8-2 root.render与unmount函数的流程

资源

1. createRoot

`createRoot` 函数接受两个参数，`container` 与 `options`，返回一个 `RootType` 类型，即 `ReactDOMRoot` 的实例。

```
Flow
export type RootType = {
  render(children: ReactNodeList): void,
  unmount(): void,
  _internalRoot: FiberRoot | null,
};
```

```
Flow
function ReactDOMRoot(internalRoot: FiberRoot) {
  this._internalRoot = internalRoot;
}
```

```
ReactDOMHydrationRoot.prototype.render = ReactDOMRoot.prototype.render
  function (children: ReactNodeList): void {
    const root = this._internalRoot;
    if (root === null) {
      throw new Error('Cannot update an unmounted root.');
    }
    updateContainer(children, root, null, null);
  };
```

`updateContainer`

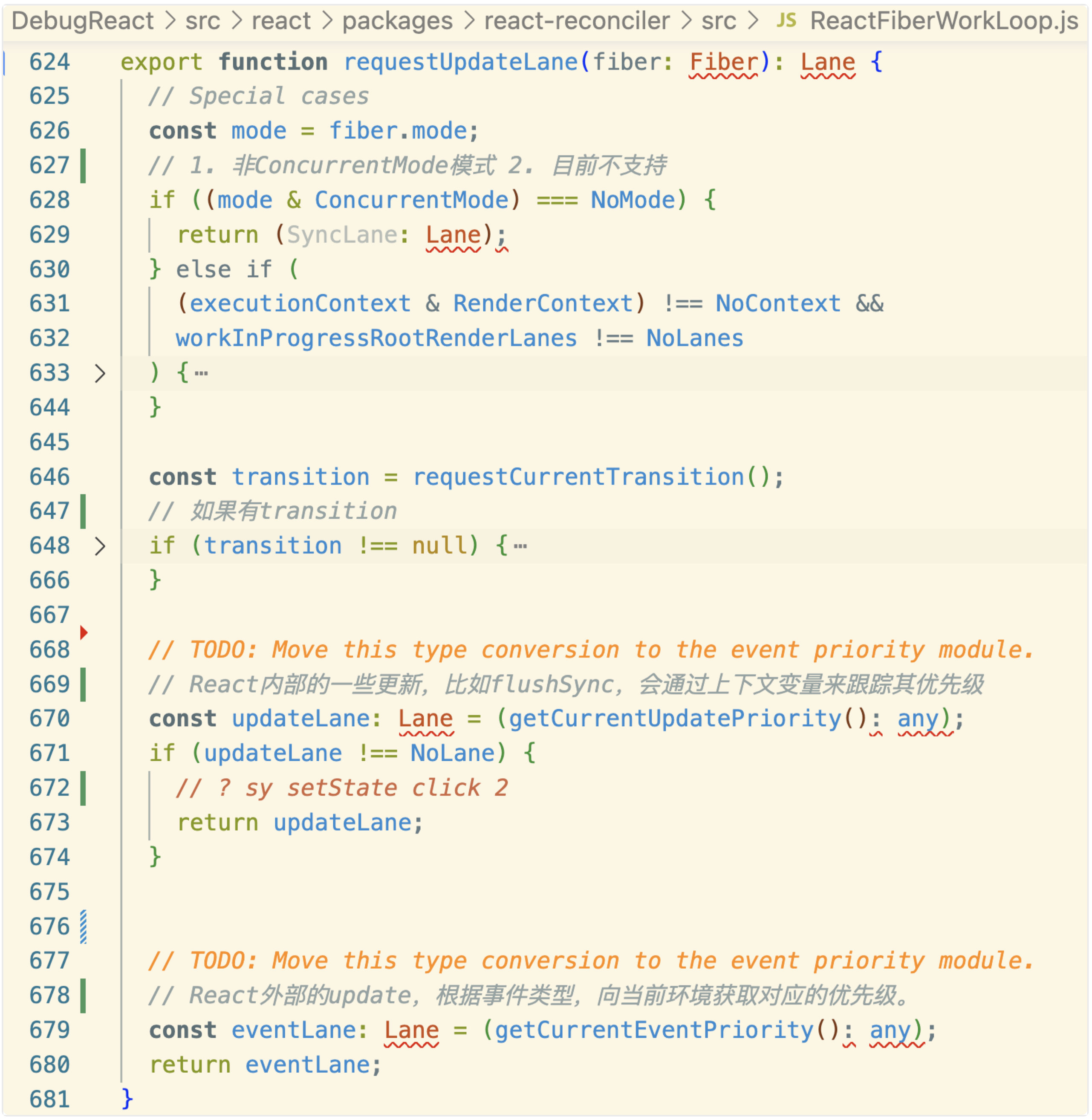
react/packages/react-reconciler/src/ReactFiberReconciler.js

