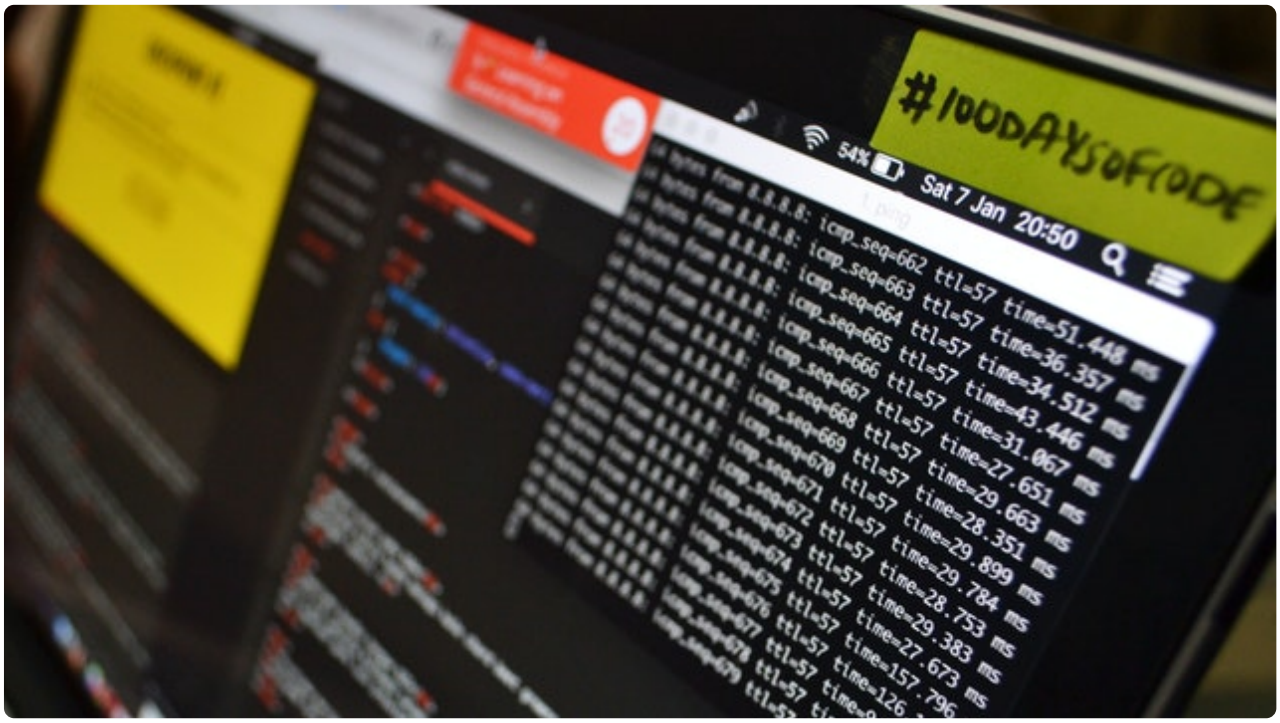
10 React 如何定义更新队列以及它们之间的相互作用关系？

更新时间：2020-08-17 15:01:49



> 没有引发任何行动的思想都不是思想，而是梦想。 —— 马丁

前言

更新（update）与更新队列（updateQueue）是 React 应用程序运行时内部维护的两个携带更新数据的对象，如何定义两者的对象结构以及两者之间的关系对于 React 来说是非常关键的。本文将主要介绍 React 对更新与更新队列的定义，更新加入更新队列及处理更新队列的逻辑。

面试官经常会问到这样一个问题：**在一段代码中连续使用多个** `setState(...)` **时 React 的处理方式是什么？**

```
// 同步更新
handleClick() {
  this.setState({
   count: this.state.count + 1
  });
  this.setState({
    text: '点击计数' + this.state.count
  });
}

// 异步更新
handleClick() {
  this.setState({
   count: this.state.count + 1
  });
  setTimeout(() => {
    this.setState({
      text: '点击计数' + this.state.count
    });
  }, 1000);
}
```

