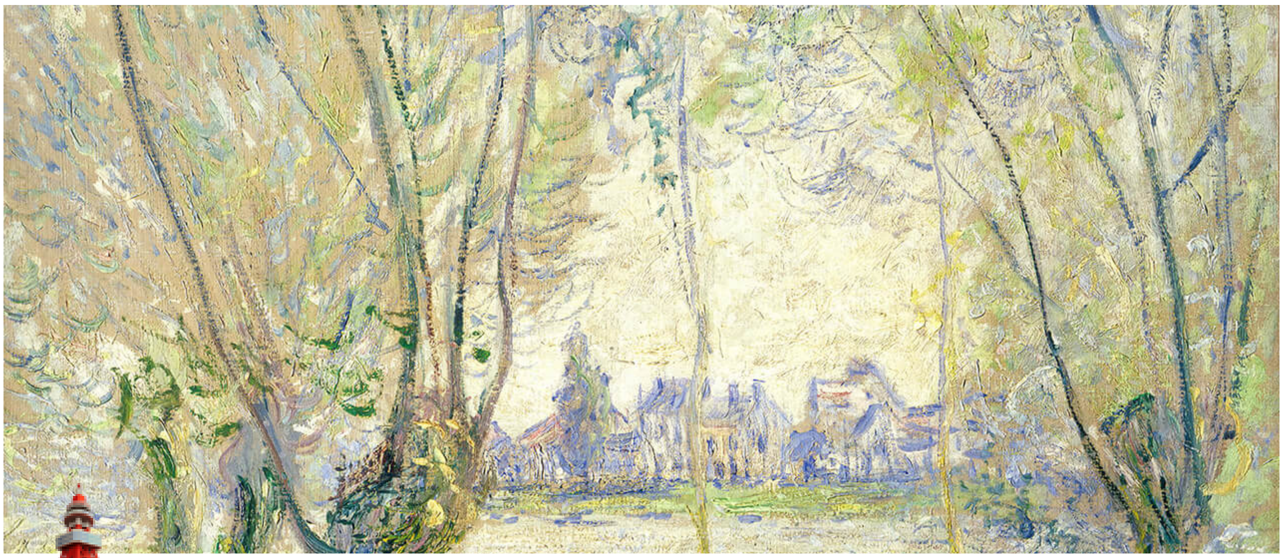


🗼

8-1 在浏览器 DOM 节点中创建根节点：createRoot

资源

1. createRoot

`createRoot` API

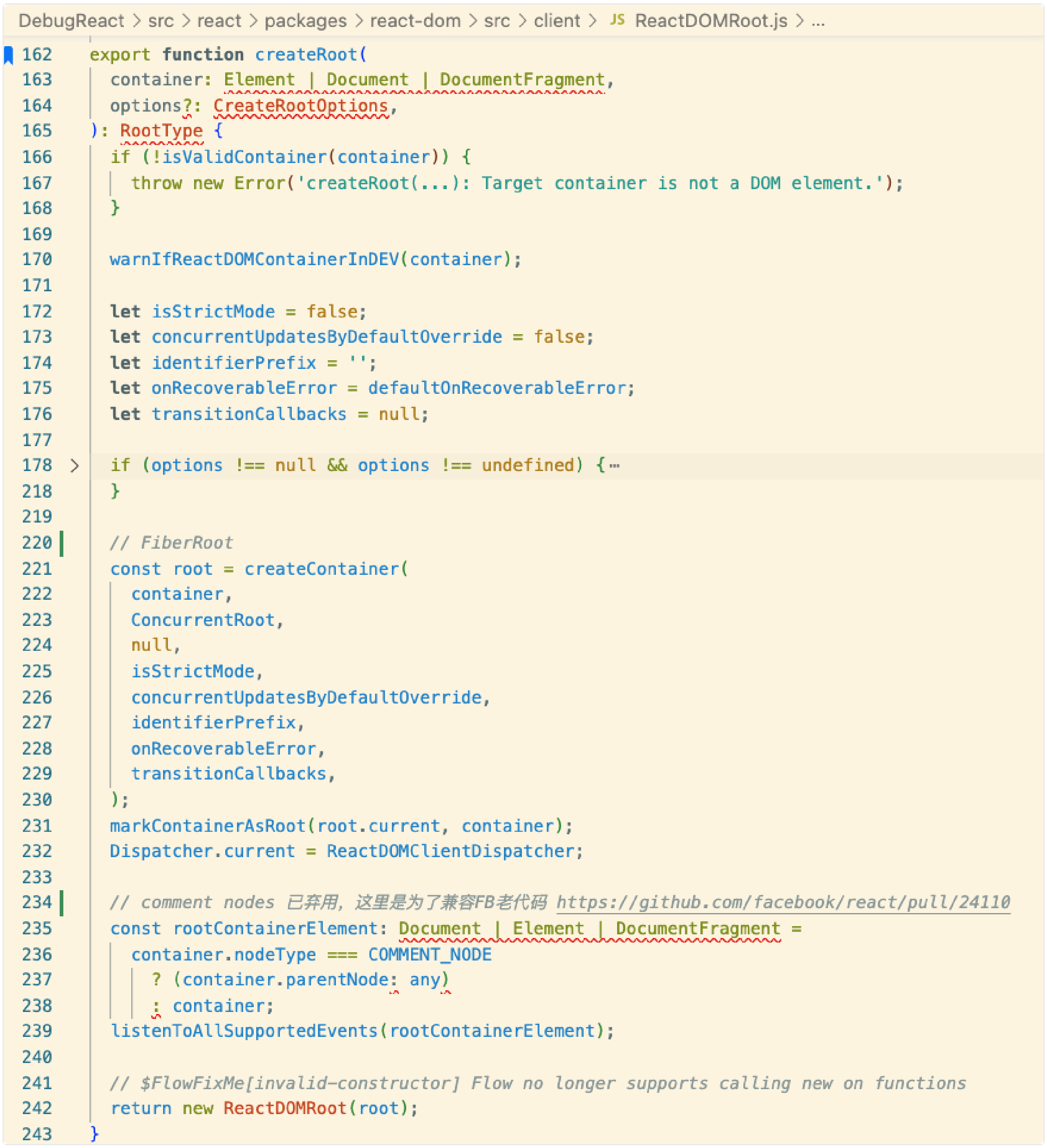
`createRoot` 允许在浏览器的 DOM 节点中创建根节点以显示 React 组件。

```javascript
import { createRoot } from "react-dom/client";
import jsx from "./pages/ExamplePage";

const root = createRoot(document.getElementById("root"));
root.render(jsx);
```

