10-3 如何实现 Fragment 渲染的源码

资源

1. Fragment

基本使用

`<Fragment>` 通常使用 `<>...</>` 代替，它们都允许你在不添加额外节点的情况下将子元素组合。

第一种 Fragment

React TSX

```tsx
let fragment1: any = (
  <>
    <h3>1</h3>
    <h4>2</h4>
  </>
);
```

```
const jsx = (
  <div className="box border">
    <h1 className="border">omg</h1>
    123
    <h2>react</h2>
    omg2
    {fragment1}
  </div>
);

ReactDOM.createRoot(document.getElementById("root") as HTMLElement).re
  jsx
);
```

render 阶段

创建 Fragment 的 fiber

创建 Fragment Fiber

ReactFiber.ts

TypeScript

```
// 根据 TypeAndProps 创建fiber
export function createFiberFromTypeAndProps(
  type: any,
  key: null | string,
  pendingProps: any
) {
  let fiberTag: WorkTag = IndeterminateComponent;
  if (isStr(type)) {
    // 原生标签
    fiberTag = HostComponent;
  } else {
    switch (type) {
      case REACT_FRAGMENT_TYPE:
        fiberTag = Fragment;
        break;
```

```
    }
  }

  const fiber = createFiber(fiberTag, pendingProps, key);
  fiber.elementType = type;
  fiber.type = type;
  return fiber;
}

export function createFiberFromFragment(
  elements: ReactFragment,
  key: null | string
): Fiber {
  const fiber = createFiber(Fragment, elements, key);
  return fiber;
}
```

beginWork

TypeScript

```
function updateFragment(current: Fiber | null, workInProgress: Fiber)
  const nextChildren = workInProgress.pendingProps.children;
  reconcileChildren(current, workInProgress, nextChildren);
  return workInProgress.child;
}
```

协调

TypeScript

```
function updateFragment(current: Fiber | null, workInProgress: Fiber)
  const nextChildren = workInProgress.pendingProps.children;
  reconcileChildren(current, workInProgress, nextChildren);
  return workInProgress.child;
}
```

completeWork

ReactFiberCompleteWork.ts

TypeScript

```
import { isNum, isStr } from "shared/utils";
import { Fiber } from "./ReactInternalTypes";
import { Fragment, HostComponent, HostRoot, HostText } from "./ReactWo

export function completeWork(
  current: Fiber | null,
  workInProgress: Fiber
): Fiber | null {
  const newProps = workInProgress.pendingProps;
  switch (workInProgress.tag) {
    case Fragment:
    case HostRoot: {
      return null;
    }
    case HostComponent: {
      // 原生标签,type是标签名
      const { type } = workInProgress;
      // 1. 创建真实DOM
      const instance = document.createElement(type);
      // 2. 初始化DOM属性
      finalizeInitialChildren(instance, newProps);
      // 3. 把子dom挂载到父dom上
      appendAllChildren(instance, workInProgress);
      workInProgress.stateNode = instance;
      return null;
    }

    case HostText: {
      workInProgress.stateNode = document.createTextNode(newProps);
      return null;
    }
    // todo
  }

  throw new Error(
    `Unknown unit of work tag (${workInProgress.tag}). This error is l
      "React. Please file an issue."
  );
```

```
}

// 初始化属性
function finalizeInitialChildren(domElement: Element, props: any) {
  for (const propKey in props) {
    const nextProp = props[propKey];
    if (propKey === "children") {
      if (isStr(nextProp) || isNum(nextProp)) {
        // 属性
        domElement.textContent = nextProp;
      }
    } else {
      // 3. 设置属性
      (domElement as any)[propKey] = nextProp;
    }
  }
}

function appendAllChildren(parent: Element, workInProgress: Fiber) {
  let nodeFiber = workInProgress.child; // 单链表结构
  while (nodeFiber !== null) {
    if (nodeFiber.tag === HostComponent || nodeFiber.tag === HostText)
      parent.appendChild(nodeFiber.stateNode);
    } else if (nodeFiber.child !== null) {
      // 如果node这个fiber本身不直接对应DOM节点，那么就往下找它的子节点，直到找到
      nodeFiber = nodeFiber.child;
      continue;
    }

    if (nodeFiber === workInProgress) {
      return;
    }
    // 如果nodeFiber没有兄弟节点了，那么就往上找它的父节点
    while (nodeFiber.sibling === null) {
      if (nodeFiber.return === null || nodeFiber.return === workInProg
        return;
      }
      nodeFiber = nodeFiber.return;
    }
    nodeFiber = nodeFiber.sibling;
```

```
  }
}
```

第二种

TypeScript

```
let fragment1: any = (
  <>
    <h3>1</h3>
    <h4>2</h4>
  </>
);

ReactDOM.createRoot(document.getElementById("root") as HTMLElement).re
  fragment1
);
```

commit

ReactFiberCommitWork.ts

TypeScript

```
function commitPlacement(finishedWork: Fiber) {
  if (
    finishedWork.stateNode &&
    (finishedWork.tag === HostComponent || finishedWork.tag === HostTe
  ) {
    // finishedWork是有dom节点
    const domNode = finishedWork.stateNode;
    // 找domNode的父DOM节点对应的fiber
    const parentFiber = getHostParentFiber(finishedWork);

    let parentDom = parentFiber.stateNode;

    if (parentDom.containerInfo) {
      // HostRoot
```

```
      parentDom = parentDom.containerInfo;
    }

    parentDom.appendChild(domNode);
  } else {
    let kid = finishedWork.child;
    while (kid !== null) {
      commitPlacement(kid);
      kid = kid.sibling;
    }
  }
}
```

第三种

React TSX

```
const fragment1: any = (
  <Fragment key='sy'>
    <h3>1</h3>
    <h4>2</h4>
  </Fragment>
);
```

packages/react/src/index.ts

TypeScript

```
export { REACT_FRAGMENT_TYPE as Fragment } from "shared/ReactSymbols";
```