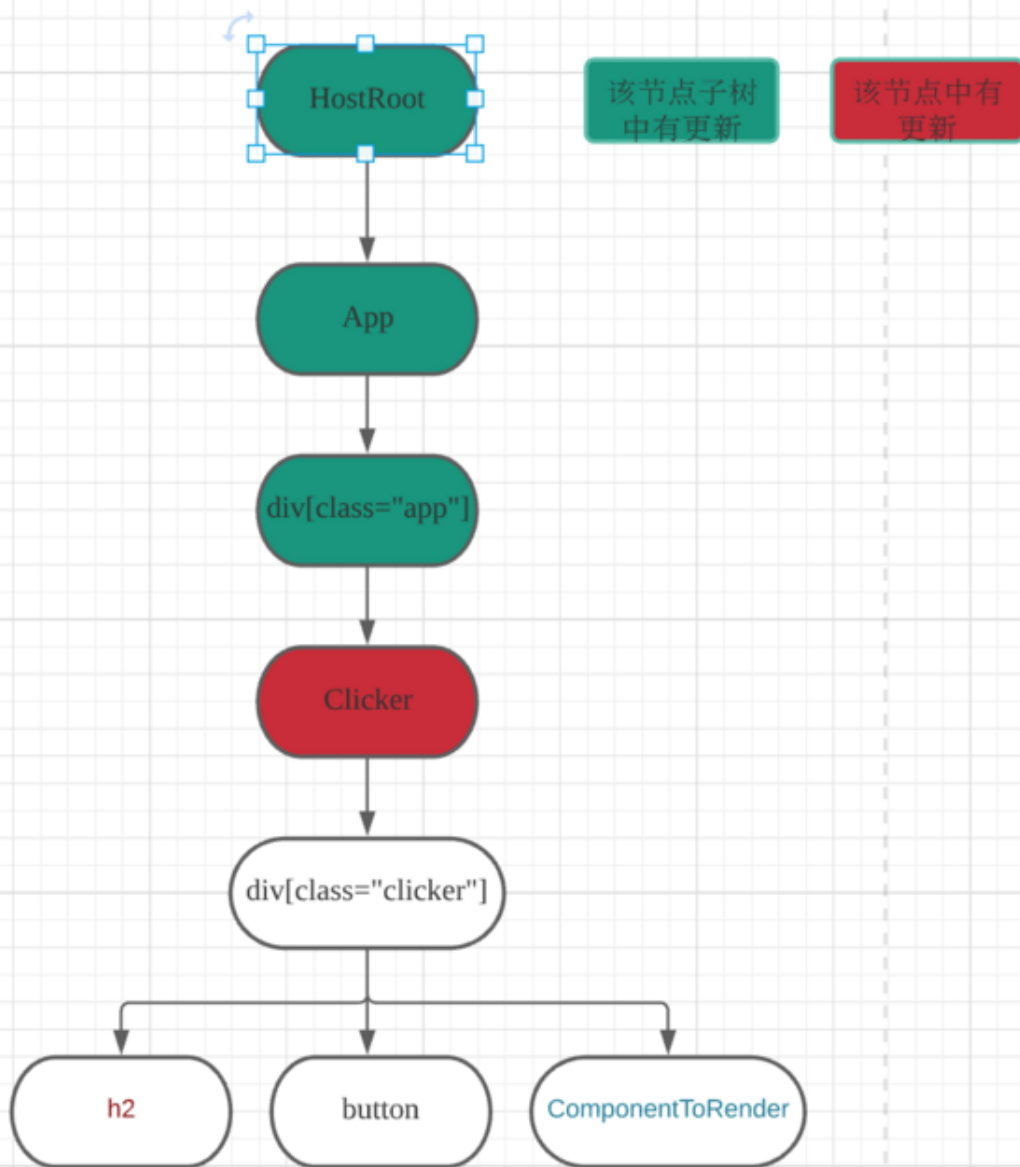
2021-10-08

✓ 已被采纳

要想知道这个问题的原因, 我们得知道一些`react`的原理, 在`react`中在每次更新产生后都会进行一个叫做`reconciliation(协调)`的过程, 在其中`react`中会找到此次更新中`fiber`树发生的变化, 并将他们打上一些标记, 在下一个阶段中在根据这些标记对该`fiber`节点进行一些操作, 比如当一个`fiber`节点被打上`Placement`标记就说明待会该`fiber`节点对应的`dom`节点需要被插入`dom`树, 理所当然的如果我们想要每次更新都能迅速的完成那么我们肯定就不能傻乎乎的去比较整颗`fiber`树, 这是一个非常耗时的操作, 所以`react`实际在进行`reconciliation`的过程中使用了一种启发式的`diff`算法, 这种算法能根据现有信息提前将一些压根没有更新的`fiber`子树的`reconciliation`过程省去, 以免白白的花时间做无用工, 接下来我们会简单的介绍以下这个算法, 根据代码我们可以构建出这样的一颗`fiber`树(省略了一些底层节点)



在`setState`后, `react`会将该更新从产生更新的节点上向上冒泡并一路为他的父节点打上需要更新



在图中我们把有更新的节点标记为红色，子树中存在更新的标记为绿色，由于Clicker都是HostRoot, App, div[class="app"]的节点他所以他们都需要被标记为绿色

标记完成后，react就会开始从HostRoot开始reconciliation的过程，在其中会更具该节点更新前后的props是否严格相等(Object.is)和是否被标记了更新决定是否继续进行他子树的reconciliation过程,HostRoot是个特例他是React中的一个抽象根节点，他不对应任何组件和dom节点他的props也永远都是为null，所以再进行他的diff时他的前后props永远是相等的，我们可以把reconciliation的过程中的情况分为三种：

(1)一个节点更新前后props相等，且该节点没有更新，并且其子树中也没有更新，这种情况下以该节点为根的fiber子树的reconciliation过程就会到此为止停止diff

(2)一个节点更新前后props不相等，则继续他子节点的diff过程

(3)一个节点更新前后props相等，且他自身没有更新，但是他的子树中存在更新，这种情况要复

props, 所以更新前后复用节点的props总是严格相等的, 光说可能还有点不太直观让我们考虑以下代码

```
        onClick={() => {
          setCount(count + 1);