`createRoot` 源码

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,
};
```



DebugReact > src > react > packages > react-dom > src > client > JS ReactDOMRoot.js > ...

```
export function createRoot(
  container: Element | Document | DocumentFragment,
  options?: CreateRootOptions,
): RootType {
  if (!isValidContainer(container)) {
    throw new Error('createRoot(...): Target container is not a DOM element.');
  }

  warnIfReactDOMContainerInDEV(container);

  let isStrictMode = false;
  let concurrentUpdatesByDefaultOverride = false;
  let identifierPrefix = '';
  let onRecoverableError = defaultOnRecoverableError;
  let transitionCallbacks = null;

  if (options !== null && options !== undefined) {…
  }

  // FiberRoot
  const root = createContainer(
    container,
    ConcurrentRoot,
    null,
    isStrictMode,
    concurrentUpdatesByDefaultOverride,
    identifierPrefix,
    onRecoverableError,
    transitionCallbacks,
  );
  markContainerAsRoot(root.current, container);
  Dispatcher.current = ReactDOMClientDispatcher;

  // comment nodes 已弃用，这里是为了兼容FB老代码 https://github.com/facebook/react/pull/24110
  const rootContainerElement: Document | Element | DocumentFragment =
    container.nodeType === COMMENT_NODE
      ? (container.parentNode: any)
      : container;
  listenToAllSupportedEvents(rootContainerElement);