1. 获取 current 和 lane

在 React 中，`lane` 是用于标识 update 优先级，可以理解为表示 update 的优先级的一种机制。每个 update 都会被分配一个或多个 `lane`，以确定其在更新队列中的优先级顺序。

```flow
const current = container.current;
// 获取本次update对应的lane
const lane = requestUpdateLane(current);
```

`requestUpdateLane`



DebugReact > src > react > packages > react-reconciler > src > JS ReactFiberWorkLoop.js

```
export function requestUpdateLane(fiber: Fiber): Lane {
  // Special cases
  const mode = fiber.mode;
  // 1. 非ConcurrentMode模式 2. 目前不支持
  if ((mode & ConcurrentMode) === NoMode) {
    return (SyncLane: Lane);
  } else if (
    (executionContext & RenderContext) !== NoContext &&
    workInProgressRootRenderLanes !== NoLanes
  ) {…
  }

  const transition = requestCurrentTransition();
  // 如果有transition
  if (transition !== null) {…
  }

  // TODO: Move this type conversion to the event priority module.
  // React内部的一些更新，比如flushSync，会通过上下文变量来跟踪其优先级
  const updateLane: Lane = (getCurrentUpdatePriority(): any);
  if (updateLane !== NoLane) {
    // ? sy setState click 2
    return updateLane;
  }


  // TODO: Move this type conversion to the event priority module.
  // React外部的update，根据事件类型，向当前环境获取对应的优先级。
  const eventLane: Lane = (getCurrentEventPriority(): any);
  return eventLane;
}
```

`getCurrentEventPriority`

react/packages/react-dom-bindings/src/client/ReactFiberConfigDOM.js

```Flow
```

```
export function getCurrentEventPriority(): EventPriority {
  const currentEvent = window.event;
  if (currentEvent === undefined) {
    // ? sy 页面初次渲染
    return DefaultEventPriority;
  }
  return getEventPriority(currentEvent.type);
}
```

`DefaultEventPriority`、`getCurrentUpdatePriority`

react/packages/react-reconciler/src/ReactEventPriorities.js

```Flow
export opaque type EventPriority = Lane;

export const DiscreteEventPriority: EventPriority = SyncLane;
export const ContinuousEventPriority: EventPriority = InputContinuousL
export const DefaultEventPriority: EventPriority = DefaultLane; // 页面
export const IdleEventPriority: EventPriority = IdleLane;

let currentUpdatePriority: EventPriority = NoLane;

// get
export function getCurrentUpdatePriority(): EventPriority {
  return currentUpdatePriority;
}

// set
export function setCurrentUpdatePriority(newPriority: EventPriority) {
  currentUpdatePriority = newPriority;
}
```

2. 创建 update

```Flow
const update = createUpdate(lane);
update.payload = {element};
// React18中已取消callback，只有老版本有效
callback = callback === undefined ? null : callback;
```

```
if (callback !== null) {
  update.callback = callback;
}
```

`createUpdate`

创建 update

react/packages/react-reconciler/src/ReactFiberClassUpdateQueue.js

```Flow
export const UpdateState = 0;

export type Update<State> = {
  lane: Lane,

  tag: 0 | 1 | 2 | 3,
  payload: any,
  callback: (() => mixed) | null,

  next: Update<State> | null,
};

export function createUpdate(lane: Lane): Update<mixed> {
  const update: Update<mixed> = {
    lane,

    tag: UpdateState,
    payload: null,
    callback: null,

    next: null,
  };
  return update;
}
```

3. update 入队

react/packages/react-reconciler/src/ReactFiberClassUpdateQueue.js

```
Flow
export function enqueueUpdate<State>(
  fiber: Fiber,
  update: Update<State>,
  lane: Lane,
): FiberRoot | null {
  const updateQueue = fiber.updateQueue;
  if (updateQueue === null) {
    // Only occurs if the fiber has been unmounted.
    return null;
  }

  const sharedQueue: SharedQueue<State> = (updateQueue: any).shared;

  if (isUnsafeClassRenderPhaseUpdate(fiber)) {
    // 类组件旧的生命周期相关的update，这里不再展开详解
    // 代码略
  } else {
    // sy
    return enqueueConcurrentClassUpdate(fiber, sharedQueue, update, la
  }
}
```

`enqueueConcurrentClassUpdate`

react/packages/react-reconciler/src/ReactFiberConcurrentUpdates.js

全局变量与类型

```
Flow
export type ConcurrentUpdate = {
  next: ConcurrentUpdate,
  lane: Lane,
};

type ConcurrentQueue = {
  pending: ConcurrentUpdate | null,
};
```

```
const concurrentQueues: Array<any> = [];
let concurrentQueuesIndex = 0;

let concurrentlyUpdatedLanes: Lanes = NoLanes;
```

