9–4 render 阶段

performConcurrentWorkOnRoot

TypeScript
```typescript
export function performConcurrentWorkOnRoot(root: FiberRoot) {
  // ! 1. render
  renderRootSync(root);

  const finishedWork: Fiber = root.current.alternate as Fiber;

  // !3. commit
  root.finishedWork = finishedWork;

  commitRoot(root);
}
```

executionContext

TypeScript
```typescript
type ExecutionContext = number;

export const NoContext = /*                */ 0b000;
```

```
const BatchedContext = /*                    */ 0b001;
export const RenderContext = /*              */ 0b010;
export const CommitContext = /*              */ 0b100;

// Describes where we are in the React execution stack
let executionContext: ExecutionContext = NoContext;
```

render 阶段

```typescript
export function renderRootSync(root: FiberRoot) {
  // ! 1. 记录 render阶段 开始
  const prevExecutionContext = executionContext;
  executionContext |= RenderContext;

  // ! 3. 初始化
  prepareFreshStack(root);

  workLoopSync();

  // ! 5. 重置
  executionContext = prevExecutionContext;

  // 设置为null，表示没有进行中的render了
  workInProgressRoot = null;
}
```

1. `prepareFreshStack`

```typescript
function prepareFreshStack(root: FiberRoot): Fiber {
  root.finishedWork = null;

  workInProgressRoot = root; // FiberRoot
  const rootWorkInProgress = createWorkInProgress(root.current, null);
  workInProgress = rootWorkInProgress; // Fiber
  return rootWorkInProgress;
}
```

`createWorkInProgress`

```
                                                                    TypeScript
export function createWorkInProgress(current: Fiber, pendingProps: any
  let workInProgress = current.alternate;
  if (workInProgress === null) {
    workInProgress = createFiber(current.tag, pendingProps, current.ke
    workInProgress.elementType = current.elementType;
    workInProgress.type = current.type;
    workInProgress.stateNode = current.stateNode;

    workInProgress.alternate = current;
    current.alternate = workInProgress;
  } else {
    workInProgress.pendingProps = pendingProps;
    workInProgress.type = current.type;

    workInProgress.flags = NoFlags;
  }

   workInProgress.flags = current.flags;

  workInProgress.child = current.child;
  workInProgress.memoizedProps = current.memoizedProps;
  workInProgress.memoizedState = current.memoizedState;

  workInProgress.sibling = current.sibling;
  workInProgress.index = current.index;

  return workInProgress;
}
```

2. `workLoopSync`

```TypeScript
function workLoopSync() {
  while (workInProgress !== null) {
    performUnitOfWork(workInProgress);
  }
}
```

`performUnitOfWork`

```JavaScript
function performUnitOfWork(unitOfWork: Fiber): void {
  const current = unitOfWork.alternate;

  let next = beginWork(current, unitOfWork);

  // ! 把pendingProps更新到memoizedProps
  unitOfWork.memoizedProps = unitOfWork.pendingProps;
  if (next === null) {
    // 如果不再产生新的work，那么当前work结束
    completeUnitOfWork(unitOfWork);
  } else {
    workInProgress = next;
  }
}
```

`beginWork`

1. 更新当前 fiber，比如 props/state 更新，生命周期函数执行、Hooks 函数执行等。
2. 返回一个下一个 fiber。

详情参考下节 `beginWork` 。

`completeUnitOfWork`

深度优先遍历。

`completeWork` 详情参考 `completeWork` 章节（ReactFiberCompleteWork.ts）

```javascript
function completeUnitOfWork(unitOfWork: Fiber): void {
  let completedWork: Fiber | null = unitOfWork;
  do {
    const current = completedWork.alternate;
    const returnFiber = completedWork.return;
    let next = completeWork(current, completedWork);

    if (next !== null) {
      workInProgress = next;
      return;
    }

    const siblingFiber = completedWork.sibling;
    if (siblingFiber !== null) {
      workInProgress = siblingFiber;
      return;
    }

    completedWork = returnFiber as Fiber;
    workInProgress = completedWork;
  } while (completedWork !== null);
}
```

3. 重置 workInProgressX

这里没有重置 `workInProgress`，因为 `workInProgress` 已经在 `performUnitOfWork` 阶段更新。

```javascript
  // ! 重置
  executionContext = prevExecutionContext;

  // 设置为null，表示没有进行中的render了
  workInProgressRoot = null;
```