  // $FlowFixMe[invalid-constructor] Flow no longer supports calling new on functions
  return new ReactDOMRoot(root);
}
```

1. 检查 container 是否是 DOM

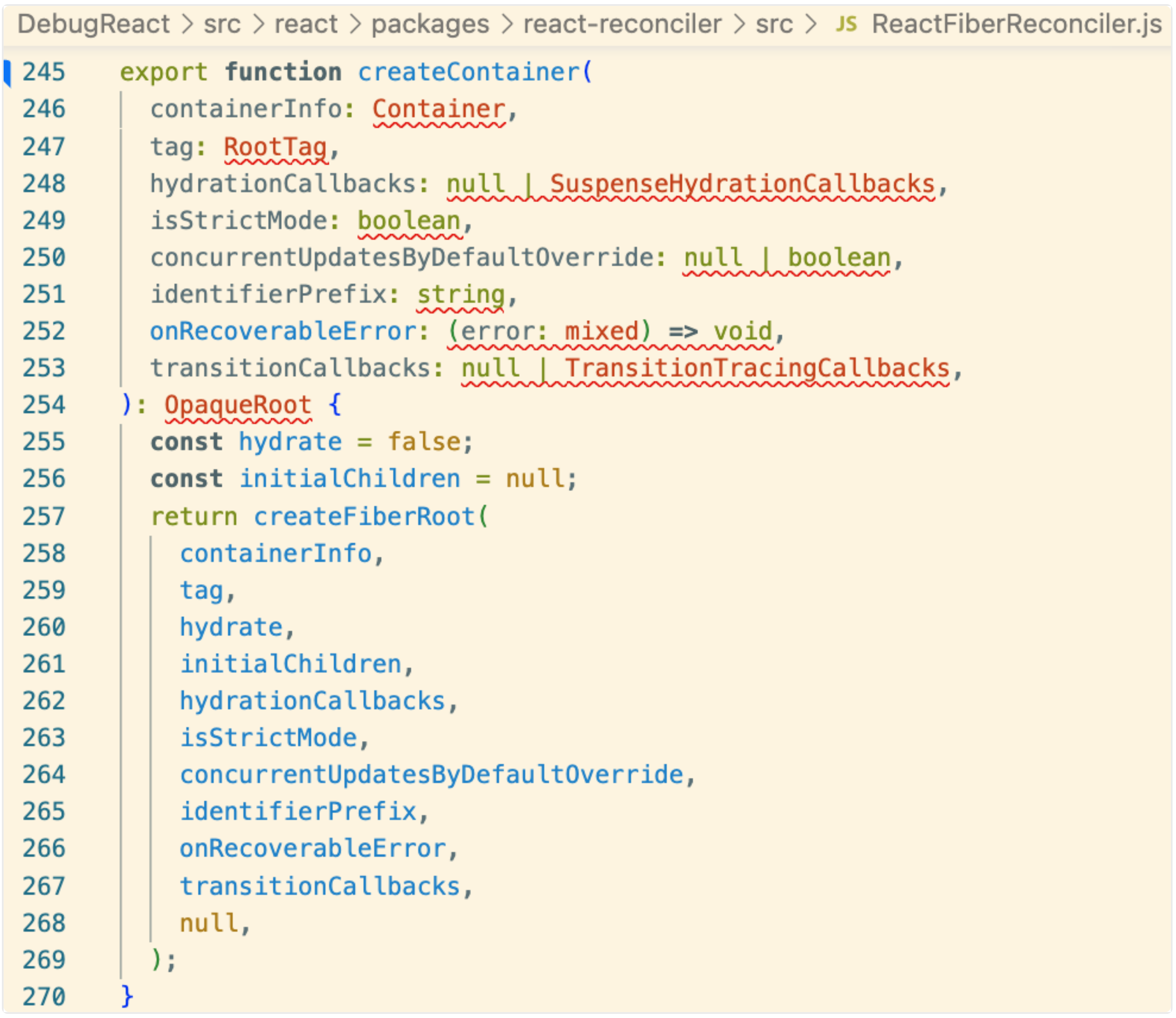
如果不是，throw new Error('createRoot(...): Target container is not a DOM element.');

2. 检查 options

目前文档中有两个参数可用：`onRecoverableError` 与 `identifierPrefix`。但是源码中实际上还有一些 unstable 值，属于非稳定值，不要使用～

3. `createContainer` 创建 `FiberRoot`，即源码里的 root。

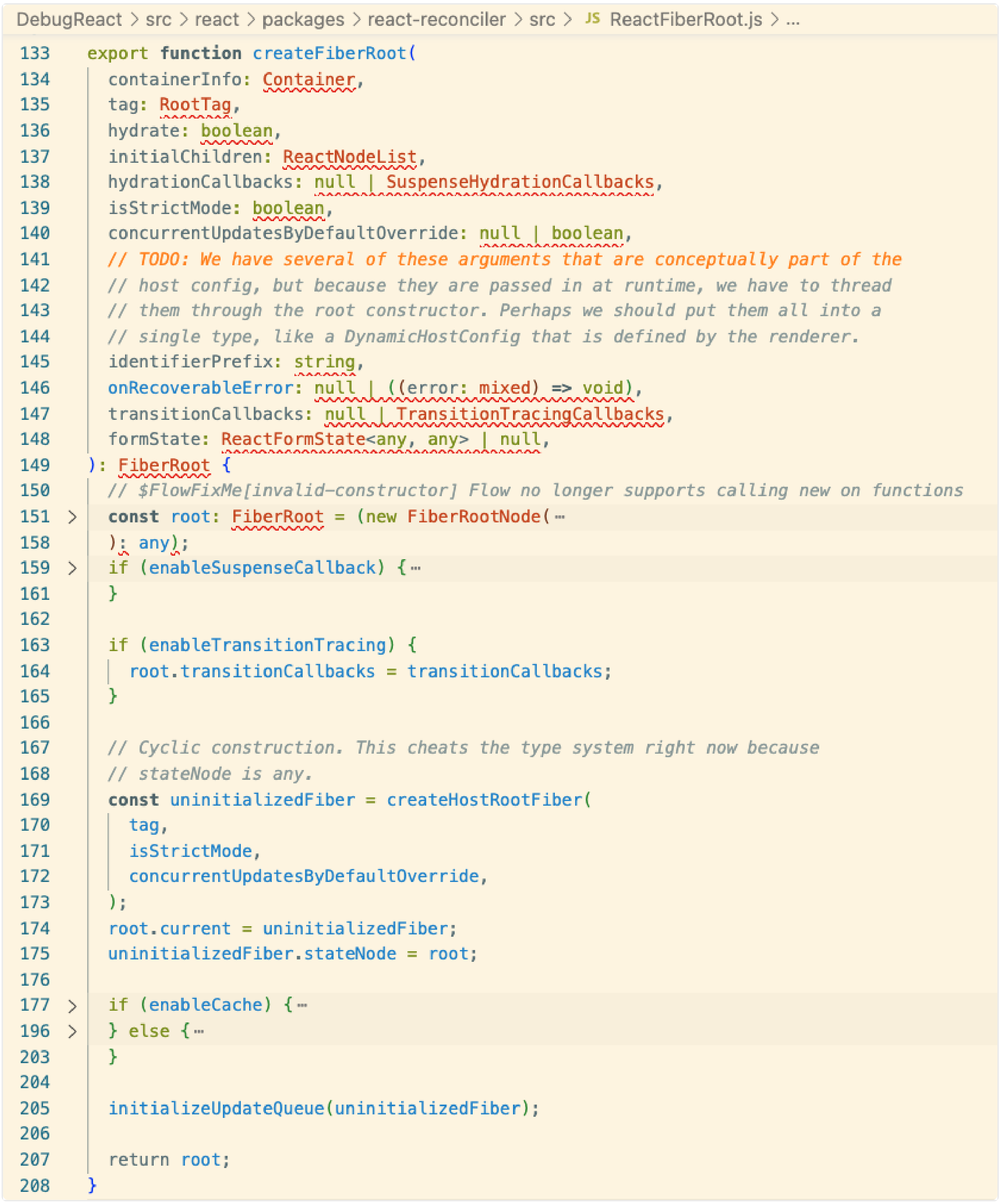
这里的 `containerInfo` 就是根 dom 节点。(就是我代码例子里那个 id 为 root 的 div)。这个变量在 `createRoot` 里叫 `container`，到这里换名成了 `containerInfo`。



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

