React 核心源码解析（下）

星河老师

2021.12.26

课程回顾

React 基本概念

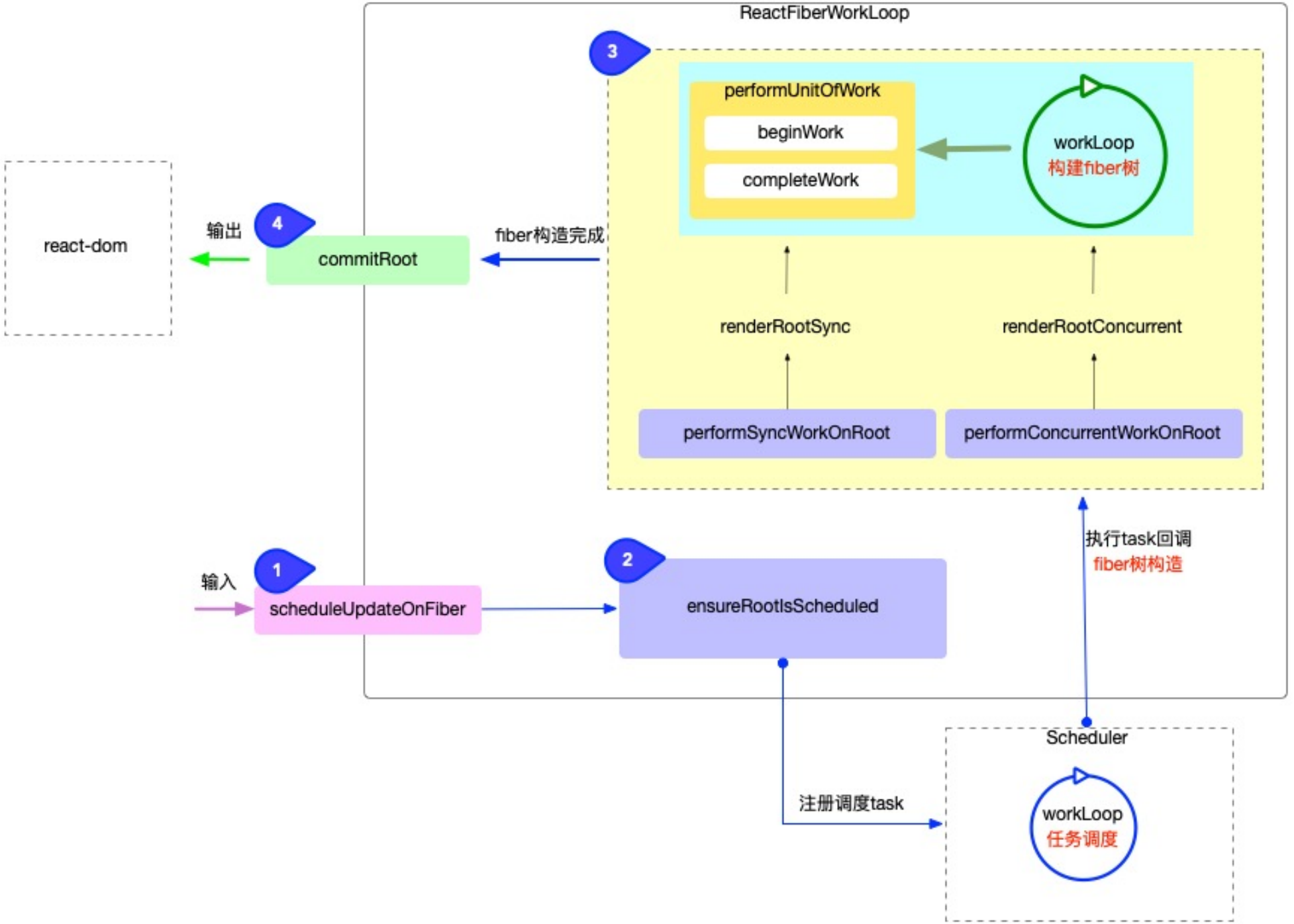
React 基础模块

React 中的数据结构

React 渲染流程

React 更新流程

总结


ReactFiberWorkLoop
3
performUnitOfWork
beginWork
completeWork
workLoop
构建fiber树
react-dom
输出
4
commitRoot
fiber构造完成
renderRootSync
renderRootConcurrent
performSyncWorkOnRoot
performConcurrentWorkOnRoot
执行task回调
fiber树构造
输入
1
scheduleUpdateOnFiber
2
ensureRootIsScheduled
注册调度task
Scheduler
workLoop
任务调度