        }}
      >
        increment
      </button>
    </div>
  );
};

const App = () => {
  return (
    <div id="container">
      <div
        style={{
          background: "red"
        }}
        id="static"
      >
        Static Node
      <div>Static Node</div>
    </div>
    <TriggerUpdate />
  </div>
  );
};
```

下面会使用jquery选择器的方式指明我们说的时哪个fiber节点比如\$('#container')就表示哪个id为container的div所对应的fiber, 虽然App中的TriggerUpdate会触发更新但是他冒泡的更新标记并不会影响到并不在他冒泡路径上和他同级的\$('#static')节点所以他自己和他子树中都是没有更新标记的, 因为也知道他的父级节点\$('#container') div也没有更新, 所以在创建\$('#static') div对于的fiber节点时复用前一次的props, 当接下来进行\$('#static') diff时, 由于前后props没变且它自身和子树中不包含更新, 也就是上面说的第一种情况他的diff过程就会终止, 即使此次更新中\$('#static')对应的jsx对象的style属性是全新的对象, 对于一些不会产生更新的节点react在运行时就会将他识别出来, 以减少diff的工作量

问题逻辑梳理

知道上面的工作原理后, 我们再来看看我们这颗fiber树的diff过程, 不难看出直到Clicker进行diff是才会重新调用创建组件创建JSX元素应为他上层的节点都满足第三种情况, 但是我们可以

Clickerprops的children都是严格相等的所以当进行ComponentToRender的diff时就会发现它属于第一种更新情况，直接结束他的diff,而你的问题的第二种情况中可就没那么幸运了，由于ComponentToRender的JSX元素在App中被重新创建，所以他的props就不相等了，所以属于第二种更新情况继续他子树的diff所以ComponentToRender也会被调用所以会看到useEffect中的打印,在这种情况下，如果ComponentToRender的执行代价很高的话，就可以将这个组件包裹在memo中这时候比较他更新前后的props是否变更，就会转换为对其中的每个属性进行浅比较，而不是直接判断严格相等，这样也会停止ComponentToRender的diff