```
export function createContainer(
  containerInfo: Container,
  tag: RootTag,
  hydrationCallbacks: null | SuspenseHydrationCallbacks,
  isStrictMode: boolean,
  concurrentUpdatesByDefaultOverride: null | boolean,
  identifierPrefix: string,
  onRecoverableError: (error: mixed) => void,
  transitionCallbacks: null | TransitionTracingCallbacks,
): OpaqueRoot {
  const hydrate = false;
  const initialChildren = null;
  return createFiberRoot(
    containerInfo,
    tag,
    hydrate,
    initialChildren,
    hydrationCallbacks,
    isStrictMode,
    concurrentUpdatesByDefaultOverride,
    identifierPrefix,
    onRecoverableError,
    transitionCallbacks,
    null,
  );
}
```

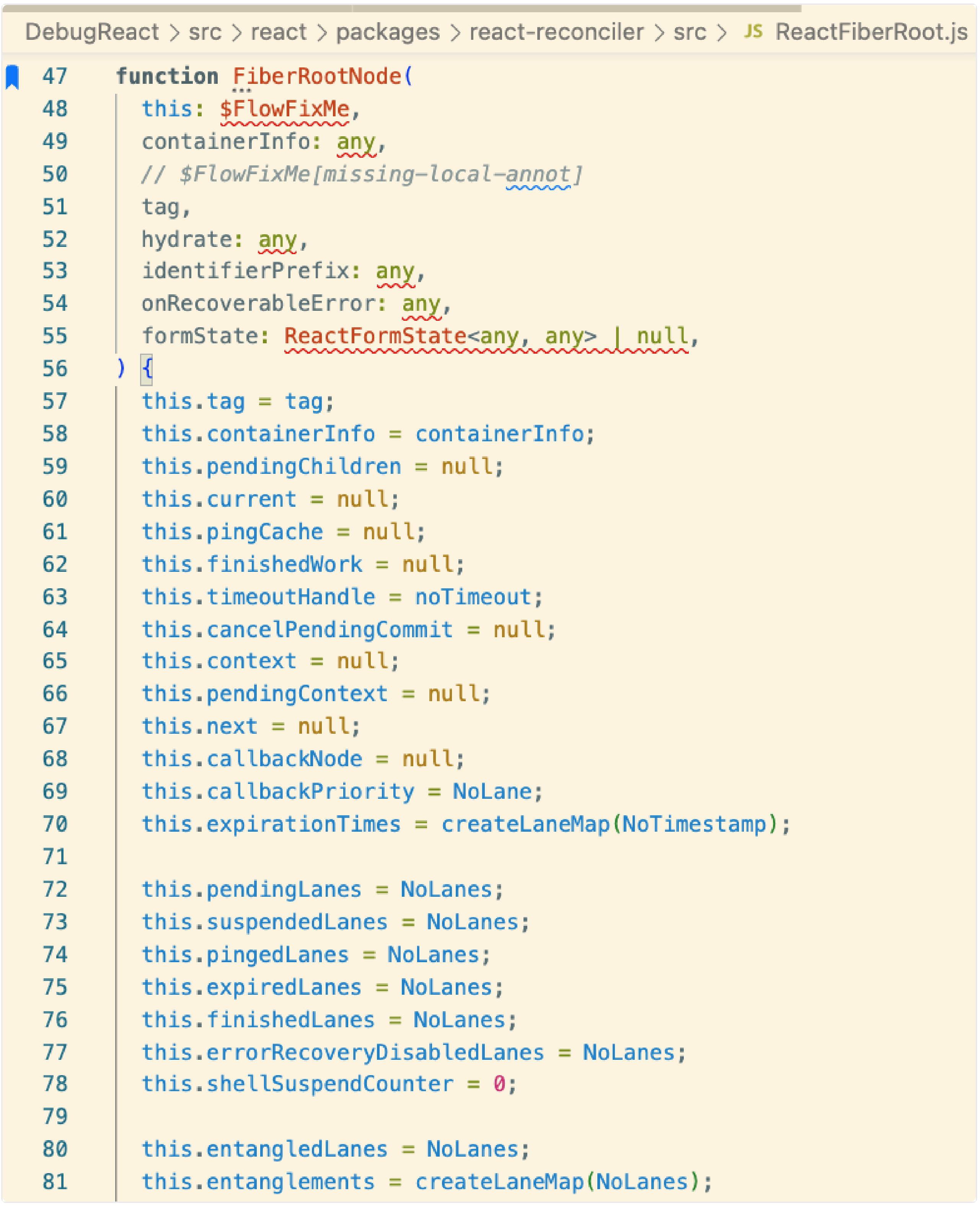
`createFiberRoot` 创建并返回 `FiberRoot`



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

