14 深入理解React Fiber 架构的两个核心构造函数FiberRootNode和FiberNode

更新时间：2020-08-26 14:28:49

> 知识犹如人体的血液一样宝贵。——高士其

前言

前面介绍过，React 应用程序运行在 prerende r阶段的主要工作是构建 fiberRoot 对象。在这个阶段会使用 FiberRootNode 构造函数实例化 fiberRoot 对象，使用 FiberNode 构造函数实例化 Fiber 树的第一个结点对象。本文将会详细介绍 FiberRootNode 和 FiberNode 两个构造函数中的一些重要属性。

FiberRootNode 和 FiberNode 两个构造函数的实例关系见图 3.4.1。

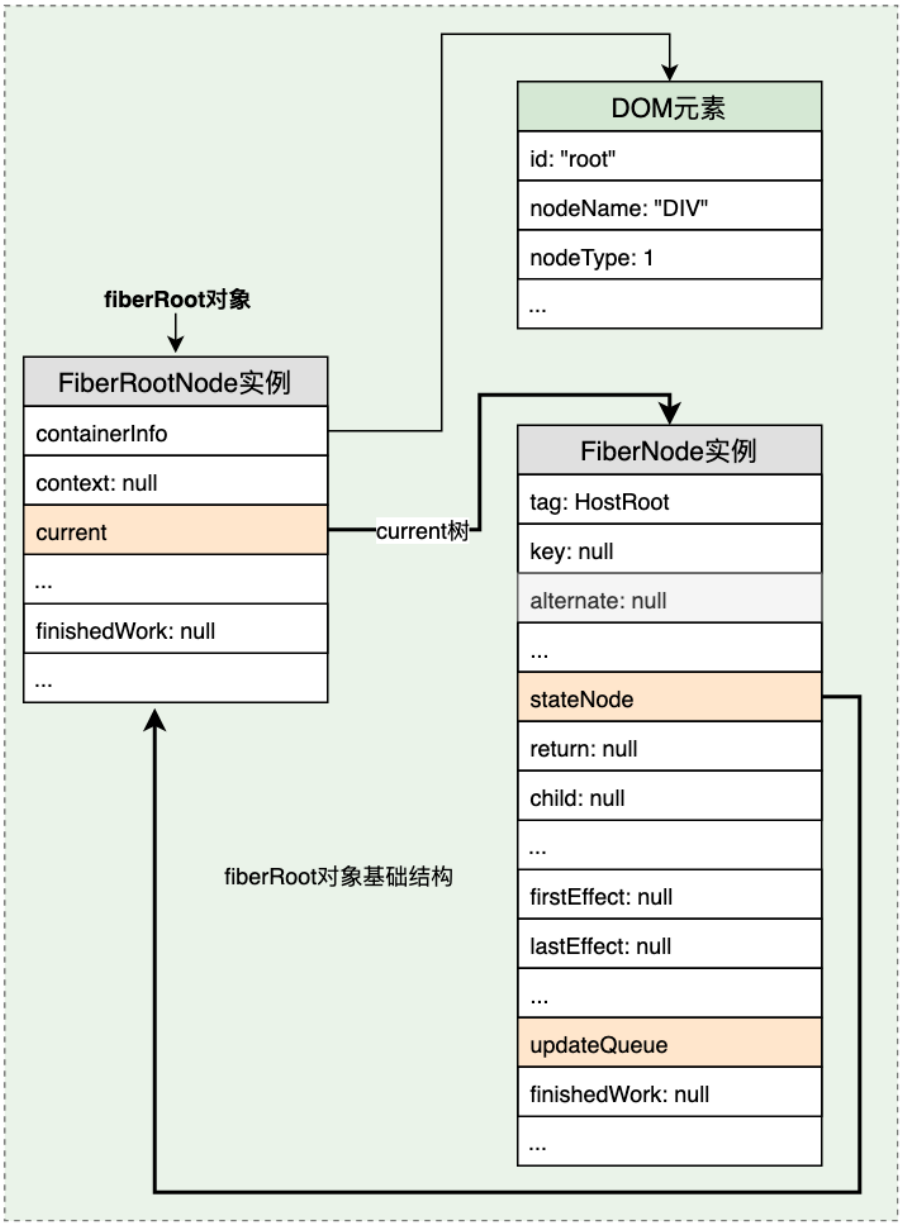

图 3.4.1 FiberRootNode 和 FiberNode 两个构造函数实例之间的结构关系

FiberRootNode 构造函数用于生成 fiberRoot 对象

React 对 FiberRootNode 构造函数的源码见代码示例 3.4.1。

```
// 源码位置：packages/react-reconciler/src/ReactFiberRoot.js
function FiberRootNode(containerInfo, tag, hydrate) {
  // 当前 Fiber 结点的类型
  this.tag = tag;
  // current树，见备注 3-4-1.1
  this.current = null;
  // 包含容器
  this.containerInfo = containerInfo;
  this.pendingChildren = null;
  this.pingCache = null;
  this.finishedExpirationTime = NoWork;
  // 存储工作循环（workLoop）结束后的副作用列表，用于commit阶段
  this.finishedWork = null;
  this.timeoutHandle = noTimeout;
  this.context = null;
  this.pendingContext = null;
  this.hydrate = hydrate;
  this.firstBatch = null;
  ...
}
```

代码示例 3.4.1 FiberRootNode 构造函数的源码定义

备注 3-4-1.1 current 属性

React 应用程序首次渲染时在 prerender 阶段会初始化「**current 树**」（本质上也是对象哦）。最开始的 current 树只有一个根结点—HostRoot 类型的 Fiber 结点。在后面的 render 阶段会根据此时的 current 树创建「**workInProgress 树**」（同样是对象哦，每个结点都是 FiberNode 的实例）。在 workInProgress 树上面进行一系列运算（计算更新等），最后将副作用列表（Effect List）传入到 commit 阶段。当 commit 阶段运行完成后将当前的 current 树替换为 workInProgress 树，至此一个更新流程就完成了。这个过程简化描述就是：