课程大纲

- React 优先级管理
- Scheduler 调度管理
- 生命周期
- 总结
- Q&A

React 优先级管理

- React 中有3种优先级管理

- SchedulerPriority
- LanePriority
- ReactPriorityLevel

SchedulerPriority

```
export type PriorityLevel = 0 | 1 | 2 | 3 | 4 | 5;

// TODO: Use symbols?
export const NoPriority = 0;
export const ImmediatePriority = 1;
export const UserBlockingPriority = 2;
export const NormalPriority = 3;
export const LowPriority = 4;
export const IdlePriority = 5;

⇩优先级对应的超时时间

// Max 31 bit integer. The max integer size in V8 for 32-bit systems.
// Math.pow(2, 30) - 1
// 0b111111111111111111111111111111
var maxSigned31BitInt = 1073741823;

// Times out immediately
var IMMEDIATE_PRIORITY_TIMEOUT = -1;
// Eventually times out
var USER_BLOCKING_PRIORITY_TIMEOUT = 250;
var NORMAL_PRIORITY_TIMEOUT = 5000;
var LOW_PRIORITY_TIMEOUT = 10000;
// Never times out
var IDLE_PRIORITY_TIMEOUT = maxSigned31BitInt;
```

LanePriority

```
export const SyncLanePriority: LanePriority = 15;
export const SyncBatchedLanePriority: LanePriority = 14;

const InputDiscreteHydrationLanePriority: LanePriority = 13;
export const InputDiscreteLanePriority: LanePriority = 12;

const InputContinuousHydrationLanePriority: LanePriority = 11;
export const InputContinuousLanePriority: LanePriority = 10;

const DefaultHydrationLanePriority: LanePriority = 9;
export const DefaultLanePriority: LanePriority = 8;

const TransitionHydrationPriority: LanePriority = 7;
export const TransitionPriority: LanePriority = 6;

const RetryLanePriority: LanePriority = 5;

const SelectiveHydrationLanePriority: LanePriority = 4;

const IdleHydrationLanePriority: LanePriority = 3;
const IdleLanePriority: LanePriority = 2;

const OffscreenLanePriority: LanePriority = 1;

export const NoLanePriority: LanePriority = 0;
```

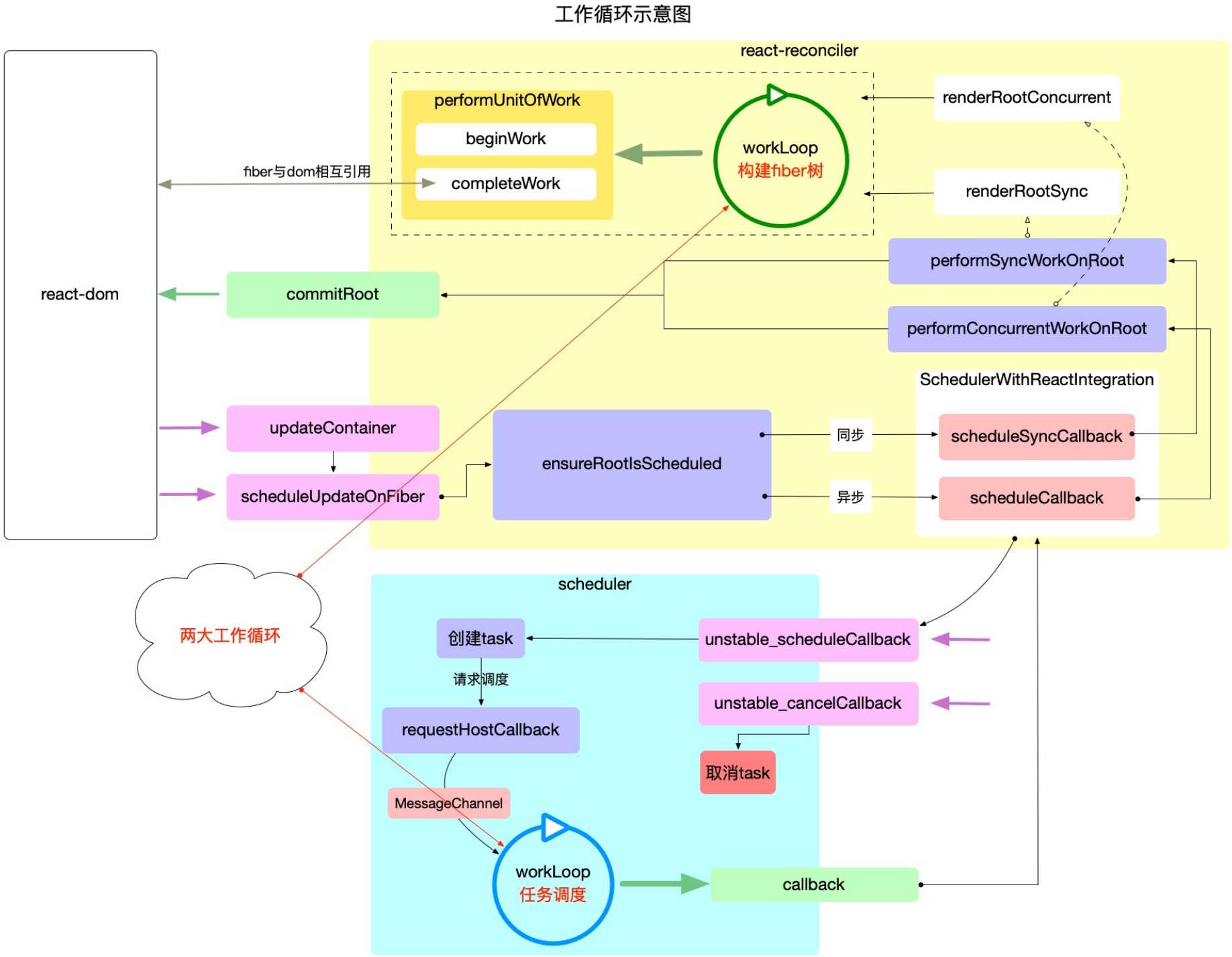
LanePriority

```
export function lanePriorityToSchedulerPriority(
  lanePriority: LanePriority,
): ReactPriorityLevel {
  switch (lanePriority) {
    case SyncLanePriority:
    case SyncBatchedLanePriority:
      return ImmediateSchedulerPriority;
    case InputDiscreteHydrationLanePriority:
    case InputDiscreteLanePriority:
    case InputContinuousHydrationLanePriority:
    case InputContinuousLanePriority:
      return UserBlockingSchedulerPriority;
    case DefaultHydrationLanePriority:
    case DefaultLanePriority:
    case TransitionHydrationPriority:
    case TransitionPriority:
    case SelectiveHydrationLanePriority:
    case RetryLanePriority:
      return NormalSchedulerPriority;
    case IdleHydrationLanePriority:
    case IdleLanePriority:
    case OffscreenLanePriority:
      return IdleSchedulerPriority;
    case NoLanePriority:
      return NoSchedulerPriority;
    default:
      invariant(
        false,
        'Invalid update priority: %s. This is a bug in React.',
        lanePriority,
      );
  }
}
```