```
export function createFiberRoot(
  containerInfo: Container,
  tag: RootTag,
  hydrate: boolean,
  initialChildren: ReactNodeList,
  hydrationCallbacks: null | SuspenseHydrationCallbacks,
  isStrictMode: boolean,
  concurrentUpdatesByDefaultOverride: null | boolean,
  // TODO: We have several of these arguments that are conceptually part of the
  // host config, but because they are passed in at runtime, we have to thread
  // them through the root constructor. Perhaps we should put them all into a
  // single type, like a DynamicHostConfig that is defined by the renderer.
  identifierPrefix: string,
  onRecoverableError: null | ((error: mixed) => void),
  transitionCallbacks: null | TransitionTracingCallbacks,
  formState: ReactFormState<any, any> | null,
): FiberRoot {
  // $FlowFixMe[invalid-constructor] Flow no longer supports calling new on functions
  const root: FiberRoot = (new FiberRootNode(…
  ): any);
  if (enableSuspenseCallback) {…
  }

  if (enableTransitionTracing) {
    root.transitionCallbacks = transitionCallbacks;
  }

  // Cyclic construction. This cheats the type system right now because
  // stateNode is any.
  const uninitializedFiber = createHostRootFiber(
    tag,
    isStrictMode,
    concurrentUpdatesByDefaultOverride,
  );
  root.current = uninitializedFiber;
  uninitializedFiber.stateNode = root;

  if (enableCache) {…
  } else {…
  }

  initializeUpdateQueue(uninitializedFiber);

  return root;
}
```

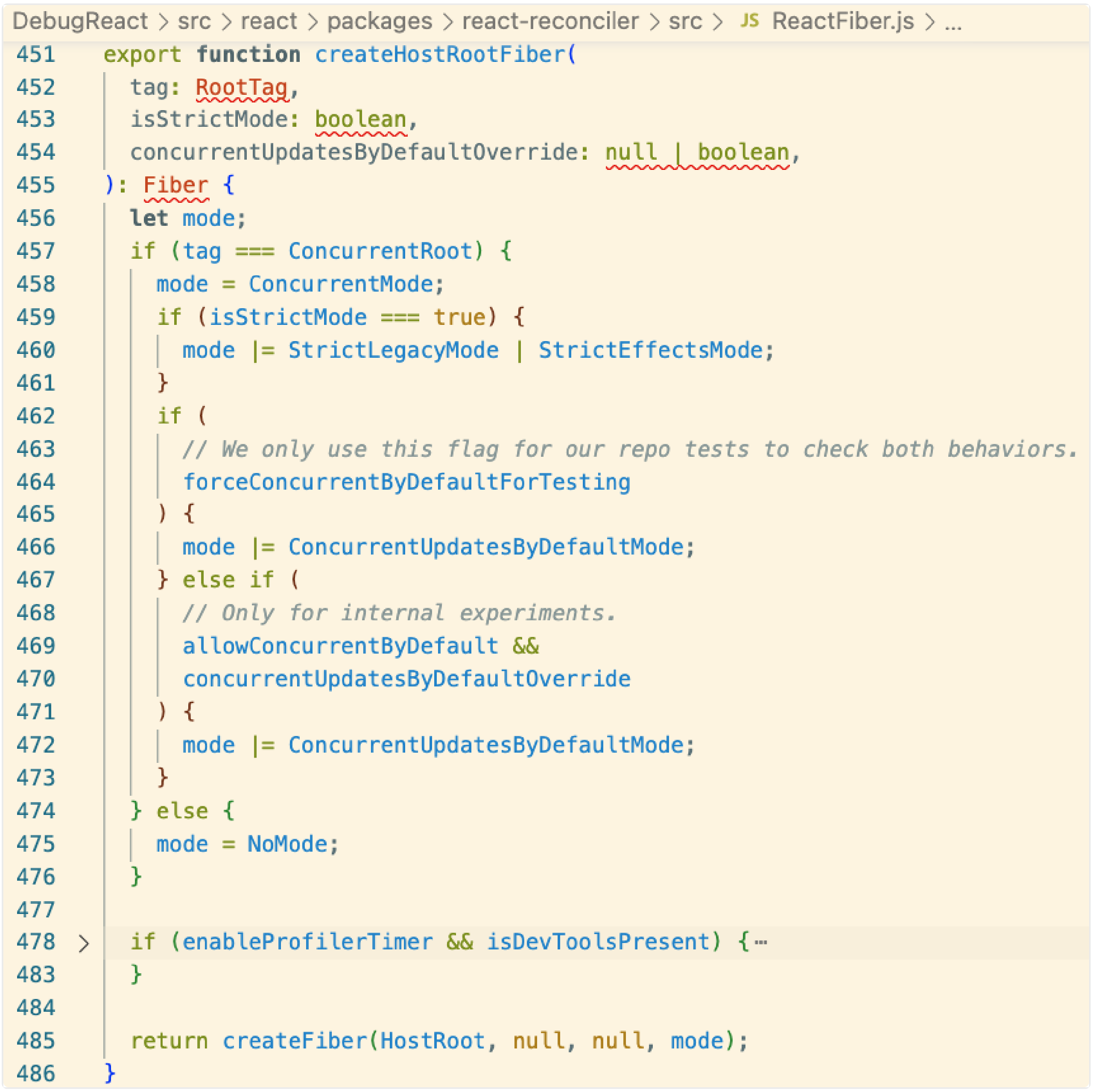
a. 实例化 `FiberRootNode`，创建 `FiberRoot`


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