`enqueueConcurrentClassUpdate`

更新入队，并且

```Flow
export function enqueueConcurrentClassUpdate<State>(
  fiber: Fiber,
  queue: ClassQueue<State>,
  update: ClassUpdate<State>,
  lane: Lane,
): FiberRoot | null {
  const concurrentQueue: ConcurrentQueue = (queue: any);
  const concurrentUpdate: ConcurrentUpdate = (update: any);
  // 1. update入队
  enqueueUpdate(fiber, concurrentQueue, concurrentUpdate, lane);
  // 2. 返回FiberRoot
  return getRootForUpdatedFiber(fiber);
}
```

`enqueueUpdate`

把 update 存储到 concurrentQueues 中，虽然这个函数也叫 `enqueueUpdate`。

这里的 `enqueueUpdate` 是数字式存储，并且是依次存储 fiber、queue、update、lane，下次依然这个顺序。当然，最后执行处理的时候也要按照这个规律取值。

React 源码中其它地方这种结构都是对象式写法，这里比较罕见地写了这个结构 ~

```Flow
function enqueueUpdate(
  fiber: Fiber,
  queue: ConcurrentQueue | null,
  update: ConcurrentUpdate | null,
  lane: Lane,
) {
  // Don't update the `childLanes` on the return path yet. If we alrea
  // the middle of rendering, wait until after it has completed.
  concurrentQueues[concurrentQueuesIndex++] = fiber;
  concurrentQueues[concurrentQueuesIndex++] = queue;
  concurrentQueues[concurrentQueuesIndex++] = update;
  concurrentQueues[concurrentQueuesIndex++] = lane;

  concurrentlyUpdatedLanes = mergeLanes(concurrentlyUpdatedLanes, lane

  // The fiber's `lane` field is used in some places to check if any w
  // scheduled, to perform an eager bailout, so we need to update it i
  // TODO: We should probably move this to the "shared" queue instead.
  fiber.lanes = mergeLanes(fiber.lanes, lane);
  const alternate = fiber.alternate;
  if (alternate !== null) {
    alternate.lanes = mergeLanes(alternate.lanes, lane);
  }
}
```

`getRootForUpdatedFiber` 找到 `FiberRoot`

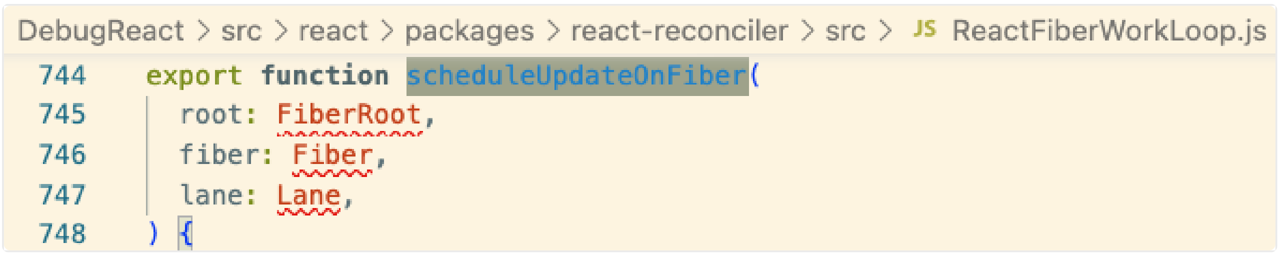
从 sourceFiber 开始，找到根 Fiber，返回其 stateNode，即 FiberRoot

```
Flow
```

```
function getRootForUpdatedFiber(sourceFiber: Fiber): FiberRoot | null
  // 如果循环超过限制次数(50次)，抛出错误。比如在类组件的render函数里执行setStat
  throwIfInfiniteUpdateLoopDetected();

  // __DEV__，检查是否有未挂载的Fiber，如Can't perform a React state updat
  detectUpdateOnUnmountedFiber(sourceFiber, sourceFiber);
  let node = sourceFiber;
  let parent = node.return;
  // 循环往上查找，找到根节点
  while (parent !== null) {
    detectUpdateOnUnmountedFiber(sourceFiber, node);
    node = parent;
    parent = node.return;
  }
  // 根节点一定是HostRoot，返回根节点的stateNode，即FiberRoot
  return node.tag === HostRoot ? (node.stateNode: FiberRoot) : null;
}
```

4. `scheduleUpdateOnFiber` 调度更新


DebugReact > src > react > packages > react-reconciler > src > JS ReactFiberWorkLoop.js

```
export function scheduleUpdateOnFiber(
  root: FiberRoot,
  fiber: Fiber,
  lane: Lane,
) {
```

调度 update。

`scheduleUpdateOnFiber` 详细查看后面章节。

5. `entangleTransitions` 非紧急更新

处理 transitions，transitions 是非紧急更新。本篇暂不展开，详情查看后面章节。