30 通过实例分析应用程序更新渲染时的副作用列表

更新时间：2020-10-14 10:26:45

> 富贵必从勤苦得。——杜甫

前言

上一节介绍了 React Fiber 架构下的「协调」算法与 diff 算法的核心逻辑，本节我们将以具体的场景来分析应用程序执行完协调逻辑后收集到的副作用列表（Effect List）。

应用程序首次渲染时的副作用列表

本节使用的 demo 程序见代码示例 6.4.1。

```
class UpdateCounter extends Component{
 constructor(props) {
  super(props);
  this.state = {
   count: 0,
   text: '点击计数'
  };
  this.handleClick = this.handleClick.bind(this);
 }

  handleClick() {
    this.setState({
   count: this.state.count + 1
  });
  }

  render() {
  return (
   <div className="wrap-box">
    <button key="1" onClick={this.handleClick}>{this.state.text}</button>
    <span key="2" id="spanText" className="span-text">{this.state.count}</span>
   </div>
  )
 }
}
```

代码示例 6.4.1 点击计数组件

代码示例 6.4.1 实现了一个点击计数的组件，该组件可以统计按钮被点击的次数。下图 6.4.1 是该应用程序首次渲染时 render 阶段执行结束后 fiberRoot 对象的结构。

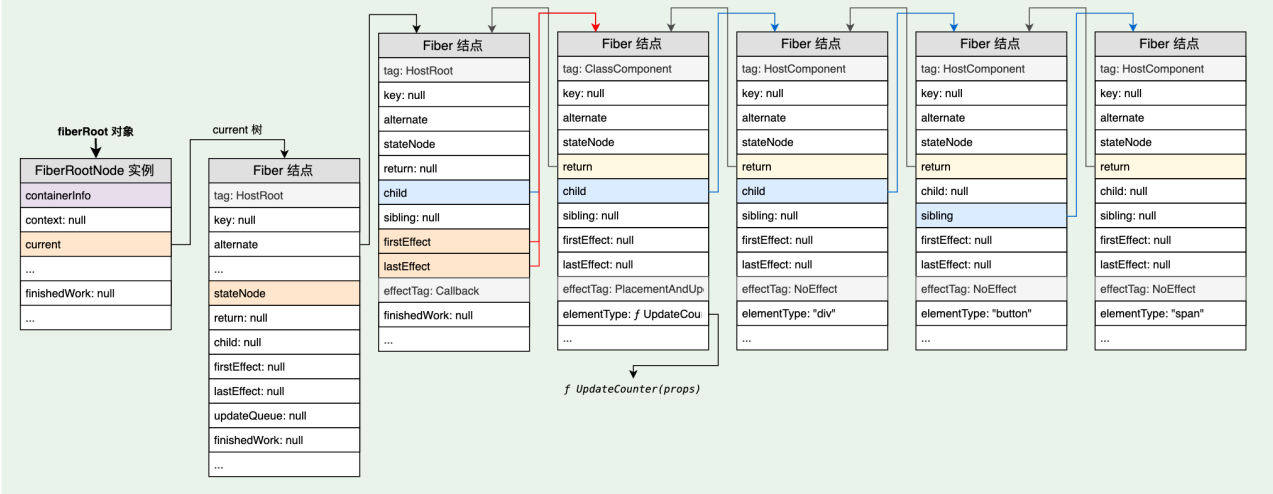

图 6.4.1 应用程序首次渲染 render 阶段完成后的 fiberRoot 对象结构

render 阶段完成后，workInProgress 树已经构建完成并且副作用列表（Effect List）也收集完成。应用程序首次渲染时的副作用列表实际上只有一个 Fiber 结点，这个结点就是应用程序根组件元素对应的结点（该程序中是 Update Counter 对应的结点）。

如果我们触发了上面 demo 中的按钮点击事件，则该应用程序就会进入更新渲染过程，那么其更新渲染收集到的副作用列表是什么样的呢？

应用程序更新渲染时的副作用列表

Case1 — 只涉及元素更新的情况

当我们触发了代码示例 6.4.1 中的按钮 click 事件后，应用成开始执行更新渲染流程，render 阶段执行结束后 fiberRoot 对象的结构见下图 6.4.2。