```
function FiberRootNode(
  this: $FlowFixMe,
  containerInfo: any,
  // $FlowFixMe[missing-local-annot]
  tag,
  hydrate: any,
  identifierPrefix: any,
  onRecoverableError: any,
  formState: ReactFormState<any, any> | null,
) {
  this.tag = tag;
  this.containerInfo = containerInfo;
  this.pendingChildren = null;
  this.current = null;
  this.pingCache = null;
  this.finishedWork = null;
  this.timeoutHandle = noTimeout;
  this.cancelPendingCommit = null;
  this.context = null;
  this.pendingContext = null;
  this.next = null;
  this.callbackNode = null;
  this.callbackPriority = NoLane;
  this.expirationTimes = createLaneMap(NoTimestamp);

  this.pendingLanes = NoLanes;
  this.suspendedLanes = NoLanes;
  this.pingedLanes = NoLanes;
  this.expiredLanes = NoLanes;
  this.finishedLanes = NoLanes;
  this.errorRecoveryDisabledLanes = NoLanes;
  this.shellSuspendCounter = 0;

  this.entangledLanes = NoLanes;
  this.entanglements = createLaneMap(NoLanes);
```

b. `createHostRootFiber` 创建原生标签的根 `Fiber`

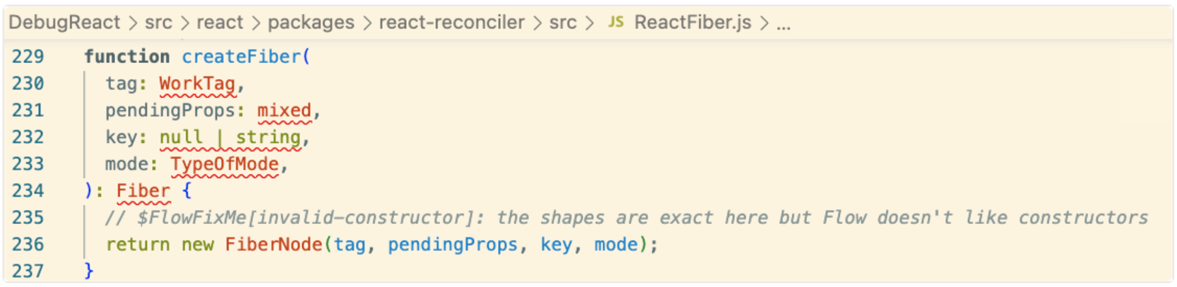
注意这里创建的是 `Fiber`，只是属于根部的 `Fiber`。和 a.的 `FiberRoot` 不同，`FiberRoot` 与 `Fiber` 是两个类型。



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

```
export function createHostRootFiber(
  tag: RootTag,
  isStrictMode: boolean,
  concurrentUpdatesByDefaultOverride: null | boolean,
): Fiber {
  let mode;
  if (tag === ConcurrentRoot) {
    mode = ConcurrentMode;
    if (isStrictMode === true) {
      mode |= StrictLegacyMode | StrictEffectsMode;
    }
    if (
      // We only use this flag for our repo tests to check both behaviors.
      forceConcurrentByDefaultForTesting
    ) {
      mode |= ConcurrentUpdatesByDefaultMode;
    } else if (
      // Only for internal experiments.
      allowConcurrentByDefault &&
      concurrentUpdatesByDefaultOverride
    ) {
      mode |= ConcurrentUpdatesByDefaultMode;
    }
  } else {
    mode = NoMode;
  }

  if (enableProfilerTimer && isDevToolsPresent) {…
  }

  return createFiber(HostRoot, null, null, mode);
}
```

`createFiber` 创建 `Fiber`


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