代码示例 2.7.1 连续多个 setState 操作

在代码示例 2.7.1 中，如果有多个同步 `setState(...)` 操作，React 会将它们的更新（update）先后依次加入到更新队列（updateQueue），在应用程序的 render 阶段处理更新队列时会将队列中的所有更新合并成一个，**合并原则是相同属性的更新取最后一次的值**。如果有异步 `setState(...)` 操作，则先进行同步更新，异步更新则遵循 EventLoop 原理后续处理。那么，React 对更新和更新队列是如何定义的呢？

React 对更新（update）的定义

React 对更新对象（update）的定义见代码示例 2.7.2。

```
// 源码位置：packages/react-reconciler/src/ReactUpdateQueue.js
function createUpdate(expirationTime, suspenseConfig) {
  var update = {
    // 过期时间与任务优先级相关联
    expirationTime: expirationTime,
    suspenseConfig: suspenseConfig,
  // tag用于标识更新的类型如UpdateState，ReplaceState，ForceUpdate等
    tag: UpdateState,
    // 更新内容
    payload: null,
    // 更新完成后的回调
    callback: null,
  // 下一个更新（任务）
    next: null,
    // 下一个副作用
    nextEffect: null
  };
  {
    // 优先级会根据任务体系中当前任务队列的执行情况而定
    update.priority = getCurrentPriorityLevel();
  }
  return update;
}
```

代码示例 2.7.2 React 对更新对象的定义

每一个更新对象都有自己的过期时间（`expirationTime`）、更新内容（`payload`），优先级（`priority`）以及指向下一个更新的引用（`next`）。其中当前更新的优先级由任务体系统一指定。

React 对更新队列（updateQueue）的定义

React 对更新队列对象（updateQueue）的定义见代码示例 2.7.3。

```
// 源码位置：packages/react-reconciler/src/ReactUpdateQueue.js
function createUpdateQueue(baseState) {
  var queue = {
    // 当前的state
    baseState: baseState,
    // 队列中第一个更新
    firstUpdate: null,
    // 队列中的最后一个更新
    lastUpdate: null,
    // 队列中第一个捕获类型的update
    firstCapturedUpdate: null,
    // 队列中第一个捕获类型的update
    lastCapturedUpdate: null,
    // 第一个副作用
    firstEffect: null,
    // 最后一个副作用
    lastEffect: null,
    firstCapturedEffect: null,
    lastCapturedEffect: null
  };
  return queue;
}
```

代码示例 2.7.3 React 对更新队列对象的定义

值得注意的是，更新队列的数据结构不是数组，而是一个普通对象（一个单向链表结构）。要想实现数据驱动页面更新，更新内容需要加入到更新队列，这个过程的逻辑是什么样的呢？