ReactPriorityLevel

```
// Except for NoPriority, these correspond to Scheduler priorities. We use
// ascending numbers so we can compare them like numbers. They start at 90 to
// avoid clashing with Scheduler's priorities.
export const ImmediatePriority: ReactPriorityLevel = 99;
export const UserBlockingPriority: ReactPriorityLevel = 98;
export const NormalPriority: ReactPriorityLevel = 97;
export const LowPriority: ReactPriorityLevel = 96;
export const IdlePriority: ReactPriorityLevel = 95;
// NoPriority is the absence of priority. Also React-only.
export const NoPriority: ReactPriorityLevel = 90;
```

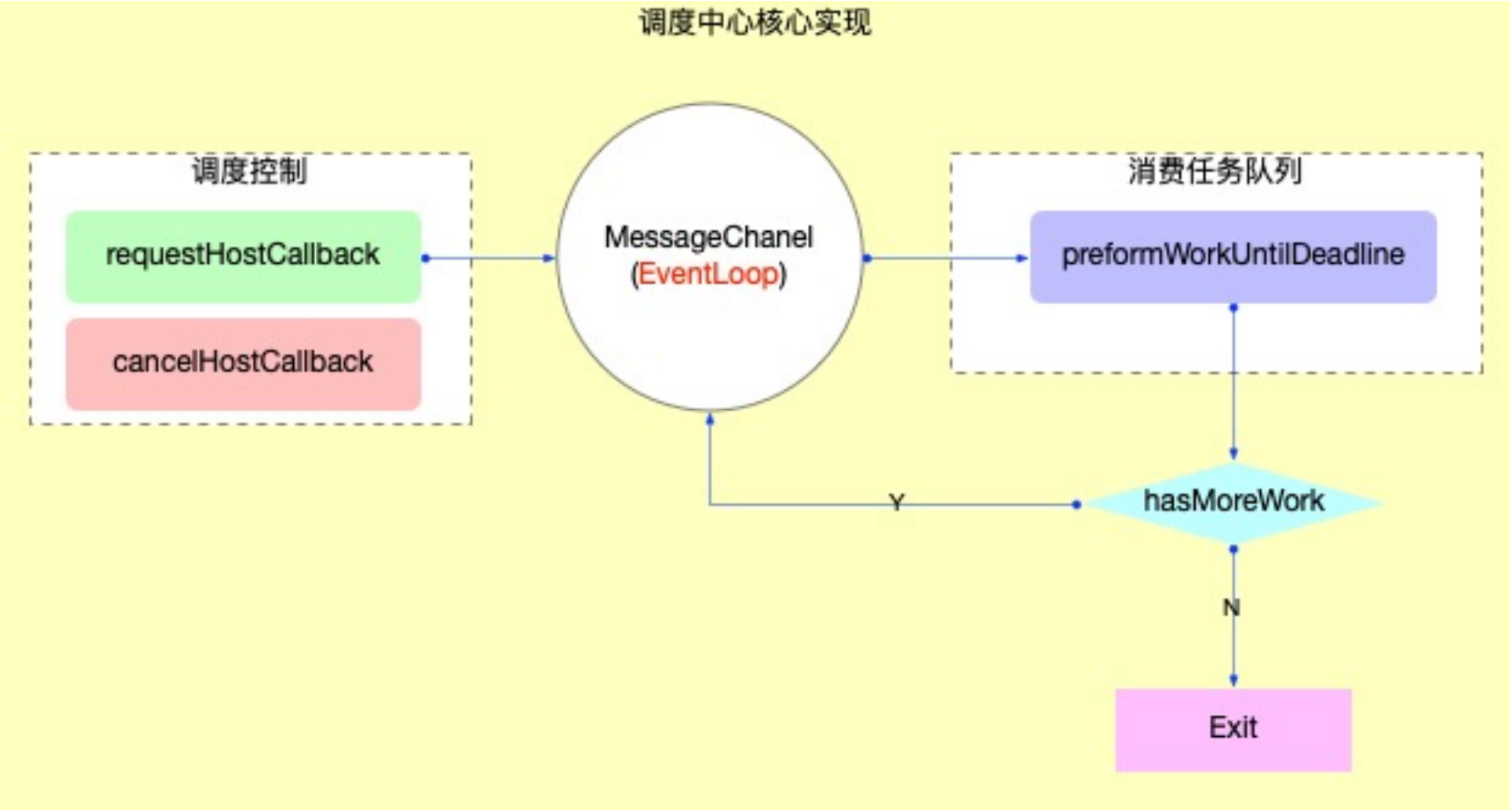
Scheduler 调度管理



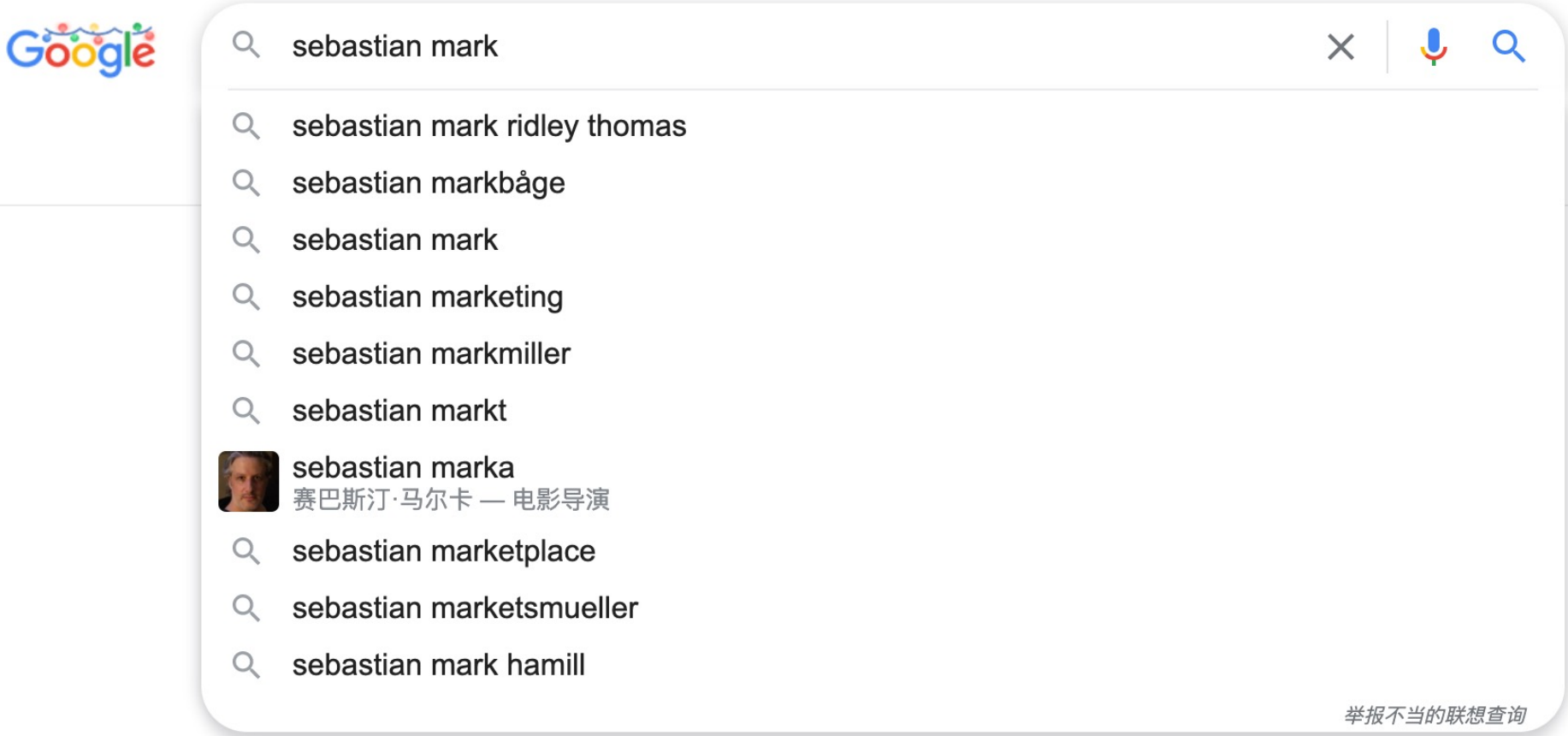
Scheduler 调度管理

requestIdleCallback
window.requestIdleCallback()方法插入一个函数，这个函数将在浏览器空闲时期被调用。

Scheduler 调度管理



Scheduler 调度管理



Scheduler 调度管理

```
// ReactFiberWorkLoop.old.js

function workLoopConcurrent() {
  // Perform work until Scheduler asks us to yield
  while (workInProgress !== null && !shouldYield()) {
    performUnitOfWork(workInProgress);
  }
}


// SchedulerHostConfig.default.js

shouldYield = function() {