```
function createFiber(
  tag: WorkTag,
  pendingProps: mixed,
  key: null | string,
  mode: TypeOfMode,
): Fiber {
  // $FlowFixMe[invalid-constructor]: the shapes are exact here but Flow doesn't like constructors
  return new FiberNode(tag, pendingProps, key, mode);
}
```

FiberNode

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

Flow

```
function FiberNode(
  this: $FlowFixMe,
  tag: WorkTag,
  pendingProps: mixed,
  key: null | string,
  mode: TypeOfMode,
) {
  // Instance
  this.tag = tag;
  this.key = key;
  this.elementType = null;
  this.type = null;
  this.stateNode = null;

  // Fiber
  this.return = null;
  this.child = null;
  this.sibling = null;
  this.index = 0;

  this.ref = null;
  this.refCleanup = null;

  this.pendingProps = pendingProps;
  this.memoizedProps = null;
  this.updateQueue = null;
  this.memoizedState = null;
  this.dependencies = null;
```

```
    this.mode = mode;

    // Effects
    this.flags = NoFlags;
    this.subtreeFlags = NoFlags;
    this.deletions = null;

    this.lanes = NoLanes;
    this.childLanes = NoLanes;

    this.alternate = null;
  }
```

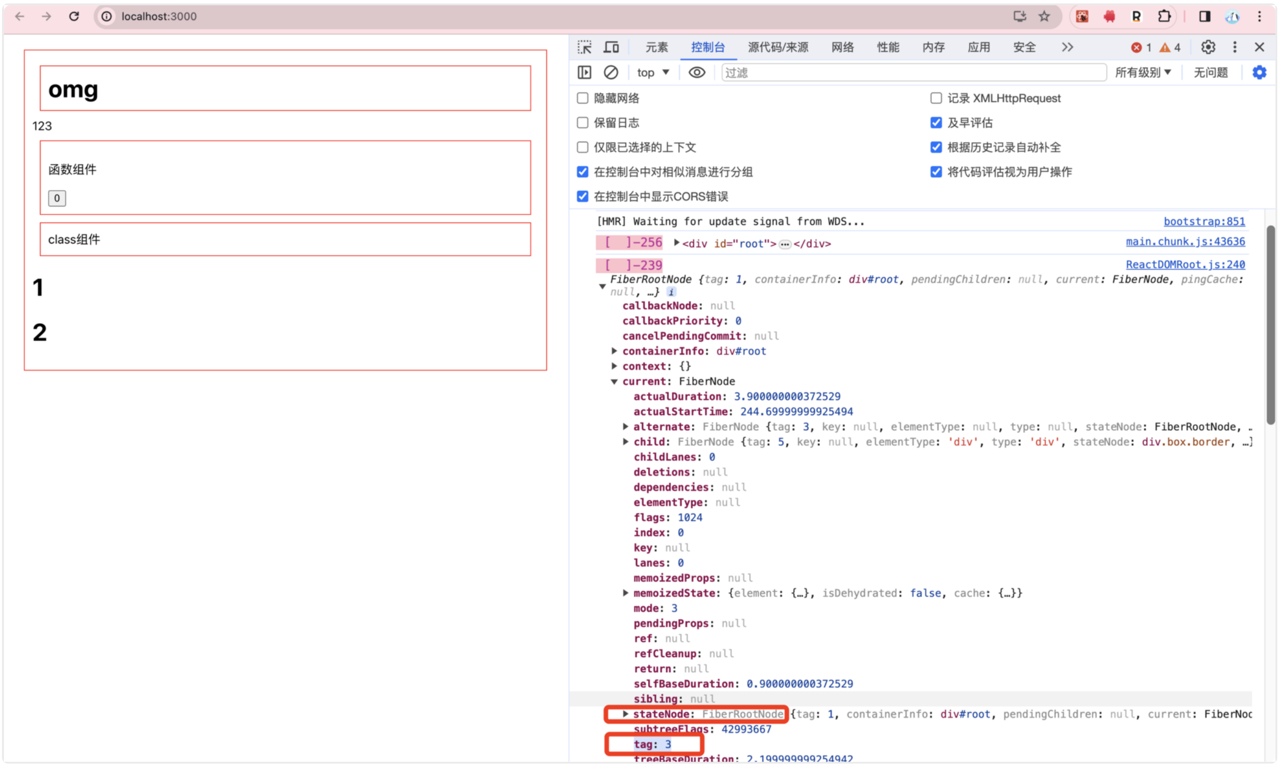
c.循环构造 `root` 与 `uninitializedFiber`

root.current 是 `Fiber`

uninitializedFiber.stateNode 是根 `FiberRoot`

```Flow
root.current = uninitializedFiber; // Fiber
uninitializedFiber.stateNode = root; // FiberRoot
```

参考 DebugReact 中的 console.log 截图如下：



☐ 隐藏网络　☐ 记录 XMLHttpRequest
☐ 保留日志　☑ 及早评估
☐ 仅限已选择的上下文　☑ 根据历史记录自动补全
☑ 在控制台中对相似消息进行分组　☑ 将代码评估视为用户操作
☑ 在控制台中显示CORS错误