备注

下面是对上面用到概念的一些解释

(1)fiber节点:

每一个组件或者dom都对应一个fiber节点，用他们组成的树也就是我们平时说的虚拟dom树，fiber节点由JSX元素中的信息创建而来,比如<div id="3" foo={4} />对于的jsx元素和fiber节点就是下面这样的

jsx表达式

```
<div id="3" foo={4} />
```

jsx表达式创建出来的jsx元素

```
{
  "type": "div",
  "key": null,
  "ref": null,
  "props": {
    "id": "3",
    "foo": 4
  }
}
```

由jsx元素的信息创建出来的fiber节点的结构(不完整，只显示了部分属性),其中type中存了'div',节点是否更新和子树中是否有更新则分别存储在了lanes和childLanes中

```
type Fiber = {
  /**
```

```
index: number
/**
 * 此次commit中需要删除的fiber节点
 */
deletions: Fiber[] | null
/**
 * 子树带有的更新操作，用于减少查找fiber树上更新的时间复杂度
 */
subtreeFlags: Flags
/**
```

[注册登录](#)

```
/**
 * 新创建jsx对象的第二个参数,像HostRoot这种内部自己创建的Fiber节点为null
 */
pendingProps: any
/**
 * 上一轮更新完成后的props
 */
memoizedProps: any
```

(1)reconciliation:

你可以把 reconciliation和平时说的diff理解成一个意思

更多

- (1) 如果你看到这里还不明白的话可以去看下这个问题,[我用几十行代码实现了和react启发式diff思路相同的关键代码](#)
- (2) [如果想了解更多启发式算法内容可以查看](#)
- (3) [如果想了解更多react原理，可以了解我的项目](#)

赞 8

回复

jsdeferred: @array_huang 如果把像上面那样的把fiber树画出来，第二种情况下setState在App中调用那么App组件就会被染成红色，那他必定会被重新调用

1 · 回复 · 2021-10-12

TORYTANG: 超级感谢你的回复!!! 之前我在stackoverflow上问了这个问题，有回答说是得去看下fiber的工作原理，然后我看了一些关于fiber原理的文章，还是一知半解的。但是看了你的回答之后感觉清晰了很多，谢谢你!!! 大佬的其他文章感觉也非常值得我拜读了!



array_huang：“而你的问题的第二种情况中可就没那么幸运了，由于ComponentToRender的JSX元素在App中被重新创建，所以他的props就不相等了”，请问为什么ComponentToRender的JSX元素会被重新创建呢？

 · 回复 · 2021-10-11

共 4 条评论

[查看全部 3 个回答](#)

被 1 篇内容引用

 SegmentFault 思否 2021 年度 Top Writer  16

推荐问题


树形数据前端生成还是后台生成返回比较好？

树形数据前端生成还是后台生成返回比较好？

8 回答 · 2.3k 阅读  已解决

请教一个算法问题，把数字转换为A,B,C.....，如何实现？

现在我想根据数字转换为英文字母：比如 A->1,B->2,.....AA->27,AB->28。我的思路本来是想维...

5 回答 · 2.2k 阅读  已解决


JavaScript 这段判空可以优化写法吗？

{代码...}

11 回答 · 2.2k 阅读

react中,click事件的源码怎么查看？

背景:开发模式与生产模式下都开启了source-map,但是用chrome开发工具查看click事件的源码时,发现并不能...

3 回答 · 921 阅读  已解决

前端如何无感自动刷新token？

最近接到一个新需求，就是登录后进入到大屏，大屏是需要一直挂着展示，不要让它频繁掉线重新登录，方案...

7 回答 · 1.9k 阅读

11 回答 · 1.3k 阅读

北大官网首页动态图标怎么做出来？

[链接]

4 回答 · 789 阅读 ✓ 已解决