React 将更新加入到更新队列

当我们使用 `setState(...)` 时， React 会创建一个更新（update）对象，然后通过调用 `enqueueUpdate` 函数将其加入到更新队列（updateQueue），见代码示例 2.7.4。

```
// 源码位置：packages/react-reconciler/src/ReactUpdateQueue.js
// 每次setState都会创建update并入updateQueue
function enqueueUpdate(fiber, update) {
  // 每个Fiber结点都有自己的updateQueue，其初始值为null，一般只有ClassComponent类型的结点updateQueue才会被赋值
  // fiber.alternate指向的是该结点在workInProgress树上面对应的结点
  var alternate = fiber.alternate;
  var queue1 = void 0;
  var queue2 = void 0;
  if (alternate === null) {
    // 如果fiber.alternate不存在
    queue1 = fiber.updateQueue;
    queue2 = null;
    if (queue1 === null) {
      queue1 = fiber.updateQueue = createUpdateQueue(fiber.memoizedState);
    }
  } else {
    // 如果fiber.alternate存在，也就是说存在current树上的结点和workInProgress树上的结点都存在
    queue1 = fiber.updateQueue;
    queue2 = alternate.updateQueue;
    if (queue1 === null) {
      if (queue2 === null) {
        // 如果两个结点上面均没有updateQueue，则为它们分别创建queue
        queue1 = fiber.updateQueue = createUpdateQueue(fiber.memoizedState);
        queue2 = alternate.updateQueue = createUpdateQueue(alternate.memoizedState);
      } else {
        // 如果只有其中一个存在updateQueue，则将另一个结点的updateQueue克隆到该结点
        queue1 = fiber.updateQueue = cloneUpdateQueue(queue2);
      }
    } else {
      if (queue2 === null) {
        // 如果只有其中一个存在updateQueue，则将另一个结点的updateQueue克隆到该结点
        queue2 = alternate.updateQueue = cloneUpdateQueue(queue1);
      } else {
        // 如果两个结点均有updateQueue，则不需要处理
      }
    }
  }
  if (queue2 === null || queue1 === queue2) {
    // 经过上面的处理后，只有一个queue1或者queue1 == queue2的话，就将更新对象update加入到queue1
    appendUpdateToQueue(queue1, update);
  } else {
    // 经过上面的处理后，如果两个queue均存在
    if (queue1.lastUpdate === null || queue2.lastUpdate === null) {
      // 只要有一个queue不为null，就需要将将update加入到queue中
      appendUpdateToQueue(queue1, update);
      appendUpdateToQueue(queue2, update);
    } else {
      // 如果两个都不是空队列，由于两个结构共享，所以只在queue1加入update
      appendUpdateToQueue(queue1, update);
      // 仍然需要在queue2中，将lastUpdate指向update
      queue2.lastUpdate = update;
    }
  }
  ...
}

function appendUpdateToQueue(queue, update) {
  if (queue.lastUpdate === null) {
    // 如果队列为空，则第一个更新和最后一个更新都赋值当前更新
    queue.firstUpdate = queue.lastUpdate = update;
  } else {
    // 如果队列不为空，将update加入到队列的末尾
    queue.lastUpdate.next = update;
    queue.lastUpdate = update;
  }
}
```

代码示例 2.7.4 更新加入到更新队列逻辑

在 enqueueUpdate 函数中，React 将更新加入到更新队列时会同时维护两个队列对象 queue1 和 queue2，其中 queue1 是应用程序运行过程中 current 树上当前 Fiber 结点最新队列，queue2 是应用程序上一次更新时（workInProgress 树）Fiber 结点的更新队列，它们之间的相互逻辑是下面这样的。

- queue1 取的是 fiber.updateQueue，queue2 取的是 fiber.alternate.updateQueue；
- 如果两者均为 null，则调用 createUpdateQueue(...) 获取初始队列；
- 如果两者之一为 null，则调用 cloneUpdateQueue(...) 从对方中获取队列；
- 如果两者均不为 null，则将 update 作为 lastUpdate 加入多 queue1 中。

前面提到 React 的更新队列在内存中的结构是一个单链表，代码示例 2.7.1 中同步情况下连续执行两个 setState(...) 后更新队列在内存中的结构如图 2.7.1这样。