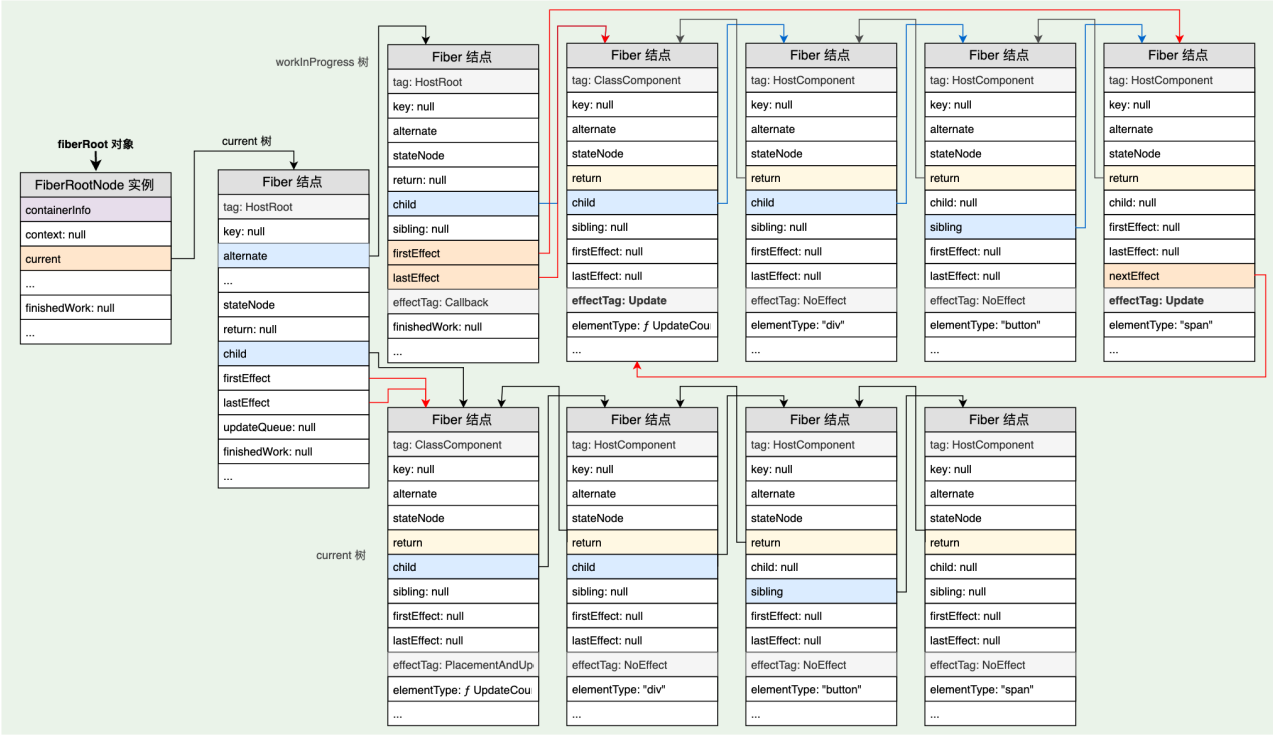

图 6.4.2 应用程序更新渲染 render 阶段完成后的 fiberRoot 对象结构 — 只涉及元素更新的情况

此时的 fiberRoot 对象上面同时存在两颗 Fiber 树，分别是 fiberRoot.current 指向的 current 树和 fiberRoot.current.alternate 指向的 workInProgress 树。current 树上面保留了应用程序上一次渲染完成后的状态，workInProgress 树是当前更新过程中产生的临时 Fiber 树。

在应用程序更新渲染时创建的 workInProgress 树上面，React 对每一个需要更新的结点标记了对应的 tag。比如图 6.4.2 中的 elementType 为 span 和 UpdateCounter 的结点。副作用列表中分别使用 firstEffect 和 lastEffect 指向了链表中的第一个和最后一个需要更新的结点，每个被标记了副作用的结点之间使用 nextEffect 连接。

如果通过应用程序更新过程中产生新的结点时，副作用列表会是什么样的呢？

Case2 — 有新增元素的情况

在很多场景中，执行事件处理程序后会产生新的元素，如代码示例 6.4.2。

```
handleClick() {
  this.setState({
  count: this.state.count + 1
 });
 }

 render() {
 return (
  <div className="wrap-box">
    {
   this.state.count > 0 && (
    <div className="title-text">{this.state.text}</div>
   )
  }
   <button key="1" onClick={this.handleClick}>{this.state.text}</button>
   <span key="2" id="spanText" className="span-text">{this.state.count}</span>
  </div>
 )
}
```

代码示例 6.4.2 点击按钮控制元素显示逻辑

代码示例 6.4.2 实现的逻辑就是，点击按钮后更新 span 元素的同时也会产生新的 div 元素。应用程序 render 阶段执行结束后，fiberRoot 对象的内部结构见下图 6.4.3。

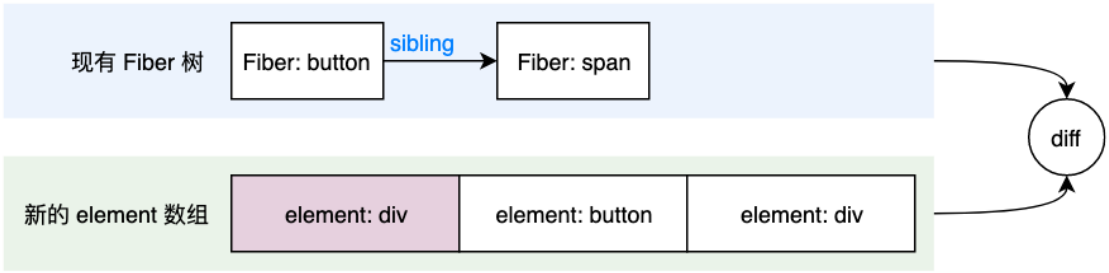

图 6.4.3 应用程序更新渲染 render 阶段完成后的 fiberRoot 对象结构 — 有新增元素的情况

当程序中通过执行 setState(...) 操作后有新的 React 元素产生时，React 在构建 workInProgress 树的过程中通过 diff 算法会检测到为该新增的元素，然后为该元素创建对应的 Fiber 结点并将其副作用类型标记为 Placement。在完成工作单元收集副作用时，该（新增的）Fiber 结点将被收集到副作用列表中。

Case3 — 有删除元素的情况

将代码示例 6.4.2 中的控制元素显示的条件改为 this.state.count === 0 后，点击按钮就使得该 div 元素被去掉。这个过程 React 的工作机制与新增元素的工作机制基本相同，不同的地方主要就是将对应 Fiber 结点的副作用标记为 Deletion。

小结

本节以具体实例分析了应用程序在首次渲染、更新渲染没有新增和删除元素的情况以及更新渲染有新增元素的情况下收集到副作用列表后 fiberRoot 对象在内存中的具体形态。有兴趣的同学可以通过在浏览器打断点查看 fiberRoot 对象的详细结构。

下一节将会介绍 React 如何将更新渲染时的副作用列表更新到屏幕。

}

29 React v16 版的协调算法是什么样的呢？

31 React 应用程序更新渲染时如何 commit 副作用