1. 在 render 阶段 React 依赖 current 树通过工作循环（workLoop）构建 workInProgress 树；
2. 在 workInProgress 树进行一些更新计算，得到副作用列表（Effect List）；
3. 在 commit 阶段将副作用列表渲染到页面后，将 current 树替换为 workInProgress 树（执行 current = workInProgress ）。

current 树是未更新前应用程序对应的 Fiber 树，workInProgress 树是需要更新屏幕的 Fiber 树。

FiberNode 构造函数用于生成 Fiber 结点

React 对 FiberNode 构造函数源码见代码示例 3.4.2。

```
// 源码位置：packages/react-reconciler/src/ReactFiber.js
function FiberNode() {
  // tag见备注3-4-2.1
  this.tag = tag;
  this.key = key;
  // 元素类型
  this.elementType = null;
  this.type = null;
  // stateNode见备注3-4-2.2
  this.stateNode = null;

  // return，child，sibling见备注3-4-2.3
  this.return = null;
  this.child = null;
  this.sibling = null;
  this.index = 0;

  this.ref = null;
 // 需要更新的props
  this.pendingProps = pendingProps;
  // 当前（未更新前）的props
  this.memoizedProps = null;
  // 更新队列
  this.updateQueue = null;
  // 当前（未更新前）的state
  this.memoizedState = null;
  this.dependencies = null;
  // 普通模式，严格模式，并发模式等
  this.mode = mode;

  // 副作用类型（指插入、更新、插入并更新、删除等）
  this.effectTag = NoEffect;
  // 下一个副作用
  this.nextEffect = null;
  // 第一个副作用
  this.firstEffect = null;
  // 最后一个副作用
  this.lastEffect = null;
 // 过期时间
  this.expirationTime = NoWork;
  this.childExpirationTime = NoWork;

  this.alternate = null;
  ...
}
```

代码示例 3.4.2 FiberNode 构造函数的源码定义

return， child 和 sibling 三个属性主要用途是将每个 Fiber 结点连接起来。effectTag， nextEffect， firstEffect 和 lastEffect 四个属性主要用途是生成副作用列表。