  // needPaint 会在 commitRoot 流程中被调用
  // isInputPending 仅在PC端Chrome和Android可用
  if (needsPaint || scheduling.isInputPending()) {
    // There is either a pending paint or a pending input.
    return true;
  }
}

shouldYieldToHost = function() {
  // deadline in performWorkUntilDeadline()
  return getCurrentTime() >= deadline;
};
```

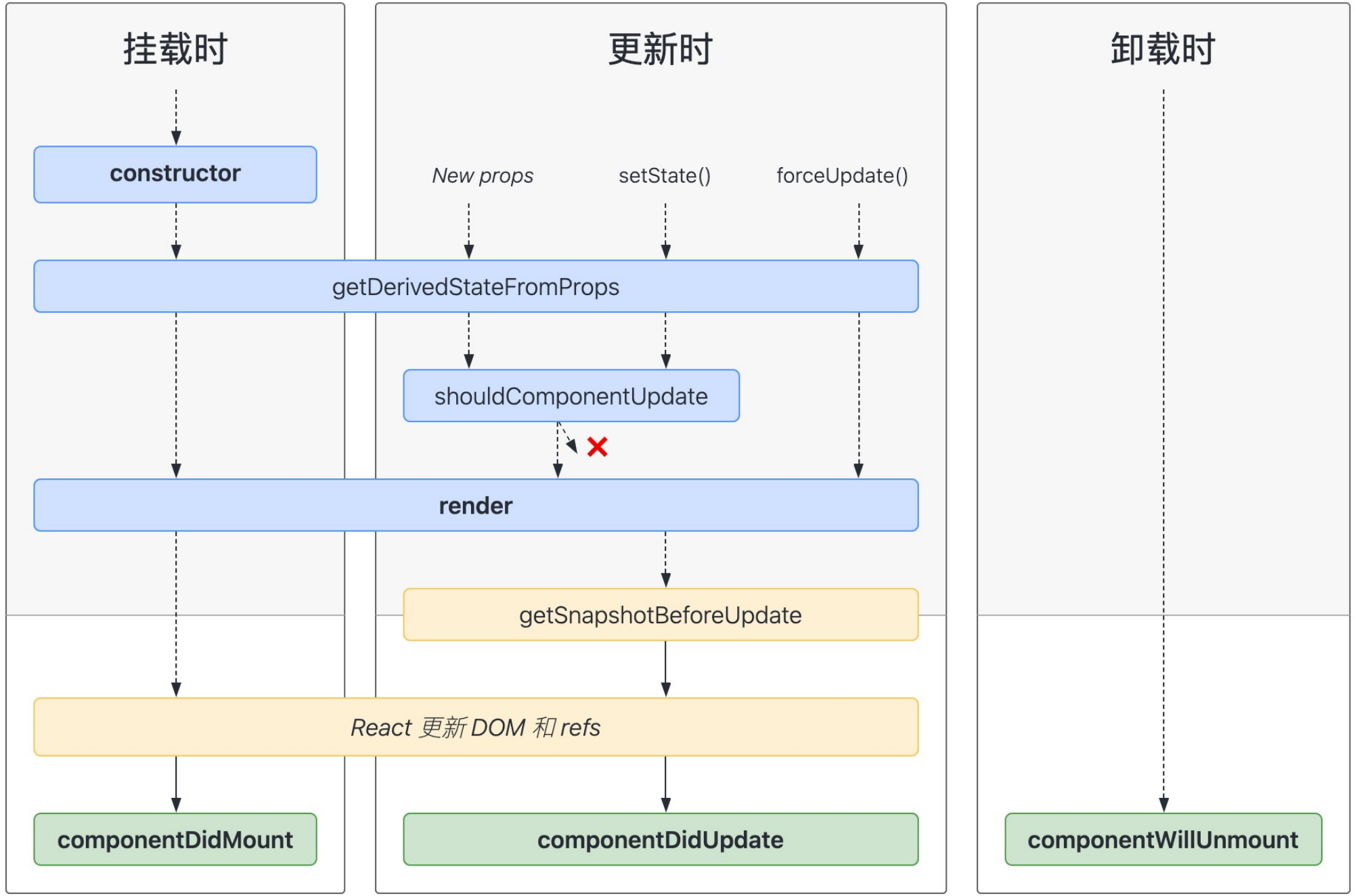
Scheduler 调度管理

```
getCurrentTime  // 获取当前时间
shouldYieldToHost  // 是否让出主线程
requestPaint  // 请求绘制
forceFrameRate  // 强制设置 yieldInterval
deadline  // currentTime + yieldInterval
```