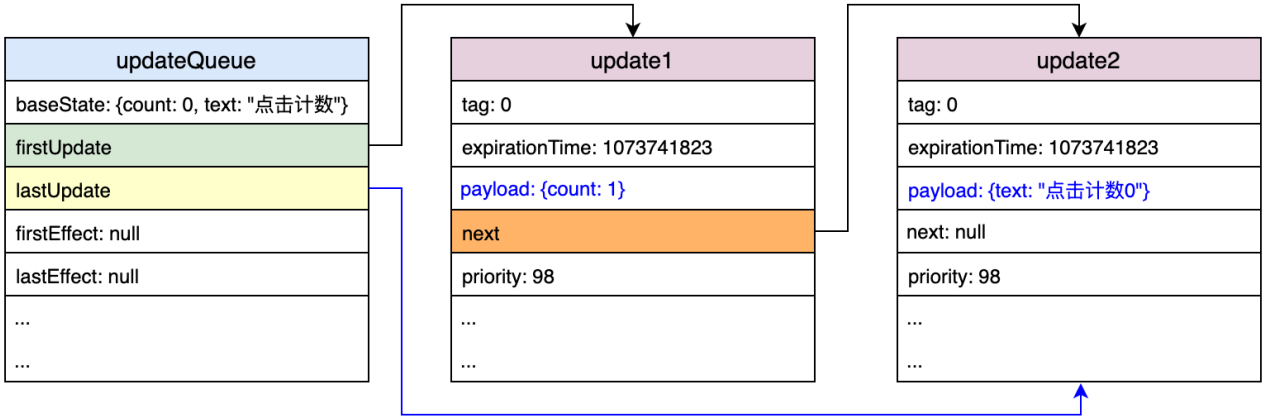


图 2.7.1 React 应用程序运行时更新队列的结构

值得注意的是，整个更新队列对象通过 firstUpdate 属性和更新对象的 next 属性层层引用形成了链表结构。同时更新队列对象中也可以通过 lastUpdate 属性直接连接到最后一个更新对象，即 updateQueue.firstUpdate.next...next 的值会一直和 updateQueue.lastUpdate 执行的更新对象相同。

React 如何处理更新队列

React 应用程序运行到 render 阶段时会处理更新队列，处理更新队列的函数是 processUpdateQueue，见代码示例 2.7.5。

```
// 源码位置：packages/react-reconciler/src/ReactUpdateQueue.js
function processUpdateQueue(workInProgress, queue, props, instance, renderExpirationTime) {
  ...
  // 从队列中取出第一个更新
  var update = queue.firstUpdate;
  var resultState = newBaseState;
  // 遍历更新队列，处理更新
  while (update !== null) {
    ...
    // 如果第一个更新不为空，紧接着要遍历更新队列
    // getStateFromUpdate函数用于合并更新，合并方式见下面函数实现
    resultState = getStateFromUpdate(workInProgress, queue, update, resultState, props, instance);
    ...
    update = update.next;
  }
  ...
  // 设置当前fiber结点的memoizedState
  workInProgress.memoizedState = resultState;
  ...
}

// 获取下一个更新对象并与现有state对象合并
function getStateFromUpdate(workInProgress, queue, update, prevState, nextProps, instance) {
  switch (update.tag) {
    case UpdateState:
      {
        var _payload2 = update.payload;
        var partialState = void 0;
        if (typeof _payload2 === 'function') {
          // setState传入的参数_payload2类型是function
          ...
          partialState = _payload2.call(instance, prevState, nextProps);
          ...
        } else {
          // setState传入的参数_payload2类型是object
          partialState = _payload2;
        }
        // 合并当前state和上一个state.
        return _assign({}, prevState, partialState);
      }
  }
}
```

代码示例 2.7.5 React 处理更新队列

代码示例 2.7.5 中，`processUpdateQueue` 函数用于处理更新队列，在该函数内部使用循环的方式来遍历队列，通过 `update.next` 依次取出更新（对象）进行合并，合并更新对象的方式是：

- 如果 `setState` 传入的参数类型是 `function`，则通过 `payload2.call(instance, prevState, nextProps)` 获取更新对象；
- 如果 `setState` 传入的参数类型是object，则可直接获取更新对象；
- 最后通过使用 `Object.assign()` 合并两个更新对象并返回，如果属性相同的情况下则取最后一次值。

小结

更新和更新队列的处理是 React 应用程序渲染过程中必须要做的工作，它们携带了更新内容的详细信息。更新和更新队列的数据类型均是普通 JavaScript 对象，而不是数组类型。在处理更新队列时，React 会根据更新对象中携带的 `state` 相同属性进行合并，保留队列中最后一次属性值，以此作为前后结点 diff 的数据。

在本节提到了 current 树和 workInProgress 树，它们到底是什么呢？在下一章介绍 React Fiber 架构时将会揭开它们的面纱。

```
}
```

09 元素：从设计者的角度理解 React 元素

11 什么是 React Fiber