```
[HMR] Waiting for update signal from WDS...                          bootstrap:851
[  ]-256 ▶<div id="root">…</div>                                main.chunk.js:43636
[  ]-239                                                        ReactDOMRoot.js:240
▼FiberRootNode {tag: 1, containerInfo: div#root, pendingChildren: null, current: FiberNode, pingCache:
   null, …} i
    callbackNode: null
    callbackPriority: 0
    cancelPendingCommit: null
  ▶ containerInfo: div#root
  ▶ context: {}
  ▼ current: FiberNode
      actualDuration: 3.900000000372529
      actualStartTime: 244.69999999925494
    ▶ alternate: FiberNode {tag: 3, key: null, elementType: null, type: null, stateNode: FiberRootNode, …
    ▶ child: FiberNode {tag: 5, key: null, elementType: 'div', type: 'div', stateNode: div.box.border, …}
      childLanes: 0
      deletions: null
      dependencies: null
      elementType: null
      flags: 1024
      index: 0
      key: null
      lanes: 0
      memoizedProps: null
    ▶ memoizedState: {element: {…}, isDehydrated: false, cache: {…}}
      mode: 3
      pendingProps: null
      ref: null
      refCleanup: null
      return: null
      selfBaseDuration: 0.900000000372529
      sibling: null
    ▶ stateNode: FiberRootNode {tag: 1, containerInfo: div#root, pendingChildren: null, current: FiberNoc
      subtreeFlags: 42993667
      tag: 3
      treeBaseDuration: 2.199999999254942
```

d. 初始化 `initializeUpdateQueue`

类似 fiber，update queues 也是成对出现的，一个已经完成的即对应目前页面，一个正在工作中的。

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

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

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

  next: Update<State> | null,
};

export type SharedQueue<State> = {
  pending: Update<State> | null, // 单向循环链表
  lanes: Lanes,
  // 如果类组件是Activity(以前叫OffScreen)的后代组件，需要延迟执行的其setState
  // Activity目前还是unstable，了解即可~
  hiddenCallbacks: Array<() => mixed> | null,
};

export type UpdateQueue<State> = {
  baseState: State,
  // 单链表 firstBaseUpdate->...->lastBaseUpdate
  firstBaseUpdate: Update<State> | null,
  // 一般情况下，单链表是不用记录尾节点，这里记录尾节点是为了快速比较两个单链表，用
  lastBaseUpdate: Update<State> | null,
  shared: SharedQueue<State>,
  callbacks: Array<() => mixed> | null,
};

// 这里初始化fiber.updateQueue。在beginWork阶段，updateHostRoot中使用proces
export function initializeUpdateQueue<State>(fiber: Fiber): void {
  const queue: UpdateQueue<State> = {
    baseState: fiber.memoizedState,
    firstBaseUpdate: null,
    lastBaseUpdate: null,
    shared: {
      pending: null,
      lanes: NoLanes,
      hiddenCallbacks: null,
    },
    callbacks: null,
  };
  fiber.updateQueue = queue;
}
```

4. `markContainerAsRoot` 标记 Container 是根 Fiber

这个函数给 container 根 DOM 节点赋值根 Fiber。

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

Flow

```
const randomKey = Math.random().toString(36).slice(2);

const internalContainerInstanceKey = '__reactContainer$' + randomKey;
// 标记根节点
export function markContainerAsRoot(hostRoot: Fiber, node: Container):
  node[internalContainerInstanceKey] = hostRoot;
}
```

这个属性值在函数中用于 `getClosestInstanceFromNode` 和 `getInstanceFromNode` 中会用于根据根 DOM 取 Fiber 值。

对应的还有两个函数：

Flow

```
// 取消标记，在ReactDOMRoot.prototype.unmount函数里调用
export function unmarkContainerAsRoot(node: Container): void {
  node[internalContainerInstanceKey] = null;
}
// 检查是否被标记为根节点
export function isContainerMarkedAsRoot(node: Container): boolean {
  return !!node[internalContainerInstanceKey];
}
```

5. 从 container 层监听 listenToAllSupportedEvents

React 事件比较复杂，这里暂时不展开，具体查看后面的事件章节。

6. 最后返回一个 `ReactDOMRoot` 实例

Flow

```
return new ReactDOMRoot(root);
```

`ReactDOMRoot` 函数：

Flow

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

ExamplePage.jsx

JavaScript

```
class ClassComponent extends Component {
  render() {
    return <div className="class border">{this.props.name}</div>;
  }
}

function FunctionComponent(props) {
  const [count1, setCount1] = useReducer((x) => x + 1, 0);
  useEffect(() => {
    return () => {
      console.log("销毁");
    };
  }, []);

  return (
    <div className="border">
      <p>{props.name}</p>
      <button
        onClick={() => {
          setCount1();
        }}
      >
        {count1}
      </button>
    </div>
  );
}

const jsx = (
  <div className="box border">
    <h1 className="border">omg</h1>
    123
    <FunctionComponent name="函数组件" />
    <ClassComponent name="class组件" />
    <>
      <h1>1</h1>
      <h1>2</h1>
    </>
  </div>
);

export default jsx;
```