Scheduler 调度管理

```
// 恢复被中断的任务
// performConcurrentWorkOnRoot 方法
if (root.callbackNode === originalCallbackNode) {
  // The task node scheduled for this root is the same one that's
  // currently executed. Need to return a continuation.
  return performConcurrentWorkOnRoot.bind(null, root);
}
```

React 生命周期



“Render 阶段”

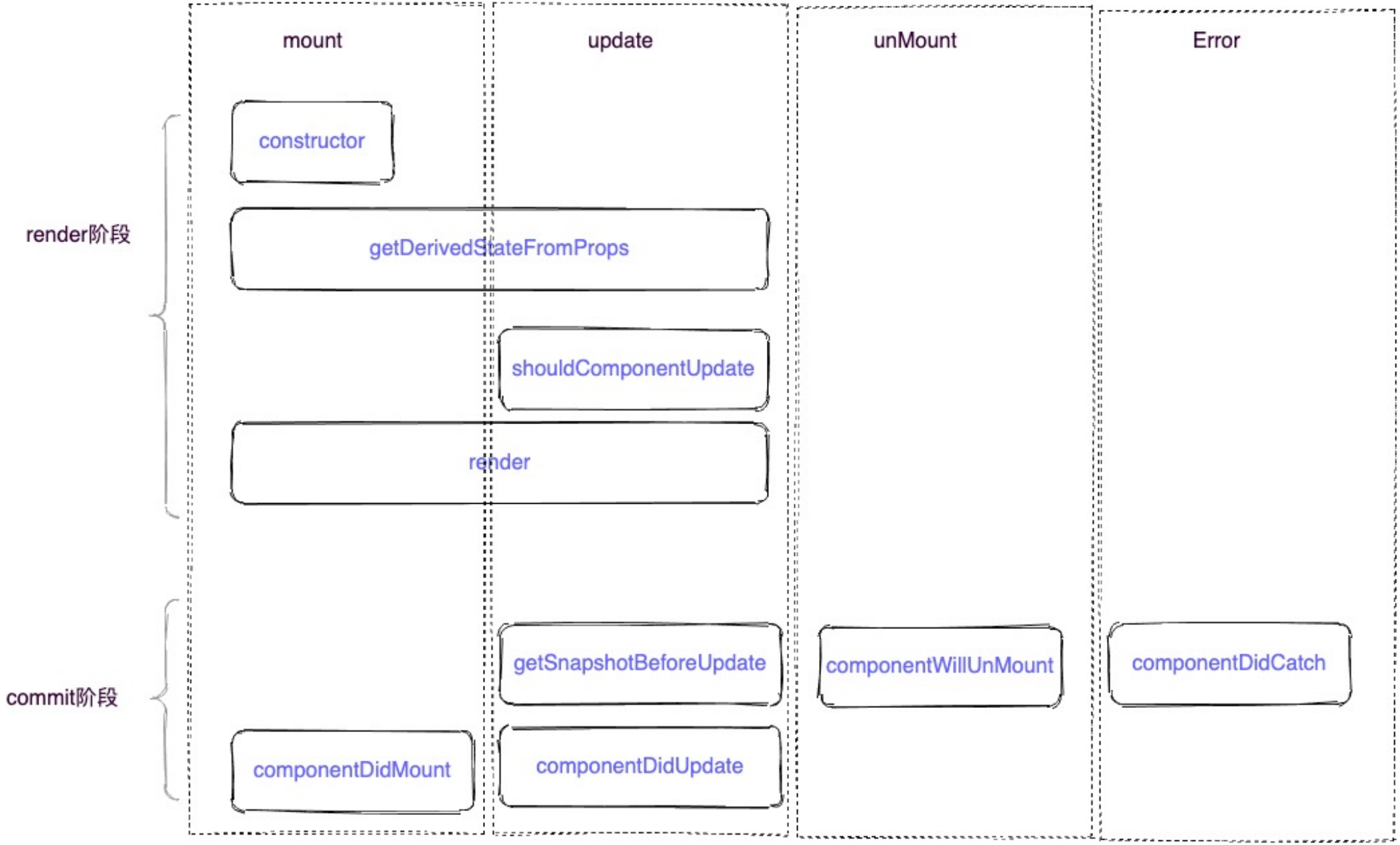
纯净且不包含副作用。可能会被 React 暂停，中止或重新启动。

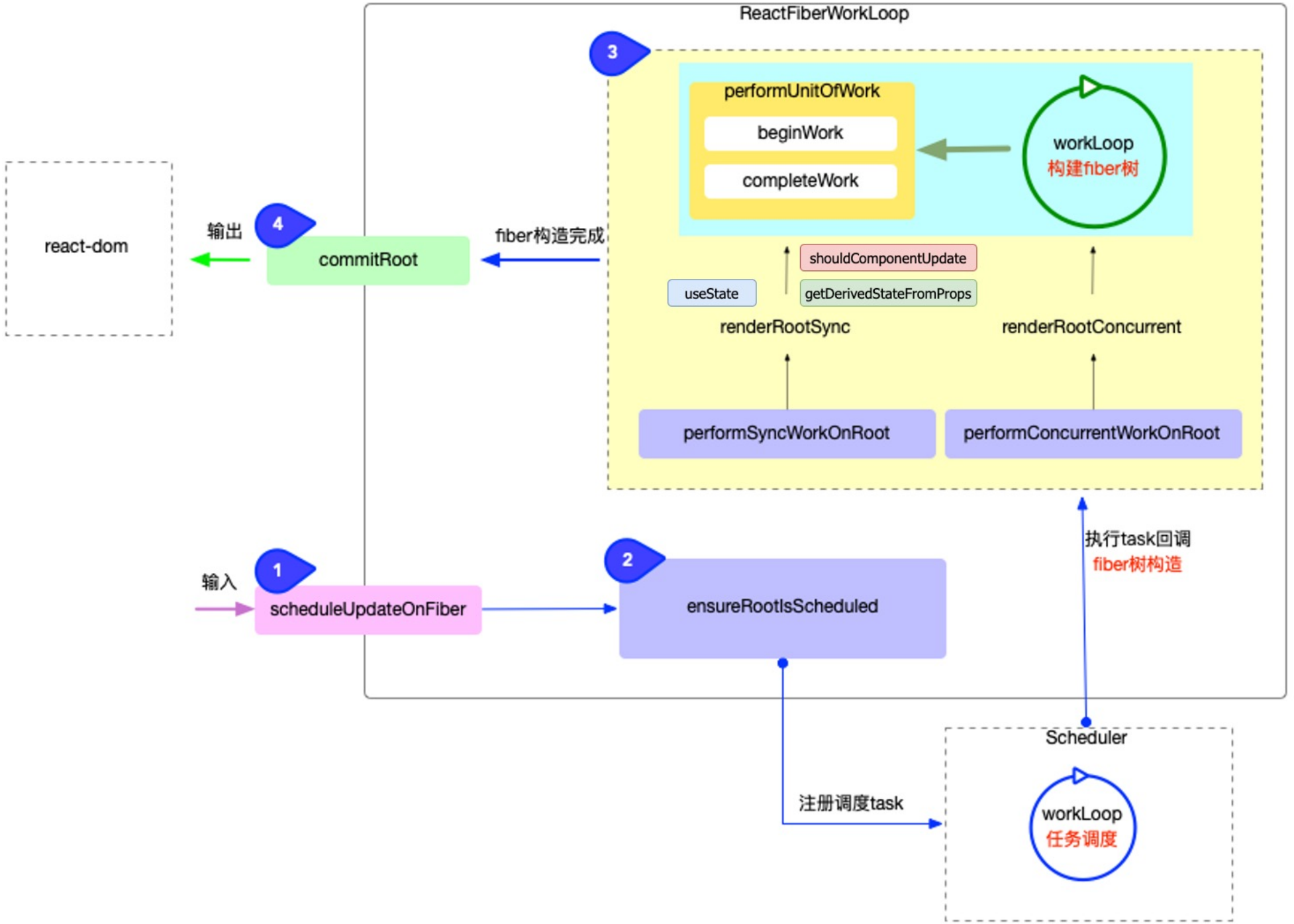
“Pre-commit 阶段”