备注3-4-2.1 tag 属性

tag 属性标识了当前 Fiber 结点的类型，它常见的取值有以下几种，见代码示例 3.4.3。

```
// 源码位置：packages/shared/ReactWorkTags.js
export const FunctionComponent = 0; // 函数组件元素对应的 Fiber 结点
export const ClassComponent = 1; // Class组件元素对应的 Fiber 结点
export const IndeterminateComponent = 2; // 在不确定是 Class 组件元素还是函数组件元素时的取值
export const HostRoot = 3; // 对应 Fiber 树的根结点
export const HostPortal = 4; // 对应一颗子树，可以另一个渲染器的入口
export const HostComponent = 5; // 宿主组件元素（如div，button等）对应的 Fiber 结点
export const HostText = 6; // 文本元素（如div，button等）对应的 Fiber 结点
export const Fragment = 7;
...
```

代码示例 3.4.3 Fiber 结点的类型

Fiber 结点中 elementType 属性和 type 的取值基本是保持一致的，这两种属性描述的是 React 元素的类型。**Fiber 结点由元素转换而来，相同的元素类型所对应的 Fiber 结点的 tag 属性值也是相同的。** 如 elementType 值为 'div' 和 'span' 的 React 元素在 Fiber 架构中都属于「宿主组件元素」，对应 Fiber 结点的 tag 属性值是 HostComponent。换句话说就是，一种按照 React 元素类型进行分类，一种是按照Fiber 对象的类型进行分类，两者之间有一定的映射关系。

备注3-4-2.2 stateNode 属性

Fiber 对象中 stateNode 属性用于存储 Fiber 结点在更新完成后的状态，比如 HostComponent 类型的 Fiber 结点的 stateNode 属性值是其 DOM 元素实例， ClassComponent 类型的 Fiber 结点的 stateNode 属性值是其组件实例。

```
HostComponent.stateNode --> #div, #span
ClassComponent.stateNode --> new Component(...)
```

备注3-4-2.3 current 树和 Fiber 树到底是什么

我们经常提到 「current 树」 和 「Fiber 树」，为什么会这样称呼呢？

Fiber 树其实是一个对象，这个对象通过层层 Fiber 结点的嵌套形成了一个「树」形结构，在数据结构层面它是一个 **链表**。Fiber 树的根结点是 HostRoot 类型的 FiberNode 实例。Fiber 结点中的 return， child 和 sibling 三个属性分别用于指向 **父结点，第一个孩子结点和兄弟结点**。

现在我们以一个简单的组件为例分析 Fiber 树的实际结构，见代码示例 3.4.4。

```
class UpdateCounter extends Component{
  render() {
    return (
      <div className="wrap-box">
        <button key="1">点击计数</button>
        <span className="span-text" key="2">1</span>
      </div>
    )
  }
}
```

代码示例 3.4.4

通过对代码示例 3.4.4 中的程序进行断点调试，我们绘制出其在内存中的 Fiber 树结构，见图 3.4.2 。

图 3.4.2 Fiber 树结构

> 建议将本节图 3.4.2 和本章第一节中图 3.1.3 结合在一起理解。此外，本文中提到的 workInProgress 树，Effect list 等内容在第五章介绍 React 应用程序首次渲染时会详细介绍。

小结

FiberRootNode 构造函数只有一个实例就是 fiberRoot 对象。而每个 Fiber 结点都是 FiberNode 构造函数的实例，它们通过 return，child 和 sibling 三个属性连接起来，形成了一个巨大链表。React 对每个结点的更新计算都是在这个链表上完成的。React 在对 Fiber 结点标记更新标识的时候的做法就是为结点的 effectTag 属性赋不同的值，这个赋值逻辑用了较多的位运算，下一节将会详细 Fiber 结点中的 effectTag 与位运算。

}

13 了解基于 React Fiber 架构的应用程序整体渲染流程

15 理解 React Fiber 架构中中的 effectTag 与位运算