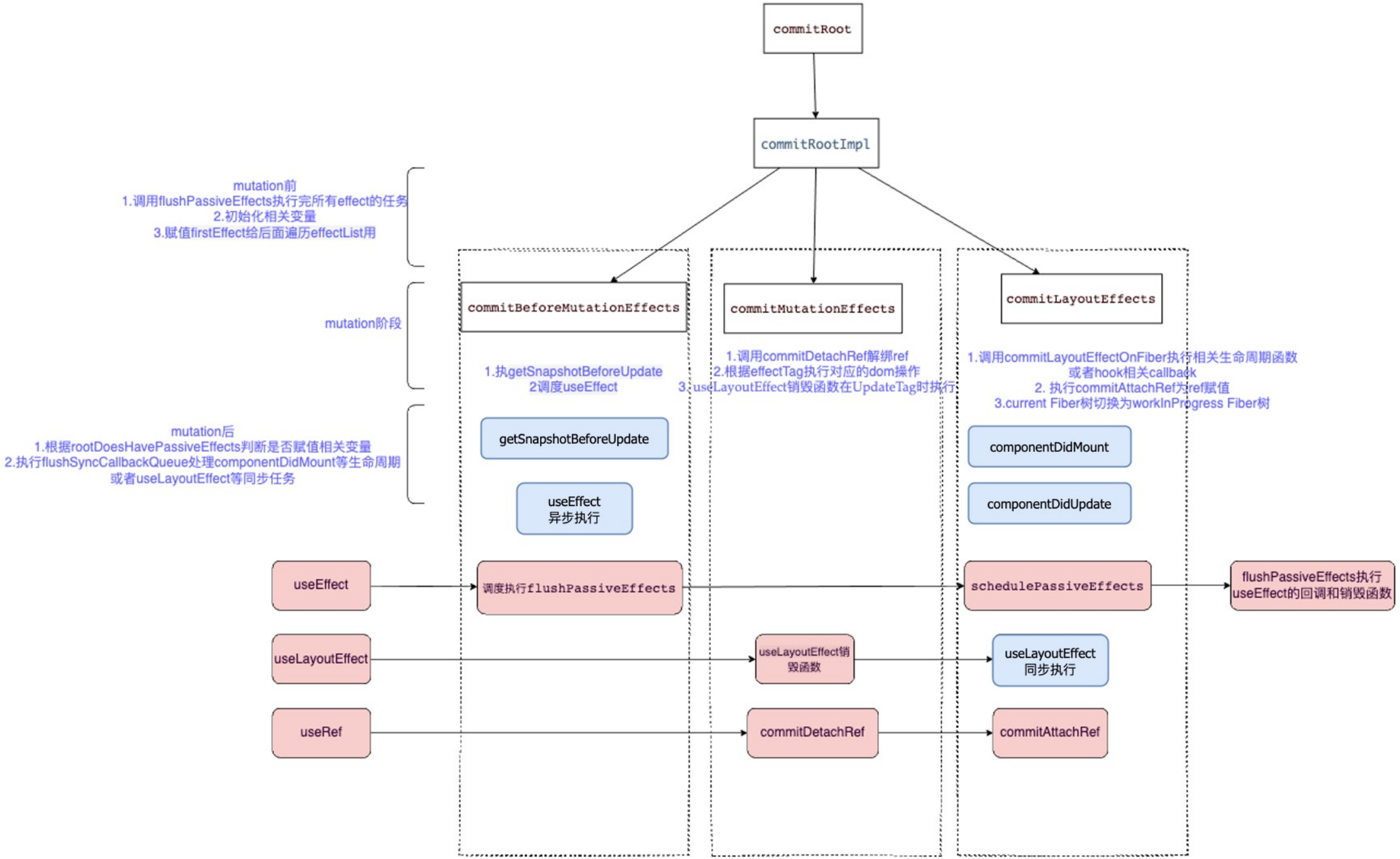
可以读取 DOM。

“Commit 阶段”

可以使用 DOM，运行副作用，安排更新。


mount
update
unMount
Error
render阶段
commit阶段
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
componentWillUnMount
componentDidCatch
componentDidMount
componentDidUpdate


ReactFiberWorkLoop
3
performUnitOfWork
beginWork
completeWork
workLoop
构建fiber树
shouldComponentUpdate
useState
getDerivedStateFromProps
renderRootSync
renderRootConcurrent
performSyncWorkOnRoot
performConcurrentWorkOnRoot
react-dom
输出
4
commitRoot
fiber构造完成
输入
1
scheduleUpdateOnFiber
2
ensureRootIsScheduled
执行task回调
fiber树构造
注册调度task
Scheduler
workLoop
任务调度


commitRoot
commitRootImpl
mutation前
1.调用flushPassiveEffects执行完所有effect的任务
2.初始化相关变量
3.赋值firstEffect给后面遍历effectList用
mutation阶段
commitBeforeMutationEffects
commitMutationEffects
commitLayoutEffects
1.执getSnapshotBeforeUpdate
2调度useEffect
1.调用commitDetachRef解绑ref
2.根据effectTag执行对应的dom操作
3.useLayoutEffect销毁函数在UpdateTag时执行
1.调用commitLayoutEffectOnFiber执行相关生命周期函数
或者hook相关callback
2. 执行commitAttachRef为ref赋值
3.current Fiber树切换为workInProgress Fiber树
mutation后
1.根据rootDoesHavePassiveEffects判断是否赋值相关变量
2.执行flushSyncCallbackQueue处理componentDidMount等生命周期
或者useLayoutEffect等同步任务
getSnapshotBeforeUpdate
useEffect
异步执行
componentDidMount
componentDidUpdate
useEffect
调度执行flushPassiveEffects
schedulePassiveEffects
flushPassiveEffects执行
useEffect的回调和销毁函数
useLayoutEffect
useLayoutEffect销
毁函数
useLayoutEffect
同步执行
useRef
commitDetachRef
commitAttachRef

总结

- 基础概念
- 基础模块
- 哪些数据结构
- 挂载流程
- 更新流程
- 优先级
- 调度器
- 生命周期

爪哇教育

Q&A

Thanks